Basic Computer Coding:

SQL

2nd Edition

BIBLIOTEX
Digital Library

BASIC COMPUTER CODING: SQL

2nd Edition

BIBLIOTEX
Digital Library

www.bibliotex.com

BASIC COMPUTER CODING: SQL

2ND EDITION

www.bibliotex.com

email: info@bibliotex.com

e-book Edition 2022

ISBN: 978-1-98467-606-1 (e-book)

This book contains information obtained from highly regarded resources. Reprinted material sources are indicated. Copyright for individual articles remains with the authors as indicated and published under Creative Commons License. A Wide variety of references are listed. Reasonable efforts have been made to publish reliable data and views articulated in the chapters are those of the individual contributors, and not necessarily those of the editors or publishers. Editors or publishers are not responsible for the accuracy of the information in the published chapters or consequences of their use. The publisher assumes no responsibility for any damage or grievance to the persons or property arising out of the use of any materials, instructions, methods or thoughts in the book. The editors and the publisher have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission has not been obtained. If any copyright holder has not been acknowledged, please write to us so we may rectify.

Notice: Registered trademark of products or corporate names are used only for explanation and identification without intent of infringement.

© 2022 3G E-learning LLC

In Collaboration with 3G E-Learning LLC. Originally Published in printed book format by 3G E-Learning LLC with ISBN 978-1-98465-899-9

EDITORIAL BOARD

Aleksandar Mratinković was born on May 5, 1988 in Arandjelovac, Serbia. He has graduated on Economic high school (2007), The College of Tourism in Belgrade (2013), and also has a master degree of Psychology (Faculty of Philosophy, University of Novi Sad). He has been engaged in different fields of psychology (Developmental Psychology, Clinical Psychology, Educational Psychology and Industrial Psychology) and has published several scientific works.

Dan Piestun (PhD) is currently a startup entrepreneur in Israel working on the interface of Agriculture and Biomedical Sciences and was formerly president-CEO of the National Institute of Agricultural Research (INIA) in Uruguay. Dan is a widely published scientist who has received many honours during his career including being a two-time recipient of the Amit Golda Meir Prize from the Hebrew University of Jerusalem, his areas of expertise includes stem cell molecular biology, plant and animal genetics and bioinformatics. Dan's passion for applied science and technological solutions did not stop him from pursuing a deep connection to the farmer, his family and nature. Among some of his interest and practices counts enjoying working as a beekeeper and onboard fishing.

Hazem Shawky Fouda has a PhD. in Agriculture Sciences, obtained his PhD. From the Faculty of Agriculture, Alexandria University in 2008, He is working in Cotton Arbitration & Testing General Organization (CATGO).

Felecia Killings is the Founder and CEO of LiyahAmore Publishing, a publishing company committed to providing technical and educational services and products to Christian Authors. She operates as the Senior Editor and Writer, the Senior Writing Coach, the Content Marketing Specialist, Editor-in-Chief to the company's quarterly magazine, the Executive and Host of an international virtual network, and the Executive Director of the company's online school for Authors. She is a former high-school English instructor and professional development professor. She possesses a Master of Arts degree in Education and a Bachelor's degree in English and African American studies.

Dr. Sandra El Hajj, Ph.D. in Health Sciences from Nova Southeastern University, Florida, USA is a health professional specialized in Preventive and Global Health. With her 12 years of education obtained from one of the most prominent universities in Beirut, in addition to two leading universities in the State of Florida (USA), Dr. Sandra made sure to incorporate interdisciplinary and multicultural approaches in her work. Her long years of studies helped her create her own miniature world of knowledge linking together the healthcare field with Medical Research, Statistics, Food Technology, Environmental & Occupational Health, Preventive Health and most noteworthy her precious last degree of Global Health. Till today, she is the first and only doctor specialized in Global Health in the Middle East area.

Fozia Parveen has a Dphil in Sustainable Water Engineering from the University of Oxford. Prior to this she has received MS in Environmental Sciences from National University of Science and Technology (NUST), Islamabad Pakistan and BS in Environmental Sciences from Fatima Jinnah Women University (FJWU), Rawalpindi.

Igor Krunic 2003-2007 in the School of Economics. After graduating in 2007, he went on to study at The College of Tourism, at the University of Belgrade where he got his bachelor degree in 2010. He was active as a third-year student representative in the student parliament.Then he went on the Faculty of science, at the University of Novi Sad where he successfully defended his master's thesis in 2013. The crown of his study was the work titled Opportunities for development of cultural tourism in Cacak". Later on, he became part of a multinational company where he got promoted to a deputy director of logistic. Nowadays he is a consultant and writer of academic subjects in the field of tourism.

Dr. Jovan Pehcevski obtained his PhD in Computer Science from RMIT University in Melbourne, Australia in 2007. His research interests include big data, business intelligence and predictive analytics, data and information science, information retrieval, XML, web services and service-oriented architectures, and relational and NoSQL database systems. He has published over 30 journal and conference papers and he also serves as a journal and conference reviewer. He is currently working as a Dean and Associate Professor at European University in Skopje, Macedonia.

Dr. Tanjina Nur finished her PhD in Civil and Environmental Engineering in 2014 from University of Technology Sydney (UTS). Now she is working as Post-Doctoral Researcher in the Centre for Technology in Water and Wastewater (CTWW) and published about eight International journal papers with 80 citations. Her research interest is wastewater treatment technology using adsorption process.

Stephen obtained his PhD from the University of North Carolina at Charlotte in 2013 where his graduate research focused on cancer immunology and the tumor microenvironment. He received postdoctoral training in regenerative and translational medicine, specifically gastrointestinal tissue engineering, at the Wake Forest Institute of Regenerative Medicine. Currently, Stephen is an instructor for anatomy and physiology and biology at Forsyth Technical Community College.

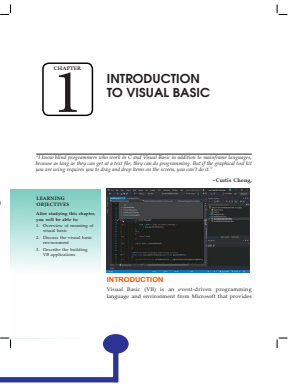
Michelle holds a Masters of Business Administration from the University of Phoenix, with a concentration in Human Resources Management. She is a professional author and has had numerous articles published in the Henry County Times and has written and revised several employee handbooks for various YMCA organizations throughout the United States.

HOW TO USE THE BOOK

This book has been divided into many chapters. Chapter gives the motivation for this book and the use of templates. The text is presented in the simplest language. Each paragraph has been arranged under a suitable heading for easy retention of concept. Keywords are the words that academics use to reveal the internal structure of an author's reasoning. Review questions at the end of each chapter ask students to review or explain the concepts. References provides the reader an additional source through which he/she can obtain more information regarding the topic.

LEARNING OBJECTIVES

See what you are going to cover and what you should already know at the start of each chapter


CHAPTER 1

INTRODUCTION TO VISUAL BASIC

"I know blind programmers who work in C and Visual Basic in addition to mainframe languages, because as long as they can get at a text file, they can do programming. But if the graphical tool kit you are using requires you to drag and drop items on the screen, you can't do it."

–Curtis Chong,

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Overview of meaning of visual basic
2. Discuss the visual basic programming
3. Describe the building VB applications

INTRODUCTION

Visual Basic (VB) is an event-driven programming language and environment from Microsoft that provides

enhancements, including the striking ability of creating web based applications. The extended support for Visual Basic 6.0 was ceased in the month of March in 2008. The basic parts of development environment of Visual Basic 6, however, still run in all the 32-bit Microsoft windows, including Windows 8.1.

After the cessation of mainstream and extended support for Visual Basic 6.0 caused a number of programs to show concern. The community members then created a lobby of users and a petition was signed by them. The basic aim of this petition was to ensure that the product remains alive. However, the petition did not attain its aim effectively.

1.1.2 The Importance of Visual Basic Programming Language

Visual Basic is regarded as the third generation event-drivon programming language. It was released in 1987. Being the first visual development tool from Microsoft, it is considered as one of the most powerful programming languages. As compared to other computer programming languages, such as, C, C++, it is easy to learn and understand, provided that one has determination and dedication to do so.

Visual basic programming language allows programmers to create software interface and codes in an easy to use graphical environment. VB is the combination of different components that are used on forms having specific attributes and actions with the help of those components. On the one hand it allows programmers to develop seldom based applications rapidly; on the other hand, it helps greatly in accessing data bases, using ADO while letting the programmers use ActiveX controls and various objects. While it is intended more to develop applications, it is also useful for games development for particular or limited purposes, unlike C++ that is more suitable for developing games.

As compared to other languages, Visual basic may be slower though, yet it is flexible and it can be rightly said that things that are difficult in other languages are comparatively easier in visual basic programming language. It may also be said that, since it is one of the most popular programming languages, lots of related books and material and other resources are available and can be accessed for developing programming skills at visual basic programming language conveniently.

One of the most important things to be considered with regard to programming in Visual basic is that the structure of VB is designed in a way that allows programmers to create executable code – Exe files. It enables programmers to develop programs that can be used as front end to databases. Besides, it's with the help of visual basic tools, one can change the abstract ideas into programs or into the whole software while it allows revising and modifying the programs fittingly.

ABOUT THIS CHAPTER

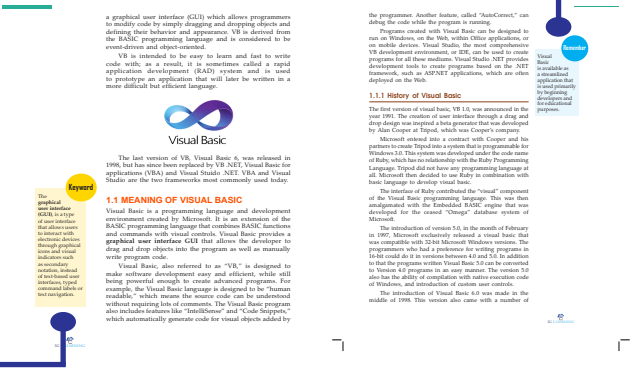
An introduction is a beginning of section which states the purpose and goals of the topics which are discussed in the chapter. It also starts the topics in brief.

REMEMBER

This revitalizes a must read information of the topic.

KEYWORDS

This section contains some important definitions that are discussed in the chapter. A keyword is an index entry that identifies a specific record or document. It also gives the extra information to the reader and an easy way to remember the word definition.


Keyword

The graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to of text-based user interfaces, typed command labels or text navigation.

a graphical user interface (GUI) which allows programmers to modify code by simply dragging and dropping objects and defining their behavior and appearance. VB is derived from the BASIC programming language and is considered to be event-driven and object-oriented.

VB is intended to be easy to learn and fast to write code with; as a result, it is sometimes called a rapid application development (RAD) system and is used to prototype an application that will later be written in a more difficult but efficient language.

Visual Basic

The last version of VB, Visual Basic 6, was released in 1998, but has since been replaced by VB .NET, Visual Basic for applications (VBA) and Visual Studio .NET. VBA and Visual Studio are the two frameworks most commonly used today.

1.1 MEANING OF VISUAL BASIC

Visual Basic is a programming language and development environment created by Microsoft. It is an extension of the BASIC programming language that combines BASIC functions and commands with visual controls. Visual Basic provides a **graphical user interface GUI** that allows the developer to drag and drop objects into the program as well as manually write program code.

Visual Basic, also referred to as "VB," is designed to make software development easy and efficient, while still being powerful enough to create advanced programs. For example, the Visual Basic language is designed to be "human readable," which means the source code can be understood without requiring lots of comments. The Visual Basic program also includes features like "IntelliSense" and "Code Snippets," which automatically generate code for visual objects added by

the programmer. Another feature, called "AutoCorrect," can debug the code while the program is running.

Programs created with Visual Basic can be designed to run on Windows, on the Web, within Office applications, or on mobile devices. Visual Studio, the most comprehensive VB development environment, or IDE, can be used to create programs for all these mediums. Visual Studio .NET provides development tools to create programs based on the .NET framework, such as ASP.NET applications, which are often deployed on the Web.

Remember

Visual Basic is available as a standalone application that is used primarily by beginning developers and for educational purposes.

1.1.1 History of Visual Basic

The first version of visual basic, VB 1.0, was announced in the year 1991. The creation of user interface through a drag and drop design was inspired a beta generator that was developed by Alan Cooper at Tripod, which was Cooper's company.

Microsoft entered into a contract with Cooper and his partners to create Tripod into a system that is programmable for Windows 3.0. This system was developed under the code name of Ruby, which has no relationship with the Ruby Programming Language. Tripod did not have any programming language at all. Microsoft then decided to use Ruby in combination with basic language to develop visual basic.

The interface of Ruby contributed the "visual" component of the Visual Basic programming language. This was then amalgamated with the Embedded BASIC engine that was developed for the ceased "Omega" database system of Microsoft.

The introduction of version 5.0, in the month of February in 1997, Microsoft exclusively released a visual basic that was compatible with 32-bit Microsoft Windows versions. The programmers who had a preference for writing programs in 16-bit could do it in versions between 4.0 and 5.0. In addition to that the programs written Visual Basic 5.0 can be converted to Version 4.0 programs in an easy manner. The version 5.0 also has the ability of compilation with native execution code of Windows, and introduction of custom user controls.

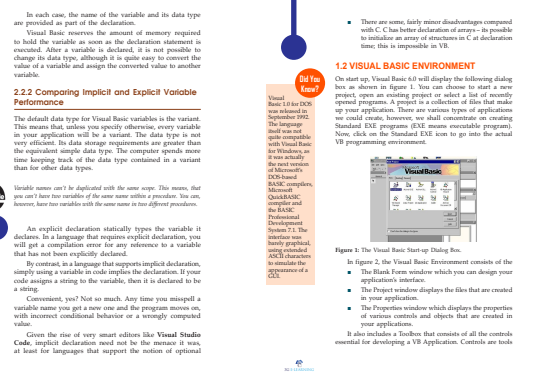
The introduction of Visual Basic 6.0 was made in the middle of 1998. This version also came with a number of

DID YOU KNOW?

This section equip readers the interesting facts and figures of the topic.

EXAMPLE

The book cabinets' examples to illustrate specific ideas in each chapter.



In each case, the name of the variable and its data type are provided as part of the declaration.

Visual Basic reserves the amount of memory required to hold the variable as soon as the declaration statement is executed. After a variable is declared, it is not possible to change its data type, although it is quite easy to convert the value of a variable and assign the converted value to another variable.

2.2.2 Comparing Implicit and Explicit Variable Performance

The default data type for Visual Basic variables is the variant. This means that, unless you specify otherwise, every variable in your application will be a variant. The data type is not very efficient. Its data storage requirements are greater than the equivalent simple data type. The computer spends more time keeping track of the data type contained in a variant than for other data types.

Variable names can't be duplicated with the same scope. This means, that you can't have two variables of the same name within a procedure. You can, however, have two variables with the same name in two different procedures.

An explicit declaration statically types the variable it declares. In a language that requires explicit declaration, you will get a compilation error for any reference to a variable that has not been explicitly declared.

By contrast, in a language that supports implicit declaration, simply using a variable in code implies the declaration. If your code assigns a string to the variable, then it is declared to be a string.

Convenient, yes? Not so much. Any time you misspell a variable name you get a new one and the program moves on, with incorrect conditional behavior or a wrongly computed value.

Given the rise of very smart editors like **Visual Studio Code**, implicit declaration need not be the menace it was, at least for languages that support the notion of optional

- There are some, fairly minor disadvantages compared with C. C has better declaration of arrays – its possible to initialize an array of structures in C at declaration time; this is impossible in VB.

1.2 VISUAL BASIC ENVIRONMENT

Did You Know?

Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, Microsoft QuickBASIC compiler and the BASIC Professional Development System 7.1. The interface was barely graphical, using extended ASCII characters to simulate the appearance of a GUI.

On start up, Visual Basic 6.0 will display the following dialog box as shown in figure 1. You can choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications we could create, however, we shall concentrate on creating Standard EXE programs (EXE means executable program). Now, click on the Standard EXE icon to go into the actual VB programming environment.

Figure 1: The Visual Basic Start-up Dialog Box.

In figure 2, the Visual Basic Environment consists of the

- The Blank Form window which you can design your application's interface.
- The Project window displays the files that are created in your application.
- The Properties window which displays the properties of various controls and objects that are created in your applications.

It also includes a Toolbox that consists of all the controls essential for developing a VB Application. Controls are tools

ROLE MODEL

A biography of someone who has/had acquired remarkable success in their respective field as Role Models are important because they give us the ability to imagine our future selves.

CASE STUDY

This reveals what students need to create and provide an opportunity for the development of key skills such as communication, group working and problem solving.

ROLE MODEL

ALAN COOPER: FATHER OF VISUAL BASIC

Born in San Francisco in 1952 and raised in Marin County, California, Alan Cooper has always taken the path less traveled. A rebellious teenager, he dropped out of high school, but eventually made his way to the College of Marin to pursue his interest in architecture. After an exploratory course in programming, it became clear that his future was in architecture—software architecture. After getting his associate degree and a COBOL programming job, he saw an advertisement for one of the first personal computers and .conceived an idea for a new business venture

In 1976, Cooper founded Structured Systems Group (SSG), a company Fire in the Valley authors Paul Freiberger and Michael Swaine said created "the first serious business software for microcomputers." In four years, Cooper wrote and shipped a dozen application programs. SSG became the archetype for many software startups in the early days of the personal computer revolution.

During the 1980s, after leaving SSG, Cooper invented, wrote, and sold three major software packages to prominent publishers. One of those was the visual programming front-end code named "Ruby," for what became Visual BASIC. Bill Gates purchased it from Cooper in 1988, noting that it would have significant impact across Microsoft's entire product line. Visual BASIC was deemed both a commercial and critical success, earning Cooper the moniker "Father of Visual BASIC." Visual BASIC has influenced integrated development languages ever since.

In 1990 Cooper became fascinated with the challenge of making software products that were easy to use and understand. He and his wife, Susan, founded Cooper Interaction Design (now "Cooper") to assist in what Cooper calls "interaction design." In the design field, Cooper's software development background was unique and, over the next few years, he invented many of the tools and techniques now standard in the user experience industry, including personas and scenarios,

CASE STUDY

FUJITSU FACILITATES SMOOTH MIGRATION TO VB.NET AT AN POST

Fujitsu has an excellent technical team, which works closely with our staff. We have had a good working relationship for many years and Fujitsu has an in-depth knowledge of our mission-critical application gained from several years' development and support work.

Challenge

A Post, one of Ireland's largest companies, is a major commercial organization providing a wide range of postal, communication, retail and financial services. With 9,600 employees throughout its national network of retail, processing and delivery points, the business also provides services to government departments, the National Treasury Management Agency and its own National Lottery Company. A decade ago, A Post implemented a new nationwide time and attendance system to calculate and record staff salary and wages functions. The Staff Remuneration and Administration Management System (STREAMS) is a bespoke, mission-critical application developed by Fujitsu as a reliable, scalable client server system using Microsoft technologies. The STREAMS front-end system gathers information and feeds the data to the company's HR, payroll and financial departments. It primarily creates more efficient processes for A Post to capture data for the weekly payroll run whilst simultaneously minimizing the number of payroll queries by employees. Following deployment, STREAMS improved cost center reporting, significantly lowered the time to record pay details and enhanced the processing of casual staff pay. During this period, Fujitsu provided quality support and maintenance services and application enhancements to increase functionality, ensuring the long-term reliability of STREAMS. For instance, as employee numbers steadily increased to exceed original expectations, Fujitsu boosted system performance by upgrading the infrastructure and optimizing the software. STREAMS originally employed Visual Basic (VB), a third-generation event-driven programming language and integrated development environment (IDE) from Microsoft. IDE provides programmers with comprehensive facilities for software development and comprises a source code editor, a compiler and/or an interpreter, build automation tools and a debugger. However, Microsoft no longer supports VB version 6.0, the edition employed by A Post. Syl Byrne, IT Manager Remuneration Services, A Post, explains: "To ensure that our business-critical application is future-proof, we needed to move to a platform that Microsoft will support for the foreseeable future." A Post therefore decided to migrate STREAMS to the VB.NET platform, an object-orientated programming language. This strategy would protect its investment for the next 10 years by creating a secure, scalable

KNOWLEDGE CHECK

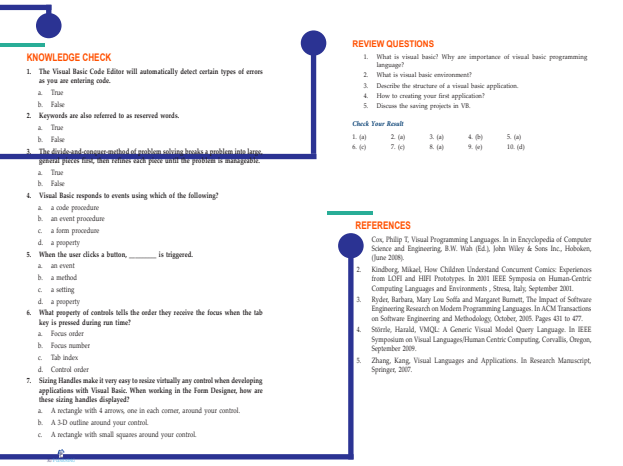
This is given to the students for progress check at the end of each chapter.

REVIEW QUESTIONS

This section is to analyze the knowledge and ability of the reader.

REFERENCES

References refer those books which discuss the topics given in the chapters in almost same manner.



KNOWLEDGE CHECK

1. **The Visual Basic Code Editor will automatically detect certain types of errors as you are entering code.**
   a. True
   b. False
2. **Keywords are also referred to as reserved words.**
   a. True
   b. False
3. **The divide-and-conquer method of problem solving breaks a problem into large general pieces first, then refines each piece until the problem is manageable.**
   a. True
   b. False
4. **Visual Basic responds to events using which of the following?**
   a. a code procedure
   b. an event procedure
   c. a form procedure
   d. a property
5. **When the user clicks a button, _____ is triggered.**
   a. an event
   b. a method
   c. a setting
   d. a property
6. **What property of controls tells the order they receive the focus when the tab key is pressed during run time?**
   a. Focus order
   b. Focus number
   c. Tab index
   d. Control order
7. **Sizing Handles make it very easy to resize virtually any control when developing applications with Visual Basic. When working in the Form Designer, how are these sizing handles displayed?**
   a. A rectangle with 4 arrows, one in each corner, around your control.
   b. A 3-D outline around your control.
   c. A rectangle with small squares around your control.

REVIEW QUESTIONS

1. What is visual basic? Why are importance of visual basic programming language?
2. What is visual basic environment?
3. Describe the structure of a visual basic application.
4. How to creating your first application?
5. Discuss the saving projects in VB.

Check Your Result

1. (a)	2. (a)	3. (a)	4. (b)	5. (a)
6. (c)	7. (c)	8. (a)	9. (e)	10. (d)

REFERENCES

1. Cox, Philip T, Visual Programming Languages. In in Encyclopedia of Computer Science and Engineering, B.W. Wah (Ed.), John Wiley & Sons Inc., Hoboken, (June 2008).
2. Kindborg, Mikael, How Children Understand Concurrent Comics: Experiences from LOFI and HIFI Prototypes. In 2001 IEEE Symposia on Human-Centric Computing Languages and Environments , Stresa, Italy, September 2001.
3. Ryder, Barbara, Mary Lou Soffa and Margaret Burnett, The Impact of Software Engineering Research on Modern Programming Languages. In ACM Transactions on Software Engineering and Methodology, October, 2005. Pages 431 to 477.
4. Störrle, Harald, VMQL: A Generic Visual Model Query Language. In IEEE Symposium on Visual Languages/Human Centric Computing, Corvallis, Oregon, September 2009.
5. Zhang, Kang, Visual Languages and Applications. In Research Manuscript, Springer, 2007.

TABLE OF CONTENTS

Chapter 8 SQL Table 197

PREFACE

Basically, SQL stands for Structured Query Language which is basically a language used by databases. SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. It has been around in some form since the 70s and is just about as ubiquitous as data management itself. In order to get the most of the mounds of data they collect, many businesses must become versed in SQL. Now into its third decade of existence, SQL offers great flexibility to users by supporting distributed databases, i.e. databases that can be run on several computer networks at a time. SQL is used in health care, business (inventories, trends analysis), and education. It even has applications in the defense industry.

Organization of the Book

This updated edition is systematically divided into ten chapters. You'll quickly learn how to put the power and flexibility of this language to work. This book is a guide to SQL covers such topics as retrieving records, metadata queries, working with strings, data arithmetic, date manipulation, reporting and warehousing, and hierarchical queries.

Chapter 1 presents an overview of SQL. You will take a look on SQL commands and SQL server. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.

Chapter 2 is intended to focus on SQL in perspective. In this chapter, you will learn about the SQL and database management, including with SQL standards although it is the most widely recognized, the ANSI/ISO standard is not the only standard for SQL. X/OPEN, a European vendor group, also adopted SQL as part of its suite of standards for a portable application environment based on UNIX.

Chapter 3 aims to data retrieval i.e. obtaining data from a database management system such as ODBMS. In order to retrieve the desired data the user present a set of criteria by a query.

Chapter 4 focuses on updating data. The topics, adding data to the database, deleting data from the database, and modifying data in the database are discussed in this chapter.

Chapter 5 is aimed to explain the database design, the database schema versus database instance, and the database models.

Chapter 6 is intended to focus on programming with SQL such as embedded SQL, dynamic SQL, and SQL APIs.

Chapter 7 is focused on SQL security concept. Security is especially important in an SQL-based DBMS because interactive SQL makes database access very easy.

Chapter 8 aims to cover SQL Table. The data in a table does not have to be physically stored in the database. Views also function as relational tables, but their data are calculated at query time. External tables (in Informix or Oracle, for example) can also be thought of as views.

Chapter 9 explores on SQL Clause. Clauses are in-built functions available to us in SQL. With the help of clauses, we can deal with data easily stored in the table. Clauses help us filter and analyze data quickly. When we have large amounts of data stored in the database, we use Clauses to query and get data required by the user.

Chapter 10 is database processing and stored procedural SQL. With the advent of relational databases and SQL, the DBMS took on expanded responsibilities. Database searching and sorting were embodied in SQL language clauses and provided by the DBMS, along with the capability to summarize data.

CHAPTER

1

AN OVERVIEW OF SQL

"SQL Developer does it all. When you need a quick overview of a client database, you plug in your USB key with SQL Developer on and run it directly from the key. No need for install, no need for permissions or licenses. It saves a lot of work, when working on different hardware and different customers, and you need to browse the data dictionary."

——Jakob Hammer-Jakobsen

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss about the SQL language
2. Explain SQL commands
3. Describe SQL server

INTRODUCTION

SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system

(RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL offers two main advantages over older read–write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify *how* to reach a record, e.g. with or without an index.

Originally based upon relational algebra and tuple relational calculus, SQL consists of many types of statements, which may be informally classed as sublanguages, commonly: a data query language (DQL), a data definition language (DDL), a data control language (DCL), and a data manipulation language (DML). The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control. Although SQL is essentially a declarative language (4GL), it also includes procedural elements.

1.1 THE SQL LANGUAGE

SQL stands for Structured Query Language. SQL is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

1.1.1 A Brief History of SQL

1970 – Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.

1974 – Structured Query Language appeared.

1978 – IBM worked to develop Codd's ideas and released a product named System/R.

1986 – IBM developed the first prototype of **relational database** and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

1.1.2 SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

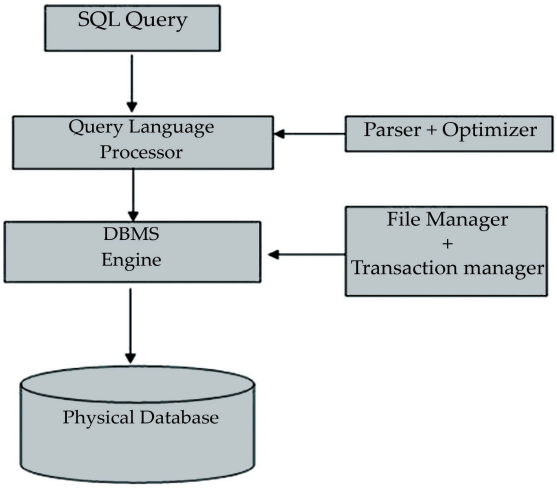
There are various components included in this process.

These components are –

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture –



In a distributed database system, a program often referred to as the database's "back end" runs constantly on a server, interpreting data files on the server as a standard relational database. Programs on client computers allow users to manipulate that data, using tables, columns, rows, and fields. To do this, client programs send SQL statements to the server. The server then processes these statements and returns result sets to the client program.

SELECT Statements

An SQL SELECT statement retrieves records from a database table according to clauses (e.g., FROM and WHERE) that specify criteria. The syntax is:

Keyword

Relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

```
SELECT column1, column2 FROM table1, table2 WHERE column2='value';
```

In the above SQL statement:

- The SELECT clause specifies one or more columns to be retrieved; to specify multiple columns, use a comma and a space between column names. To retrieve all columns, use the wild card * (an asterisk).
- The FROM clause specifies one or more tables to be queried. Use a comma and space between table names when specifying multiple tables.
- The WHERE clause selects only the rows in which the specified column contains the specified value. The value is enclosed in single quotes (e.g., WHERE last_name='Vader')
- The semicolon (;) is the statement terminator. Technically, if you're sending only one statement to the back end, you don't need the statement terminator; if you're sending more than one, you need it. It's best practice to include it.

Keyword

Clause is a group of words that contains a verb (and usually other components too).

Note**:** SQL is not case sensitive (i.e., SELECT is the same as select). For readability purposes, some programmers use uppercase for commands and **clauses**, and lowercase for everything else.

Examples

Following are examples of SQL SELECT statements:

- To select all columns from a table (Customers) for rows where the Last_Name column has Smith for its value, you would send this SELECT statement to the server back end:

```
SELECT * FROM Customers WHERE Last_Name='Smith';
```

The server back end would reply with a result set similar to this:

```
+---------+-----------+------------+
| Cust_No | Last_Name | First_Name |
+---------+-----------+------------+
| 1001    | Smith     | John       |
| 2039    | Smith     | David      |
| 2098    | Smith     | Matthew    |
+---------+-----------+------------+
3 rows in set (0.05 sec)
```

- To return only the Cust_No and First_Name columns, based on the same criteria as above, use this statement:

```
SELECT Cust_No, First_Name FROM Customers WHERE
Last_Name='Smith';
```

The subsequent result set might look like:

```
+---------+------------+
| Cust_No | First_Name |
+---------+------------+
| 1001    | John       |
| 2039    | David      |
| 2098    | Matthew    |
+---------+------------+
3 rows in set (0.05 sec)
```

To make a WHERE clause find inexact matches, add the pattern-matching operator LIKE. The LIKE operator uses the % (percent symbol) wild card to match zero or more characters, and the underscore (_) wild card to match exactly one character. For example:

- To select the First_Name and Nickname columns from the Friends table for rows in which the Nicknamecolumn contains the string “brain”, use this statement:

```
SELECT First_Name, Nickname FROM Friends WHERE Nickname
LIKE '%brain%';
```

The subsequent result set might look like:

```
+------------+------------+
| First_Name | Nickname   |
+------------+------------+
| Ben        | Brainiac   |
| Glen       | Peabrain   |
| Steven     | Nobrainer  |
+------------+------------+
3 rows in set (0.03 sec)
```

- To query the same table, retrieving all columns for rows in which the First_Name column’s value begins with any letter and ends with “en”, use this statement:

```
SELECT * FROM Friends WHERE First_Name LIKE '_en';
```

The result set might look like:

```
+------------+------------+-----------+
| First_Name | Last_Name  | Nickname  |
+------------+------------+-----------+
| Ben        | Smith      | Brainiac  |
| Jen        | Peters     | Sweetpea  |
+------------+------------+-----------+
2 rows in set (0.03 sec)
```

- If you used the % wild card instead (e.g., '%en') in the example above, the result set might look like:

```
+------------+------------+-----------+
| First_Name | Last_Name  | Nickname  |
+------------+------------+-----------+
| Ben        | Smith      | Brainiac  |
| Glen       | Jones      | Peabrain  |
| Jen        | Peters     | Sweetpea  |
| Steven     | Griffin    | Nobrainer |
+------------+------------+-----------+
4 rows in set (0.05 sec)
```

Remember

SQL can execute queries against a database.

1.1.3 The Role of SQL

SQL is not itself a database management system, nor is it a stand-alone product. You cannot go to a computer retailer or a web site selling computer software and buy SQL. Instead, SQL is an integral part of a database management system, a language and a tool for communicating with the DBMS. Figure 1 shows some of the components of a typical DBMS and how SQL links them together.

The database engine is the heart of the DBMS, responsible for actually structuring, storing, and retrieving the data in the database. It accepts SQL requests from other DBMS components (such as a forms facility, report writer, or interactive query facility), from user-written application programs, and even from other computer systems. As the figure shows, SQL plays many different roles:

- SQL is an interactive query language. Users type SQL commands into an interactive SQL program to retrieve data and display it on the screen, providing a convenient, easy-to-use tool for ad hoc database queries.

- SQL is a database programming language. Programmers embed SQL commands into their application programs to access the data in a database. Both user-written programs and database utility programs (such as report writers and data entry tools) use this technique for database access.

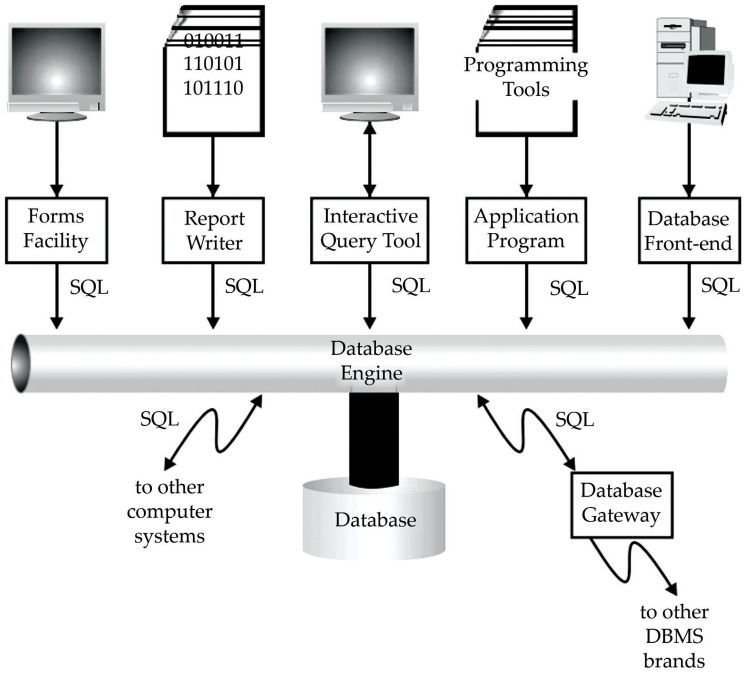


Figure 1: Components of a typical database management system.

- SQL is a database administration language. The database administrator responsible for managing a minicomputer or mainframe database uses SQL to define the database structure and to control access to the stored data.
- SQL is a client/server language. Personal computer programs use SQL to communicate over a network with database servers that store shared data. This client/server architecture is used by many popular enterprise-class applications.
- SQL is an Internet data access language. Internet web servers that interact with corporate data and Internet application servers all use SQL as a standard

Did You Know?

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

language for accessing corporate databases, often by embedding SQL database access within popular scripting languages like PHP or Perl.

- SQL is a distributed database language. Distributed database management systems use SQL to help distribute data across many connected computer systems. The DBMS software on each system uses SQL to communicate with the other systems, sending requests for data access.
- SQL is a database gateway language. In a computer network with a mix of different DBMS products, SQL is often used in a gateway that allows one brand of DBMS to communicate with another brand.

SQL has thus emerged as a useful, powerful tool for linking people, **computer programs**, and computer systems to the data stored in a relational database.

Keyword

Computer program is a collection of instructions that performs a specific task when executed by a computer.

1.1.4 Applications of SQL (Structured Query Language)

Have a look at some main SQL applications:

Data Integration Scripts

The main application of SQL is to write data integration scripts by the database administrators and developers.

Analytical Queries

The data analysts use structured query language for setting and running analytical queries on a regular basis.

Retrieve Information

Another popular application of this language is to retrieve the subsets of information within a database for analytics applications and transaction processing. The most commonly used SQL elements are select, insert, update, add, delete, create, truncate and alter.

Other Important Applications

The SQL is used for modification of the index structures and database table. Additionally, the users can add, update and delete the rows of the data by using this language.

1.1.5 Advantages of SQL

There are numerous advantages of Structured Query Language and some of them are mentioned below:

No Coding Needed

It is very easy to manage the database systems without any need to write the substantial amount of code by using the standard SQL.

Well Defined Standards

Long established are used by the SQL databases that is being used by ISO and ANSI. There are no standards adhered by the non-SQL databases.

Portability

SQL can be used in the program in PCs, servers, laptops, and even some of the mobile phones.

Interactive Language

This domain language can be used for communicating with the databases and receive answers to the complex questions in seconds.

Multiple data views

With the help of SQL language, the users can make different views of database structure and databases for the different users.

1.1.6 Disadvantages of SQL

Along with some benefits, the Structured query language also has some certain disadvantages:

Difficult Interface

SQL has a complex interface that makes it difficult for some users to access it.

Partial Control

The programmers who use SQL doesn't have a full control over the database because of the hidden business rules.

Implementation

Some of the databases go to the proprietary extensions to standard SQL for ensuring the vendor lock-in.

Cost

The operating cost of some SQL versions makes it difficult for some programmers to access it.

1.2 SQL COMMANDS

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature –

DDL - Data Definition Language

Sr.No.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

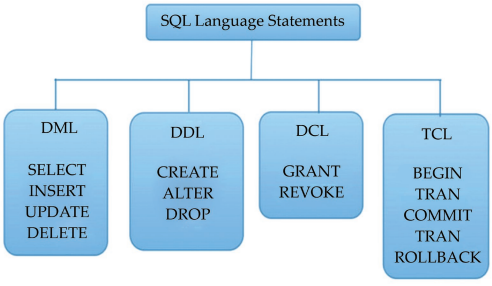
Sr.No.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

Keyword

Database administration is the function of managing and maintaining database management systems (DBMS) software.

1.2.1 Types of SQL Commands

The following sections discuss the basic categories of commands used in SQL to perform various functions. These functions include building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration.



The main categories are:

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DQL (Data Query Language)
- DCL (Data Control Language)
- Data administration commands
- Transactional control commands

Defining Database Structures

Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.

Remember

SQL can set permissions on tables, procedures, and views.

Manipulating Data

Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database.

There are three basic DML commands:

INSERT

UPDATE

DELETE

Selecting Data

Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is as follows:

SELECT

This command, accompanied by many options and clauses, is used to compose queries against a relational database. Queries, from simple to complex, from vague to specific, can be easily created.

A *query* is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command line prompt.

Data Control Language

Data control commands in SQL allow you to control access to data within the database. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

ALTER PASSWORD

GRANT

REVOKE

CREATE SYNONYM

GRANT

Syntax

GRANT privilege_name
ON object_name

TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];

Data Administration Commands

Data administration commands allow the user to perform audits and perform analyses on operations within the database. They can also be used to help analyze system performance. Two general data administration commands are as follows:

START AUDIT

STOP AUDIT

Do not get data administration confused with database administration. *Database administration* is the overall administration of a **database**, which envelops the use of all levels of commands. *Database administration* is much more specific to each SQL implementation than are those core commands of the SQL language.

Database is an organized collection of data, stored and accessed electronically.

Transactional Control Commands

In addition to the previously introduced categories of commands, there are commands that allow the user to manage database transactions.

- COMMIT Saves database transactions
- ROLLBACK Undoes database transactions
- SAVEPOINT Creates points within groups of transactions in which to ROLLBACK
- SET TRANSACTION Places a name on a transaction

1.3 SQL SERVER

SQL Server is Microsoft's relational database management system (RDBMS). It is a full-featured database primarily designed to compete against competitors Oracle Database (DB) and MySQL.

Remember

Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

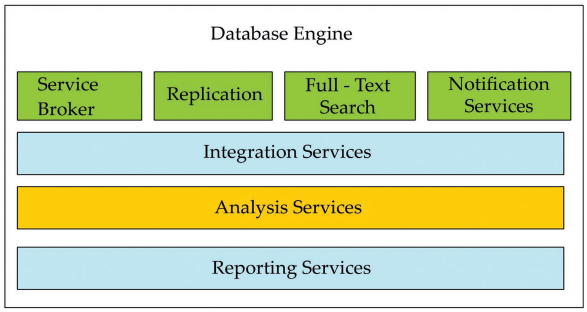
Like all major RBDMS, SQL Server supports ANSI SQL, the standard SQL language. However, SQL Server also contains T-SQL, its own SQL implementation. SQL Server Management Studio (SSMS) (previously known as Enterprise Manager) is SQL Server's main interface tool, and it supports 32-bit and 64-bit environments.

SQL Server is sometimes referred to as MSSQL and Microsoft SQL Server.

1.3.1 SQL Server Components

SQL Server contains a number of components. Each component is provided with the specific services and support to the clients connected to the server.

The following diagram shows the components of the SQL Server



The server contains the following components:

- Database Engine
- Integration Services
- Analysis Services
- Reporting Services

Database Engine

The component provides support to store query, process and secure data on the server. It allows user to create and manage database objects. The following background services are provided by the engine.

Service Broker

It provides support for asynchronous communication between clients and the database server. The client sends a request to the server and continues to work. The broker ensures that the request is processed whenever the server is available.

Replication

It allows the user to copy and distribute data and database objects from one server to another. The servers can be located at remote locations to provide fast access to users.

Full Text search

It allows the user to implement fast and intelligent search in large databases. It allows searching records containing words and phrases.

Notification services

It allows generating and sending notification messages to the user or administrators about the event.

Integration Services

The service allows gathering and integrating varied data in a consistent format in a common database. The database is known as **data warehouse**. The warehouse contains integrated databases, text files or flat files

Keyword

Data warehouse is a system used for reporting and data analysis, and is considered a core component of business intelligence.

Analysis Services

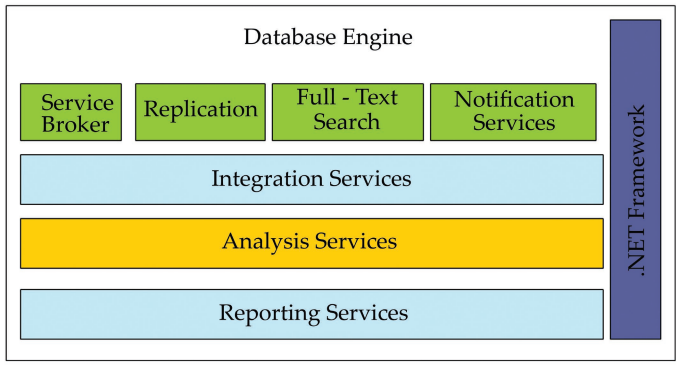
The warehouses are designed to facilitate reporting and analysis. The applications are widely using this data store for analytical purpose. The applications used for this purpose are known as BI applications.

Reporting Services

They provide support to generate complete reports on data in the database engine in the data warehouse. These services provide a set of tools that help in creating and managing reports in different formats.

1.3.2 SQL Server integration with the .NET Framework

The SQL Server is integrated with the .NET framework as shown in the following figure.



The .NET framework is an environment used to build, deploy, and execute business applications. The server used various services provided by the framework. The component uses the framework services to generate and send notification messages.

The .NET framework consists of the following components:

- Development tools and languages
- Base Class Library
- Common Language Runtime

Development Tools and Language

They are used to create interface for Windows forms, Web forms, and console applications. They include Visual Studio and Visual C# developer. The languages that can be used are VB.NET, C#, and F#.

Base Class Library

The framework consists of the class library that acts as a base class for any .NET language such as VB.NET or C#. The library is object oriented.

Common Language Runtime

It is the most important component in the framework. It provides the following features:

- Automatic memory management: It is used for allocating and de-allocating the memory of an application.
- Standard type system: It provides the user with some common data types known as Common Type System (CTS).
- Language Interoperability: It helps the user to create applications that can be used with many programming languages.
- Platform Independence: It allows the code execution from the platform that is supported by the CLR.

1.3.3 Features of SQL Server

The features provided by the SQL Server are as mentioned below:

- Scalability: It allows distributing data in the large tables into different file groups. The server can access the file groups simultaneously.
- CLR integration: It allows user to use the CLR features of the .NET Framework into the server database.

- Service oriented architecture: It provides distributed, asynchronous application framework for large scale applications.
- Web services support: It allows direct access to the data from web services by implementing the HTTP endpoints.
- High security: It implements high security by adding policies for log on and passwords.
- Support for data migration and analysis: It provides tools to migrate data from data sources to a common database.
- Policy based management: It is used to define a set of policies for configuring and managing data.
- Resource governor: It is used to manage the workload of the server by allocating and managing resources.

1.3.4 SQL Statements

The SQL statements can be divided into following categories.

Data Definition Language (DDL)

It is used to define database, data types and data structure and constraints. Such DDL statements are as follows:

- **CREATE:** It is used to create a new database object.
- **ALTER:** It is used to modify the database objects
- **DROP:** It is used to delete the objects

Data Manipulation Language (DML)

It is used to manipulate the data in the database objects. Some of the DML statements are as follows:

- **INSERT:** It is used to insert the new data record into the table
- **UPDATE:** It is used to modify the existing record in the table
- **DELETE:** It is used to delete a record from a table

Data Control Language

It is used to control the data access in the database. Some of the DCL statements are as follows:

- **GRANT:** It is used to assign the permissions to users to access the objects
- **REVOKE:** It is used to deny the permissions to users to access the objects

Data Query Language

It is sued to query data from database objects. The SELECT statement is used to select the data from the database.

ROLE MODEL

DONALD D. CHAMBERLIN:

Known For his Fundamental Work on Structured Query Language (SQL) and Database Architectures

Donald D. Chamberlin (born 21 December 1944) is an American computer scientist who is best known as one of the principal designers of the original SQL language specification with Raymond Boyce. He also made significant contributions to the development of XQuery.

Biography

Donald D. Chamberlin was born in San Jose, California. After attending Campbell High School, he studied engineering at Harvey Mudd College from where he holds a B.S. After graduating, he went to Stanford University on a National Science Foundation grant. At Stanford, he studied electrical engineering and minored in computer science. Chamberlin holds an M.Sc and a PhD degree in electrical engineering from Stanford University. After graduating, Chamberlin went to work for IBM Research at the Yorktown Heights research facility in New York, where he had previously had a summer internship.

Chamberlin is probably best known as co-inventor of SQL (Structured Query Language), the world's most widely used database language. Developed in the mid-1970s by Chamberlin and Raymond Boyce, SQL was the first commercially successful language for relational databases. Chamberlin also was one of the managers of IBM's System R project, which produced the first SQL implementation and developed much of IBM's relational database technology. System R, together with the Ingres project at U.C. Berkeley, received the ACM Software System Award in 1988. Until his retirement in 2009 he was based at the Almaden Research Center. He was appointed an IBM Fellow in 2003.

In 2000, jointly with Jonathan Robie and Daniela Florescu, he drafted a proposal for an XML query language called Quilt. Many ideas from this proposal found their way into

the XQuery language specification, which was developed by W3C with Chamberlin as an editor. XQuery became a W3C Recommendation in January 2007.

Chamberlin is also an ACM Fellow, IEEE Fellow and a member of the National Academy of Engineering. In 2005, he was awarded an honorary doctorate by the University of Zurich.

In 2009, he was made a Fellow of the Computer History Museum "for his fundamental work on structured query language (SQL) and database architectures."

SUMMARY

- SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.
- SQL stands for Structured Query Language. SQL is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.
- The database engine is the heart of the DBMS, responsible for actually structuring, storing, and retrieving the data in the database. It accepts SQL requests from other DBMS components (such as a forms facility, report writer, or interactive query facility), from user-written application programs, and even from other computer systems.
- The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP.
- Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.
- Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database.
- SQL Server is Microsoft's relational database management system (RDBMS). It is a full-featured database primarily designed to compete against competitors Oracle Database (DB) and MySQL.
- SQL Server contains a number of components. Each component is provided with the specific services and support to the clients connected to the server.
- The .NET framework is an environment used to build, deploy, and execute business applications. The server used various services provided by the framework. The component uses the framework services to generate and send notification messages.

KNOWLEDGE CHECK

1. You can add a row using SQL in a database with which of the following?

a. ADD
b. CREATE
c. INSERT
d. MAKE

2. The command to remove rows from a table 'CUSTOMER' is:

a. REMOVE FROM CUSTOMER ...
b. DROP FROM CUSTOMER ...
c. DELETE FROM CUSTOMER WHERE ...
d. UPDATE FROM CUSTOMER ...

3. The SQL WHERE clause:

a. limits the column data that are returned.
b. limits the row data are returned.
c. Both A and B are correct.
d. Neither A nor B are correct.

4. Which of the following is the original purpose of SQL?

a. To specify the syntax and semantics of SQL data definition language
b. To specify the syntax and semantics of SQL manipulation language
c. To define the data structures
d. All of the above.

5. The wildcard in a WHERE clause is useful when?

a. An exact match is necessary in a SELECT statement.
b. An exact match is not possible in a SELECT statement.
c. An exact match is necessary in a CREATE statement.
d. An exact match is not possible in a CREATE statement.

6. What is the full form of SQL?

a. Structured Query List
b. Structure Query Language
c. Sample Query Language
d. None of these.

7. **Which of the following is not a valid SQL type?**
   a. FLOAT
   b. NUMERIC
   c. DECIMAL
   d. CHARACTER
8. **Which of the following is not a DDL command?**
   a. TRUNCATE
   b. ALTER
   c. CREATE
   d. UPDATE
9. **Which is the subset of SQL commands used to manipulate Oracle Database structures, including tables?**
   a. Data Definition Language(DDL)
   b. Data Manipulation Language(DML)
   c. DDL and DML
   d. None of the Mentioned
10. **Which of the following are TCL commands?**
   a. COMMIT and ROLLBACK
   b. UPDATE and TRUNCATE
   c. SELECT and INSERT
   d. GRANT and REVOKE

REVIEW QUESTIONS

1. What is SQL? Write its applications.
2. What is the SQL CASE statement used for? Explain with an example?
3. What is the difference between DELETE and TRUNCATE commands?
4. What is an ALIAS command?
5. What port does SQL server run on?

Check Your Result

1. (c)	2. (c)	3. (b)	4. (d)	5. (b)
6. (b)	7. (c)	8. (d)	9. (a)	10. (a)

REFERENCES

1. http://whatisdbms.com/what-is-sql-applications-advantages-and-disadvantages/
2. http://www.informit.com/articles/article.aspx?p=29583&seqNum=3
3. https://searchsqlserver.techtarget.com/definition/SQL
4. https://www.c-sharpcorner.com/blogs/types-of-sql-statements-with-example
5. https://www.tutorialspoint.com/sql/sql-overview.htm
6. Itzik Ben-Gan. Inside Microsoft SQL Server 2008: T-SQL Querying: Microsoft Press, 2009
7. Joe Celko. SQL for Smarties: Advanced SQL Programming. -2nd Edition. - Morgan Kaufmann Publishers, 2000
8. Kalen Delaney. Inside Microsoft SQL Server 2005: The Storage Engine (Microsoft Press, 2006) ISBN 978-0735621053
9. Muthusamy Anantha Kumar. SQL Server and Collation, 2004

CHAPTER

2

SQL IN PERSPECTIVE

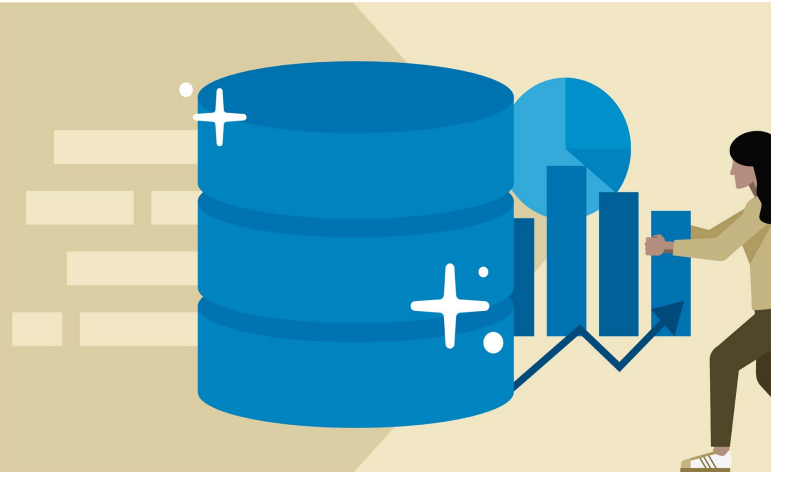
"Computing should be taught as a rigorous - but fun - discipline covering topics like programming, database structures, and algorithms. That doesn't have to be boring."

—Geoff Mulgan

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Learn about the SQL and database management
2. Discuss on SQL standard



INTRODUCTION

A perspective is a definition that allows users to see a cube in a simpler way. A perspective is a subset of the features of a cube. A perspective enables administrators to

create views of a cube, helping users to focus on the most relevant data for them. A perspective contains subsets of all objects from a cube. A perspective cannot include elements that are not defined in the parent cube.

A simple Perspective object is composed of: basic information, dimensions, measure groups, calculations, KPIs, and actions. Basic information includes the name and the default measure of the perspective. The dimensions are a subset of the cube dimensions. The measure groups are a subset of the cube measure groups. The calculations are a subset of the cube calculations. The KPIs are a subset of the cube KPIs. The actions are a subset of the cube actions.

SQL is a standardized query language for requesting information from a database. The original version called SEQUEL (structured English query language) was designed by an IBM research center in 1974 and 1975. SQL was first introduced as a commercial database system in 1979 by Oracle Corporation.

