

SQL PROGRAMMING

2

SQL PROGRAMMING

1

SQL

2 BOOKS IN 1

THE ULTIMATE BEGINNER'S & INTERMEDIATE GUIDE
TO LEARN SQL PROGRAMMING STEP BY STEP



RYAN TURNER

SQL: 2 BOOKS IN 1

*The Ultimate Beginner's & Intermediate Guide to Learn Python
Machine Learning Step by Step using Scikit-Learn and Tensorflow*

RYAN TURNER

© Copyright 2018 - All rights reserved.

The content contained within this book may not be reproduced, duplicated or transmitted without direct written permission from the author or the publisher.

Under no circumstances will any blame or legal responsibility be held against the publisher, or author, for any damages, reparation, or monetary loss due to the information contained within this book. Either directly or indirectly.

Legal Notice:

This book is copyright protected. This book is only for personal use. You cannot amend, distribute, sell, use, quote or paraphrase any part, or the content within this book, without the consent of the author or publisher.

Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. All effort has been executed to present accurate, up to date, and reliable, complete information. No warranties of any kind are declared or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice. The content within this book has been derived from various sources. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, — errors, omissions, or inaccuracies.

TABLE OF CONTENTS

SQL: THE ULTIMATE BEGINNER'S GUIDE

Introduction

1. [Advantages Of Databases](#)
2. [Types Of Sql Queries](#)

Chapter 1: The Data Definition Language (Ddl)

3. [Ddl For Database And Table Creation](#)
4. [Alter Ddl For Foreign Key Addition](#)
5. [Foreign Key Ddl In Tables](#)
6. [Unique Constraint Ddl In Tables](#)
7. [Delete And Drop Ddl In Tables](#)
8. [Ddl To Create Views](#)

Chapter 2: Sql Joins And Union

1. [Sql Inner Join](#)
2. [Sql Right Join](#)
3. [Sql Left Join](#)
4. [Sql Union Command](#)
5. [Sql Union All Command](#)

Chapter 3: How To Ensure Data Integrity

[Integrity Constraints – The Basics](#)
[The Not Null Constraint](#)
[The Unique Constraint](#)
[The Primary Key Constraint](#)
[The Foreign Key Constraints](#)
[The Check Constraint](#)

Chapter 4: How To Create An Sql View

[How To Add A View To A Database](#)
[How To Create An Updateable View](#)
[How To Drop A View](#)
[Database Security](#)
[The Security Model Of Sql](#)
[Creating And Deleting A Role](#)
[Granting And Revoking A Privilege](#)

Chapter 5: Database Creation

[Creating A Database](#)
[Removing A Database](#)
[Schema Creation](#)
[Creating Tables And Inserting Data Into Tables](#)
[How To Create A Table](#)
[Creating A New Table Based On Existing Tables](#)

[Inserting Data Into Table](#)
[Populating A Table With New Data](#)
[Inserting Data Into Specific Columns](#)
[Inserting Null Values](#)
[Data Definition Language \(Ddl\)](#)
[Running The Ddl Script](#)
[Data Manipulation Language \(Dml\)](#)
[Running The Dml Script](#)

Chapter 6: Database Administration

Chapter 7: Sql Transaction

Chapter 8: Logins, Users And Roles

[Like Clause](#)
[Sql Functions](#)
[Sql Avg Function](#)
[Sql Round Function](#)
[Sql Sum Function](#)
[Sql Max\(\) Function](#)

Chapter 9: Modifying And Controlling Tables

[Modifying Column Attributes](#)
[Renaming Columns](#)
[Deleting A Column](#)
[Adding A New Column](#)
[Modifying An Existing Column Without Changing Its Name](#)
[Rules To Remember When Using Alter Table](#)
[Deleting Tables](#)
[Combining And Joining Tables](#)
[Sql Character Functions](#)

Note:

[Sql Constraints](#)
[Not Null Constraint](#)
[Default Constraint](#)
[Unique Constraint](#)
[Primary Key](#)
[Foreign Key](#)
[Check Constraint](#)
[Index Constraint](#)

Conclusion And Next Steps

References

SQL: THE ULTIMATE INTERMEDIATE GUIDE

Introduction

Chapter 1: The Simple Basics Of Sql

[What Is Sql?](#)

[Working On The Databases](#)

[Relational Database](#)

[Client And Server Technology](#)

[Internet-Based Database Systems](#)

Chapter 2: Installing And Configuring Mysql

[What Is Mysql?](#)

[How To Install Mysql On Microsoft Windows On Your Computer](#)

Chapter 3: The Sql Server

[Installing Oracle Database 11g Express Edition](#)

[Installing Sql Developer](#)

[Creating A System Connection](#)

[Creating A Database Account](#)

[Creating Your Account Connection](#)

[Showing Line Numbers](#)

[Deleting The System Connection](#)

[Using Sql Developer](#)

[Running An Sql Statement](#)

[Inserting Rows](#)

[Running A Pl/Sql Program](#)

[Multiple Worksheets For A Connection](#)

[Storing The Source Code](#)

[How To Open A Source Code](#)

[Storing The Listings In Appendix A In Files](#)

[Running Sql Or Pl/Sql From A File](#)

[Clearing A Worksheet](#)

[Displaying Output](#)

[Clearing Dbms _Output](#)

Chapter 4: Data Types

[Categories Of Data Types](#)

[Character](#)

[Number](#)

[Date And Time](#)

[Arithmetic](#)

Chapter 5: Sql Script Boundaries

[Dropping Restricts](#)

[Integrity Restricts](#)

[Delegating Responsibility](#)

Chapter 6: Filters

[Where Clause](#)

[Having Clause](#)

[Evaluating A Condition](#)

[And Operator](#)

[Or Operator](#)

[Usage Of Parentheses](#)

[The Not Operator](#)

[Sequences](#)

[Renumbering A Sequence](#)

[Chapter 7: Views](#)

[Encrypting A View](#)

[Creating A View](#)

[Indexing A View](#)

[Chapter 8: Triggers](#)

[Trigger Syntax](#)

[After Trigger](#)

[Instead Of Trigger](#)

[Chapter 9: Stored Routines And Functions In Sql](#)

[Stored Procedures](#)

[Benefits Offered By Stored Procedures In Sql](#)

[Creating A Stored Procedure](#)

[Executing A Stored Procedure](#)

[Inserting Records](#)

[Updating Records](#)

[Deleting Records](#)

[Modifying A Stored Procedure](#)

[Deleting A Stored Procedure](#)

[Functions](#)

[Scalar Functions](#)

[Table-Valued Functions](#)

[Notes On Functions](#)

[Cursors](#)

[Chapter 10: How To Pivot Data](#)

[How Can You Compose Your Pivot Query?](#)

[Chapter 11: Clone Tables](#)

[How To Create A Clone Table](#)

[Purposes Of Clone Tables](#)

[Chapter 12: Normalization Of Your Data](#)

[How To Normalize The Database](#)

[Raw Databases](#)

[Logical Design](#)

[The Needs Of The End User](#)

[Data Repetition](#)

[Normal Forms](#)

[Naming Conventions](#)

[Benefits Of Normalizing Your Database](#)

[Denormalization](#)

[Database Normal Forms](#)

[First Normal Form \(1nf\)](#)

[Second Normal Form \(2nf\)](#)

[Third Normal Form \(3nf\)](#)

[Boyce-Codd Normal Form \(Bcnf\)](#)

[Fourth Normal Form \(4nf\)](#)

[Fifth Normal Form \(5nf\)](#)

Chapter 13: Security

[Components Of Database Security](#)

[Authentication](#)

[Encryption](#)

[Authorization](#)

[Access Control](#)

[Three-Class Security Model](#)

[Schemas](#)

[Server Roles](#)

[Logins](#)

[Mixed Mode Authentication](#)

[Database Roles](#)

[Encryption](#)

[Master Keys](#)

[Transparent Data Encryption \(Tde\)](#)

Chapter 14: Sql Injections

[How Do They Work?](#)

[Preventing An Sql Injection](#)

[Like Quandary](#)

[Hacking Scenario](#)

Chapter 15: Fine-Tune Your Indexes

[Sql Tuning Tools](#)

Chapter 16: Deadlocks

[Deadlock Analysis And Prevention](#)

[Reading Deadlock Info Via Sql Server Error Log](#)

[Some Tips For Avoiding Deadlock](#)

Chapter 17: Database Administration

[Setting Up A Maintenance Plan In Sql Server](#)

[Define Backup Database \(Full\) Task](#)

[Define Database Check Integrity Task](#)

[Define Shrink Database Task](#)

[Define Reorganize Index Task](#)

[Define Rebuild Index Task](#)

[Define Update Statistics Task](#)

[Define History Cleanup Task](#)

[Define Backup Database \(Differential\) Task](#)

[Define Backup Database \(Transaction Log\) Task](#)

[Define Execute Sql Server Agent Job Task](#)

[Define Maintenance Cleanup Task](#)

[Report Options](#)

[Complete The Wizard](#)

[Running The Maintenance Plan](#)

[*Emailing The Reports*](#)
[*Configuring The Database Mail*](#)
[*Sql Server Agent*](#)

Backup And Recovery

[*The Transaction Log*](#)
[*Recovery*](#)
[*Changing The Recovery Model*](#)
[*Backups*](#)
[*Performing A Backup*](#)
[*Restoring A Database*](#)
[*Attaching And Detaching Databases*](#)

Chapter 18: Working With Ssms

[*Downloading Sql Server Management Studio \(Ssms\)*](#)
[*Starting The Database Engine Services*](#)
[*Connect To Sql Server With Ssms*](#)

The Basics And Features Of Ssms

[*Managing Connections*](#)
[*Choosing Your Database*](#)
[*New Query Window*](#)
[*Executing Statements*](#)
[*Intellisense*](#)
[*Results Presentation*](#)
[*Object Explorer*](#)
[*Databases*](#)

Chapter 19: Real-World Uses

[*Sql In An Application*](#)

Conclusion

References

SQL

**THE ULTIMATE BEGINNER'S GUIDE TO LEARN SQL
PROGRAMMING STEP BY STEP**

INTRODUCTION

“Never write when you can talk. Never talk when you can nod. And never put anything in an e-mail.” – Eliot Spitzer

On a hard disk, data can be stored in different file formats. It can be stored in the form of text files, word files, mp4 files, etc. However, a uniform interface that can provide access to different types of data under one umbrella in a robust and efficient manner is required. Here, the role of databases emerges.

The definition of a database is “a collection of information stored in computer in a way that it can easily be accessed, managed and manipulated.”

Databases store data in the form of a collection of tables where each table stores data about a particular entity. The information that we want to store about students will be represented in the columns of the table. Each row of the table will contain the record of a particular student. Each record will be distinguished by a particular column, which will contain a unique value for each row.

Suppose you want to store the ID, name, age, gender, and department of a student. The table in the database that will contain data for this student will look like this:

SID	SName	SAge	SGender	SDepartment
1	Tom	14	Male	Computer
2	Mike	12	Male	Electrical
3	Sandy	13	Female	Electrical
4	Jack	10	Male	Computer
5	Sara	11	Female	Computer

Student Table

Here, the letter “S” has been prefixed with the name of each column. This is just one of the conventions used to denote column names. You can give any name to the columns. (We will look at how to create tables and columns within it in the coming chapters.) It is much easier to access, manipulate, and manage data stored in this form. SQL queries can be executed on the data stored in the form of tables that have relationships with other tables.

A database doesn’t contain a single table. Rather, it contains multiple related tables. Relationships maintain database integrity and prevent data redundancy.

For instance, if the school decides to rename the Computer department from “Computer” to “Comp & Soft,” you will have to update the records of all students in the Computer department. You will have to update the 1st, 4th, and 5th records of the student table.

It is easy to update three records; however, in real life scenarios, there are thousands of students and it is an uphill task to update the records of all of them. In such scenarios, relationships between data tables become important. For instance, to solve the aforementioned redundancy problem, we can create another table named Department and store the records of all the departments in that table. The table will look like this:

DID	DName	DCapacity
101	Electrical	800
102	Computer	500
103	Mechanical	500

Department Table

Now, in the student table, instead of storing the department name, the department ID will be stored. The student table will be updated like this:

SID	SName	SAge	SGender	DID
1	Tom	14	Male	102
2	Mike	12	Male	101
3	Sandy	13	Female	101
4	Jack	10	Male	102
5	Sara	11	Female	102

Table Student

You can see that the department name column has been replaced by the department id column, represented by “DID”. The 1st, 4th, and 5th rows that were previously assigned the “Computer” department now contain the id of the department, which is 102. Now, if the name of the department is changed from “Computer” to “Comp & Soft”, this change has to be made only in one record of the department table and all the associated students will be automatically referred to the updated department name.

1. Advantages of Databases

The following are some of the major advantages of databases:

- Databases maintain data integrity. This means that data changes are carried out at a single place and all the entities accessing the data get the latest version of the data.
- Through complex queries, databases can be efficiently accessed, modified, and manipulated. SQL is designed for this purpose.
- Databases avoid data redundancy. Through tables and relationships, databases avoid data redundancy and data belonging to particular entities resides in a single place in a database.
- Databases offer better and more controlled security. For example, usernames and passwords can be stored in tables with excessive security levels.

2. Types of SQL Queries

On the basis of functionality, SQL queries can be broadly classified into a couple of major categories as follows:

- **Data Definition Language (DDL)**

Data Definition Language (DDL) queries are used to create and define schemas of databases. The following are some of the queries that fall in this category:

1. CREATE – to create tables and other objects in a database
2. ALTER – to alter database structures, mainly tables.
3. DROP - delete objects, mainly tables from the database

- **Data Manipulation Language**

Data Manipulation Language (DML) queries are used to manipulate data within databases. The following are some examples of DML queries.

1. SELECT – select data from tables of a database
2. UPDATE - updates the existing data within a table
3. DELETE - deletes all rows from a table, but the space for the record remains

CHAPTER 1: THE DATA DEFINITION LANGUAGE (DDL)

“Every man has a right to his opinion, but no man has a right to be wrong in his facts. “– Bernard Mannes Baruch

SQL data definition language is used to define new databases, data tables, delete databases, delete data tables, and alter data table structures with the following key words; create, alter and drop. In this chapter, we will have a detailed discussion about the SQL Data Definition Language in a practical style.

3. DDL for Database and Table Creation

The database creation language is used to create databases in a database management system. The language syntax is as described below:

CREATE DATABASE my_Database

For example, to create a customer_details database in your database management system, use the following SQL DDL statement:

CREATE DATABASE customer_details

Please remember that SQL statement is case insensitive. Next, we need to create the various customer tables that will hold the related customers records, in the earlier created ‘customer_details’ database. This is why the system is called a relational database management system, as all tables are related for easy record retrieval and information processing. To create the different but related customer tables in the customer_details database, we apply the following DDL syntax:

CREATE TABLE my_table

(

table_column-1 data_type,

table_column-2 data_type,

table_column-3 data_type,

table_column-n data_type

```
)  
CREATE TABLE customer_accounts  
(  
  acct_no INTEGER, PRIMARY KEY,  
  acct_bal  DOUBLE,  
  acct_type INTEGER,  
  acct_opening_date DATE  
)\[1\]
```

The attribute “PRIMARY KEY” ensures the column named ‘acct_no’ has unique values throughout, with no null values. Every table should have a primary key column to uniquely identify each record of the table. Other column attributes are ‘NOT NULL’ which ensures that a null value is not accepted into the column, and ‘FOREIGN KEY’ which ensures that a related record in another table is not mistakenly or unintentionally deleted. A column with a ‘FOREIGN KEY’ attribute is a copy of a primary key column in another related table. For example, we can create another table ‘customer_personal_info’ in our ‘customer_details’ database like below:

```
CREATE TABLE customer_personal_info  
(  
  cust_id      INTEGER PRIMARY KEY,  
  first_name   VARCHAR(100) NOT NULL,  
  second_name  VARCHAR(100),  
  lastname     VARCHAR(100) NOT NULL,  
  sex          VARCHAR(5),  
  date_of_birth DATE,  
  address      VARCHAR(200)  
)\[2\]
```

The newly created ‘customer_personal_info’ table has a primary key column

named 'cust_id'. The 'customer_accounts' table needs to include a column named 'cust_id' in its field to link to the 'customer_personal_info' table in order to access the table for more information about the customer with a given account number.

Therefore, the best way to ensure data integrity between the two tables, so that an active record in a table is never deleted is to insert a key named 'cust_id' in the 'customer_accounts' table as a foreign key. This ensures that a related record in 'customer_personal_info' to another in 'customer_accounts' table is never accidentally deleted. We will discuss how to go about this in the next section.

4. Alter DDL for Foreign Key Addition

Since 'customer_accounts' table is already created, we need to alter the table to accommodate the new foreign key. To achieve this, we use the SQL Data Definition Language syntax described below:

```
ALTER TABLE mytable  
ADD FOREIGN KEY (targeted_column)  
REFERENCES related_table(related_column)
```

Now, to add the foreign key to the the 'customer_accounts' table and make it reference the key column 'cust_id' of table 'customer_personal_info', we use the following SQL statements:

```
ALTER TABLE customer_accounts  
ADD FOREIGN KEY (cust_id)  
REFERENCES customer_personal_info(cust_id)
```

5. Foreign Key DDL in Tables

In situations where we need to create foreign keys as we create new tables, we make use of the following DDL syntax:

```
CREATE TABLE my_table  
(  
Column-1 data_type FOREIGN KEY, REFERENCES (related column)
```

)

6. Unique Constraint DDL in Tables

The unique constraint can be placed on a table column to ensure that the values stored in the column are unique, just like the primary key column. The only difference is that you can only have one primary key column in a table but as many unique columns as you like. The DDL syntax for the creation of a unique table column or field is as described below:

```
CREATE TABLE my_table  
(  
Column-1 data_type UNIQUE  
)
```

7. Delete and Drop DDL in Tables

DROP TABLE my_table

It must be noted that this action is irreversible and plenty of care must be taken before doing this. Also, a database can be deleted with the following DDL syntax:

DROP DATABASE my_database

This action is also irreversible. At the atomic level, you may decide to delete a column from a data table. To achieve this, we use the 'Delete' DDL rather than 'DROP'. The syntax is as below:

DELETE column_name FROM data_table

8. DDL to Create Views

```
CREATE VIEW virtual_table AS  
SELECT column-1, column-2, ..., column-n  
FROM data_table
```

WHERE column-n operator value^[3]

For example, given a table (customer_personal_info) like the one below:

cust_id	first_nm	second_nm	lastname	sex	Date_of_birth
03051	Mary	Ellen	Brown	Female	1980-10-19
03231	Juan	John	Maslow	Female	1978-11-18
03146	John	Ken	Pascal	Male	1983-07-12
03347	Alan	Lucas	Basal	Male	1975-10-09

Table 2.1 customer_personal_info

To create a view of female customers from the table, we use the following SQL Create View statement:

```
CREATE VIEW [Female Depositors] AS  
SELECT cust_id, first_nm, second_nm, lastname, sex, date_of_birth  
FROM customer_personal_info  
WHERE sex = 'Female'[4]
```

To query and display the records of the created view, use the following select statement:

```
SELECT * FROM [Female Depositors]
```

The execution of the above select statement would generate the following view:

cust_id	first_nm	second_nm	lastname	sex	Date_of_birth
03051	Mary	Ellen	Brown	Female	1980-10-19
03231	Juan		Maslow	Female	1978-11-18

However, it must be noted that a view is never stored in memory but recreated when needed. A view can be processed just like you would process a real table.

CHAPTER 2: SQL JOINS AND UNION

“I have noted that persons with bad judgment are most insistent that we do what they think best”. – Lionel Abe

The select statement can be made to process more than one table at the same time in a single select statement with the use of the logical operators ('OR' and 'AND'). It can sometimes be more efficient to use the Left, right and the inner join operator for more efficient processing.

1. SQL INNER JOIN

The SQL 'INNER JOIN' can also be used to process more than one data table in an SQL query or statement. However, the data tables must have relating columns (the primary and its associated foreign key columns). The "INNER JOIN" keyword returns records from two data tables if a match is found between columns in the affected tables. The syntax of usage is as described below:

```
SELECT column-1, column-2... column-n
FROM data_table-1
INNER JOIN data_table-2
ON data_table-1.keycolumn = data_table-2.foreign_keycolumn
```

For example, to select customer acct_no, first_name, surname, sex and account bal data from across the two tables 'customer_accounts' (table 4.1) and 'customer_personal_info' (table 4.2) based on matching columns 'cust_id', we use the following SQL INNER JOIN query:

acct_no	cust_id	acct_bal	acct_type	acct_opening_date
0411003	03231	2540.33	100	2000-11-04
0412007	03146	10350.02	200	2000-09-13
0412010	03347	7500.00	200	2002-12-05

Table 4.1 customer_accounts

cust_id	first_nm	second_nm	lastname	sex	Date_of_birth	addr

03051	Mary	Ellen	Brown	Female	1980-10-19	Coventry
03231	Juan		Maslow	Female	1978-11-18	York
03146	John	Ken	Pascal	Male	1983-07-12	Liverpool
03347	Alan		Basal	Male	1975-10-09	Easton

Table 4.2 customer_personal_info

```
SELECT a.acct_no, b.first_nm AS first_name, b. lastname AS surname, b.sex,
a.acct_bal
```

```
FROM customer_accounts AS a
```

```
INNER JOIN customer_personal_info AS b
```

```
ON b.cust_id = a.cust_id[5]
```

The above SQL query would produce the result set in table 4.3 below:

acct_no	first_name	surname	sex	acct_bal
0411003	Juan	Maslow	Female	2540.33
0412007	John	Pascal	Male	10350.02
0412010	Alan	Basal	Male	7500.00

Table 4.3

2. SQL RIGHT JOIN

When used with the SQL select statement, The “RIGHT JOIN” keyword includes all records in the right data table even when no matching records are found in the left data table. The syntax of usage is as described below:

```
SELECT column-1, column-2... column-n
```

```
FROM left-data_table
```

```
RIGHT JOIN right-data_table
```

```
ON left-data_table.keyColumn = right-data_table.foreign_keycolumn
```

For example, to select customers’ acct_no, first_name, surname, sex and account_bal details across the two tables, customer_accounts and customer_personal_info display customer personal information whether they have

an active account or not. We use the following SQL ‘RIGHT JOIN’ query:

```
SELECT a.acct_no, b.first_nm AS first_name, b.lastname AS surname, b.sex,
a.acct_bal
FROM customer_accounts AS a
RIGHT JOIN customer_personal_info AS b
ON b.cust_id = a.cust_id
```

The output of execution of the above SQL RIGHT JOIN query is presented in table 4.4 below:

acct_no	first_name	surname	sex	acct_bal
0411003	Juan	Maslow	Female	2540.33
0412007	John	Pascal	Male	10350.02
0412010	Alan	Basal	Male	7500.00
	Mary	Brown	Female	

Table 4.4

3. SQL LEFT JOIN

The ‘LEFT JOIN’ is used with the SQL select statement to return all records in the left data table (first table) even if there were no matches found in the relating right table (second table). The syntax of usage is as described below:

```
SELECT column-1, column-2... column-n
FROM left-data_table
LEFT JOIN right-data_table
ON left-data_table.keyColumn = right-data_table.foreign_keycolumn
```

For example, to display all active accounts holders’ details across the ‘customer_accounts and ‘customer_personal_info’ tables, we use the following SQL ‘LEFT JOIN’ query:

```
SELECT a.acct_no, b.first_nm AS first_name, b.lastname AS surname, b.sex,
```

```

a.acct_bal
FROM customer_accounts AS a
LEFT JOIN customer_personal_info AS b
ON b.cust_id = a.cust_id[7]

```

The above SQL LEFT JOIN query would produce the result set in table 4.5 below:

acct_no	first_name	surname	sex	acct_bal
0411003	Juan	Maslow	Female	2540.33
0412007	John	Pascal	Male	10350.02
0412010	Alan	Basal	Male	7500.00

Table 4.5

4. SQL UNION Command

Two or more SQL select statements' result sets can be joined with the 'UNION' command. The syntax of usage is as described below:

```

SELECT column-1, column-2... column-n
FROM data_table-1
UNION
SELECT column-1, column-2... column-n
FROM data_table-2

```

Source^[8]

For example, to display a list of our phantom bank's customers in UK and US branches, you may display one record for customers that have accounts in both branches according to the following tables 4.6 and 4.7:

acct_no	first_name	surname	sex	acct_bal
0411003	Juan	Maslow	Female	2540.33

0412007	John	Pascal	Male	10350.02
0412010	Alan	Basal	Male	7500.00

Table 4.6 London_Customers

acct_no	first_name	surname	sex	acct_bal
0413112	Deborah	Johnson	Female	4500.33
0414304	John	Pascal	Male	13360.53
0414019	Rick	Bright	Male	5500.70
0413014	Authur	Warren	Male	220118.02

Table 4.7 Washington_Customers

We use the following SQL query:

```
SELECT first_name, surname, sex, acct_bal
```

```
FROM London_Customers
```

```
UNION
```

```
SELECT first_name, surname, sex, acct_bal
```

```
FROM Washington_Customers[9]
```

The following is the result set from the execution of the above SQL query:

first_name	surname	sex	acct_bal
Juan	Maslow	Female	2540.33
John	Pascal	Male	10350.02
Alan	Basal	Male	7500.00
Deborah	Johnson	Female	4500.33
Rick	Bright	Male	5500.70
Authur	Warren	Male	220118.02

Table 4.8 SQL UNION

Note that the record for a customer (John Pascal) is only listed once, even if he is a customer of the two branches. This is because the UNION command lists only distinct records across tables. So, to display all records across associated tables, use 'UNION ALL' instead.

5. SQL UNION ALL Command

The UNION ALL command is basically the same as the UNION command, except that it displays all records across unionized tables, as explained earlier. The syntax of usage is as described below:

```
SELECT column-1, column-2... column-n
FROM data_table-1
UNION
SELECT column-1, column-2... column-n
FROM data_table-2
```

Apply the 'UNION ALL' command on the data of the two tables 'London_Customers' and 'Washington_Customers' with the SQL query below:

```
SELECT first_name, surname, sex, acct_bal
FROM London_Customers
UNION ALL
SELECT first_name, surname, sex, acct_bal
FROM Washington_Customers[10]
```

The records would then be displayed in two tables, as shown below:

first_name	surname	sex	acct_bal
Juan	Maslow	Female	2540.33
John	Pascal	Male	10350.02
Alan	Basal	Male	7500.00
Deborah	Johnson	Female	4500.33
John	Pascal	Male	13360.53
Rick	Bright	Male	5500.70

Authur	Warren	Male	220118.02
--------	--------	------	-----------

Table 4.9

CHAPTER 3: HOW TO ENSURE DATA INTEGRITY

“The possession of facts is knowledge, the use of them is wisdom.” – Thomas Jefferson

SQL databases don't just store information. If the information's integrity has been compromised, its reliability becomes questionable. If the data is unreliable, the database that contains it also becomes unreliable.

To secure data integrity, SQL offers a wide range of rules that can limit the values a table can hold. These rules, known as "integrity constraints," work on columns and tables. This chapter will explain each kind of constraint.

Integrity Constraints – The Basics

SQL users divide integrity constraints into the following categories:

The Assertions – You need to define this constraint inside a separate definition (which is called the “assertion definition”). This means that you don't indicate an assertion in your table's definition. In SQL, you may apply an assertion to multiple tables.

The Table-Related Constraints – This is a constraint that you need to define inside a table's definition. You may define a constraint as a component of a table or column's definition.

The Domain Constraints – Similar to the assertions, you need to create domain constraints in a separate definition. This kind of constraint works on the column/s that you declared inside the domain involved.

Table-related constraints offer various constraint options. Consequently, these days, it is the most popular category of integrity constraints. You can divide this category into two: column constraints and table constraints. The former belong to the definition of a column. The latter, on the other hand, act as elements of a table.

The table and column constraints work with different kinds of constraints. The domain constraints and assertions, however, can only work with one constraint type.

The Not Null Constraint

In the previous chapter, you learned that “null” represents an unknown/undefined value. Keep in mind that undefined/unknown is different from zeroes, blanks, default values, and empty strings. Rather, it signifies the absence of a value. You may consider this value as a “flag” (i.e. a bit, number, or character that expresses some data regarding a column). If you leave a column empty, and the value is therefore null, the system will place the “flag” to indicate that it’s an unknown value.

Columns have an attribute called “nullability.” This attribute shows whether the columns can take unknown values or not. In SQL, columns are set to take null values. However, you may change this attribute according to your needs. To disable the nullability of a column, you just have to use the NOT NULL constraint. This constraint informs SQL that the column won't accept any null value.

In this language, you need to use NOT NULL on a column. That means you can't use this constraint on an assertion, domain constraint, or table-based constraint. Using NOT NULL is a simple process. Just add the syntax below to your column definition:

```
(name of column) [ (domain) | (data type) ] NOT NULL
```

As an example, let's assume that you need to generate a table called FICTION_NOVEL_AUTHORS. This table needs to have three columns: AUTHOR_ID, AUTHOR_NAME, and AUTHOR_DOB. You need to ensure that each entry you add has values for AUTHOR_ID and AUTHOR_NAME. To accomplish this, you must insert the NOT NULL constraint into the definition of both columns. Here's the code:

```
CREATE TABLE FICTION_NOVEL_AUTHORS  
( AUTHOR_ID    INT    NOT NULL ,  
  AUTHOR_NAME  CHARACTER(50) NOT NULL ,  
  AUTHOR_DOB   CHARACTER(50) );\[11\]
```

As you can see, this code did not set NOT NULL for the AUTHOR_DOB column. Consequently, if a new entry doesn't have any value for AUTHOR_DOB, the system will insert a null value in that column.

The Unique Constraint

Table and column constraints accept unique constraints. In SQL, unique constraints belong to one of these two types:

1. UNIQUE
2. PRIMARY KEY

Important Note: This part of the book will concentrate on the first type. You'll learn about the second one later.

Basically, you can use UNIQUE to make sure that a column does not accept duplicate values. This constraint will stop you from entering a value that already exists in the column.

Let's assume that you want to apply this constraint on the AUTHOR_DOB column. This way, you can make sure that the values inside that column are all unique. Now, let's say you realized that requiring dates of birth to be unique is a bad idea since people may be born on the same date. You may adjust your approach by placing the UNIQUE constraint on AUTHOR_NAME and AUTHOR_DOB. Here, the table will stop you from repeating an AUTHOR_NAME/AUTHOR_DOB pair. You may repeat values in the AUTHOR_NAME and AUTHOR_DOB columns. However, you can't reenter an exact pair that already exists in the table.

Keep in mind that you may tag UNIQUE constraints as table constraints or column constraints. To generate column constraints, add them to the definition of a column. Here is the syntax:

```
(name of column) [ (domain) | (data type) ] UNIQUE
```

If you need to use the UNIQUE constraint on a table, you must insert it into the table definition as an element. Here is the code:

```
{ CONSTRAINT (name of constraint) }
```

```
UNIQUE < (name of column) { [, (name of column) ] ... } >
```

As the syntax above shows, using UNIQUE on a table is more complicated than using the constraint on a column. However, you cannot apply UNIQUE on multiple columns. Regardless of how you use this constraint (i.e. either as a table constraint or a column constraint), you may define any number of UNIQUE constraints within each table definition.

Let's apply this constraint on a columnar level:

```
CREATE TABLE BOOK_LIBRARY  
( AUTHOR_NAME    CHARACTER (50) ,  
  BOOK_TITLE     CHARACTER (70) UNIQUE,  
  PUBLISHED_DATE INT ) ;\[12\]
```

You may also use UNIQUE on other columns. However, its result would be different than if we had used a table constraint on multiple columns. The following code will illustrate this idea:

```
CREATE TABLE BOOK_LIBRARY  
( AUTHOR_NAME    CHARACTER (50) ,  
  BOOK_TITLE     CHARACTER (70) ,  
  PUBLISHED_DATE INT,  
  CONSTRAINT UN_AUTHOR_BOOK UNIQUE ( AUTHOR_NAME,  
  BOOK_TITLE ) ) ;\[13\]
```

Now, for the table to accept a new entry, the AUTHOR_NAME and BOOK_TITLE columns must have unique values.

As you've read earlier, the UNIQUE constraint ensures that one or more columns do not have duplicate values. That is an important rule to remember. However, you should also know that UNIQUE doesn't work on "null." Thus, a column will accept any number of null values even if you have set a UNIQUE constraint on it.

If you want to set your columns not to accept a null value, you must use NOT NULL. Let's apply NOT NULL on the column definition of BOOK_TITLE:

```
CREATE TABLE BOOK_LIBRARY  
( AUTHOR_NAME    CHARACTER (50) ,  
  BOOK_TITLE     CHARACTER (70)    UNIQUE NOT NULL,  
  PUBLISHED_DATE INT ) ;\[14\]
```

In SQL, you may also insert NOT NULL into column definitions that a table-level constraint is pointing to:

```
CREATE TABLE BOOK_LIBRARY  
( AUTHOR_NAME    CHARACTER (50) ,  
  BOOK_TITLE     CHARACTER (70) NOT NULL,  
  PUBLISHED_DATE INT,  
  CONSTRAINT UN_AUTHOR_BOOK UNIQUE (BOOK_TITLE) ) ;
```

Source https://www.w3schools.com/sql/sql_notnull.asp

In both cases, the BOOK_TITLE column gets the constraint. That means BOOK_TITLE won't accept null or duplicate values.

The PRIMARY KEY Constraint

The PRIMARY KEY constraint is almost identical to the UNIQUE constraint. You may use a PRIMARY KEY to prevent duplicate entries. In addition, you may apply it to multiple columns and use it as a table constraint or a column constraint. The only difference is that PRIMARY KEY has two distinct restrictions. These restrictions are:

If you apply PRIMARY key on a column, that column won't accept any null value. Basically, you won't have to use the NOT NULL constraint on a column that has PRIMARY KEY.

A table can't have multiple PRIMARY KEY constraints.

These restrictions exist because primary keys (also known as “unique identifiers”) play an important role in each table. As discussed in the first chapter, tables cannot have duplicate rows. This rule is crucial since the SQL language cannot identify redundant rows. If you change a row, all of its duplicates will also be affected.

You need to choose a primary key from the candidate keys of your database. Basically, candidate keys are groups of columns that identify rows in a unique manner. You may enforce a candidate key's uniqueness using UNIQUE or PRIMARY KEY. However, you must place one primary key on each table even if you did not define any unique constraint. This requirement ensures the uniqueness of each data row.

To define a primary key, you need to indicate the column/s you want to use. You can complete this task through PRIMARY KEY (i.e. the SQL keyword). This process is similar to the one discussed in the previous section. To apply PRIMARY KEY on a new column, use the following syntax:

```
(name of column) [ (domain) | (data type) ] PRIMARY KEY
```

To use PRIMARY key on a table, you must enter it as an element of the table you're working on. Check the syntax below:

```
{ CONSTRAINT (name of constraint) }
```

```
PRIMARY KEY < (name of column) {, (name of column) ] ... } >
```

SQL allows you to define primary keys using column constraints. However, you

can only use this feature on a single column. Analyze the following example:

```
CREATE TABLE FICTION_NOVEL_AUTHORS  
(AUTHOR_ID INT,  
AUTHOR_NAME CHARACTER (50) PRIMARY KEY ,  
PUBLISHER_ID INT ) ;\[15\]
```

If you want to apply PRIMARY KEY on multiple columns (or store it as another definition), you may use it on the tabular level:

```
CREATE TABLE FICTION_NOVEL_AUTHORS  
(AUTHOR_ID INT,  
AUTHOR_NAME CHARACTER (50) ,  
PUBLISHER_ID INT,  
CONSTRAINT PK_AUTHOR_ID PRIMARY KEY (AUTHOR_ID,  
AUTHOR_NAME ) ) ;
```

This approach places a primary key on two columns (i.e. AUTHOR_ID and AUTHOR_NAME). That means the paired values of the two columns need to be unique. However, duplicate values may exist inside any of the columns. Experienced database users refer to this kind of primary key as a “superkey.” The term “superkey” means that the primary key exceeds the number of required columns.

In most cases, you need to set both UNIQUE and PRIMARY KEY constraints on a table. To achieve this, you just have to define the involved constraints, as usual. For instance, the code given below applies both of these constraints:

```
CREATE TABLE FICTION_NOVEL_AUTHORS  
(AUTHOR_ID INT,  
AUTHOR_NAME CHARACTER (50) PRIMARY KEY ,  
PUBLISHER_ID INT,  
CONSTRAINT UN_AUTHOR_NAME UNIQUE (AUTHOR_NAME) ) ;
```

The following code will give you the same result:

```
CREATE TABLE FICTION_NOVEL_AUTHORS  
(AUTHOR_ID INT,  
AUTHOR_NAME CHARACTER (50) -> UNIQUE,  
PUBLISHER_ID INT,  
CONSTRAINT PK_PUBLISHER_ID PRIMARY KEY (PUBLISHER_ID) );
```

[.116\]](#)

The FOREIGN KEY Constraints

The constraints discussed so far focus on securing the data integrity of a table. NOT NULL stops columns from taking null values. PRIMARY KEY and UNIQUE, on the other hand, guarantee that the values of one or more columns are unique. In this regard, FOREIGN KEY (i.e. another SQL constraint) is different. FOREIGN KEY, also called “referential constraint,” focuses on how information inside a table works with the information within another table.

This connection ensures information integrity throughout the database. In addition, the connection between different tables results to "referential integrity." This kind of integrity makes sure that data manipulation done on one table doesn't affect the data inside other tables. The tables given below will help you understand this topic. Each of these tables, named PRODUCT_NAMES and PRODUCT_MANUFACTURERS, have one primary key:

PRODUCT_NAMES

PRODUCT_NAME_ID: INT	PRODUCT_NAME: CHARACTER (50)	MANUFACTURER_ID: INT
1001	X Pen	91
1002	Y Eraser	92
1003	Z Notebook	93

PRODUCT_MANUFACTURERS

MANUFACTURER_ID: INT	BUSINESS_NAME: CHARACTER (50)
91	THE PEN MAKERS INC.
92	THE ERASER MAKERS INC.
93	THE NOTEBOOK MAKERS INC.

The `PRODUCT_NAME_ID` column of the `PRODUCT_NAMES` table has a `PRIMARY KEY`. The `MANUFACTURER_ID` of the `PRODUCT_MANUFACTURERS` table has the same constraint. These columns are in yellow (see the tables above).

As you can see, the `PRODUCT_NAMES` table has a column called `MANUFACTURER_ID`. That column has the values of a column in the `PRODUCT_MANUFACTURERS` table. Actually, the `MANUFACTURER_ID` column of the `PRODUCT_NAMES` table can only accept values that come from the `MANUFACTURER_ID` column of the `PRODUCT_MANUFACTURERS` table.

Additionally, the changes that you'll make on the `PRODUCT_NAMES` table may affect the data stored in the `PRODUCT_MANUFACTURERS` table. If you remove a manufacturer, you also need to remove the entry from the `MANUFACTURER_ID` column of the `PRODUCT_NAMES` table. You can achieve this by using `FOREIGN KEY`. This constraint ensures the referential integrity of your database by preventing actions on any table from affecting the protected information.

Important Note: If a table has a foreign key, it is called “referencing table.” The table a foreign key points to is called “referenced table.”

When creating this kind of constraint, you need to obey the following guidelines:

You must define a referenced column by using `PRIMARY KEY` or `UNIQUE`. Most SQL programmers choose `PRIMARY KEY` for this purpose.

You may tag `FOREIGN KEY` constraints as column constraints or table constraints. You may work with any number of columns if you are using `FOREIGN KEY` as a table constraint. On the other hand, if you use this constraint at the column-level, you can only work on a single column.

A referencing table's foreign key should cover all of the columns you are trying to reference. In addition, the columns of the referencing table should match the data type of their counterparts (i.e. the columns being referenced). However, you don't have to use the same names for your referencing and referenced columns.