Historically, SQL has been the favorite query language for database management systems running on minicomputers and mainframes. Increasingly, however, SQL is being supported by PC database systems because it supports distributed databases (databases that are spread out over several computer systems). This enables several users on a local-area network to access the same database simultaneously.

2.1 SQL AND DATABASE MANAGEMENT

Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy. Let's discuss few examples. An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc. Your electricity service provider is obviously using a database to manage billing, client related issues, to handle fault data, etc. Let's also consider the Facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more.

We can provide countless number of examples for usage of databases.

One of the major tasks of a computer system is to store and manage data. To handle this task, specialized computer programs known as database management systems began to appear in the late 1960s and early 1970s. A database management system, or DBMS, helped computer users to organize and structure their data and allowed the computer system to play a more active role in managing the data. Although database management systems were first developed on large mainframe systems, their popularity quickly spread to minicomputers, and then to personal computers and workstations. Today, many database management systems operate on specialized server computers. Database management has also played a key role in the explosion of computer networking and the Internet. Early database systems ran on large, monolithic computer systems, where the data, the database management software, and the user

or application program accessing the database all operated on the same system. The 1980s and 1990s saw the explosion of a new client/server model for database access, in which a user or an application program running on a personal computer accesses a database on a separate computer system using a network. In the late 1990s, the increasing popularity of the Internet and the World Wide Web intertwined the worlds of networking and data management even further. Now users require little more than a web browser to access and interact with databases, not only within their own organizations, but around the world. Often, these Internet-based architectures involve three or more separate computer systems—one computer that runs the web browser and interacts with the user, connected to a second system that runs an application program or application server, which is in turn connected to a third system that runs the database management system.

Today, database management is very big business. Independent software companies and computer vendors ship billions of dollars' worth of database management products every year. The vast majority of enterprise-class computer applications that support the daily operation of large companies and other organizations use databases. These applications include some of the fastest-growing application categories, such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), Sales Force Automation (SFA), and financial applications. Computer manufacturers develop and deliver server computers that are specially configured as database servers; these systems constitute a multibillion-dollar-per-year market of their own. Databases provide the intelligence behind most transaction-oriented web sites, and they are used to capture and to analyze user interactions with web sites. Database management thus touches every segment of the computer market. Since the late 1980s a specific type of DBMS, called a relational database management system (RDBMS), has become so popular that it is the standard database form. Relational databases organize data in a simple, tabular form and provide many advantages over earlier types of databases. SQL is specifically a relational database language used to work with relational databases.

2.1.1 Database Management System

As one of the oldest components associated with computers, the database management system, or DBMS, is a computer software program that is designed as the means of managing all databases that are currently installed on a system hard drive or network. Different types of database management systems exist, with some of them designed for the oversight and proper control of databases that are configured for specific purposes.

As the tool that is employed in the broad practice of managing databases, the DBMS is marketed in many forms. Some of the more popular examples of these solutions include Microsoft Access, FileMaker, DB2, and Oracle. All these products provide for the creation of a series of rights or privileges that can be associated with

a specific user. This means that it is possible to designate one or more database administrators who may control each function, as well as provide other users with various levels of administration rights. This flexibility makes the task of using DBMS methods to oversee a system something that can be centrally controlled, or allocated to several different people.

Keyword

Modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules.

There are four essential elements that are found with just about every example of DBMS currently on the market. The first is the implementation of a modeling language that serves to define the language of each database that is hosted via the system. There are several approaches currently in use, with hierarchical, network, relational, and object examples. Essentially, the **modeling language** ensures the ability of the databases to communicate with the DBMS and thus operate on the system.

Second, data structures also are administered by the DBMS. Examples of data that are organized by this function are individual profiles or records, files, fields and their definitions, and objects such as visual media. Data structures are what allows these systems to interact with the data without causing damage to the integrity of the data itself.

A third component of DBMS software is the data query language. This element is involved in maintaining the security of the database, by monitoring the use of login data, the assignment of access rights and privileges, and the definition of the criteria that must be employed to add data to the system. The data query language works with the data structures to make sure it is harder to input irrelevant data into any of the databases in use on the system.

Last, a mechanism that allows for transactions is an essential basic for any DBMS. This helps to allow multiple and concurrent access to the database by multiple users, prevents the manipulation of one record by two users at the same time, and preventing the creation of duplicate records.

2.1.2 Operations of DBMS

The database is a kind of data collection. It stores data, which is in connection with the given task, orderly. The access to the data is also taken care of by the database. Besides, it guarantees the protection of the data, and also protects the integration of

the data. The management of the data was also made easier by database management systems. The ANSI/SPARC model shows the connection between the user and of the physically stored data on the computer's mass storage. We distinguish three levels, based on that:

- Outer level, alias user view, which examines the data from user's point of view.
- Conceptual level, which includes all of the user views. In this level the database is given with logical schema.
- Inner level, alias physical level, it means the actual presentation of the data on the current computer.

When we talk about ANSI/SPARC model it is important to mention two things. These are the logical data independence and the physical independence. Physical independence means that if we change anything in the inner level it will not effect anything on the logical schema.

So, we will not have to perform changes on them. If any changes occur in the storage of data it will have no effect on the upper levels. The logical data independence is data independence between outer level and conceptual level.

Those program systems which are responsible for guaranteeing access to the database are called database management systems. Furthermore, the database management system takes care of the tasks of the inner maintenance of the database such as

- Create database
- Defining the content of the database
- Data storage
- Querying data
- Data protection
- Data encryption
- Access rights management
- Physical organization of the data structure

We must keep in mind how the architecture of the database has changed. Furthermore, it is also important how we can put these together. It is very important for the programmers, because they are in a situation where they have to choose what they are going to working with after they have got the order. Because, those are not good programmers or software developers who can only use one database management system, or those who can write programs only in one programming language. That is the expectation of an elementary school. If you get a task it is good if you can decide which route is the one you have to start. What database manager you should use and in which programming language you are going to write your program. Of course, one could not say that know all of the existing programming languages by heart. We will talk about two or three of them. But everybody knows who have tried to make web

pages that it might not be a good idea to start a webpage development for example with an aspx.net. In one hand, it is possible in the case of a bigger task that aspx.net is good. On the other hand, one could possibly do a smaller task with html code without putting any dynamism in it, or maybe in php the things could be done easier. These are specific things. Returning to the database architectures, now the question is in which environment certain database managers can do good performance. Because, it is not true that every database manager can satisfy our needs in all environments.

Local database

The first such architectural level is the local database: these are the "best". It contains a computer, a database, a user, nobody has any problem. The story started sometime around 1980s. Database managers have appeared in the computers. It was the world of the dBase, which was based on the Dos. (From the beginning, DOS did not allow multi users and to run on more paths.) Back then there were no such problems as web collusion or concurrent access. Such database were dBase 3, 4, 5, the developed version of this were Paradox 4, 5, 7, which had more stable data table management, but in return we have got a more damageable index table. The following things were true for all of them: one database - one file; one index - one file; one descriptor table - one file; one check term for a table – one file. If we had a database with 100 tables then there was created 100 files in a directory. These were managed by a database management engine. It worked on file levels, moved bytes and managed blocks. As it worked on file level it was damageable. There were a lot of files. So, there were already a big possibility of damage and big possibility of delete on the level of the operation system. If there was a power shortage, it was necessary to call the programmer, because the whole system has turned upside down. Something for something. I always say that these are dangerous systems, especially, if we do not use them in local database system. Nowadays, it would be very hard to use local database. The MS Access is also belonging to there. It is only more modern, because of the fact that all of the tools, data and descriptive tools are stored in the same file. From there it knows it knows the same as Paradox or dBase. It could become very damageable if we want to use under bigger stress. They are perfect for teaching (ECDL, for final examination). The LibreOffice also has the Base database. That is similar to Access. It is also free, and it is good for familiarization and teaching. These database managers have limits. In a traffic table the numbers of records are continuously growing. It can easily reach the quantity of 100000. It may seem to be more, however if somebody write a system that is also being used, it turns out to be few. One could not say that up to 100000 it works well, but at 100001 the whole system fall apart. It works well two to three hundred-thousand, but after it more and more error occurs. The system is start slowing down and index damages are coming up. So, the efficiency of the local, file-based systems

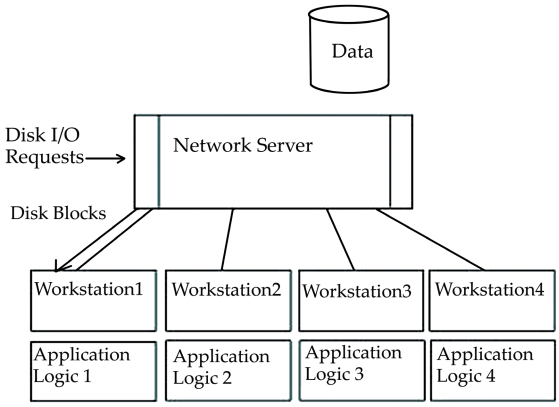
has the volume of 100000. If we know that and if we know the kind of work they want to give us then it is not a problem to use them. If we have to make a database for Marika's flower shop where she would put her data. For example, she wants to store that she has got 10 tulips and 30 roses and that she has sold 9 tulips and 34 roses, and nothing more. In this case the Access is more than enough for her. Don't try to convince her that she needs Oracle.

File – server architecture

The file server system brought a complete change in implementation of the computer architecture from the mainframe. In this system, the application logic was now executed on the client workstation instead of the server. These servers also provided access to computing resources like printers and large hard drives. The complete File Server Architecture is illustrated in the figure shown below.

The advantage of the file server system is the low cost entry point with flexible arrangement. Computer resources can be added or reduced as and when necessary using this system.

The drawback of the file server architecture is that all application logic is executed on the client machine. The job of the server is to provide files only to store the data. Though the application's file might be located on the server, the application runs in the client machine's memory space using the client's processor. This results in the client machine's need for large amount of power to run the application.



Taking into account the disadvantages of the centralized system and file server system architectures, the client-server architecture made its advent.

Client – server architecture

Keyword

Networking is a process that fosters the exchange of information and ideas among individuals or groups that share a common interest. It may be for social or business purposes. Professionals connect their business network through a series of symbolic ties and contacts.

Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources.

Client/server architecture is also known as a **networking** computing model or client/server network because all the requests and services are delivered over a network.

Client/server architecture is a producer/consumer computing architecture where the server acts as the producer and the client as a consumer. The server houses and provides high-end, computing-intensive services to the client on demand. These services can include application access, storage, file sharing, printer access and/or direct access to the server's raw computing power.

Client/server architecture works when the client computer sends a resource or process request to the server over the network connection, which is then processed and delivered to the client. A server computer can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services. In its simplest form, the internet is also based on client/server architecture where web servers serve many simultaneous users with website data.

Client-server architecture, architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them. Ideally, a server provides a standardized transparent interface to clients so that clients need not be aware of the specifics of the system (i.e., the hardware and software) that is providing the service. Clients are often situated at workstations or on personal computers, while servers are located elsewhere on the network, usually on more powerful machines. This computing model is especially effective when clients and the server each have distinct tasks

that they routinely perform. In hospital data processing, for example, a client computer can be running an application program for entering patient information while the server computer is running another program that manages the database in which the information is permanently stored. Many clients can access the server's information simultaneously, and, at the same time, a client computer can perform other tasks, such as sending e-mail. Because both client and server computers are considered intelligent devices, the client-server model is completely different from the old "mainframe" model, in which a centralized mainframe computer performed all the tasks for its associated "dumb" terminals.

Multi-Tier

Multitier architecture (often referred to as n-tier architecture) or multilayered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. The most widespread use of multitier architecture is the three-tier architecture.

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. A three-tier architecture is typically composed of a presentation tier, a domain logic tier, and a data storage tier.

While the concepts of layer and tier are often used interchangeably, one fairly common point of view is that there is indeed a difference. This view holds that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure. For example, a three-layer solution could easily be deployed on a single tier, such as a personal workstation.

Thin client

The thin client is a client minimal tool. This type of client uses the required sources of energy at remote (host) computers. The task of the thin client is mainly get exhausted in showing graphic data send by the application server.

A thin client is a stateless, fanless desktop terminal that has no hard drive. All features typically found on the desktop PC, including applications, sensitive data, memory, etc., are stored back in the data center when using a thin client.

A thin client running Remote Desktop Protocols (RDP), like Citrix ICA and Windows Terminal Services, and/or virtualization software, accesses hard drives in the data center stored on servers, blades, etc. Thin clients, software services, and backend hardware make up thin client computing, a virtual desktop computing model.

Thin clients are used as a PC replacement technology to help customers immediately access any virtual desktop or virtualized application. Thin clients provide businesses a cost-effective way to create a **virtual desktop infrastructure (VDI)**. Thin clients are utilized in various industries and enterprises worldwide that all have different requirements but share common goals. The cost, security, manageability, and scalability benefits of thin clients are all reasons that IT personnel are exploring –and switching– to thin clients.

Keyword

Virtual desktop infrastructure (VDI) is the practice of hosting a desktop operating system within a virtual machine (VM) running on a centralized server

Cost-wise, the price per seat of a thin client deployment has dropped to the point where it is more cost effective than regular PCs. This has been a claim that many in the thin client industry have made in the past, but the fact is that the technology that has been developed within the past year has made it a definitive reality.

Basic structures

Schema: every database has an inner structure that includes the description of all data elements and the connection among them. This structure is called the schema of the database.

The most significant metadata contains the definition of the data's type and references to what connections and relations are between data. Furthermore, they contain information in connection with the administration of the database. So, with their help can store structural information besides the actual data.

The construction of the database be different. It depends on the applied model. However, there are some general principles which are almost used in every application based on database. These are:

- The table, or data table is a two dimensional table which demonstrate logically closely connected data. The table consists of columns and rows.
- The record is a row of the database. We store in a record those data which are depending on each other. The rows of the table contain the concrete values of single features.

- The field is a column of the table. Every single column means the feature of the certain thing which has name and type.
- The elementary data are the values in cell of the table that are the concrete attributes of the entity.
- The entity is what we would like to describe and whose data we would like to store and collect in the database. We consider entity for example a person. We call those things or objects entity that can be well separated from and from which we store data, and what we feature with attributes. For example, entity can be the payment of a worker, a material, a person, etc. In this form the entity function as abstract notion. We can also say that the entity is the abstraction of concrete things. It is a habit to use the expression of entity type to abstract entities.
- The attribute is one of the features of the entity. The entity can be featured by the sum of attributes. For example, the name of a person can be a feature. The entity type is the sum of given features related to entity. For example, a person can be described jointly by name, date of birth, height, the color of hair and the color of eyes.
- The entity occurrence is the given concrete features of entity. For example, Koltai Lea Kiara is 5 years old. She has brown hair, blue eyes, her height is 110 cm, and she is in nursery schools. The occurrences of the entity are corresponding to the records. In practice, the entity type also can be called record type (record type or structure type).

When we store data in more than one place then we talk about data redundancy. Because, it is almost impossible to avoid the redundancy we have to endeavor to minimize the multiple occurrences. The method of that is to pick the repeated data out during the designing of the database, and store them separately referring to it in the right place.

Content management is an ongoing process and you should update your website frequently. You should gather a possible list of topics and make those into separate pages.

2.2 SQL STANDARD

SQL (Structured Query Language) is a standardized programming language used for managing relational databases and performing various operations on the data in them. Initially created in the 1970s, SQL is regularly used by database administrators, as well as by developers writing data integration scripts and data analysts looking to set up and run analytical queries.

The uses of SQL include modifying database table and index structures; adding, updating and deleting rows of data; and retrieving subsets of information from within a database for transaction processing and analytics applications. Queries and other SQL operations take the form of commands written as statements -- commonly used SQL statements include select, add, insert, update, delete, create, alter and truncate.

Keyword

Relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

SQL became the de facto standard programming language for relational databases after they emerged in the late 1970s and early 1980s. Also known as SQL databases, relational systems comprise a set of tables containing data in rows and columns. Each column in a table corresponds to a category of data -- for example, customer name or address -- while each row contains a data value for the intersecting column.

One of the most important developments in the market acceptance of SQL is the emergence of SQL standards. References to "the SQL standard" usually mean the official standard adopted by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). However, there are other important SQL standards, including the de facto standard for some parts of the SQL language that have been defined by IBM's DB2 product family, and Oracle's SQL dialect, which has a dominant installed-base market share.

Other SQL Standards

Although it is the most widely recognized, the ANSI/ISO standard is not the only standard for SQL. X/OPEN, a European vendor group, also adopted SQL as part of its suite of standards for a portable application environment based on UNIX. The X/OPEN standards have played a major role in the European computer market, where portability among computer systems from different vendors is a key concern. Unfortunately, the X/

OPEN standard differs from the ANSI/ISO standard in several areas. IBM also included SQL in the specification of its bold 1990s Systems Application Architecture (SAA) blueprint, promising that all of its SQL products would eventually move to this SAA SQL dialect. Although SAA failed to achieve its promise of unifying the IBM product line, the momentum toward a unified IBM SQL continued. With its mainframe DB2 database as the flagship, IBM introduced DB2 implementations for OS/2, its personal computer operating system, and for its RS/6000 line of UNIX-based workstations and servers. By 1997, IBM had moved DB2 beyond its own product line and shipped versions of DB2-Universal Database for systems made by rival manufacturers Sun Microsystems, Hewlett-Packard, and Silicon Graphics, and for Windows NT. IBM further shored up its database software position on non-IBM hardware platforms with its 2001 acquisition of the Informix database. With IBM's historical leadership in **relational database** technology, the SQL dialect supported by DB2 is a very powerful de facto standard.

2.2.1 The ANSI/ISO Standards

Work on the official SQL standard began in 1982, when ANSI charged its X31-12 committee with defining a standard relational database language. At first, the committee debated the merits of various proposed database languages. However, as IBM's commitment to SQL increased and SQL emerged as a de facto standard in the market, the committee selected SQL as their relational database language and turned their attention to standardizing it. The resulting ANSI standard for SQL was largely based on DB2 SQL, although it contains some major differences from DB2. After several revisions, the standard was officially adopted as ANSI standard X3.135 in 1986, and as an ISO standard in 1987. The ANSI/ISO standard has since been adopted as a Federal Information Processing Standard (FIPS) by the U.S. government. This standard, slightly revised and expanded in 1989, is usually called the SQL-89 or SQL1 standard. Many of the ANSI and ISO standards committee members were representatives from database vendors who had existing SQL products, each implementing a slightly different SQL dialect like dialects of human languages, the SQL dialects were generally very similar to one another but were incompatible in their details. In many areas, the committee simply sidestepped these differences by omitting some parts of the language from the standard and specifying others as "implementor-defined." These decisions allowed existing SQL implementations to claim broad adherence to the resulting ANSI/ ISO standard but made the standard relatively weak. To address the holes in the original standard, the ANSI committee continued its work, and drafts for a new, more rigorous SQL2 standard were circulated. Unlike the 1989 standard, the SQL2 drafts specified features considerably beyond those found in current commercial SQL products. Even more far-reaching changes were proposed for a follow-on SQL3 standard. In addition, the draft standards attempted to officially standardize parts of the SQL language where different proprietary standards had long since been set by the various major

DBMS brands. As a result, the proposed SQL2 and SQL3 standards were a good deal more controversial than the initial SQL standard. The SQL2 standard weaved its way through the ANSI approval process and was finally approved in October 1992. While the original 1986 standard took less than 100 pages, the SQL2 standard (officially called SQL-92) takes nearly 600 pages. The SQL2 standards committee acknowledged the large step from SQL] to SQL2 by explicitly creating three levels of SQL2 standards compliance. The lowest compliance level (Entry-Level) requires only minimal additional capability beyond the SQL-89 standard. The middle compliance level (Intermediate-Level) was created as an achievable major step beyond SQL-89, but one that avoids the most complex and most system-dependent and DBMS brand-dependent issues. The third compliance level (Full) requires a full implementation of all SQL2 capabilities. Throughout the 600 pages of the standard, each description of each feature includes a definition of the specific aspects of that feature that must be supported to achieve Entry, Intermediate, or Full compliance.

Despite the existence of a SQL2 standard for more than ten years, popular commercial SQL products do not, in practice, fully implement the SQL2 specification, and no two commercial SQL products support exactly the same SQL dialect. Moreover, as database vendors introduce new capabilities, they continually expand their SQL dialects and move them slightly further apart. The central core of the SQL language has become fairly standardized, however. Where it could be done without hurting existing customers or features, vendors have brought their products into conformance with the SQL-89 standard, and with the most useful capabilities of the SQL2 standard. In the meantime, work continues on standards beyond SQL2. The SQL3 effort effectively fragmented into separate standardization efforts and focused on different extensions to SQL. Some of these, such as stored procedure capabilities, are already found in many commercial SQL products and pose the same standardization challenges faced by SQL2. Others, such as proposed object extensions to SQL, are not yet widely available or fully implemented, but have generated a great deal of controversy. With most vendors only recently implementing major SQL2 capabilities, and with the diversity of SQL extensions now available in commercial products, work on SQL3 has taken on less commercial importance.

The “real” SQL standard, of course, is the SQL implemented in products that are broadly accepted by the marketplace. For the most part, programmers and users tend to stick with those parts of the language that are fairly similar across a broad range of products. The innovation of the database vendors continues to drive the invention of new SQL capabilities; some products remain years later only for backward compatibility, and some find commercial success and move into the mainstream.

2.2.2 SQL Standard and Proprietary Extensions

An official SQL standard was adopted by the American National Standards Institute (ANSI) in 1986 and then by the International Organization for Standardization, known as ISO, in 1987. More than a half-dozen joint updates to the standard have been released by the two standards development bodies since then; as of this writing, the most recent version is SQL:2011, approved that year.

Microsoft offers a set of extensions called Transact-SQL (T-SQL), while Oracle's extended version of the standard is PL/SQL. As a result, the different variants of SQL offered by vendors aren't fully compatible with one another.

Both proprietary and open source relational database management systems built around SQL are available for use by organizations. They include Microsoft SQL Server, Oracle Database, IBM DB2, SAP HANA, SAP Adaptive Server, MySQL (now owned by Oracle) and PostgreSQL. However, many of these database products support SQL with proprietary extensions to the standard language for procedural programming and other functions.

2.2.3 SQL Commands and Syntax

SQL commands are divided into several different types, among them data manipulation language (DML) and **data definition language (DDL)** statements, transaction controls and security measures. The DML vocabulary is used to retrieve and manipulate data, while DDL statements are for defining and modifying database structures. The transaction controls help manage transaction processing, ensuring that transactions are either completed or rolled back if errors or problems occur. The security statements are used to control database access as well as to create user roles and permissions.

SQL syntax is the coding format used in writing statements. Figure 1 shows an example of a DDL statement written in Microsoft's T-SQL to modify a database table in SQL Server 2016:

Keyword

Data Definition Language (DDL) is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

```
Use tempdb
GO

CREATE TABLE Sample (Numbers INT);
GO

INSERT INTO Sample(Numbers) VALUES (10)
GO 20

ALTER TABLE Sample
   ALTER COLUMN Numbers DECIMAL (4, 2) WITH (ONLINE = ON);
GO

SP_HELP Sample
GO

DROP TABLE Sample
GO
```

Figure 1. An example of T-SQL code in SQL Server 2016. This is the code for the ALTER TABLE WITH (ONLINE = ON | OFF) option.

2.2.4 SQL-on-Hadoop tools

SQL-on-Hadoop query engines are a newer offshoot of SQL that enable organizations with big data architectures built around Hadoop systems to take advantage of it instead of having to use more complex and less familiar languages -- in particular, the MapReduce programming environment for developing batch processing applications.

More than a dozen SQL-on-Hadoop tools have become available through Hadoop distribution providers and other vendors; many of them are open source software or commercial versions of such technologies. In addition, the Apache Spark processing engine, which is often used in conjunction with Hadoop, includes a Spark SQL module that similarly supports SQL-based programming.

In general, SQL-on-Hadoop is still an emerging technology, and most of the available tools don't support all of the functionality offered in relational implementations of SQL. But they're becoming a regular component of Hadoop deployments as companies look to get developers and data analysts with SQL skills involved in programming big data applications.

Within the big data landscape there are multiple approaches to accessing, analyzing, and manipulating data in Hadoop. Each depends on key considerations such as latency, ANSI SQL completeness (and the ability to tolerate machine-generated SQL), developer and analyst skillsets, and architecture tradeoffs. Below is a discussion segmented by broad latency characteristics of each approach.

Batch SQL: Technologies such as Hive are designed for batch queries on Hadoop by providing a declarative abstraction layer (HiveQL), which uses the MapReduce processing framework in the background. Hive is used primarily for queries on very large data sets and large ETL jobs. The queries can take anywhere between a few minutes to several hours depending on the complexity of the job. The Apache Tez project aims to provide targeted performance improvements for Hive to deliver interactive query capabilities. MapR ships and supports Apache Hive today and provides an early Developer Preview of Apache Tez.

Interactive SQL: Technologies such as Impala and Apache Drill provide interactive query capabilities to enable traditional business intelligence and analytics on Hadoop-scale datasets. The response times vary between milliseconds to minutes depending on the query complexity. Users expect SQL-on-Hadoop technologies to support common BI tools such as Tableau and MicroStrategy (to name a couple) for reporting and ad-hoc queries. MapR supports customers using both Apache Drill and Impala on the MapR Converged Data Platform.

In-Memory SQL: In-memory computing has enabled new ecosystem projects such as Apache Spark to further accelerate query processing. Spark SQL uses in-memory computing while retaining full Hive compatibility to provide 100x faster queries than Hive. MapR customers are using Spark with the MapR Converged Data Platform today.

Operational SQL: Unlike batch and interactive queries that are used by business teams for decision making and operate as read-only operations on large datasets (OLAP), point queries are typically done by OLTP and web applications, operating over smaller datasets and typically include insert, update, and deletes. The expected latency is usually very low (e.g., milliseconds) due to the high volume of requests from these applications. MapR ships and supports operational SQL capabilities by providing Apache HBase support on the MapR Converged Enterprise Edition.

Interactive SQL-on-Hadoop Technology Landscape SQL technologies complement traditional data warehouse and analytical environments for:

- Interactive and ad-hoc queries on large-scale data
- Data exploration to discover new insights worth modeling into a data warehouse schema

- Interactive queries on more or new types of data
- Queries for online data archives in Hadoop vs. backing up to tape

Technologies and approaches for interactive SQL vary and include (but are not limited to)

- Querying the data using connectors from Hadoop to analytic platforms (upfront or at query run time with external tables)
- Running traditional SQL engines side by side on every node of the Hadoop cluster
- Providing purpose-built SQL engines directly on top of Hadoop (native SQL options)
- Efforts to improve MapReduce performance to make it suitable for interactive queries

Options (1) and (2) excel at SQL support, performance optimizations, and overall enterprise readiness. However, native SQL-on-Hadoop options (3) are evolving as cost-effective alternatives because they have stronger compatibility with the Hadoop ecosystem (e.g., use Hadoop native file formats and common metadata store through Hive)

SQL technologies available on MapR

The table below describes at a high level some of the key considerations for picking the right SQL-on-Hadoop technology. Please contact us for specific questions about your use case.

		Drill	Hive	Impala	Spark SQL
Key Use Cases		Self-service Data Exploration Interactive BI / Ad-hoc queries	Batch/ ETL/ Long-running jobs	Interactive BI / Ad-hoc queries	SQL as part of Spark pipelines / Advanced analytic workflows
Data Sources	Files Support	Parquet, JSON, Text, all Hive file formats	Yes (all Hive file formats)	Yes (Parquet, Sequence, RC, Text, AVRO ...)	Parquet, JSON, Text, all Hive file formats
	HBase/ MapR-DB	Yes	Yes	Yes	Yes

	Beyond Hadoop	Yes	No	No	Yes
Data Types	Relational	Yes	Yes	Yes	Yes
	Complex/ Nested	Yes	Limited	No	Limited
Metadata	Schema-less/ Dynamic schema	Yes	No	No	Limited
	Hive Meta store	Yes	Yes	Yes	Yes
SQL / BI tools	SQL support	ANSI SQL	HiveQL	HiveQL	ANSI SQL (limited) & HiveQL
	Client support	ODBC/JDBC	ODBC/ JDBC	ODBC/JDBC	ODBC/JDBC
	Beyond Memory	Yes	Yes	Yes	Yes
	Optimizer	Limited	Limited	Limited	Limited
Platform	Latency	Low	Medium	Low	Low (in-memory) / Medium
	Concurrency	High	Medium	High	Medium
Decentralized Granular Security		Yes	No	No	No

2.2.5 ODBC and SQL

An important area of database technology not addressed by official standards is database interoperability—the methods by which data can be exchanged among different databases, usually over a network. In 1989, a group of vendors formed the SQL. Access Group to address this problem. The resulting SQL Access Group specification for Remote Database Access (RDA) was published in 1991. Unfortunately, the RDA specification was closely tied to the OSI protocols, which were never widely implemented, so it had little impact. Transparent interoperability among different vendors' databases remains an elusive goal. A second standard from the SQL Access Group had far more market impact. At Microsoft's urging and insistence, the SQL Access Group expanded

its focus to include a call-level interface for SQL. Based on a draft from Microsoft, the resulting Call-Level Interface (CLI) specification was published in 1992. Microsoft's own Open Database Connectivity (ODBC) specification, based on the CLI standard, was published the same year. With the market power of Microsoft behind it, and the "open standards" blessing of the SQL Access Group, ODBC has emerged as the de facto standard interface for PC access to SQL databases. Apple and Microsoft announced an agreement to support ODBC on Macintosh and Windows in the spring of 1993, giving ODBC industry standard status in both popular graphical user interface environments. ODBC implementations for UNIX-based systems soon followed. In 1995, the ODBC interface effectively became an ANSI/ISO standard, with the publication of the SQL/Call-Level Interface (CLI) standard. Today, ODBC is in its fourth major revision as a cross-platform database access standard. ODBC support is available for all major DBMS brands. Most packaged application programs that have database access as an important part of their capabilities support ODBC, and they range from multimillion-dollar enterprise-class applications like **Enterprise Resource Planning (ERP)** and Supply Chain Management (SCM) to PC applications such as spreadsheets, query tools, and reporting programs. Microsoft's focus has moved beyond ODBC to higher-level interfaces (such as OLE/DB) and more recently to Active/X Data Objects (ADO), but these new interfaces are layered on top of ODBC for relational database access, and it remains a key cross-platform database access technology.

Keyword

Enterprise resource planning (ERP) is business process management software that allows an organization to use a system of integrated applications to manage the business and automate many back office functions related to technology, services and human resources.

Role of Open Database Connectivity

Open Database Connectivity (ODBC) is an interface between computer applications and databases. This interface provides a buffer layer in between the database and the software used to access it. This means that any software may connect to any database regardless of platform or method as long as both systems use ODBC. Essentially, the two programs speak in their own languages and the Open Database Connectivity routines translate the information.

The original Open Database Connectivity system was developed by Microsoft® in 1992. This system operated very well in some circumstances, but not in others. In 1995,

Microsoft® released Version 3 of the system, which coincided with it being adopted as a base standard for structured query language (SQL). As part of the SQL standard, the interface became widely used for all sorts of different database purposes.

Before the adoption of Open Database Connectivity, database-using programs needed coded methods for talking to different styles of databases. Programmers believed these systems would need access to three different types of databases; then, three different commands for each function were programmed into the system. Databases had the same redundancy; each program required the information be sent out in a specific manor. If either of these programs were off in syntax or encountered an unfamiliar system, no communication was possible.

This all changed with the development and implementation of Open Database Connectivity. This essentially works as a translator. The programmers of the database and applications write up the methods they use in the syntax used by ODBC. When the application requires information, it sends the query, and ODBC translates its syntax to the methods used by the database. The database sends the answer back, and ODBC translates it back into the syntax required by the application.

This process works via an installed set of drivers. Each database has a specific Open Database Connectivity driver associated with it. This driver does the actual translation between the database and the outside world. If changes to the specification make the driver obsolete, then only the driver needs changing; the rest of the database may remain unaltered. This allows updates to the interface without a lot of additional coding work.

Applications essentially have Open Database Connectivity drivers built into them. These drivers are part of the programming for the application. They may be updated as the program receives patches, but they are separate from the standard ODBC system.

The real communication happens between these driver sets. The program's drivers translate the information before it is actually sent. The database's drivers receive the request in its own language, get the information and send it out, still in the ODBC syntax. The program's drivers take the information back in and translate it back to the program's language. This two-layer system creates an interface that is nearly foolproof, as the actual designers of the systems make their own interfaces.

ODBC Driver

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data.

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data. ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users

can then add ODBC database drivers to link the application to their choice of DBMS.

The ODBC driver interface defines:

- A library of ODBC function calls of two types:
  - Core functions that are based on the X/Open and SQL Access Group
  - Call Level Interface specification
  - Extended functions that support additional functionality, including scrollable cursors
- SQL syntax based on the X/Open and SQL Access Group SQL CAE specification (1992)
- A standard set of error codes
- A standard way to connect and logon to a DBMS
- A standard representation for data types

The ODBC solution for accessing data led to ODBC database drivers, which are dynamic-link libraries on Windows and shared objects on Linux/UNIX. These drivers allow an application to gain access to one or more data sources.

Remember

ODBC provides a standard interface to allow application developers and vendors of database drivers to exchange data between applications and data sources.

Establish an ODBC Connection to the SQL Database

Windows 7 includes an ODBC manager that lets you connect from your desktop to a SQL server. You create a data source name (DSN) to save database setup information, so you can open a connection to the SQL Server database without re-entering the server information. The ODBC manager in the Control Panel stores the connection for future use.

- Click the Windows "Start" button and select "Control Panel." Click "System and Security," then click "Administrative Tools." Double-click the icon labeled "Data Sources (ODBC)." A list of current database connections displays.
- Click the "Add" button to create a new connection and start the connection wizard. Click the "SQL Native Client" in the list of drivers. The SQL client is included with the Windows 7 operating system. Click "Finish."
- Type the SQL server information in the window that opens. You must type the SQL server name and a display name for the saved connection. Click "Next."

- Type your username and password to access the SQL server. Click the option labeled "With SQL Server authentication." This option uses the separate SQL server username and password instead of logging you in with the Windows 7 account information. Click "Next."
- Type the default database name and click "Next." In the final window, leave the default settings and click "Finish." Return to the list of database connections, and the new SQL server connection displays

2.2.6 SQL and Portability

The existence of published SQL standards has spawned quite a few exaggerated claims about SQL and applications portability. Diagrams such as the one in Figure 2 are frequently drawn to show how an application using SQL can work interchangeably with any SQL-based database management system. In fact, the holes in the SQL-89 standard and the current differences between SQL dialects are significant enough that an application must always be modified when moved from one SQL database to another.

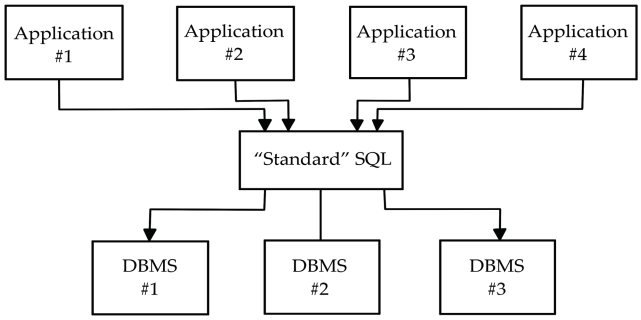


Figure 2. The SQL Portability myth.

These differences, many of which were eliminated by the SQL2 standard but have not yet been implemented in commercial products, include:

- Error codes. The SQL-89 standard does not specify the error codes to be returned when SQL detects an error, and all of the commercial implementations use their own set of error codes. The SQL2 standard specifies standard error codes.
- Data types. The SQL-89 standard defines a minimal set of data types, but it omits some of the most popular and useful types, such as variable-length character strings, dates and times, and money data. The SQL2 standard addresses these, but not "new" data types such as graphics and multimedia objects.

- System tables. The SQL-89 standard is silent about the system tables that provide information regarding the structure of the database itself. Each vendor has its own structure for these tables, and even IBM's four SQL implementations differ from one another. The tables are standardized in SQL2, but only at the higher levels of compliance, which are not yet provided by most vendors.
- Interactive SQL. The standard specifies only the programmatic SQL used by an application program, not interactive SQL. For example, the SELECT statement used to query the database in interactive SQL is absent from the SQL-89 standard. Again, the SQL2 standard addressed this issue, but long after all of the major DBMS vendors had well-established interactive SQL capabilities.
- Programmatic interface. The original standard specifies an abstract technique for using SQL from within an applications program written in COBOL, C, FORTRAN, and other programming languages. No commercial SQL product uses this technique, and there is considerable variation in the actual programmatic interfaces used. The SQL2 standard specifies an embedded SQL interface for popular programming languages but not a call-level interface. The 1995 SQL/CLI standard finally addressed programmatic SQL access, but not before commercial DBMS products had popularized proprietary interfaces and deeply embedded them in hundreds of thousands of user applications and application packages.
- Dynamic SQL. The SQL-89 standard does not include the features required to develop general-purpose database front-ends, such as query tools and report writers. These features, known as dynamic SQL, are found in virtually all SQL database systems, but they vary significantly from product to product. SQL2 includes a standard for dynamic SQL. But with hundreds of thousands of existing applications dependent on backward compatibility, DBMS vendors have not implemented it.

Did You Know?

ODBC 1.0 was released in September 1992. At the time, there was little direct support for SQL databases (versus ISAM), and early drivers were noted for poor performance. Some of this was unavoidable due to the path that the calls took through the Jet-based stack; ODBC calls to SQL databases were first converted from Simba Technologies's SQL dialect to Jet's internal C-based format, then passed to a driver for conversion back into SQL calls for the database.

- Semantic differences. Because the standards specify certain details as implementer-defined, it's possible to run the same query against two different conforming SQL implementations and produce two different sets of query results. These differences occur in the handling of NULL values, column functions, and duplicate row elimination.
- Collating sequences. The SQL-89 standard does not address the collating (sorting) sequence of characters stored in the database. The results of a sorted query will be different if the query is run on a personal computer (with ASCII characters) and a mainframe (with EBCDIC characters). The SQL2 standard includes an elaborate specification for how a program or a user can request a specific collating sequence, but it is an advanced-level feature that is not typically supported in commercial products.
- Database structure. The SQL-89 standard specifies the SQL language to be used once a particular database has been opened and is ready for processing. The details of database naming and how the initial connection to the database is established vary widely and are not portable. The SQL2 standard creates more uniformity but cannot completely mask these details.

Despite these differences, commercial database tools boasting portability across several different brands of SQL databases began to emerge in the early 1990s. In every case, however, the tools require a special adapter for each supported DBMS, which generates the appropriate SQL dialect, handles data type conversion, translates error codes, and so on. Transparent portability across different DBMS brands based on standard SQL is the major goal of SQL2 and ODBC, and significant progress has been made. Today, virtually all programs that support multiple databases include specific drivers for communicating with each of the major DBMS brands, and usually include an ODBC driver for accessing the others.

SUMMARY

- A perspective is a definition that allows users to see a cube in a simpler way. A perspective is a subset of the features of a cube. A perspective enables administrators to create views of a cube, helping users to focus on the most relevant data for them. A perspective contains subsets of all objects from a cube. A perspective cannot include elements that are not defined in the parent cube.
- A simple Perspective object is composed of: basic information, dimensions, measure groups, calculations, KPIs, and actions. Basic information includes the name and the default measure of the perspective. The dimensions are a subset of the cube dimensions.
- SQL is a standardized query language for requesting information from a database. The original version called SEQUEL (structured English query language) was designed by an IBM research center in 1974 and 1975. SQL was first introduced as a commercial database system in 1979 by Oracle Corporation.
- Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy.
- The database is a kind of data collection. It stores data, which is in connection with the given task, orderly. The access to the data is also taken care of by the database. Besides, it guarantees the protection of the data, and also protects the integration of the data. The management of the data was also made easier by database management systems.
- Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection.
- Multitier architecture (often referred to as n-tier architecture) or multilayered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. The most widespread use of multitier architecture is the three-tier architecture.
- The thin client is a client minimal tool. This type of client uses the required sources of energy at remote (host) computers. The task of the thin client is mainly get exhausted in showing graphic data send by the application server.
- SQL (Structured Query Language) is a standardized programming language used for managing relational databases and performing various operations on the data in them. Initially created in the 1970s, SQL is regularly used by database administrators, as well as by developers writing data integration scripts and data analysts looking to set up and run analytical queries.
- SQL commands are divided into several different types, among them data manipulation language (DML) and data definition language (DDL) statements, transaction controls and security measures. The DML vocabulary is used to retrieve and manipulate data, while DDL statements are for defining and modifying database structures.

- SQL-on-Hadoop query engines are a newer offshoot of SQL that enable organizations with big data architectures built around Hadoop systems to take advantage of it instead of having to use more complex and less familiar languages -- in particular, the MapReduce programming environment for developing batch processing applications.

KNOWLEDGE CHECK

1. **Which one of the following is used to define the structure of the relation ,deleting relations and relating schemas ?**
   a. DML(Data Manipulation Language)
   b. DDL(Data Definition Language)
   c. Query
   d. Relational Schema
2. **To remove a relation from an SQL database, we use the _____ command.**
   a. Delete
   b. Purge
   c. Remove
   d. Drop table
3. **In the SQL given there is an error. Identify the error.**
   a. Dept_name
   b. Employee
   c. "Comp Sci"
   d. From
4. **Which of the following is used to store movie and image files ?**
   a. Clob
   b. Blob
   c. Binary
   d. Image
5. **The user defined data type can be created using**
   a. Create data type
   b. Create data
   c. Create define type
   d. Create type
6. **_____ command makes the updates performed by the transaction permanent in the database?**
   a. ROLLBACK
   b. COMMIT
   c. TRUNCATE
   d. DELETE

7. Which of the following options are correct regarding these three keys (Primary Key, Super Key, and Candidate Key) in a database?

I. Minimal super key is a candidate key

II. Only one candidate key can be a primary key

III. All super keys can be a candidate key

IV. We cannot find a primary key from the candidate key

a. I and II

b. II and III

c. I and III

d. II and IV

8. _____ is a program that performs some common action on database data and also stored in the database.

a. Stored Procedure

b. Trigger

c. Stored Function

d. None of the above

9. Which of the following are the DATETIME data types that can be used in column definitions?

a. TIMESTAMP

b. INTERVAL MONTH TO DAY

c. INTERVAL YEAR TO MONTH

d. TIMESTAMP WITH DATABASE TIMEZONE

e. Both a and c

10. Using which language can a user request information from a database?

a. Query

b. Relational

c. Structural

d. Compiler

REVIEW QUESTIONS

1. What do you understand by the database management system?
2. Discuss on ANSI/ISO standards.
3. Define the features of SQL standard and proprietary extensions.
4. Write the SQL commands and syntax.
5. What is the relationship between ODBC and SQL?

Check Your Result

1. (b)	2. (d)	3. (c)	4. (b)	5. (d)
6. (b)	7. (a)	8. (a)	9. (e)	10. (a)

REFERENCES

1. Abraham Silberschatz, Henry F. Korth and S. Sudarshan, Database System Concepts, McGraw-Hill Education (Asia), Fifth Edition, 2006.
2. C. J. Date, A. Kannan and S. Swamynathan, An Introduction to Database Systems, Pearson Education, Eighth Edition, 2009.
3. http://aries.ektf.hu/~dream/e107/e107_files/downloads/dbms.pdf
4. http://www.devonit.com/thin-client-education
5. http://www.kciti.edu/wp-content/uploads/2017/07/dbms_tutorial.pdf
6. https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models-olap-logical-cube-objects/perspectives?view=sql-server-2017
7. https://mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details/
8. https://nptel.ac.in/courses/106106095/pdf/4_The_SQL_Standard.pdf
9. https://searchnetworking.techtarget.com/definition/thin-client
10. https://searchsqlserver.techtarget.com/definition/SQL
11. https://smallbusiness.chron.com/establish-odbc-connection-sql-database-28127.html
12. https://www.britannica.com/technology/client-server-architecture
13. https://www.freetutes.com/learn-vb6/lesson22-2.2.html
14. https://www.guru99.com/introduction-to-database-sql.html
15. https://www.progress.com/faqs/datadirect-odbc-faqs/what-is-an-odbc-driver
16. https://www.techopedia.com/definition/438/clientserver-architecture

CHAPTER

3

RETRIEVING DATA

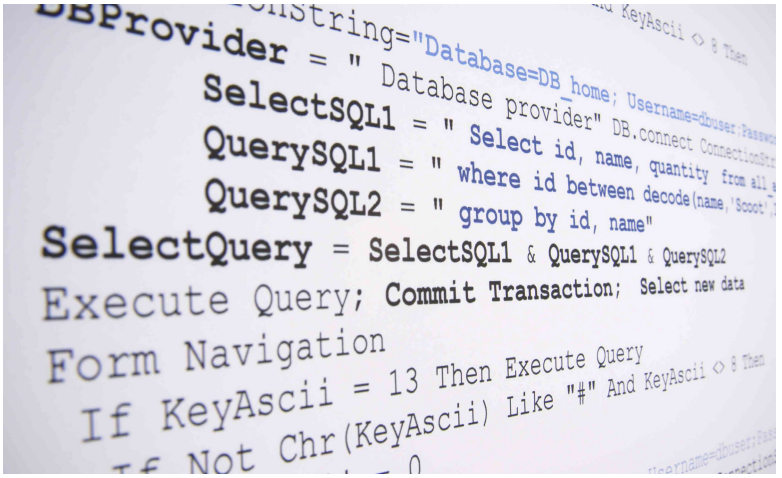
"The causes and severity of NSA infractions vary widely. One in 10 incidents is attributed to a typographical error in which an analyst enters an incorrect query and retrieves data about U.S phone calls or emails."

—Barton Gellman

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss about SQL data types
2. Understand the SQL expressions
3. Learn about SQL queries



INTRODUCTION

Data retrieval means obtaining data from a database management system such as ODBMS. In this case, it is considered that data is represented in a structured way, and there is no ambiguity in data.

In order to retrieve the desired data the user present a set of criteria by a query. Then the Database Management System (DBMS), software for managing databases, selects the demanded data from the database. The retrieved data may be stored in a file, printed, or viewed on the screen.

A query language, such as Structured Query Language (SQL), is used to prepare the queries. SQL is an American National Standards Institute (ANSI) standardized query language developed specifically to write database queries. Each DBMS may have its own language, but most relational.

Reports and queries are the two primary forms of the retrieved data from a database. There are some overlaps between them, but queries generally select a relatively small portion of the database, while reports show larger amounts of data. Queries also present the data in a standard format and usually display it on the monitor; whereas reports allow formatting of the output however you like and is normally printed.

3.1 SQL DATA TYPES

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

Remember

Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

A data type defines what kind of value a column can hold: integer data, character data, monetary data, date and time data, binary strings, and so on.

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data

3.1.1 MySQL Data Types

In MySQL there are three main data types: text, number, and date.

Text Data Types

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number Data Types

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the **integer** goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date Data Types

Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMISS, YYMMDDHHMISS, YYYYMMDD, or YYMMDD.

Keyword

Integer is a datum of integral data type, a data type that represents some range of mathematical integers.

3.1.2 SQL Server Data Types

String Data Types

Data type	Description	Max size	**Storage**
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Number Data Types

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes

money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Date Data Types

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Other Data Types

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a **cursor** used for database operations
table	Stores a result-set for later processing

3.1.3 Microsoft Access Data Types

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:**You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes

AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

Remember

Datetime has 3.33 milliseconds accuracy whereas smalldatetime has 1 minute accuracy.

3.2 SQL EXPRESSIONS

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the **database** for a specific set of data.

Syntax

Consider the basic syntax of the SELECT statement as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below –

- Boolean
- Numeric
- Date

3.2.1 Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax –

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the CUSTOMERS table having the following records –

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
7 rows in set (0.00 sec)
```

The following table is a simple example showing the usage of various SQL Boolean Expressions –

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|  7 | Muffy |  24 | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
1 row in set (0.00 sec)
```

Keyword

Database is an organized collection of data, stored and accessed electronically.

Keyword

Cursor is a control structure that enables traversal over the records in a database.