You don't need to indicate reference columns manually. If you don't specify any column for the constraint, SQL will consider the columns of the referenced table's primary key as the referenced columns. This process happens automatically.

You will understand these guidelines once you have analyzed the examples given below. For now, let's analyze the syntax of this constraint. Here's the format that you must use to apply FOREIGN KEY at the columnar level:

```
(name of column) [ (domain) | (data type) ] { NOT NULL }  
REFERENCES (name of the referenced table) { < (the referenced columns) > }  
{ MATCH [ SIMPLE | FULL | PARTIAL ] }  
{ (the referential action) } \[17\]
```

To use this FOREIGN KEY as a tabular constraint, you need to insert it as a table element. Here's the syntax:

```
{ CONSTRAINT (name of constraint) }  
FOREIGN KEY < (the referencing column) { [, (the referencing column) ] ... } >  
REFERENCES (the referenced table) { < (the referenced column/s) > }  
{ MATCH [ SIMPLE | FULL | PARTIAL ] }  
{ (the referential action) } \[18\]
```

You've probably noticed that FOREIGN KEY is more complex than the constraints you've seen so far. This complexity results from the constraint's option-filled syntax. However, generating this kind of constraint is easy and simple. Let's first analyze a basic example:

```
CREATE TABLE PRODUCT_NAMES  
( PRODUCT_NAME_ID -> INT,  
  PRODUCT_NAME -> CHARACTER (50) ,  
  MANUFACTURER_ID -> INT -> REFERENCES  
  PRODUCT_MANUFACTURERS ) ;
```

This code applies the constraint on the MANUFACTURER_ID column. To apply this constraint on a table, you just have to type REFERENCES and indicate the referenced table's name. In addition, the columns of this foreign key are equal to that of the referenced table's primary key. If you don't want to reference your target's primary key, you need to specify the column/s you want to use. For instance, REFERENCES PRODUCT_MANUFACTURERS (MANUFACTURER_ID).

Important Note: The FOREIGN KEY constraint requires an existing referenced table. In addition, that table must have a PRIMARY KEY or UNIQUE constraint.

For the second example, you will use FOREIGN KEY as a tabular constraint. The code that you see below specifies the referenced column's name, even if that information is not required.

```
CREATE TABLE PRODUCT_NAMES
( PRODUCT_NAME_ID INT,
  PRODUCT_NAME CHARACTER (50) ,
  MANUFACTURER_ID INT,
  CONSTRAINT TS_MANUFACTURER_ID FOREIGN KEY
  (MANUFACTURER_ID)
  REFERENCES PRODUCT_MANUFACTURERS (MANUFACTURER_ID) );
```

You may consider the two lines at the bottom as the constraint's definition. The constraint's name, TS_MANUFACTURER_ID, comes after the keyword CONSTRAINT. You don't need to specify a name for your constraints since SQL will generate one for you in case this information is missing. On the other hand, you may want to set the name of your constraint manually since that value appears in errors (i.e. when SQL commands violate an existing constraint). In addition, the names you will provide are more recognizable than system-generated ones.

Next, you should set the kind of constraint you want to use. Then, enter the name of your referencing column (MANUFACTURER_ID for the current example). You will then place the constraint on that column. If you are dealing with multiple columns, you must separate the names using commas. Afterward, type REFERENCES as well as the referenced table's name. Finally, enter the name of your referenced column.

That's it. Once you have defined this constraint, the MANUFACTURER_ID column of PRODUCT_NAMES won't take values except those that are already listed in the PRODUCT_MANUFACTURERS table's primary key. As you can see, a foreign key doesn't need to hold unique values. You may repeat the values inside your foreign keys as many times as you want, unless you placed the UNIQUE constraint on the column you're working on.

Now, let's apply this constraint on multiple columns. You should master this technique before studying the remaining elements of the constraint's syntax. For this example, let's use two tables: BOOK_AUTHORS and BOOK_GENRES.

The table named BOOK_AUTHORS has a primary key defined in the AUTHOR_NAME and AUTHOR_DOB columns. The SQL statement found below generates a table called BOOK_GENRES. This table has a foreign key consisting of the AUTHOR_DOB and DATE_OF_BIRTH columns.

```
CREATE TABLE BOOK_GENRES
(AUTHOR_NAME  CHARACTER (50) ,
DATE_OF_BIRTH  DATE,
GENRE_ID  INT,
CONSTRAINT TS_BOOK_AUTHORS FOREIGN KEY ( AUTHOR_NAME,
DATE_OF_BIRTH ) REFERENCES BOOK_AUTHORS (AUTHOR_NAME,
AUTHOR_DOB) );[20]
```

This code has a pair of referenced columns (i.e. AUTHOR_NAME, AUTHOR_DOB) and a pair of referencing columns (i.e. AUTHOR_NAME and DATE_OF_BIRTH). The columns named AUTHOR_NAME inside the data tables contain the same type of data. The data type of the DATE_OF_BIRTH column is the same as that of AUTHOR_DOB. As this example shows, the name of a referenced column doesn't need to match that of its referencing counterpart.

The MATCH Part

Now, let's discuss another part of the constraint's syntax:

```
{ MATCH [ SIMPLE | FULL | PARTIAL ] }
```

The curly brackets show that this clause is optional. The main function of this clause is to let you choose how to handle null values inside a foreign key column, considering the values that you may add to a referencing column. This clause won't work on columns that don't accept null values.

This part of the syntax offers three choices:

SIMPLE – If you choose this option, and at least one of your referencing columns has a null value, you may place any value on the rest of the referencing columns. The system will automatically trigger this option if you don't specify the MATCH

section of your FOREIGN KEY's definition.

FULL – This option requires all of your referencing columns to accept null values; otherwise, none of them can accept a null value.

PARTIAL – With this option, you may place null values on your referencing columns if other referencing columns contain values that match their respective referenced columns.

The (referential action) Part

This is the final section of the FOREIGN KEY syntax. Just like the MATCH part, “referential action” is completely optional. You can use this clause to specify which actions to take when updating or removing information from one or more referenced columns.

For example, let's assume that you want to remove an entry from the primary key of a table. If a foreign key references the primary key you're working on, your desired action will violate the constraint. So, you should always include the data of your referencing columns inside your referenced columns.

When using this clause, you will set a specific action to the referencing table's definition. This action will occur once your referenced table gets changed.

ON UPDATE (the referential action) { ON DELETE (the referential action) } |
ON DELETE (the referential action) { ON UPDATE (the referential action) }
(the referential action) ::=

RESTRICT | SET NULL | CASCADE | NO ACTION | SET DEFAULT

According to this syntax, you may set ON DELETE, ON UPDATE, or both. These clauses can accept one of the following actions:

RESTRICT – This referential action prevents you from performing updates or deletions that can violate the FOREIGN KEY constraint. The information inside a referencing column cannot violate FOREIGN KEY.

SET NULL – This action changes the values of a referencing column to "null" if its corresponding referenced column gets removed or updated.

CASCADE – With this referential action, the changes you'll apply on a referenced column will also be applied to its referencing column.

NO ACTION – Just like RESTRICT, NO ACTION stops you from performing

actions that will violate FOREIGN KEY. The main difference is that NO ACTION allows data violations while you are executing an SQL command. However, the information within your foreign key will not be violated once the command has been executed.

SET DEFAULT – With this option, you may set a referencing column to its default value by updating or deleting the data inside the corresponding referenced column. This referential action won't work if your referencing columns don't have default values.

To use this clause, you just have to insert it to the last part of a FOREIGN KEY's definition. Here's an example:

```
CREATE TABLE AUTHORS_GENRES
```

```
( AUTHOR_NAME    CHARACTER (50) ,
```

```
DATE_OF_BIRTH   DATE,
```

```
GENRE_ID        INT,
```

```
CONSTRAINT TS_BOOK_AUTHORS FOREIGN KEY ( AUTHOR_NAME,  
DATE_OF_BIRTH ) REFERENCES BOOK_AUTHORS ON DELETE  
RESTRICT ON UPDATE RESTRICT );
```

The CHECK Constraint

You can apply this constraint on a table, column, domain, or inside an assertion. This constraint lets you set which values to place inside your columns. You may use different conditions (e.g. value ranges) that define which values your columns may hold.

According to SQL programmers, the CHECK constraint is the most complex and flexible constraint currently available. However, this constraint has a simple syntax. To use CHECK as a column constraint, add the syntax below to your column definition:

```
(name of column) [ (domain) | (data type) ] CHECK < (the search condition) >
```

If you want to use this constraint on a table, insert the syntax below to your table's definition:

```
{ CONSTRAINT (name of constraint) } CHECK < (the search condition) >
```

Important Note: You'll later learn how to use this constraint on assertions and domains.

As this syntax shows, CHECK is easy to understand. However, its search condition may involve complex and extensive values. This constraint tests the assigned search condition for the SQL commands that try to alter the information inside a column protected by CHECK. If the result of the test is TRUE, the commands will run; if the result is false, the system will cancel the commands and display error messages.

You need to analyze examples in order to master this clause. However, almost all components of the search condition involve predicates. Predicates are expressions that work on values. In SQL, you may use a predicate to compare different values (e.g. COLUMN_3 < 5). The “less than” predicate checks whether the values inside COLUMN_3 are less than 5.

Most components of the search condition also utilize subqueries. Basically, subqueries are expressions that act as components of other expressions. You use a subquery if an expression needs to access or compute different layers of information. For instance, an expression might need to access TABLE_X to insert information to TABLE_Z.

In the example below, CHECK defines the highest and lowest values that you may

enter in a column. This table definition generates a CHECK constraint and three columns:

```
CREATE TABLE BOOK_TITLES
( BOOK_ID      INT,
  BOOK_TITLE    CHARACTER (50) NOT NULL,
  STOCK_AVAILABILITY  INT,
  CONSTRAINT TS_STOCK_AVAILABILITY ( STOCK_AVAILABILITY < 50
AND STOCK_AVAILABILITY > 1 ) );[21]
```

The resulting table will reject values that are outside the 1-50 range. Here's another way to write the table:

```
CREATE TABLE BOOK_TITLES
( BOOK_ID      INT,
  BOOK_TITLE    CHARACTER (50) NOT NULL,
  STOCK_AVAILABILITY      INT CHECK ( STOCK_AVAILABILITY < 50
AND STOCK_AVAILABILITY > 1 ) );
```

Now, let's analyze the condition clause of these statements. This clause tells SQL that all of the values added to the STOCK_AVAILABILITY column must be lower than 50. The keyword AND informs SQL that there's another condition that must be applied. Finally, the clause tells SQL that each value added to the said column should be higher than 1. To put it simply, each value should be lower than 50 and higher than 1.

This constraint also allows you to simply list your “acceptable values.” SQL users consider this a powerful option when it comes to values that won't be changed regularly. In the next example, you will use the CHECK constraint to define a book's genre:

```
CREATE TABLE BOOK_TITLES
( BOOK_ID      INT,
  BOOK_TITLE    CHARACTER (50) ,
  GENRE         CHAR (10) ,
```

```
CONSTRAINT TS_GENRE CHECK ( GENRE IN ( ' DRAMA ' , ' HORROR ' , '
SELF HELP ' , ' ACTION ' , ' MYSTERY ' , ' ROMANCE ' ) ) ) ;[22]
```

Each value inside the GENRE column should be included in the listed genres of the condition. As you can see, this statement uses IN (i.e. an SQL operator). Basically, IN makes sure that the values within GENRE are included in the listed entries.

This constraint can be extremely confusing since it involves a lot of parentheses. You may simplify your SQL codes by dividing them into multiple lines. As an example, let's rewrite the code given above:

```
CREATE TABLE BOOK_TITLES
(
BOOK_ID      INT,
BOOK_TITLE   CHAR (50) ,
GENRE       CHAR (10) ,
CONSTRAINT TS_GENRE CHECK
(
GENRE IN
( ' DRAMA ' , ' HORROR ' , ' SELF HELP ' , ' ACTION ' , ' MYSTERY ' , '
ROMANCE '
)
)
);
```

This style of writing SQL commands ensures code readability. Here, you need to indent the parentheses and their content so that they clearly show their position in the different layers of the SQL statement. By using this style, you can quickly identify the clauses placed in each pair of parentheses. Additionally, this statement works like the previous one. The only drawback of this style is that you need to use lots of spaces.

Let's analyze another example:

```
CREATE TABLE BOOK_TITLES
```

```
( BOOK_ID      INT,
```

```
  BOOK_TITLE   CHAR (50) ,
```

```
  STOCK_AVAILABILITY INT,
```

```
          CONSTRAINT TS_STOCK_AVAILABILITY CHECK ( ( STOCK_AVAILABILITY BETWEEN 1 AND 50 ) OR ( STOCK_AVAILABILITY BETWEEN 79 AND 90 ) ) );
```

This code uses BETWEEN (i.e. another SQL operator) to set a range that includes the lowest and highest points. Because it has two ranges, it separates the range specifications using parentheses. The OR keyword connects the range specifications. Basically, OR tells SQL that one of the conditions need to be satisfied. Consequently, the values you enter in the column named STOCK_AVAILABILITY should be from 1 through 50 or from 79 through 90.

How to Define an Assertion

Assertions are CHECK constraints that you can apply on multiple tables. Due to this, you can't create assertions while defining a table. Here's the syntax that you must use while creating an assertion:

```
CREATE ASSERTION (name of constraint) CHECK (the search conditions)
```

Defining an assertion is similar to defining a table-level CHECK constraint. After typing CHECK, you need to specify the search condition/s.

Let's analyze a new example. Assume that the BOOK_TITLES table has a column that holds the quantity of books in stock. The total for this table should always be lower than your desired inventory. This example uses an assertion to check whether the total of the STOCK_AVAILABILITY column is lower than 3000.

```
CREATE ASSERTION LIMIT_STOCK_AVAILABILITY CHECK ( ( SELECT SUM(STOCK_AVAILABILITY) FROM BOOK_TITLES ) < 3000 ) ;
```

This statement uses a subquery (i.e. “SELECT SUM (STOCK_AVAILABILITY) FROM BOOK_TITLES”) and compares it with 3000. The subquery starts with an SQL keyword, SELECT, which queries information from any table. The SQL function called SUM adds up all of the values inside STOCK_AVAILABILITY. The keyword FROM, on the other hand, sets the column that holds the table. The

system will then compare the subquery's result to 3000. You will get an error message if you add an entry to the STOCK_AVAILABILITY column that makes the total exceed 3000.

How to Create a Domain and a Domain Constraint

As mentioned earlier, you may also insert the CHECK constraint into your domain definitions. This kind of constraint is similar to the ones you've seen earlier. The only difference is that you don't attach a domain constraint to a particular table or column. Actually, a domain constraint uses VALUE, another SQL keyword, while referring to a value inside a column specified for that domain. Now, let's discuss the syntax you need to use to generate new domains:

```
CREATE DOMAIN (name of domain) {AS } (type of data)
```

```
{ DEFAULT (the default value) }
```

```
{ CONSTRAINT (name of constraint) } CHECK < (the search condition) >
```

This syntax has elements you've seen before, as you've seen default clauses and data types in the third chapter. The definition of the constraint, on the other hand, has some similarities with the ones discussed in the last couple of sections.

In the example below, you will generate an INT-type domain. This domain can only accept values between 1 and 50:

```
CREATE DOMAIN BOOK_QUANTITY AS INT CONSTRAINT  
TS_BOOK_QUANTITY CHECK (VALUE BETWEEN 1 and 50 );
```

This example involves one new item, which is the VALUE keyword. As mentioned earlier, this keyword refers to a column's specified value using the BOOK_QUANTITY domain. Consequently, you will get an error message if you will enter a value that doesn't satisfy the assigned condition (i.e. each value must be between 1 and 50).

CHAPTER 4: HOW TO CREATE AN SQL VIEW

“You can tell whether a man is clever by his answers... You can tell whether a man is wise by his questions.” – Naguib Mahfouz

Your database stores SQL information using “persistent” (i.e. permanent) tables. However, persistent tables can be impractical if you just want to check particular entries from one or more tables. Because of this, the SQL language allows you to use “views” (also called “viewed tables”).

Views are virtual tables whose definitions act as schema objects. The main difference between views and persistent tables is that the former doesn't store any data. Actually, viewed tables don't really exist – only their definition does. This definition lets you choose specific data from a table or a group of tables, according to the definition's query statements. To invoke a view, you just have to include its name in your query, as if it was an ordinary table.

How to Add a View to a Database

Views are extremely useful when you're trying to access various kinds of information. If you use a view, you may define complicated queries and save them inside a view definition. Rather than typing queries each time you use them, you may just call the view. In addition, views allow you to present data to other people without showing any unnecessary or confidential information.

For instance, you might need to allow some users to access certain parts of employee records. However, you don't want the said users to access the SSN (i.e. social security number) or pay rates of the listed employees. Here, you may generate views that show only the data needed by users.

How to Define an SQL View

In SQL, the most basic view that you can create is one which points to a single table and collects information from columns without changing anything. Here is the basic syntax of a view:

```
CREATE VIEW (name of view) { < (name of the view's columns) > }  
AS (the query)  
{ WITH CHECK OPTION }
```

Important Note: This part of the book focuses on the first and second lines of the format. You'll learn about the third line later.

You need to set the view's name in the first part of the definition. Additionally, you should name the view's columns if you are facing any of the following circumstances:

If you need to perform an operation to get the column's values, instead of just copying them from a table.

If you are working with duplicate column names. This situation happens when you combine tables.

You may set names for your columns even if you don't need to. For instance, you may assign logical names to your columns so that even an inexperienced user can understand them.

The second part of the format has a mandatory keyword (i.e. AS) and a placeholder for the query. Despite its apparent simplicity, the query placeholder may involve a complicated structure of SQL statements that perform different operations.

Let's analyze a basic example:

```
CREATE VIEW BOOKS_IN_STOCK  
( BOOK_TITLE, AUTHOR, STOCK_AVAILABILITY ) AS  
SELECT BOOK_TITLE, AUTHOR, STOCK_AVAILABILITY  
FROM BOOK_INVENTORY[23]
```

This sample is one of the simplest views that you can create. It gets three columns from a table. Remember that SQL isn't strict when it comes to line breaks and spaces. For instance, while creating a view, you may list the column names (if applicable) on a separate line. Database management systems won't care which coding technique you use. However, you can ensure the readability of your codes by adopting a coding style.

Now, let's dissect the sample code given above. The first part sets BOOKS_IN_STOCK as the view's name. The second part sets the name of the columns and contains the SQL keyword AS.

If you don't specify the names you want to use, the view's columns will just copy the names of the table's columns. The last two lines hold the search expression, which is a SELECT statement. Here it is:

```
SELECT BOOK_TITLE, AUTHOR, STOCK_AVAILABILITY  
FROM BOOK_INVENTORY
```

SELECT is flexible and extensive: it allows you to write complex queries that give you the exact kind of information you need.

The SELECT statement of this example is basic. It only has two clauses: SELECT and FROM. The first clause sets the column to be returned. The second clause, however, sets the table where the information will be pulled from. Once you call the BOOKS_IN_STOCKS view, you will actually call the embedded SELECT command of the view. This action gets the information from the correct table/s.

For the second example, let's create a view that has an extra clause:

```
CREATE VIEW BOOKS_IN_STOCK_80s  
( BOOK_TITLE, YEAR_PUBLISHED, STOCK_AVAILABILITY ) AS  
SELECT BOOK_TITLE, YEAR_PUBLISHED, STOCK_AVAILABILITY  
FROM BOOK_INVENTORY  
WHERE YEAR_PUBLISHED > 1979 AND YEAR_PUBLISHED < 1990;
```

The last clause sets a criterion that should be satisfied for the system to retrieve data. The only difference is that, rather than pulling the authors' information, it filters search results based on the year each book was published.

Important Note: The contents of the last clause don't affect the source table in any way. They work only on the information returned by the view.

You may use WHERE in your SELECT statements to set different types of criteria. For instance, you can use this clause to combine tables. Check the following code:

```
CREATE VIEW BOOK_PUBLISHERS  
(BOOK_TITLE, PUBLISHER_NAME) AS  
SELECT BOOK_INVENTORY.BOOK_TITLE, TAGS.PUBLISHER_NAME  
FROM BOOK_INVENTORY, TAGS  
WHERE BOOK_INVENTORY.TAG_ID = TAGS.TAG_ID;\[24\]
```

This code creates a view named BOOK_PUBLISHERS. The BOOK_PUBLISHERS view contains two columns: BOOK_TITLE and PUBLISHER_NAME. With this view, you'll get data from two different sources: (1) the BOOK_TITLE column of the BOOK_INVENTORY table and (2) the PUBLISHER_NAME column of the TABS table.

For now, let's focus on the third clause (i.e. the SELECT statement). This clause qualifies the columns based on the name of their respective tables (e.g. BOOK_INVENTORY.BOOK_TITLE). If you are joining tables, you need to specify the name of each table to avoid confusion. Obviously, columns can be highly confusing if they have duplicate names. However, if you're dealing with simple column names, you may omit the name of your tables. For instance, your SELECT clause might look like this:

```
SELECT BOOK_TITLE, PUBLISHER_NAME
```

Now, let's discuss the statement's FROM section. When combining tables, you need to name all of the tables you want to use and separate the entries using commas. Aside from the concern regarding duplicate names, this clause is identical to that of previous examples.

WHERE, the last clause of this statement, matches data rows together. This clause is important since, if you don't use it, you won't be able to match values you've gathered from different tables. In the current example, the values inside the TAG_ID column of BOOK_INVENTORY should match the values inside the TAG_ID column of the table named TAGS.

SQL allows you to qualify a query by expanding the latter's WHERE clause. In the next example, WHERE restricts the returned rows to those that hold “999” in the BOOK_INVENTORY table's TAG_ID column:

```
CREATE VIEW BOOK_PUBLISHERS  
(BOOK_TITLE, BOOK_PUBLISHER ) AS  
SELECT BOOK_INVENTORY .BOOK_TITLE, TAGS .BOOK_PUBLISHER  
FROM BOOK_INVENTORY, TAGS  
WHERE BOOK_INVENTORY .TAG_ID = TAGS .TAG_ID  
AND BOOK_INVENTORY .TAG_ID = 999;\[25\]
```

Let's work on another example. Similar to the examples you've seen earlier, this view collects information from a single table. This view, however, performs computations that return the modified information. Here is the statement:

```
CREATE VIEW BOOK_DISCOUNTS  
(BOOK_TITLE, ORIGINAL_PRICE, REDUCED_PRICE ) AS  
SELECT BOOK_TITLE, ORIGINAL_PRICE, REDUCED_PRICE * 0.8  
FROM BOOK_INVENTORY;
```

This statement creates a view that has three columns: BOOK_TITLE, ORIGINAL_PRICE, and REDUCED_PRICE. Here, SELECT indicates the columns that hold the needed information. The statement defines BOOK_TITLE and ORIGINAL_PRICE using the methods discussed in the previous examples.

The system will copy the data inside the BOOK_INVENTORY table's BOOK_TITLE and ORIGINAL_PRICE columns. Then, the system will paste the data to the columns of the same name inside the BOOK_DISCOUNTS view.

However, the last column is different,. Aside from taking values from its corresponding column, it multiplies the collected values by 0.8 (i.e. 80%). This way, the system will determine the correct values to display in the view's REDUCED_PRICE column.

SQL also allows you to insert the WHERE clause to your SELECT statements. Here's an example:

```
CREATE VIEW BOOK_DISCOUNTS
( BOOK_TITLE, ORIGINAL_PRICE, REDUCED_PRICE ) AS
SELECT BOOK_TITLE, ORIGINAL_PRICE, REDUCED PRICE * 0.8
FROM BOOK_INVENTORY
WHERE STOCK_AVAILABILITY > 20;[26]
```

This WHERE clause limits the search to entries whose STOCK_AVAILABILITY value is higher than 20. As this example shows, you may perform comparisons on columns that are included in the view.

How to Create an Updateable View

In the SQL language, some views allow you to perform updates. Simply put, you may use a view to alter the information (i.e. add new rows and/or alter existing information) inside the table you're working on. The “updateability” of a view depends on its SELECT statement. Usually, views that involve simple SELECT statements have higher chances of becoming updateable.

Remember that SQL doesn't have syntax to create updateable views. Rather, you need to write a SELECT statement that adheres to certain standards. This is the only way for you to create an updateable view.

The examples you've seen in this chapter imply that the SELECT statement serves as the search expression of a CREATE VIEW command. To be precise, query expressions may belong to different kinds of expressions. As an SQL user, most of the time, you'll be dealing with query specifications. Query expressions are SQL expressions that start with SELECT and contain different elements. To keep it simple, let's assume that SELECT is a query specification. Database products also use this assumption, so it is certainly effective.

You can't summarize, combine, or automatically delete the information inside the view.

The table you're working with should have at least one updateable column.

Every column inside the view should point to a single column in a table.

Every row inside the view should point to a single row in a table.

How to Drop a View

In some cases, you need to delete a view from a database. To do that, you need to use the following syntax:

```
DROP VIEW (name of the view);
```

The system will delete the view as soon as you run this statement. However, the process won't affect the underlying information (i.e. the data stored inside the actual tables). After dropping a view, you may recreate it or use its name to generate another view. Let's analyze a basic example:

```
DROP VIEW BOOK_PUBLISHERS;
```

This command will delete the `BOOK_PUBLISHERS` view from the database. However, the underlying information will be unaffected.

Database Security

Security is an important element of every database. You need to make sure that your database is safe from unauthorized users who may view or alter data. Meanwhile, you also need to ensure that authorized users can access and/or change data without any problems. The best solution for this problem is to provide each user with the privileges they need to do their job.

To protect databases, SQL has a security scheme that lets you specify which database users can view specific information from. This scheme also allows you to set what actions each user can perform. This security scheme (or model) relies on authorization identifiers. As you've learned in the second chapter, authorization identifiers are objects that represent one or more users that can access/modify the information inside the database.

The Security Model of SQL

The security of your database relies on authorization identifiers. You can use these identifiers to allow other people to access and/or alter your database entries. If an authorization identifier lacks the right privileges to alter a certain object, the user won't be able to change the information inside that object. Additionally, you may configure each identifier with various kinds of privileges.

In SQL, an authorization identifier can be a user identifier (i.e. “user”) or a role name (i.e. “role”). A “user” is a security profile that may represent a program, a person, or a service. SQL doesn't have specific rules regarding the creation of a user. You may tie the identifier to the OS (i.e. operating system) where the database system runs. Alternatively, you may create user identifiers inside the database system itself.

A role is a group of access rights that you may assign to users or other roles. If a certain role has access to an object, all users you've assigned that role to can access the said object.

SQL users often utilize roles to provide uniform sets of access rights to other authorization identifiers. One of the main benefits offered by a role is that it can exist without any user identifier. That means you can create a role before creating a user. In addition, a role will stay in the database even if you have deleted all of your user identifiers. This functionality allows you to implement a flexible process to administer access rights.

The SQL language has a special authorization identifier called PUBLIC. This identifier covers all of the database users. Similar to other identifiers, you may assign access rights to a PUBLIC profile.

Important Note: You need to be careful when assigning access rights to the PUBLIC identifier, as users might use that identifier for unauthorized purposes.

Creating and Deleting a Role

Generating new roles is a simple process. The syntax has two clauses: an optional clause and a mandatory clause.

```
CREATE ROLE (name of role)
```

```
{ WITH ADMIN [ CURRENT_ROLE | CURRENT_USER ] }
```

As you can see, `CREATE ROLE` is the only mandatory section of this statement. You don't need to set the statement's `WITH ADMIN` part. Actually, SQL users rarely set that clause. `WITH ADMIN` becomes important only if your current role name/user identifier pair doesn't have any null values.

Let's use the syntax to create a role:

```
CREATE ROLE READERS;
```

That's it. After creating this role, you will be able to grant it to users or other roles.

To drop (or delete) a role, you just have to use the following syntax:

```
DROP ROLE (name of role)
```

This syntax has a single requirement: the name of the role you want to delete.

Granting and Revoking a Privilege

Whenever you grant a privilege, you are actually linking a privilege to an authorization identifier. You will place this privilege/authorization identifier pair on an object, allowing the former to access the latter based on the defined privileges.

GRANT [(list of privileges) | ALL PRIVILEGES]

ON (type of object) (name of object)

TO [(list of authorization identifiers) | PUBLIC] { WITH GRANT OPTION }

{ GRANTED BY [CURRENT_ROLE | CURRENT_USER] }

This syntax has three mandatory clauses, namely: ON, TO and GRANT. The last two clauses, GRANTED BY and WITH GRANT OPTION, are completely optional.

The process of revoking privileges is simple and easy. You just have to use the syntax given below:

REVOKE { GRANT OPTION FOR } [(list of privileges) | ALL PRIVILEGES]

ON (type of object) name of object)

FROM [{ list of authorization identifiers) | PUBLIC }

CHAPTER 5: DATABASE CREATION

“To improve is to change; to be perfect is to change often.” – Sir Winston Churchill

Data is stored in tables and indexed to make queries more efficient. Before you can create tables, you need a database to hold the table. If you're starting from zero, you will have to learn to create and use a database.

Creating a Database

“We are what we pretend to be, so we must be careful about what we pretend to be.” – Kurt Vonnegut

To create a database, you will use the CREATE command with the name of the database.

The following is the syntax:

```
CREATE DATABASE database_name;
```

To demonstrate, assume you wanted to create a database and name it xyzcompany:

```
CREATE DATABASE xyzcompany;
```

With that statement, you have just created the xyzcompany database. Before you can use this database, you need to designate it as the active database. You have to run the USE command with the database name to activate your new database.

Here's the statement:

```
USE xyzcompany;
```

In following sessions, you can just type the statement 'USE xyzcompany' to access the database.

Removing a Database

If you need to remove an existing database, you can easily do so with this syntax:

`DROPDATABASE databasename;`

Therefore, you must exercise caution when using the `DROP` command to remove a database. You will also need admin privileges to drop a database.

Schema Creation

The CREATE SCHEMA statement is used to define a schema. On the same statement, you can also create objects and grant privileges on these objects.

The CREATE SCHEMA command can be embedded within an application program. Likewise, it can be issued using dynamic SQL statements. For example, if you have database admin privileges, you can issue this statement which creates a schema called USER1 with the USER1 as its owner:

```
CREATE SCHEMA USER1 AUTHORIZATION USER1
```

The following statement creates a schema with an inventory table. It also grants authority on the inventory table to USER2:

```
CREATE SCHEMA INVENTORY
```

```
    CREATE TABLE ITEMS (IDNO INT(6) NOT NULL,  
                        SNAME VARCHAR(40),  
                        CLASS INTEGER)
```

```
    GRANT ALL ON ITEMS TO USER2
```

MySQL 5.7

If you're using MySQL 5.7, CREATE SCHEMA is synonymous to the CREATE DATABASE command.

Here's the syntax:^[27]

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name  
[create_specification]...
```

create_specification:

```
[DEFAULT] CHARACTER SET [=] charset_name | [DEFAULT] COLLATE  
[=] collation_name
```

Oracle 11g

In Oracle 11g, you can create several views and tables and perform several grants in one transaction within the CREATE SCHEMA statement. To successfully execute the CREATE SCHEMA command, Oracle runs each statement within the

block and commits the transaction if no errors are encountered. If a statement returns an error, all statements are rolled back.

The statements CREATE VIEW, CREATE TABLE, and GRANT may be included within the CREATE SCHEMA statement. Hence, you must not only have the privilege to create a schema, you must also have the privileges needed to issue the statements within it.

The syntax is the following:[\[28\]](#)

```
{ create_table_statement
| create_view_statement
| grant_statement
} ...;
```

SQL Server 2014

In SQL Server 2014, the CREATE SCHEMA statement is used to create a schema in the current database. This transaction may also create views and tables within the newly-created schema and set GRANT, REVOKE, or DENY permission on these objects.

This statement creates a schema and sets the specifications for each argument:[\[29\]](#)

```
CREATE SCHEM A schem a_name_clause [ <schem a_element> [ ...n ] ]

<schem a_name_clause> ::=
{
schem a_name
| AUTHORIZATION owner_name
| scheme a_name AUTHORIZATION owner_name
}

<schem a_element> ::=
{
table_definition | view_definition | grant_statement |
```

```
revoke_statement | deny_statement
```

```
}
```

PostgreSQL 9.3.13

The CREATE SCHEMA statement is used to enter a new schema into a database. The schema name should be unique within the current database. [\[30\]](#)

Here's the syntax:

```
CREATE SCHEMA schema_name [AUTHORIZATION user_name][schema_element[...]]
```

```
CREATE SCHEMA AUTHORIZATION user_name [schema_element[...]]
```

```
CREATE SCHEMA IF NOT EXISTS schema_name [AUTHORIZATION user_name]
```

```
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

Creating Tables and Inserting Data Into Tables

Tables are the main storage of information in databases. Creating a table means specifying a name for a table, defining its columns, as well as the data type of each column.

How to Create a Table

The keyword `CREATE TABLE` is used to create a new table. It is followed by a unique identifier and a list that defines each column and the type of data it will hold.

```
CREATE TABLE table_name  
(  
  column1 datatype [NULL | NOT NULL].  
  column2 datatype [NULL | NOT NULL].  
  ...  
):
```

This is the basic syntax to create a table:

Parameters

`table_name`: This is the identifier for the table.

`column1`, `column2`: These are the columns that you want the table to have. All columns should have a data type. A column is defined as either `NULL` or `NOT NULL`. If this is not specified, the database will assume that it is `NULL`.

The following set of questions can serve as a guide when creating a new table:

- What is the most appropriate name for this table?
- What data types will I be working with?
- What is the most appropriate name for each column?
- Which column(s) should be used as the main key(s)?
- What type of data can be assigned to each column?
- What is the maximum width for each column?
- Which columns can be empty and which columns should not be empty?

The following example creates a new table with the `xyzcompany` database. It will be named `EMPLOYEES`:

```
CREATE TABLE EMPLOYEES(  
  ID INT(6) auto_increment, NOT NULL,
```

```
FIRST_NAME VARCHAR(35) NOT NULL,  
LAST_NAME VARCHAR(35) NOT NULL,  
POSITION VARCHAR(35),  
SALARY DECIMAL(9,2).  
ADDRESS VARCHAR(50),  
PRIMARY KEY (id)  
);
```

The code creates a table with 6 columns. The ID field was specified as its primary key. The first column is an INT data type with a precision of 6. It does not accept a NULL value. The second column, FIRST_NAME, is a VARCHAR type with a maximum range of 35 characters. The third column, LAST_NAME, is another VARCHAR type with a maximum of 35 characters. The fourth column, POSITION, is a VARCHAR type which is set at a maximum of 35 characters. The fifth column, SALARY, is a DECIMAL type with a precision of 9 and scale of 2. Finally, the fifth column, ADDRESS, is a VARCHAR type with a maximum of 50 characters. The id column was designated as the primary key.

Creating a New Table Based on Existing Tables

You can create a new table based on an existing table by using the CREATE TABLE keyword with SELECT.

Here's the syntax:[\[31\]](#)

```
CREATE TABLE new_table_name AS
(
SELECT [column n1.column, column n, 2'''column nN]
FROM existing_table_name
[WHERE]
);
```

Executing this code will create a new table with column definitions that are identical to the original table. You may copy all columns or select specific columns for the new table. The new table will be populated by the values of the original table.

To demonstrate, create a duplicate table named STOCKHOLDERS from the existing EMPLOYEES table within the xyzcompany. Here's the code:

```
CREATE TABLE STOCKHOLDERS AS
SELECT ID, FIRST_NAME, LAST_NAME, POSITION, SALARY, ADDRESS
FROM EMPLOYEES;
```

Inserting Data into Table

SQL's Data Manipulation Language (DML) is used to perform changes to databases. You can use DML clauses to fill a table with fresh data and update an existing table.

Populating a Table with New Data

There are two ways to fill a table with new information: manual entry or automated entry through a computer program.

Populating data manually involves data entry using a keyboard, while automated entry involves loading data from an external source. It may also include transferring data from one database to a target database.

Unlike SQL keywords or clauses that are case-insensitive, data is case-sensitive. Hence, you have to ensure consistency when using or referring to data. For instance, if you store an employee's first name as 'Martin', you should always refer to it in the future as 'Martin' and never 'MARTIN' or 'martin'.

The INSERT Keyword

The INSERT keyword is used to add records to a table. It inserts new rows of data to an existing table.

There are two ways to add data with the INSERT keyword. In the first format, you simply provide the values for each field and they will be assigned sequentially to the table's columns. This form is generally used if you need to add data to all columns.

You will use the following syntax for the first form:

```
INSERT INTO table_name  
VALUES ('value1', 'value2', [NULL]);
```

In the second form, you'll have to include the column names. The values will be assigned based on the order of the columns' appearance. This form is typically used if you want to add records to specific columns.

Here's the syntax:

```
INSERT INTO table_name (column n1, column n2, column n3)  
VALUES ('value1', 'value2', 'value3');
```

Notice that in both forms, a comma is used to separate the columns and the values. In addition, you have to enclose character/string and date/time data within quotation marks.

Assuming that you have the following record for an employee:

First NameRobert

Last NamePage

PositionClerk

Salary5,000.00

282 Patterson Avenue Illinois

To insert this data into the EMPLOYEES table, you can use the following statement:

```
INSERT INTO EMPLOYEES (FIRST_NAME, LAST_NAME, POSITION,  
SALARY, ADDRESS)
```

```
VALUES ('Robert', 'Page', 'Clerk', 5000.00, '282 Patterson Avenue, Illinois');
```

To view the updated table, here's the syntax:

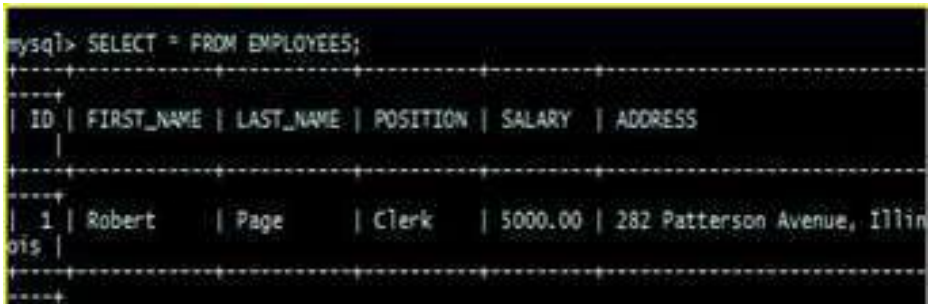
```
SELECT * FROM table_name;
```

To display the data stored in the EMPLOYEES' table, use the following statement:

SELECT * FROM EMPLOYEES:

The wildcard (*) character tells the database system to select all fields on the table.

Here's a screenshot of the result:[\[32\]](#)



```
mysql> SELECT * FROM EMPLOYEES;
```

ID	FIRST_NAME	LAST_NAME	POSITION	SALARY	ADDRESS
1	Robert	Page	Clerk	5000.00	282 Patterson Avenue, Illinois

Now, try to encode the following data for another set of employees:

First Name	Last Name	Position	Salary	Address
John	Malley	Supervisor	7,000.00	5 Lake View, New York
Kristen	Johnston	Clerk	4,500.00	25 Jump Road, Florida
Jack	Burns	Agent	5,000.00	5 Green Meadows, California

You will have to repeatedly use the INSERT INTO keyword to enter each employee data to the database. Here's how the statements would look:

```
INSERT INTO EMPLOYEES(FIRST_NAME, LAST_NAME, POSITION, SALARY, ADDRESS)
```

```
VALUES('John', 'Malley', 'Supervisor', 7000.00, '5 Lake View New York');
```

```
INSERT INTO EMPLOYEES(FIRST_NAME, LAST_NAME, POSITION, SALARY, ADDRESS)
```

```
VALUES('Kristen', 'Johnston', 'Clerk', 4000.00, '25 Jump Road, Florida');
```

```
INSERT INTO EMPLOYEES(FIRST_NAME, LAST_NAME, POSITION, SALARY, ADDRESS)
```

```
VALUES('Jack', 'Burns', 'Agent', 5000.00, '5 Green Meadows, California');
```

To fetch the updated EMPLOYEES table, use the SELECT command with the wild card character.