3.2.2 Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax –

SELECT numerical_expression as OPERATION_NAME
[FROM table_name
WHERE CONDITION];

Here, the numerical_expression is used for a **mathematical expression** or any formula. Following is a simple example showing the usage of SQL Numeric Expressions –

Keyword

Mathematical expression is a finite combination of symbols that is well-formed according to rules that depend on the context.

```
SQL> SELECT (15 + 6) AS ADDITION
+----------+
| ADDITION |
+----------+
|       21 |
+----------+
1 row in set (0.00 sec)
```

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+---------+
| RECORDS |
+---------+
|       7 |
+---------+
1 row in set (0.00 sec)
```

3.2.3 Date Expressions

Date Expressions return current system date and time values –

```
SQL>  SELECT CURRENT_TIMESTAMP;
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2009-11-12 06:40:23 |
+---------------------+
1 row in set (0.00 sec)
```

Another date expression is as shown below –

```
SQL>  SELECT  GETDATE();;
+------------------------+
| GETDATE                |
+------------------------+
| 2009-10-22 12:07:18.140 |
+------------------------+
1 row in set (0.00 sec)
```

3.3 SQL QUERIES

The SQL queries are the most common and essential SQL operations. Via an SQL query, one can search the database for the information needed. SQL queries are executed with the "SELECT" statement. An SQL query can be more specific, with the help of several clauses:

- FROM - it indicates the table where the search will be made.
- WHERE - it's used to define the rows, in which the search will be carried. All rows, for which the WHERE clause is not true, will be excluded.
- ORDER BY - this is the only way to sort the results in SQL. Otherwise, they will be returned in a random order.

3.3.1 SQL INSERT Query

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

Did You Know?

In 1854, George Boole published his work "An Investigation into the Laws of thought," where he described a system for symbolic and logical reasoning. This system was later called "Boolean logic". Presently, Boolean logic is the basis for computer and program design

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows –

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
```

```
| 6 | Komal    | 22 | MP        | 4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Populate One Table Using another Table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax –

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

Keyword

Result set is a set of rows from a database, as well as metadata about the query such as the column names, and the types and sizes of each column.

3.3.2 SQL SELECT Query

Select is the most commonly used statement in SQL. The SELECT Statement in SQL is used to retrieve or fetch data from a database. We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called **result-set**.

With the SELECT clause of a SELECT command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table.

The select clause is the first clause and is one of the last clauses of the select statement that the database server evaluates. The reason for this is that before we can determine what to include in the final result set, we need to know all of the possible columns that could be included in the final result set.

Sample Table

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18

Remember

The semicolon character is required to indicate the end of a SQL statement. Alternatively, you can use the go command or \g to tell mysql to execute a query.

Syntax

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT * FROM table_name;

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

SQL> SELECT * FROM CUSTOMERS;

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

3.3.3 SQL UPDATE Query

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows –

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

```
| 5 | Hardik   | 27 | Bhopal   | 8500.00 |
| 6 | Komal    | 22 | Pune     | 4500.00 |
| 7 | Muffy    | 24 | Indore   | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following **code block**.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records –

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
| 1 | Ramesh   | 32 | Pune    | 1000.00 |
| 2 | Khilan   | 25 | Pune    | 1000.00 |
| 3 | kaushik  | 23 | Pune    | 1000.00 |
| 4 | Chaitali | 25 | Pune    | 1000.00 |
| 5 | Hardik   | 27 | Pune    | 1000.00 |
| 6 | Komal    | 22 | Pune    | 1000.00 |
| 7 | Muffy    | 24 | Pune    | 1000.00 |
+----+----------+-----+---------+---------+
```

Keyword

Code block is a lexical structure of source code which is grouped together.

3.3.4 SQL DELETE Query

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows –

```
DELETE FROM table_name
```

WHERE [condition];

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows –

```
SQL> DELETE FROM CUSTOMERS;
```

Now, the CUSTOMERS table would not have any record.

ROLE MODEL

RAYMOND F. BOYCE

Raymond F. Boyce (1947–1974) was an American computer scientist who was known for his research in relational databases. He is best known for his work co-developing the SQL database language and Boyce-Codd normal form.

Biography

Boyce grew up in New York, and went to college at Providence College, from which he graduated in 1968. He earned his PhD in computer science at Purdue in 1972. His wife Sandy, whom he met in college, was a nurse. After leaving Purdue he worked on database projects for IBM in Yorktown Heights, New York. In the short period that he had, which was not quite two years long, he co-developed Boyce–Codd normal form. Together with Donald D. Chamberlin, he co-developed Structured Query Language (SQL) while managing the Relation Database development group for IBM in San Jose, California. He died in 1974 as a result of an aneurysm, leaving behind his wife Sanndy and his infant daughter Kristin.

SQL

SQL was initially co-developed at IBM by Boyce alongside Donald D. Chamberlin in the early 1970s. Initially called SEQUEL (Structured English Query Language) and based on their original language called SQUARE (Specifying Queries As Relational Expressions). SEQUEL was designed to manipulate and retrieve data in relational databases. By 1974, Chamberlin and Boyce published "SEQUEL: A Structured English Query Language" which detailed their refinements to SQUARE and introduced us to the data retrieval aspects of SEQUEL.[1] It was one of the first languages to use Edgar F. Codd's relational model. SEQUEL was later renamed to SQL by dropping the vowels, because SEQUEL was a trade mark registered by the Hawker Siddeley aircraft company.[1] Today, SQL has become the most widely used relational database language.

Boyce-Codd Normal Form

Boyce–Codd normal form (or BCNF) was developed in 1974 by Boyce and Edgar F. Codd. It is a type of normal form that is used in database normalization. The goal of relational database design is to generate a set of database schemas that store information without unnecessary redundancy. Boyce-Codd accomplishes this and allows users to retrieve information easily. Using BCNF, databases will have all redundancy removed based on functional dependencies. It is a slightly stronger version of the third normal form.

SUMMARY

- Data retrieval means obtaining data from a database management system such as ODBMS. In this case, it is considered that data is represented in a structured way, and there is no ambiguity in data.
- A query language, such as Structured Query Language (SQL), is used to prepare the queries. SQL is an American National Standards Institute (ANSI) standardized query language developed specifically to write database queries. Each DBMS may have its own language, but most relational.
- Reports and queries are the two primary forms of the retrieved data from a database. There are some overlaps between them, but queries generally select a relatively small portion of the database, while reports show larger amounts of data.
- SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL.
- An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language.
- The SQL queries are the most common and essential SQL operations. Via an SQL query, one can search the database for the information needed. SQL queries are executed with the "SELECT" statement.
- The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.
- Select is the most commonly used statement in SQL. The SELECT Statement in SQL is used to retrieve or fetch data from a database. We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called result-set.
- The SQL UPDATE Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

KNOWLEDGE CHECK

1. Consider the following schema –

STUDENTS(student_code, first_name, last_name, email,

phone_no, date_of_birth, honours_subject, percentage_of_marks);

Which of the following query would display all the students where the second letter in the first name is 'i'?

a. select first_name from students where first_name like '_i%';

b. select first_name from students where first_name like '%i_';

c. select first_name from students where first_name like '%i%';

d. select first_name from students where first_name like '_i_';

2. Consider the following schema –

STUDENTS(student_code, first_name, last_name, email,

phone_no, date_of_birth, honours_subject, percentage_of_marks);

Which of the following query would correctly display the students' first name, last name, honours subject and date of birth, born between July 1st 1996, and 30th June 1999.

a. select first_name, last name, honours_subject, date_of_birth from students where date_of_birth between '30-JUN-1999' and '01-JUL-1996';

b. select first_name, last name, honours_subject, date_of_birth from students where date_of_birth in ('30-JUN-1999' , '01-JUL-1996');

c. select first_name, last name, honours_subject, date_of_birth from students where date_of_birth like '30-JUN-1999' and '01-JUL-1996';

d. select first_name, last name, honours_subject, date_of_birth from students where date_of_birth between '01-JUL-1996' and '30-JUN-1999';

3. Consider the following schema –

STUDENTS(student_code, first_name, last_name, email,

phone_no, date_of_birth, honours_subject, percentage_of_marks);

Which query will display the names and honours subjects of all students and if a student has not yet been given a honours subject yet, then it should display 'No Honours Yet'.

a. select first_name, last name, nvl(honours_subject, 'No Honours Yet') from students;

b. select first_name, last name, nvl2(honours_subject, 'No Honours Yet') from students;

c. select first_name, last name, honours_subject, from students;

d. select first_name, last name, nullif(honours_subject, 'No Honours Yet') from students;

4. Consider the following schema –

HONOURS_SUBJECT(subject_code, subject_name, department_head);

LOCATIONS(subject_code, department_name, location_id, city);

Select the right query for retrieving records from the tables HONOURS_ SUBJECT and LOCATIONS with a left outer join

a. select h.subject_name, l.department_name, h.department_head, l.city from honours_subject h left outer join location l on(h.subject_code = l.subject_code);

b. select h.subject_name, l.department_name, h.department_head, l.city from honours_subject h left outer join location l on(subject_code);

c. select h.subject_name, l.department_name, h.department_head, l.city from honours_subject h left join location l on(h.subject_code = l.subject_code);

d. None of the above.

5. Consider the following schema –

STUDENTS(student_code, first_name, last_name, email,

phone_no, date_of_birth, honours_subject, percentage_of_marks);

Which of the following query will correctly list the average percentage of marks in each honours subject, when the average is more than 50 percent?

a. select honours_subject, avg(percentage_of_marks) from students where avg(percentage_of_marks) > 50.0 group by honours_subject;

b. select honours_subject, avg(percentage_of_marks) from students having avg(percentage_of_marks) > 50.0 group by honours_subject;

c. select honours_subject, avg(percentage_of_marks) from students group by honours_subject having avg(percentage_of_marks) > 50.0;

d. None of the above.

6. Creating a table in Base using _____ gives more flexibility and control.

a. SQL

b. Design View

c. Wizard

d. Functions

7. _____ is a standard language used to query a relational database.

a. SQL

b. Design View

c. Wizard
d. Functions

8. The SQL queries are in the form of _____

a. Commands
b. Statements
c. List
d. Functions

9. The SQL option can be found under _____ menu.

a. File
b. Edit
c. Tools
d. Help

10. The dialog box for writing SQL statements is called _____.

a. Text for SQL Commands dialog box
b. Execute SQL Statement dialog box
c. Statements for SQL dialog box
d. Command to Execute dialog box

REVIEW QUESTIONS

1. Discuss about MySQL data types.
2. Describe the Microsoft access data types.
3. What are the different types of SQL expressions?
4. What do you understand by SQL query?
5. Discuss about the SQL select query.

Check Your Result

1. (a)	2. (d)	3. (a)	4. (a)	5. (b)
6. (a)	7. (a)	8. (b)	9. (c)	10. (b)

REFERENCES

1. Abraham Silberschatz; Henry Korth; S. Sudarshan (2010). Database System Concepts (6th ed.). McGraw-Hill. pp. 187–192. ISBN 978-0-07-352332-3.
2. Beaulieu, Alan (April 2009). Mary E Treseler, ed. Learning SQL (2nd ed.). Sebastapol, CA, USA: O'Reilly. ISBN 978-0-596-52083-0.
3. C. J. Date (2011). SQL and Relational Theory: How to Write Accurate SQL Code (2nd ed.). O'Reilly Media. pp. 159–163. ISBN 978-1-4493-1640-2.
4. Capron, H. L.; J. A. Johnson (2004). Computers: Tools for an Information Age (8 ed.). Pearson/Prentice Hall. ISBN 0-13-122723-8.
5. E.g. for Java see Brogden, William B.; Green, Marcus (2003), Java 2 Programmer, Que Publishing, p. 45, ISBN 9780789728616.
6. Hector Garcia-Molina; Jeffrey D. Ullman; Jennifer Widom (2009). Database systems: the complete book (2nd ed.). Pearson Prentice Hall. pp. 437–445. ISBN 978-0-13-187325-4.
7. http://www.informit.com/articles/article.aspx?p=482319
8. https://www.geeksforgeeks.org/sql-select-query/
9. https://www.tutorialspoint.com/sql/sql-data-types.htm
10. https://www.tutorialspoint.com/sql/sql-insert-query.htm
11. Padron-McCarthy, Thomas; Tore Risch (2005). Databasteknik. Studentlitteratur. ISBN 91-44-04449-6.
12. Raghu Ramakrishnan; Johannes Gehrke (2003). Database management systems (3rd ed.). McGraw-Hill. ISBN 978-0-07-246563-1. Chapter 24.
13. van Melkebeek, Dieter (2000), Randomness and Completeness in Computational Complexity, Lecture Notes in Computer Science, 1950, Springer, p. 22, ISBN 9783540414926.

CHAPTER

4

UPDATING DATA

"Database means a tables collected different information, so one site is a result of a collected tables"

—Deyth Banger

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss the adding data to the database
2. Explain the deleting data from the database
3. Understand the modifying data in the database

INTRODUCTION

SQL is a complete data manipulation language that is used not only for database queries, but also to modify and update data in the database. Compared to the complexity of the SELECT statement, which supports SQL queries,

the SQL statements that modify database contents are extremely simple. However, database updates pose some challenges for a DBMS beyond those presented by database queries. The DBMS must protect the integrity of stored data during changes, ensuring that only valid data is introduced into the database, and that the database remains self-consistent, even in the event of system failures. The DBMS must also coordinate simultaneous updates by multiple users, ensuring that the users and their changes do not interfere with one another.

4.1 ADDING DATA TO THE DATABASE

A new row of data is typically added to a relational database when a new entity represented by the row appears in the outside world. For example, in the sample database:

- When you hire a new salesperson, a new row must be added to the SALESREPS table to store the salesperson's data.
- When a salesperson signs a new customer, a new row must be added to the CUSTOMERS table, representing the new customer.
- When a customer places an order, a new row must be added to the ORDERS table to contain the order data.

In each case, the new row is added to maintain the database as an accurate model of the real world. The smallest unit of data that can be added to a **relational database** is a single row. In general, a SQL-based DBMS provides three ways to add new rows of data to a database:

- Single-row INSERT. A single-row INSERT statement adds a single new row of data to a table. It is commonly used in daily applications—for example, data entry programs.
- Multi-row INSERT. A multirow INSERT statement extracts rows of data from another part of the database and adds them to a table. It is commonly used in end-of-month or end-of-year processing when old rows of a table are moved to an inactive table.
- Bulk load. A bulk load utility adds data to a table from a file that is outside of the database. It is commonly used to initially load the database or to incorporate data downloaded from another computer system or collected from many sites.

4.1.1 The Single-Row INSERT Statement

The single-row INSERT statement, shown in Figure 1, adds a new row to a table. The INTO clause specifies the table that receives the new row (the target table), and the VALUES clause specifies the data values that the new row will contain. The column list indicates which data value goes into which column of the new row.

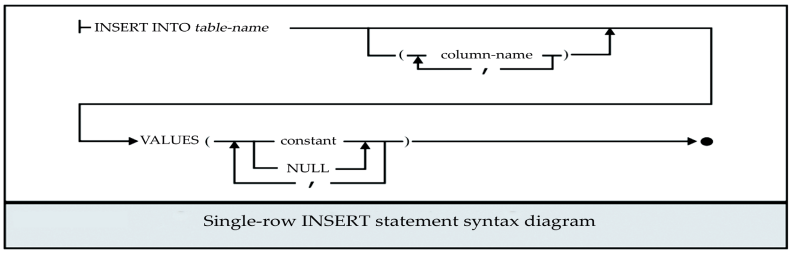


Figure 1: Single-row INSERT statement syntax diagram.

Suppose you just hired a new salesperson, Henry Jacobsen, with the following personal data:

Name:	Henry Jacobsen
Age:	36
Employee Number:	111
Title:	Sales Manager
Office:	Atlanta (office number 13)
Hire Date:	July 25, 1990
Quota:	Not yet assigned
Year-to-Date Sales:	$0.00

Here is the INSERT statement that adds Mr. Jacobsen to the sample database:

Add Henry Jacobsen as a new salesperson.

```
INSERT INTO SALESREPS (NAME, AGE, EMPL_NUM, SALES, TITLE,
                       HIRE_DATE, REP_OFFICE)
     VALUES ('Henry Jacobsen', 36, 111, 0.00, 'Sales Mgr',
                       '25-JUL-90', 13)

1 row inserted.
```

Keyword

A **relational database** is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

Figure 2 graphically illustrates how SQL carries out this INSERT statement. Conceptually, the INSERT statement builds a single row of data that matches the column structure of the table, fills it with the data from the VALUES clause, and then adds the new row to the table. The rows of a table are unordered, so there is no notion of inserting the row at the top, at the bottom, or between two rows of the table. After the INSERT statement, the new row is simply a part of the table. A subsequent query against the SALESREPS table will include the new row, but it may appear anywhere among the rows of query results.

Suppose Mr. Jacobsen now receives his first order, from InterCorp, a new customer who is assigned customer number 2126. The order is for 20 ACI-41004 widgets, for a total price of $2340, and has been assigned order number 113069. Here are the INSERT statements that add the new customer and the order to the database:

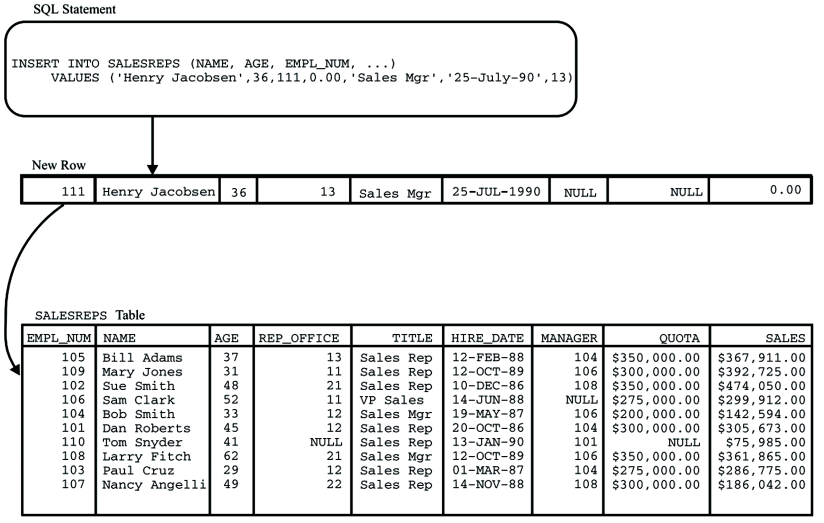


Figure 2: Inserting a single row.

Insert a new customer and order for Mr. Jacobsen.

```
INSERT INTO CUSTOMERS (COMPANY, CUST_NUM, CREDIT_LIMIT, CUST_REP)
     VALUES ('InterCorp', 2126, 15000.00, 111)

1 row inserted.

INSERT INTO ORDERS (AMOUNT, MFR, PRODUCT, QTY, ORDER_DATE,
                    ORDER_NUM, CUST, REP)
     VALUES (2340.00, 'ACI', '41004', 20, CURRENT_DATE, 113069,
                    2126, 111)

1 row inserted.
```

the INSERT statement can become lengthy if there are many columns of data, but its format is still very straightforward. The second INSERT statement uses the system constant CURRENT DATE in its VALUES clause, causing the current date to be inserted as the order date. This system constant is specified in the SQL2 standard and is supported by many of the popular SQL products. Other brands of DBMS provide other system constants or built-in functions to obtain the current date and time.

You can use the INSERT statement with interactive SQL to add rows to a table that grows very rarely, such as the OFFICES table. In practice, however, data about a new customer, order, or salesperson is almost always added to a database through a formsoriented data entry program. When the data entry is complete, the application program inserts the new row of data using programmatic SQL. Regardless of whether interactive or programmatic SQL is used, however, the INSERT statement is the same. The table name specified in the INSERT statement is normally an unqualified table name, specifying a table that you own. To insert data into a table owned by another user, you can specify a qualified table name. Of course, you must also have permission to insert data into the table, or the INSERT statement will fail.

The purpose of the column list in the INSERT statement is to match the data values in the VALUES clause with the columns that are to receive them. The list of values and the list of columns must both contain the same number of items, and the data type of each value must be compatible with the data type of the corresponding column, or an error will occur. The ANSI/ISO standard mandates unqualified column names in the column list, but many implementations allow qualified names. There can be no ambiguity in the column names anyway, because they must all reference columns of the target table.

Inserting NULL Values

When SQL inserts a new row of data into a table, it automatically assigns a NULL value to any column whose name is missing from the column list in the INSERT statement. In this INSERT statement, which added Mr. Jacobsen to the SALESREPS table, the QUOTA and MANAGER columns were omitted:

```
INSERT INTO SALESREPS (NAME, AGE, EMPL_NUM, SALES, TITLE,
                       HIRE_DATE, REP_OFFICE)
     VALUES ('Henry Jacobsen', 36, 111, 0.00, 'Sales Mgr',
                       '25-JUL-90', 13)
```

The newly added row has a NULL value in the QUOTA and MANAGER columns, as shown in Figure 2. You can make the assignment of a NULL value more explicit by including these columns in the column list and specifying the keyword NULL in the values list. This INSERT statement has exactly the same effect as the previous one:

```
INSERT INTO SALESREPS (NAME, AGE, EMPL_NUM, SALES, QUOTA, TITLE,
                       MANAGER, HIRE_DATE, REP_OFFICE)
     VALUES ('Henry Jacobsen', 36, 111, 0.00, NULL, 'Sales Mgr',
                       NULL, '25-JUL-90', 13)
```

Inserting All Columns

Keyword

Database administrators (DBAs) use specialized software to store and organize data. The role may include capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery.

As a convenience, SQL allows you to omit the column list from the INSERT statement. When the column list is omitted, SQL automatically generates a column list consisting of all columns of the table, in left-to-right sequence. This is the same column sequence generated by SQL when you use a SELECT * query. Using this shortcut, the previous INSERT statement could be rewritten equivalently as:

```
INSERT INTO SALESREPS
     VALUES (111, 'Henry Jacobsen', 36, 13, 'Sales Mgr',
             '25-JUL-90', NULL, NULL, 0.00)
```

When you omit the column list, the NULL keyword must be used in the values list to explicitly assign NULL values to columns, as shown in the example. In addition, the sequence of data values must correspond exactly to the sequence of columns in the table. Omitting the column list is convenient in interactive SQL because it reduces the length of the INSERT statement you must type. For programmatic SQL, the column list should always be specified because it makes the program easier to read and understand. In addition, table structures often change over time to include new columns or drop columns that are no longer used. A program that contains an INSERT statement without an explicit column list may work correctly for months or years, and then suddenly begin producing errors if the number of columns or data types of columns is changed by a **database administrator**.

4.1.2 The Multirow INSERT Statement

The second form of the INSERT statement, shown in Figure 3, adds multiple rows of data to its target table. In this form of the INSERT statement, the data values for the new rows are not explicitly specified within the statement text. Instead, the source of new rows is a database query, specified in the statement.

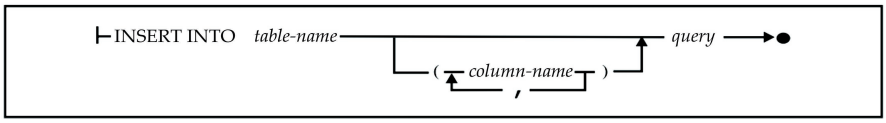


Figure 3: Multirow INSERT statement syntax diagram.

Adding rows whose values come from within the database itself may seem strange at first, but it's very useful in some special situations. For example, suppose you want to copy the order number, date, and amount of all orders placed before January 1, 1990, from the ORDERS table into another table, called OLDORDERS. The multirow INSERT statement provides a concise, efficient way to copy the data:

Copy old orders into the OLDORDERS table.

```
INSERT INTO OLDORDERS (ORDER_NUM, ORDER_DATE, AMOUNT)
     SELECT ORDER_NUM, ORDER_DATE, AMOUNT
       FROM ORDERS
      WHERE ORDER_DATE < '01-JAN-90'

9 rows inserted.
```

This INSERT statement looks complicated, but it's really very simple. The statement identifies the table to receive the new rows (OLDORDERS) and the columns to receive the data, just like the single-row INSERT statement. The remainder of the statement is a query that retrieves data from the ORDERS table. Figure 4 graphically illustrates the operation of this INSERT statement. Conceptually, SQL first performs the query against the ORDERS table and then inserts the query results, row by row, into the OLDORDERS table.

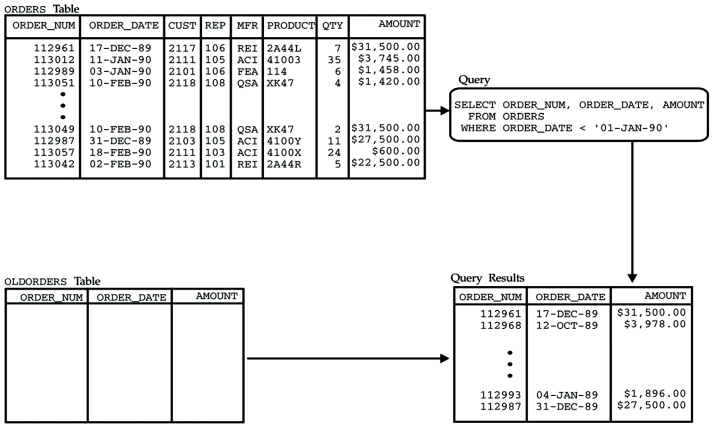


Figure 4: Inserting multiple rows.

Here's another situation where you could use the multirow INSERT statement. Suppose you want to analyze customer buying patterns by looking at which customers and salespeople are responsible for big orders—those over $15,000. The queries that you will be running will combine data from the CUSTOMERS, SALESREPS, and ORDERS tables. These three-table queries will execute fairly quickly on the small sample database, but in a real corporate database with many thousands of rows, they would take a long time. Rather than running many long, three-table queries, you could create a new table named BIGORDERS to contain the required data, defined as follows:

Column	Information
AMOUNT	Order amount (from ORDERS)
COMPANY	Customer name (from CUSTOMERS)
NAME	Salesperson name (from SALESREPS)
PERF	Amount over/under quota (calculated from SALESREPS)
MFR	Manufacturer ID (from ORDERS)
PRODUCT	Product ID (from ORDERS)
QTY	Quantity ordered (from ORDERS)

Once you have created the BIGORDERS table, this multirow INSERT statement can be used to populate it:

Load data into the BIGORDERS table for analysis.

```
INSERT INTO BIGORDERS (AMOUNT, COMPANY, NAME, PERF, PRODUCT, MFR, QTY)
     SELECT AMOUNT, COMPANY, NAME, (SALES - QUOTA), PRODUCT, MFR, QTY
       FROM ORDERS, CUSTOMERS, SALESREPS
      WHERE CUST = CUST_NUM
        AND REP = EMPL_NUM
        AND AMOUNT > 15000.00

6 rows inserted.
```

In a large database, this INSERT statement may take a while to execute because it involves a three-table query. When the statement is complete, the data in the BIGORDERS table will duplicate information in other tables. In addition, the BIGORDERS table won't be automatically kept up to date when new orders are added to the database, so its data may quickly become outdated. Each of these factors seems like a disadvantage. However, the subsequent data analysis queries against the BIGORDERS table can be expressed very simply—they become single-table queries.

Furthermore, each of those queries will run much faster than if it were a three table join. Consequently, this is probably a good strategy for performing the analysis, especially if the three original tables are large. In this situation, it's likely that the BIGORDERS table will be used as a temporary table for doing the analysis. It will be created and populated with data, representing a snapshot of the order status in time, the analysis programs will be run, and then the table will be emptied or dropped.

The SQL1 standard specifies several logical restrictions on the query that appears within the multirow INSERT statement:

- The query cannot contain an ORDER BY clause. It's useless to sort the query results anyway, because they're being inserted into a table that is, like all tables, unordered.
- The query results must contain the same number of columns as the column list in the INSERT statement (or the entire target table, if the column list is omitted), and the data types must be compatible, column by column.
- The query cannot be the UNION of several different SELECT statements. Only a single SELECT statement may be specified.
- The target table of the INSERT statement cannot appear in the FROM clause of the query or any subqueries that it contains. This prohibits inserting part of a table into itself.

The first two restrictions are structural, but the latter two were included in the standard simply to avoid complexity. As a result, these restrictions were relaxed in the SQL2 standard. The standard now allows UNION and JOIN operations and expressions in the query, basically allowing the results of a general database query to be retrieved and then inserted into a table with the INSERT statement. It also allows various forms of self-insertion, where the source table for the data to be inserted and the destination table are the same.

Did You Know?

SQL was adopted as a standard by the American National Standards Institute (ANSI) in 1986 as SQL-86 and the International Organization for Standardization (ISO) in 1987.

4.1.3 Bulk Load Utilities

Data to be inserted into a database is often downloaded from another computer system or collected from other sites and stored in a **sequential file**. To load the data into a table, you could write a program with a loop that reads each record of the file and uses the single-row INSERT statement to add the row to the table. However, the overhead of having the DBMS repeatedly execute single-row INSERT statements may be quite high. If inserting a single row takes half a second under a typical system load that is probably acceptable performance for an interactive program. But that performance quickly becomes

unacceptable when applied to the task of bulk loading 50,000 rows of data. In this case, loading the data would require over six hours.

For this reason, most commercial DBMS products include a bulk load feature that loads data from a file into a table at high speed. The ANSI/ISO SQL standard does not address this function, and it is usually provided as a stand-alone utility program rather than as part of the SQL language. Each vendor's utility provides a slightly different set of features, functions, and commands.

When SQL is used from within an application program, another technique is frequently provided for more efficiently inserting a large amount of data into a database. The standard programmatic INSERT statement inserts a single row of data, just like the interactive single-row INSERT statements in the preceding examples. But many commercial DBMS products allow data from two or more rows (often up to hundreds of rows) to be supplied as part of a single bulk INSERT statement. All of the supplied data must be for new rows of the single table that is the target of the INSERT statement, and named in the INTO clause. Executing a bulk INSERT statement for 100 rows of data has exactly the same effect as executing 100 individual single-row INSERT statements. However, it is usually much more efficient, because it involves only one call to the DBMS. Efficiency gains from 20% to 30% and up to 300% or more times over single-row INSERT statements are common, depending on the DBMS brand and the particular kind of data being inserted.

Keyword

A sequential file is one that contains and stores data in chronological order. The data itself may be ordered or unordered within the file.

4.2 Deleting Data from the Database

A row of data is typically deleted from a database when the entity represented by the row disappears from the outside world. For example, in the sample database:

- When a customer cancels an order, the corresponding row of the ORDERS table must be deleted.
- When a salesperson leaves the company, the corresponding row of the SALESREPS table must be deleted.

- When a sales office is closed, the corresponding row of the OFFICES table must be deleted. If the salespeople in the office are terminated, their rows should be deleted from the SALESREPS table as well. If they are reassigned, their REP_ OFFICE columns must be updated.

In each case, the row is deleted to maintain the database as an accurate model of the real world. The smallest unit of data that can be deleted from a relational database is a single row.

4.2.1 The DELETE Statement

The DELETE statement, shown in Figure 5, removes selected rows of data from a single table. The FROM clause specifies the target table containing the rows. The WHERE clause specifies which rows of the table are to be deleted. Suppose Henry Jacobsen, has just decided to leave the company. The DELETE statement that removes his row from the SALESREPS table is shown next.

Figure 5: DELETE statement syntax diagram.

Remove Henry Jacobsen from the database.

```
DELETE FROM SALESREPS
 WHERE NAME = 'Henry Jacobsen'

1 row deleted.
```

Recall that search conditions in the WHERE clause of a SELECT statement can specify a single row or an entire set of rows, depending on the specific search condition. The same is true of the WHERE clause in a DELETE statement. Suppose, for example, that Mr. Jacobsen's customer, InterCorp (customer number 2126) has called to cancel all its orders. Here is the DELETE statement that removes the orders from the ORDERS table:

Remove all orders for InterCorp (customer number 2126).

```
DELETE FROM ORDERS
WHERE CUST = 2126
```

2 rows deleted.

The WHERE clause selects several rows of the ORDERS table, and SQL removes all of the selected rows from the table. Conceptually, SQL applies the WHERE clause to each row of the ORDERS table, deleting those where the search condition yields a TRUE result and retaining those where the search condition yields a FALSE or NULL result. Because this type of DELETE statement searches through a table for the rows to be deleted, it is sometimes called a searched DELETE statement. This term is used to contrast it with another form of the DELETE statement, called the positioned DELETE statement, which always deletes a single row.

Here are some additional examples of searched DELETE statements:

Delete all orders placed before November 15, 1989.

```
DELETE FROM ORDERS
 WHERE ORDER_DATE < '15-NOV-89'

5 rows deleted.
```

Delete all rows for customers served by Bill Adams, Mary Jones, or Dan Roberts (employee numbers 105, 109, and 101).

```
DELETE FROM CUSTOMERS
 WHERE CUST_REP IN (105, 109, 101)

7 rows deleted.
```

Delete all salespeople hired before July 1988 who have not yet been assigned a quota.

```
DELETE FROM SALESREPS
 WHERE HIRE_DATE < '01-JUL-88'
   AND QUOTA IS NULL

0 rows deleted.
```

4.2.2 Deleting All Rows

The WHERE clause in a DELETE statement is optional, but it is almost always present. If the WHERE clause is omitted from a DELETE statement, all rows of the target table are deleted, as in this example:

Delete all orders.

```
DELETE FROM ORDERS
```

30 rows deleted.

Although this DELETE statement produces an empty table, it does not erase the ORDERS table from the database. The definition of the ORDERS table and its columns is still stored in the database. The table still exists, and new rows can still be inserted into the ORDERS table with the INSERT statement. To erase the definition of the table from the database, the DROP TABLE statement must be used.

Because of the potential damage from such a DELETE statement, be careful to always specify a search condition, and be sure that it actually selects the rows you want. When using **interactive SQL**, it's a good idea first to use the WHERE clause in a SELECT statement to display the selected rows. Make sure they are the ones you want to delete, and only then use the WHERE clause in a DELETE statement.

Keyword

Interactive SQL (dbisql) is a utility for entering SQL statements. If you use Interactive SQL to work with your database schema, instead of executing the SQL statements one at a time, build up the set of commands in a dbisql command file.

4.2.3 DELETE with Subquery

DELETE statements with simple search conditions, such as those in the previous examples, select rows for deletion based solely on the contents of the rows themselves. Sometimes the selection of rows must be made based on data from other tables. For example, suppose you want to delete all orders taken by Sue Smith. Without knowing her employee number, you can't find the orders by consulting the ORDERS table alone. To find the orders, you could use a two-table query:

Find the orders taken by Sue Smith.

```
SELECT ORDER_NUM, AMOUNT
  FROM ORDERS, SALESREPS
 WHERE REP = EMPL_NUM
   AND NAME = 'Sue Smith'

 ORDER_NUM       AMOUNT
---------- -----------
    112979  $15,000.00
    113065   $2,130.00
    112993   $1,896.00
    113048   $3,750.00
```

But you can't use a join in a DELETE statement. The parallel DELETE statement is illegal:

```
DELETE FROM ORDERS, SALESREPS
 WHERE REP = EMPL_NUM
   AND NAME = 'Sue Smith'

Error: More than one table specified in FROM clause
```

The way to handle the request is with one of the subquery search conditions. Here is a valid form of the DELETE statement that handles the request:

Delete the orders taken by Sue Smith.

```
DELETE FROM ORDERS
 WHERE REP = (SELECT EMPL_NUM
                FROM SALESREPS
               WHERE NAME = 'Sue Smith')

4 rows deleted.
```

The subquery finds the employee number for Sue Smith, and the WHERE clause then selects the orders with a matching value. As this example shows, subqueries can play an important role in the DELETE statement because they let you delete rows based on information in other tables. Here are two more examples of DELETE statements that use subquery search conditions:

Delete customers served by salespeople whose sales are less than 80% of quota.

```
DELETE FROM CUSTOMERS
 WHERE CUST_REP IN (SELECT EMPL_NUM
                      FROM SALESREPS
                     WHERE SALES < (.8 * QUOTA))

2 rows deleted.
```

Delete any salesperson whose current orders total less than 2 percent of their quota.

```
DELETE FROM SALESREPS
 WHERE (.02 * QUOTA) > (SELECT SUM(AMOUNT)
                          FROM ORDERS
                         WHERE REP = EMPL_NUM)

1 row deleted.
```

Subqueries in the WHERE clause can be nested just as they can be in the WHERE clause of the SELECT statement. They can also contain outer references to the target table of the DELETE statement. In this respect, the FROM clause of the DELETE statement functions like the FROM clause of the SELECT statement. Here is an example of a deletion request that requires a subquery with an outer reference:

Delete customers who have not ordered since November 10, 1989.

```
DELETE FROM CUSTOMERS
 WHERE NOT EXISTS (SELECT *
                     FROM ORDERS
                    WHERE CUST = CUST_NUM
                      AND ORDER_DATE < '10-NOV-89')

16 rows deleted.
```

Conceptually, this DELETE statement operates by going through the CUSTOMERS table, row by row, and checking the search condition. For each customer, the subquery selects any orders placed by that customer before the cutoff date. The reference to the CUST_NUM column in the subquery is an outer reference to the customer number in the row of the CUSTOMERS table currently being checked by the DELETE statement.

Outer references will often be found in subqueries of a DELETE statement, because they implement the join between the table(s) in the subquery and the target table of the DELETE statement. In the SQL1 standard, a restriction on the use of subqueries in a DELETE statement prevents the target table from appearing in the FROM clause of a subquery or any of its subqueries at any level of nesting. This prevents the subqueries from referencing the target table (some of whose rows may already have been deleted), except for outer references to the row currently being tested by the DELETE statement's search condition. The SQL2 standard eliminates this restriction by specifying that the DELETE statement should treat such a subquery as applying to the entire target table, before any rows have been deleted. This places more overhead on the DBMS (which must handle the subquery processing and row deletion more carefully), but the behavior of the statement is well defined by the standard.

4.3 MODIFYING DATA IN THE DATABASE

Typically, the values of data items stored in a database are modified when corresponding changes occur in the outside world. For example, in the sample database:

- When a customer calls to change the quantity on an order, the QTY column in the appropriate row of the ORDERS table must be modified.
- When a manager moves from one office to another, the MGR column in the OFFICES table and the REP_OFFICE column in the SALESREPS table must be changed to reflect the new assignment.
- When sales quotas are raised by 5% in the New York sales office, the QUOTA column of the appropriate rows in the SALESREPS table must be modified.

In each case, data values in the database are updated to maintain the database as an accurate model of the real world. The smallest unit of data that can be modified in a database is a single column of a single row.

4.3.1 The UPDATE Statement

The UPDATE statement, shown in Figure 6, modifies the values of one or more columns in selected rows of a single table. The WHERE clause selects the rows of the table to be modified. The SET clause specifies which columns are to be updated and calculates the new values for them. Here is a simple UPDATE statement that changes the credit limit and salesperson for a customer:

Raise the credit limit for Acme Manufacturing to $60,000 and reassign them to Mary Jones (employee number 109).

```
UPDATE CUSTOMERS
   SET CREDIT_LIMIT = 60000.00, CUST_REP = 109
 WHERE COMPANY = 'Acme Mfg.'

1 row updated.
```

Remember

The target table to be updated is named in the statement, and you must have the required permission to update the table as well as each of the individual columns that will be modified.

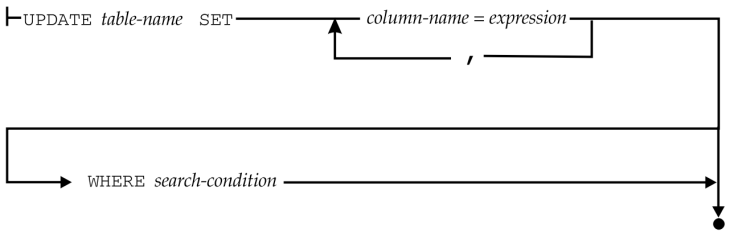


Figure 6: UPDATE statement syntax diagram.

In this example, the WHERE clause identifies a single row of the CUSTOMERS table, and the SET clause assigns new values to two of the columns in that row. The WHERE clause is exactly the same one you would use in a DELETE or SELECT statement to identify the row. In fact, the search conditions that can appear in the WHERE clause of an **UPDATE statement** are exactly the same as those available in the SELECT and DELETE statements. Like the DELETE statement, the UPDATE statement can update several rows at once with the proper search condition, as in this example:

Transfer all salespeople from the Chicago office (number 12) to the New York office (number 11), and lower their quotas by 10 percent.

```
UPDATE SALESREPS
   SET REP_OFFICE = 11, QUOTA = .9 * QUOTA
 WHERE REP_OFFICE = 12

3 rows updated.
```

Keyword

UPDATE statement changes the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition.

The WHERE clause selects several rows of the SALESREPS table, and the value of the REP_OFFICE and QUOTA columns are modified in all of them. Conceptually, SQL processes the UPDATE statement by going through the SALESREPS table row by row, updating those rows for which the search condition yields a TRUE result and skipping over those for which the search condition yields a FALSE or NULL result. Because it searches the table, this form of the UPDATE statement is sometimes called a searched UPDATE statement. This term distinguishes it from a different form of the UPDATE statement, called a positioned UPDATE statement, which always updates a single row.

Here are some additional examples of searched UPDATE statements:

Reassign all customers served by employee numbers 105, 106, or 107 to employee number 102.

```
UPDATE CUSTOMERS
   SET CUST_REP = 102
 WHERE CUST_REP IN (105, 106, 107)

5 rows updated.
```

Assign a quota of $100,000 to any salesperson who currently has no quota.

```
UPDATE SALESREPS
SET QUOTA = 100000.00
WHERE QUOTA IS NULL
1 row updated.
```

The SET clause in the UPDATE statement is a list of assignments separated by commas. Each assignment identifies a target column to be updated and specifies how to calculate the new value for the target column. Each target column should appear only once in the list; there should not be two assignments for the same target column. The ANSI/ISO specification mandates unqualified names for the target

columns, but some SQL implementations allow qualified column names. There can be no ambiguity in the column names anyway, because they must refer to columns of the target table. The expression in each assignment can be any valid SQL expression that yields a value of the appropriate data type for the target column. The expression must be computable based on the values of the row currently being updated in the target table. In most DBMS implementations, the expression may not include any column functions or subqueries.

If an expression in the assignment list references one of the columns of the target table, the value used to calculate the expression is the value of that column in the current row before any updates are applied. The same is true of column references that occur in the WHERE clause. For example, consider this (somewhat contrived) UPDATE statement:

```
UPDATE OFFICES
   SET QUOTA = 400000.00, SALES = QUOTA
 WHERE QUOTA < 400000.00
```

Before the update, Bill Adams had a QUOTA value of $350,000 and a SALES value of $367,911. After the update, his row has a SALES value of $350,000, not $400,000. The order of the assignments in the SET clause is thus immaterial; the assignments can be specified in any order.

4.3.2 Updating All Rows

The WHERE clause in the UPDATE statement is optional. If the WHERE clause is omitted, then all rows of the target table are updated, as in this example:

Raise all quotas by 5 percent.

```
UPDATE SALESREPS
   SET QUOTA = 1.05 * QUOTA

10 rows updated.
```

Unlike the DELETE statement, in which the WHERE clause is almost never omitted, the UPDATE statement without a WHERE clause performs a useful function. It basically performs a bulk update of the entire table, as demonstrated in the preceding example.

4.3.3 UPDATE with Subquery

As with the DELETE statement, subqueries can play an important role in the UPDATE statement because they let you select rows to update based on information contained in other tables. Here are several examples of UPDATE statements that use subqueries:

Raise by $5000 the credit limit of any customer who has placed an order for more than $25,000.

```
UPDATE CUSTOMERS
   SET CREDIT_LIMIT = CREDIT_LIMIT + 5000.00
 WHERE CUST_NUM IN (SELECT DISTINCT CUST
                      FROM ORDERS
                     WHERE AMOUNT > 25000.00)

4 rows updated.
```

Reassign all customers served by salespeople whose sales are less than 80 percent of their quota.

```
UPDATE CUSTOMERS
   SET CUST_REP = 105
 WHERE CUST_REP IN (SELECT EMPL_NUM
                      FROM SALESREPS
                     WHERE SALES < (.8 * QUOTA))

2 rows updated.
```

Have all salespeople who serve over three customers report directly to Sam Clark (employee number 106).

```
UPDATE SALESREPS
   SET MANAGER = 106
 WHERE 3 < (SELECT COUNT(*)
              FROM CUSTOMERS
             WHERE CUST_REP = EMPL_NUM)

1 row updated.
```

As in the DELETE statement, subqueries in the WHERE clause of the UPDATE statement can be nested to any level and can contain outer references to the target table of the UPDATE statement. The column EMPL_NUM in the subquery of the preceding example is such an outer reference; it refers to the EMPL_NUM column in the row of the SALESREPS table currently being checked by the UPDATE statement.

Outer references will often be found in subqueries of an UPDATE statement, because they implement the join between the table(s) in the subquery and the target table of the UPDATE statement. The same SQL1 restriction applies as for the DELETE statement: the target table cannot appear in the FROM clause of any subquery at any level of nesting. This prevents the subqueries from referencing the target table (some of whose rows may have already been updated). Any references to the target table in the subqueries are thus outer references to the row of the target table currently being tested by the UPDATE statement's WHERE clause. The SQL2 standard again removed this restriction and specifies that a reference to the target table in a subquery is evaluated as if none of the target table had been updated.

CASE STUDY

COMMONWEALTH SWIMMING

New Zealand sent a team of 18 swimmers to the Melbourne 2006 Commonwealth Games. Information about the swimmers, the events they competed in, and the results of their races are shown in Figure 1.

first	last	length	stroke	gender	stage	time	place
Zoe	Baker	50	Breaststroke	female	heat	31.7	4
Zoe	Baker	50	Breaststroke	female	semi	31.84	5
Zoe	Baker	50	Breaststroke	female	final	31.45	4
Lauren	Boyle	200	Freestyle	female	heat	121.11	8
Lauren	Boyle	200	Freestyle	female	semi	120.9	8
Lauren	Boyle	100	Freestyle	female	heat	56.7	10
Lauren	Boyle	100	Freestyle	female	semi	56.4	9

...

Figure 1: A subset of the data recorded for New Zealand swimmers at the Melbourne 2006 Commonwealth Games, including the name and gender of each swimmer and the distance, stroke, stage, and result for each event that they competed in.

These data have been stored in a database with six tables.

The swimmer_table has one row for each swimmer and contains the first and last name of each swimmer. Each swimmer also has a unique numeric identifier.

swimmer_table (ID [PK], first, last)

There are four tables that define the set of valid events: the distances are 50m, 100m, and 200m; the swim strokes are breaststroke (Br), freestyle (Fr), butterfly (Bu), and backstroke (Ba); the genders are male (M) and female (F); and the possible race stages are heats (heat), semifinals (semi), and finals (final).

stroke_table (ID [PK], stroke)

gender_table (ID [PK], gender)

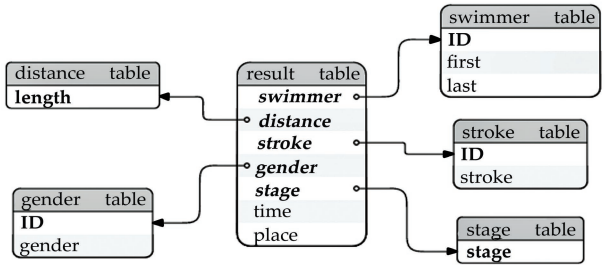
stage_table (stage [PK])

The result_table contains information on the races swum by individual swimmers. Each row specifies a swimmer and the type of race (distance, stroke, gender, and stage). In addition, the swimmer's time and position in the race (place) are recorded.

result_table (swimmer [PK] [FK swimmer_table.ID],

distance [PK] [FK distance_table.length],
stroke [PK] [FK stroke_table.ID],
gender [PK] [FK gender_table.ID],
stage [PK] [FK stage_table.stage],
time, place)

The database design is illustrated in the diagram below.



As an example of the information stored in this database, the following code shows that the swimmer with an ID of 1 is called Zoe Baker. This SQL query, and the next, are not joins, they are just simple one-table queries to show what sort of data is contained in the database.

```
> SELECT * FROM swimmer_table
        WHERE ID = 1;
ID  first  last
--  -----  -----
1   Zoe    Baker
```

Notice the use of * in this query to denote that we want all columns from the table in our result.

The following code shows that Zoe Baker swam in three races--a heat, a semifinal and the final of the women's 50m breaststroke--and she came $4^{\rm th}$ in the final in a time of 31.45 seconds.

```
> SELECT * FROM result_table
        WHERE swimmer = 1;
swimmer  distance  stroke  gender  stage  time   place
-------  --------  ------  ------  -----  -----  -----
1        50        Br      F       final  31.45  4
1        50        Br      F       heat   31.7   4
1        50        Br      F       semi   31.84  5
```

SUMMARY

- SQL is a complete data manipulation language that is used not only for database queries, but also to modify and update data in the database. Compared to the complexity of the SELECT statement, which supports SQL queries, the SQL statements that modify database contents are extremely simple. However, database updates pose some challenges for a DBMS beyond those presented by database queries.
- A new row of data is typically added to a relational database when a new entity represented by the row appears in the outside world.
- A single-row INSERT statement adds a single new row of data to a table. It is commonly used in daily applications—for example, data entry programs.
- A multirow INSERT statement extracts rows of data from another part of the database and adds them to a table. It is commonly used in end-of-month or end-of-year processing when old rows of a table are moved to an inactive table.
- A bulk load utility adds data to a table from a file that is outside of the database. It is commonly used to initially load the database or to incorporate data downloaded from another computer system or collected from many sites.
- A row of data is typically deleted from a database when the entity represented by the row disappears from the outside world.
- The UPDATE statement modifies the values of one or more columns in selected rows of a single table. The WHERE clause selects the rows of the table to be modified. The SET clause specifies which columns are to be updated and calculates the new values for them.

KNOWLEDGE CHECK

1. Which statement is used for updating existing information in the table?

a. Update
B. Where
C. Modify
D. Alter

2. In SQL, which command is used to add new rows to a table?

a. Alter Table
b. Add row
c. Insert
d. Append

3. A table that displays data redundancies yields ..... anomalies.

a. Update
b. Insertion
c. Deletion
d. All of the Mentioned

4. In the following query how many rows will be updated?

```
UPDATE person
SET lname='s',
Fname='p',
WHERE person_id&lt;10;
/* person_id is a primary key */
```

a. 0-9
b. 1-6
c. No row
d. None of the mentioned

5. "INSERT" is same as "UPDATE"?

a. NO
b. YES
c. May be
d. None of the mentioned

6. **Which one is correct syntax for Update Statement?**
   a. Update Table Columns(Col1, Col2,Col3);
   b. Update into (Col1, Col2,Col3) VALUES (Val1,Val2,Val3);
   c. Update Set Col_name=Value;
   d. None of the above.
7. **Which of the following SQL clauses is used to DELETE tuples from a database table?**
   a. DELETE
   b. REMOVE
   c. DROP
   d. CLEAR
8. **Which of the following is not a DDL command?**
   a. UPDATE
   b. TRUNCATE
   c. ALTER
   d. None of the Mentioned
9. **Which of the following command makes the updates performed by the transaction permanent in the database?**
   a. ROLLBACK
   b. COMMIT
   c. TRUNCATE
   d. DELETE
10. **The result of a SQL SELECT statement is a _____.**
   a. file
   b. report
   c. table
   d. form

REVIEW QUESTIONS

1. What is relational database?
2. When is the UPDATE_STATISTICS command used?
3. Discuss the Multirow INSERT statement.
4. What is DELETE statements?
5. Describe the modifying data in the database.

Check Your Result

1. (a)	2. (c)	3. (d)	4. (a)	5. (a)
6. (c)	7. (a)	8. (a)	9. (b)	10. (c)

REFERENCES

1. Bernhard Thalheim, Klaus-Dieter Schewe (2011). "NULL 'Value' Algebras and Logics". Frontiers in Artificial Intelligence and Applications. 225 (Information Modelling and Knowledge Bases XXII). doi:10.3233/978-1-60750-690-4-354.
2. Claude Rubinson, Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique, SIGMOD Record, December 2007 (Vol. 36, No. 4)
3. Date, C.J. (2000). The Database Relational Model: A Retrospective Review and Analysis: A Historical Account and Assessment of E. F. Codd's Contribution to the Field of Database Technology. Addison Wesley Longman. ISBN 978-0-201-61294-3.
4. Drake, Mark (August 9, 2019). "A Comparison of NoSQL Database Management Systems and Models". Digital Ocean. Retrieved 2021-02-26.
5. Enrico Franconi and Sergio Tessaris, On the Logic of SQL Nulls, Proceedings of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, Ouro Preto, Brazil, June 27–30, 2012. pp. 114–128
6. John Grant, Null Values in SQL. SIGMOD Record, September 2008 (Vol. 37, No. 3)
7. Rosenberg, Burton. "Relational Databases". University of Miami. Retrieved 2021-02-26.
8. Waraporn, Narongrit, and Kriengkrai Porkaew. "Null semantics for subqueries and atomic predicates". IAENG International Journal of Computer Science 35.3 (2008): 305-313.

CHAPTER

5

DATABASE STRUCTURE

"Web pages are designed for people. For the Semantic Web, we need to look at existing databases".

—Tim Berners-Lee

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Explain the database design
2. Define the database schema versus database instance
3. Understanding the database models

INTRODUCTION

A database is an organized collection of data stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques.

The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database.

Computer scientists may classify database-management systems according to the database models that they support. Relational databases became dominant in the 1980s. These model data as rows and columns in a series of tables, and the vast majority use SQL for writing and querying data. In the 2000s, non-relational databases became popular, referred to as NoSQL because they use different query languages.

Database Structure," deals with creating and administering a SQL-based database. It describes the SQL security scheme that prevents unauthorized access to data, and the SQL system catalog that describes the structure of a database. This part also discusses the significant differences between the database structures supported by various SQL-based DBMS products.

5.1 DATABASE DESIGN

Database design, as the name might suggest, is much like house design, though the term also can be used to refer to actual database construction. The design process is something of a blueprint that outlines a database's details, from relationships between tables to what information is important and how the data will be implemented. Aside from helping the builder know what tables and information to collect, a design uses naming conventions, and spelling errors are checked before the database is completed. The database also goes through normalization, which seeks to remove redundancy, during the design process. Without first working out a design, a database creator can easily mess up the order of tables or the primary key for tables, or simply miss a few sections, among a slew of other potential errors.

The first step of database design is to know the purpose of the database. There are no diagrams or abstract representations; the designer just thinks about the database's objectives. Some information may be written down, but generally the designer simply considers the best way to organize and use the database.

Next, the designer creates four data models. The conceptual model is a simple diagram that shows table names. After this, the logical data model is created, filling the tables with primary key and information to be collected. A primary key is a title for a column that makes it unique and tells users the purpose of the column. Relationships between tables also are detailed during this database design stage.

In the entity-relationship model, the designer focuses more on relationships and less on the primary keys. This model may sometimes be skipped, but it helps during database creation to show how the entities interact with one another. In the physical data model, live information is fed into the database design.

During each model stage, the spelling of the tables and primary keys must be checked. Naming conventions also are employed, so users know how to enter data. For example, a table could be named "ThisTable," "This_Table," "This-Table," or "This.Table", based on the naming convention picked by the database designer. Spelling has to be checked, because an error can cause relationship issues when the database is constructed.

The rules of normalization also are applied to the database model. These rules eliminate repeating data, dissolve large tables into small tables and ensure that relationships are optimized. Normalizing the database design is the last step and will aid the designer in determining if the database is functional or if it needs to be rearranged or reworked.

Keyword

SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

5.1.1 The database design process

A well-structured database:

- Saves disk space by eliminating redundant data.
- Maintains data accuracy and integrity.
- Provides access to the data in useful ways.

Designing an efficient, useful database is a matter of following the proper process, including these phases:

- Requirements analysis, or identifying the purpose of your database
- Organizing data into tables
- Specifying primary keys and analyzing relationships
- Normalizing to standardize the tables

Let's take a closer look at each step. Note that this guide deals with Edgar Codd's relational database model as written in **SQL** (rather than the hierarchical, network, or object data models). To learn more about database models, read our guide here.

5.1.2 Requirements analysis: identifying the purpose of the database

Understanding the purpose of your database will inform your choices throughout the design process. Make sure you consider the database from every perspective. For instance, if you were making a database for a public library, you'd want to consider the ways in which both patrons and librarians would need to access the data.

Here are some ways to gather information before creating the database:

- Interview the people who will use it
- Analyze business forms, such as invoices, timesheets, surveys
- Comb through any existing data systems (including physical and digital files)

Start by gathering any existing data that will be included in the database. Then list the types of data you want to store and the entities, or people, things, locations, and events, that those data describe, like this:

Customers

- Name
- Address
- City, State, Zip
- Email address

Products

- Name
- Price
- Quantity in stock
- Quantity on order

Orders

- Order ID
- Sales representative
- Date
- Product(s)
- Quantity
- Price
- Total

This information will later become part of the data dictionary, which outlines the tables and fields within the database. Be sure to break down the information into the smallest useful pieces. For instance, consider separating the street address from

the country so that you can later filter individuals by their country of residence. Also, avoid placing the same data point in more than one table, which adds unnecessary complexity.

Once you know what kinds of data the database will include, where that data comes from, and how it will be used, you're ready to start planning out the actual database.

5.1.3 Database structure: the building blocks of a database

The next step is to lay out a visual representation of your database. To do that, you need to understand exactly how relational databases are structured.

Within a database, related data are grouped into tables, each of which consists of rows (also called tuples) and columns, like a spreadsheet.

To convert your lists of data into tables, start by creating a table for each type of entity, such as products, sales, customers, and orders. Here's an example: Each row of a table is called a record. Records include data about something or someone, such as a particular customer. By contrast, columns (also known as fields or attributes) contain a single type of information that appears in each record, such as the addresses of all the customers listed in the table.

SQL-based databases citizens use every day include banking systems, .computerized medical records, and online shopping to name just a few

First Name	Last Name	Age	ZIP Code
Roger	Williams	43	34760
Jerrica	Jorgensen	32	97453
Samantha	Hopkins	56	64829

To keep the data consistent from one record to the next, assign the appropriate data type to each column. Common data types include:

- CHAR - a specific length of text
- VARCHAR - text of variable lengths
- TEXT - large amounts of text

- INT - positive or negative whole number
- FLOAT, DOUBLE - can also store floating point numbers
- BLOB - binary data

Some database management systems also offer the Auto number data type, which automatically generates a unique number in each row.

For the purposes of creating a visual overview of the database, known as an entity-relationship diagram, you won't include the actual tables. Instead, each table becomes a box in the diagram. The title of each box should indicate what the data in that table describes, while attributes are listed below, like this:

Student
Student ID
Birth Date
Grade level

Keyword

Primary Key is a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table).

Finally, you should decide which attribute or attributes will serve as the primary key for each table, if any. A **primary key** (PK) is a unique identifier for a given entity, meaning that you could pick out an exact customer even if you only knew that value.

Attributes chosen as primary keys should be unique, unchanging, and always present (never NULL or empty). For this reason, order numbers and usernames make good primary keys, while telephone numbers or street addresses do not. You can also use multiple fields in conjunction as the primary key (this is known as a composite key).

When it comes time to create the actual database, you'll put both the logical data structure and the physical data structure into the data definition language supported by your database management system. At that point, you should also estimate the size of the database to be sure you can get the performance level and storage space it will require.

5.1.4 Creating relationships between entities

With your database tables now converted into tables, you're ready to analyze the relationships between those tables. Cardinality refers to the quantity of elements that interact between two related tables. Identifying the cardinality helps make sure you've divided the data into tables most efficiently.

Each entity can potentially have a relationship with every other one, but those relationships are typically one of three types:

One-to-one relationships

When there's only one instance of Entity A for every instance of Entity B, they are said to have a one-to-one relationship (often written 1:1). You can indicate this kind of relationship in an ER diagram with a line with a dash on each end:

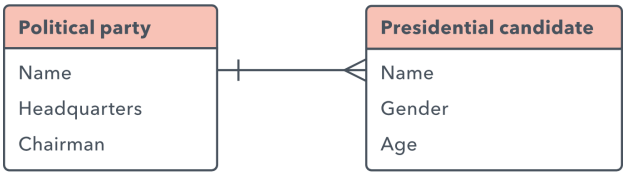
Unless you have a good reason not to, a 1:1 relationship usually indicates that you'd be better off combining the two tables' data into a single table.

However, you might want to create tables with a 1:1 relationship under a particular set of circumstances. If you have a field with optional data, such as "description," that is blank for many of the records, you can move all of the descriptions into their own table, eliminating empty space and improving database performance.

To guarantee that the data matches up correctly, you'd then have to include at least one identical column in each table, most likely the primary key.

One-to-many relationships

These relationships occur when a record in one table is associated with multiple entries in another. For example, a single customer might have placed many orders, or a patron may have multiple books checked out from the library at once. One-to-many (1:M) relationships are indicated with what's called "Crow's foot notation," as in this example:



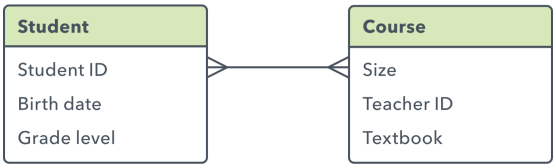
To implement a 1:M relationship as you set up a database, simply add the primary key from the "one" side of the relationship as an attribute in the other table. When a primary key is listed in another table in this manner, it's called a foreign key. The

table on the "1" side of the relationship is a considered a parent table to the child table on the other side.

Many-to-many relationships

When multiple entities from a table can be associated with multiple entities in another table, they are said to have a many-to-many (M: N) relationship. This might happen in the case of students and classes, since a student can take many classes and a class can have many students.

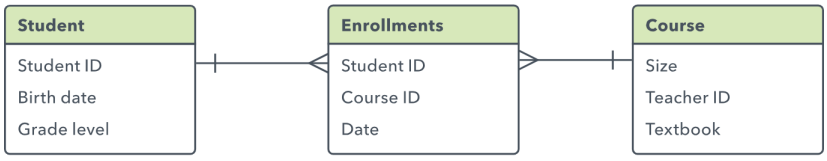
In an ER diagram, these relationships are portrayed with these lines:



Unfortunately, it's not directly possible to implement this kind of relationship in a database. Instead, you have to break it up into two one-to-many relationships.

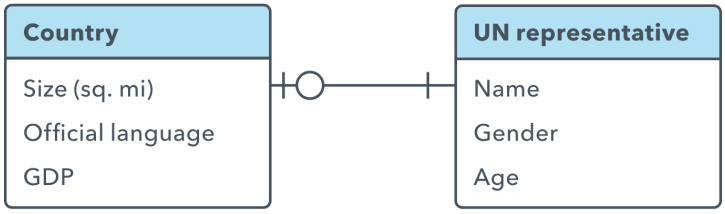
To do so, create a new entity between those two tables. If the M: N relationship exists between sales and products, you might call that new entity "sold_products," since it would show the contents of each sale. Both the sales and products tables would have a 1:M relationship with sold_products. This kind of go-between entity is called a link table, associative entity, or junction table in various models.

Each record in the link table would match together two of the entities in the neighboring tables (it may include supplemental information as well). For instance, a link table between students and classes might look like this:



Mandatory or not?

Another way to analyze relationships is to consider which side of the relationship has to exist for the other to exist. The non-mandatory side can be marked with a circle on the line where a dash would be. For instance, a country has to exist for it to have a representative in the United Nations, but the opposite is not true:



Two entities can be mutually dependent (one could not exist without the other).

Recursive relationships

Sometimes a table points back to itself. For example, a table of employees might have an attribute "manager" that refers to another individual in that same table. This is called a recursive relationship.

Redundant relationships

A redundant relationship is one that is expressed more than once. Typically, you can remove one of the relationships without losing any important information. For instance, if an entity "students" has a direct relationship with another called "teachers" but also has a relationship with teachers indirectly through "classes," you'd want to remove the relationship between "students" and "teachers." It's better to delete that relationship because the only way that students are assigned to teachers is through classes.

Keyword

Online analytical processing is an approach to answering multi-dimensional analytical (MDA) queries swiftly in computing.

5.1.5 Database Normalization

Once you have a preliminary design for your database, you can apply normalization rules to make sure the tables are structured correctly. Think of these rules as the industry standards.

That said, not all databases are good candidates for normalization. In general, online transaction processing (OLTP for short) databases, in which users are concerned with creating, reading, updating, and deleting records, should be normalized.

Online analytical processing (OLAP) databases which favor analysis and reporting might fare better with a degree of denormalization, since the emphasis is on speed of calculation.

These include decision support applications in which data needs to be analyzed quickly but not changed.

Each form, or level of normalization, includes the rules associated with the lower forms.

First normal form

The first normal form (abbreviated as 1NF) specifies that each cell in the table can have only one value, never a list of values, so a table like this does not comply:

ProductID	Color	Price
1	brown, yellow	$15
2	red, green	$13
3	blue, orange	$11

You might be tempted to get around this by splitting that data into additional columns, but that's also against the rules: a table with groups of repeated or closely related attributes does not meet the first normal form. The table below, for example, fails to comply:

Product
Color1
Color2
Color3
Price

Instead, split the data into multiple tables or records until each cell holds only one value and there are no extra columns. At that point, the data is said to be atomic, or broken down to the smallest useful size. For the table above, you could create an additional table called "Sales details" that would match specific products with sales. "Sales" would then have a 1:M relationship with "Sales details."

Second normal form

The second normal form (2NF) mandates that each of the attributes should be fully dependent on the entire primary key. That means each attribute should depend directly on the primary key, rather than indirectly through some other attribute.

For instance, an attribute "age" that depends on "birthdate" which in turn depends on "studentID" is said to have a partial functional dependency, and a table containing these attributes would fail to meet the second normal form.

Furthermore, a table with a primary key made up of multiple fields violates the second normal form if one or more of the other fields do not depend on every part of the key.

Thus, a table with these fields wouldn't meet the second normal form, because the attribute "product name" depends on the product ID but not on the order number:

- Order number (primary key)
- Product ID (primary key)
- Product name

Third normal form

The third normal form (3NF) adds to these rules the requirement that every non-key column be independent of every other column. If changing a value in one non-key column causes another value to change, that table does not meet the third normal form.

This keeps you from storing any derived data in the table, such as the "tax" column below, which directly depends on the total price of the order:

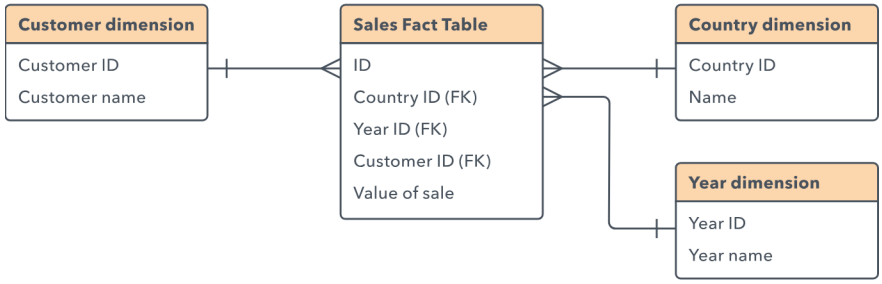
Order	Price	Tax
14325	$40.99	$2.05
14326	$13.73	$.69
14327	$24.15	$1.21

Additional forms of normalization have been proposed, including the Boyce-Codd normal form, the fourth through sixth normal forms, and the domain-key normal form, but the first three are the most common.

While these forms explain the best practices to follow generally, the degree of normalization depends on the context of the database.

5.1.6 Multidimensional data

Some users may want to access multiple dimensions of a single type of data, particularly in OLAP databases. For instance, they may want to know the sales by customer, state, and month. In this situation, it's best to create a central fact table that other customer, state, and month tables can refer to, like this:



5.1.7 Data integrity rules

You should also configure your database to validate the data according to the appropriate rules. Many database management systems, such as Microsoft Access, enforce some of these rules automatically.

The entity integrity rule says that the primary key can never be NULL. If the key is made up of multiple columns, none of them can be NULL. Otherwise, it could fail to uniquely identify the record.

The referential integrity rule requires each foreign key listed in one table to be matched with one primary key in the table it references. If the primary key changes or is deleted, those changes will need to be implemented wherever that key is referenced throughout the database.

Business logic integrity rules make sure that the data fits within certain logical parameters. For instance, an appointment time would have to fall within normal business hours.

5.1.8 Adding indexes and views

An index is essentially a sorted copy of one or more columns, with the values either in ascending or descending order. Adding an index allows users to find records more quickly. Instead of re-sorting for each query, the system can access records in the order specified by the index. Although indexes speed up data retrieval, they can slow down inserting, updating, and deleting, since the index has to be rebuilt whenever a record is changed.

A view is simply a saved query on the data. They can usefully join data from multiple tables or else show part of a table.

5.1.9 Extended properties

Once you have the basic layout completed, you can refine the database with extended properties, such as instructional text, input masks, and formatting rules that apply to a particular schema, view, or column. The advantage is that, because these rules are stored in the database itself, the presentation of the data will be consistent across the multiple programs that access the data.

5.1.10 SQL and UML

The **Unified Modeling Language** (UML) is another visual way of expressing complex systems created in an object-oriented language. Several of the concepts mentioned in this guide are known in UML under different names. For instance, an entity is known as a class in UML.

UML is not used as frequently today as it once was. Today, it is often used academically and in communications between software designers and their clients.

Keyword

Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

5.1.11 Database Management Systems

Many of the design choices you will make depend on which database management system you use. Some of the most common systems include:

- Oracle DB
- MySQL
- Microsoft SQL Server
- PostgreSQL
- IBM DB2

When given the choice, pick an appropriate database management system based on cost, operating systems

5.2 DATABASE SCHEMA VERSUS DATABASE INSTANCE

While working with any data model, it is necessary to distinguish between the overall design or description of the

database (database schema) and the database itself. As discussed, the overall design or description of the database is known as database schema or simply schema. The database schema is also known as intension of the database, and is specified while designing the database. Figure 1 shows the database schema for online book database.

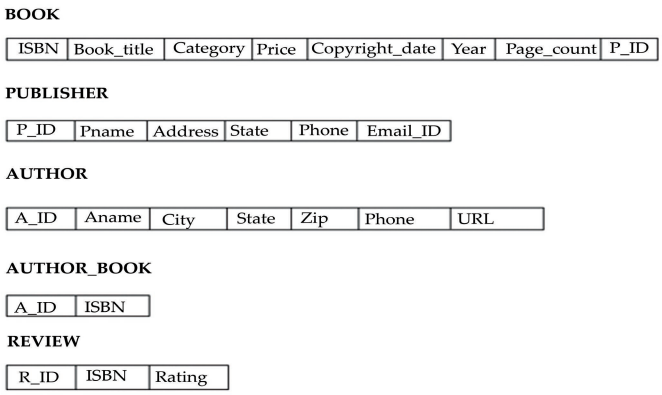


Figure 1: Database schema for online book database.

The data in the database at a particular point of time is known as database instance or database state or snapshot. The database state is also called an extension of the schema.

The various states of the database are given here:

Empty State: When a new database is defined, only its schema is specified. At this point, the database is said to be in empty state as it contains no data.

Initial State: When the database is loaded with data for the first time, it is said to be in initial state.

Current State: The data in the database is updated frequently. Thus, at any point of time, the database is said to be in the current state.

The DBMS is responsible to check whether the database state is valid state. Thus, each time the database is updated, DBMS ensures that the database remains in the valid state. The DBMS refers to DBMS catalog where the metadata is stored in order to check whether the database state satisfies the structure and constraints specified in the schema. Table 1 shows an example of an instance for PUBLISHER schema.

Table 1: An example of instance PUBLISH

P_ID	Pname	Address	State	Phone	Email_id
P001	Hills Publications	12, Park street, Atlanta	Georgia	7134019	h_pub@hills.com
P002	Sunshine Publishers Ltd.	45, Second street, Newark	New Jersey	6548909	Null
P003	Bright Publications	123, Main street, Honolulu	Hawai	6548142	bright@bp.com
P004	Paramount Publishing House	789, Oak street, New York	New York	5683452	param_house@param.com
P005	Wesley Publications	456, First street, Las Vegas	Nevada	9254834	Null

The schema and instance can be compared with a program written in a programming language. The database schema is similar to a variable declared along with the type description in a program. The variable contains a value at a given point of time. This value of a variable corresponds to an instance of a database schema.

A subschema is the applications programmer's view of the data within the database pertinent to the specific application. A subschema has access to those areas, set types, record types, data items, and data aggregates of interest in the pertinent application to which it was designed. Naturally, a software system usually has more than one programmer assigned and includes more than one application. This means there are usually many different subschemas for each schema.

The following are a few of the many reasons subschemas are used:

A subschema is a subset of the schema and inherits the same property that a schema has. The plan (or scheme) for a view is often called subschema. Subschema refers to an application programmer's (user's) view of the data item types and record types, which he or she uses. It gives the users a window through which he or she can view only that part of

Remember

This description is in terms of the names and characteristics of the data items, data aggregates, records, areas, and sets included in the database, and the relationships that exist and must be maintained between occurrences of those elements in the database.

the database, which is of interest to him. Therefore, different application programs can have different view of data.

Subschemas provide different views of the data to the user and the programmer, who do not need to know all the data contained in the entire database. Subschemas enhance security factors and prohibit data compromise. Subschemas aid the DBA while assuring data integrity. Each data item included in the subschema will be assigned a location in the user working area (UWA).The UWA is conceptually a loading and unloading zone, where all data provided by the DBMS in response to a CALL for data is delivered. It is also where all data to be picked up by the DBMS must be placed. Schema **data definition language** (DDL) the schema data definition language (DDL) is used for describing a database, which may be shared by many programs written in many languages.

Keyword

Data Definition Language (DDL) is a syntax similar to a computer programming language for defining data structures, especially database schemas.

Data Item: A data item is an occurrence of the smallest unit of named data. It is represented in a database by a value.

Data Aggregate: A data aggregate is an occurrence of a named collection of data items within a record.

There are two kinds-vectors and repeating groups. A vector is a one-dimensional sequence of data items, all of which have identical characteristics. A repeating group is a collection of data that occurs a number of times within a record occurrence collection may consist of data items, vectors repeating groups.

5.3 DATABASE MODELS

Keyword

Data Definition Language is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users.

The main objective of database system is to highlight only the essential features and to hide the storage and data organization details from the user. This is known as data abstraction. A database model provides the necessary means to achieve data abstraction. A database model or simply a data model is an abstract model that describes how the data is represented and used. A data model consists of a set of data structures and conceptual tools that is used to describe the structure (data types, relationships, and constraints) of a database.

A data model not only describes the structure of the data; it also defines a set of operations that can be performed on the data. A data model generally consists of data model theory, which is a formal description of how data may be structured

and used, and data model instance, which is a practical data model designed for a particular application. The process of applying a data model theory to create a data model instance is known as data modeling.

Depending on the concept they use to model the structure of the database, the data models are categorized into three types, namely, high-level or conceptual data models, representational or implementation data models and low-level or physical data models.

5.3.1 Conceptual Data Model

Conceptual data model describes the information used by an organization in a way that is independent of any implementation-level issues and details. The main advantage of conceptual data model is that it is independent of implementation details and hence, can be understood even by the end users having non-technical background. The most popular conceptual data model, that is, entity-relationship (E-R) model.

5.3.2 Representational Data Model

The representational or implementation data models hide some data storage details from the users; however, can be implemented directly on a computer system. Representational data models are used most frequently in all traditional commercial DBMS.

The various representational data models are:

Hierarchical Data Model

The hierarchical data model is the oldest type of data model, developed by IBM in 1968. This data model organizes the data in a tree-like structure, in which each child node (also known as dependents) can have only one parent node. The database based on the hierarchical data model comprises a set of records connected to one another through links. The link is an association between two or more records. The top of the tree structure consists of a single node that does not have any parent and is called the root node.

The root may have any number of dependents; each of these dependents may have any number of lower level dependents. Each child node can have only one parent node and a parent node can have any number of (many) child nodes. It, therefore, represents only one-to-one and one-to-many relationships. The collection of same type of records is known as a record type. Figure 2 shows the hierarchical model of Online Book database. It consists of three record types, namely, PUBLISHER, BOOK, and REVIEW. For simplicity, only few fields of each record type are shown. One complete record of each record type represents a node.

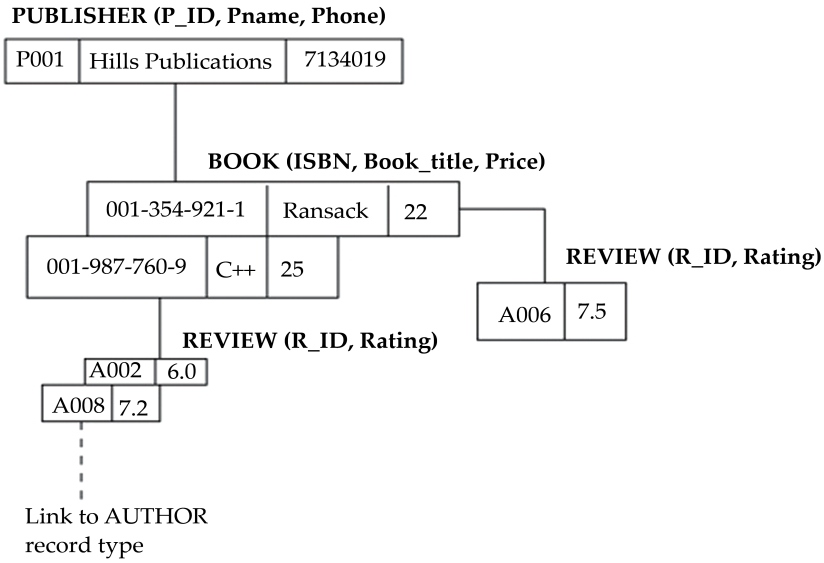


Figure 2: Hierarchical data model for Online Book database.

The main advantage of the hierarchical data model is that the data access is quite predictable in the structure and, therefore, both the retrieval and updates can be highly optimized by the DBMS. However, the main drawback of this model is that the links are 'hard coded' into the data structure, that is, the link is permanently established and cannot be modified. The hard coding makes the hierarchical model rigid. In addition, the physical links make it difficult to expand or modify the database and the changes require substantial redesigning efforts.

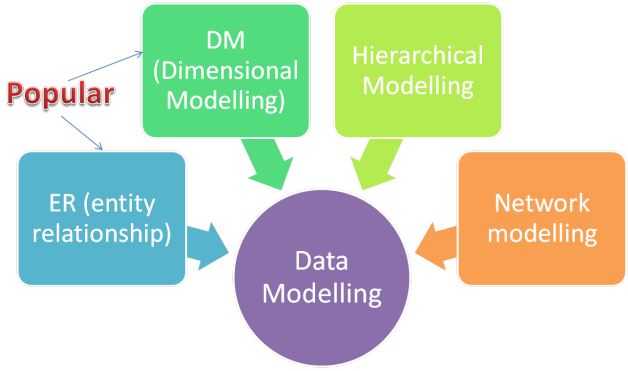


Figure 3: Data modeling.

Network Data Model

In a network model the data is also represented by a collection of records, and relationships among data are represented by links. However, the link in a network data model represents an association between precisely two records. Like hierarchical data model, each record of a particular record type represents a node. However, unlike hierarchical data model, all the nodes are linked to each other without any hierarchy. The main advantage of network data model is that a parent node can have many child nodes and a child can also have many parent nodes. Thus, the network model permits the modeling of many-to-many relationships in data. The main limitation of the network data model is that it can be quite complicated to maintain all the links and a single broken link can lead to problems in the database. In addition, since there are no restrictions on the number of relationships, the database design can become complex. Figure 4 shows the network model of Online Book database.

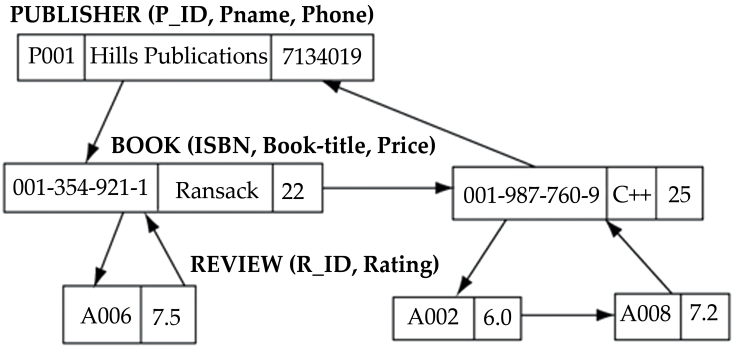


Figure 4: Network data model for online book database.

Relational Data Model

In the relational data model, unlike the hierarchical and network models, there are no physical links. All data is maintained in the form of tables (generally, known as relations) consisting of rows and columns. Each row (record) represents an entity and a column (field) represents an attribute of the entity. The relationship between the two tables is implemented through a common attribute in the tables and not by physical links or pointers. This makes the querying much easier in a relational database system than in the hierarchical or network database systems. Thus, the relational model has become more programmers friendly and much more dominant and popular in both industrial and academic scenarios. Oracles, Sybase, DB2, Ingres, and Informix, MS-SQL Server are few of the popular relational DBMSs.

Object-based Data Model

In the recent years, the object-oriented paradigm has been applied to database technology, creating two new data models known as object-oriented data model and object-relational data model. The object-oriented data model extends the concepts of object-oriented programming language with persistence, versioning, concurrency control, data recovery, security, and other database capabilities. On the other hand, the object-relational data model is an extension of relational data model. It combines the features of both the relational data model and object-oriented data model.

Did You Know?

The concept of the data definition language and its name was first introduced in relation to the Codasyl database model, where the schema of the database was written in a language syntax describing the records, fields, and sets of the user data model. Later it was used to refer to a subset of Structured Query Language (SQL) for declaring tables, columns, data types and constraints.

Semi Structured Data Model

Unlike other data models, where every data item of a particular type must have the same set of attributes, the semi structured data model allows individual data items of the same type to have different set of attributes. In semi structured data model, the information about the description of the data (schema) is contained within the data itself, which is sometimes called self-describing data. In such databases there is no clear separation between the data and the schema and thus, allowing data of any type.

Semi structured data model has recently emerged as an important topic of study for different reasons given as:

- There are data sources such as the Web, which is to be treated as databases; however, they cannot be constrained by a schema.
- The need of flexible format for data exchange between heterogeneous databases.
- To facilitate browsing of data.

Semi structured data model facilitates data exchange among heterogeneous data sources. It helps to discover new data easily and store it. It also facilitates querying the database without knowing the data types. However, it loses the data type information.

5.3.3 Physical Data Model

Physical data model describes the data in terms of a collection of files, indices, and other storage structures such as record formats, record ordering, and access paths. This model specifies how the database will be executed in a particular DBMS software such as Oracle, Sybase, etc., by taking into account the facilities and constraints of a given database management system. It also describes how the data is stored on disk and what access methods are available to it.

CASE STUDY

HOSPITAL MANAGEMENT SYSTEM

Aim: XYZ hospital is a multi specialty hospital that includes a number of departments, rooms, doctors, nurses, compounders, and other staff working in the hospital. Patients having different kinds of ailments come to the hospital and get checkup done from the concerned doctors. If required they are admitted in the hospital and discharged after treatment.

The aim of this case study is to design and develop a database for the hospital to maintain the records of various departments, rooms, and doctors in the hospital. It also maintains records of the regular patients, patients admitted in the hospital, the check up of patients done by the doctors, the patients that have been operated, and patients discharged from the hospital.

Description: In hospital, there are many departments like Orthopedic, Pathology, Emergency, Dental, Gynecology, Anesthetics, I.C.U., Blood Bank, Operation Theater, Laboratory, M.R.I., Neurology, Cardiology, Cancer Department, Corpse, etc. There is an OPD where patients come and get a card (that is, entry card of the patient) for check up from the concerned doctor. After making entry in the card, they go to the concerned doctor's room and the doctor checks up their ailments. According to the ailments, the doctor either prescribes medicine or admits the patient in the concerned department. The patient may choose either private or general room according to his/her need. But before getting admission in the hospital, the patient has to fulfill certain formalities of the hospital like room charges, etc. After the treatment is completed, the doctor discharges the patient. Before discharging from the hospital, the patient again has to complete certain formalities of the hospital like balance charges, test charges, operation charges (if any), blood charges, doctors' charges, etc

Next we talk about the doctors of the hospital. There are two types of the doctors in the hospital, namely, regular doctors and call on doctors. Regular doctors are those doctors who come to the hospital daily. Calls on doctors are those doctors who are called by the hospital if the concerned doctor is not available.

Table Description:

Following are the tables along with constraints used in Hospital Management database.

1. DEPARTMENT: This table consists of details about the various departments in the hospital. The information stored in this table includes department name, department location, and facilities available in that department.

   Constraint: Department name will be unique for each department.

2. ALL_DOCTORS: This table stores information about all the doctors working for the hospital and the departments they are associated with. Each doctor is given an identity number starting with DR or DC prefixes only.

   Constraint: Identity number is unique for each doctor and the corresponding department should exist in DEPARTMENT table.

3. DOC_REG: This table stores details of regular doctors working in the hospital. Doctors are referred to by their doctor number. This table also stores personal details of doctors like name, qualification, address, phone number, salary, date of joining, etc.

   Constraint: Doctor's number entered should contain DR only as a prefix and must exist in ALL_DOCTORS table.

4. DOC_ON_CALL: This table stores details of doctors called by hospital when additional doctors are required. Doctors are referred to by their doctor number. Other personal details like name, qualification, fees per call, payment due, address, phone number, etc., are also stored.

   Constraint: Doctor's number entered should contain DC only as a prefix and must exist in ALL_DOCTORS table.

5. PAT_ENTRY: The record in this table is created when any patient arrives in the hospital for a check up. When patient arrives, a patient number is generated which acts as a primary key. Other details like name, age, sex, address, city, phone number, entry date, name of the doctor referred to, diagnosis, and department name are also stored. After storing the necessary details patient is sent to the doctor for checkup.

   Constraint: Patient number should begin with prefix PT. Sex should be M or F only. Doctor's name and department referred must exist.

6. PAT_CHKUP: This table stores the details about the patients who get treatment from the doctor referred to. Details like patient number from patient entry table, doctor number, date of check up, diagnosis, and treatment are stored. One more field status is used to indicate whether patient is admitted, referred for operation or is a regular patient to the hospital. If patient is admitted, further details are stored in PAT_ADMIT table. If patient is referred for operation, the further details are stored in PAT_OPR table and if patient is a regular patient to the hospital, the further details are stored in PAT_REG table.

   Constraint: Patient number should exist in PAT_ENTRY table and it should be unique.

7. PAT_ADMIT: When patient is admitted, his/her related details are stored in this table. Information stored includes patient number, advance payment, mode of payment, room number, department, date of admission, initial condition,

diagnosis, treatment, number of the doctor under whom treatment is done, attendant name, etc.

Constraint: Patient number should exist in PAT_ENTRY table. Department, doctor number, room number must be valid.

8. PAT_DIS: An entry is made in this table whenever a patient gets discharged from the hospital. Each entry includes details like patient number, treatment given, treatment advice, payment made, mode of payment, date of discharge, etc.

   Constraint: Patient number should exist in PAT_ENTRY table.

9. PAT_REG: Details of regular patients are stored in this table. Information stored includes date of visit, diagnosis, treatment, medicine recommended, status of treatment, etc.

   Constraint: Patient number should exist in patient entry table. There can be multiple entries of one patient as patient might be visiting hospital repeatedly for check up and there will be entry for patient's each visit.

10. PAT_OPR: If patient is operated in the hospital, his/her details are stored in this table. Information stored includes patient number, date of admission, date of operation, number of the doctor who conducted the operation, number of the operation theater in which operation was carried out, type of operation, patient's condition before and after operation, treatment advice, etc.

    Constraint: Patient number should exist in PAT_ENTRY table. Department, doctor number should exist or should be valid.

11. ROOM_DETAILS: It contains details of all rooms in the hospital. The details stored in this table include room number, room type (general or private), status (whether occupied or not), if occupied, then patient number, patient name, charges per day, etc. Constraint: Room number should be unique. Room type can only be G or P and status can only be Y or N

SUMMARY

- Database design, as the name might suggest, is much like house A database is an organized collection of data stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques.
- The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database.
- A redundant relationship is one that is expressed more than once. Typically, you can remove one of the relationships without losing any important information.
- An index is essentially a sorted copy of one or more columns, with the values either in ascending or descending order. Adding an index allows users to find records more quickly.
- A view is simply a saved query on the data. They can usefully join data from multiple tables or else show part of a table.
- The Unified Modeling Language (UML) is another visual way of expressing complex systems created in an object-oriented language.
- The database schema is also known as intension of the database, and is specified while designing the database.
- A subschema is the applications programmer's view of the data within the database pertinent to the specific application. A subschema has access to those areas, set types, record types, data items, and data aggregates of interest in the pertinent application to which it was designed.
- A data item is an occurrence of the smallest unit of named data. It is represented in a database by a value.
- A data aggregate is an occurrence of a named collection of data items within a record.

KNOWLEDGE CHECK

1. Database is collection of ……..

a. None of these
b. Data
c. Modules
d. Programs

2. ……..is collection of interrelated data and set of program to access them.

a. Programming language
b. Database Management System
c. Database
d. Data Structure

3. DBMS should provide following feature(s) ……...

a. Authorized access
b. All of these
c. Safety of the information stored
d. Protect data from system crash

4. Before use of DBMS information was stored using ……..

a. Cloud Storage
b. Data System
c. File Management System
d. None of these

5. Which of the following is considered as DBMS?

a. All of these
b. Access
c. Oracle
d. Foxpro

6. **A Database Management System is a type of _____software.**
   a. It is a type of system software
   b. It is a kind of application software
   c. It is a kind of general software
   d. Both a and c
7. **A huge collection of the information or data accumulated form several different sources is known as _____:**
   a. Data Management
   b. Data Mining
   c. Data Warehouse
   d. Both b and c
8. **Which one of the following refers to the copies of the same data (or information) occupying the memory space at multiple places.**
   a. Data Repository
   b. Data Inconsistency
   c. Data Mining
   d. Data Redundancy
9. **The term "Data" refers to:**
   a. The electronic representation of the information(or data)
   b. Basic information
   c. Row Facts and figures
   d. Both a and c
10. **Which of the following commands is used to save any transaction permanently into the database?**
    a. Commit
    b. Rollback
    c. Savepoint
    d. None of the above

REVIEW QUESTIONS

1. What is second normal form?
2. Why database design is important?
3. What is database design?
4. What is normalization?
5. What is physical data model in DBMS?

Check Your Result

1. (b)	2. (b)	3. (b)	4. (c)	5. (a)
6. (a)	7. (c)	8. (d)	9. (c)	10. (a)

REFERENCES

1. An Introduction to Database Systems", C. J. Date, A. Kannan and S. Swamynathan, Pearson Education, Eighth Edition, 2009.
2. Aqda, M.F., Hamidi, F., & Rahimi, M. (2011). The comparative effect of computer-aided instruction and
3. Database System Concepts", Abraham Silberschatz, Henry F. Korth and S. Sudarshan, McGraw-Hill Education (Asia), Fifth Edition, 2006.
4. Database Systems Design, Implementation and Management", Peter Rob and Carlos Coronel, Thomson Learning-Course Technology, Seventh Edition, 2007.
5. Database Systems, Design, Implementation and Management", Carlos Cornol and Steven Morris, 12th edition, 2016.
6. Transactional Information Systems", Gerhard Weikum, Gottfried Vossen, Elsevier, ISBN 1-55860-508-8, 2001.

CHAPTER

6

PROGRAMMING WITH SQL

"Every successful application starts with a strong data model. Oracle SQL Developer Data Modeler not only provides a graphical way to develop data models, but an effective way to communicate existing data models to application developers."

—Mike Hichwa

LEARNING OBJECTIVES

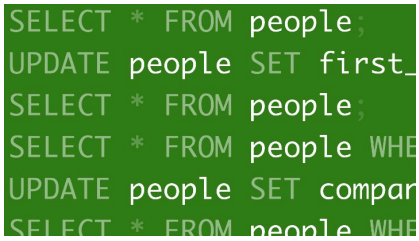
After studying this chapter, you will be able to:

1. Discuss about embedded SQL
2. Explain dynamic SQL
3. Understand SQL APIs

INTRODUCTION

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of

the SQL language. In addition to its role as an interactive data access language, SQL supports database access by application programs.



Programming with SQL describes how application programs use SQL for database access. It discusses the embedded SQL specified by the ANSI standard and used by IBM, Oracle, Ingres, Informix, and many other SQL-based DBMS products. It also describes the dynamic SQL interface that is used to build general-purpose database tables, such as report writers and database browsing programs. Finally, this describes the popular SQL APIs, including ODBC, the ISO-standard Call-Level Interface, and JDBC, the standard call-level interface for Java, as well as proprietary call-level interfaces such as Oracle's OCI API.

Keyword

Information system (IS) is an organized system for the collection, organization, storage and communication of information.

6.1 EMBEDDED SQL

Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.

Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.

Embedded SQL is not supported by all relational database management systems (RDBMS). Oracle DB and PostgreSQL provide embedded SQL support. MySQL, Sybase and SQL Server 2008 do not, although support was provided by earlier versions of SQL Server (2000 and 2005).

The C programming language is commonly used for embedded SQL implementation. For example, a commercial bank's **information system** (IS) has a front-end user interface created in the C language, and the IS interfaces with a back-end Oracle DB database. One of the front-end interface modules allows quick viewing and commission calculation for sales agents during specified periods. An inefficient approach to handling this process would be to store each commission value in a database table. However, a more effective solution is to calculate and return commission values based on unique user requests on specified dates. The application accomplishes this by embedding a SQL query within the C code, as follows:

SELECT 0.2*SALE_AMOUNT FROM TOTAL_SALES WHERE SALE_DATE='MM/DD'YYYY' AND AGENT_NO=xx

In this example, the SQL statement calculates and returns 20 percent of the sale amount from a TOTAL_SALES table, while the user is expected to input the SALE_DATE and AGENT_NO values. This SQL query is then inserted inline into the C code of the front-end module. The C code and SQL query work together to deliver seamless user results.

Remember

The embedded SQL statements are contained in a package that must be bound to the target database server.

6.1.1 Concepts for Embedding the SQL Statements

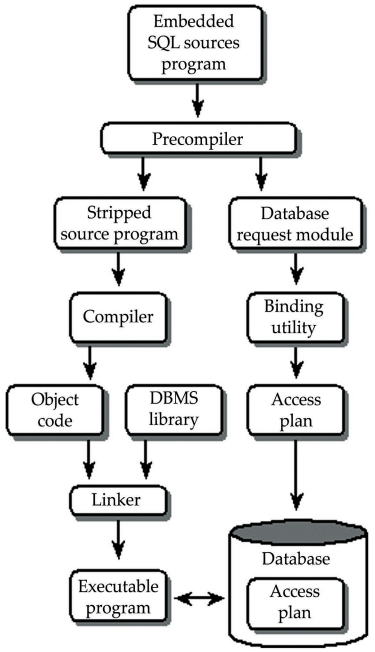
We can mix the SQL statements directly into general purpose programming language like C, Java or Pascal. There are some techniques to embed SQL statements in the programming languages.

- The programming language in which the SQL statements are embedded is called the host language. The SQL statements and host language statements make the source program which is fed to a SQL precompiler for processing the SQL statements
- The host programming languages variables can be referenced in the embedded SQL statements, which allows values calculated by the programs to be used by SQL statements.
- There are some special program variables which are used to assign null values to database columns. These program variables support the retrieval of null values from the database.

6.1.2 Embedded SQL Program Development

Since the embedded SQL is a mixture of SQL and programming language, so it cannot be fed directly to a general purpose programming language compiler. Actually the program execution is a multi-step which is as follows.

- First, the embedded SQL source code is fed to the SQL precompiler. The precompiler scans the program and processes the embedded SQL statements present in the code. There can be different precompilers for different type of programming languages.
- After processing the source code, the precompiler produces 2 files as its output. The first file contains the source program without embedded SQL statements and the second file contains all the embedded SQL statements used in the program.



- The first file prodiced by precompiler (that contains the source program) is fed to the compiler for the host programming language (like C compiler). The compiler processes the source code and produces object code as its output.
- Now the linker takes the object modules produced by the compiler and link them with various library routines and produces an executable program.
- The database request modules, produced by the precompiler (in steps) are submitted to a special BIND program. The BIND program examines the SQL

statements, parse them, validates them, optimizes them and finally produces an application plan for each statement. The result is a combined application plan for the entire program, that represents a DBMS-executable version of its embedded SQL statements. The BIND program stores the plan in the database, usually assigning it the name of the application program that has created it.

6.1.3 An Embedded SQL Example in C

Although the SQL statements can be embedded in any general purpose programming language, still we just take an example in C language so that a clear picture can be drawn. We just take an interactive SQL statement and see how it can be embedded in C language.

Increase the salary of teacher by 10% who are B.Tech

update teacher set salary=1.1*salary where qualification='B.Tech';

The embedded SQL program for above written SQL statement will be:

```
main()
        exec sql include sqlca;
          exec sql declare table teacher (tid char(6) not null,tname char(20),sex
char(1),age number(3),qualification char(7),salary number(7),city varchar(15));

        //Display a message to user
        printf("updating teacher salary who are B.Techn");
        //this code executes the SQL statement
       exec sql update teacher set salary=1.1*salary where qualification='B.Tech';
        printf(update done");

        exit();
}
```

Explanation

Although the above shown program is very easy to understand, still we would like to discuss some very basic features of embedded SQL.

- The embedded SQL statement can be written in any case (lower or upper). Although we should follow the convention of that programming language in which we are embedding the SQL statements. For e.g., COBOL and FORTRAN are written in upper case so, the SQL statements are also written in upper

case, while in C, the SQL statements are written in lower case as shown in above program.

- Each embedded SQL statement begins with an introducer which indicates that it is a SQL statement. For most of the programming language *EXEC SQL* is used as an introducer.
- Each embedded SQL statement ends with a terminator. There can be different terminators for different programming languages. For example, there is *END EXEC* for COBOL and a semicolon (;) for C.
- The *DECLARE TABLE* statement is used to declare a table. With the use of *DECLARE TABLE* statement our program specifies the column and **data type** explicitly.
- When we type SQL statement, we may make an error. This error is displayed by the interactive SQL program and we are prompted to type a new statement. There can be two types of errors: compile time and runtime.

Keyword

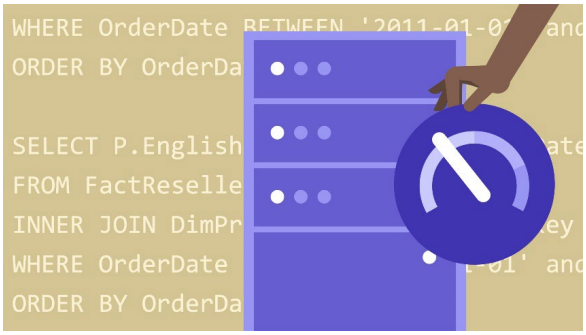
Data type or simply type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data.

6.1.4 Error Handling with SQL Code

In this scheme the DBMS communicates the status to the embedded SQL program through an area of program storage called the SQL communication are or SQLCA. The SQLCA is a data structure that contains the error variables and the status indicators. By examining the SQLCA, the application program can determine the success or failure of its embedded SQL statements and can take actions accordingly.

6.2 DYNAMIC SQL

Dynamic SQL refers to SQL statements that are constructed and executed at run-time. Dynamic is the opposite of static. *Static SQL* refers to SQL statements that are fixed at the time a program is compiled. *Dynamic PL/SQL* refers to entire PL/SQL blocks of code that are constructed dynamically, then compiled and executed.



Ever since Oracle7 Release 1, we PL/SQL developers have been able to use the built-in DBMS_SQL package to execute dynamic SQL . In Oracle8*i* Database, we were given a second option for executing dynamically constructed SQL statements: *native dynamic SQL* (NDS). NDS is a *native* part of the PL/SQL language; it is much easier to use than DBMS_SQL, and, for many applications, it will execute more efficiently.

6.2.1 Programming with Dynamic SQL

For the sake of consistency, this section discusses dynamic SQL mainly from the perspective of PL/SQL. To process most dynamic SQL statements, you use the EXECUTE IMMEDIATE statement. To process a multi-row query in a PL/SQL procedure, you use the OPEN-FOR, FETCH, and CLOSE statements.

Oracle Database enables you to implement dynamic SQL in a PL/SQL application in the following ways:

- Using native dynamic SQL, which involves placing dynamic SQL statements directly into PL/SQL blocks
- Calling procedures in the DBMS_SQL package

Although this section discusses PL/SQL support for dynamic SQL, you can call dynamic SQL from other languages:

- If you use C/C++, you can call dynamic SQL with the Oracle Call Interface (OCI), or you can use the Pro*C/C++ precompiler to add dynamic SQL extensions to your C code.
- If you use COBOL, you can use the Pro*COBOL precompiler to add dynamic SQL extensions to your COBOL code.

Did You Know?

You can use dynamic SQL to build a SQL statement that optimizes execution by concatenating hints into a dynamic SQL statement. This technique enables you change the hints based on your current database statistics without recompiling.

- If you use Java, you can develop applications that use dynamic SQL with JDBC.

If you have a program that uses OCI, Pro*C/C++, or Pro*COBOL to execute dynamic SQL, consider switching to native dynamic SQL inside PL/SQL stored procedures and functions. The network round-trips required to perform dynamic SQL operations from client-side applications might hurt performance. Stored procedures can reside on the server, eliminating network overhead. You can call the PL/SQL stored procedures and stored functions from the OCI, Pro*C/C++, or Pro*COBOL application.

Example :Using SELECT . . . TABLE in Dynamic SQL

```
-- Create an object t_emp and a datatype t_emplist as a table of type t_emp
CREATE TYPE t_emp AS OBJECT (id NUMBER, name VARCHAR2(20))
/
CREATE TYPE t_emplist AS TABLE OF t_emp
/
-- Create a table with a nested table of type t_emplist
CREATE TABLE dept_new (id NUMBER, emps t_emplist)
  NESTED TABLE emps STORE AS emp_table;
-- Populate the dept_new table with data
INSERT INTO dept_new VALUES
(
  10,
  t_emplist
  (
    t_emp(1, 'SCOTT'),
    t_emp(2, 'BRUCE')
  )
);
-- Write a PL/SQL block that queries table dept_new and nested table emps
-- SELECT ... FROM ... TABLE is not allowed in static SQL in PL/SQL
DECLARE
  v_deptid   NUMBER;
  v_ename    VARCHAR2(20);
BEGIN
```

```
  EXECUTE IMMEDIATE 'SELECT d.id, e.name
                     FROM dept_new d, TABLE(d.emps) e
                     WHERE e.id = 1'
  INTO v_deptid, v_ename;
END;
```

6.2.2 Writing Dynamic SQL

Coding effective routines that provide performance and simplification of tasks requires that we understand the intent of the tool. If misused, any tool can be a hindrance, but when applied correctly to a problem for which it was intended, Dynamics SQL really shines. In this section, we will look at how SQL Server allows us to execute Dynamic SQL, and some techniques for writing effective code.

Keyword

Local variable is a variable which is either a variable declared within the function or is an argument passed to a function.

EXEC

You have seen the EXECUTE command used to run stored procedures, but it can also be used to execute a character string. For example the simple statement to list sales by title can be called like this:

```
EXEC ('SELECT title_id, count(*) FROM sales GROUP BY title_id')
```

We are not limited to executing static strings using the EXEC command. We can generate a SQL statement based on the current environment and execute that statement as well. If we need to summarize data by the frequency of values on a particular column we could declare a **local variable**, set the value equal to the command we want to run. In this case we use concatenation to build the command string, and we declare a variable to hold the name of the column to group by:

```
DECLARE @col VARCHAR (50)
DECLARE @cmd VARCHAR(4000)
SET @col = 'stor_id'
SET @cmd = 'SELECT '+@col+', count(*) FROM sales GROUP BY '+@col
EXEC (@cmd)
```

This could be run from Query Analyzer as its own batch, or it could be part of a larger stored procedure. Using variables to hold names of columns or tables that may need to be changed simplifies support and maintenance of the code.

One consideration to keep in mind is that every time the database processes an EXEC command it treats the statement as a new command that needs to be treated in its own scope. This means that any variables declared within the command string being run are not visible to the calling batch, and likewise variables that are in the scope of the calling batch are not visible within the EXEC'd command.

The statement below will result in an error because the context of the variable @ table is the calling batch of statements, and there is no table with the name "@table" in the database.

```
DECLARE @table VARCHAR(50)
SET @table = 'authors'
EXEC ('SELECT * FROM master..sysobjects WHERE name = @table') -- BOOM!
```

If you change the database context with the USE command the effects do not last beyond the end of the statement. This is important to keep in mind when you are working with multiple databases and don't fully qualify the tables with the database. owner.tablename syntax.

```
Use pubs
go
declare @cmd varchar (4000)
set @cmd = 'EXEC spCurrDB'
set @cmd = 'select ''The current database is: ['''+d.name+''']'''
+ ' from master..sysdatabases d, master..sysprocesses p '
+ ' where p.spid = @@SPID and p.dbid = d.dbid '
EXEC (@cmd)
EXEC (N'Use master;'+@cmd)
EXEC (@cmd)
```

This example will return the name of the current database by using the @@SPID which returns the current process id and then joining the system tables sysprocesses and sysdatabases on the database id column (dbid) and then filtering the results to the row that matches our id. When you run it the first EXEC shows current context to be pubs, the second master and the third is back to pubs. When the second EXEC runs, it changes the database context just for the duration of that EXEC call, and doesn't

change the calling batch's context. The database engine treats each EXEC as separate batches, which have no knowledge of the other.

If the first three characters following the EXEC statement are sp_, it assumes that you are running a system stored procedure and will search the master **catalog** of procedures before it looks at the current database. For that reason, it is a good idea to use a different naming standard for your own stored procedures. The performance gain might be small, but why waste resources if you don't have to?

Keyword

Catalog can be any compilation of items organized in a systematic manner.

sp_executesql

Using sp_executesql to run dynamic statements gives us a couple advantages over EXEC that are worth noting. The first is that while both evaluate the SQL statement at the point of execution, sp_executesql will store and potentially reuse execution plans while EXEC does not. The other benefit is that sp_executesql supports parameter substitution and allows you to better integrate the statements into your program.

The calling syntax for sp_executesql is as follows:

sp_executesql <@stmt> [<@param1 data_type>,<@param2 data_type>, ...]

The @stmt parameter is a Unicode string containing valid SQL commands, and the parameters are specified with a name and type. We can specify the parameters for both input and output. In this example we are going to return as output the count of books where the author is contained in the variable au_name. The output type @retType is passed as the second parameter to sp_executesql, and the variable @retVal that will be set to the returned value is passed as the third parameter.

```
declare @cmd nvarchar(4000)
declare @retType nvarchar(50)
declare @retVal nvarchar(20)

declare @au_name varchar(50)
set @@au_name = 'Ringer'
set @retType = N'@cnt varchar(20) OUTPUT'
set @cmd = N'SELECT @cnt = convert(varchar(20), count(*)) '
```

```
+ ' from titles t, titleauthor ta, authors a, sales s '
+ ' where a.au_id = ta.au_id '
+ ' and ta.title_id = t.title_id '
+ ' and s.title_id = t.title_id '
+ ' and a.au_lname like ''' + @@au_name + N'''
exec sp_executesql @cmd, @retType, @retVal OUTPUT
select @retVal
```

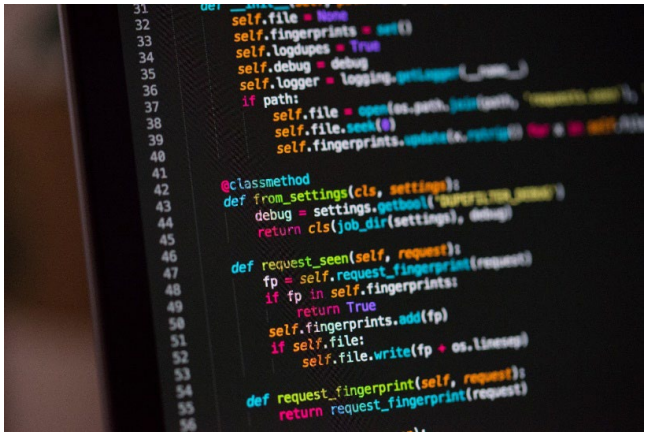
Remember

When placing your Dynamic SQL code into production (typically in stored procedures), be careful about concatenating alphanumeric parameters directly because .of SQL injection

6.3 SQL APIS

An application program interface (API) is code that allows two software programs to communicate with each other.

The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application being called.



6.3.1 How APIs Work

APIs are made up of two related elements. The first is a specification that describes how information is exchanged between programs, done in the form of a request for processing and a return of the necessary data. The second is a software interface written to that specification and published in some way for use.

The software that wants to access the features and capabilities of the API is said to call it, and the software that creates the API is said to publish it.

A database application programming interface (database API) is a library of functions to carry out database operations. Each database management system provides a library of its own. For example, Oracle has a library called Oracle Call Level Interface (Oracle CLI). This is actually the library used in the calls generated by the embedded SQL pre-compiler. Database management system specific libraries are called native APIs.

There are also libraries, for example ODBC (Microsoft Open Database Connection) and JDBC (for Java), that are independent of database management systems. These libraries make it possible to use various database management systems, even within one program. On the programmers point of view these libraries provide a common interface for database programming. However, to use a certain type of database, a *driver* for that particular type is needed. For example, an Oracle driver is needed in order to use Oracle databases. In the following we outline the use of database in **Java** programming language using JDBC -API.

Keyword

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

JDBC is a class library that provides classes and methods for using a database. The kernel classes of the library are

- DriverManager,
- Connection,
- Statement and PreparedStatement, and
- ResultSet.

DriverManager

DriverManager class provides services to attach a database management system specific driver to the application. This driver is a class library that should be obtained from the databases management system supplier. It is necessary in order to be able to communicate with the database management system. DriverManager has the method 'registerDriver' for making the attachment. Database drivers are able to register themselves to the DriverManager. Thus it's usually enough to load the driver into the Java engine.

Connection

Connection is a class devoted for taking care of a database session. It provides services for logging in and out as a database user and for building up and controlling database transactions. All operations on the database rely on the existence of a connection. DriverManager establishes a connection by creating an instance of the Connection class. For example the following program, first loads the driver and then establishes a connection to the Oracle database *test* in computer *bodbacka.cs.helsinki.fi* via port *1521* on the account *scott* and password *tiger*.

```
final String dbDriver=”oracle.jdbc.OracleDriver”;
final String dbServer= “jdbc:oracle:thin:@bodbacka.cs.helsinki.fi:1521:test”;
final String dbUser=”scott”;
final String dbPassword)”tiger”;

try{
        Class.forName(dbDriver);                    // load driver
} catch (ClassNotFoundException e) {
          out.println(“Couldn’t find driver “+dbDriver);
          return null;
}
Connection con=null;
try {
       con = DriverManager.getConnection(dbServer,dbUser,dbPassword);
} catch (SQLException se) {
          out.println(“Couldn\’t get connection to “+dbServer+ “ for “+ dbUser+”<br>”);
          out.println(se.getMessage());
```

Keyword

Database server is a server which houses a database application that provides database services to other computer programs or to computers, as defined by the client–server model.

A connection should be closed (using close-method) when it’s no longer needed. Closing a connection releases resources in both the client workstation and in the **database server**.

Statement

Statement class provides the mechanism for passing operation requests to the database server and returning their results to the application. It provides services like

- executeQuery and
- executeUpdate.

A connection instance is needed for creating statements.

ResultSet

A ResultSet instance contains the answer for a query and provides means to get the data out of the answer for use in the program. It also provides information about the answer. Instances of class ResultSet are created by Statement.executeQuery method. The actual query to be executed is given as a parameter to this method.

```
String eQuery = "select EMPNO, ENAME, JOB, "+
                "MGR, to_char(HIREDATE,'DD.MM.YYYY') HIREDATE, SAL, "+
                "COMM, DEPTNO "+
                "From EMP "+
                "ORDER BY ENAME ";
    Statement stmt= con.createStatement();
    ResultSet rs= stmt.executeQuery(equery);
```

ResultSet provides methods for processing the answer of the query. Boolean function next() activates the next row of the answer table. When it is called the first time it activates the first row. It returns the value 'false' when there are no more rows to activate. Otherwise it returns true.

Originally ResultSet did not provide any means for proceeding in reverse direction. It could be used only for one way forward pass through the answer rows. However, new versions implement a technique known as *scroll cursor*. This makes also reverse proceeding possible.

ResultSet provides a collection of data type specific functions, for example, getString, getBoolean, getInt, getDate, etc. to retrieve data from the active row. These functions use either the name of the column or the sequence number of the column as their parameter.