SELECT * FROM EMPLOYEES:

Here's a screenshot of the result:

```
mysql> SELECT * FROM EMPLOYEES;
```

ID	FIRST_NAME	LAST_NAME	POSITION	SALARY	ADDRESS
1	Robert	Page	Clerk	5000.00	282 Patterson Avenue, Illinois
2	John	Malley	Supervisor	7000.00	5 Lake View New York
3	Kristen	Johnston	Clerk	4000.00	25 Jump Road Florida
4	Jack	Burns	Agent	5000.00	5 Green Meadows California

```
4 rows in set (0.00 sec)
```

Notice that SQL assigned an ID number for each set of data you entered. This is because you have defined the ID column with the `auto_increment` attribute. This property will prevent you from using the first form when inserting data. So, you have to specify the rest of the columns in the `INSERT INTO table_name` line. [\[33\]](#)

Inserting Data into Specific Columns

You may also insert data into specific column(s). You can do this by specifying the column name inside the column list and the corresponding values inside the VALUES list of the INSERT INTO statement. For example, if you just want to enter an employee’s full name and position, you will need to specify the column names FIRST_NAME, LAST_NAME, and SALARY in the columns list and the values for the first name, last name, and salary inside the VALUES list.

To see how this works, try entering the following data into the EMPLOYEES table:

First Name	Last Name	Position	Salary	Address
James	Hunt		7,500.00	

Here’s a screenshot of the updated EMPLOYEES table:[\[34\]](#)

```
SQL> SELECT * FROM EMPLOYEES;
+-----+-----+-----+-----+-----+-----+
| ID | FIRST_NAME | LAST_NAME | POSITION | SALARY | ADDRESS |
+-----+-----+-----+-----+-----+-----+
| 1 | Robert | Page | Clerk | 5000.00 | 282 Patterson Avenue, Illinois |
| 2 | John | Malley | Supervisor | 7000.00 | 5 Lake View New York |
| 3 | Kristen | Johnston | Clerk | 4000.00 | 25 Jump Road Florida |
| 4 | Jack | Burns | Agent | 5000.00 | 5 Green Meadows California |
| 5 | James | Hunt | NULL | 7500.00 | NULL |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Inserting NULL Values

In some instances, you may have to enter NULL values into a column. For example, you may not have data on hand to enter a new employee's salary. It may be misleading to provide just about any salary figure.

Here's the syntax:

```
INSERT INTO schem a.table_name  
VALUES ('column1', NULL, 'column3');
```

You'll need this answer sheet to be able to check your syntax in each exercise to ensure that it is correct. You are more than welcome to use it if you're stuck on an exercise as well. At the end of each exercise, I encourage you to check your answers.

Each exercise gives an overview, as well as examples, before you start applying what you learned in each section.

Below is a sample of the Product table that you will be using in the next few exercises.

Product ID	Name	Product Number	Color	Standard Cost	List Price	Size
317	LL Crankarm	CA-5965	Black	0	0	NULL
318	ML Crankarm	CA-6738	Black	0	0	NULL
319	HL Crankarm	CA-7457	Black	0	0	NULL
320	Chainring Bolts	CB-2903	Silver	0	0	NULL
321	Chainring Nut	CN-6137	Silver	0	0	NULL

Query Structure and SELECT Statement

Understanding the syntax and its structure will be extremely helpful for you in the future. Let's delve into the basics of one of the most common statements, the SELECT statement, which is used solely to retrieve data.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

In the above query, you're selecting two columns and all rows from the entire table. Since you're selecting two columns, the query will perform much faster than if you had selected all of the columns like this:

```
SELECT *
```

```
FROM Table_Name
```

Select all of the columns from the Production.Product table.

Using the same table, select only the ProductID, Name, Product Number, Color and Safety Stock Level columns.

(Hint: The column names do not contain spaces in the table!)

The WHERE Clause

The WHERE clause is used to filter the amount of rows returned by the query. This clause works by using a condition, such as if a column is equal to, greater than, less than, a certain value.

When writing your syntax, it's important to remember that the WHERE condition comes after the FROM statement. The types of operators that can be used vary based on the type of data that you're filtering.

EQUALS – This is used to find an exact match. The syntax below uses the WHERE clause for a column that contains the exact string of 'Value'. Note that strings need single quotes around them, but numerical values do not.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

```
WHERE Column_Name3 = 'Value'
```

BETWEEN – Typically used to find values between a certain range of numbers or date ranges. It's important to note that the first value in the BETWEEN comparison operator should be lower than the value on the right. Note the example below comparing between 10 and 100.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 BETWEEN 100 AND 1000
```

GREATER THAN, LESS THAN, LESS THAN OR EQUAL TO, GREATER THAN OR EQUAL TO – SQL Server has comparison operators that you can use to compare certain values.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 <= 1000 --Note that you can also use <, >, >= or even  
<> for not equal to
```

LIKE – This searches for a value or string that is contained within the column. You would still use single quotes, but include the percent symbols indicating if the string is in the beginning, at the end or anywhere between ranges of strings.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 LIKE '%Value%' --Searches for the word 'value' in any  
field
```

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 LIKE 'Value%' --Searches for the word 'value' at the  
beginning of the field
```

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name \[35\]  
WHERE Column_Name3 LIKE '%Value' --Searches for the word 'value' at the  
end of the field
```

IS NULL and IS NOT NULL – As previously discussed, NULL is an empty cell that contains no data. Eventually, you'll work with a table that does contain NULL values, which you can handle in a few ways.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

WHERE Column_Name3 IS NULL --Filters any value that is NULL in that column, but don't put NULL in single quotes since it's not considered a string.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

WHERE Column_Name3 IS NOT NULL --Filters any value that is not NULL in that column.

Using the Production.Product table again, select the Product ID, Name, Product Number and Color. Filter the products that are silver colored.

Using ORDER BY

The ORDER BY clause is simply used to sort data in ascending or descending order, and is specified by which column you want to sort. This command also sorts in ascending order by default, so if you want to sort in descending order, use DESC in your command.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

```
WHERE Column_Name3 = 'Value'
```

ORDER BY Column_Name1, Column_Name2 --Sorts in ascending order, but you can also include ASC to sort in ascending order.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

```
WHERE Column_Name3 = 'Value'
```

ORDER BY Column_Name1, Column_Name2 DESC --This sorts the data in descending order by specifying DESC after the column list.

Select the Product ID, Name, Product Number and List Price from the Production.Product table where the List Price is between \$100.00 and \$400.00. Then, sort the results by the list price from smallest to largest.

(Hint: You won't use the \$ in your query and also, you may refer back to the

sorting options you just reviewed if you need to.)[\[36\]](#)

Data Definition Language (DDL)

“There is nothing worse than aggressive stupidity.” – Johann Wolfgang von Goethe

DDL is the SQL syntax that is used to create, alter or remove objects within the instance or database itself. Below are some examples to help you get started. DDL is split up into three types of commands:

CREATE - This will create objects within the database or instance, such as other databases, tables, views, etc.

--Creates a Database

CREATE DATABASE DatabaseName

--Creates a schema (or container) for tables in the current database

CREATE SCHEMA SchemaName

--Creates a table within the specified schema

CREATE TABLE SchemaName.TableName

(
Column1 datatype PRIMARY KEY,
Column2 datatype(n),
Column3 datatype
)[\[37\]](#)

--Creates a View

CREATE VIEW ViewName

AS

SELECT

Column1,

Column2

FROM TableName

ALTER - This command will allow you to alter existing objects, like adding an additional column to a table or changing the name of the database, for example.

--Alters the name of the database

ALTER DATABASE DatabaseName MODIFY NAME = NewDatabaseName

--Alters a table by adding a column

ALTER TABLE TableName

ADD ColumnName datatype(n)^[38]

DROP - This command will allow you to drop objects within the database or the database itself. This can be used to drop tables, triggers, views, stored procedures, etc. Please note that these items will not exist within your database anymore – or the database will cease to exist if you drop it.

--Drops a Database - use the master db first

USE master

GO

--Then drop the desired database

DROP DATABASE DatabaseName

GO

--Drops a table; should be performed in the database where the specified table exists

DROP TABLE Table_Name

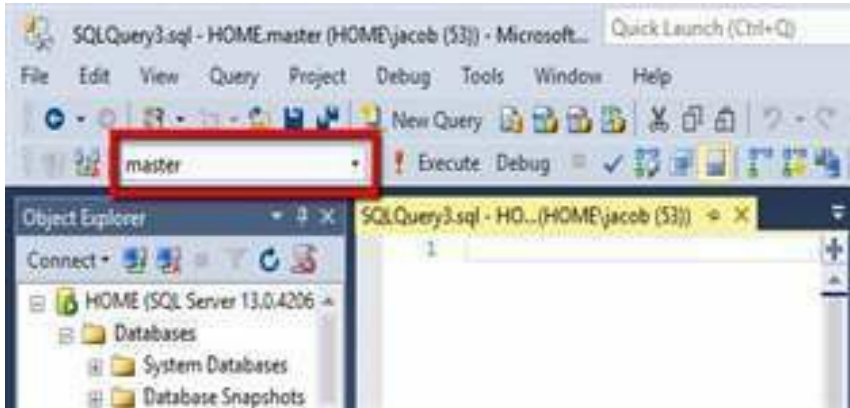
--Drops a view from the current database

DROP VIEW ViewName

Applying DDL Statements

Open a new query window on your own machine by clicking the “New Query” button and ensure that you’re using the ‘master’ database.

You can tell which database you’re using and switch between databases by checking below and using the drop-down menu.



Back in the [Guidelines for Writing T-SQL Syntax](#) section, you learned about the ‘USE’ statement and splitting up batches of code with ‘GO’.

You’ll be using this in the next exercise. So, feel free to refer back to that section before or during the exercise.

Create a database called Company. Don’t forget to use the GO statement between using the master database and creating your new database!

After you’ve performed this, you’ll now have a database that you can begin adding tables to. Below are a series of exercises in which you’ll begin creating your first schema and table within this database.

Create a schema called Dept. The table that you’ll be creating in the next exercise will be associated to this schema.

Create a table called Department. The table should have two columns: Department_Id, which should be an integer data type and the primary key. The other column should be Department_Name, which should be a data type of VARCHAR of 30 bytes.

Alter the name of your database from Company to Company_Db.

Running the DDL Script

Now, it's time for you to download the following DDL script to create the rest of the tables that you'll be working with. Don't worry, I've written the script so that it will successfully run by clicking the 'Execute' button. Just make sure that you have completed the exercises correctly, otherwise it will not work.

Once you've performed this, you'll start using your knowledge of DML to add and retrieve data.

Data Manipulation Language (DML)

DML or Data Manipulation Language is exactly what it sounds like; it manipulates the data. The statements that are part of DML allow you to modify existing data, create new data, or even delete data.

Take note that DML is not specific to objects like modifying a table's structure or a database's settings, but it works with the data within such objects.

SELECT - This statement is the most common SQL statement, and it allows you to retrieve data. It is also used in many reporting scenarios.

--Selects two columns from a table

SELECT

Column_Name1,

Column_Name2

FROM Table_Name

INSERT - This statement allows you to insert data into the tables of a particular database.

--Inserts three rows into the table

INSERT INTO Table_Name

(Column_Name1, Column_Name2) --These two items are the columns

VALUES

('Value1', 1), --Here, I'm inserting Value1 into Column_Name1 and the number 1 in Column_Name2.

('Value2', 2),

('Value3', 3)

Now, it is time to insert data into your Department table. Insert the values as

shown below. For example, the Department Name of Marketing must have a Department ID of 1, and so on.

Department_Id: 1, Department_Name: Marketing

Department_Id: 2, Department_Name: Development

Department_Id: 3, Department_Name: Sales

Department_Id: 4, Department_Name: Human Resources

Department_Id: 5, Department_Name: Customer Support

Department_Id: 6, Department_Name: Project Management

Department_Id: 7, Department_Name: Information Technology

Department_Id: 8, Department_Name: Finance/Payroll

Department_Id: 9, Department_Name: Research and Development

UPDATE - Allows you to update existing data. For instance, you may change an existing piece of data in a table to another value.

It's recommended to use caution when updating values in a table. You could give all rows within one column the same value if you don't specify a condition in the WHERE clause.

--Updating a value or set of values in one column of a table

UPDATE Table_Name

SET

Column_Name1 = 'New Value 1', --Specify what your new value should be here

Column_Name2 = 'New Value 2'

WHERE

Column_Name1 = 'Old Value' --Using the WHERE clause as a condition

--Updates all rows in the table with the same value for one specific column

UPDATE Table_Name

SET Column_Name1 = 'Value'[\[39\]](#)

Now, you will update one of the department names in your table. Let's say that you feel like the name 'Finance/Payroll' won't work because payroll is handled by employees via an online web portal. So, that leaves finance, but it's best to call this department 'Accounting' instead, due to the responsibilities of those in the department.

Update the Department table and change the value from 'Finance/Payroll' to 'Accounting' based on its ID.

DELETE - This action is self-explanatory; it deletes data from a specified object, like a table. It's best to use caution when running the DELETE statement.

If you don't use the WHERE clause in your query, you'll end up deleting all of the data within the table. So, it's always best to use a WHERE clause in a DELETE statement.

*Note: Much like the UPDATE statement, you can also swap out DELETE with SELECT in your statement to see what data you will be deleting, as long as you're using the WHERE clause in your query. Like the UPDATE statement, when you delete one value, it's best to use the primary key as a condition in your WHERE clause!

--Uses a condition in the DELETE statement to only delete a certain value or set of values

```
DELETE FROM Table_Name
```

```
WHERE Column_Name1 = 'Some Value'
```

--Deletes all of the data within a table since the WHERE clause isn't used

```
DELETE FROM Table_Name
```

Let's say that this company doesn't actually have a Research and Development (R&D) department. Perhaps they haven't grown enough yet or don't require that department.

For this exercise, delete the 'Research and Development' department from the Department table based on its ID.

Running the DML Script

Now, it's time for you to download the following DML script in order to populate the Company_Db database with data. Again, make sure that you have completed the exercises correctly, otherwise it will not work.

This data will be used in the following exercises in the 'Transforming Data' section. So, please keep in mind that each exercise refers to the Company_Db database, unless specified otherwise.

CHAPTER 6: DATABASE ADMINISTRATION

“Laws too gentle are seldom obeyed; too severe, seldom executed.” – Benjamin Franklin

In order to start on the path of a Database Administrator, Database Developer or even a Data Analyst, you’ll need to know how to back up, restore and administer databases. These are essential components to maintaining a database and are definitely important responsibilities.

Recovery Models

There are several different recovery models that can be used for each database in the SQL Server. A recovery model is an option or model for the database that determines the type of information that can be backed up and restored.

Depending on your situation, like if you cannot afford to lose critical data or you want to mirror a Production environment, you can set the recovery model to what you need. Also, keep in mind that the recovery model that you choose could also affect the time it takes to back up or restore the database and logs, depending on their size.

Below are the recovery models and a brief explanation of each. The SQL statement example at the end of each is what you would use in order to set the recovery model.

Full

This model covers all of the data within the database, as well as the transaction log history.

When using this model and performing a restore of the database, it offers “point in time” recovery. This means that if there was a point where data was lost from the database during a restore for example, it allows you to roll back and recover that data.

This is the desired recovery model if you cannot afford to lose any data.

It may take more time to back up and restore, depending on the size of the data.

- Can perform full, differential and transaction log backups.

ALTER DATABASE DatabaseName SET RECOVERY FULL

Simple

This model covers all of the data within the database too, but recycles the transaction log. When the database is restored, the transaction log will be empty and will log new activity moving forward.

Unlike the “Full” recovery model, the transaction log will be reused for new transactions and therefore cannot be rolled back to recover data if it has accidentally been lost or deleted.

It’s a great option if you have a blank server and need to restore a database fairly quickly and easily, or to just mirror another environment and use it as a test instance.

- Can perform full and differential backups only, since the transaction log is recycled per this recovery model.

ALTER DATABASE DatabaseName SET RECOVERY SIMPLE

Bulk Logged

This particular model works by forgoing the bulk operations in SQL Server, like BULK INSERT, SELECT INTO, etc. and does not store these in the transaction logs. This will help free up your transaction logs and make it easier and quicker during the backup and restore process.

Using this method, you have the ability to use “point in time” recovery, as it works much like the “Full” recovery model.

If your data is critical to you and your database processes often use bulk statements, this recovery model is ideal. However, if your database does not often use bulk statements, the “Full” recovery model is recommended.

- Can perform full, differential and transaction log backups.

ALTER DATABASE DatabaseName SET RECOVERY BULK_LOGGED

Let’s say you’re in the middle of a large database project and need to back up the database. Use SQL syntax (and the examples above) to set the recovery model for Company_Db to “Full” to prepare for the backup process.

Database Backup Methods

There are a few main backup methods that can be used to back up databases in SQL Server, and each one is dependent on the recovery model being used. In the previous section regarding recovery models, the last bullet point in each model discussed the types of backups that can be performed, i.e. full, differential and transaction log.

***Note :** if you don't remember the types of backups that can be performed for each recovery model, just give yourself time! Eventually, you will remember them!*

Each backup method will perform the following action:

Full:

Backs up the database in its entirety, as well as the transaction log (if applicable.)

Ideal if the same database needs to be added to a new server for testing.

It may take a longer period of time to back up if the database is large in size, in addition to it backing up the entire database and transaction log.

- However, this doesn't need to be performed nearly as often as a differential since you're backing up the entire database.

```
BACKUP      DATABASE      DatabaseName      TO      DISK      =  
'C:\SQLBackups\DatabaseName.BAK'
```

Differential:

It is dependent upon a full back up, so the full backup must be performed first. However, this will back up any data changes between the last full back up and the time that the differential backup takes place.

It is ideal in any situation, but must be done more frequently since there is new data being added to the database frequently.

It is much faster to back up than a full backup and takes up less storage space.

- However, this doesn't capture the database entirely, so this is the reason it must be performed periodically.

```
BACKUP DATABASE DatabaseName TO DISK =  
'C:\SQLBackups\DatabaseName.BAK' WITH DIFFERENTIAL
```

Transaction Log:

Like a differential backup, it's dependent upon a full backup.

Backs up all of the activity that has happened in the database.

Useful and necessary in some cases, like when restoring a database in full.

Since they hold the activity within the database's history, it's wise to back this up frequently so that the logs do not grow large in size.

Some logs are initially larger in size, but can become smaller as more frequent backups are done (depends on the activity within the database).

- Therefore, the log could be quick to back up and restore, if the database has minimal activity and is backed up frequently.

```
BACKUP LOG DatabaseName TO DISK =  
'C:\SQLBackups\DatabaseName.TRN'
```

In the last exercise, you set the recovery model of the Company_Db to "Full". Now for this exercise, you'll be backing up the database.

First, back up the database in full using the full backup method.

Next, delete all of the data from the Sales.Product_Sales table (intended to be a simulation of losing data.)

Finally, query the table that you just deleted data from to ensure that the data doesn't exist. Later on, you'll be restoring the database and ensuring that the data is the Sales.Product_Sales table.

Database Restores

In the database world, you will probably hear that the most important part of restoring a database is using a solid backup. That's entirely true! This is why it's important to ensure that you're using the proper recovery model.

Preparing to Restore the Database

An important thing to note is that the .bak file typically contains file groups or files within them. Each file could either be a full or differential backup. It's important to know which one you're restoring, but you can also specify which file(s) you'd like to restore. However, if you don't specify, SQL Server will pick the default file, which is 1. This is a typical full backup.

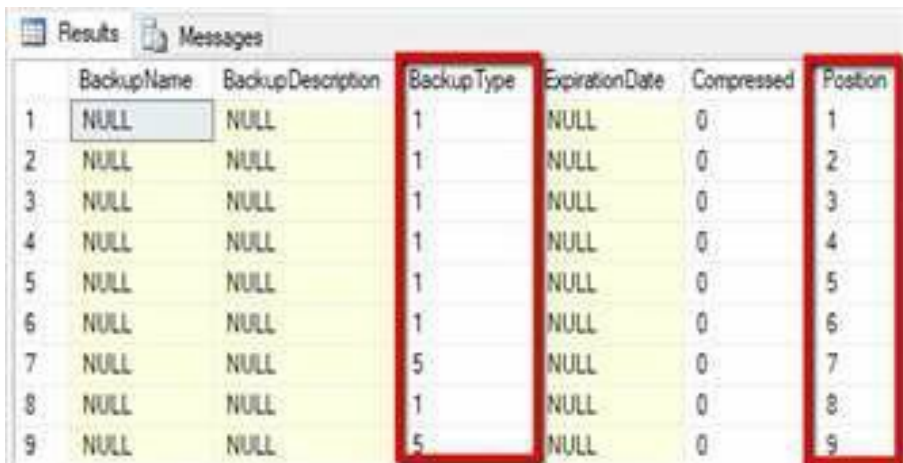
First, use RESTORE HEADERONLY to identify the files within your .BAK file. This is something you should do prior to restoring a database.

--View database backup files

```
RESTORE          HEADERONLY          FROM          DISK          =  
'C:\SQLBackups\DatabaseName.BAK'
```

Once you run the above query, look for the number in the “BackupType” column. The 1 indicates a full backup, whereas a 5 indicates a differential backup.

Also, the file number that you need to use is in the “Position” column. In the example below, note that the file number 8 is a full backup and file number 9 is a differential backup.



	BackupName	BackupDescription	BackupType	ExpirationDate	Compressed	Position
1	NULL	NULL	1	NULL	0	1
2	NULL	NULL	1	NULL	0	2
3	NULL	NULL	1	NULL	0	3
4	NULL	NULL	1	NULL	0	4
5	NULL	NULL	1	NULL	0	5
6	NULL	NULL	1	NULL	0	6
7	NULL	NULL	5	NULL	0	7
8	NULL	NULL	1	NULL	0	8
9	NULL	NULL	5	NULL	0	9

Additionally, you must first use the ‘master’ database and then set the desired database to a non-active state (SINGLE_USER) to only allow one open connection to the database prior to restoring.

In the below syntax, the WITH ROLLBACK IMMEDIATE means that any

incomplete transactions will be rolled back (not completed) in order to set the database to a single, open connection state.

--Sets the database to allow only one open connection

--All other connections to the database must be closed, otherwise the restore fails

```
ALTER DATABASE DatabaseName SET SINGLE_USER WITH ROLLBACK IMMEDIATE
```

Once the restore has completed, you can set the database back to an active state (MULTI_USER), as shown below. This state allows multiple users to connect to the database, rather than just one.

--Sets the database to allow multiple connections

```
ALTER DATABASE DatabaseName SET MULTI_USER
```

Database Restore Types

Below is an overview of the types of restores that you can perform depending on your situation/needs.

Full Restore

Restores the entire database, including its files.

Overwrites the database if it already exists using the WITH REPLACE option.

If the recovery model is set to “Full”, you need to use the WITH REPLACE option.

Use the FILE = parameter to choose which file you’d like to restore (can be full or differential).

If the recovery model is set to “Simple”, you don’t need the WITH REPLACE option.

If the database does not exist, it will create the database, including its files and data.

Restores from a .BAK file.

--If the recovery model is set to Full - also choosing the file # 1 in the backup set

```
RESTORE     DATABASE     DatabaseName     FROM     DISK     =  
'C:\SQLBackups\DatabaseName.BAK' WITH FILE = 1, REPLACE
```

--If recovery model is set to Simple

```
RESTORE     DATABASE     DatabaseName     FROM     DISK     =  
'C:\SQLBackups\DatabaseName.BAK'
```

Differential Restore

Use RESTORE HEADERONLY in order to see the backup files you have available, i.e. the full and differential backup file types.

You must perform a full restore of the .BAK file first with the NORECOVERY OPTION, as the NORECOVERY option indicates other files need to be restored as well.

Once you've specified the full backup file to be restored and use WITH NORECOVERY, you can restore the differential.

Finally, you should include the RECOVERY option in your last differential file restore in order to indicate that there are no further files to be restored.

As a reference, the syntax below uses the files from the previous screenshot of RESTORE HEADERONLY.

--Performs a restore of the full database backup file first

```
RESTORE     DATABASE     DatabaseName     FROM     DISK     =  
'C:\SQLBackups\DatabaseName.BAK' WITH FILE = 8, NORECOVERY
```

--Now performs a restore of the differential backup file

```
RESTORE     DATABASE     DatabaseName     FROM     DISK     =  
'C:\SQLBackups\DatabaseName.BAK' WITH FILE = 9, RECOVERY
```

Log Restore

This is the last step to perform a thorough database restore.

You must restore a “Full” or “Differential” copy of your database first, then restore the log.

- Finally, you must use the NORECOVERY option when restoring your database backup(s) so that your database stays in a restoring state and allows you to restore the transaction log.

Since you already performed a backup of your database and “lost” data in the Sales.Product_Sales table, it’s now time to restore the database in full.

In order to do this, you’ll first need to use the ‘master’ database, then set your database to a single user state, perform the restore, and finally set it back to a multi-user state.

Once you’ve restored the database, retrieve all data from the Sales.Product_Sales table to verify that the data exists again.

Attaching and Detaching Databases

As you know, because you already went through this portion, the methodology of attaching and detaching databases is similar to backups and restores.

Essentially, here are the details of this method:

Allows you to copy the .MDF file and .LDF file to a new disk or server.

Performs like a backup and restore process, but can be faster at times, depending on the situation.

- The database is taken offline and cannot be accessed by any users or applications. It will remain offline until it’s been reattached.

So, which one should you choose? Though a backup is the ideal option, there are cases where an attachment/detachment of the database may be your only choice.

Consider the following scenarios:

Your database contains many file groups. Attaching that can be quite cumbersome.

The best solution would be to back up the database and then restore it to the desired destination, as it will group all of the files together in the backup process.

Based on the size of the database, the backup/restore process takes a long time. However, the attaching/detaching of the database could be much quicker if it's needed as soon as possible.

- In this scenario, you can take the database offline, detach it, and re-attach to the new destination.

As you know, there are two main file groups when following the method of attaching databases. These files are .MDF and .LDF. The .MDF file is the database's primary data file, which holds its structure and data. The .LDF file holds the transactional logging activity and history.

However, a .BAK file that's created when backing up a database groups all of the files together and you restore different file versions from a single backup set.

Consider your situation before using either option, but also consider a backup and restore to be your first option, then look into the attach/detach method as your next option. Also, be sure to test it before you move forward with live data!

Attaching/Detaching the AdventureWorks2012 Database

Since you already attached this database, let's now detach it from the server. After that, you'll attach it again using SQL syntax.

Detaching the Database

In SQL Server, there's a stored procedure that will detach the database for you. This particular stored procedure resides in the 'master' database. Under the hood, you can see the complexity of the stored procedure by doing the following:

Click to expand the Databases folder

Click on System Databases, then the Master database

Click on Programmability

Click on Stored Procedures, then System Stored Procedures

- Find sys.sp_detach_db, right-click it and select 'Modify' in SSMS.

You'll then see its syntax.

For this, simply execute the stored procedure as is.

The syntax is the following:

```
USE master
```

```
GO
```

```
ALTER DATABASE DatabaseName SET SINGLE_USER WITH ROLLBACK  
IMMEDIATE
```

```
GO
```

```
EXEC master.dbo.sp_detach_db @dbname = N'DatabaseName', @skipchecks =  
'false'
```

```
GO
```

To expand a little on what is happening, you want to use the 'master' database to alter the database you'll be detaching and set it to single user instead of multi-user.

Finally, the value after @dbname allows you to specify the name of the database to be detached, and the @skipchecks being set to false means that the database engine will update the statistics information, identifying that the database has been detached. It's ideal to set this as @false whenever detaching a database so that the system holds current information about all databases.

Now, detach the AdventureWorks2012 database from your server instance. Use the above SQL example for some guidance.

Attaching Databases

Once you have detached your database, if you navigate to where your data directory is, you'll see that the AdventureWorks2012_Data.MDF file still exists – and it should since you only detached it and haven't deleted it.

Next, take the file path of the .MDF file and copy and paste it in a place that you can easily access, like on the notepad.

My location is C:\Program Files\Microsoft SQL

Server\MSSQL13.MSSQLSERVER\MSSQL\DATA.

Once you've connected to your instance and opened up a new query session, you will just need to use the path where the data file is stored. Once you have that, you can enter that value in the following SQL syntax examples in order to attach your database.

Below is the syntax used to attach database files and log files. So, in the following exercise, you'll skip attaching the log file completely, since you're not attaching this to a new server. So, you may omit the statement to attach the log file.

```
CREATE DATABASE DatabaseName ON (FILENAME = 'C:\SQL Data
Files\DatabaseName.mdf), (FILENAME = 'C:\SQL Data
Files\DatabaseName_log.ldf') FOR ATTACH
```

In the above example, I am calling out the statement to attach the log file if one is available. However, if you happen not to have the .LDF file but only the .MDF file, that's alright. You can just attach the .MDF file and the database engine will create a new log file and start writing activity to that particular log.

For this exercise, imagine that you've had to detach this database from an old server and that you're going to attach it to a new server. Go ahead and use the syntax above as an example in order to attach the AdventureWorks2012 database.

Each exercise gives an overview, as well as examples, before you start applying what you learn in each section.

Below is a sample of the Production.Product table that you'll be using in the next couple of exercises.

Product ID	Name	Product Number	Color	Standard Cost	List Price	Size
317	LL Crankarm	CA-5965	Black	0	0	NULL
318	ML Crankarm	CA-6738	Black	0	0	NULL
319	HL Crankarm	CA-7457	Black	0	0	NULL
320	Chainring Bolts	CB-2903	Silver	0	0	NULL
321	Chainring Nut	CN-6137	Silver	0	0	NULL

Query Structure and SELECT Statement

Understanding the syntax and its structure will be extremely helpful for you in the future. Let's delve into the basics of one of the most common statements, the SELECT statement, which is used solely to retrieve data.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name
```

In the above query, you're selecting two columns and all rows from the entire table. Since you're selecting two columns, the query performs much faster than if you had selected all of the columns like this:

```
SELECT *  
FROM Table_Name
```

Select all of the columns from the table Production.Product.

Using the same table, select only the ProductID, Name, Product Number, Color and Safety Stock Level columns.

(Hint: The column names do not contain spaces in the table!)

The WHERE Clause

The WHERE clause is used to filter the amount of rows returned by the query. This clause works by using a condition, such as if a column is equal to, greater than, less than, between, or like a certain value.

When writing your syntax, it's important to remember that the WHERE condition comes after the FROM statement.

EQUALS – This is used to find an exact match. The syntax below uses the WHERE clause for a column that contains the exact string of 'Value'. Note that strings need single quotes around them, while numerical values do not.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 = 'Value'
```

BETWEEN – Typically used to find values between a certain range of numbers or date ranges. It's important to note that the first value in the BETWEEN

comparison operator should be lower than the value on the right. Note the example below comparing between 10 and 100.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 BETWEEN 10 AND 100
```

GREATER THAN, LESS THAN, LESS THAN OR EQUAL TO, GREATER THAN OR EQUAL TO – SQL Server has comparison operators that you can use to compare certain values.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 <= 1000 --Note that you can also use <, >, >= or even  
<> for not equal to
```

LIKE – This searches for a value or string that is contained within the column. You still use single quotes but include the percent symbols indicating if the string is in the beginning, end or anywhere between ranges of strings.

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 LIKE '%Value%' --Searches for the word 'value' in any  
field
```

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 LIKE 'Value%' --Searches for the word 'value' at the  
beginning of the field
```

```
SELECT Column_Name1, Column_Name2  
FROM Table_Name  
WHERE Column_Name3 LIKE '%Value' --Searches for the word 'value' at the  
end of the field
```

IS NULL and IS NOT NULL – As previously discussed, NULL is an empty cell that contains no data. Eventually, you'll work with a table that does contain NULL

values, which you can handle in a few ways.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

WHERE Column_Name3 IS NULL --Filters any value that is NULL in that column, but don't put NULL in single quotes since it's not considered a string.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

WHERE Column_Name3 IS NOT NULL --Filters any value that is not NULL in that column. [\[40\]](#)

Using the Production.Product table again, select the Product ID, Name, Product Number and Color. Filter the products that are silver colored.

Using ORDER BY

The ORDER BY clause is simply used to sort data in ascending or descending order, specific to which column you want to start. This command also sorts in ascending order by default, so if you want to sort in descending order, use DESC in your command.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

```
WHERE Column_Name3 = 'Value'
```

ORDER BY Column_Name1, Column_Name2 --Sorts in descending order, but you can also include ASC to sort in ascending order.

```
SELECT Column_Name1, Column_Name2
```

```
FROM Table_Name
```

```
WHERE Column_Name3 = 'Value'
```

ORDER BY Column_Name1, Column_Name2 DESC --This sorts the data in descending order by specifying DESC after the column list.

Then, sort the results in the list price from smallest to largest.

(Hint: You won't use the \$ in your query and if needed, you can refer back to the sorting options you just reviewed.)

CHAPTER 7: SQL TRANSACTION

“Just because something doesn’t do what you planned it to do doesn’t mean it’s useless.” – Thomas Edison

What is an SQL Transaction?

Transactions are a sequence of tasks performed in a logical order against a database. We consider each transaction as a single unit of work. Transactions give you more control over your SQL behavior, which becomes essential if you are continuing to maintain your data integrity, or planning to avoid database errors.

In Chapter 3, we talked at length about data integrity and the various methods one can adopt to ensure the presence of such integrity. However, that takes into account the fact that somehow, multiple users will not have access to the same data.

In reality, this may not be the case.

In a situation where many users are trying to modify the data, there are chance occurrences of actions overlapping each other. In many instances, one user takes specific operations on data that may not be valid. She or he does not realize that the data has been compromised because another user took action at the same time. As neither user will be immediately aware of any form of change taking place, the data might continue to remain inconsistent for quite a while.

With this in mind, all users will continue to assume the data is intact and that there are no problems for them to oversee.

Let us try to understand the above scenario with an example. Assume that there are two users, User A, and User B, who are both working on a customer’s data at the same time.

User A might notice that the customer’s last name is incorrect, and will set about changing the name, immediately saving the data after making the change. User B might be working on another section of the customer’s data, such as the customer’s address or telephone number. However, User A’s changes might not reflect on User B’s side. User B might still be working on the customer’s data using the old, and wrong, last time, inadvertently changing the last name to the

incorrect entry when saving the data.

Now both users are unaware of the inconsistencies in their work, and this problem might go unnoticed for a long while. It might seem somewhat trivial when contemplating the fact that only a single error took place. However, imagine the occurrences repeating themselves overtime, creating inconsistencies on numerous data. When the time comes, you will be forced to look through all the data you have. If your database is a large one, this might take a considerable amount of your time.

With SQL transactions, consecutive actions or changes will not affect the validity of data that is seen by another user.

SQL transactions give you the power to commit SQL statements, which will apply said statements into the database. Alternatively, you have the ability to rollback, which will undo statements from the database.

In an SQL transaction, an action will convert into a transaction if it successfully passes four characteristics, denoted by the acronym ACID.

Atomic

This property focuses on the “everything or nothing” principle of a transaction. It ensures the performance of all the operations in a transaction. If an operation fails any point, the entire transaction they will be aborted. Additionally, if only a few statements are executed, the transactions are rolled back to their previous state. To complete a transaction successfully and to apply it to the database, all operations should be correct and none should be missing.

Consistent

With this property, the database must not merely be consistent at the beginning of the transaction, but after its completion as well. Hence, if a set of operations result in actions that change the consistency at any point of their performance, then the transaction will be reverted to its original state.

Isolated

When a particular transaction is taking place, the data might be inconsistent. This

inconsistency has the potential to affect the database and hence until the transaction is completed, the data will not be available for other operations. It will be free from any outside effect. In other words, it will be isolated. Additionally, no other transaction can influence or affect the isolated transaction in any way, ensuring that there is no compromise in integrity.

Durable

When all transactions are completed, any changes or modifications must be preserved and maintained. In the event of hardware or application error, there should be no loss of data; it should be available reliably and consistently. This characteristic ensures that your data is available in the event of such

One has to note that at any time, whether operations are completed or reverted, the transaction maintains the integrity of the database.

There are seven commands that you can use to manage transactions. These are:

COMMIT: Saves changes and commits them to the database.

ROLLBACK: Rolls back changes to a specific SAVEPOINT, or the beginning of a transaction. Changes are not applied to the database.

SAVEPOINT: Creates a savepoint – a form of marker – within a transaction. When you ROLLBACK, you can reach this point instead of the beginning of the transaction.

RELEASE SAVEPOINT: Removes or releases a savepoint. After this command, any ROLLBACK will revert the transaction to its original state.

SET TRANSACTION: Creates characteristics for the execution of a particular transaction.

START TRANSACTION: Sets the characteristics of the transactions and begins the transaction.

SET CONSTRAINTS: Establishes the constraint mode within a transaction. A constraint creates a condition where a decision is passed to either apply the constraint as soon as modifications on the data begin to occur or delay the application of the constraint until the completion of the transaction.

Time for another example.

It is important to remember that when you are about to start a transaction, the database is in its original state. At this state, you can expect the data to be consistent. Once you execute the transactions, the SQL statements within those transactions begin processing. If the process is successful, then one uses the COMMIT command. This command causes the SQL statements to update the database and dismiss the transaction. If the process of updating is not successful, then the ROLLBACK command comes into play. This process gives you a brief overview of commands. However, we will be going in-depth into their functions.

Commit

When all the statements are accomplished in a specific transaction, then the next step would be to terminate the transaction. One of the recommended ways to terminate a transaction is to commit all the changes to the database made so far. When you are confident of the changes you have performed so far, you can then utilize the COMMIT command. See the below syntax statements that you can use to commit a statement:

```
COMMIT [WORK] [AND CHAIN]
```

```
COMMIT [WORK] [AND NO CHAIN]
```

You must remember that the usage of the WORK entry is not mandatory. Only the COMMIT keyword will suffice to process the COMMIT command. In actuality, the COMMIT and COMMIT WORK entries have the same purpose. Earlier versions of SQL used to have the inclusion of the WORK keyword and some users might be more familiar with it. However, if you are not used to its usage, then you do not have to include it to execute the COMMIT command.

The AND CHAIN article is another optional inclusion in the COMMIT statement. It is not a commonly used statement in SQL implementations. The AND CHAIN clause simply tells the system to begin with a new transaction as soon as the current one ends. With this clause, you will have no need to use the SET TRANSACTION or START TRANSACTION statements, leaving the work to the system to continue to the next step automatically.

However, this might prove to be a disadvantage rather than a convenience. As you will need complete control over your transaction, it is best to use an alternative. Rather than using AND CHAIN in your COMMIT statement, try to utilize the

AND NO CHAIN parameter. This essentially lets your system know that it should not begin a new transaction with the same settings as the current one. In the case where there is no mention of either of the parameters (AND CHAIN or AND NO CHAIN), the AND NO CHAIN will be the default clause the system will adopt.

Your COMMIT statement might look like the below:

COMMIT;

We have not included any of the two clauses mentioned. However, know that if you would like the system to start a new transaction automatically, use the below statement:

COMMIT AND CHAIN;

Assume you have a table named EMPLOYEES with “names” and “salaries” columns.

NAME	SALARIES
John	5,000
Mary	7,000
Jennifer	9,000
Mark	10,000
Adam	7,000

If you would like to remove salaries that are 7,000, then you simply have to use the below:

SQL > DELETE FROM EMPLOYEES

WHERE SALARIES = 7,000

SQL > COMMIT;

Now, as mentioned before, if you would like the next transaction to begin after the execution of the current one, then your statement will look something like this:

SQL > DELETE FROM EMPLOYEES

WHERE SALARIES = 7,000

SQL > COMMIT AND CHAIN;

Rollback

Human error is a permanent fixture in any process. For this reason, if you find a situation that calls for a roll back, then this statement gives you the power to commit such an action.