```
String name= rs.getString("ename"); // column ename
Int pno= rs.getInt(1); // first column
```

The get-functions are able to perform some data type conversions, for example, from string to integer or vice versa. If the conversion fails, an exception (SQLException) is raised. Similar exception is raised also in other error conditions. Each database

operation may result in some type of error. The program should be constructed so that it catches the exceptions and provides the error messages.

SQL-queries may return NULL-values. Some get-operations, for example getString, return Java nulls for null values, but some return real values, for example, getInt returns a zero (0). Method wasNull may be used for checking whether the returned value was null.

The answer of a query is processed within a loop in a program. The following program prints an employee list as an HTML-table and finds out the biggest employee number.

```
// list the employees
  out.println("<H2>Current employees</H2>");
  out.println("<table border=1>"+
        "<tr>"+
     "<th "+HDC+">NAME</th><th "+HDC+">EMPNO</TH><TH "+HDC+">JOB</th>"+
     "<TH "+HDC+">MGR</TH><TH "+HDC+">HIREDATE</TH><TH "+HDC+">SAL</
TH><TH "+HDC+
        ">COMM</TH><TH "+HDC+">DEPTNO"+
        "</TH></TR>");

   int bigNo=0;  // the biggest empno
   int curNo=0;
   try {
        stmt = con.createStatement();
        rs = stmt.executeQuery(eQuery);
        // list the employees
        while(rs.next()) {
           out.println("<TR>");
           out.println("<TD>"+rs.getString("ENAME")+"</TD>");
           curNo= rs.getInt("EMPNO");
           if (curNo>bigNo)
              bigNo=curNo;
           out.println("<TD>"+curNo+"</TD>");
           out.println("<TD>"+rs.getString("JOB")+"</TD>");
           out.println("<TD>"+rs.getString("MGR")+"</TD>");
           out.println("<TD>"+rs.getString("HIREDATE")+"</TD>");
```

```
            out.println("<TD>"+rs.getString("SAL")+"</TD>");
            out.println("<TD>"+rs.getString("COMM")+"</TD>");
            out.println("<TD>"+rs.getString("DEPTNO")+"</TD>");
            out.println("</TR>");
        }
        out.println("</table><p><hr>");
    } catch (SQLException ex) { /* report about error */ }
```

Operations that change the contents or the structure of the database are executed using the method Statement.executeUpdate. For example,

```
stmt.executeUpdate("update emp " +
    "set sal= sal + 10000 " +
    " where ename= 'Laine'");
```

gives a small rise of salary to an employee.

PreparedStatement

Class PreparedStatement is used instead of Statement when we need parameterized operations. Above we have discussed about simple cases where the requested database operation is directly given as the parameter of the execute-methods. Programs often use database operations that differ only by some constant values provided by the user. For example, the user may provide names of persons and a database query is made for each of them to provide information about this person. We may, of course, programmatically build up a query for each person, but it is usually more efficient and safe to use parameterized queries instead.

Parameterized queries are prepared for use, i.e. compiled to be executed repeatedly. JDBC contains the class PreparedStatement that is a subclass of Statement. This class provides the services for working with parametrized database operations. A parameterized SQL-operation is given as a parameter to the function Connection. prepareStatement that creates an instance of class PreparedStatement.

```
PreparedStatement pst = con.prepareStatement(
    "select ename,hiredate,sal"+
    "from emp"+
    "where ename like ?" );
```

A question mark (?) in the SQL-operation indicates a parameter. Parameters must be assigned values before the operation is executed. PreparedStatement provides data

type specific set-functions for assigning these values (setInt, setString, setDate, etc). Set-functions have two parameters: the sequence number of the parameter (question mark) and the value to be assigned for the parameter. For example,

```
pst.setString(1,"Smi%;%");
```

assigns the value 'Smi%' to the first parameter of the query in pst. so that when executed the query will be

```
select ename,hiredate,sal
from emp
where ename like 'Smi%'
```

The operation is executed using executeQuery or executeUpdate methods of PreparedStatetement. These methods do not require parameters.

Remember

To connect to a database, a program must use a library specific to that database manager.

It is also possible to use dynamic SQL. This is used, for example, when the programmer does not know what kind of answer is obtained as the result of a query (what are the columns in the answer). To find this out, the programmer may use functions that provide information about the answer (metadata) and then use this information for retrieving data out of the answer.

6.3.2 Three Basic Types of APIs

APIs take three basic forms: local, web-like and program-like.

Local APIs are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local API form.

Web APIs are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

Program APIs are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. **Service oriented**

architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are program APIs.

APIs take three basic forms: local, web-like and program-like. Here's a look at each type.

Keyword

Service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.

6.3.3 Why API Design Matters

Traditionally the applications that publish APIs have to be written in a programming language, but because APIs are increasingly generalized, additional validation of an API's structure is important.

Good API design is critical for successful API use, and software architects spend considerable time reviewing all the possible applications of an API and the most logical way for it to be used.

The data structures and parameter values are of particular importance because they must match between the caller of an API and its publisher.

REST and the Web

Although applications that call APIs have traditionally been written in programming languages, the internet and the cloud are changing that. Web APIs can be called through any programming languages, but can also be accessed by webpages created in HTML or application generator tools.

The increased role the web plays in our lives and business activities has resulted in an explosion in the REST model and the use of simple programming tools, or even no programming at all, for API access.

API Examples in the Developer Community

Operating systems and middleware tools expose their features through collections of APIs usually called "toolkits," and two different sets of tools that support the same API specifications are interchangeable to programmers, which is the basis for compatibility and interoperability claims. Microsoft's .NET API specifications are the basis for an open source Linux equivalent middleware package now supported by Microsoft, for example.

The internet is currently the primary driver for APIs, and companies like Facebook, Google and Yahoo publish APIs to encourage developers to build on their capabilities. These APIs have given us everything from new internet features that browse the sites of other services, to mobile device apps that offer easy access to web resources.

New features, such as content delivery, augmented reality and novel applications of wearable technology, are created in large part though these APIs.

APIs Trends in the Cloud

Cloud computing introduces new capabilities in dividing software into reusable components, connecting components to requests and scaling the number of copies of software as demand changes.

These cloud capabilities have already begun to shift the focus of APIs from simple RPC-programmer-centric models to RESTful web-centric models, and even to what is called "functional programming" or "lambda models" of services that can be instantly scaled as needed in the cloud.

APIs as Services

The trend to think of APIs as representing general resources has changed terminology. Whereas APIs are expected to be used as a general tool by many applications and users, they are said to be services, and will normally require more controlled development and deployment. SOA and microservices are examples of service APIs. Services are the hottest trend in APIs, to the point where it's possible that all APIs in the future will be seen as representing services.

API Testing

Like all software, APIs have to be tested. The purpose of testing is validation of the published APIs against the specifications, which users of those APIs will use in formatting their requests.

This testing is usually done as a part of application lifecycle management (ALM), both for the software that publishes the APIs and for all the software that uses them. APIs also have to be tested in their published form to ensure that they can be accessed properly.

API Management

API management is a step beyond what's normally associated with software development. It's the set of activities associated with publishing the API for use, making it possible for users to find it and its specifications and regulating access to the API based on owner-defined permissions or policies.

CASE STUDY

ORACLE9I IN ACTION: ORACLE E-BUSINESS SUITE

The database component of the E-Business Suite has about 46 thousand PL/SQL objects, due mainly to packages and their bodies. There are about 23 thousand packages. (This number is growing: it was 24.5 at the last count!). This corresponds to about 20 million lines of PL/SQL source code The performance improvement due to Native Compilation was investigated using the standard concurrent programs Order Import, Payroll, and Auto Invoice.

Hardware Specification

The testing was purposely carried out on a lower-end system in order to determine if the run-time performance improvement justified the increase in compilation time.

- Server Class: Sun Ultra E450
- CPU Speed: 296 MHz
- Number of CPUs: 4
- Main Memory: 4 GB
- Operating System: Solaris 2.6 with Patch 105181-33

On this system, the recompilation of the whole database took approx. 22hr for Native and approx. 8hr for Interpreted.

Software Versions and Configuration

- Oracle Applications 11i Release 11.5.7
- Oracle9i Release 2, Version 9.2.0.1 (production release)
- Sun Workshop 6, Update 1
- Sun Workshop C Compiler Version 5.2

The whole database was compiled Native, producing about 46 thousand .so files on a single directory (the plsql_native_library_subdir_count initialization parameter was set to zero).

Results

Order Import is a PL/SQL intensive program. It was tested through the entire order to cash cycle. Its performance is characterized by the number of order lines processed per hour. Payroll consists mainly of a PRO*C client. However, the PRO*C client

makes calls to PL/SQL server side packages including fast formulas. Its performance is characterized by the number of assignments processed per hour. Auto Invoice again consists mainly of a PRO*C client. Again, the PRO*C client makes calls to PL/SQL server side packages including the tax engine and auto accounting. Its performance is characterized by the number of invoice lines processed per hour.

Table 1: Throughput in items per hour

	Interpreted	Native	Improvement
Order Import	3,015	3,983	32%
Payroll	26,105	27,342	5%
Auto Invoice	1,007,633	1,102,088	9%

The average CPU utilization was measure during the tests. It was 94% for the Interpreted case, and 89% for the Native case. The reduced CPU utilization in the Native case improves performance and scalability. In the Interpreted case, CPU utilization often reached 100%, and was continuously sporadic. In the Native case it was much more uniform and stable.

E-Business Suite Case Study: Conclusion

The overall performance improvement provided by Native PL/SQL Compilation varies from module to module depending on the amount of PL/SQL processing within the module. In some of the modules such as Order Import, a 30% performance improvement was observed while other modules resulted in less gain. Native Compilation is transparent to Oracle Applications including patching. When patches are installed, the PL/SQL unit will automatically be compiled as a native unit.

The results of this performance study demonstrate the value of the PL/SQL Native Compilation feature. Oracle Corporation recommends it to customers to improve performance and scalability of their E-Business Suite environment.

SUMMARY

- SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language.
- Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.
- Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.
- Dynamic SQL refers to SQL statements that are constructed and executed at run-time. Dynamic is the opposite of static. Static SQL refers to SQL statements that are fixed at the time a program is compiled. Dynamic PL/SQL refers to entire PL/SQL blocks of code that are constructed dynamically, then compiled and executed.
- An application program interface (API) is code that allows two software programs to communicate with each other.
- The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application being called.
- DriverManager class provides services to attach a database management system specific driver to the application. This driver is a class library that should be obtained from the databases management system supplier. It is necessary in order to be able to communicate with the database management system.
- Connection is a class devoted for taking care of a database session. It provides services for logging in and out as a database user and for building up and controlling database transactions.
- API management is a step beyond what's normally associated with software development. It's the set of activities associated with publishing the API for use, making it possible for users to find it and its specifications and regulating access to the API based on owner-defined permissions or policies.

KNOWLEDGE CHECK

1. **In database management systems, record which contains all data regarding tuples of database is called**
   a. statement record
   b. environment record
   c. description record
   d. connection record
2. **Type of iterator which is used to list types of attributes that are included in query result is called**
   a. positional iterator
   b. named iterator
   c. unnamed iterator
   d. non-positioned iterator
3. **Variable in DBMS used to communicate error conditions between database management system and program are**
   a. SQLCODE
   b. SQLSECTION
   c. SQLSTATE
   d. both b and c
4. **In database management system, if cursor is to be moved to next rows in queries result then command is classified as**
   a. OPEN CURSOR command
   b. FETCH command
   c. UPDATE command
   d. CLOSE CURSOR command
5. **General purpose programming languages such as COBOL and ADA is classified as**
   a. server language
   b. client language
   c. host language
   d. referential language

6. **The technique in which source program are replaced by database management generated code is classified as**
   a. referential SQL
   b. embedded SQL
   c. interpreted SQL
   d. embossed SQL
7. **The database interface in which the data is directly entered into the monitor is classified as**
   a. interactive interface
   b. direct interface
   c. monitored interface
   d. command interface
8. **The device which scans program code and extract them from database management system for processing is classified as**
   a. foreign processor
   b. secondary processor
   c. preprocessor
   d. primary processor
9. **Programming languages normally operate on a**
   a. Instances
   b. Variable
   c. Tuple
   d. Attributes
10. **Which of the following is used to access the database server at the time of executing the program and get the data from the server accordingly?**
    a. Embedded SQL
    b. Dynamic SQL
    c. SQL declarations
    d. SQL data analysis

REVIEW QUESTIONS

1. Differentiate between embedded SQL and dynamic SQL.
2. What is an ALIAS command?
3. Does SQL support programming language features?
4. Name different types of case manipulation functions available in SQL.
5. What is the SQL CASE statement used for? Explain with an example?
6. List and explain the different types of JOIN clauses supported in ANSI-standard SQL.

Check Your Result

1. (c)	2. (a)	3. (d)	4. (b)	5. (c)
6. (b)	7. (a)	8. (c)	9. (b)	10. (b)

REFERENCES

1. Discusses SQL history and syntax, and describes the addition of *INTERSECT* and *EXCEPT* constructs into PostgreSQL. Prepared as a Master's Thesis with the support of O. Univ. Prof. Dr. Georg Gottlob and Univ. Ass. Mag. Katrin Seyr at Vienna University of Technology.
2. http://zetcode.com/db/mysqlc/
3. https://searchmicroservices.techtarget.com/definition/application-program-interface-API
4. https://www.cs.helsinki.fi/u/laine/tikas/material/jdbc.html
5. https://www.thecrazyprogrammer.com/2014/11/embedded-sql-static-sql-in-dbms.html
6. *SQL Server 2000 Design and T-SQL Programming* (McGraw-Hill Professional, 2000)
7. Zelaine Fong, *The design and implementation of the POSTGRES query optimizer*, University of California, Berkeley, Computer Science Department.

CHAPTER

7

SQL SECURITY

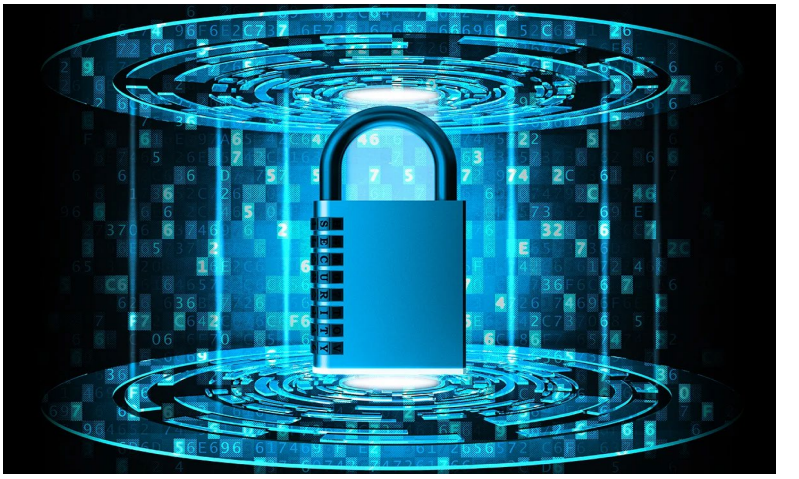
"Securing a computer system has traditionally been a battle of wits: the penetrator tries to find the holes, and the designer tries to close them."

—Gosser

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the SQL security concept
2. Learn about the SQL injection (SQLi)



INTRODUCTION

Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset, such as a person, dwelling, community, nation, or organization.

When you entrust your data to a database management system, the security, of the stored data is a major concern. Security is especially important in an SQL-based DBMS because interactive SQL makes database access very easy. The security requirements of a typical production database are many and varied:

- The data in any given table should be accessible to some users, but access by other users should be prevented.
- Some users should be allowed to update data in a particular table; others should be allowed only to retrieve data.
- For some tables, access should be restricted on a column-by-column basis.
- Some users should be denied interactive SQL access to a table but should be allowed to use application programs that update the table.

7.1 SQL SECURITY CONCEPT

Implementing a security scheme and enforcing security restrictions are the responsibility of the DBMS software. The SQL language defines an overall framework for database security, and SQL statements are used to specify security restrictions. The SQL security scheme is based on three central concepts:

Users. The actors in the database. Each time the DBMS retrieves, inserts, deletes, or updates data, it does so on behalf of some user. The DBMS permits or prohibits the action depending on which user is making the request.

Database objects. The items to which SQL security protection can be applied. Security is usually applied to tables and views, but other objects such as forms, application programs, and entire databases can also be protected. Most users will have permission to use certain database objects but will be prohibited from using others.

Privileges. The actions that a user is permitted to carry out for a given database object. A user may have permission to SELECT and INSERT rows in a certain table, for example, but may lack permission to DELETE or UPDATE rows of the table. A different user may have a different set of privileges.

Figure 1 shows how these security concepts might be used in a security scheme for the sample database. To establish a security scheme for a database, you use the SQL GRANT statement to specify which users have which privileges on which database objects. For example, here is a GRANT statement that lets Sam Clark retrieve and insert data in the OFFICES table of the sample database:

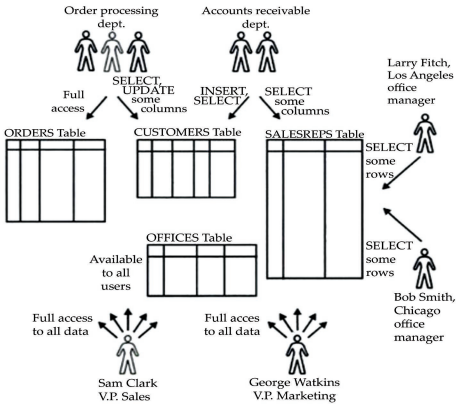


Figure 1. A security scheme for the sample database.

Let Sam Clark retrieve and insert data in the OFFICES table.

```
GRANT SELECT, INSERT
    ON OFFICES
    TO SAM
```

The GRANT statement specifies a combination of a user-id (SAM), an object (the OFFICES table), and privileges (SELECT and INSERT). Once granted, the privileges can be rescinded later with this REVOKE statement:

Take away the privileges granted earlier to Sam Clark.

```
REVOKE SELECT, INSERT
        ON OFFICES
        FROM SAM
```

7.1.1 User Ids

User ID is a feature in Matomo (Piwik) that lets you connect together a given user's data collected from multiple devices and multiple browsers. In this guide you will learn how User ID works and the steps a developer must take to implement User ID on your website and/or app. As a result, when your users connect to your website or app on their smartphone, tablet and their laptop then Matomo will be able to connect together these visits and report them under the same unique use.

Keyword

Database is a collection of information that is organized so that it can be easily accessed, managed and updated. Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information.

Each user of a SQL-based **database** is typically assigned a user-id, a short name that identifies the user to the DBMS software. The user-id is at the heart of SQL security. Every SQL statement executed by the DBMS is carried out on behalf of a specific user-id. The user-id determines whether the statement will be permitted or prohibited by the DBMS. In a production database, user-ids are assigned by the database administrator. A personal computer database may have only a single user-id, identifying the user who created and who owns the database. In special-purpose databases (for example, those designed to be embedded within an application or a special-purpose system), there may be no need for the additional overhead associated with SQL security. These databases typically operate as if there were a single user-id. In practice, the restrictions on the names that can be chosen as user-ids vary from implementation to implementation. The SQL1 standard permitted user-ids of up to 18 characters and required them to be valid SQL names. In some mainframe DBMS systems, user-ids may have no more than eight characters. In Sybase and SQL Server, user-ids may have up to 30 characters. If portability is a concern, it's best to limit user-ids to eight or fewer characters. Figure 2 shows various users who need access to the sample database and typical user-ids assigned to them.

Remember

All of the users in the order-processing department can be assigned the same user-id because they are to have identical privileges in the database.

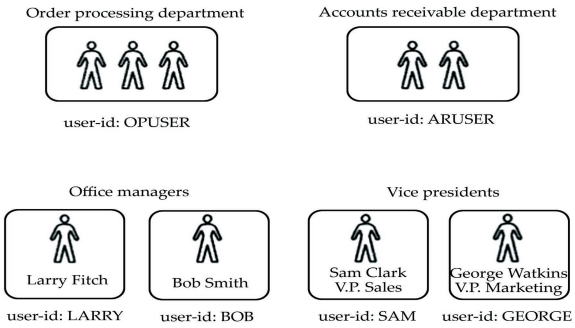


Figure 2. User Ids assignments for sample database.

The ANSI/ISO SQL standard uses the term authorization-id instead of user-id, and you will occasionally find this term used in other SQL documentation. Technically, authorization-id is a more accurate term because the role of the ID is to determine authorization or privileges in the database. There

are situations, as in Figure 2, where it makes sense to assign the same user-id to different users. In other situations, a single person may use two or three different user-ids. In a production database, authorization-ids may be associated with programs and groups of programs, rather than with human users. In each of these situations, authorization-id is a more precise and less confusing term than user-id. However, the most common practice is to assign a different user-id to each person, and most SQL-based DBMS use the term user-id in their documentation.

7.1.2 User Authentication

User authentication is a process that allows a device to verify the identify of someone who connects to a network resource. There are many technologies currently available to a network administrator to authenticate users. Fireware operates with frequently used applications, including RADIUS, Windows Active Directory, LDAP, and token-based SecurID. The Firebox also has its own authentication server. You can use the Firebox authentication features to monitor and control connections through the Firebox.

When you type SQL statements into an interactive SQL utility, how does the DBMS determine which user-id is associated with the statements? If you use a forms-based data entry or query program, how does the DBMS determine your user-id? On a database server, a report-generating program might be scheduled to run at a preset time every evening; what is the user-id in this situation, where there is no human user? Finally, how are user-ids handled when you access a database across a network, where your user-id on the system where you are actively working might be different than the user-id established on the system where the database resides?

Authentication is very important when you use dynamic IP addressing (DHCP) for computers on the trusted or optional network. It is also important if you must identify your users before you let them connect to resources on the external network. Because the Firebox® associates a user name to an IP address, we do not recommend that you use authentication features in a network with multi-user computers such as Unix servers, terminal servers or Citrix servers. The Firebox authenticates one user per computer.

The SQL standard specifies that user-ids provide database security; however, the specific mechanism for associating a user-id with a SQL statement is outside the scope of the standard because a database can be accessed in many different ways.

Most commercial SQL implementations establish a user-id for each database session. In interactive SQL, the session begins when you start the interactive SQL program, and it lasts until you exit the program. In an application program using programmatic SQL, the session begins when the application program connects to the DBMS, and it ends when the application program terminates. All of the SQL statements used during the session are associated with the user-id specified for the session. Usually, you must supply both a user-id and an associated password at the beginning of a session. The DBMS checks the password to verify that you are, in fact, authorized to use the user-id that you supply. Although user-ids and passwords are common across most SQL products, the specific techniques used to specify the user-id and password vary from one product to another.

Some DBMS brands, especially those that are available on many different operating system platforms, implement their own user-id/password security. For example, when you use Oracle's interactive SQL program, called SQL PLUS, you specify a user name and associated password in the command that starts the program, like this:

```
SQLPLUS      SCOTT/TIGER
```

The Sybase interactive SQL program, called ISQL, also accepts a user name and password, using this command format:

```
ISQL /USER=SCOTT /PASSWORD=TIGER
```

In each case, the DBMS validates the user-id (SCOTT) and the password (TIGER) before beginning the interactive SQL session. Many other DBMS brands, including Ingres and Informix, use the user names of the host computer's operating system as database user-ids. For example, when you log in to a UNIX-based computer system, you must supply a valid UNIX user name and password to gain access. To start the Ingres interactive SQL utility, you simply give the command:

```
ISQL SALESDB
```

where SALESDB is the name of the Ingres database you want to use. Ingres automatically obtains your UNIX user name and makes it your Ingres user-id for the session. Thus, you don't have to specify a separate database user-id and password. DB2's interactive SQL, running under MVS/TSO, uses a similar technique. Your TS0 login name automatically becomes your DB2 user-id for the interactive SQL session. SQL security also applies to programmatic access to a database, so the DBMS must determine and authenticate the user-id for every application program that tries to access the database. Again, the techniques and rules for establishing the user-id vary from one brand of DBMS to another. For widely used utility programs, such as a data entry or an inquiry program, it is common for the program to ask the user for a user-id and password at the beginning of the session, via a screen dialog. For more

specialized or custom-written programs, the appropriate user-id may be obvious from the application to be performed and hard-wired into the program.

The SQL2 standard also allows a program to use an **authorization**-id associated with a specific set of SQL statements (called a module), rather than the user-id of the particular person running the program. With this mechanism, a program may be given the ability to perform very specific operations on a database on behalf of many different users, even if those users are not otherwise authorized to access the target data. This is a convenient capability that is finding its way into mainstream SQL implementations.

Keyword

Authorization is the function of specifying access rights/privileges to resources related to information security and computer security in general and to access control in particular.

7.1.3 User Groups

A large production database often has groups of users with similar needs. In the sample database, for example, the three people in the order-processing department form a natural user group, and the two people in the accounts receivable department form another natural group. Within each group, all of the users have identical needs for data access and should have identical privileges. Under the ANSI/ISO SQL security scheme, you can handle groups of users with similar needs in one of two ways:

- You can assign the same user-id to every person in the group, as shown in Figure 2. This scheme simplifies security administration because it allows you to specify data access privileges once for the single user-id. However, under this scheme, the people sharing the user-id cannot be distinguished from one another in system operator displays and DBMS reports.
- You can assign a different user-id to every person in the group. This scheme lets you differentiate between the users in reports produced by the DBMS, and it lets you establish different privileges for the individual users later. However, you must specify privileges for each user individually, making security administration tedious and error-prone.

The scheme you choose depends on the trade-offs in your particular database and application.

Several DBMS brands, including Sybase and SQL Server, offer a third alternative for dealing with groups of similar users. They support group-ids, which identify groups of related user-ids. Privileges can be granted both to individual user-ids and to group-ids, and a user may carry out a database action if it is permitted by either the user-id or group-id privileges. Group-ids thus simplify the administration of privileges given to groups of users. However, they are nonstandard, and a database design using them may not be portable to another DBMS brand.

DB2 also supports groups of users but takes a different approach. The DB2 database administrator can configure DB2 so that when you first connect to DB2 and supply your user-id (known as your primary authorization-id), DBZ automatically looks up a set of additional user-ids (known as secondary authorization-ids) that you may use.

Keyword

Resource Access Control Facility (RACF) provides software access control measures that can be used to enhance data security in a computing system.

When DB2 later checks your privileges, it checks the privileges for all of your authorization-ids, primary and secondary. On an IBM mainframe system, the DB2 database administrator normally sets up the secondary authorization-ids so that they are the same as the user group names used by **Resource Access Control Facility (RACF)**, the IBM mainframe security facility. Thus, the DBZ approach effectively provides group-ids but does so without adding to the user-id mechanism.

7.2 SQL INJECTION (SQLI)

SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. An SQL query is a request for some action to be performed on a database. Typically, on a Web form for user authentication, when a user enters their name and password into the text boxes provided for them, those values are inserted into a SELECT query. If the values entered are found as expected, the user is allowed access; if they aren't found, access is denied. However, most Web forms have no mechanisms in place to block input other than names and passwords. Unless such precautions are taken, an attacker can use the input boxes to send their own request to the database, which could allow them to download the entire database or interact with it in other illicit ways.

The risk of SQL injection exploits is on the rise because of automated tools. In the past, the danger was somewhat limited because an exploit had to be carried out manually: an attacker had to actually type their SQL statement into a text box. However, automated SQL injection programs are now available, and as a result, both the likelihood and the potential damage of an exploit has increased enormously. In an interview with Security Wire Perspectives, Caleb Sima, CTO of SPI Dynamics spoke of the potential danger: "This technology being publicly released by some black hat will give script-kiddies the ability to pick up a freeware tool, point it at a Web site and automatically download a database without any knowledge whatsoever. I think that makes things a lot more critical and severe. The automation of SQL injection gives rise to the possibility of a SQL injection worm, which is very possible..

According to security experts, the reason that SQL injection and many other exploits, such as cross-site scripting, are possible is that security is not sufficiently emphasized in development. To protect the integrity of Web sites and applications, experts recommend simple precautions during development such as controlling the types and numbers of characters accepted by input boxes.

7.2.1 SQL Injection Attacks

An SQL Injection Attack is probably the easiest attack to prevent, while being one of the least protected against forms of attack. The core of the attack is that an SQL command is appended to the back end of a form field in the web or application front end (usually through a website), with the intent of breaking the original SQL Script and then running the SQL script that was injected into the form field. This SQL injection most often happens when you have dynamically generated SQL within your front-end application. These attacks are most common with legacy Active Server Pages (ASP) and Hypertext Preprocessor (PHP) applications, but they are still a problem with ASP.NET web-based applications. The core reason behind an SQL Injection attack comes down to poor coding practices both within the front-end application and within the database stored procedures. Many developers have learned better development practices since ASP.NET was released, but SQL Injection is still a big problem between the number of legacy applications out there and newer applications built by developers who didn't take SQL Injection seriously while building the application.

As an example, assume that the front-end web application creates a dynamic SQL Script that ends up executing an SQL Script similar to that shown in Example 1.

```
SELECT * FROM Orders WHERE OrderId=25
```

Example 1: A simple dynamic SQL statement as expected from the application.

This SQL Script is created when the customer goes to the sales order history portion of the company's website. The value passed in as the OrderId is taken from the query string in the URL, so the query shown above is created when the customer goes to

the URL http://www.yourcompany.com/orders/orderhistory.aspx?Id=25. Within the .NET code, a simple string concatenation is done to put together the SQL Query. So any value that is put at the end of the query string is passed to the database at the end of the select statement. If the attacker were to change the query string to something like "/orderhistory.aspx?id=25; delete from Orders," then the query sent to the SQL Server will be a little more dangerous to run as shown in Example 2.

SELECT * FROM Orders WHERE ORderId=25; delete from Orders;

Example 2: A dynamic SQL String that has had a delete statement concatenated to the end of it.

The way the query in Example 2 works is that the SQL database is told via the semicolon ";" that the statement has ended and that there is another statement that should be run. The SQL Server then processes the next statement as instructed. While the initial query is run as normal now, and without any error being generated but when you look at the Orders table, you won't see any records in the Orders table because the second query in that batch will have executed against the database as well. Even if the attacker omits the value that the query is expecting, they can pass in "; delete from Orders;" and while the first query attempting to return the data from the Orders table will fail, the batch will continue moving on to the next statement, which will delete all the records in the Orders table.

Many people will inspect the text of the parameters looking for various key words in order to prevent these SQL Injection attacks. However, this only provides the most rudimentary protection as there are many, many ways to force these attacks to work. Some of these techniques include passing in binary data, having the SQL Server convert the binary data back to a text string, and then executing the string.

```
DECLARE @v varchar(255)
SELECT @v = cast(0x73705F68656C706462 as varchar(255))
EXEC (@v)
```

These new, unanticipated requests cause the database to perform the task the attacker intends. To clarify, consider the following simple example. Assume we have an application whose Web page contains a simple form with input fields for

Remember

SQL injection attacks are initiated by manipulating the data input on a Web form such that fragments of SQL instructions are passed to the Web application. The Web application then combines these rogue SQL fragments with the proper SQL dynamically generated by the application, thus creating valid .SQL requests

username and password. With these credentials the user can get a list of all credit card accounts they hold with a bank. Further assume that the bank's application was built with no consideration of SQL injection attacks. As such, it is reasonable to assume the application merely takes the input the user types and places it directly into an SQL query constructed to retrieve that user's information. In PHP that query string would look something like this:

```
$query = "select accountName, accountNumber from
creditCardAccounts where username='".$_POST["username"]."'
and password='".$_POST["password"]."'"
```

Normally this would work properly as a user entered their credentials, say johnSmith and myPassword, and formed the query:

```
$query = "select accountName, accountNumber from
creditCardAccounts where username='johnSmith' and
password='myPassword'
```

This query would return one or more accounts linked to Mr. Smith. Now consider someone with a devious intent. This person decides they want to see if they can access the account information of one or more of the bank's customers. To accomplish this they enter the following credential into the form:

```
' or 1=1 -- and anyThingsAtAll
```

When this SQL fragment is inserted into the SQL query by the application it becomes:

```
$query = "select accountName, accountNumber from
creditCardAccounts where username='' or 1=1 -- and
password= anyThingsAtAll
```

The injection of the term, ' or 1=1 --, accomplishes two things. First, it causes the first term in the SQL statement to be true for all rows of the query; second, the -- causes the rest of the statement to be treated as a comment and, therefore, ignored during run time. The result is that all the credit cards in the database, up to the limit the Web page will list, are returned and the attacker has stolen the valuable information they were seeking.

It should be noted that this simple example is just one of literally an infinite number of variations that could be used to accomplish the same attack. Further, there are many other ways to exploit a vulnerable application. We will discuss more of these attacks as we delve into the efficacy of various attack mitigation techniques.

7.2.2 Applications Vulnerable to SQL Injection

There are a number of factors that conspire to make securely written applications a rarity. First, many applications were written at a time when Web security was not a

major consideration. This is especially true of SQL injection. While recently SQL injection is being discussed at security conferences and other settings, the attack frequency of SQL injection only five or so years ago was low enough that most developers were simply not aware.

In addition, the application may have been initially written as an internal application with a lower security threshold and subsequently exposed to the Web without considering the security ramifications. Even applications being written and deployed today often inadequately address security concerns. IBM's X-Force project recently found that 47% of all vulnerabilities that result in unauthorized disclosures are Web application vulnerabilities. Cross-Site Scripting & SQL injection vulnerabilities continue to dominate as the attack vector of choice. Note that these reported vulnerabilities are for packaged applications from commercial software vendors. Vulnerabilities in custom applications were not reported. Since this software is generally not as carefully vetted for security robustness, it is reasonable to assume the problem is actually much bigger. According to Neira Jones, head of payment security for Barclays, 97% of data breaches worldwide are still due to an SQL injection somewhere along the line.

Keyword

JavaScript is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Interestingly, modern environments and development approaches create a subtle vulnerability. With the advent of Web 2.0 there has been a shift in how developers treat user input. In these applications input is rarely provided by a simple form that directly transmits the information into the Web server for processing. In many cases, the **JavaScript** portion of the application performs input validation so the feedback to the user is handled more smoothly. This often creates the sense that the application is protected because of this very specific input validation; therefore, the validation on the server side is largely neglected. Unfortunately, attackers won't use the application to inject their input into the server component of the application. Rather, they leverage intermediate applications to capture the client-side input and allow them to manipulate it. Since the majority of the input validation is bypassed, the attacker can simply enter the SQL fragments needed to change the behavior of the database to accomplish their intent.

7.2.3 The challenge with detection

Effective Security

The goal of any security technology is to provide a robust threat detection and avoidance mechanism that requires little or no setup, configuration or tuning. Further, if that technology relies on learning or training to determine what is normal or to improve its ability to detect threats, those learning periods must be short and well-defined. This is needed to expedite installation and minimize the risk of attacks contaminating the learned dataset. Keep in mind the longer the learning period, the more likely an attack will occur and the larger the dataset you need to review to ensure that an attack has not occurred. Finally, given that few Web applications remain static, effective protection must be easy to maintain in the face of on-going changes to the Web application.

Types of attacks

A simple attack on a vulnerable application was described to illustrate how a SQL Injection attack can occur. The general class of attacks that the simple example falls into can be described as Tautological attacks. Tautologies are statements composed of simpler statements in such a way that makes the statement true regardless if simpler statements are true or false. For example, the statement "Either it will rain tomorrow or it will not rain tomorrow" is a tautology. The complexity of detecting SQL injection can best be understood through a variety of examples demonstrating the various SQL injection attack classifications. This list is not exhaustive, but rather provides a sample of the most common injections seen in real deployments.

Tautologies

This attack works by inserting an "always true" statement into a WHERE clause of the SQL statement to extract data. These are often used in combination with the insertion of a -- to cause the remainder of a statement to be ignored ensuring extraction of largest amount of data. Tautological injections can include techniques to further mask SQL expression snippets, as demonstrated by the following example:

' or 'simple' like 'sim%' --

' or 'simple' like 'sim' || 'ple' --

The || in the example is used to concatenate strings, when evaluated the text 'sim' || 'ple' becomes 'simple'.

Union Query

This attack exploits a vulnerable parameter by injecting a statement of the form

foo'UNION SELECT <rest of injected query>

The attacker can insert any appropriate query to retrieve information from a table different from the one that was the target of the original statement. The database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Illegal/Logically Incorrect Queries

Attackers use this approach to gather important information about the type of database and its structure. Attacks of this nature are often used in the initial reconnaissance phase to gather critical knowledge used in other attacks. Returned error pages that are not filtered can be very instructive. Even if the application sanitizes error messages, the fact that an error is returned or not returned can reveal vulnerable or injectable parameters. Syntax errors identify injectable parameters; type errors help decipher data types of certain columns; logical errors, if returned to the user, can reveal table or column names.

The specific attacks within this class are largely the same as those used in a Tautological attack. The difference is that these are intended to determine how the system responds to different attacks by looking at the response to a normal input, an input with a logically true statement appended (typical tautological attack), an input with a logically false statement appended (to catch the response to failure) and an invalid statement to see how the system responds to bad SQL. This will often allow the attacker to see if an attack got through to the database even if the application does not allow the output from that statement to be displayed.

There are a myriad of examples. In fact, the attacker may initially use a bot to detect a vulnerable web site and then recursively use this class of attack forensically to learn application and database specifics. The key point in listing this classification is that WAFs are unable to detect such attacks if the injections fall outside of the signatures created by the WAF learning process. As well, the WAF may not be exposed to error messages that the application (and a Database Firewall) will receive.

Stored Procedure Attacks

These attacks attempt to execute database stored procedures. The attacker initially determines the database type (potentially using illegal/logically incorrect queries) and then uses that knowledge to determine what stored procedures might exist. Contrary to popular belief using stored procedures does not make the database invulnerable to SQL injection attacks. Stored procedures can be susceptible to privilege escalation, buffer overflows, and even provide access to the operating system.

Alternate Encoding Obfuscation

In this case, text is injected as to avoid detection by defensive coding practices. It can also be very difficult to generate rules for a WAF to detect encoded input. Encodings, in fact, can be used in combination with other attack classifications. Since databases parse comments out of an SQL statement prior to processing it, comments are often used in the middle of an attack to hide the attack's pattern.

Scanning and detection techniques, including those used in WAFs, have not been effective against alternate encodings or comment based obfuscation because all possible encodings must be considered. Note that these attacks may have no SQL keywords embedded as plain text, though it could run arbitrary SQL.

Combination Attacks

Many attack vectors may be employed in combination:

- Learn information useful in generating additional successful injections (illegal/logically incorrect)
- Gain access to systems other than the initial database accessed by the application (stored procedures)
- Evade detection by masking intent of injection (alternate encoding)

7.2.4 Detection at the Web Tier

Detecting SQL Injection Challenges

Given the large variation in the form or pattern of SQL attacks, it can be very challenging to detect them from a point in front of the Web server. At this network location the Web Application Firewall is attempting to identify a possible snippet of SQL in the input stream of a Web application.

Why is it difficult to detect input injections at the Web tier? Remember, the WAF is not inspecting the SQL request as sent to the database by the application tier. Rather, it has URL's, cookies and form inputs (POSTs and GETs) to inspect. Inspecting each set of input values, a WAF must consider the wide range of acceptable input against what is considered unacceptable for each input field on each form.

Although many attacks use special characters that may not be expected in a typical form, two problems complicate detection. With no prior knowledge of the application it is not possible to know with certainty what characters are expected in any given field. Furthermore, in some cases the characters used do, in fact, occur in normal input and blocking them at the character level is not possible. Consider the single quote often used to delimit a string. Unfortunately, this character appears in names such as

O'Brien or in possessive expressions like Steve's; therefore, single quotes are valid in some input fields.

As a result larger patterns must be considered, which are more demonstrative of an actual attack, to bring the false positives down to a reasonable rate. And this is where the problem begins. The choice then becomes: use a very general set of patterns such as checking for a single quote or the word "like" or possibly "or" to catch every conceivable attack or use a more complicated pattern that reduces the false positive rate.

Since there is a reasonable likelihood that general patterns exist in normal input, the WAF must then inspect all form input (in learning or training mode) for an extended period of time before it can determine which of these simple patterns can reliably be used to validate each form and each input field in the Web application. Considering the complexity, range and limited structure within the natural language used in forms, it can take a very long time to ensure that an adequate sample size has been gathered to confirm that selected detection patterns are not found in legitimate input. Complicating this further is the fact that some sections of an application are often used infrequently, extending even further the training time. An example would be business logic exercised according to the business cycle. Add it all up and you can see this approach requires an extensive time period to ensure that the learning cycle has adequately considered all the variations of valid input for each field on each form of the Web application.

Alternatively, as mentioned above, much more complex patterns that are clearly indicative of an attack can be used. Unfortunately, as we demonstrated in our discussion of the attack types, the number and variation of possible attacks is so large that it is impossible to effectively cover all possible attack patterns. Creating the initial pattern set, keeping up with the evolving attacks and verifying that they are sufficiently unique as to not show up in some fields is an almost impossible task. And now, consider that the applications are also changing and evolving over time, requiring further, time consuming learning.

Remember

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

Web Tier Detection in Practice

So how are WAF's used in the real world? One way is to use a combination of approaches, each aimed at reducing the

negative effects of the other approach. These negative effects include limited capability to detect a SQL injection versus high number of false positives, complex configurations, and long training times. Specifically, a large set of patterns ranging from relatively simple to much more complex are used. Some patterns are configured to be applied to all input sources regardless of what is learned during training; some patterns are configured such that they will be removed, for a given input field, if they are contained within the training data. Some rules and patterns also attempt to classify the range of input by length and character set, for example, numerical fields.

The WAF is then placed into learning mode and allowed to learn until it is believed that a large enough set of each input field has been examined to reduce subsequent false positives. The resulting sets are then reviewed to determine if the learned set for some fields is considered too small, requiring additional learning time or manual manipulation. Other fields, whose default rule set have been reduced too far, are reviewed to determine what hand crafted rules can be constructed to increase the coverage.

This manual inspection process on top of the long learning cycle, while more effective than any one approach in isolation, is far from efficient. However, it still suffers the weaknesses of an administrator having to make decisions, configuring a significant number of rule/pattern sets for fields not effectively configured through training. This can be true even after a substantial learning period has been used.

This, in a nutshell, is why WAFs have been ineffective in curtailing SQL injection attacks. It's self-evident, had WAFs been effective the size and scope of SQL injection attacks would not be increasing year over year.

7.2.5 A Better way – a Database Firewall

Thus far we have described the method of detecting SQL injection attacks at the Web tier interface. A more effective and efficient method is to analyze the actual SQL generated by the application and presented to the database. The Database Firewall monitors the networks between the application servers and databases (see Figure 3). Why is this more effective and more efficient? The simple answer is that while the input into the Web tier has an enormous pattern set with very little structure associated with each input field, an application creates a comparatively small set of SQL statements (ignoring the literal values associated with those statements). In addition the structure of SQL statement lends themselves to structured analysis. Both of these factors make analysis more determinant than the rudimentary input pattern validation of a WAF. We will discuss how to deal with the variation of the literal values (the actual intended user input) below.

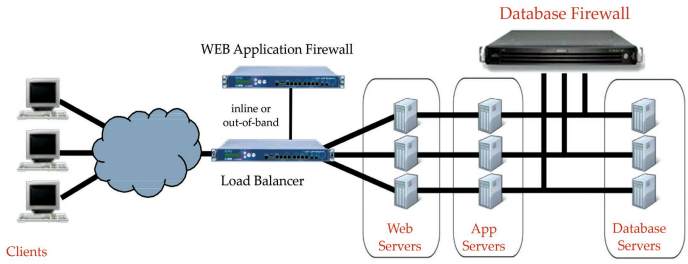


Figure 3. Placement of Database Firewall.

Keyword

Code generator is a tool or resource that generates a particular sort of code or computer programming language.

At the database interface, an SQL statement can be processed in much the same way the database itself processes it – breaking it down into the statement structure and separating out the literals. Once this is done the very first use of any given input will generate the unique SQL statements associated with that input – as opposed to needing a large sample set to determine what patterns are not present.

As a result the sample set for learning is already reduced from that required for a WAF to a much smaller set needed to train a device inspecting traffic between the application and database. Once a working training set is developed it can be used to analyze all subsequent SQL statements and any structure differs from the known set can be immediately flagged. By inspecting traffic at the interface to the database, it is clear which commands are leveraging stored procedures and it is easy to analyze the strings passed to stored procedures to determine if they contain any attacks. Several techniques can be applied in this analysis, such as observing the lack of delimiting special characters within literal strings.

Although analyzing the stream of SQL statements as described above provides a significant improvement over a WAF sitting at the Web tier, a true Database Firewall requires additional capabilities. As pointed out during the discussion about training a WAF, many of the input fields within an application may not be exercised often during normal operations. Fortunately, most modern applications build their SQL from a set of logic that operates much like a **code generator**. This fact means that, using a relatively small sample set, it is possible to construct a model of how an application

builds statements. An Adaptive Database Firewall can then use that knowledge to analyze newly discovered statements and assess their likeliness of being an attack.

In addition, given the fact that an SQL injection attack must be constructed out of an existing statement in the application further simplifies the analysis. If a new statement can be created wholly by inserting a string into the literal field of an existing statement, then it becomes highly suspect. Combining these concepts provides a means of assessing any new statement using algorithms that determine:

- Uniqueness relative to other statements previously seen
- Ability for that statement to have been constructed from a previously known statement
- Likelihood that the statement could have been generated within the application itself

Although an Adaptive Database Firewall uses a number of other important algorithms for analyzing incoming SQL against the learned model (for each application), the three algorithms highlighted above demonstrate the substantial value of operating at the interface to the database. No other approach can come close to the accuracy provided with this architecture. Furthermore, no other solution can be deployed with as little configuration and as short a training interval.

Did You Know?

SQL injection (SQLI) was considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the Open Web Application Security Project. In 2013, SQLI was rated the number one attack on the OWASP top ten.

7.2.6 Cleaning Up the Database after an SQL Injection Attack

There are a few different attacks that an attacker can perform against an SQL Server database. Delete commands can be passed into the SQL engine. However, other commands can be executed as well. Usually, attackers don't want to delete data or take a system offline; they instead want to use the SQL database to help launch other attacks. A simple method is to identify tables and columns that are used to display data on the website that uses the database as a backend. Then extra data is included in the columns of the database, which will allow attacking code to be executed against the database. This can be done using an update statement that puts an HTML iframe tag into each row of a table. This way when customers

view the website, they get the iframe put into their web browser, which could be set to a height of 0 so that it isn't visible. This hidden iframe could then install viruses or spyware on the user's computer without their knowledge.

Once this attack has occurred and viruses or spyware have been installed on the customer's computer, the most important thing now is to stop additional users' computers from being attacked. This means going through every record of every table looking for the attack code that is pushing the iframe to the customer's web browser. Obviously, you can go through each table manually looking for the records in question, or you can use the included sample code, which searches through each column in every table for the problem code and removes it. All you need to do is supply the variable with the attack code. The only columns that are not cleaned by this code are columns that use the TEXT or NTEXT data types. This is because the TEXT and NTEXT data types require special attention as they do not support the normal search functions.

```
DECLARE @injected_value NVARCHAR(1000)
SET @injected_value = 'Put the code which has been injected here.'
/)Change nothing below this line.)/
SET @injected_value = REPLACE(@injected_value, '''', '''''')
CREATE TABLE #ms_ver (indexid INT, name sysname, internal_value
INT, character_value VARCHAR(50))
INSERT INTO #ms_ver
EXEC xp_msver 'ProductVersion'
DECLARE @database_name sysname, @table_schema sysname,
@table_name sysname, @column_name sysname, @cmd NVARCHAR
(4000),
@internal_value INT
SELECT @internal_value = internal_value
FROM #ms_ver
DECLARE cur CURSOR FOR SELECT TABLE_CATALOG, TABLE_SCHEMA,
TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.columns c
JOIN systypes st ON c.DATA_TYPE = st.name
WHERE xtype IN (97, 167, 175, 231, 239, 241)
OPEN cur
FETCH NEXT FROM cur INTO @database_name, @table_schema,
@table_name, @column_name
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
SET @cmd = 'SELECT NULL
WHILE @@ROWCOUNT <> 0
BEGIN
'
IF @internal_value > 530000
SET @cmd = @cmd þ ' SET ROWCOUNT 1000
UPDATE'
ELSE
SET @cmd = @cmd þ ' UPDATE TOP (1000)'
SET @cmd = @cmd þ ' [' þ @database_name þ '].[' þ @table_
schema þ '].[' þ @table_name þ ']
SET [' þ @column_name þ '] = REPLACE([' þ @column_name þ '], '
'' þ @injected_value þ ''', '''')
WHERE [' þ @column_name þ '] LIKE ''%' þ @injected_value þ '%''
END'
exec (@cmd)
FETCH NEXT FROM cur INTO @database_name, @table_schema,
@table_name, @column_name
END
CLOSE cur
DEALLOCATE cur
DROP TABLE #ms_ver
```

SUMMARY

- Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset, such as a person, dwelling, community, nation, or organization.
- When you entrust your data to a database management system, the security, of the stored data is a major concern. Security is especially important in an SQL-based DBMS because interactive SQL makes database access very easy.
- Implementing a security scheme and enforcing security restrictions are the responsibility of the DBMS software. The SQL language defines an overall framework for database security, and SQL statements are used to specify security restrictions.
- User ID is a feature in Matomo (Piwik) that lets you connect together a given user's data collected from multiple devices and multiple browsers.
- User authentication is a process that allows a device to verify the identify of someone who connects to a network resource. There are many technologies currently available to a network administrator to authenticate users.
- SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. An SQL query is a request for some action to be performed on a database.
- An SQL Injection Attack is probably the easiest attack to prevent, while being one of the least protected against forms of attack. The core of the attack is that an SQL command is appended to the back end of a form field in the web or application front end (usually through a website), with the intent of breaking the original SQL Script and then running the SQL script that was injected into the form field.

KNOWLEDGE CHECK

1. **Which of the following condition in the where clause will return the login identification name of the user?**
   a. UserName = SUSER_NAME()
   b. UserName = SUSER_SNAME()
   c. UserName = CURRENT_USER()
   d. UserName = USER()
2. **Point out the correct statement:**
   a. Implementing row level security based on security labels is possible in SQL Server 2008
   b. A security label is a marking that describes the securable content of an item
   c. Row-level permissions are used for applications that store information in a single table
   d. None of the mentioned
3. **View that contains the list of all the security labels present in the database:**
   a. vwVisibleLabels
   b. VisibleLabels
   c. vwVisibleLabel
   d. All of the mentioned
4. **Which of the following retrieves a SecurityLabel instance describing the subject label of the current database user?**
   a. fn_Dominates
   b. usp_GetUserLabel
   c. usp_GetCurrentUserLabel
   d. usp_GetSecLabelDetails
5. **Point out the wrong statement :**
   a. usp_EnableCellVisibility opens all the symmetric keys that are mapped to security labels
   b. usp_DisableCellVisibility opens all the symmetric keys that were previously opened
   c. On SQL Server 2012 you can use the Contained Database feature to create a user without a login
   d. No arguments are available for usp_EnableCellVisibility

6. QL injection is an attack in which _____ code is inserted into strings that are later passed to an instance of SQL Server.

a. malicious

b. redundant

c. clean

d. non malicious

7. Any user-controlled parameter that gets processed by the application includes vulnerabilities like _____

a. Host-related information

b. Browser-related information

c. Application parameters included as part of the body of a POST request

d. All of the mentioned

8. Which of the stored procedure is used to test the SQL injection attack?

a. xp_write

b. xp_regwrite

c. xp_reg

d. all of the mentioned

9. If xp_cmdshell has been disabled with sp_dropextendedproc, we can simply inject the following code?

a. sp_addextendedproc 'xp_cmdshell','xp_log70.dll'

b. sp_addproc 'xp_cmdshell','xp_log70.dll'

c. sp_addextendedproc 'xp_cmdshell','log70.dll'

d. none of the mentioned

10. _____ is time based SQL injection attack.

a. Quick detection

b. Initial Exploitation

c. Blind SQL Injection

d. Inline Comments

REVIEW QUESTIONS

1. Give an overview on user ids and user authentication.
2. Discuss on the effect of SQL injection attacks.
3. How to applications are vulnerable to SQL injection?
4. What are the better way of a database firewall?
5. How to cleaning up the database after an SQL injection attack?

Check Your Result

1. (b)	2. (c)	3. (a)	4. (c)	5. (b)
6. (a)	7. (d)	8. (b)	9. (a)	10. (c)

REFERENCES

1. Database Benchmarking: Practical methods for Oracle & SQL Server by Bert Scalzo, Kevin E. Kline, Donald K. Burleson, Mike Ault, Claudia Fernandez
2. https://cdn.ttgtmedia.com/rms/pdf/Cherry_Securing_SQL_Chap6.pdf
3. https://crypto.stanford.edu/cs142/lectures/16-sql-inj.pdf
4. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/overview-of-sql-server-security
5. https://downloads.mysql.com/docs/mysql-security-excerpt-5.5-en.pdf
6. https://matomo.org/docs/user-id/#about-user-id
7. https://repo.zenk-security.com/Techniques%20d.attaques%20%20.%20%20Failles/SQL%20Injection.pdf
8. https://searchsecurity.techtarget.com/definition/user-authentication
9. https://searchsoftwarequality.techtarget.com/definition/SQL-injection
10. https://www.acunetix.com/websitesecurity/sql-injection/
11. https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf
12. https://www.dbcybertech.com/pdf/sql-injection-detection-web-environment.pdf
13. https://www.netsparker.com/blog/web-security/sql-injection-vulnerability/
14. https://www.watchguard.com/training/fireware/82/authent2.htm
15. Pro SQL Server 2008 Relational Database Design and Implementation by Louis Davidson, Kevin E. Kline
16. Michael Zino (May 1, 2008). “ASCII Encoded/Binary String Automated SQL Injection Attack”. Archived from the original on June 1, 2008.
17. Sumner Lemon, IDG News Service (May 19, 2008). “Mass SQL Injection Attack Targets Chinese Web Sites”. PCWorld. Retrieved May 27, 2008.

CHAPTER

8

SQL TABLE

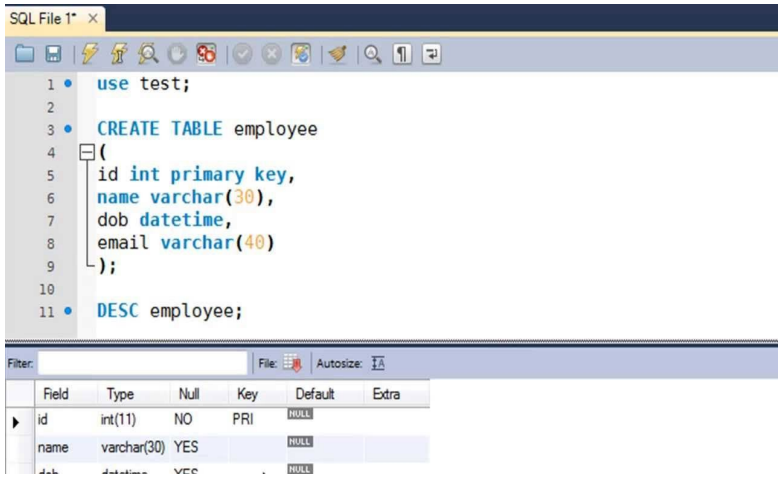
"In order to have a decentralised database, you need to have security. In order to have security, you need to - you need to have incentives. "

— Vitalik Buterin

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Describe the use of table variable in SQL
2. Explain how to create a table using SQL query
3. Discuss how to drop a table
4. Describe how to delete all the records of a table
5. Explain how to rename, truncate, copy a table
6. Learn about the types of temp tables
7. Discuss about ALTER table



INTRODUCTION

A table is a collection of related data held in a table format within a database. It consists of columns and rows.

In relational databases, and flat file databases, a *table* is a set of data elements (values) using a model of vertical columns (identifiable by name) and horizontal rows, the cell being the unit where a row and column intersect. A table has a specified number of columns, but can have any number of rows. Each row is identified by one or more values appearing in a particular column subset. A specific choice of columns which uniquely identify rows is called the primary key.

"Table" is another term for "relation"; although there is the difference in that a table is usually a multiset (bag) of rows where a relation is a set and does not allow duplicates. Besides the actual data rows, tables generally have associated with them some metadata, such as constraints on the table or on the values within particular columns.

The data in a table does not have to be physically stored in the database. Views also function as relational tables, but their data are calculated at query time. External tables (in Informix or Oracle, for example) can also be thought of as views. In many systems for computational statistics, such as R and Python's pandas, a data frame or data table is a data type supporting the table abstraction. Conceptually, it is a list of records or observations all containing the same fields or columns. The implementation consists of a list of arrays or vectors, each with a name.

8.1 TABLE

Keyword

Data storage is a technology consisting of computer components and recording media that are used to retain digital data.

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

Table is the simple form of **data storage**. A table is also considered as a convenient representation of relations.

Let's see an example of an employee table:

Employee		
EMP_NAME	**ADDRESS**	**SALARY**
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

In the above table, "Employee" is the table name, "EMP_NAME", "ADDRESS" and "SALARY" are the column names. The combination of data of multiple columns forms a row e.g. "Ankit", "Lucknow" and 15000 are the data of one row.

8.1.1 SQL TABLE Variable

The SQL Table variable is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.

It is a variable where we temporary store records and results. This is same like temp table but in the case of temp table we need to explicitly drop it.

Table variables are used to store a set of records. So declaration syntax generally looks like CREATE TABLE syntax.

1. **create table** "tablename"
2. ("column1" "data type",
3. "column2" "data type",
4. ...
5. "columnN" "data type");

When a transaction rolled back the data associated with table variable is not rolled back.

A table variable generally uses lesser resources than a temporary variable.

Table variable cannot be used as an input or an output parameter.

Remember

A table has a specified number of columns, but can have any number of rows.

Did You Know?

Table variable was introduced with SQL server 2000 to be an alternative of temporary tables.

8.2 SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

1. **create table** "tablename"
2. ("column1" "data type",

3. "column2" "data type",
4. "column3" "data type",
5. ...
6. "columnN" "data type");

The data type of the columns may vary from one database to another.

NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

Let us take an example to create a STUDENTS table with ID as primary key and NOT NULL are the constraint showing that these fields cannot be NULL while creating records in the table.

1. SQL> **CREATE TABLE** STUDENTS (
2. ID **INT** NOT NULL,
3. **NAME VARCHAR** (20) NOT NULL,
4. AGE **INT** NOT NULL,
5. ADDRESS **CHAR** (25),
6. **PRIMARY KEY** (ID)
7.);

You can verify it, if you have created the table successfully by looking at the message displayed by the SQL Server, else you can use DESC command as follows:

SQL> DESC STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			
AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

4 rows in set (0.00 sec)

Now you have the STUDENTS table available in your database and you can use to store required information related to students.

8.2.1 SQL CREATE TABLE Example in MySQL

Let's see the command to create a table in MySQL database.