By using ROLLBACK, you will undo the statements you have done so far, either bringing them back to a specific savepoint – if you had established such a savepoint – or sending them to the beginning of the transaction.

Let us look at some of the parameters or inclusions of a ROLLBACK statement:

ROLLBACK [WORK] [AND CHAIN]

ROLLBACK [WORK] [AND NO CHAIN]

When a user makes changes to data but does not use the COMMIT statement, then those changes are stored in a temporary format called a transaction log. Users can look at this unsaved data, analyzing it to check if it meets particular requirements. If the changes do not express the desired result users are aiming for, they can hit ROLLBACK, ensuring that the data is not saved, and they can work on it again.

Users can also use the ROLLBACK feature to send the database to a savepoint or the beginning of a transaction in the event of a hardware malfunction or application crash. If a power loss interrupted your work, then as soon as the system restarts, the ROLLBACK feature will review all pending transactions and will roll back all statements.

Unlike the COMMIT statement, ROLLBACK has the option to include a TO SAVEPOINT parameter. When using the TO SAVEPOINT clause, the system will not cancel the transaction. You will merely be taken to a specific point from where you can continue your work.

Should you wish to cancel the transaction simply remove the TO STATEMENT parameter from your statement.

Here is an example of the most fundamental form of a ROLLBACK statement:

ROLLBACK;

Notice that we have not used the AND CHAIN parameter. This is because, as

mentioned before in the section for COMMIT command, the system uses the AND NO CHAIN command by default. Alternatively, you can use the AND CHAIN command should you wish to initiate a new transaction after the conclusion of the current one. However, bear in mind that you cannot use the TO SAVEPOINT and AND CHAIN parameters at the same time. This is because a TO SAVEPOINT command requires the termination of the current transaction.

When including the TO SAVEPOINT parameter, ensure that you add the name of the savepoint. See an example of this process below:

```
ROLLBACK TO SAVEPOINT;
```

The SAVEPOINT value is replaced by the name of the savepoint as shown in the example below:

```
BEGIN
```

```
SELECT customer_number, customer_lastname, purchases
```

```
DELETE FROM customer_number WHERE purchases < 10000;
```

```
SAVEPOINT section_1;
```

```
ROLLBACK TO section_1;
```

```
END;
```

While the above example is a rather simple form of the command, its purpose is to give you clarity on SAVEPOINT usages and naming. Do note that you can enter as many parameters between the SAVEPOINT command and the ROLLBACK clause.

If you specify a savepoint name, then the rollback command will take you back to that particular savepoint, irrespective of the fact that there could be other savepoints in between.

```
BEGIN
```

```
SELECT customer_number, customer_lastname, purchases
```

```
DELETE FROM customer_number WHERE purchases < 10000;
```

```
SAVEPOINT section_1;
```

```
SAVEPOINT section_2;
```

```
SAVEPOINT section_3;  
SAVEPOINT section_4;  
ROLLBACK TO section_1;  
END;
```

Notice that even though you have SAVEPOINTS named from section_2 to section_4, you have asked the system to roll back to section_1. By doing so, the system will ignore all of the other savepoints.

Savepoint

While we are on the subject of savepoints, let us look at why these commands are important, and how to work with them.

Essentially, you will be working with a complex set of transactions. To make it easier to understand the transaction, you will group different sections into units. Breaking down transactions in this manner will allow you to identify problems easily and know which section to reach in order to solve the issue.

But the question arises: how can you reach these sections without having to change the entire transaction?

Why, you use the SAVEPOINT command, of course!

At this point, you should be aware of an important point. MySQL and Oracle support the SAVEPOINT parameter, but if you are working with SQL Server, you might be using the SAVE TRANSACTION query instead.

However, the functions of both SAVEPOINT and SAVE TRANSACTION parameters are the same.

Let us look at an example to understand how a savepoint function works.

STEP 1: Start Transaction

STEP 2: SQL Statements

STEP 3: If the SQL statement is successful, the commence execution. If the SQL statement is not successful, then roll back to the beginning of the transaction

STEP 4: Enter SAVEPOINT 1

STEP 5: SQL Statements

STEP 6: If the SQL statement is successful, the commence execution. If the SQL statement is not successful, then roll back to SAVEPOINT 1

STEP 7: Enter SAVEPOINT 2

STEP 8: SQL Statements

STEP 9: If the SQL statement is successful, the commence execution. If the SQL statement is not successful, then roll back to SAVEPOINT 2

STEP 10: COMMIT

In the above example, we set savepoints after working on SQL statements. When we perform this routine, we begin to work on specific SQL statements before jumping on to the next one. This ensures that we execute all statements properly, and if there is an error in the statements, we can rework them. Once the system deems statements fully successful, we can use a savepoint and move on to the next set of operations. By doing this, we save the integrity of the first group of operations. After the savepoint, we can continue our progress, knowing that there will not be any changes occurring to the first statements.

This becomes a convenient method of dealing with a set of actions without worrying about the integrity of previous actions.

The process of creating a savepoint is rather simple. You just have to use the following command:

`SAVEPOINT <name of the savepoint>`

The savepoint name does not have to be “SECTION_1”. That was the name chosen for the purpose of the example. You can select your own name, preferably something you can easily recollect when you want to.

Release savepoint

After you have completed certain operations within a transaction, you might not require the savepoints you had previously established. In order to remove them, you can use the `RELEASE SAVEPOINT` command.

However, do note that once you release a savepoint, you will not be able to roll back to it again. Which is why the ideal way to release a savepoint is to check if

you are satisfied with your work so far. Due diligence might be an added task, but it will provide you with the capability to complete your work in confidence.

To release a savepoint, simply enter the below command:

```
RELEASE SAVEPOINT <name of the savepoint>
```

Let us take an example where you have the below savepoints:

```
SAVEPOINT section_1;
```

```
SAVEPOINT section_2;
```

```
SAVEPOINT section_3;
```

```
SAVEPOINT section_4;
```

In order to release all of them, you will have to specify each one.

Hence, the process is as shown below:

```
RELEASE SAVEPOINT section_1;
```

```
RELEASE SAVEPOINT section_2;
```

```
RELEASE SAVEPOINT section_3;
```

```
RELEASE SAVEPOINT section_4;
```

You might release all savepoints at one time or you might choose to release one of the savepoints while keeping the rest. This is acceptable. All you have to ensure is that you use the right savepoint name. Another note to remember is that you do not have to use the release command in the order of the savepoints you have created. You can choose to release any savepoint at any time, and in any order.

Set Transaction

This command allows you to work with the different properties of a transaction. These could be one of the below:

- Read Only
- Read Write
- Specify an isolation level
- Attach it to a rollback property

Any effect of the SET TRANSACTION operation affects only your transaction. Other users will not notice any changes on their end.

You can establish SET TRANSACTION using the below command:

SET TRANSACTION <mode>

In the above command, the <mode> refers to the type of option you would like to specify. You can use three modes in SET TRANSACTION. These are:

- Access Level
- Isolation Level
- Diagnostics Size

You can add multiple transaction modes into the <mode> placeholder. If you wish to do that, separate the different modes by a comma. However, you cannot repeat the same mode again.

Example: you can include an isolation level and work on the diagnostics size. However, you cannot include two diagnostics size mode.

Now let us try to look at each mode separately.

Access Level

In a SET TRANSACTION, there are two types of access levels; READ ONLY and READ WRITE. If you select the READ ONLY option, the system will not allow you to make any changes to the database. However, if you choose the READ WRITE access level, you will be able to add statements in your transactions to modify the data or even the database structure.

Isolation Level

Do you remember how we talked about isolating transactions so that nothing can influence it while you are working on it? Well, here we will talk about setting up isolation levels.

For a SET TRANSACTION, you can use four isolation levels, as seen in the syntax below:

SET TRANSACTION ISOLATED LEVEL

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZABLE

The isolation levels are arranged in the order of their effectiveness, with the READ UNCOMMITTED being the least level of isolation you can use and the SERIALIZABLE being the highest level of isolation. If you do not specify the level of isolation, the system will assume the SERIALIZATION level, giving your transaction maximum isolation.

Diagnostic Size

The SET TRANSACTION allows you to specify a diagnostic size. The size you include lets the system know how much area to allow you for conditions. In an SQL statement, a condition is a message, warning, or other notifications raised by the statement. If you do not specify a diagnostic size, then your database will automatically assign you one. This number is not fixed. It varies from database to database.

Start Transaction

In a database, you can start a transaction explicitly or automatically when you begin executing a command.

For example, you can start a transaction when you use commands such as DELETE, CREATE TABLE, and so on.

Alternatively, you can commence a transaction by using the START TRANSACTION statement.

As with the SET TRANSACTION, you can specify one of more modes here as well. These are the access level, isolation level, and diagnostic size modes that we had mentioned before.

Here is an example of a START TRANSACTION command:

START TRANSACTION

READ ONLY

INSOLATION LEVEL SERIALIZED

DIAGNOSTICS SIZE 10;

Once you enter the syntax above, the START TRANSACTION will execute the statement and its operations.

Set Constraint

When you are working with transactions, you might encounter scenarios where your work will go against the constraints established in the database. Let us take an example.

Assume that you have a table that includes the NO NULL constraint. Now, while you are working on the table, you realize that you might not have a value to place instead of NO NULL constraint at that point in time. But you cannot leave without entering a value. This forces you to insert random values into the section.

To avoid such situations, you can define a constraint as deferrable.

The syntax for this action is shown below:

SET CONSTRAINTS (constraint_names) DEFERRED/IMMEDIATE

To mention multiple constraint names, you need to separate them by a bar. For example:

(constraint_1 | constraint_2 | constraint_3)

However, if you wish to choose all constraints – and this could be useful when you have lots of them in your database – then you simply have to use the word ALL

Here is an example:

SET CONSTRAINTS ALL DEFERRED/IMMEDIATE

Another thing to note is that you do not have to mention both DEFERRED and IMMEDIATE. You have to choose the one that fits your activity at that point.

For example, if you are working on the database, you will choose to defer the constraints, in which case your statement should look like this:

SET CONSTRAINTS ALL DEFERRED

Once you have completed operations on the database, you can then choose to apply the changes. In that case, the statement will look like this:

SET CONSTRAINT ALL IMMEDIATE

So through this chapter, you have understood the idea behind SQL transactions, and why they are essential when working with SQL programming.

CHAPTER 8: LOGINS, USERS AND ROLES

“See first that the design is wise and just; that ascertained, pursue it resolutely.” – William Shakespeare

Server Logins

Server logins are user accounts that are created in SQL servers and use SQL Server authentication, which entails entering a password and username to log into the SQL Server instance. SQL Server authentication is different from Windows Authentication, as Windows doesn't prompt you to enter your password.

These logins can be granted access to be able to log in to the SQL Server, but don't necessarily have database access depending on permissions that are assigned. They can also be assigned certain server level roles that provide certain server permissions within SQL Server.

Server Level Roles

Permissions are wrapped into logins by Server Level Roles. The Server Level Roles determine the types of permissions a user has.

There are nine types of predefined server level roles:

Sysadmin - can perform any action on the server, essentially the highest level of access.

Serveradmin - can change server configurations and shutdown the SQL server.

Securityadmin - can GRANT, DENY and REVOKE server level and database level permissions, if the user has access to the database. The securityadmin can also reset SQL Server passwords.

Processadmin - provides the ability to end processes that are running in a SQL Server instance.

Setupadmin - can add or remove linked servers by using SQL commands.

Diskadmin - this role is used to manage disk files in the SQL Server.

Dbcreator - provides the ability to use DDL statements, like CREATE, ALTER, DROP and even RESTORE databases.

Public - all logins are part of the 'public' role and inherit the permissions of the 'public' role if it does not have specific server or database object permissions.

If you really want to delve deeper into this topic (by all means, I encourage you to do so), then go ahead and [click this link](#) to get more information on server roles via the Microsoft website.

Below is an example of syntax that can be used to create server logins. You are able to swap out 'dbcreator' with any of the roles above when you assign a server level role to your login.

You'll also notice the brackets, [], around 'master' and 'User Name' for instance. These are delimited identifiers, which we have previously discussed.

--First, use the master database

USE [master]

GO

--Creating the login - replace 'User A' with the login name that you'd like to use

--Enter a password in the 'Password' field - it's recommended to use a strong password

--Also providing it a default database of 'master'

CREATE LOGIN [User A] WITH PASSWORD= N'123456',

DEFAULT_DATABASE=[master]

GO^[41]

Create a user with a login name called Server User. Give this user a simple password of 123456, and give it default database access to 'master'.

Assigning Server Roles

There are a few different ways to assign these roles to user logins in the SQL Server. One of the options is to use the interface in Management Studio, and the other is to use SQL syntax.

Below is the syntax to give a user a particular server role. If you recall the previous syntax we used to create the login for 'User A', this will give 'User A' a

server role of dbcreator.

--Giving the User a login a role of 'dbcreator', (it has the public role by default)

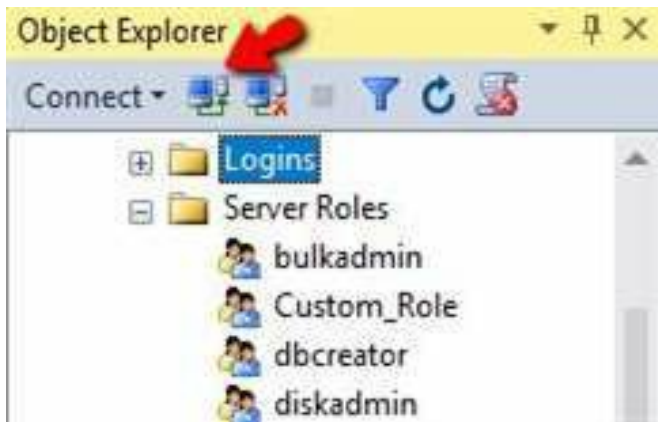
```
ALTER SERVER ROLE dbcreator ADD MEMBER [User A]
```

GO

For the above user that you created, called Server User, give it a server role of serveradmin.

After you've successfully completed the exercise, you'll receive a message telling you that your command has been completed successfully. You can now login to SQL Server using that account!

To log in, just navigate to the 'Connect Object Explorer' button in the Object Explorer window and click it to bring up the connection window.



When the window comes up to login, select the drop-down where it says 'Windows Authentication' and select 'SQL Server Authentication' instead. Just enter the user name and password and log in!

Connect to Server

SQL Server

Server type: Database Engine

Server name: HOME

Authentication: SQL Server Authentication

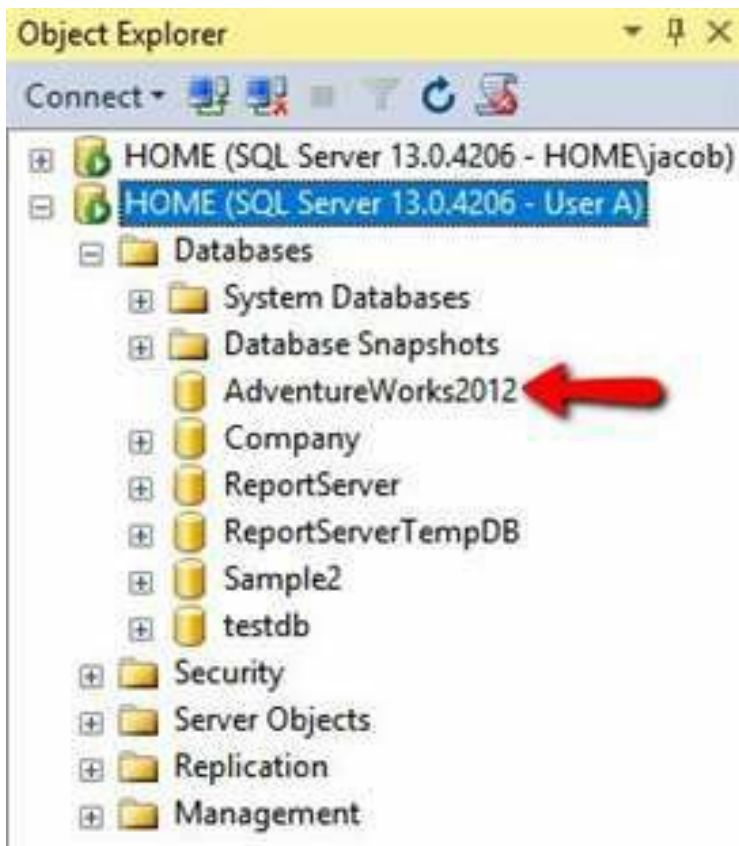
Login: User A

Password: *****

☐ Remember password

Connect Cancel Help Options >>

Once you are able to log in with this user, try to open the databases (other than the system databases) in the Object Explorer. It should look something like this; when you try to expand a database, it displays as blank instead.



You shouldn't be able to since this user doesn't have any database access. We'll cover this more thoroughly in the next section.

Database Users

A database user is a user that's typically associated with a login and mapped to a database or set of databases. The purpose of this database user is to provide access to the databases within the server itself. You can also restrict access to other databases.

In many cases, there are multiple databases on one server instance and you wouldn't want to give all database access to all of the users.

Database Level Roles

Much like the server level roles, there are database level roles that can be assigned to a user to control their permissions within certain databases.

There are nine predefined database roles:

Db_owner - provides the ability to perform all configuration and maintenance jobs on the database. This role also allows a user to DROP databases - a powerful command!

Db_securityadmin - provides the ability to modify role membership and manage permissions.

Db_accessadmin – provides the ability to add or remove access to the database for Windows logins, Windows groups and SQL logins.

Db_backupoperator - allows the user to back up the database.

Db_ddladmin - provides the ability to run any DDL statement in SQL Server.

Db_datawriter - provides the ability to add, delete or modify data within any user table.

Db_datareader - provides the ability to view any data from user tables.

Db_denydatawriter - cannot add, delete or modify data within any user table.

db_denydatareader - cannot view any data from user tables.

Assigning Database Roles and Creating Users

Like assigning server roles and creating logins, you can use either Management Studio or SQL syntax to create a database user and assign database level roles to that particular user. You can also map that user to a login so that the two are correlated.

Below is some sample syntax to associate a database user to a login, as well as assigning it a database role. It can become a little extensive, but I've broken it down into four parts to make it easier.

--Using the database that we'd like to add the user to

USE [AdventureWorks2012]

GO

--Creating the user called 'User A' for the login 'User A'

```
CREATE USER [User A] FOR LOGIN [User A]
```

```
GO
```

--Using AdventureWorks2012 again, since the database level role needs to be properly

--mapped to the user

```
USE [AdventureWorks2012]
```

```
GO
```

--Altering the role for db_datawriter and adding the database user 'User A' so

--the user can add, delete or modify existing data within AdventureWorks2012

```
ALTER ROLE [db_datawriter] ADD MEMBER [User A]
```

```
GO
```


LIKE Clause

SQL provides us with the LIKE clause that helps us compare a value to similar values using the wildcard operators.

The wildcard operators that are used together with the LIKE clause include the percentage sign (%) and the underscore (_).

The symbols have the syntax below:

```
SELECT FROM tableName  
WHERE column LIKE 'XXX%'
```

Or the following syntax:

```
SELECT FROM tableName  
WHERE column LIKE '%XXX%'
```

Or the following syntax:

```
SELECT FROM tableName  
WHERE column LIKE 'XXX_'
```

Or the following syntax:

```
SELECT FROM tableName  
WHERE column LIKE '_XXX'
```

Or the following syntax:

```
SELECT FROM tableName  
WHERE column LIKE '_XXX_'
```

If you have multiple conditions, combine them using the conjunctive operators (AND, OR). The XXX in the above case represents any string or numerical value.

Let's describe the meaning of some of the statements that you can create with the LIKE clause:

1. WHERE SALARY LIKE '300%'

To return any values that begin with 300.

2. WHERE SALARY LIKE '%300%'

This will return any values with 300 anywhere.

3. WHERE SALARY LIKE '_00%'

This will find the values with 00 in second and third positions.

4. WHERE SALARY LIKE '4_%_ %'

This will find the values that begin with 3 and that are at least 3 characters long.