```
1. CREATE TABLE Employee
2. (
3. EmployeeID int,
4. FirstName varchar(255),
5. LastName varchar(255),
6. Email varchar(255),
7. AddressLine varchar(255),
8. City varchar(255)
9. );
```

Keyword

Oracle Database is a multi-model database management system produced and marketed by Oracle Corporation.

8.2.2 SQL CREATE TABLE Example in Oracle

Let's see the command to create a table in **Oracle database**.

```
1. CREATE TABLE Employee
2. (
3. EmployeeID number(10),
4. FirstName varchar2(255),
5. LastName varchar2(255),
6. Email varchar2(255),
7. AddressLine varchar2(255),
```

8. City varchar2(255)
9.);

8.2.3 SQL CREATE TABLE Example in Microsoft SQLServer

Let's see the command to create a table in SQLServer database. It is same as MySQL and Oracle.

1. **CREATE TABLE** Employee
2. (
3. EmployeeID **int**,
4. FirstName **varchar**(255),
5. LastName **varchar**(255),
6. Email **varchar**(255),
7. AddressLine **varchar**(255),
8. City **varchar**(255)
9.);

8.2.4 Create a Table Using another Table

We can create a copy of an existing table using the create table command. The new table gets the same column signature as the old table. We can select all columns or some specific columns.

If we create a new table using an old table, the new table will be filled with the existing value from the old table.

The basic syntax for creating a table with the other table is:

1. **CREATE TABLE** table_name **AS**
2. **SELECT** column1, column2,...
3. **FROM** old_table_name **WHERE** ..... ;
4. The following SQL creates a copy **of** the employee **table**.
5. **CREATE TABLE** EmployeeCopy **AS**

6. **SELECT** EmployeeID, FirstName, Email
7. **FROM** Employee;

8.2.5 SQL Primary Key with CREATE TABLE Statement

The following query creates a **PRIMARY KEY** on the "D" column when the "Employee" table is created.

Keyword

Primary key is a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table).

MySQL

1. **CREATE TABLE** Employee(
2. EmployeeID NOT NULL,
3. FirstName **varchar**(255) NOT NULL,
4. LastName **varchar**(255),
5. City **varchar**(255),
6. **PRIMARY KEY** (EmployeeID)
7.);

SQL Server / Oracle / MS Access

1. **CREATE TABLE** Employee(
2. EmployeeID NOT NULL **PRIMARY KEY**,
3. FirstName **varchar**(255) NOT NULL,
4. LastName **varchar**(255),
5. City **varchar**(255)
6.);

Use the following query to define a PRIMARY KEY constraints on multiple columns, and to allow naming of a PRIMARY KEY constraints.

For MySQL / SQL Server /Oracle / MS Access

```
1. CREATE TABLE Employee(
2. EmployeeID NOT NULL,
3. FirstName varchar(255) NOT NULL,
4. LastName varchar(255),
5. City varchar(255),
6. CONSTRAINT    PK_Employee PRIMARY KEY (EmployeeID, FirstName)
7. );
```

8.3 SQL DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

Let's see the syntax to drop the table from the database.

- **DROP TABLE** "table_name";

Let us take an example:

First we verify STUDENTS table and then we would delete it from the database.

- SQL> **DESC** STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			
AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

4 rows in set (0.00 sec)

This shows that STUDENTS table is available in the database, so we can drop it as follows:

1. SQL>**DROP TABLE** STUDENTS;

Now, use the following command to check whether table exists or not.

1. SQL> **DESC** STUDENTS;

1. Query OK, 0 rows affected (0.01 sec)

As you can see, table is dropped so it doesn't display it.

8.3.1 SQL DROP TABLE Example in MySQL

Let's see the command to drop a table from the MySQL database.

1. **DROP TABLE** table_name;

8.3.2 SQL DROP TABLE Example in Oracle

Let's see the command to drop a table from Oracle database. It is same as MySQL.

1. **DROP TABLE** table_name;

8.3.3 SQL DROP TABLE Example in Microsoft SQLServer

Let's see the command to drop a table from SQLServer database. It is same as MySQL.

1. **DROP TABLE** table_name;

8.4 SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM** table_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM** table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

8.4.1 Difference between DELETE and TRUNCATE Statements

There is a slight difference b/w delete and truncate statement. The DELETE statement only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The TRUNCATE statement**:** it is used to delete all the rows from the table **and free the containing space.**

Let's see an "employee" table.

Emp_id	Name	Address	Salary
1	Aryan	Allahabad	22000
2	Shurabhi	Varanasi	13000
3	Pappu	Delhi	24000

Execute the following query to truncate the table:

1. **TRUNCATE TABLE** employee;

8.4.2 Difference b/w DROP and TRUNCATE Statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints will be dropped
- Access privileges will also be dropped

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

8.5 SQL RENAME TABLE

In some situations, **database administrators** and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.

Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

The RENAME TABLE and ALTER TABLE syntax help in changing the name of the table.

8.5.1 Syntax of RENAME Statement in SQL

1. RENAME old_table _name To new_table_name ;

8.5.2 Examples of RENAME Statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to change the name of the SQL table in the database using **RENAME** statement:

Example 1: Let's take an example of a table named **Cars:**

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose, you want to change the above table name into "Car_2021_Details". For this, you have to type the following RENAME statement in SQL:

1. RENAME Cars To Car_2021_Details ;

- After this statement, the table "Cars" will be changed into table name "Car_2021_Details".

Keyword

Database administrators use specialized software to store and organize data.

Keyword

Rename refers to the altering of a name of a file. This can be done manually by using a shell command such as ren or mv, or by using batch renaming software that can automate the renaming process.

Example 2: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

- Suppose, you want to change the name of the above table into the **"Coding_ Employees".** For this, you have to type the following RENAME statement in SQL:

1. RENAME Employee To Coding_Employees ;

- After this statement, the table **"Employee"** will be changed into the table name **"Coding_Employees".**

8.5.3 Syntax of ALTER TABLE Statement in SQL

1. ALTER TABLE old_table_name RENAME TO new_table_name;

In the Syntax, we have to specify the RENAME TO keyword after the old name of the table.

Examples of ALTER TABLE Statement in SQL

Here, we have taken the following three different SQL examples, which will help you how to change the name of the table in the SQL database using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Bikes:**

Table: Bikes

Bike_Name	Bike_Color	Bike_Cost
KTM DUKE	Black	185,000
Royal Enfield	Black	NULL
Pulsar	Red	90,0000

Apache	White	NULL
Livo	Black	80,000
KTM RC	Red	195,000

- Suppose, you want to change the name of the above table into "Bikes_Details" using ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Bikes RENAME TO Bikes_Details ;

After this statement, the table "Bikes" will be changed into the table name "Bikes_Details".

Example 2: Let's take an example of a table named **Student:**

Table: Student

Stu_ID	Stu_Name	Stu_Marks
1001	Abhay	85
1002	Ankit	75
1003	Bheem	60
1004	Ram	79
1005	Sumit	80

- Suppose, you want to change the name of the above table into "MCA_Student_Details" using ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Student RENAME TO MCA_Student_Details ;

After this statement, the table "Student" will be changed into table name "MCA_Student_Details".

Example 3: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

- Suppose, you want to change the name of the above table into the **"Coding_Employees"** using an ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Employee RENAME To Coding_ Employees ;

After this statement, the table **"Employee"** will be changed into the table name **"Coding_Employees".**

8.6 SQL TRUNCATE TABLE

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

8.6.1 TRUNCATE TABLE Vs DELETE TABLE

Truncate table is faster and uses lesser resources than DELETE TABLE command.

8.6.2 TRUNCATE TABLE Vs. DROP TABLE

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

Let's see the syntax to truncate the table from the database.

1. **TRUNCATE TABLE** table_name;

For example, you can write following command to truncate the data of employee table

1. **TRUNCATE TABLE** Employee;

Remember

The rollback process is not possible after truncate table statement. Once you truncate a table you cannot use a flashback table statement to retrieve the content of the table.

8.7 SQL COPY TABLE

If you want to copy the data of one SQL table into another SQL table in the same SQL server, then it is possible by using the SELECT INTO statement in SQL.

The SELECT INTO statement in Structured Query Language copies the content from one existing table into the new table. SQL creates the new table by using the structure of the existing table.

8.7.1 Syntax of SELECT INTO statement in SQL

1. SELECT * INTO New_table_name FROM old_table_name;

8.7.2 Examples of SELECT INTO statement in SQL

In this section, we have taken the following three different SQL examples which will help you how to copy the content of one table into another table in SQL:

Example 1: In this example, we have a table called **Cars** with three columns:

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose you want to copy the content of the above Car table into the new table **Car_Details.** For this, you have to type the following query in SQL:

1. SELECT * INTO Car_Details FROM Cars;

- Let's check the **Car_Details** table is created successfully or not in the database:

1. SELECT * FROM Car_Details;

Table: Car_Details

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000

Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

Example 2: In this example, we have a table called **Employee** with four columns:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

Suppose you want to copy the record of the above Employee table into the new table **Coding_Employees.** For this, you have to type the following query in SQL:

1. SELECT * INTO Coding_Employees FROM Employee;

- Let's check the **Coding_Employees** table is created successfully or not in the database:

1. SELECT * FROM Coding_Employees;

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

Table: Coding_Employees

Example 3: In this example, we have a table called **Student** with four columns:

Table: Student

RollNo	Name	Marks	Age
1001	Bhanu	88	17
1002	Raman	82	16

1003	Sumit	80	16
1004	Shobhit	95	15
1005	Akash	85	16

- Suppose you want to copy the record of the above Student table into the new table **Class_12_Students.** For this, you have to type the following query in SQL:

```
1. SELECT * INTO Class_12_Students FROM Student;
```

- Let's check the table is **Class_12_Students** table created successfully or not in the database:

```
1. SELECT * FROM Class_12_Students;
```

RollNo	Name	Marks	Age
1001	Bhanu	88	17
1002	Raman	82	16
1003	Sumit	80	16
1004	Shobhit	95	15
1005	Akash	85	16

Table: Class_12_Students

Example 4: In this example, we have a table called **Cars** with three columns:

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose you want to copy **Car_Color** and **Car_Name** columns of the above Cars table into the new table **Car_Color**. For this, you have to type the following query in SQL:

```
1. SELECT Car_Name, Car_Color INTO Car_Color FROM Cars;
```

- Let's check the **Car_Color** table is created successfully or not in the database:

1. SELECT * FROM Car_Color;

Table: Car_Color

Car Name	Car Color
Hyundai Creta	White
Hyundai Venue	White
Hyundai i20	Red
Kia Sonet	White
Kia Seltos	Black
Swift Dezire	Red

8.7.3 Syntax of SELECT INTO Statement with WHERE Clause in SQL

1. SELECT * INTO New_table_name FROM old_table_name WHERE [condition] ;

Examples of SELECT INTO Statement with WHERE Clause in SQL

Here, we have taken the following three different SQL examples, which will help you how to copy the content of one table into another table with a specific condition in SQL:

Example 1: In this example, we have a table called **Cars** with three columns:

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	Black	10,85,000
Hyundai Venue	Black	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

Suppose we want to copy only the record of those cars whose color is black. For this, we have to type the following query in SQL:

1. SELECT * INTO Black_Car_Details FROM Cars WHERE Car_Color = 'Black';

- Let's check the **Black_Car_Details** table is created successfully or not in the database:

1. SELECT * FROM Black_Car_Details;

Table: Black_Car_Details

Car Name	Car Color	Car Cost
Hyundai Creta	Black	10,85,000
Hyundai Venue	Black	9,50,000
Kia Seltos	Black	8,00,000

Example 2: In this example, we have a table called **Employee** with four columns:

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	45000	Goa
202	Ankit	45000	Delhi
203	Bheem	38000	Goa
204	Ram	49000	Goa
205	Sumit	40000	Delhi

- Suppose we want to copy only the record of those employees whose Salary is more than 40,000. For this, we have to type the following query in SQL:

1. SELECT * INTO Emp_Salary_40000 FROM Cars WHERE Emp_Salary > 40000;

- Let's check the **Emp_Salary_40000** table created successfully or not in the database:

1. SELECT * FROM Emp_Salary_40000;

Table: Emp_Salary_40000

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	45000	Goa
202	Ankit	45000	Delhi
204	Ram	49000	Goa

8.8 SQL TEMP TABLE

The concept of temporary table is introduced by SQL server. It helps developers in many ways:

Temporary tables can be created at run-time and can do all kinds of operations that a normal table can do. These temporary tables are created inside tempdb database.

There are two types of temp tables based on the behavior and scope.

- Local Temp Variable
- Global Temp Variable

8.8.1 Local Temp Variable

Local temp tables are only available at current connection time. It is automatically deleted when user disconnects from instances. It is started with hash (#) sign.

Exception Handling in Java - Javatpoint

1. **CREATE TABLE** #**local temp table** (
2. User id **int**,
3. Username **varchar** (50),
4. User address **varchar** (150)
5.)

8.8.2 Global Temp Variable

Global temp tables name starts with double hash (##). Once this table is created, it is like a permanent table. It is always ready for all users and not deleted until the total connection is withdrawn.

1. **CREATE TABLE** ##new **global temp table** (
2. User id **int**,
3. User **name varchar** (50),
4. User address **varchar** (150)
5.)

8.9 SQL ALTER TABLE

The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table. This statement also allows database users to add and remove various SQL constraints on the existing tables.

Any user can also change the name of the table using this statement.

8.9.1 ALTER TABLE ADD Column Statement in SQL

In many situations, you may require to add the columns in the existing table. Instead of creating a whole table or database again you can easily add single and multiple columns using the ADD keyword.

8.9.2 Syntax of ALTER TABLE ADD Column Statement in SQL

1. ALTER TABLE table_name ADD column_name column-definition;

The above syntax only allows you to add a single column to the existing table. If you want to add more than one column to the table in a single SQL statement, then use the following syntax:

1. ALTER TABLE table_name
2. ADD (column_Name1 column-definition,
3. column_Name2 column-definition,
4. .....
5. column_NameN column-definition);

8.9.3 Examples of ALTER TABLE ADD Column Statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to add the single and multiple columns in the existing table using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Cars:**

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000

Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose, you want to add the new column Car_Model in the above table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Cars ADD Car_Model Varchar(20);

This statement will add the Car_Model column to the Cars table.

Example 2: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

- Suppose, you want to add two columns, **Emp_ContactNo.** and **Emp_EmailID,** in the above Employee table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Employee ADD (Emp_ContactNo. Number(13), Emp_EmailID varchar(50) ;

This statement will add Emp_ContactNo. and Emp_EmailID columns to the Employee table.

8.9.4 ALTER TABLE MODIFY Column Statement in SQL

The MODIFY keyword is used for changing the column definition of the existing table.

8.9.5 Syntax of ALTER TABLE MODIFY Column Statement in SQL

1. ALTER TABLE table_name MODIFY column_name column-definition;

This syntax only allows you to modify a single column of the existing table. If you want to modify more than one column of the table in a single SQL statement, then use the following syntax:

1. ALTER TABLE table_name
2. MODIFY (column_Name1 column-definition,
3. column_Name2 column-definition,
4. .....
5. column_NameN column-definition);

8.9.6 Examples of ALTER TABLE MODIFY Column Statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to modify single and multiple columns of the existing table using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Cars:**

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose, you want to modify the datatype of the **Car_Color** column of the above table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Cars ADD Car_Color Varchar(50);

Example 2: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

- Suppose, you want to modify the datatypes of two columns **Emp_ContactNo.** and **Emp_EmailID** of the above Employee table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Employee ADD (Emp_ContactNo. Int, Emp_EmailID varchar(80) ;

8.9.7 ALTER TABLE DROP Column Statement in SQL

In many situations, you may require to delete the columns from the existing table. Instead of deleting the whole table or database you can use DROP keyword for deleting the columns.

8.9.8 Syntax of ALTER TABLE DROP Column Statement in SQL

1. ALTER TABLE table_name DROP Column column_name ;

8.9.9 Examples of ALTER TABLE DROP Column Statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to delete a column from the existing table using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Cars:**

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000

Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose, you want to delete the Car_Color column from the above table. For this, you have to type the following query in the SQL:

```
1. ALTER TABLE Cars DROP COLUMN Car_Color ;
```

- Let's check using the following statement that the Car_Color column is deleted from the table or not:

```
1. SELECT * FROM Cars;
```

Table: Cars

Car Name	Car Cost
Hyundai Creta	10,85,000
Hyundai Venue	9,50,000
Hyundai i20	9,00,000
Kia Sonet	10,00,000
Kia Seltos	8,00,000
Swift Dezire	7,95,000

Example 2: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

- Suppose, you want to delete the **Emp_Salary and Emp_City** column from the above Employee table. For this, you have to type the following two different queries in the SQL:

```
1. ALTER TABLE Cars DROP COLUMN Emp_Salary ;
```

2. ALTER TABLE Cars DROP COLUMN Emp_City ;

8.9.10 ALTER TABLE RENAME Column Statement in SQL

The RENAME keyword is used for changing the name of columns or fields of the existing table.

8.9.11 Syntax of ALTER TABLE RENAME Column Statement in SQL

1. ALTER TABLE table_name RENAME COLUMN old_name to new_name;

8.9.12 Examples of ALTER TABLE RENAME Column Statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to change the name of a column of the existing table using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Cars:**

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

- Suppose, you want to change the name of the **Car_Color** column of the above Cars table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Cars RENAME COLUMN Car_Color to Colors;

This statement will change the name of a column of the Cars table. To see the changes, you have to type the following query:

1. SELECT * FROM Cars;

Table: Cars

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

Example 2: Let's take an example of a table named **Employee:**

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

Suppose, you want to change the name of **the Emp_City** column of the above Employee table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Employee RENAME COLUMN Emp_City to Emp_Address;

This statement will change the name of a column of the Employee table. To see the changes, you have to type the following query:

1. SELECT * FROM Employee;

Table: Employee

Emp_Id	Emp_Name	Emp_Salary	Emp_Address
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

SUMMARY

- A *table* is a set of data elements (values) using a model of vertical columns (identifiable by name) and horizontal rows, the cell being the unit where a row and column intersect.
- Table is another term for "relation"; although there is the difference in that a table is usually a multiset (bag) of rows where a relation is a set and does not allow duplicates. Besides the actual data rows, tables generally have associated with them some metadata, such as constraints on the table or on the values within particular columns.
- Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.
- The SQL Table variable is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.
- SQL CREATE TABLE statement is used to create table in a database.
- A SQL DROP TABLE statement is used to delete a table definition and all data from a table.
- The DELETE statement only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.
- The TRUNCATE statement: it is used to delete all the rows from the table and free the containing space.
- Database administrators and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.
- A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.
- Local temp tables are only available at current connection time. It is automatically deleted when user disconnects from instances. It is started with hash (#) sign.
- Global temp tables name starts with double hash (##). Once this table is created, it is like a permanent table. It is always ready for all users and not deleted until the total connection is withdrawn.
- The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table. This statement also allows database users to add and remove various SQL constraints on the existing tables.

KNOWLEDGE CHECK

1. **Which statement is used to delete all rows in a table without having the action logged?**
   a. DELETE
   b. REMOVE
   c. DROP
   d. TRUNCATE
2. **How many Primary keys can have in a table?**
   a. Only 1
   b. Only 2
   c. Depends on no of Columns
   d Depends on DBA
3. **Which of the following statement is true?**
   a. TRUNCATE free the table space while DELETE does not.
   b. Both TRUNCATE and DELETE statements free the table's space.
   c. Both TRUNCATE and DELETE statement does not free the table's space.
   d. DELETE free the table space while TRUNCATE does not.
4. **Which command is used to change the definition of a table in SQL?**
   a CREATE
   b. UPDATE
   c. ALTER
   d. SELECT
5. **Why we need to create an index if the primary key is already present in a table?**
   a. Index improves the speed of data retrieval operations on a table.
   b. Indexes are special lookup tables that will be used by the database search engine.
   c. Indexes are synonyms of a column in a table.
   d. All of the above

REVIEW QUESTIONS

1. What TABLE variable can do?
2. How to copy a table?
3. What is temporary table? What are the advantage of temporary table?
4. How to add, modify, rename and drop column.
5. What are the difference between DELETE and TRUNCATE statements?

Check Your Result

1. (d) 2. (a) 3. (a) 4. (c) 5. (a)

REFERENCES

1. Bryla, Bob; Thomas, Biju (2006). OCP: Oracle 10g New Features for Administrators Study Guide: Exam 1Z0-040. John Wiley & Sons. p. 90. ISBN 9780782150858. Retrieved 2015-08-14.
2. Drake, Mark (August 9, 2019). "A Comparison of NoSQL Database Management Systems and Models". Digital Ocean. Retrieved 2021-02-26.

CHAPTER

9

SQL CLAUSE

"Forging differs from hoaxing, inasmuch as in the later the deceit is intended to last for a time, and then be discovered, to the ridicule of those who have credited it; whereas the forger is one who, wishing to acquire a reputation for science, records observations which he has never made."

— Charles Babbage

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the use of WHERE clause
2. Describe the AND, OR and NOT operators in SQL
3. Learn about WITH clause in SQL
4. Discuss about HAVING Clause in SQL
5. Know the ORDER BY clause in SQL

INTRODUCTION

SQL is a query language which queries and returns the desired data from the database. We use SQL for multiple

operations related to data, some of them being viewing the data and analyzing the data.

Clauses are in-built functions available to us in SQL. With the help of clauses, we can deal with data easily stored in the table.

Clauses help us filter and analyze data quickly. When we have large amounts of data stored in the database, we use Clauses to query and get data required by the user.

Some of the examples of clauses are – where, and, or, with, as, etc.

9.1 WHERE CLAUSE

A **WHERE clause** in SQL is a **data manipulation language** statement. WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.

Actually, it filters the records. It returns only those queries which fulfill the specific conditions.

WHERE clause is used in SELECT, UPDATE, DELETE statement etc.

Let's see the syntax for sql where:

1. **SELECT** column1, **column** 2, ... **column** n
2. **FROM** table_name
3. **WHERE** [conditions]

WHERE clause uses some conditional selection

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
< >	not equal to

Keyword

Data manipulation language (DML) is a computer programming language used for adding (inserting), deleting, and modifying (updating) data in a database.

9.2 SQL AND, OR AND NOT OPERATORS

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

9.2.1 SQL AND

- The SQL **AND** condition is used in SQL query to create two or more conditions to be met.
- It is used in SQL **SELECT, INSERT, UPDATE** and **DELETE**
- Let's see the syntax for SQL AND:
- SELECT columns FROM tables WHERE condition 1 AND condition 2;
- The SQL AND condition require that both conditions should be met.
- The SQL AND condition also can be used to join multiple tables in a SQL statement.
- To understand this concept practically, let us see some examples.

Consider we have an employee table created into the database with the following data:

ID	First_ Name	Last_ Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

SQL "AND" example with "SELECT" statement

This is how an SQL "**AND**" condition can be used in the SQL **SELECT** statement.

Example 1:

Write a query to get the records from emp tables in which department of the employee is IT and location is Chennai.

Query:

- mysql> **SELECT *FROM** emp **WHERE** Department = "IT" AND Location = "Chennai";

ID	First_Name	Last_Name	Department	Location
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai

In the emp table, there are three employees whose department is IT. But we have specified the AND condition according to which the employee's location should not be other than Chennai. So, there are only two employees whose department is IT and Location is Chennai.

Keyword

Employee is an individual who was hired by an employer to do a specific job.

Example 2:

Write a query to get the records from emp tables in which department of the **employee** is IT and location is Mumbai.

Query:

- mysql> **SELECT *FROM** emp **WHERE** Department = "IT" AND Location = "Mumbai";

ID	First_ Name	Last_ Name	Department	Location
2	Anurag	Rajput	IT	Mumbai

In the emp table, there are three employees whose department is IT. Among these three employees, there is only one employee whose location is Mumbai. Due to the presence of the AND **operator** used in the query, a record must satisfy both conditions.

SQL "AND" example with "UPDATE" statement

This is how the "AND" condition can be used in the SQL UPDATE statement.

Example 1:

Write a query to update the records in emp tables in which department of the employee is Marketing, and the first name is Suraj. For that particular employee, set the updated value of the location as Delhi.

Query:

- mysql> **UPDATE** emp **SET** Location = "Delhi" **WHERE** Department = "Marketing" AND First_Name = "Suraj";

```
mysql> UPDATE emp SET Location = "Delhi" WHERE Department = "Marketing" AND First_Name = "Suraj";
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

We will use the SELECT query to verify the updated record.

- mysql> **SELECT** ***FROM** emp;

ID	First_ Name	Last_ Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

In the emp table, there are three employees whose department is IT. Among these three employees, there is only one employee whose location is Mumbai. Due to the presence of the AND operator used in the query, a record must satisfy both conditions.

Keyword

Operators are constructs defined within programming languages which behave generally like functions, but which differ syntactically or semantically.

Example 2:

Write a query to update the records in the emp table in which department of the employee is Finance and ID is 7. For that particular employee, set the updated value of the department as HR.

Query:

- mysql> **UPDATE** emp **SET** Department = "HR" **WHERE** Department = "Finance" AND ID = 7;

```
mysql> UPDATE emp SET Department = "HR" WHERE Department = "Finance" AND ID = 7;
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

We will use the SELECT query to verify the updated record.

- mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	HR	Bangalore

Keyword

Finance is a term for matters regarding the management, creation, and study of money and investments.

In the emp table, there are two employees whose department is Finance. Among these two employees, there is only one employee whose ID is 7. Due to the presence of AND operator used in the query, a record must have the department as **Finance** and ID as 7.

SQL "AND" Example with "DELETE" Statement

This is how an SQL "AND" condition can be used in the SQL DELETE statement.

Example 1:

Write a query to delete the records from the emp table in which the last name of the employee is Jain, and the Location is Bangalore.

Query:

- mysql> **DELETE FROM** emp **WHERE** Last_Name = 'Jain' AND Location = 'Bangalore';

```
mysql> DELETE FROM emp WHERE Last_Name = 'Jain' AND Location = 'Bangalore';
Query OK, 1 row affected (0.09 sec)
```

We will use the SELECT query to verify the deleted record.

- mysql> **SELECT *FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore

There is only one record in the emp table whose last name is Jain. But still, due to the presence of AND operator, the second condition will also be checked according to which employee's location should be Bangalore. So, only that particular record is deleted.

Example 2:

Write a query to delete the records from the emp table in which department of the employee is IT and Location is Mumbai.

Query:

- mysql> **DELETE FROM** emp **WHERE** Department = 'IT' AND Location = 'Mumbai';

```
mysql> DELETE FROM emp WHERE Department = 'IT' AND Location = 'Mumbai';
Query OK, 1 row affected (0.11 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT *FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
3	Chaitali	Tarle	IT	Chennai

4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore

There are three records in the emp table whose department is IT. But only one record is deleted from the emp table, which contains a total of 6 records. This happened because of the AND operator according to which the employee's location should mandatorily be Mumbai. Therefore there is only one record that satisfies both the conditions. Hence, it is deleted.

9.2.2 SQL OR

The SQL **OR** condition is used in SQL query to create a SQL statement where records are returned when any one condition met. It can be used in a **SELECT** statement, **INSERT** statement, **UPDATE** statement or **DELETE** statement.

Let's see the syntax for the OR condition:

- **SELECT** columns **FROM** tables **WHERE** condition 1 OR condition 2;

ID	First_ Name	Last_ Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

SQL "OR" example with SQL SELECT

This is how an SQL "OR" condition can be used in the SQL SELECT statement.

Example 1:

Write a query to get the records from emp tables in which department of the employee is IT or location is Chennai.

Query:

- mysql> **SELECT** ***FROM** emp **WHERE** Department = "IT" OR Location = "Chennai";

ID	First_ Name	Last_ Name	Department	Location
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai

In the emp table, there are three employees whose department is IT. But there are only two records whose location is Chennai. Still, all three records are displayed. This happened because we have specified OR operator in the query, according to which the record will be considered in the result set even any one condition is met.

Example 2:

Write a query to get the records from emp tables in which department of the employee is Marketing or location is Noida.

Query:

- mysql> **SELECT *FROM** emp **WHERE** Department = "Marketing" OR Location = "Noida";

ID	First_ Name	Last_ Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
5	Suraj	Tripathi	Marketing	Pune
7	Sandhya	Jain	Finance	Bangalore

There are two employees whose department is **Marketing** in the emp table, but still, three records are displayed. This happened because of the use of the OR operator in the query. Among the three records displayed above, the first two records satisfy condition 1; the second record satisfies both the conditions and the third record satisfies only condition 1. Due to the OR operator, even if anyone condition is satisfied, the record is considered in the result-set.

- SQL "OR" example with SQL UPDATE

This is how the "OR" condition can be used in the SQL UPDATE statement.

Example 1:

Write a query to update the records in emp tables in which department of the employee is Marketing, or the last name

Keyword

Marketing refers to the process an organization undertakes to engage its target audience, build strong relationships to create value in order to capture value in return.

is Tarle. For that particular employee, set the updated value of the location as Delhi.

Query:

- mysql> **UPDATE** emp **SET** Location = "Delhi" **WHERE** Department = "Marketing" OR Last_Name = "Tarle";

```
mysql> UPDATE emp SET Location = "Delhi" WHERE Department = "Marketing" OR Last_Name = "Tarle";
Query OK, 3 rows affected (0.07 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

We will use the SELECT query to verify the updated record.

- mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

There are two employees whose department is 'Marketing' and one record whose last name is 'Tarle' in the emp table. Though only one condition is still met, that record is considered and updated in the table due to the OR operator.

Example 2:

Write a query to update the records in the emp table in which department of the employee is Finance, or the first name is Sandhya. For that particular employee, set the updated value of the department as HR.

Query:

- mysql> **UPDATE** emp **SET** Department = "HR" **WHERE** Department = "Finance" OR First_Name = "Sandhya";

```
mysql> UPDATE emp SET Department = "HR" WHERE Department = "Finance" OR First_Name = "Sandhya";
Query OK, 2 rows affected (0.08 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

We will use the SELECT query to verify the updated record.

- mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Delhi
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Delhi
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	HR	Bangalore
7	Sandhya	Jain	HR	Noida

There are two employees whose department is 'Finance,' and among these two records, one record satisfies both the conditions in the emp table. However, both the records are considered and updated in the table due to the OR operator.

- SQL "OR" example with SQL DELETE

This is how an SQL "OR" condition can be used in the SQL DELETE statement.

Example 1:

Write a query to delete the records from the emp table in which the last name of the employee is Jain or Location is Bangalore.

Query:

- mysql> **DELETE FROM** emp **WHERE** Last_Name = 'Jain' OR Location = 'Bangalore';

```
mysql> DELETE FROM emp WHERE Last_Name = 'Jain' OR Location = 'Bangalore';
Query OK, 2 rows affected (0.09 sec)
```

We will use the SELECT query to verify the deleted record.

- mysql> **SELECT *FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune

There is only one record in the emp table whose last name is Jain and one record whose location is Bangalore. But still, due to the presence of an OR operator, even if anyone condition is satisfied, that particular record is deleted.

Example 2:

Write a query to delete the records from the emp table in which department of the employee is marketing and Location is Delhi.

Query:

- mysql> **DELETE FROM** emp **WHERE** Department = 'Marketing' OR Location = 'Delhi';

```
mysql> DELETE FROM emp WHERE Department = 'Marketing' OR Location = 'Delhi';
Query OK, 3 rows affected (0.07 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
2	Anurag	Rajput	IT	Mumbai
4	Pranjal	Patil	IT	Chennai

There is only one record in the emp table whose department is Marketing and one record whose location is Delhi. But still, due to the presence of an OR operator, even if anyone condition is satisfied, that particular record is deleted.

9.3 SQL WITH CLAUSE

The SQL WITH clause is used to provide a sub-query block which can be referenced in several places within the main SQL query. It was introduced by oracle in oracle 9i release2 database.

There is an example of employee table:

Syntax for the SQL WITH clause -

This syntax is for SQL WITH clause using a single sub-query alias.

- **WITH** <alias_name> **AS** (sql_sub-query_statement)
- **SELECT** column_list **FROM** <alias_name> [**table name**]
- [**WHERE** <join_condition>]

When you use multiple sub-query aliases, the syntax will be as follows.

- **WITH** <alias_name_A> **AS** (sql_sub-query_statement)
- <alias_name_B> **AS** (sql_sub-query_statement_from_alias_name_A
- Or sql_sub-query_statement)

- **SELECT** <column_list>
- **FROM** <alias_name_A >,< alias_name_B >, [tablenames]
- [**WHERE** < join_condition>]

9.3.1 SQL SELECT AS

- SQL '**AS**' is used to assign a new name temporarily to a table column or even a table.
- It makes an easy presentation of query results and allows the developer to label results more accurately without permanently renaming table columns or even the table itself.
- Let's see the syntax of select as:

1. **SELECT** Column_Name1 **AS** New_Column_Name, Column_Name2 **As** New_Column_Name **FROM** Table_Name;

Here, the Column_Name is the name of a column in the original table, and the New_Column_Name is the name assigned to a particular column only for that specific query. This means that New_Column_Name is a temporary name that will be assigned to a query.

Assigning a temporary name to the column of a table:

Let us take a table named orders, and it contains the following data:

Day_of_order	Customer	Product	Quantity
11-09-2001	Ajeet	Mobile	2
13-12-2001	Mayank	Laptop	20
26-12-2004	Balaswamy	Water cannon	35

Example:

Suppose you want to rename the 'day_of_order' column and the 'customer' column as 'Date' and 'Client', respectively.

Query:

- **SELECT** day_of_order **AS** 'Date', Customer **As** 'Client', Product, Quantity **FROM** orders;

The result will be shown as this table:

Day_of_order	Customer	Product	Quantity
11-09-2001	Ajeet	Mobile	2
13-12-2001	Mayank	Laptop	20
26-12-2004	Balaswamy	Water cannon	35

Remember

SQL AS is the same as SQL ALIAS.

From the above results, we can see that temporarily the 'Day_of_order' is renamed as 'date' and 'customer' is renamed as 'client'.

Let us take another example. Consider we have a students table with the following data.

Student_ RollNo	Student_ Name	Student_ Gender	Student_ Mobile Number	Student_ Home Town	Student_ Age	Student_ Percentage
1	Rohit More	Male	9890786123	Lucknow	23	75
2	Kunal Shah	Male	7789056784	Chandigarh	20	92
3	Kartik Goenka	Male	9908743576	Ahemdabad	22	89
4	Anupama Shah	Female	8890907656	Chennai	24	92
5	Snehal Jain	Female	8657983476	Surat	21	94

Example 1:

Write a query to get the student name and the average of the percentage of the student under the temporary column name 'Student' and 'Student_Percentage', respectively.

Query:

1. **SELECT** Student_Name **AS** Student, AVG (Student_ Percentage) **AS** Average_Percentage **FROM** students;

Here, to calculate the average, we have used **AVG () function**. Further, the calculated average value of the percentage will be stored under the temporary name 'Average_Percentage'.

The result will be shown as this table:

Student	Average_Percentage
Rohit More	88.4000

Example 2:

Write a query to get the student roll number and the student mobile number under the temporary column name 'Roll No' and 'Mobile Number', respectively.

Query:

1. mysql> **SELECT** Student_RollNo **AS** 'Roll No', Student_PhoneNumber **AS** 'Mobile Number' **FROM** students;

The result will be shown as this table:

Roll No	Mobile Number
1	9890786123
2	7789056784
3	9908743576
4	8890907656
5	8657983476

Example 3:

Write a query to get the student roll number and the student phone number, home town under the temporary column name 'Roll No' and 'Student_Info', respectively.

Query:

- mysql> **SELECT** Student_RollNo **AS** 'Roll No', CONCAT (Student_PhoneNumber, ', ', Student_HomeTown) **AS** Student_Info **FROM** students;

Here, the **CONCAT () function** combines two different columns, student phone number and the home town, together in a single column. Further, the combined values of both these columns are stored under the temporarily assigned name 'Student_Info'.

The result will be shown as this table:

Roll No	Mobile Number
1	9890786123, Lucknow
2	7789056784, Chandigarh
3	9908743576, Ahemdabad
4	8890907656, Chennai
5	8657983476, Surat

9.3.2 Assigning a Temporary Name to a Table

Instead of remembering the table names, we can create an alias of them. We can assign a temporary name to the columns of a table; similarly, we can create an alias of a table.

Let's understand it with the help of an example.

Write a query to create an alias of a table named 'students'.

Query:

- mysql> **SELECT** s.Student_RollNo, s.Student_Name, s.Student_Gender, s.Student_PhoneNumber, s.Student_HomeTown **FROM** students **AS** s **WHERE** s.Student_RollNo = 3;

Here, 's' is the alias, i.e., the temporary name assigned to the 'students' table.

The result will be shown as this table:

Student_ RollNo	Student_ Name	Student_ Gender	Student_ MobileNumber	Student_ HomeTown
3	Kartik Goenka	Male	9908743576	Ahemdabad

9.4 HAVING CLAUSE IN SQL

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement.

This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

9.4.1 Difference between HAVING and WHERE Clause

The difference between the WHERE and HAVING clauses in the database is the most important question asked during an IT interview.

The following table shows the comparisons between these two clauses, but the main difference is that the WHERE clause uses condition for filtering records before any groupings are made, while HAVING clause uses condition for filtering values from a group.

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.
5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. We can implements this SQL clause in column operations.	6. We can implements this SQL clause in row operations.
7. It is a post-filter.	7. It is a pre-filter.
8. It is used to filter groups.	8. It is used to filter the single record of the table.

9.4.2 Syntax of HAVING Clause in SQL

- **SELECT** column_Name1, column_Name2, ..., column_NameN aggregate_function_name(column_Name) **FROM** table_name **GROUP BY** column_Name1 **HAVING** condition;

9.4.3 Examples of HAVING Clause in SQL

In this article, we have taken the following four different examples which will help you how to use the HAVING clause with different SQL aggregate functions:

Example 1: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with SUM aggregate function:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	2000	Goa
202	Ankit	4000	Delhi
203	Bheem	8000	Jaipur
204 Ram	2000	Goa	
205	Sumit	5000	Delhi

If you want to add the salary of employees for each city, you have to write the following query:

SELECT SUM(Emp_Salary), Emp_City **FROM** Employee **GROUP BY** Emp_City;

The output of the above query shows the following output:

SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

Now, suppose that you want to show those cities whose total salary of employees is more than 5000. For this case, you have to type the following query with the HAVING clause in SQL:

- **SELECT** SUM(Emp_Salary), Emp_City **FROM** Employee **GROUP BY** Emp_City **HAVING** SUM(Emp_Salary)>5000;

The output of the above SQL query shows the following table in the output:

SUM(Emp_Salary)	Emp_City
9000	Delhi
8000	Jaipur

Example 2: Let's take the following **Student_details** table, which helps you to analyze the HAVING clause with the COUNT aggregate function:

Roll_No	Name	Marks	Age
1	Rithik	91	20
2	Kapil	60	19
3	Arun	82	17
4	Ram	92	18
5	Anuj	50	20
6	Suman	88	18
7	Sheetal	57	19
8	Anuj	64	20

Suppose, you want to count the number of students from the above table according to their age. For this, you have to write the following query:

- **SELECT** COUNT(Roll_No), Age **FROM** Student_details **GROUP BY** Age ;

The above query will show the following output:

Count(Roll_No)	Age
3	20
2	19
1	17
2	18

Now, suppose that you want to show the age of those students whose roll number is more than and equals 2. For this case, you have to type the following query with the HAVING clause in SQL:

- **SELECT** COUNT(Roll_No), Age **FROM** Student_details **GROUP BY** Age **HAVING** COUNT(Roll_No) >= 2 ;

The output of the above SQL query shows the following table in the output:

Count(Roll_No)	Age
3	20
2	19
2	18

Example 3: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with MIN and MAX aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	9000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding
1005	Seenu	5000	HR
1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	8000	Finance

9.4.4 MIN Function with HAVING Clause

If you want to show each department and the minimum salary in each department, you have to write the following query:

- **SELECT MIN**(Emp_Salary), Emp_Dept **FROM** Employee **GROUP BY** Emp_Dept;

The output of the above query shows the following output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
4000	HR
3000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose minimum salary of employees is greater than 4000. For this case, you have to type the following query with the HAVING clause in SQL:

SELECT MIN(Emp_Salary), Emp_Dept **FROM** Employee **GROUP BY** Emp_Dept **HAVING MIN**(Emp_Salary) > 4000 ;

The above SQL query shows the following table in the output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
10000	Marketing

9.4.5 MAX Function with HAVING Clause

In the above employee table, if you want to list each department and the maximum salary in each department. For this, you have to write the following query:

- **SELECT MAX**(Emp_Salary), Emp_Dept **FROM** Employee **GROUP BY** Emp_Dept;

The above query will show the following output:

MAX(Emp_Salary)	Emp_Dept
9000	Finance
5000	HR
6000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose maximum salary of employees is less than 8000. For this case, you have to type the following query with the HAVING clause in SQL:

- **SELECT MAX**(Emp_Salary), Emp_Dept **FROM** Employee **GROUP BY** Emp_Dept **HAVING MAX**(Emp_Salary) < 8000 ;

The output of the above SQL query shows the following table in the output:

MAX(Emp_Salary)	Emp_Dept
5000	HR
6000	Coding

Example 4: Let's take the following **Employee_Dept** table, which helps you to analyze the HAVING clause with AVG aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	8000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding
1005	Seenu	5000	HR
1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	6000	Finance

If you want to find the average salary of employees in each department, you have to write the following query:

- **SELECT** AVG(Emp_Salary), Emp_Dept **FROM** Employee_Dept **GROUP BY** Emp_Dept;

The above query will show the following output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
4500	HR
6500	Coding
10000	Marketing

Now, suppose that you want to show those departments whose average salary is more than and equals 6500. For this case, you have to type the following query with the HAVING clause in SQL:

- **SELECT** AVG(Emp_Salary), Emp_Dept **FROM** Employee_Dept **GROUP BY** Emp_Dept **HAVING** AVG(Emp_Salary) > 6500 ;

The above SQL query will show the following table in the output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
6500	Coding
10000	Marketing

9.5 SQL ORDER BY CLAUSE

An ORDER BY clause in SQL specifies that a SQL SELECT statement returns a result set with the rows being sorted by the values of one or more columns. The sort criteria do not have to be included in the result set. The sort criteria can be expressions, including column names, user-defined functions, arithmetic operations, or CASE expressions. The expressions are evaluated and the results are used for the sorting, i.e., the values stored in the column or the results of the function call.

Remember

If an ordering is required, the ORDER BY must be provided in the SELECT statement sent by the application.

ORDER BY is the *only* way to sort the rows in the result set. Without this clause, the relational database system may return the rows in any order.

Although some database systems allow the specification of an ORDER BY clause in subselects or view definitions, the presence there has no effect. A view is a logical relational table, and the relational model mandates that a table is a set of rows, implying no sort order whatsoever. The only exception are constructs like ORDER BY ORDER OF ... (not standardized in SQL:2003) which allow the propagation of sort criteria through nested subselects.

The SQL standard's core functionality does not explicitly define a default sort order for Nulls. With the SQL:2003 extension T611, "Elementary OLAP operations", nulls can be sorted before or after all data values by using the NULLS FIRST or NULLS LAST clauses of the ORDER BY list, respectively. Not all DBMS vendors implement this functionality, however. Vendors who do not implement this functionality may specify different treatments for Null sorting in the DBMS.

Structure ORDER BY ... DESC will order in descending order, otherwise ascending order is used.

The SQL ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns.

Some databases sort query results in ascending order by default.

9.5.1 SQL ORDER BY Syntax

1. **SELECT** expressions
2. **FROM** tables
3. **WHERE** conditions
4. **ORDER BY** expression [**ASC** | **DESC**];

Let us take a CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Himani gupta	21	Modinagar	22000
2	Shiva tiwari	22	Bhopal	21000
3	Ajeet bhargav	45	Meerut	65000
4	Ritesh yadav	36	Azamgarh	26000
5	Balwant singh	45	Varanasi	36000
6	Mahesh sharma	26	Mathura	22000

This is an example that would sort the result in ascending order by NAME and SALARY.

1. **SELECT** * **FROM** CUSTOMERS
2. **ORDER BY NAME**, SALARY;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
3	Ajeet bhargav	45	Meerut	65000
5	Balwant singh	45	Varanasi	36000
1	Himani gupta	21	Modinagar	22000
6	Mahesh sharma	26	Mathura	22000
4	Ritesh yadav	36	Azamgarh	26000

Did You Know?

SQL:2003 is the fourth revision of the SQL database query language. The standard consists of 9 parts which are described in detail in SQL. It was updated by SQL:2006.

2	Shiva tiwari	22	Bhopal	21000

This is an example to sort the result in descending order by NAME.

- **SELECT** * **FROM** CUSTOMERS
- **ORDER BY NAME DESC**;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
2	Shiva tiwari	22	Bhopal	21000
4	Ritesh yadav	36	Azamgarh	26000
6	Mahesh sharma	26	Mathura	22000
1	Himani gupta	21	Modinagar	22000
5	Balwant singh	45	Varanasi	36000
3	Ajeet bhargav	45	Meerut	65000

9.5.2 SQL ORDER BY Clause with Ascending Order

This statement is used to sort data in ascending order. If you miss the ASC attribute, SQL ORDER BY query takes ascending order by default.

Let's take an example of supplier

1. **SELECT** supplier_city
2. **FROM** suppliers
3. **WHERE** supplier_name = 'IBM'
4. **ORDER BY** supplier_city;

Let us take a CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Himani gupta	21	Modinagar	22000
2	Shiva tiwari	22	Bhopal	21000
3	Ajeet bhargav	45	Meerut	65000
4	Ritesh yadav	36	Azamgarh	26000
5	Balwant singh	45	Varanasi	36000
6	Mahesh sharma	26	Mathura	22000

This is an example to sort the result in ascending order by NAME and SALARY.

Features of Java - Javatpoint

- **SELECT** * **FROM** CUSTOMERS
- **ORDER BY NAME**, SALARY;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
3	Ajeet bhargav	45	Meerut	65000
5	Balwant singh	45	Varanasi	36000
1	Himani gupta	21	Modinagar	22000
6	Mahesh sharma	26	Mathura	22000
4	Ritesh yadav	36	Azamgarh	26000
2	Shiva tiwari	22	Bhopal	21000

9.5.3 SQL ORDER BY Clause with Descending Order

This statement is used to sort data in descending order. You should use the DESC attribute in your ORDER BY clause as follows.

- **SELECT** supplier_city
- **FROM** suppliers
- **WHERE** supplier_name = 'IBM'
- **ORDER BY** supplier_city **DESC**;

Let's see an example of an employee table:

ID	NAME	AGE	ADDRESS	SALARY
1	Himani gupta	21	Modinagar	22000
2	Shiva tiwari	22	Bhopal	21000
3	Ajeet bhargav	45	Meerut	65000
4	Ritesh yadav	36	Azamgarh	26000
5	Balwant singh	45	Varanasi	36000
6	Mahesh sharma	26	Mathura	22000

This is an example to sort the result in descending order by NAME.

- **SELECT** * **FROM** CUSTOMERS
- **ORDER BY NAME DESC**;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
2	Shiva tiwari	22	Bhopal	21000
4	Ritesh yadav	36	Azamgarh	26000
6	Mahesh sharma	26	Mathura	22000
1	Himani gupta	21	Modinagar	22000
5	Balwant singh	45	Varanasi	36000
3	Ajeet bhargav	45	Meerut	65000

9.5.4 SQL ORDER BY RANDOM

If you want the resulting record to be ordered randomly, you should use the following codes according to several databases.

Here is a question: what is the need to fetch a random record or a row from a database?

Sometimes you may want to display random information like *articles, links, pages*, etc., to your user.

If you want to fetch random rows from any of the databases, you have to use some altered queries according to the databases.

- Select a random row with MySQL:

If you want to return a random row with MY SQL, use the following syntax:

1. **SELECT column FROM table ORDER BY** RAND () LIMIT 1;

- **Select a random row with Postgre SQL:**

1. **SELECT column FROM table ORDER BY** RANDOM () LIMIT 1;

- **Select a random row with SQL Server:**

1. **SELECT TOP** 1 **column FROM table ORDER BY** NEWID ();

- **Select a random row with oracle:**

1. **SELECT column FROM** (**SELECT column FROM table ORDER BY** dbms_ random.value) **WHERE** rownum = 1;

- **Select a random row with IBM DB2:**

1. **SELECT column** RAND () **as** IDX **FROM table ORDER BY** IDX **FETCH FIRST** 1

ROWS ONLY;

To understand this concept practically, let us see some examples using the MySQL database. Consider we have a table items created into the database with the following data:

Table: items

ID	Item_Name	Item_ Quantity	Item_Price	Purchase_ Date
1	Soap	5	200 2021-07-08	
2	Toothpaste	2	80	2021-07-10
3	Pen	10	50	2021-07-12
4	Bottle	1	250	2021-07-13
5	Brush	3	90	2021-07-15

Suppose we want to retrieve any random record from the items table.

We will write the query as follows:

- mysql> **SELECT** * **FROM** items **ORDER BY** RAND () LIMIT 1;

We may get the following results:

ID	Item_ Name	Item_ Quantity	Item_ Price	Purchase_ Date
3	Pen	10	20	2021-07-12

Now let us try executing the same query one more time.

1. mysql> **SELECT** * **FROM** items **ORDER BY** RAND () LIMIT 1;

We may get the following results:

ID	Item_ Name	Item_ Quantity	Item_ Price	Purchase_ Date
5	Brush	3	90	2021-07-15

From the above results, we can conclude that we get different records as output both times even though we executed the same query twice. RAND () function has selected random records both times for the same query from a single table. Therefore, even we execute the same query again, we will get different output every time. There is a rare possibility of getting the same record consecutively using the RAND () function.

Now, suppose you want all the records of the table to be fetched randomly.

To do so, we need to execute the following query:

- mysql> **SELECT** * **FROM** items **ORDER BY** RAND ();

We may get the following results:

ID	Item_Name	Item_ Quantity	Item_ Price	Purchase_ Date
4	Bottle	1	250	2021-07-13
5	Brush	3	90	2021-07-15
1	Soap	5	200	2021-07-08
2	Toothpaste	2	80	2021-07-10
3	Pen	10	50	2021-07-12

There is also a possibility of getting some different arrangements of records if we execute the RAND () function again on the employees table.

9.5.5 SQL ORDER BY LIMIT

We can retrieve limited rows from the database. I can be used in pagination where are forced to show only limited records like 10, 50, 100 etc.

LIMIT Clause for ORACLE SQL

If you want to use LIMIT clause with SQL, you have to use ROWNUM queries because it is used after result are selected.

You should use the following code:

1. **SELECT name**, age
2. **FROM**
3. (**SELECT name**, age, ROWNUM r
4. **FROM**
5. (**SELECT name**, age, **FROM** employee_data
6. **ORDER BY** age **DESC**
7.)
8. **WHERE** ROWNUM <=40

9.)

10. **WHERE** r >= 21;

This query will give you 21th to 40th rows.

9.5.6 SQL SORTING on Multiple Columns

Let's take an example of customer table which has many columns, the following SQL statement selects all customers from the table named "customer", stored by the "country" and "Customer-Name" columns:

1. **SELECT** * **FROM** customers
2. **ORDER BY** country, Customer-**Name**;

SUMMARY

- Clauses are in-built functions available to us in SQL. With the help of clauses, we can deal with data easily stored in the table.
- Clauses help us filter and analyze data quickly. When we have large amounts of data stored in the database, we use Clauses to query and get data required by the user.
- A WHERE clause in SQL is a data manipulation language statement. WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.
- The SQL OR condition is used in SQL query to create a SQL statement where records are returned when any one condition met. It can be used in a SELECT statement, INSERT statement, UPDATE statement or DELETE statement.
- The SQL WITH clause is used to provide a sub-query block which can be referenced in several places within the main SQL query. It was introduced by oracle in oracle 9i release2 database.
- The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.
- An ORDER BY clause in SQL specifies that a SQL SELECT statement returns a result set with the rows being sorted by the values of one or more columns. The sort criteria do not have to be included in the result set. The sort criteria can be expressions, including column names, user-defined functions, arithmetic operations, or CASE expressions.
- ORDER BY is the *only* way to sort the rows in the result set. Without this clause, the relational database system may return the rows in any order.
- The SQL ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns.

KNOWLEDGE CHECK

1. **If we have not specified ASC or DESC after a SQL ORDER BY clause, the following is used by default**
   a. DESC
   b. ASC
   c. There is no default value
   d. None of the mentioned
2. **Which of the following is true about the HAVING clause?**
   a. Similar to the WHERE clause but is used for columns rather than groups.
   b. Similar to WHERE clause but is used for rows rather than columns.
   c. Similar to WHERE clause but is used for groups rather than rows.
   d. Acts exactly like a WHERE clause.
3. **_____ clause creates temporary relation for the query on which it is defined.**
   a. WITH
   b. FROM
   c. WHERE
   d. SELECT
4. **Which of the following is true about the SQL AS clause?**
   a. The AS clause in SQL is used to change the column name in the output or assign a name to a derived column.
   b. The SQL AS clause can only be used with the JOIN clause.
   c. The AS clause in SQL is used to defines a search condition.
   d. All of the mentioned
5. **When the wildcard in a WHERE clause is useful?**
   a. When an exact match is required in a SELECT statement.
   b. When an exact match is not possible in a SELECT statement.
   c. When an exact match is required in a CREATE statement.
   d. When an exact match is not possible in a CREATE statement.

REVIEW QUESTIONS

1. What is the WHERE clause used for?
2. What are the difference between HAVING and WHERE clause?
3. What is order SQL?
4. How to select data from database in ascending order and descending order?
5. How to sort data on multiple columns?

Check Your Result

1. (b) 2. (c) 3. (a) 4. (a) 5. (b)

REFERENCES

1. Chatham, Mark (2012). Structured Query Language By Example - Volume I: Data Query Language. p. 8. ISBN 978-1-29119951-2.
2. Christian S. Jensen; Torben Bach Pedersen; Christian Thomsen (2010). Multidimensional Databases and Data Warehousing. Morgan & Claypool Publishers. p. 26. ISBN 978-1-60845-537-9.
3. Eisenberg, Andrew; et al. (March 2004). "SQL:2003 Has Been Published". SIGMOD Record. 33 (1): 119. doi:10.1145/974121.974142. Archived from the original (pdf) on 2007-11-11. Retrieved 2007-08-14.
4. Norbert E. Fuchs; Kaarel Kaljurand; Gerold Schneider (2006). "Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces" (PDF). FLAIRS 2006.

CHAPTER

10

DATABASE PROCESSING AND STORED PROCEDURAL SQL

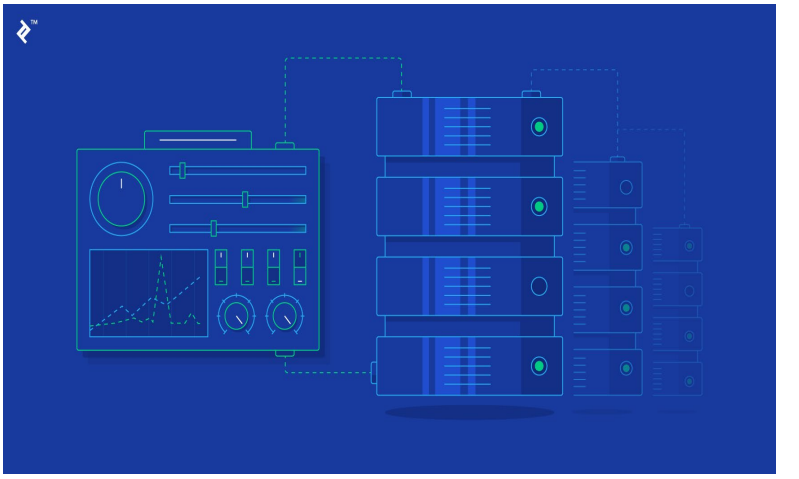
"Relational databases just could not meet the [data volume] requirements, so we turned to in-memory databases."

—Michael Gross

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Describe the procedural SQL concepts
2. Learn about triggers
3. Discuss about stored procedures, functions, triggers, and the SQL standard



INTRODUCTION

The long-term trend in the database market is for databases to take on a progressively larger role in the overall data

processing architecture. The pre-relational database systems basically handled only data storage and retrieval; application programs were responsible for navigating their way through the database, sorting and selecting data, and handling all processing of the data. With the advent of relational databases and SQL, the DBMS took on expanded responsibilities. Database searching and sorting were embodied in SQL language clauses and provided by the DBMS, along with the capability to summarize data. Explicit navigation through the database became unnecessary. Subsequent SQL enhancements such as primary key, foreign key (referential), and check constraints continued the trend, taking over data validation and data integrity functions that had remained the sole responsibility of application programs with earlier SQL implementations. At each step, having the DBMS take on more responsibility provided more centralized control and reduced the possibility of data corruption due to application programming errors.

In many information technology (IT) departments within large companies and organizations, this DBMS trend paralleled an organizational trend. The corporate database and the data it contains came to be viewed as a major corporate asset, and in many IT departments, a dedicated database administration (DBA) group emerged, with responsibility for maintaining the database, defining (and in some cases updating) the data it contained, and providing structured access to it. Other groups within the IT department, or elsewhere within the company, could develop application programs, reports, queries, or other logic that accessed the database. In most organizations, application programs, and the businesspeople using them, have had primary responsibility for updating the data within the database. However, the DBA group sometimes has had responsibility for updating reference (lookup) table data and for assisting with scripts and utilities to perform tasks such as the bulk loading of newly acquired data. But the security of the database, the permitted forms of access, and in general, everything within the realm of the database, became the province of the DBA.

Three important features of modern enterprise-scale relational databases—stored procedures, functions, and triggers—have been a part of this trend. Stored procedures can perform database-related application processing within the database itself. For example, a stored procedure might implement the application's logic to accept a customer order or to transfer money from one bank account to another. Functions are stored SQL programs that return only a single value for each row of data. Unlike stored procedures, functions are invoked by referencing them in SQL statements in almost any clause where a column name can be used. This makes them ideal for performing calculations and data transformations on data to be displayed in query results or used in search conditions. Nearly all relational DBMS products come with a set of vendor-supplied functions for general use, and therefore functions added by local database users are often called user-defined functions. Triggers are used to automatically invoke the processing capability of a stored procedure based on conditions that arise within the

database. For example, a trigger might automatically transfer funds from a savings account to a checking account if the checking account becomes overdrawn. This chapter describes the core concepts behind stored procedures, functions, and triggers, and their implementation in several popular DBMS brands.

10.1 PROCEDURAL SQL CONCEPTS

In its original form, SQL was not envisioned as a complete programming language. It was designed and implemented as a language for expressing database operations—creating database structures, entering data into the database, updating database data—and especially for expressing database queries and retrieving the answers. SQL could be used interactively by typing SQL statements at a keyboard, one by one. In this case, the sequence of database operations was determined by the human user. SQL could also be embedded within another programming language, such as COBOL or C. In this case, the sequence of database operations was determined by the flow of control within the COBOL or C program.

With stored procedural SQL, the SQL language is extended with several capabilities normally associated with programming languages. Sequences of extended SQL statements are grouped together to form SQL programs or procedures. (For simplicity, we refer to stored procedures, functions, and triggers collectively as SQL procedures.) The specifics vary from one implementation to another, but generally, these capabilities are provided:

- **Conditional execution** An IF…THEN…ELSE structure allows a SQL procedure to test a condition and to carry out different operations depending on the result.
- **Looping** A WHILE or FOR loop or similar structure allows a sequence of SQL operations to be performed repeatedly, until some terminating condition is met. Some implementations provide a special cursor-based looping structure to process each row of query results.
- **Block structure** A sequence of SQL statements can be grouped into a single block and used in other flow-of-control constructs as if the statement block were a single statement.

Did You Know?

COBOL was designed in 1959 by CODASYL and was partly based on the programming language FLOW-MATIC designed by Grace Hopper. It was created as part of a US Department of Defense effort to create a portable programming language for data processing.

- **Named variables** A SQL procedure may store a value that it has calculated, retrieved from the database, or derived in some other way into a program variable, and later retrieve the stored value for use in subsequent calculations.
- **Named procedures** A sequence of SQL statements may be grouped together, given a name, and assigned formal input and output parameters, like a subroutine or function in a conventional **programming language**. Once defined in this way, the procedure may be called by name, passing it appropriate values for its input parameters. If the procedure is a function returning a value, it may be used in SQL value expressions.

Collectively, the structures that implement these capabilities form a stored procedural language (SPL).

Keyword

Programming language is a formal language comprising a set of strings that produce various kinds of machine code output.

Stored procedures were first introduced by Sybase in the original Sybase SQL Server product. Much of the original enthusiasm for stored procedures was because of their performance advantages in a client/server database architecture. Without stored procedures, every SQL operation requested by an application program (running on the client computer system) would be sent across the network to the database server and would wait for a reply message to be returned across the network. If a logical transaction required six SQL operations, six network round trips were required. With stored procedures, the sequence of six SQL operations could be programmed into a procedure and stored in the database. The application program would simply request the execution of the stored procedure and await the results. In this way, six network round trips could be cut to one round trip—the request and reply for executing the stored procedure.

Stored procedures proved to be a natural fit for the client/server model, and Sybase used them to establish an early lead with this architecture. A competitive response quickly followed from many of the other DBMS vendors. Today, most enterprise DBMS products provide a stored procedure capability, and the benefits of stored procedures in corporate

databases has expanded considerably beyond the early focus on network performance. Stored procedures are less relevant for other types of specialized DBMS systems, such as data warehousing systems or in-memory databases. Some DBMS products have modeled their SPL structures on C or Pascal language constructs. Others have tried to match the style of the SQL Data Manipulation Language (DML) and **Data Definition Language (DDL)** statements. Oracle, on the other hand, based its SPL (PL/SQL) on the Ada programming language, because it was the standard language of its large U.S. government customers. While stored procedure concepts are very similar from one SQL dialect to another, the specific syntax varies considerably.

Keyword

Data definition language (DDL) is a syntax for creating and modifying database objects such as tables, indices, and users.

10.1.1 A Basic Example

It's easiest to explain the basics of stored procedures through an example. Consider the process of adding a customer to the sample database. Here are the steps that may be involved:

- Obtain the customer number, name, credit limit, and target sales amount for the customer, as well as the assigned salesperson and office.
- Add a row to the customer table containing the customer's data.
- Update the row for the assigned salesperson, raising the quota target by the specified amount.
- Update the row for the office, raising the sales target by the specified amount.
- Commit the changes to the database, if all previous statements were successful.

Without a stored procedure capability, here is a SQL statement sequence that does this work for XYZ Corporation, new customer number 2137, with a credit limit of $30,000 and first-year target sales of $50,000 to be assigned to Paul Cruz (employee #103) of the Chicago office:

```
INSERT INTO CUSTOMERS (CUST_NUM, COMPANY, CUST_REP, CREDIT_LIMIT)
       VALUES (2137, 'XYZ Corporation', 103, 30000.00);

UPDATE SALESREPS
   SET QUOTA = QUOTA + 50000.00
 WHERE EMPL_NUM = 103;

UPDATE OFFICES
   SET TARGET = TARGET + 50000.00
 WHERE CITY = 'Chicago';

COMMIT;
```

With a stored procedure, all of this work can be embedded into a single defined SQL routine. Figure 1 shows a stored procedure for this task, expressed in Oracle's PL/SQL stored procedure dialect. The procedure is named ADD_CUST, and it accepts six parameters—the customer name, number, credit limit, and target sales, the employee number of the assigned salesperson, and the city where the assigned sales office is located. Once this procedure has been created in the database, a statement like this one:

```
ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago');
```

calls the stored procedure and passes it the six specified values as its parameters. The DBMS executes the stored procedure, carrying out each SQL statement in the procedure definition one by one. If the ADD_CUST procedure completes its execution successfully, a committed transaction has been carried out within the DBMS. If not, the returned error code and message indicates what went wrong.

```
/* Add a customer procedure */
create procedure add_cust (
   c_name   in varchar2,          /* input customer name */
   c_num    in integer,           /* input customer number */
   cred_lim in number,            /* input credit limit */
   tgt_sls  in number,            /* input target sales */
   c_rep    in integer,           /* input salesrep emp # */
   c_offc   in varchar2)          /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);
```

```
    /* Update row of SALESREPS table */
    update salesreps
       set quota = quota + tgt_sls
     where empl_num = c_rep;

    /* Update row of OFFICES table */
    update offices
       set target = target + tgt_sls
     where city = c_offc;

    /* Commit transaction and we are done */
    commit;
end;
```

Figure 1. A basic stored procedure in PL/SQL.

10.1.2 Using Stored Procedures

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system (RDBMS) as a group, so it can be reused and shared by multiple programs.

Stored procedures can access or modify data in a database, but it is not tied to a specific database or object, which offers a number of advantages.

The procedure defined in Figure 1 illustrates several of the basic structures common to all SPL dialects. Nearly all dialects use a CREATE PROCEDURE statement to initially define a stored procedure. A corresponding DROP PROCEDURE statement is used to discard procedures that are no longer needed. The CREATE PROCEDURE statement defines the following:

- The name of the stored procedure
- The number and data types of its parameters
- The names and data types of any local variables used by the procedure
- The sequence of statements executed when the procedure is called

The following sections describe these elements and the special SQL statements that are used to control the flow of execution within the body of a stored procedure.

Creating a Stored Procedure

In many common SPL dialects, the CREATE PROCEDURE statement is used to create a stored procedure and to specify how it operates. The CREATE PROCEDURE statement assigns the newly defined procedure a name, which is used to call it. The name must typically follow the rules for SQL identifiers. (The procedure in Figure 1 is named ADD_CUST.) A stored procedure accepts zero or more parameters as its arguments. (This one has six parameters: C_NAME, C_NUM, CRED_LIM, TGT_SLS, C_REP, and C_OFFC.) In all of the common SPL dialects, the values for the parameters appear in a comma-separated list, enclosed in parentheses, following the procedure name when the procedure is called. The header of the stored procedure definition specifies the names of the parameters and their data types. The same SQL data types supported by the DBMS for columns within the database can be used as parameter data types.

In Figure 1, all of the parameters are input parameters (signified by the IN keyword in the procedure header in the Oracle PL/SQL dialect). When the procedure is called, the parameters are assigned the values specified in the procedure call, and the statements in the procedure body begin to execute. The parameter names may appear within the procedure body (and particularly within standard SQL statements in the procedure body) anywhere that a constant may appear. When a parameter name appears, the DBMS uses its current value. In Figure 1, the parameters are used in the INSERT statement and the UPDATE statement, both as data values to be used in column calculations and as search conditions. In addition to input parameters, some SPL dialects also support output parameters.

These allow a stored procedure to pass back values that it calculates during its execution. Output parameters provide an important capability for passing back information from one stored procedure to another stored procedure that calls it, and can also be useful for debugging stored procedures using interactive SQL. Some SPL dialects support parameters that operate as both input and output parameters. In this case, the parameter passes a value to the stored procedure, and any changes to the value during the procedure execution are reflected in the calling procedure.

Figure 2 shows the same ADD_CUST procedure definition, expressed in the Sybase Transact-SQL dialect. (The Transact-SQL dialect is also used by Microsoft SQL Server; its basics are largely unchanged since the original Sybase SQL Server version, which was the foundation for both the Microsoft and Sybase product lines.) Note the differences from the Oracle dialect:

- The keyword PROCEDURE can be abbreviated to PROC.
- No parenthesized list of parameters follows the procedure name. Instead, the parameter declarations immediately follow the name of the stored procedure.
- The parameter names all begin with an "at" sign (@), both when they are declared at the beginning of the procedure and when they appear within SQL statements in the procedure body.

- There is no formal end-of-procedure body marker. Instead, the procedure body is a single Transact-SQL statement. If more than one statement is needed, the TransactSQL block structure is used to group the statements.

```
/* Add a customer procedure */
create proc add_cust
   @c_name   varchar(20),              /* input customer name */
   @c_num    integer,                  /* input customer number */
   @cred_lim decimal(9,2),             /* input credit limit */
   @tgt_sls  decimal(9,2),             /* input target sales */
   @c_rep    integer,                  /* input salesrep emp # */
   @c_offc   varchar(15)               /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (@c_num, @c_name, @c_rep, @cred_lim)


   /* Update row of SALESREPS table */
   update salesreps
      set quota = quota + quota + @tgt_sls
    where empl_num = @c_rep

   /* Update row of OFFICES table */
   update offices
      set target = target + @tgt_sls
    where city = @c_offc

   /* Commit transaction and we are done */
   commit trans
end
```

Figure 2. The ADD_CUST stored procedure in Transact-SQL.

Figure 3 shows the ADD_CUST procedure again, this time expressed in the Informix stored procedure dialect. The declaration of the procedure head itself and the parameters more closely follow the Oracle dialect. Unlike the Transact-SQL example, the local variables and parameters use ordinary SQL identifiers as their names, without

any special identifying symbols. The procedure definition is formally ended with an END PROCEDURE clause, which makes the syntax less error-prone.

In all dialects that use the CREATE PROCEDURE statement, the procedure can be dropped when no longer needed by using a corresponding DROP PROCEDURE statement:

```
DROP PROCEDURE ADD_CUST;

/* Add a customer procedure */
create procedure add_cust (
   c_name   varchar(20),              /* input customer name */
   c_num    integer,                  /* input customer number */
   cred_lim numeric(16,2),            /* input credit limit */
   tgt_sls  numeric(16,2),            /* input target sales */
   c_rep    integer,                  /* input salesrep emp # */
   c_offc   varchar(15))              /* input office city */

   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);


   /* Update row of SALESREPS table */
   update salesreps
      set quota = quota + quota + tgt_sls
    where empl_num = c_rep;

   /* Update row of OFFICES table */
   update offices
      set target = target + tgt_sls
    where city = c_offc;

   /* Commit transaction and we are done */
   commit work;
end procedure;
```

Figure 3. The ADD_CUST stored procedure in Informix SPL.

3G E-LEARNING

Calling a Stored Procedure

Once defined by the CREATE PROCEDURE statement, a stored procedure can be used. An application program may request execution of the stored procedure, using the appropriate SQL statement. Another stored procedure may call it to perform a specific function. The stored procedure may also be invoked through an interactive SQL interface.

The various SQL dialects differ in the specific syntax used to call a stored procedure. Here is a call to the ADD_CUST procedure in the PL/SQL dialect:

```
EXECUTE ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103,
                 'Chicago');
```

The values to be used for the procedure's parameters are specified, in order, in a list that is enclosed by parentheses. When called from within another procedure or a **trigger**, the EXECUTE statement may be omitted, and the call becomes simply:

```
ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago');
```

The procedure may also be called using named parameters, in which case the parameter values can be specified in any sequence. Here is an example:

```
EXECUTE ADD_CUST (c_name = 'XYZ Corporation',
                  c_num = 2137,
                  cred_lim = 30000.00,
                  c_offc = 'Chicago',
                  c_rep = 103,
                  tgt_sales = 50000.00);
```

In the Transact-SQL dialect, the call to the stored procedure becomes

```
EXECUTE ADD_CUST 'XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago';
```

The parentheses aren't required, and the values to be used for parameters again form a comma-separated list. The keyword EXECUTE can be abbreviated to EXEC, and the parameter names can be explicitly specified in the call, allowing you to specify the parameter values in any order you wish. Here is an alternative, equivalent Transact-SQL call to the ADD_CUST stored procedure:

Keyword

Trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database.

```
EXEC ADD_CUST @C_NAME = 'XYZ Corporation',
              @C_NUM = 2137,
           @CRED_LIM = 30000.00,
             @C_OFFC = 'Chicago',
              @C_REP = 103,
           @TGT_SLS = 50000.00;
```

The Informix SPL form of the same EXECUTE command is

```
EXECUTE PROCEDURE ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00,
                           103, 'Chicago');
```

Again, the parameters are enclosed in a comma-separated, parenthesized list. This form of the EXECUTE statement may be used in any context. For example, it may be used by an embedded SQL application program to invoke a stored procedure. Within a stored procedure itself, another stored procedure can be called using this equivalent statement:

```
CALL ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago');
```

Stored Procedure Variables

In addition to the parameters passed into a stored procedure, it's often convenient or necessary to define other variables to hold intermediate values during the procedure's execution. All stored procedure dialects provide this capability. Usually, the variables are declared at the beginning of the procedure body, just after the procedure header and before the list of SQL statements. The data types of the variables can be any of the SQL data types supported as column data types by the DBMS.

Remember

Transact-SQL local variable names, like parameter names, begin with an "@" sign.

Figure 4 shows a simple Transact-SQL stored procedure fragment that computes the total outstanding order amount for a specific customer number, and sets up one of two messages depending on whether the total order amount is under $30,000.

```
/* Check order total for a customer */
create proc chk_tot
   @c_num integer      /* one input parameter */
as

   /* Declare two local variables */
   declare @tot_ord money, @msg_text varchar(30)

   begin
      /* Calculate total orders for customer */
      select @tot_ord = sum(amount)
        from orders
       where cust = @c_num

      /* Load appropriate message, based on total */
      if tot_ord < 30000.00
         select @msg_text = "high order total"
      else
         select @msg_text = "low order total"

      /* Do other processing for message text */
      . . .

   end
```

Figure 4. Using local variables in Transact-SQL

The DECLARE statement declares the local variables for this procedure. In this case, there are two variables: one with the MONEY data type and one VARCHAR.

In Transact-SQL, the SELECT statement assumes the additional function of assigning values to variables. A simple form of this use of SELECT is the assignment of the message text:

```
SELECT @MSG_TEXT = "high order total";
```

The assignment of the total order amount at the beginning of the procedure body is a more complex example, where the SELECT is used both to assign a value and as the introducer of the query that generates the value to be assigned.

Figure 5 shows the Informix SPL version of the same stored procedure. There are several differences from the Transact-SQL version:

- Local variables are declared using the DEFINE statement. This example shows only a very limited subset of the options that are available.

- Variable names are ordinary SQL identifiers; there is no special first character.
- A specialized SELECT…INTO statement is used within SPL to assign the results of a singleton SELECT statement into a local variable.
- The LET statement provides simple assignment of variable values.

```
/* Check order total for a customer */
create procedure chk_tot (c_num integer)

   /* Declare two local variables */
   define tot_ord  numeric(16,2);
   define msg_text varchar(30);

   /* Calculate total orders for requested customer */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* Load appropriate message, based on total */
   if tot_ord < 30000.00
      let msg_text = "high order total"
   else
      let msg_text = "low order total"

   /* Do other processing for message text */
   . . .

end procedure;
```

Figure 5. Using local variables in Informix SPL.

Figure 6 shows the Oracle PL/SQL version of the same stored procedure. Again, there are several differences to note from the Transact-SQL and Informix SPL examples:

- The SELECT...INTO statement has the same form as the Informix procedure; it is used to select values from a single-row query directly into local variables.
- The assignment statements use Pascal-style (:=) notation instead of a separate LET statement.

```
/* Check order total for a customer */
create procedure chk_tot (c_num in number)
as
   /* Declare two local variables */
   tot_ord  number(16,2);
   msg_text varchar(30);

begin
   /* Calculate total orders for requested customer */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* Load appropriate message, based on total */
   if tot_ord < 30000.00 then
      msg_text := 'high order total';
   else
      msg_text := 'low order total';
   end if;

   /* Do other processing for message text */
   . . .

end;
```

Figure 6. Using local variables in Oracle PL/SQL.

Local variables within a stored procedure can be used as a source of data within SQL expressions anywhere that a constant may appear. The current value of the variable is used in the execution of the statement. In addition, local variables may be destinations for data derived from SQL expressions or queries, as shown in the preceding examples.

Statement Blocks

In all but the very simplest stored procedures, it is often necessary to group a sequence of SQL statements together so that they will be treated as if they were a single statement. For example, in the IF…THEN…ELSE structure typically used to control the flow of execution within a stored procedure, most stored procedure dialects expect a single statement following the THEN keyword. If a procedure needs to perform a sequence of several SQL statements when the tested condition is true, it must group the statements together as a statement block, and this block will appear after THEN.

In Transact-SQL, a statement block has this simple structure:

```
/* Transact-SQL block of statements */
begin
   /* Sequence of SQL statements appears here */
   . . .
end
```

The sole function of the BEGIN…END pair is to create a statement block; they do not impact the scope of local variables or other database objects. The Transact-SQL procedure definition, conditional execution, and looping constructs, and others, are all designed to operate with single SQL statements, so statement blocks are frequently used in each of these contexts to group statements together as a single unit.

In Informix SPL, a statement block includes not only a statement sequence, but also may optionally declare local variables for use within the block and exception handlers to handle errors that may occur within the block. Here is the structure of an Informix SQL statement block:

```
/* Informix SPL block of statements */
/* Declaration of any local variables */
define . . .

/* Declare handling for exceptions */
on exception . . .

/* Define the sequence of SQL statements */
begin . . .

end
```

The variable declaration section is optional; we have already seen an example of it in the Informix stored procedure body in Figure 5. The exception-handling section is also optional; its role is described later in the "Handling Error Conditions" section. The BEGIN… END sequence performs the same function as it does for Transact-SQL. Informix also allows a single statement to appear in this position if the block consists of just the other two components and a single SQL or SPL statement.

The Informix SQL structures don't require the use of statement blocks as often as the Transact-SQL structures. In the Informix dialect, the looping conditional execution statements each include an explicit termination (IF…END IF, WHILE…END WHILE, FOR… END FOR). Within the structure, a single SQL statement or a sequence of statements (each ending with a semicolon) may appear. As a result, an explicit block structure is not always needed simply to group together a sequence of SQL statements.

The Oracle PL/SQL block structure has the same capabilities as the Informix structure. It offers the capability to declare variables and exception conditions, using this format:

```
/* Oracle PL/SQL statement block */
/* Declaration of any local variables */
declare . . .

/* Specify the sequence of statements */
begin . . .

/* Declare handling for exceptions */
exception . . .

end;
```

All three sections of the block structure are optional. It's common to see the structure used with only the BEGIN…END sequence to define a statement sequence, or with a DECLARE…BEGIN…END sequence to declare variables and a sequence of statements. As with Informix, the Oracle structures that specify conditional execution and looping have a self-defining end-of-statement marker, so sequences of statements within these structures do not necessarily need an explicit BEGIN…END statement block structure.

Functions

In addition to stored procedures, most SPL dialects support a stored function capability. The distinction is that a function returns a single thing (such as a data value, an object, or an XML document) each time it is invoked, while a stored procedure can return many things or nothing at all. Support for returned values varies by SPL dialect. Functions are commonly used as column expressions in SELECT statements, and thus are invoked once per row in the result set, allowing the function to perform calculations, data conversion, and other processes to produce the returned value for the column. Following is a simple example of a stored function. Assume you want to define a stored procedure that, given a customer number, calculates the total current order amount for that customer. If you define the SQL procedure as a function, the total amount can be returned as its value.

Figure 7 shows an Oracle function that calculates the total amount of current orders for a customer, given the customer number.

Remember

The RETURN clause in the procedure definition, which tells the DBMS the data type of the value being returned.

In most DBMS products, if you enter a function call via the interactive SQL capability, the function value is displayed in response. Within a stored procedure, you can call a stored function and use its return value in calculations or store it in a variable.

```
SELECT COMPANY, NAME
  FROM CUSTOMERS, SALESREPS
 WHERE CUST_REP = EMPL_NUM
   AND GET_TOT_ORDS(CUST_NUM) > 10000.00;
```

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in number)
                return number
as

/* Declare one local variable to hold the total */
tot_ord number(16,2);

begin
   /* Simple single-row query to get total */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* return the retrieved value as fcn value */
   return tot_ord;
end;
```

Figure 7. An Oracle PL/SQL function.

Keyword

User-defined function is a function provided by the user of a program or environment, in a context where the usual assumption is that functions are built into the program or environment.

Many SPL dialects also allow you to use a function as a **user-defined function** within SQL value expressions. This is true of the Oracle PL/SQL dialect, so this use of the function defined in Figure 7 within a search condition is legal.

As the DBMS evaluates the search condition for each row of prospective query results, it uses the customer number of the current candidate row as an argument to the GET_TOT_ORDS function and checks to see if it exceeds the $10,000 threshold. This same query could be expressed as a grouped query, with the ORDERS table also included in the FROM clause, and the results grouped by customer and salesperson. In many implementations, the DBMS carries out the grouped query more efficiently than the preceding one, which probably forces the DBMS to process the orders table once for each customer.

Figure 8 shows the Informix SPL definition for the same stored function shown in Figure 7. Except for stylistic variations, it differs very little from the Oracle version.

The Transact-SQL dialect used in Microsoft SQL Server and Sybase Adaptive Server Enterprise (ASE) has a stored (user-defined) function capability similar to the one illustrated in Figures 7 and 8.

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in integer)
       returning numeric(16,2)

/* Declare one local variable to hold the total */
define tot_ord numeric(16,2);

begin
   /* Simple single-row query to get total */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* Return the retrieved value as fcn value */
   return tot_ord;
end function;
```

Figure 8. An Informix SPL function.

Keyword

PL/SQL is Oracle Corporation's procedural extension for SQL and the Oracle relational database.

Returning Values via Parameters

Functions provide only the ability to return a single thing from a stored routine. Several stored procedure dialects provide a method for returning more than one value (or other thing), by passing the values back to the calling routine through output parameters. The output parameters are listed in the stored procedure's parameter list, just like the input parameters seen in the previous examples. However, instead of being used to pass data values into the stored procedure when it is called, the output parameters are used to pass data back out of the stored procedure to the calling procedure.

Figure 9 shows a **PL/SQL** stored procedure to retrieve the name of a customer, his or her salesperson, and the sales office to which the customer is assigned, given a supplied customer

number. The procedure has four parameters. The first one, CNUM, is an input parameter and supplies the requested customer number. The other three parameters are output parameters, used to pass the retrieved data values back to the calling procedure.

```
/* Get customer name, salesrep, and office */
create procedure get_cust_info(c_num  in  number,
                               c_name out varchar,
                               r_name out varchar,
                               c_offc out varchar)
as
begin
   /* Simple single-row query to get info */
   select company, name, city
     into c_name, r_name, c_offc
     from customers, salesreps, offices
    where cust_num = c_num
      and empl_num = cust_rep
      and office = rep_office;
end;
```

Figure 9. PL/SQL stored procedure with output parameters

In this simple example, the SELECT...INTO form of the query places the returned variables directly into the output parameters. In a more complex stored procedure, the returned values might be calculated and placed into the output parameters with a PL/SQL assignment statement.

When a stored procedure with output parameters is called, the value passed for each output parameter must be an acceptable target that can receive the returned data value. The target may be a local variable, for example, or a parameter of a higher-level procedure that is calling a lower-level procedure to do some work for it. Here is an Oracle PL/SQL anonymous (unnamed) block that makes an appropriate call to the GET_CUST_INFO procedure in Figure 9:

```
/* Get the customer info for customer 2111 */
declare the_name varchar(20);
        the_rep  varchar(15);
        the_city varchar(15);
execute get_cust_info(2111, the_name, the_rep, the_city);
```

Of course, it would be unusual to call this procedure with a literal customer number, but it's perfectly legal since that is an input parameter. The remaining three parameters

have acceptable data assignment targets (in this case, they are PL/SQL variables) passed to them so that they can receive the returned values. The following call to the same procedure is illegal because the second parameter is an output parameter and thus cannot receive a literal value:

```
/* Get the customer info for customer 2111 */
execute get_cust_info(2111, 'XYZ Co', the_rep, the_city)
```

In addition to input and output parameters, Oracle allows you to specify procedure parameters that are both input and output (INOUT) parameters. They must obey the same previously cited restrictions for output parameters, but in addition, their values are used as input by the procedure.

Figure 10 shows a version of the GET_CUST_INFO procedure defined in the TransactSQL dialect. The way in which the output parameters are identified in the procedure header differs slightly from the Oracle version, variable names begin with the "@" sign, and the single-row SELECT statement has a different form. Otherwise, the structure of the procedure and its operation are identical to the Oracle example.

```
/* Get customer name, salesrep, and office */
create procedure get_cust_info(@c_num  integer,
                               @c_name varchar(20) out,
                               @r_name varchar(15) out,
                               @c_offc varchar(15) out)
as
begin
   /* Simple single-row query to get info */
   select @c_name = company,
          @r_name = name,
          @c_offc = city
     from customers, salesreps, offices
    where cust_num = @c_num
      and empl_num = cust_rep
      and office = rep_office;
end
```

Figure 10. Transact-SQL stored procedure with output parameters.

When this procedure is called from another Transact-SQL procedure, the fact that the second, third, and fourth parameters are output parameters must be indicated in the call to the procedure, as well as in its definition. Here is the Transact-SQL syntax for calling the procedure in Figure 10:

```
/* Get the customer info for customer 2111 */
declare the_name varchar(20);
declare the_rep  varchar(15);
declare the_city varchar(15);
exec get_cust_info @c_num = 2111,
                   @c_name = the_name output,
                   @r_name = the_rep output,
                   @c_offc = the_city output;
```

Figure 11 shows the Informix SPL version of the same stored procedure example. Informix takes a different approach to handling multiple return values. Instead of output parameters, Informix extends the definition of a stored function to allow multiple return values. Thus, the GET_CUST_INFO procedure becomes a function for the Informix dialect. The multiple return values are specified in the RETURNING clause of the procedure header, and they are actually returned by the RETURN statement.

```
/* Get customer name, salesrep, and office */
create function get_cust_info(c_num integer)
       returning varchar(20), varchar(15), varchar(15)

   define c_name varchar(20);
   define r_name varchar(15);
   define r_name varchar(15);

   /* Simple single-row query to get info */
   select company, name, city
     into cname, r_name, c_offc
     from customers, salesreps, offices
    where cust_num = c_num
      and empl_num = cust_rep
      and office = rep_office;

   /* Return the three values */
   return cname, r_name, c_offc;
end procedure;
```

Figure 11. Informix stored function with multiple return values

The Informix CALL statement that invokes the stored function uses a special RETURNING clause to receive the returned values:

```
/* Get the customer info for customer 2111 */
define the_name varchar(20);
define the_rep  varchar(15);
define the_city varchar(15);
call get_cust_info (2111)
        returning the_name, the_rep, the_city;
```

As in the Transact-SQL dialect, Informix also allows a version of the CALL statement that passes the parameters by name:

```
call get_cust_info (c_num = 2111)
         returning the_name, the_rep, the_city;
```

Conditional Execution

One of the most basic features of stored procedures is an IF…THEN…ELSE construct for decision making within the procedure. Look back at the original ADD_CUST procedure defined in Figure 1 for adding a new customer. Suppose that the rules for adding new customers are modified so that there is a cap on the amount by which a salesperson's quota should be increased for a new customer. If the customer's anticipated first-year orders are $20,000 or less, that amount should be added to the quota, but if they are more than $20,000, the quota should be increased by only $20,000. Figure 12 shows a modified procedure that implements this new policy. The IF…THEN…ELSE logic operates exactly as it does in any conventional programming language.

```
/* Add a customer procedure */
create procedure add_cust (
   c_name     in varchar2,    /* input customer name */
   c_num      in number,      /* input customer number */
   cred_lim   in number,      /* input credit limit */
   tgt_sls    in number,      /* input target sales */
   c_rep      in number,      /* input salesrep empl # */
   c_offc     in varchar2)    /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);
```

```
      if tgt_sales <= 20000.00
      then
         /* Update row of SALESREPS table */
         update salesreps
            set quota = quota + quota + tgt_sls
          where empl_num = c_rep;
      else
         /* Update row of SALESREPS table */
         update salesreps
            set quota = quota + quota + 20000.00
          where empl_num = c_rep;
      end if;


   /* Update row of OFFICES table */
   update offices
      set target = target + tgt_sls
    where city = c_offc;

   /* Commit transaction and we are done */
   commit;
end;
```

Figure 12. Conditional logic in a stored procedure.

All of the stored procedure dialects allow nested IF statements for more complex decision making. Several provide extended conditional logic to streamline multiway branching. For example, suppose you wanted to do three different things within the ADD_ CUST stored procedure, depending on whether the customer's anticipated first-year orders are under $20,000, between $20,000 and $50,000, or over $50,000. In Oracle's PL/SQL, you could express the three-way decision this way:

3G E-LEARNING

```
/* Process sales target by range */
if tgt_sls < 20000.00
   then
   /* Handle low-target customers here */
   . . .
elsif tgt_sls <= 50000.00
   then
   /* Handle mid-target customers here */
   . . .
else
   /* Handle high-target customers here */
   . . .
end if;
```

In the Informix dialect, the same multiway branch structure is supported. The keyword ELSIF becomes ELIF, but all other aspects remain the same.

Repeated Execution

Another feature common to almost all stored procedure dialects is a construct for repeated execution of a group of statements (looping). Depending on the dialect, there may be support for Basic-style FOR loops (where an integer loop control value is counted up or counted down) or for C-style WHILE loops, with a test condition executed at the beginning or end of the loop.

In the sample database, it's hard to come up with an uncontrived example of simple loop processing. Assume you want to process some group of statements repeatedly, while the value of a loop-control variable, named ITEM_NUM, ranges from 1 to 10. Here is an Oracle PL/SQL loop that handles this situation:

```
/* Process each of ten items */
for item_num in 1..10 loop
   /* Process this particular item */
   . . .

   /* Test whether to end the loop early */
   exit when (item_num = special_item);
end loop;
```

The statements in the body of the loop are normally executed ten times, each time with a larger integer value of the ITEM_NUM variable. The EXIT statement provides the capability to exit an Oracle PL/SQL loop early. It can be unconditional, or it can be used with a built-in test condition, as in this example.

Here is the same loop structure expressed in Informix SPL, showing some of its additional capabilities and the dialectic differences from PL/SQL:

```
/* Process each of ten items */
for item_num = 1 to 10 step 1
   /* Process this particular item */
   . . .

   /* Test whether to end the loop early */
   if (item_num = special_item)
      then exit for;
end for;
```

The other common form of looping is when a sequence of statements is executed repeatedly while a certain condition exists or until a specified condition exists. Here is an Oracle PL/SQL loop construct that repeats indefinitely. Such a loop must, of course, provide a test within the body of the loop that detects a loop-terminating condition (in this case, a match of two variable values) and that explicitly exits the loop:

```
/* Repeatedly process some data */
loop
   /* Do some kind of processing each time */
   . . .

   /* Test whether to end the loop early */
   exit when (test_value = exit_value);
end loop;
```

A more common looping construct is one that builds the test into the loop structure itself. The loop is repeatedly executed as long as the test is true. For example, suppose you want to reduce targets for the offices in the sample database until the total of the targets is less than $24 million. Each office's target is to be reduced by the same amount, which should be a multiple of $10,000. Here is a (not very efficient) Transact-SQL stored procedure loop that gradually lowers office targets until the total is below the threshold:

```
/* Lower targets until total below $2,400,000 */
while (select sum(target) from offices) < 2400000.00
   begin
      update offices
         set target = target - 10000.00
   end;
```

The BEGIN…END block in this WHILE loop isn't strictly necessary, but most TransactSQL WHILE loops include one. Transact-SQL repeats the single SQL statement following the test condition as the body of the WHILE loop. If the body of the loop

consists of more than one statement, you must use a BEGIN…END block to group the statements.

Here is the Oracle PL/SQL version of the same loop:

```
/* Lower targets until total below $2,400,000 */
select sum(target) into total_tgt from offices;
while (total_tgt < 2400000.00)
loop
   update offices
      set target = target - 10000.00;
   select sum(target) into total_tgt from offices;
end loop;
```

The subquery-style version of the SELECT statement from Transact-SQL has been replaced by the PL/SQL SELECT...INTO form of the statement, with a local variable used to hold the total of the office targets. Each time the loop is executed, the OFFICES table is updated, and then the total of the targets is recalculated.

Here is the same loop once more, expressed using Informix SPL's WHILE statement:

```
/* Lower targets until total below $2,400,000 */
select sum(target) into total_tgt from offices;
while (total_tgt < 2400000.00)
   update offices
      set target = target - 10000.00;
   select sum(target) into total_tgt from offices;
end while;
```

Other variants of these loop-processing constructs are provided by the various dialects, but the capabilities and syntax are similar to these examples.

Other Flow-of-Control Constructs

Some stored procedure dialects provide statements to control looping and alter the flow of control. In Informix, for example, the EXIT statement interrupts the normal flow within a loop and causes execution to resume with the next statement following the loop itself. The CONTINUE statement interrupts the normal flow within the loop but causes execution to resume with the next loop iteration. Both of these statements have three forms, depending on the type of loop being interrupted:

```
exit for;
exit while;
exit foreach;
continue for;
continue while;
continue foreach;
```

In Transact-SQL, a single statement, BREAK, provides the equivalent of the Informix EXIT statement variants, and there is a single form of the CONTINUE statement as well. In Oracle, the EXIT statement performs the same function as for Informix, and there is no CONTINUE statement.

Additional control over the flow of execution within a stored procedure is provided by statement labels and the GOTO statement. In most dialects, the statement label is an identifier, followed by a colon. The GOTO statement names the label to which control should be transferred.

There is typically a restriction that you cannot transfer control out of a loop or a conditional testing statement, and always a prohibition against transferring control into the middle of such a statement. As in structured programming languages, the use of GOTO statements is discouraged, because it makes stored procedure code harder to understand and debug

Cursor-Based Repetition

One common need for repetition of statements within a stored procedure is when the procedure executes a query and needs to process the query results, row by row. All of the major dialects provide a structure for this type of processing. Conceptually, the structures parallel the DECLARE CURSOR, OPEN CURSOR, FETCH, and CLOSE CURSOR statements in embedded SQL or in the corresponding SQL API calls. However, instead of fetching the query results into the application program, in this case, they are being fetched into the stored procedure, which is executing within the DBMS itself. Instead of retrieving the query results into application program variables (host variables), the stored procedure retrieves them into local stored procedure variables.

To illustrate this capability, assume that you want to populate two tables with data from the ORDERS table. One table, named BIGORDERS, should contain customer name and order size for any orders over $10,000. The other, SMALLORDERS, should contain the salesperson's name and order size for any orders under $1000. The best and most efficient way to do this would be to use two separate SQL INSERT statements with subqueries, but for purposes of illustration, consider this method instead:

- Execute a query to retrieve the order amount, customer name, and salesperson name for each order.

- For each row of query results, check the order amount to see whether it falls into the proper range for including in the BIGORDERS or SMALLORDERS tables.
- Depending on the amount, INSERT the appropriate row into the BIGORDERS or SMALLORDERS table.
- Repeat Steps 2 and 3 until all rows of query results are exhausted.
- Commit the updates to the database.

Figure 13 shows an Oracle stored procedure that carries out this method. The cursor that defines the query is defined early in the procedure and assigned the name O_CURSOR. The variable CURS_ROW is defined as an Oracle row type. It is a structured Oracle row variable with individual components (like a C-language structure). By declaring it as having the same row type as the cursor, the individual components of CURS_ROW have the same data types and names as the cursor's query results columns.

```
create procedure sort_orders()
   /* Cursor for the query */
   cursor o_cursor is
   select amount, company, name
     from orders, customers, salesreps
    where cust = cust_num
      and rep = empl_num;

   /* Row variable to receive query results values */
   curs_row o_cursor%rowtype;


begin

   /* Loop through each row of query results */
   for curs_row in o_cursor
   loop

      /* Check for small orders and handle */
      if (curs_row.amount < 1000.00)
      then insert into smallorders
                values (curs_row.name, curs_row.amount);
```

```
    /* Check for big orders and handle */
    elsif (curs_row.amount > 10000.00)
    then insert into bigorders
              values (curs_row.company, curs_row.amount);
    end if;
 end loop;
commit;
end;
```

Figure 13. A cursor-based FOR loop in PL/SQL

The query described by the cursor is actually carried out by the cursor-based FOR loop. It basically tells the DBMS to carry out the query described by the cursor (equivalent to the OPEN statement in embedded SQL) before starting the loop processing. The DBMS then executes the FOR loop repeatedly, by fetching a row of query results at the top of the loop, placing the column values into the CURS_ROW variable, and then executing the statements in the loop body. When no more rows of query results are to be fetched, the cursor is closed, and processing continues after the loop.

Figure 14 shows an equivalent stored procedure with the specialized FOR loop structure of Informix SPL. In this case, the query results are retrieved into ordinary local variables; there is no special row data type used. The FOREACH statement incorporates several different functions. It defines the query to be carried out, through the SELECT expression that it contains. It marks the beginning of the loop that is to be executed for each row of query results. (The end of the loop is marked by the END FOREACH statement.)

```
create procedure sort_orders()

    /* Local variables to hold query results */
    define ord_amt   numeric(16,2);          /* order amount */
    define c_name    varchar(20);            /* customer name */
    define r_name    varchar(15);            /* salesrep name */

    /* Execute query and process each results row */
    foreach select amount, company, name
              into ord_amt, c_name, r_name
              from orders, customers, salesreps
             where cust = cust_num
               and rep = empl_num;
       begin
```

```
        /* Check for small orders and handle */
        if (ord_amt < 1000.00)
        then insert into smallorders
                    values (r_name, ord_amt);

        /* Check for big orders and handle */
        elif (ord_amt > 10000.00)
        then insert into bigorders
                    values (c_name, ord_amt);
        end if;
      end;
   end foreach;
end procedure;
```

Figure 14. A cursor-based FOREACH loop in Informix SPL.

When the FOREACH statement is executed, it carries out the query and then fetches rows of query results repeatedly, putting their column values into the local variables as specified in the statement. After each row is fetched, the body of the loop is executed. When there are no more rows of query results, the cursor is automatically closed, and execution continues with the next statement following the FOREACH. Note that in this example, the cursor isn't even assigned a specific name because all cursor processing is tightly specified within the single FOREACH statement.

The Transact-SQL dialect doesn't have a specialized FOR loop structure for cursorbased query results processing. Instead, the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements of embedded SQL have direct counterparts within the Transact-SQL language. Figure 15 shows a Transact-SQL version of the sort_orders procedure. Note the separate DECLARE, OPEN, FETCH, and CLOSE statements for the cursor. Loop control is provided by testing the system variable @@SQLSTATUS, which is the Transact-SQL equivalent of the SQLSTATE code. It receives a value of zero when a fetch is successful, and a nonzero value when there are no more rows to fetch.

3G E-LEARNING

```
create proc sort_orders()
as
/* Local variables to hold query results */
declare @ord_amt  decimal(16,2);              /* order amount */
declare @c_name   varchar(20);                /* customer name */
declare @r_name   varchar(15);                /* salesrep name */

/* Declare cursor for the query */
declare o_curs cursor for
        select amount, company, name
          from orders, customers, salesreps
         where cust = cust_num
           and rep = empl_num
begin

   /* Open cursor and fetch first row of results */
   open o_curs
   fetch o_curs into @ord_amt, @c_name, @r_name

   /* If no rows, return immediately */
   if (@@sqlstatus = 2)
   begin
      close o_curs
      return
   end

   /* Loop through each row of query results */
   while (@@sqlstatus = 0)
   begin


      /* Check for small orders and handle */
      if (@ord_amt < 1000.00)
         insert into smallorders
                values (@r_name, @ord_amt)

      /* Check for big orders and handle */
      else if (curs_row.amount > 10000.00)
         insert into bigorders
                values (@c_name, @ord_amt)
   end

   /* Done with results; close cursor and return */
   close o_curs
end
```

Figure 15 A cursor-based WHILE loop in Transact-SQL

3G E-LEARNING

Handling Error Conditions

When an application program uses embedded SQL or a SQL API for database processing, the application program is responsible for handling errors that arise. Error status codes are returned to the application program, and more error information is typically available through additional API calls or access to an extended diagnostics area. When database processing takes place within a stored procedure, the procedure itself must handle errors.

Transact-SQL provides error handling through a set of global system variables. The specific error-handling variables are only a few of well over 100 system variables that provide information on the state of the server, transaction state, open connections, and other database configuration and status information. The two most useful global variables for error handling are

- @@ERROR Contains error status of the most recently executed statement batch
- @@SQLSTATUS Contains status of the last fetch operation

The normal completion values for both variables are zero; other values indicate various errors and warnings. The global variables can be used in the same way as local variables within a Transact-SQL procedure. Specifically, their values can be checked for branching and loop control.

Oracle's PL/SQL provides a different style of error handling. The Oracle DBMS provides a set of system-defined exceptions, which are errors or warning conditions that can arise during SQL statement processing. Within an Oracle stored procedure (actually, any Oracle statement block), the EXCEPTION section tells the DBMS how it should handle any exception conditions that occur during the execution of the procedure. There are over a dozen different predefined Oracle-detected exception conditions. In addition, you can define your own exception conditions.

Most of the previous examples in this chapter don't provide any real error-handling capability. Figure 16 shows a revised version of the Oracle stored function in Figure 7. This improved version detects the specific situation where the supplied customer number does not have any associated orders (that is, where the query to calculate total orders returns a NO_DATA_FOUND exception). It responds to this situation by signaling back to the application program an application-level error and associated message. Any other exception conditions that arise are caught by the WHEN OTHERS exception handler

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in number)
       return number
as

/* Declare one local variable to hold the total */
declare tot_ord number(16,2);

begin
   /* Simple single-row query to get total */
   select sum(amount)
     into tot_ord
     from orders
    where cust = c_num;

   /* return the retrieved value as fcn value */
   return tot_ord;

exception
   /* Handle the situation where no orders found */
   when no_data_found
   then raise_application_error (-20123, 'Bad cust#');


   /* Handle any other exceptions */
   when others
   then raise_application_error (-20199,'Unknown error');
end;
```

Figure 16. PL/SQL function with error handling.

The Informix SPL takes a similar approach to exception handling. Figure 17 shows the Informix version of the stored function, with Informix-style exception handling. The ON EXCEPTION statement is a declarative statement and specifies the sequence

of SQL statements to be executed when a specific exception arises. A comma-separated list of exception numbers may be specified.

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in integer)
       returning numeric(16,2)

/* Declare one local variable to hold the total */
define tot_ord numeric(16,2);

/* Define exception handler for error #-123 and -121 */
on exception in (-121, -123)
   /* Do whatever is appropriate here */
   . . .
end exception;
on exception
   /* Handle any other exceptions in here */
   . . .
end exception;
```

Figure 17. Informix SPL function with condition handling.

10.1.3 Advantages of Stored Procedures

Stored procedures offer several advantages, both for database users and database administrators, including

- **Runtime performance** Many DBMS brands compile stored procedures (either automatically or at the user's request) into an internal representation that can be executed very efficiently by the DBMS at runtime. Executing a precompiled stored procedure can be much faster than running the equivalent SQL statements through the PREPARE/EXECUTE process.
- **Reusability** Once a stored procedure has been defined for a specific function, that procedure may be called from many different application programs that need to perform the function, permitting very easy reuse of application logic and reducing the risk of application programmer error.
- **Reduced network traffic** In a client/server configuration, sending a stored procedure call across the network and receiving the results in a reply message

generates much less network traffic than using a network round trip for each individual SQL statement. This can improve overall system performance considerably in a network with heavy traffic or one that has lower-speed connections.

- **Security** In most DBMS brands, the stored procedure is treated as a trusted entity within the database and executes with its own privileges. The user executing the stored procedure needs to have only permission to execute it, not permission on the underlying tables that the stored procedure may access or modify. Thus, the stored procedure allows the database administrator to maintain tighter security on the underlying data, while still giving individual users the specific data update or data access capabilities they require.
- **Encapsulation** Stored procedures are a way to achieve one of the core objectives of object-oriented programming—the encapsulation of data values, structures, and access within a set of very limited, well-defined external interfaces. In object terminology, stored procedures can be the methods through which the objects in the underlying RDBMS are exclusively manipulated. To fully attain the object-oriented approach, all direct access to the underlying data via SQL must be disallowed through the RDBMS security system, leaving only the stored procedures for database access. In practice, few if any production relational databases operate in this restricted manner.
- **Simplicity of access** In a large enterprise database, a collection of stored procedures may be the main way in which application programs access the database. The stored procedures form a well-defined set of transactions and queries that applications can perform on the database. For most application programmers, a call to a simple, predefined function that checks an account balance, given a customer number, or one that adds an order, given a customer number, quantity, and product-id, is easier to understand than the corresponding SQL statements.
- **Business rules enforcement** The conditional processing capabilities of stored procedures are often used to place business rules into the database. For example, a stored procedure used to add an order to the database might contain logic to check the credit of the customer placing the order and check whether there is enough inventory on hand to fill the order, and reject the order if these conditions cannot be met. A large company could quite easily have several different ways in which orders are taken and entered into the corporate database—one program for use by direct salespeople, one for people in the telesales department, another that accepts orders placed via the Web, and so on. Each of these would typically have its own orderacceptance program, usually written by different programmers at different times. But if all of the programs are forced to use the same stored procedure to add an order, the

company can be assured that the business rules in that procedure are being uniformly enforced, no matter where the order originated.

10.1.4 Stored Procedure Performance

Different DBMS brands vary in the way they actually implement stored procedures. In several brands, the stored procedure text is stored within the database and is interpreted when the procedure is executed. This has the advantage of creating a very flexible stored procedure language, but it creates significant runtime overhead for complex stored procedures. The DBMS must read the statements that make up the stored procedure at runtime, parse and analyze them, and determine what to do on the fly.

Because of the overhead in the interpreted approach, some DBMS brands compile stored procedures into an intermediate form that is much more efficient to execute. Compilation may be automatic when the stored procedure is created, or the DBMS may provide the ability for the user to request stored procedure compilation. The disadvantage of compiled stored procedures is that the exact technique used to carry out the stored procedure is fixed when the procedure is compiled. Suppose, for example, that a stored procedure is created and compiled soon after a database is first created, and later some useful indexes are defined on the data. The compiled queries in the stored procedure won't take advantage of these indexes, and as a result, they may run much more slowly than if they were recompiled.

To deal with stale compiled procedures, some DBMS brands automatically mark any compiled procedures that may be affected by subsequent database changes as being in need of recompilation. The next time the procedure is called, the DBMS notices the mark and recompiles the procedure before executing it. Normally, this approach provides the best of both worlds—the performance benefits of precompilation while keeping the compiled procedure up to date. Its disadvantage is that it can yield unpredictable stored procedure execution times. When no recompile is necessary, the stored procedure may execute quickly; when a recompile is activated, it may produce a significant delay; and in most cases, the recompile delay is much longer than the disadvantage of using the old compiled version.

To determine the stored procedure compilation capabilities of a particular DBMS, you can examine its CREATE PROCEDURE and EXECUTE PROCEDURE statement options, or look for other procedure management statements such as ALTER PROCEDURE.

10.1.5 System-Defined Stored Procedures

DBMS brands that support stored procedures sometimes provide built-in, system-defined stored procedures to automate database processing or management functions. Sybase SQL Server pioneered this use of system stored procedures. Today, hundreds of Transact-SQL system stored procedures provide functions such as managing users,

database roles, job execution, distributed servers, replication, and others. Most Transact-SQL system procedures follow this naming convention:

- **sp_add_something** Adds a new object (user, server, replica, etc.)
- **sp_drop_something** Drops an existing object
- **sp_help_something** Gets information about an object or objects

For example, the sp_helpuser procedure returns information about the valid users of the current database. You will notice that in Microsoft SQL Server, the names of TransactSQL system stored procedures often have underscores between words except for the one included in the name prefix (sp_). Also, since the vendors use the prefix sp_ to distinguish their supplied system stored procedures, it's a good idea to avoid using that prefix in procedures that users add to the database.

Oracle uses the prefix DBMS_ for procedures provided with its namesake DBMS. Most of these procedures are bundled into packages by functional category.

The package DBMS_LOB contains general purpose routines (stored procedures and functions) for operations on large objects (LOBs).

10.1.6 External Stored Procedures

Although stored procedures written in the extended SQL dialects of the major enterprise DBMS brands can be quite powerful, they have limitations. One major limitation is that they do not provide access to features outside the DBMS, such as the features of the operating system or other applications running on the same computer system. The extended SQL dialects also tend to be fairly high-level languages, with limited capability for the lower-level programming usually done in C or C++. To overcome these limitations, some DBMS brands provide access to external stored procedures.

An external stored procedure is a procedure written in a conventional programming language (such as C or Pascal) and compiled outside the DBMS itself. The DBMS is given

a definition of the procedure's name and its parameters, along with other essential information such as the calling conventions used by the programming language in which the stored procedure was written. Once defined to the DBMS, the external stored procedure can be called as if it were a SQL stored procedure. The DBMS handles the call, turns over control to the external procedure, and then receives any return values and parameters.

Microsoft SQL Server provides a set of system-defined external stored procedures that provide access to selected operating system capabilities. The xp_sendmail procedure can be used to send electronic mail to users, based on conditions within the DBMS:

```
xp_sendmail @RECIPIENTS = 'Joe', 'Sam',
            @MESSAGE = 'Customer table nearly full';
```

Similarly, the xp_cmdshell external procedure can be called to pass commands to the underlying operating system on which SQL Server is operating. Beyond these predefined external procedures, SQL Server allows a user-written external procedure to be stored in a dynamic-linked library (DLL) and called from within SQL Server stored procedures.

Informix provides basic access to underlying operating system capabilities with a special SYSTEM statement. In addition, it supports user-written external procedures through its CREATE PROCEDURE statement. Where the statement block comprising the body of an Informix SPL procedure would appear, an EXTERNAL clause specifies the name, location, and language of the externally written procedure. With the procedure defined in this way, it can be called in the same way as native Informix SPL procedures. Newer versions of Oracle (Oracle8 and later) provide the same capability, also via the CREATE PROCEDURE statement. IBM's DB2 database family provides the same set of capabilities.

10.2 Triggers

A trigger is a special set of stored procedural code whose activation is caused by modifications to the database contents. Unlike stored procedures, a trigger is not activated by a CALL or EXECUTE statement. Instead, the trigger is associated with a database table. When the data in the table is changed by an INSERT, DELETE, or UPDATE statement, the trigger is fired, which means that the DBMS executes the SQL statements that make up the body of the trigger. Some DBMS brands allow definition of specific updates that cause a trigger to fire. Also, some DBMS brands, notably Oracle, allow triggers to be based on system events such as users connecting to the database or execution of a database shutdown command.

Triggers can be used to cause automatic updates of information within a database. For example, suppose you wanted to set up the sample database so that any time a

new salesperson is inserted into the SALESREPS table, the sales target for the office where the salesperson works is raised by the new salesperson's quota. Here is an Oracle PL/SQL trigger that accomplishes this goal:

```
Create or replace trigger upd_tgt
   /* Insert trigger for SALESREPS */
   before insert on salesreps
   for each row
   begin
      if :new.quota is not null
       then
         update offices
            set target = target + new.quota;
       end if;
   end;
```

The CREATE TRIGGER statement is used by most DBMS brands that support triggers to define a new trigger within the database. It assigns a name to the trigger (UPD_TGT for this one) and identifies the table the trigger is associated with (SALESREPS) and the update action(s) on that table that will cause the trigger to be executed (INSERT in this case). The body of this trigger tells the DBMS that for each new row inserted into the table, it should execute the specified UPDATE statement for the OFFICES table. The QUOTA value from the newly inserted SALESREPS row is referred to as :NEW.QUOTA within the trigger body.

10.2.1 Advantages and Disadvantages of Triggers

- **Auditing changes** A trigger can detect and disallow specific updates and changes that should not be permitted in the database.
- **Cascaded operations** A trigger can detect an operation within the database (such as deletion of a customer or salesperson) and automatically cascade the impact throughout the database (such as adjusting account balances or sales targets).
- **Enforce interrelationships** A trigger can enforce more complex interrelationships among the data in a database than those that can be expressed by simple referential integrity constraints or check constraints, such as those that require a sequence of SQL statements or IF…THEN…ELSE processing.
- **Stored procedure invocation** A trigger can call one or more stored procedures or even invoke actions outside the DBMS itself through external procedure calls in response to database updates.
- **Detecting system events** For DBMSs that support triggers based on system events, the trigger can audit or monitor such events, such as tracing a particular user whenever they connect to the database.

In each of these cases, a trigger embodies a set of business rules that govern the data in the database and modifications to that data. The rules are embedded in a single place in the database (the trigger definition). As a result, they are uniformly enforced across all applications that access the database. When they need to be changed, they can be changed once with the assurance that the change will be applied uniformly.

The major disadvantage of triggers is their potential performance impact. If a trigger is set on a particular table, then every database operation that attempts to change that table's data in the manner defined in the trigger (an insert, delete, or update to one or more columns) causes the DBMS to execute the trigger procedure. For a database that requires very high data insertion or update rates, the overhead of this processing can be considerable. This is especially true for bulk load operations, where the data may have already been prechecked for integrity. To deal with this disadvantage, some DBMS brands allow triggers to be selectively enabled and disabled, as appropriate.

10.2.2 Triggers in Transact-SQL

Transact-SQL provides triggers through a CREATE TRIGGER statement in both its Microsoft SQL Server and Sybase Adaptive Server dialects. Here is a Transact-SQL trigger definition for the sample database, which implements the same trigger as the preceding Oracle PL/ SQL example:

```
create trigger upd_tgt
   /* Insert trigger for SALESREPS */
   on salesreps
   for insert
   as
   if (@@rowcount = 1)



     begin
        update offices
           set target = target + inserted.quota
          from offices, inserted
         where offices.office = inserted.rep_office;
     end
  else
     raiserror 23456;
```

The first clause names the trigger (UPD_TGT). The second clause is required and identifies the table to which the trigger applies. The third clause is also required and tells which database update operations cause the trigger to be fired. In this case, only an INSERT statement causes the trigger to fire. You can also specify UPDATE or DELETE operations, or a combination of two or three of these operations in a comma-

separated list. Transact-SQL restricts triggers so that only one trigger may be defined on a particular table for each of the three data modification operations. The body of the trigger follows the AS keyword. To understand the body of a trigger like this one, you need to understand how Transact-SQL treats the rows in the target table during database modification operations.

For purposes of trigger operation, Transact-SQL defines two logical tables whose column structure is identical to the target table on which the trigger is defined. One of these logical tables is named DELETED, and the other is named INSERTED. These logical tables are populated with rows from the target table, depending on the data modification statement that caused the trigger to fire, as follows:

- **DELETE** Each target table row that is deleted by the DELETE statement is placed into the DELETED table. The INSERTED table is empty.
- **INSERT** Each target table row that is added by the INSERT statement is also placed into the INSERTED table. The DELETED table is empty.
- **UPDATE** For each target table row that is changed by the UPDATE statement, a copy of the row before any modifications is placed into the DELETED table. A copy of the row after all modifications is placed into the INSERTED table.

These two logical tables can be referenced within the body of the trigger, and the data in them can be combined with data from other tables during the trigger's operation. In this Transact-SQL trigger, the trigger body first tests to make sure that only a single row of the SALESREPS table has been inserted, by checking the system variable @@ROWCOUNT. If this is true, then the QUOTA column from the INSERTED logical table is added to the appropriate row of the OFFICES table. The appropriate row is determined by joining the logical table to the OFFICES table based on matching office numbers.

Here is a different trigger that detects a different type of data integrity problem. In this case, it checks for an attempt to delete a customer when there are still orders outstanding in the database for that customer. If it detects this situation, the trigger automatically rolls back the entire transaction, including the DELETE statement that fired the trigger:

```
create trigger chk_del_cust
   /* Delete trigger for CUSTOMERS */
   on customers
   for delete
```

```
as
/* Detect any orders for deleted cust #'s */
if (select count(*)
      from orders, deleted
     where orders.cust = deleted.cust_num) > 0
   begin
      rollback transaction
      print "Cannot delete; still have orders"
      raiserror 31234
   end;
```

Transact-SQL triggers can be specified to fire on any UPDATE for a target table, or just for updates of selected columns. This trigger fires on inserts or updates to the SALESREPS table and does different processing depending on whether the QUOTA or SALES column has been updated:

```
create trigger upd_reps
   /* Update trigger for SALESREPS */
   on salesreps
   for insert, update
   if update(quota)
      /* Handle updates to quota column */
      . . .
   if update (sales)
      /* Handle updates to sales column */
      . . .
```

10.2.3 Triggers in Informix SPL

Informix also supports triggers through a CREATE TRIGGER statement. As in the TransactSQL dialect, the beginning of the CREATE TRIGGER statement defines the trigger name, the table on which the trigger is being defined, and the triggering actions. Here are statement fragments that show the syntax:

```
create trigger new_sls
   insert on salesreps . . .

create trigger del_cus_chk
   delete on customers . . .

create trigger ord_upd
   update on orders . . .

create trigger sls_upd
   update of quota, sales on salesreps . . .
```

The last example is a trigger that fires only when two specific columns of the SALESREPS table are updated.

Informix allows you to specify that a trigger should operate at three distinct times during the processing of a triggered change to the target table:

- **BEFORE** The trigger fires before any changes take place. No rows of the target table have yet been modified.
- **AFTER** The trigger fires after all changes take place. All affected rows of the target table have been modified.
- **FOR EACH ROW** The trigger fires repeatedly, once as each row affected by the change is being modified. Both the old and new data values for the row are available to the trigger.

An individual trigger definition can specify actions to be taken at one or more of these steps. For example, a trigger could execute a stored procedure to calculate the sum of all orders BEFORE an update, monitor updates to each ORDERS row as they occur with a second action, and then calculate the revised order total AFTER the update with a call to another stored procedure. Here is a trigger definition that does all of this:

```
create trigger upd_ord
   update of amount on orders
   referencing old as pre new as post

   /* Calculate order total before changes */
   before (execute procedure add_orders()
              into old_total;)

   /* Capture order increases and decreases */
   for each row
      when (post.amount < pre.amount)
         /* Write decrease data into table */
         (insert into ord_less
               values (pre.cust,
                       pre.order_date,
                       pre.amount,
                       post.amount);)
      when (post.amount > pre.amount)
         /* Write increase data into table */
         (insert into ord_more
               values (pre.cust,
                       pre.order_date,
                       pre.amount,
                       post.amount);)
```

```
/* After changes, recalculate total */
after (execute procedure add_orders()
          into new_total;)
```

The BEFORE clause in this trigger specifies that a stored procedure named ADD_ORDERS is to be called before any UPDATE statement processing occurs. Presumably, this procedure calculates the total orders and returns the total value into the local variable OLD_TOTAL. Similarly, the AFTER clause specifies that a stored procedure (in this case, the same one) is to be called after all UPDATE statement processing is complete. This time, the total orders amount is placed into a different local variable, NEW_TOTAL.

The FOR EACH ROW clause specifies the action to be taken as each affected row is updated. In this case, the requested action is an INSERT into one of two order-tracking tables, depending on whether the order amount is being increased or decreased. These tracking tables contain the customer number, date, and both the old and new order amounts. To obtain the required values, the trigger must be able to refer to both the old (prechange) and the new (postchange) values of each row.

The REFERENCING clause provides names by which these two states of the row currently being modified in the ORDERS table can be used. In this example, the prechange values of the columns are available through the column name qualifier PRE, and the postchange values are available through the column name qualifier POST. These are not special names; any names can be used.

Informix is more limited than some other DBMS brands in the actions that can be specified within the trigger definition itself. These statements are available:

- INSERT
- DELETE
- UPDATE
- EXECUTE PROCEDURE

In practice, the last option provides quite a bit of flexibility. The called procedure can perform almost any processing that could be done inline within the trigger body itself.

10.2.4 Triggers in Oracle PL/SQL

Oracle provides a more complex trigger facility than either the Informix or Transact-SQL facility described in the preceding sections. It uses a CREATE TRIGGER statement to specify triggered actions. As in the Informix facility, a trigger can be specified to fire at specific times during specific update operations:

- **Statement-level trigger** A statement-level trigger fires once for each data modification statement. It can be specified to fire either before the statement is executed or after the statement has completed its action.
- **Row-level trigger** A row-level trigger fires once for each row being modified by a statement. In Oracle's structure, this type of trigger may also fire either before the row is modified or after it is modified.
- **Instead-of trigger** An instead-of trigger takes the place of an attempted data modification statement. It provides a way to detect an attempted UPDATE, INSERT, or DELETE operation by a user or procedure, and to substitute other processing instead. You can specify that a trigger should be executed instead of a statement, or that it should be executed instead of each attempted modification of a row.
- **System event trigger** A trigger that fires when a particular system event takes place, such as a user connecting to the database, or entry of a database shutdown command.

The following code is a PL/SQL trigger definition that implements the same processing as in the complex Informix example from the previous section. It has been split into three separate Oracle CREATE TRIGGER statements; one each for the BEFORE and AFTER statement-level triggers and one trigger that is executed for each update row.

```
create trigger bef_upd_ord
   before update on orders
   begin
      /* Calculate order total before changes */
      old_total = add_orders();
   end;

create trigger aft_upd_ord
   after update on orders
   begin
      /* Calculate order total after changes */
      new_total = add_orders();
   end;
```

```
create trigger dur_upd_ord
   before update of amount on orders
   referencing old as pre new as post

   /* Capture order increases and decreases */
   for each row
      when (:post.amount != :pre.amount)
         begin
            if post.amount != :pre.amount)
            then
               if (:post.amount < :pre.amount)
               then
                  /* Write decrease data into table */
                  insert into ord_less
                        values (:pre.cust,
                                :pre.order_date,
                                :pre.amount,
                                :post.amount);
               elsif (:post.amount > :pre.amount)
               then
                  /* Write increase data into table */
                  insert into ord_more
                        values (:pre.cust,
                                :pre.order_date,
                                :pre.amount,
                                :post.amount);
               end if;
            end if;
         end;
```

These trigger structures and their options provide 14 different valid Oracle trigger types (12 resulting from a choice of INSERT/DELETE/UPDATE triggers for BEFORE or AFTER processing at the row or statement level (3×2×2), and two more from instead-of triggers at the statement or row level). In practice, relational databases built using Oracle don't tend to use instead-of triggers; they were introduced in Oracle8 to support some of its newer object-oriented features.

10.2.5 Other Trigger Considerations

Triggers pose some of the same issues for DBMS processing that UPDATE and DELETE rules present. For example, triggers can cause a cascaded series of actions. Suppose a user's attempt to update a table causes a trigger to fire, and within the body of that trigger is an UPDATE statement for another table. A trigger on that table causes the UPDATE of still another table, and so on. The situation is even worse if one of the fired triggers attempts to update the original target table that caused the firing of the trigger sequence in the first place! In this case, an infinite loop of fired triggers could result.

Various DBMS systems deal with this issue in different ways. Some impose restrictions on the actions that can be taken during execution of a trigger. Others provide built-in functions that allow a trigger's body to detect the level of nesting at which the trigger is operating. Some provide a system setting that controls whether cascaded trigger processing is allowed. Finally, some provide a limit on the number of levels of nested triggers that can fire.

One additional issue associated with triggers is the overhead that can result during very heavy database usage, such as when bulk data is being loaded into a database. Some DBMS brands provide the ability to selectively enable and disable trigger processing to handle this situation. Oracle, for example, provides this form of the ALTER TRIGGER statement:

```
ALTER TRIGGER BEF_UPD_ORD DISABLE;
```

A similar capability is provided within the CREATE TRIGGER statement of Informix.

10.3 STORED PROCEDURES, FUNCTIONS, TRIGGERS, AND THE SQL STANDARD

The development of DBMS stored procedures, functions, and triggers has been largely driven by DBMS vendors and the competitive dynamics of the database industry. Sybase's initial introduction of stored procedures and triggers in SQL Server triggered a competitive response, and by the mid-1990s, many of the enterprise-class systems had added their own proprietary procedural extensions to SQL. Stored procedures were not a focus of the SQL standard, but became a part of the standardization agenda after the 1992 publication of the SQL2 standard. The work on stored procedure standards was split off from the broader object-oriented extensions that were proposed for SQL3, and was focused on a set of procedural extensions to the SQL language.

The result was a new part of the SQL standard, published in 1996 as SQL/Persistent Stored Modules (SQL/PSM), International Standard ISO/IEC 9075-4. The actual form of the standard specification is a collection of additions, edits, new paragraphs, and replacement paragraphs to the 1992 SQL2 standard (ISO/IEC 9075:1992). In addition to being a modification of the SQL standard, SQL/PSM was also drafted as a part of the planned followon standard, which was called SQL3 during its drafting. The development of the follow-on standard took longer than expected, but SQL/PSM eventually took its place as Part 4 of the SQL3 standard, officially known as SQL:1999. The SQL Call-Level Interface (CLI) standard was treated the same way; it is now Part 3 of the SQL standard.

When the SQL:1999 standard was published, selected parts of SQL/PSM that are used by other parts of the standard were moved to the core SQL/Foundation specification (Part 1). The SQL/PSM standard published in 1996 addressed only stored procedures

and functions; it explicitly did not provide a specification of a trigger facility for the ISO SQL standard. The standardization of trigger functions was considered during the development of the SQL2 and SQL/PSM standards, but the standards groups determined that triggers were too closely tied to other object-oriented extensions proposed for SQL3. The SQL:1999 standard that resulted from the SQL3 work finally provided an ANSI/ISO standard trigger facility.

The publication of the SQL/PSM and SQL:1999 standards lagged the first commercial implementation of stored procedures and triggers by many years. By the time the standard was adopted, most enterprise DBMS vendors had responded to user enthusiasm and competitive pressure by introducing stored procedure and trigger capabilities in their products. Unlike some other SQL extensions where IBM's clout and a DB2 implementation had set a de facto standard, the major DBMS vendors implemented stored procedures and triggers in different, proprietary ways, and in some cases, competed with one another based on unique features of their implementations. As a result, the ANSI/ISO standardization of stored procedures and triggers has had little impact on the DBMS market to date. It's reasonable to expect that ANSI/ ISO implementations will find their way into major DBMS products over time, but as a complement to, rather than a replacement for, the proprietary implementations.

10.3.1 The SQL/PSM Stored Procedures Standard

The capabilities specified in the SQL/PSM standard parallel the core features of the proprietary stored procedure capabilities of today's DBMS systems. They include SQL language constructs to:

- Define and name procedures and functions written in the extended SQL language
- Invoke (call) a previously-defined procedure or function
- Pass parameters to a called procedure or function, and obtain the results of its execution
- Declare and use local variables within the procedure or function
- Group a block of SQL statements together for execution
- Conditionally execute SQL statements (IF…THEN…ELSE)
- Repeatedly execute a group of SQL statements (looping)

The SQL/PSM standard specifies two types of SQL-invoked routines. A SQL-procedure is a routine that can return any number of values or no value at all. It is called with a CALL statement:

```
CALL ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago');
```

As with the proprietary stored procedure languages illustrated in the previous examples throughout this chapter, SQL/PSM stored procedures accept parameters passed via the CALL statement. SQL/PSM stored procedures can also pass data back to their caller via output parameters, again mirroring the capabilities of the proprietary stored procedure languages. SQL/PSM also supports combined input/output parameters, like some of the proprietary languages.

A SQL function does return a value. It is called just like a built-in SQL function within a value expression:

```
SELECT COMPANY
  FROM CUSTOMERS
 WHERE GET_TOT_ORDS(CUST_NUM) > 10000.00;
```

SQL/PSM restricts SQL functions to only returning a single value through the functioncall mechanism. Output parameters and input/output parameters are not allowed in SQL functions.

SQL routines are objects within the database structure described in the SQL standard. SQL/PSM allows the creation of routines within a SQL schema (a schema-level routine), where it exists along with the tables, views, assertions, and other objects. It also allows the creation of routines within a SQL module, which is the SQL procedure model carried forward from the SQL1 standard.

Creating a SQL Routine

Following the practice of most DBMS brands, the SQL/PSM standard uses the CREATE PROCEDURE and CREATE FUNCTION statements to specify the definitions of stored procedures and functions. Figure 18 shows syntax for the CREATE PROCEDURE statement, and Figure 19 shows the syntax for the CREATE FUNCTION statement. In addition to the capabilities shown in the figure, the standard provides a capability to define external stored procedures, specifying the language they are written in, whether they can read or modify data in the database, their calling conventions, and other characteristics.

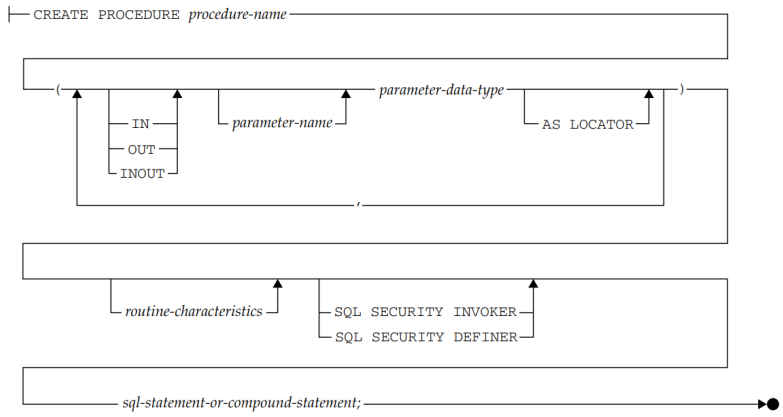


Figure 18. The SQL/PSM CREATE PROCEDURE syntax diagram.

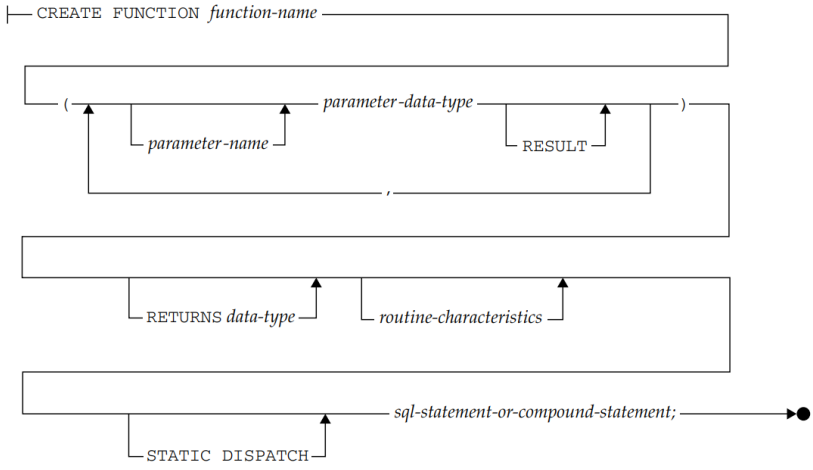


Figure 19. The SQL/PSM CREATE FUNCTION syntax diagram

Flow-of-Control Statements

The SQL/PSM standard specifies the common programming structures that are found in most stored procedure dialects to control the flow of execution. Figure 20 shows the conditional branching and looping syntax. Note that the SQL statement lists specified for each structure consist of a sequence of SQL statements, each ending with a semicolon. Thus, explicit block structures are not required for simple multistatement sequences that appear in an IF…THEN…ELSE statement or in a LOOP statement. The looping structures provide a great deal of flexibility for loop processing. There are forms that

place the test at the top of the loop or at the bottom of the loop, as well as a form that provides infinite looping and requires the explicit coding of a test to break loop execution. Each of the program control structures is explicitly terminated by an END flag that matches the type of structure, making programming debugging easier.

Cursor Operations

The SQL/PSM standard extends the cursor manipulation capabilities specified in the SQL2 standard for embedded SQL into SQL routines. The DECLARE CURSOR, OPEN, FETCH, and CLOSE statements retain their roles and functions. Instead of using application program host variables to supply parameter values and to receive retrieved data, SQL routine parameters and variables can be used for these functions.

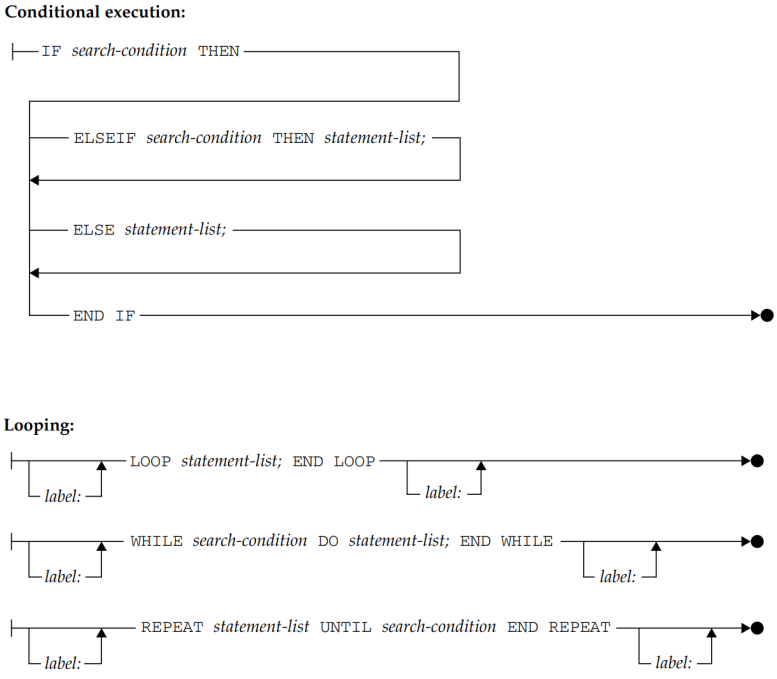


Figure 20. The SQL/PSM flow-of-control statements syntax diagram.

The SQL/PSM standard introduces one new cursor-controlled looping structure, shown in Figure 21. Like the similar structures in the Oracle and Informix dialects described in the "Cursor-Based Repetition" section earlier in this chapter, it combines the cursor definition and the OPEN, FETCH, and CLOSE statements into a single loop definition that also specifies the processing to be performed for each row of retrieved query results.

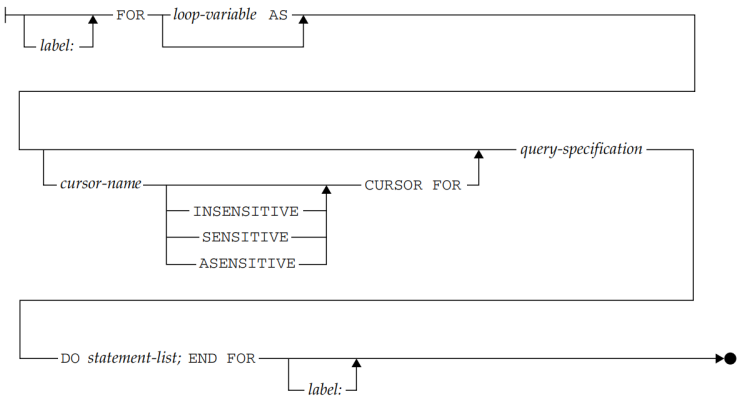
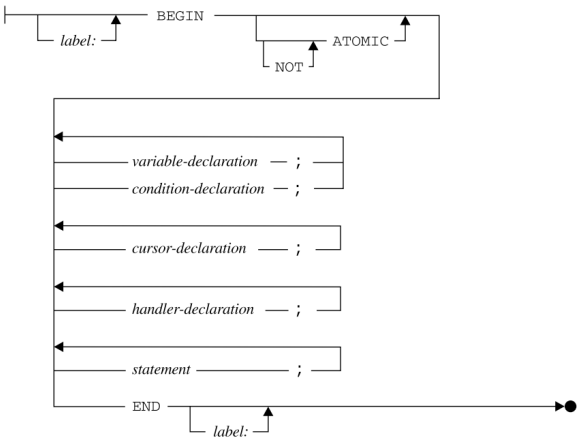


Figure 21. The SQL/PSM cursor-controlled loop syntax diagram.

Block Structure

Figure 22 shows the block structure specified by the SQL/PSM standard. It is quite a comprehensive structure, providing the following capabilities:

- Labels the block of statements with a statement label
- Declares local variables for use within the block
- Declares local user-defined error conditions
- Declares cursors for queries to be executed within the block
- Declares handlers to process error conditions that arise
- Defines the sequence of SQL statements to be executed



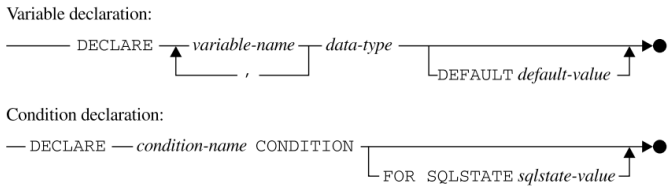


Figure 22. The SQL/PSM statement block syntax diagram.

These capabilities resemble some of those described earlier in the "Statement Blocks" section of this chapter for the Informix and Oracle dialect stored procedure dialects.

Local variables within SQL/PSM procedures and functions (actually, within statement blocks) are declared using the DECLARE statement. Values are assigned using the SET statement. Functions return a value using the RETURN statement. Here is a statement block that might appear within a stored function, with examples of these statements:

```
try_again:
   begin
      /* Declare some local variables */
      declare msg_text varchar(40);
      declare tot_amt decimal(16,2);

      /* Get the order total */
      set tot_amt = get_tot_ords();
      if (tot_amt > 0)
      then
         return (tot_amt);
      else
         return (0.00);
      end if
   end try_again;
```

Error Handling

The block structure specified by the SQL/PSM standard provides fairly comprehensive support for error handling. The standard specifies predefined conditions that can be detected and handled, including

- SQLWARNING One of the warning conditions specified in the SQL standard
- NOT FOUND The condition that normally occurs when the end of a set of query results is reached with a FETCH statement
- SQLSTATE value A test for specific SQLSTATE error codes

- User-defined condition A condition named by the stored procedure

Conditions are typically defined in terms of SQLSTATE values. Rather than using numerical SQLSTATE codes, you can assign the condition a symbolic name. You can also specify your own user-defined condition:

```
declare bad_err condition for sqlstate '12345';
declare my_err condition;
```

Once the condition has been defined, you can force the condition to occur through the execution of a SQL routine with the SIGNAL statement:

```
signal bad_err;
signal sqlstate '12345';
```

To handle error conditions that may arise, SQL/PSM allows you to declare a condition handler. The declaration specifies the list of conditions that are to be handled and the action to be taken. It also specifies the type of condition handling. The types differ in what happens to the flow of control after the handler is finished with its work:

- CONTINUE type After the condition handler completes its work, control returns to the next statement following the one that caused the condition. That is, execution continues with the next statement.
- EXIT type After the condition handler completes its work, control returns to the end of the statement block containing the statement that caused the condition. That is, execution effectively exits the block.
- UNDO type After the condition handler completes its work, all modifications are undone to data in the database caused by statements within the same statement block as the statement causing the error. The effect is the same as if a transaction had been initiated at the beginning of the statement block and was being rolled back.

Here are some examples that show the structure of the handler definition:

```
/* Handle SQL warnings here, then continue */
declare continue handler for sqlwarning
   call my_warn_routine();

/* Handle severe errors by undoing effects */
declare undo handler for user_disaster
   begin
      /* Do disaster cleanup here */
      . . .
   end;
```

Error handling can get quite complex, and it's possible for errors to arise during the execution of the handler routine itself. To avoid infinite recursion on errors, the normal condition signaling does not apply during the execution of a condition handler. The standard allows you to override this restriction with the RESIGNAL statement. It operates just like the SIGNAL statement, but is used exclusively within condition-handler routines.

Routine Name Overloading

The SQL/PSM standard permits overloading of stored procedure and function names. Overloading is a common attribute in object-oriented systems and is a way to make stored routines more flexible in handling a wide variety of data types and situations. Using the overloading capability, several different routines can be given the same routine name. The multiple routines defined with the same name must differ from one another in the number of parameters that they accept or in the data types of the individual parameters. For example, you might define these three stored functions:

```
create function combo(a, b)
   a integer;
   b integer;
   returns integer;
   as return (a+b);


create function combo(a, b, c)
   a integer;
   b integer;
   c integer;
   returns integer;
   as return (a+b+c);

create procedure combo(a, b)
   a varchar(255);
   b varchar(255);
   returns varchar(255);
   as return (a || b);
```

The first COMBO function combines two integers by adding them and returns the sum. The second COMBO function combines three integers the same way. The third COMBO function combines two character strings by concatenating them. The standard allows all of these functions named COMBO to be defined at the same time within the

database. When the DBMS encounters a reference to the COMBO function, it examines the number of arguments in the reference and their data types, and determines which version of the COMBO function to call. Thus, the overloading capability allows a SQL programmer to create a family of routines that logically perform the same function and have the same name, even though the specifics of their usage for different data types is different. In object-oriented terms, overloading is sometimes called polymorphism, meaning literally that the same function can take many different forms.

To simplify the management of a family of routines that share an overloaded name, the SQL/PSM standard has the concept of a specific name: a second name that is assigned to the routine that is unique within the database schema or module. It uniquely identifies a specific routine. The specific name is used to drop the routine, and it is reflected in the information schema views that describe stored routines. The specific name is not used to call the routine; that would defeat the primary purpose of the overloaded routine name. Support for specific names or some similar mechanism is a practical requirement for any system that permits overloading or polymorphism for objects and provides a capability to manage them by dropping or changing their definitions, since the system must be able to determine which specific object is being modified.

External Stored Procedures

The bulk of the SQL/PSM standard is concerned with the extensions to the SQL language that are used to define SQL procedures and functions. Note, however, that the method used to invoke a procedure (the CALL statement) or a function (a reference to the function by name within a SQL statement) is not particular to procedures defined in the SQL language. In fact, the SQL/PSM standard provides for external stored procedures and functions, written in some other programming language such as C or Pascal. For external procedures, the CREATE PROCEDURE and CREATE FUNCTION statements are still used to define the procedure to the DBMS, specifying its name and the parameters that it accepts or returns. A special clause of the CREATE statement specifies the language in which the stored procedure or function is written, so that the DBMS may perform the appropriate conversion of data types and call the routine appropriately.

Other Stored Procedure Capabilities

The SQL/PSM standard treats procedures and functions as managed objects within the database, using extensions to the SQL statements used to manage other objects. You use a variation of the DROP statement to delete routines when they are no longer needed, and a variation of the ALTER statement to change the definition of a function or procedure. The SQL standard permissions mechanism is similarly extended with additional privileges. The EXECUTE privilege gives a user the ability to execute a

stored procedure or function. It is managed by the GRANT and REVOKE statements in the same manner as other database privileges.

Because the stored routines defined by SQL/PSM are defined within SQL schemas, many routines can be defined in many different schemas throughout the database. When calling a stored routine, the routine name can be fully qualified to uniquely identify the routine within the database. The SQL/PSM standard provides an alternative method of searching for the definition of unqualified routine names through a new PATH concept. The PATH is the sequence of schema names that should be searched to resolve a routine reference. A default PATH can be specified as part of the schema header in the CREATE SCHEMA statement. The PATH can also be dynamically modified during a SQL session through the SET PATH statement.

The SQL/PSM standard also lets the author of a stored procedure or function give the DBMS some hints about its operation to improve the efficiency of execution. One example is the ability to define a stored routine as DETERMINISTIC or NOT DETERMINISTIC. A DETERMINISTIC routine will always return the same results when it is called with the same parameter values. If the DBMS observes that a DETERMINISTIC routine is called repeatedly, it may choose to keep a copy of the results that it returns. Later, when the routine is called again, the DBMS does not need to actually execute the routine; it can simply return the same results that it returned the last time.

Another form of hint tells the DBMS whether an external stored procedure or function reads database contents and whether it modifies database contents. This not only allows the DBMS to optimize database access, but can also impose a security restriction on external routines from other sources. Other hints determine whether a function should be called if one of its parameters has a NULL value, and control how the DBMS selects the specific function or procedure to be executed when overloading is used.

10.3.2 The SQL/PSM Triggers Standard

Triggers were addressed for standardization as part of the SQL3 effort, which led to the eventual publication of the SQL:1999 ANSI/ISO standard. By that time, many commercial DBMS products had already implemented triggers, and the standard synthesized the specific capabilities that had proven useful in practice. Like the commercial products, ANSI/ISO standard triggers are defined for a single, specific table. The standard permits trigger definitions only on tables, not on views.

The proprietary SQL Server, Oracle, and Informix trigger mechanisms shown in the examples throughout this chapter provide a context for examining the ANSI/ISO standard mechanism. The standard does not provide any radical departure from the capabilities already described for the various DBMS products. Here is how the standard compares with them:

- Naming The standard treats triggers as named objects within the database.
- Types The standard provides INSERT, DELETE, and UPDATE triggers; UPDATE triggers can be associated with the update of a specific column or group of columns.
- Timing The standard provides for triggers that operate before a database update statement or after the statement.
- Row-level or statement-level operation The standard provides for both statement-level triggers (executed once per database-updating statement) and rowlevel triggers (executed repeatedly for each row of the table that is modified).
- Aliases Access to the "before" and "after" values in a modified row or table is provided via an alias mechanism, like the table aliases used in the FROM clause.

You use the CREATE TRIGGER statement, shown in Figure 23, to define a trigger. The statement clauses are familiar from the proprietary trigger examples throughout the earlier sections of this chapter.

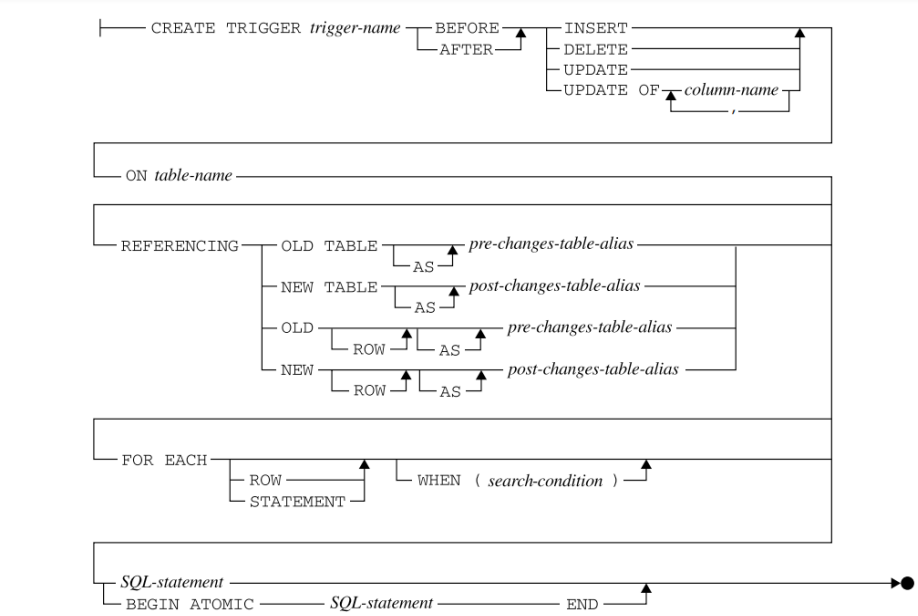


Figure 23. The SQL standard CREATE TRIGGER syntax diagram.

One very useful extension provided by the standard is the WHEN clause that can be specified as part of a triggered action. The WHEN clause is optional, and it operates like a WHERE clause for determining whether a triggered action will be carried out. When the DBMS executes the particular type of statement specified in the trigger

definition, it evaluates the search condition specified in the WHEN clause. The form of the search condition is exactly like the search condition in a WHERE clause, and it will produce either a TRUE or FALSE result. The triggered action is carried out only if the result is TRUE.

To provide security for triggers, the SQL standard establishes a new TRIGGER privilege that may be granted for specific tables to specific users. With this privilege, a user may establish a trigger on the table. The owner of a table is always allowed to establish triggers

ROLE MODEL

BOB MINER

Robert Nimrod Miner (December 23, 1941 – November 11, 1994) was an American businessman. He was the co-founder of Oracle Corporation and the producer of Oracle's relational database management system.

From 1977 until 1992, Bob Miner led product design and development for the Oracle relational database management system. In Dec., 1992, he left that role and spun off a small, advanced technology group within Oracle. He was an Oracle board member until Oct., 1993.

Early Life

Bob Miner was born on Dec 23, 1941 in Cicero, Illinois, to an Assyrian family. Both of his parents came from Ada, a village in West Azerbaijan Province, northwest Iran, and had migrated to the US in the 1920s. He was their fifth child of five. Bob Miner graduated in 1963 with a degree in mathematics from the University of Illinois at Urbana-Champaign.

Career

In 1977 Bob Miner met Larry Ellison at Ampex, where he was Larry's supervisor. Bob Miner left Ampex soon thereafter to found a company called Software Development Laboratories with Ed Oates and Bruce Scott, with Larry Ellison joining the company several months later. It was at this time that Ed Oates introduced Miner and Ellison to a paper by E. F. Codd on the relational model for database management. IBM was slow to see the commercial value of Codd's relational database management system (RDBMS), allowing Miner and Ellison to beat them to the market.

In the start-up days of Oracle Bob Miner was the lead engineer, programming the majority of Oracle Version 3 by himself. As head of engineering Bob Miner's management style was in stark contrast to Larry Ellison, who cultivated Oracle's hard-driving sales culture. Although he expected his

engineers to produce, he did not agree with the demands laid upon them by Ellison. He thought it was wrong for people to work extremely late hours and that they should have the chance to see their families. According to Ellison, Miner was "loyal to the people before the company."

Personal Life

Bob Miner was diagnosed in 1993 with pleural mesothelioma, a rare form of lung cancer caused by exposure to asbestos. He died on Friday, 11 November 1994 at the age of 52, surrounded by his wife Mary and their three children, Nicola, Justine, and Luke. His wife Mary is the founder and owner of Oakville Ranch Vineyards, a Napa winery. His daughter Nicola Miner is married to author Robert Mailer Anderson.

Miner family's charitable foundation has donated to various San Francisco arts and education institutions. The SFJAZZ Center's auditorium is named after Miner.

SUMMARY

- The long-term trend in the database market is for databases to take on a progressively larger role in the overall data processing architecture.
- A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system (RDBMS) as a group, so it can be reused and shared by multiple programs.
- The CREATE PROCEDURE statement is used to create a stored procedure and to specify how it operates. The CREATE PROCEDURE statement assigns the newly defined procedure a name, which is used to call it. The name must typically follow the rules for SQL identifiers.
- Different DBMS brands vary in the way they actually implement stored procedures. In several brands, the stored procedure text is stored within the database and is interpreted when the procedure is executed. This has the advantage of creating a very flexible stored procedure language, but it creates significant runtime overhead for complex stored procedures.
- A trigger is a special set of stored procedural code whose activation is caused by modifications to the database contents. Unlike stored procedures, a trigger is not activated by a CALL or EXECUTE statement.
- The trigger is associated with a database table. When the data in the table is changed by an INSERT, DELETE, or UPDATE statement, the trigger is fired, which means that the DBMS executes the SQL statements that make up the body of the trigger.
- The SQL/PSM standard specifies the common programming structures that are found in most stored procedure dialects to control the flow of execution.
- The SQL/PSM standard extends the cursor manipulation capabilities specified in the SQL2 standard for embedded SQL into SQL routines. The DECLARE CURSOR, OPEN, FETCH, and CLOSE statements retain their roles and functions.
- The SQL/PSM standard permits overloading of stored procedure and function names. Overloading is a common attribute in object-oriented systems and is a way to make stored routines more flexible in handling a wide variety of data types and situations.

KNOWLEDGE CHECK

1. **A _____ is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updation of data.**
   a. Procedures
   b. Triggers
   c. Functions
   d. None of the mentioned
2. **Triggers are supported in**
   a. Delete
   b. Update
   c. Views
   d. All of the mentioned
3. **The CREATE TRIGGER statement is used to create the trigger. THE _____ clause specifies the table name on which the trigger is to be attached. The _____ specifies that this is an AFTER INSERT trigger.**
   a. for insert, on
   b. On, for insert
   c. For, insert
   d. None of the mentioned
4. **What are the after triggers?**
   a. Triggers generated after a particular operation
   b. These triggers run after an insert, update or delete on a table
   c. These triggers run after an insert, views, update or delete on a table
   d. All of the mentioned
5. **The variables in the triggers are declared using**
   a. –
   b. @
   c. /
   d. /@
6. **A stored procedure in SQL is a_____**
   a. Block of functions
   b. Group of SQL statements.
   c. None

7. **Advantage of SQL stored procedure**
   a. Maintainability
   b. Re-use of code
   c. Security
   d. All
8. **Which statement(S) is/are incorrect**
   a. Stored procedure may return a value and function must return a value.
   b. Function has only IN parameter.
   c. Try and Catch can be used with both stored procedure and function.
   d. Stored procedure has IN and OUT parameter.
9. **Which statement(S) is/are incorrect**
   a. Stored procedure can be shared by multiple programs
   b. Stored procedures are in compiled form.
   c. Stored procedure is a group of SQL statements
   d. All are correct.
10. **PL/SQL stands for -**
   a. Portable Language/SQL
   b. Programming Language/SQL
   c. Procedural Language/SQL
   d. none of these

REVIEW QUESTIONS

1. What is stored procedures in database?
2. What are the advantages of stored procedures?
3. Describe the advantages and disadvantages of triggers.
4. What is trigger in Oracle PL SQL?
5. Distinguish between row level trigger and statement level trigger.

Check Your Result

1. (b)	2. (c)	3. (b)	4. (b)	5. (b)
6. (b)	7. (d)	8. (c)	9. (d)	10. (c)

REFERENCES

1. Allen, Grant (2010). The Definitive Guide to SQLite. Apresspod. Mike Owens (2 ed.). Apress. pp. 90–91. ISBN 9781430232254. Retrieved 2012-10-02.
2. Feuerstein, Steven; Bill Pribyl (2014). Oracle PL/SQL Programming (6th ed.). O'Reilly & Associates. ISBN 978-1449324452.
3. Gupta, Saurabh K. (2016) [2012]. "5: Using Advanced Interface Methods". Advanced Oracle PL/SQL Developer's Guide. Professional experience distilled (2 ed.). Birmingham: Packt Publishing Ltd. p. 143. ISBN 9781785282522. Retrieved 2017-06-08.
4. Laudenschlager, Douglas; Milener, Gene; Guyer, Craig; Byham, Rick. "Transactions (Transact-SQL)". Microsoft Docs. Microsoft. Retrieved 12 November 2018.
5. Nanda, Arup; Burleson, Donald K. (2003). "9". In Burleson, Donald K. (ed.). Oracle Privacy Security Auditing: Includes Federal Law Compliance with HIPAA, Sarbanes Oxley and the Gramm Leach Bliley Act GLB. Oracle in-focus series. 47. Kittrell, North Carolina: Rampant TechPress. p. 511. ISBN 9780972751391. Retrieved 2018-04-17.
6. Nanda, Arup; Feuerstein, Steven (2005). Oracle PL/SQL for DBAs. O'Reilly Series. O'Reilly Media, Inc. pp. 122, 429. ISBN 978-0-596-00587-0. Retrieved 2011-01-11.
7. Naudé, Frank (June 9, 2005). "Oracle PL/SQL FAQ rev 2.08".

Index

3G E-LEARNING

Level: Beginner to Advanced
Subject: Computer and Information Science

Basic Computer Coding: SQL

2nd Edition

Basically, SQL stands for Structured Query Language which is basically a language used by databases. SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. It has been around in some form since the 70s and is just about as ubiquitous as data management itself. In order to get the most of the mounds of data they collect, many businesses must become versed in SQL. Now into its third decade of existence, SQL offers great flexibility to users by supporting distributed databases, i.e. databases that can be run on several computer networks at a time. SQL is used in health care, business (inventories, trends analysis), and education. It even has applications in the defense industry.

This updated edition is systematically divided into ten chapters. You'll quickly learn how to put the power and flexibility of this language to work. This book is a guide to SQL covers such topics as retrieving records, metadata queries, working with strings, data arithmetic, date manipulation, reporting and warehousing, and hierarchical queries.

ISBN 978-1-98467-606-1

BIBLIOTEX
Digital Library

www.3ge-learning.com
email: info@3ge-learning.com