5. WHERE SALARY LIKE '%3'

This will find the values that end with a 3.

6. WHERE SALARY LIKE '_3%4'

This will look for the values with a 3 in the second position and end with a 4.

7. WHERE SALARY LIKE '3__4'

This will find the values in a 5 digit number that begin with a 3 and end with a 4.

Now, we need to demonstrate how to use this clause. We will use the EMPLOYEES table with the data shown below:[\[42\]](#)

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel142	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nicoh442	45	5000.00
6	Milly	mil1342	32	2000.00
7	Grace	gra361	35	4000.00

Let's run a command that shows us all records in which the value of salary begins with 400:

```
SELECT * FROM EMPLOYEES
```

```
WHERE SALARY LIKE '400%';
```

The command returns the following:

```
mysql> SELECT * FROM EMPLOYEES
-> WHERE SALARY LIKE '400%';
```

ID	NAME	ADDRESS	AGE	SALARY
3	Joel	Joel142	30	4000.00
7	Grace	gra361	35	4000.00

```
2 rows in set (0.13 sec)

mysql>
```

SQL Functions^{[\[43\]](#)}

Here's the syntax:

SELECT COUNT (<expression>)

FROM table_name:

ID	EMP_NAME	SALES	BRANCH
1001	ALAN MARSCH	3000.00	NEW YORK
1098	NEIL BANKS	5400.00	LOS ANGELES
2005	RAIN ALONZO	4000.00	NEW YORK
3008	MARK FIELDING	3555.00	CHICAGO
4356	JENNER BANKS	14600.00	NEW YORK
4810	MAINE ROD	7000.00	NEW YORK
5783	JACK RINGER	6000.00	CHICAGO
6431	MARK TWAIN	10000.00	LOS ANGELES
7543	JACKIE FELTS	3500.00	CHICAGO
8934	MARK GOTH	5400.00	AUSTIN

10 rows in set (0.00 sec)

In this statement, the expression can refer to an arithmetic operation or column name. You can also specify (*) if you want to calculate the total records stored in the table. [\[44\]](#)

To perform a simple count operation, like calculating how many rows are in the SALES_REP table, enter:

SELECT COUNT(EMP_NAME)

FROM SALES_REP;

Here's the result:

COUNT(EMP_NAME)	
	10
1 row in set (0.24 sec)	

You can also use (*) instead of specifying a column name:[\[45\]](#)

```
SELECT COUNT(EMP_NAME)
FROM SALES_REP;
```

This statement will produce the same result because the EMP_NAME field has no NULL value. Assuming, however, that one of the fields in the EMP_NAME contains a NULL value, this would not be included in the statement that specifies EMP_NAME but will be included in the COUNT() result if you use the * symbol as a parameter.

BRANCH	COUNT(*)
AUSTIN	1
CHICAGO	3
LOS ANGELES	2
NEW YORK	4
4 rows in set (0.04 sec)	

```
SELECT BRANCH, COUNT(*) FROM SALES_REP
GROUP BY BRANCH;
```

This would be the output:

The COUNT() function can be used with DISTINCT to find the number of distinct entries. For instance, if you want to know how many distinct branches are saved in the SALES_REP table, enter the following statement:

```
SELECT COUNT (DISTINCT BRANCH)
FROM SALES_REP;
```


SQL AVG Function^[46]

Here is the syntax:

```
SELECT AVG (<expression>)
```

```
FROM “table_name”;
```

ID	EMP_NAME	SALES	BRANCH
1001	ALAN MARSCH	3000.00	NEW YORK
1098	NEIL BANKS	5400.00	LOS ANGELES
2005	RAIN ALONZO	4000.00	NEW YORK
3008	MARK FIELDING	3555.00	CHICAGO
4356	JENNER BANKS	14600.00	NEW YORK
4810	MAINE ROD	7000.00	NEW YORK
5783	JACK RINGER	6000.00	CHICAGO
6431	MARK TWAIN	10000.00	LOS ANGELES
7543	JACKIE FELTS	3500.00	CHICAGO
8934	MARK GOTH	5400.00	AUSTIN

10 rows in set (0.00 sec)

In the above statement, the expression can refer to an arithmetic operation or to a column name. Arithmetic operations can have single or multiple columns.

In the first example, you will use the AVG() function to calculate the average sales amount. You can enter the following statement:^[47]

AVG(SALES)
6245.500000

1 row in set (0.00 sec)

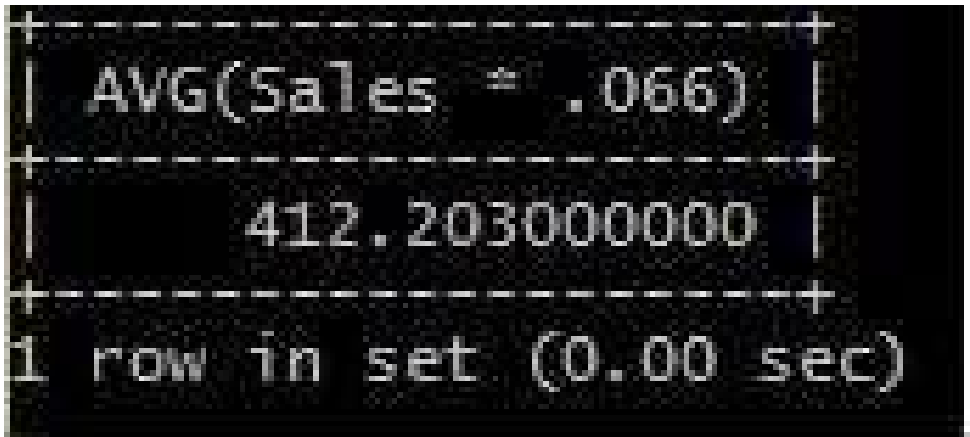
```
SELECT AVG(Sales) FROM Sales_Rep;
```

This is the result:

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

The figure 6245.500000 is the average of all sales data in the Sales_Rep table and it is computed by adding the Sales field and dividing the result by the number of records which, in this example, is 10 rows.

The AVG() function can be used in arithmetic operations. For example, assuming that sales tax is 6.6% of sales, you can use this statement to calculate the average sales tax figure:



A screenshot of a SQL query result displayed in a terminal window. The result is formatted with a dashed border. The first line shows the query: 'AVG(Sales * .066)'. The second line shows the result: '412.203000000'. The third line shows the execution summary: '1 row in set (0.00 sec)'.

AVG(Sales * .066)
412.203000000
1 row in set (0.00 sec)

```
SELECT AVG(Sales*.066) FROM Sales_Rep;
```

Here's the result:

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

To obtain the result, SQL had to calculate the result of the arithmetic operation 'Sales *.066' before applying the AVG function.

```
SELECT Branch, AVG(Sales) FROM Sales_Rep
```

Branch	AVG(Sales)
AUSTIN	5400.000000
CHICAGO	4351.666667
LOS ANGELES	7700.000000
NEW YORK	7150.000000

4 rows in set (0.06 sec)

GROUP BY Branch;

Here's the result:

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

SQL ROUND Function

The following is the syntax for SQL ROUND() function:

ROUND (expression, [decimal place])

In the above statement, the decimal place specifies the number of decimal points that will be returned. For instance, specifying -1 will round off the number to the nearest tens.

The examples on this section will use the Student_Grade table with the following data:

ID	Name	Grade
1	Jack Knight	87.6498
2	Daisy Poult	98.4359
3	James McDuff	97.7853
4	Alicia Stone	89.9753

To round off the grades to the nearest tenths, enter the following statement:

```
SELECT Name, ROUND (Grade, 1) Rounded_Grade FROM Student_Grade;
```



```
+-----+-----+
| Name          | Rounded_Grade |
+-----+-----+
| Jack Knight   | 87.6          |
| Daisy Poult   | 98.4          |
| James McDuff  | 97.8          |
| Alicia Stone  | 90.0          |
+-----+-----+
4 rows in set (0.46 sec)
```

Name	Rounded_Grade
Jack Knight	87.6
Daisy Poult	98.4
James McDuff	97.8
Alicia Stone	90.0

This would be the result:[\[48\]](#)

Assuming that you want to round the grades to the nearest tens, you will use a negative parameter for the ROUND() function:

Name	Rounded_Grade
Jack Knight	90
Daisy Poult	100
James McDuff	100
Alicia Stone	90

4 rows in set (0.04 sec)

```
SELECT Name, ROUND (Grade, -1) Rounded_Grade FROM Student_Grade;
```

Here's the result:

SQL SUM Function

The SUM() function is used to return the total for an expression.

Here's the syntax for the SUM() function:

```
SELECT SUM (<expression>)
```

```
FROM "table_name";
```

The expression parameter can refer to an arithmetic operation or a column name. Arithmetic operations may include one or more columns.

Likewise, there can be more than one column in the SELECT statement in addition to the column specified in the SUM() function. These columns should also form part of the GROUP BY clause. Here's the syntax:[\[49\]](#)

```
SELECT column n1, column n2, ... column N, SUM ("column nN+1")
```

```
FROM table_name;
```

```
GROUP BY column n1, column n2, ... column n_nameN;
```

For the examples in this section, you will use the SALES_REP table with the following data:[\[50\]](#)

To calculate the total of all sales from the Sales_Rep table, enter the following statement:

```
SELECT SUM(Sales) FROM Sales_Rep;
```



A screenshot of a terminal window displaying the result of a SQL query. The output is presented in a table-like format with a dashed border. The first row shows the column name 'SUM(Sales)'. The second row shows the calculated value '62455.00'. Below the table, it indicates '1 row in set (0.00 sec)'.

SUM(Sales)
62455.00

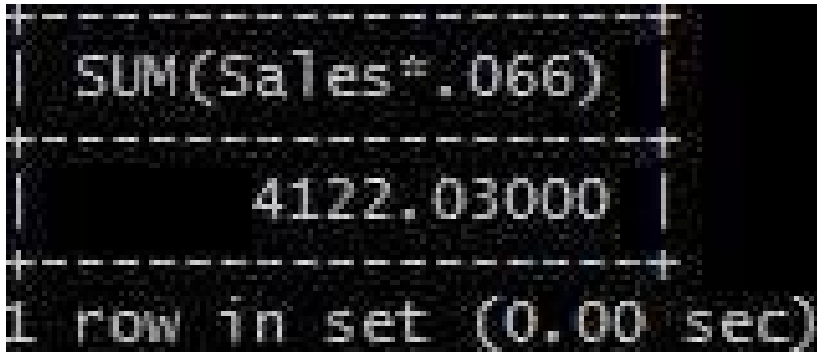
1 row in set (0.00 sec)

This would be the result: [\[51\]](#)

The figure 62455.00 represents the total of all entries in the Sales column.

To illustrate how you can use an arithmetic operation as an argument in the SUM() function, assume that you have to apply a sales tax of 6.6% on the sales figure. Here's the statement to obtain the total sales tax:

```
SELECT SUM(Sales*.066) FROM Sales_Rep;
```



A screenshot of a SQL query result displayed in a terminal window. The result is presented in a table format with dashed lines. The first row shows the query: SUM(Sales*.066). The second row shows the result: 4122.03000. The third row shows the execution summary: 1 row in set (0.00 sec).

SUM(Sales*.066)
4122.03000
1 row in set (0.00 sec)

You will get the following result: [\[52\]](#)

In this example, you will combine the SUM() function and the GROUP BY clause to calculate the total sales for each branch. You can use the following statement:

```
SELECT Branch, SUM(Sales) FROM Sales_Rep  
GROUP BY Branch;
```

Branch	SUM(Sales)
AUSTIN	5400.00
CHICAGO	13055.00
LOS ANGELES	15400.00
NEW YORK	28600.00
4 rows in set (0.00 sec)	

Here's the result: [\[53\]](#)

SQL MAX() Function

The MAX() function is used to obtain the largest value in a given expression.

Here's the syntax:

```
SELECT MAX (<expression>)
```

```
FROM table_name;
```

The expression parameter can be an arithmetic operation or a column name. Arithmetic operations can have multiple columns.

The SELECT statement can have one or more columns, aside from the column specified in the MAX() function. If this is the case, these columns will have to form part of the GROUP BY clause.

The syntax would be:

```
SELECT column1, column2, ... "columnN", MAX (<expression>)
```

```
FROM table_name;
```

```
GROUP BY column1, column2, ... "columnN";
```

ID	EMP_NAME	SALES	BRANCH
1001	ALAN MARSCH	3000.00	NEW YORK
1098	NEIL BANKS	5400.00	LOS ANGELES
2005	RAIN ALONZO	4000.00	NEW YORK
3008	MARK FIELDING	3555.00	CHICAGO
4356	JENNER BANKS	14600.00	NEW YORK
4810	MAINE ROD	7000.00	NEW YORK
5783	JACK RINGER	6000.00	CHICAGO
6431	MARK TWAIN	10000.00	LOS ANGELES
7543	JACKIE FELTS	3500.00	CHICAGO
8934	MARK GOTH	5400.00	AUSTIN

10 rows in set (0.00 sec)

To demonstrate this, you will use the Sales_Rep table with this data:[\[54\]](#)

To get the highest sales amount, enter the following statement:

```
SELECT MAX(Sales) FROM Sales_Rep;
```

Here's the result:



```
+-----+
| MAX(Sales) |
+-----+
|    14600.00 |
+-----+
1 row in set (0.08 sec)
```

A screenshot of a SQL query result displayed on a black background with white text. The result is a single row containing the value 14600.00, which is the maximum sales figure. The query used is SELECT MAX(Sales) FROM Sales_Rep;.

To illustrate how the MAX() function is applied to an arithmetic operation, assume that you have to compute a sales tax of 6.6% on the sales figure. To get the highest sales tax figure, use the following statement;

```
SELECT MAX(Sales*0.066) FROM Sales_Rep;[55]
```



```
+-----+
| MAX(Sales*.066) |
+-----+
|      963.60000 |
+-----+
1 row in set (0.00 sec)
```

A screenshot of a SQL query result displayed on a black background with white text. The result is a single row containing the value 963.60000, which is the maximum sales tax figure calculated as 6.6% of the maximum sales. The query used is SELECT MAX(Sales*0.066) FROM Sales_Rep;.

Here's the output:

You can combine the MAX() function with the GROUP BY clause to obtain the maximum sales value per branch. To do that, enter the following statement:

Branch	MAX(Sales)
AUSTIN	5400.00
CHICAGO	6000.00
LOS ANGELES	10000.00
NEW YORK	14600.00

4 rows in set (0.00 sec)

SELECT Branch, MAX(Sales) FROM Sales_Rep GROUP BY Branch;[\[56\]](#)

SQL MIN() Function

The MIN() function is used to obtain the lowest value in a given expression.

Here's the syntax:

```
SELECT MIN(<expression>)
```

```
FROM table_name;
```

The expression parameter can be an arithmetic operation or a column name. Arithmetic operations can also have several columns.

The SELECT statement can have one or several columns aside from the column specified in the MIN() function. If this is the case, these columns will have to form part of the GROUP BY clause.

The syntax would be:

```
SELECT column1, column2, ... "columnN", MIN (<expression>)
```

```
FROM table_name;
```

```
GROUP BY column1, column2, ... "columnN";
```

To demonstrate how the MIN() function is used in SQL, use the Sales_Rep table with the following data:[\[57\]](#)

ID	EMP_NAME	SALES	BRANCH
1001	ALAN MARSCH	3000.00	NEW YORK
1098	NEIL BANKS	5400.00	LOS ANGELES
2005	RAIN ALONZO	4000.00	NEW YORK
3008	MARK FIELDING	3555.00	CHICAGO
4356	JENNER BANKS	14600.00	NEW YORK
4810	MAINE ROD	7000.00	NEW YORK
5783	JACK RINGER	6000.00	CHICAGO
6431	MARK TWAIN	10000.00	LOS ANGELES
7543	JACKIE FELTS	3500.00	CHICAGO
8934	MARK GOTH	5400.00	AUSTIN

10 rows in set (0.00 sec)

To get the lowest sales amount, use the following statement:

```
SELECT MIN(Sales) FROM Sales_Rep;
```

The output would be: [\[58\]](#)

MIN(Sales)
3000.00

1 row in set (0.01 sec)

To demonstrate how the MIN() function is used on arithmetic operations, assume that you have to compute a sales tax of 6.6% on the sales figure. To get the lowest sales tax figure, use the following statement:

```
SELECT MIN(Sales*0.066) FROM Sales_Rep;
```

Here's the output:

```

+-----+
| MIN(Sales*.066) |
+-----+
|      198.00000  |
+-----+
1 row in set (0.00 sec)

```

You can also use the MIN() function with the GROUP BY clause to calculate the minimum sales value per branch. To do so, enter the following statement:

```
SELECT Branch, MIN(Sales) FROM Sales_Rep GROUP BY Branch;
```

```

+-----+-----+
| Branch      | MIN(Sales) |
+-----+-----+
| AUSTIN      | 5400.00    |
| CHICAGO     | 3500.00    |
| LOS ANGELES | 5400.00    |
| NEW YORK    | 3000.00    |
+-----+-----+
4 rows in set (0.00 sec)

```

Here's the result: [\[59\]](#)

CHAPTER 9: MODIFYING AND CONTROLLING TABLES

“The problem is not that there are problems. The problem is expecting otherwise and thinking that having problems is a problem.” – Theodore Rubin

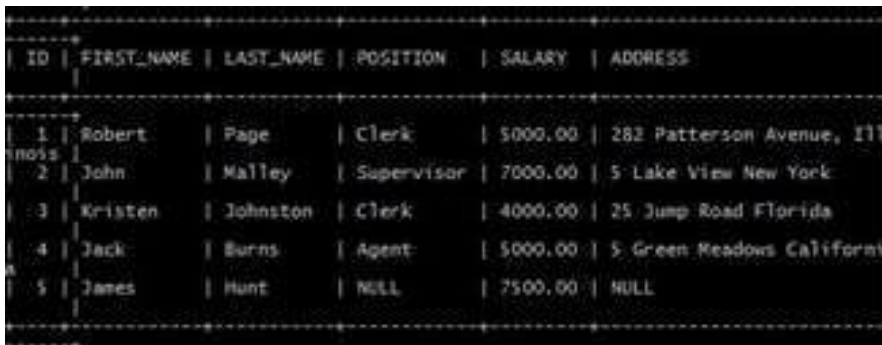
Here’s the basic syntax to alter a table:

```
ALTER TABLE TABLE_NAME [MODIFY] [COLUMN COLUMN_NAME]
[DATATYPE | NULL NOT NULL]
[RESTRICT | CASCADE]
[DROP] [CONSTRAINT CONSTRAINT_NAME]
[ADD] [COLUMN] COLUMN DEFINITION
```

Changing a Table’s Name

The ALTER TABLE command can be used with the RENAME function to change a table’s name.

To demonstrate the use of this statement, use the EMPLOYEES table with the following records:

A screenshot of a database table named EMPLOYEES. The table has six columns: ID, FIRST_NAME, LAST_NAME, POSITION, SALARY, and ADDRESS. There are five rows of data. The first row shows an employee with ID 1, first name Robert, last name Page, position Clerk, salary 5000.00, and address 282 Patterson Avenue, IT1. The second row shows an employee with ID 2, first name John, last name Malley, position Supervisor, salary 7000.00, and address 5 Lake View New York. The third row shows an employee with ID 3, first name Kristen, last name Johnston, position Clerk, salary 4000.00, and address 25 Jump Road Florida. The fourth row shows an employee with ID 4, first name Jack, last name Burns, position Agent, salary 5000.00, and address 5 Green Meadows California. The fifth row shows an employee with ID 5, first name James, last name Hunt, position NULL, salary 7500.00, and address NULL.

ID	FIRST_NAME	LAST_NAME	POSITION	SALARY	ADDRESS
1	Robert	Page	Clerk	5000.00	282 Patterson Avenue, IT1
2	John	Malley	Supervisor	7000.00	5 Lake View New York
3	Kristen	Johnston	Clerk	4000.00	25 Jump Road Florida
4	Jack	Burns	Agent	5000.00	5 Green Meadows California
5	James	Hunt	NULL	7500.00	NULL

To change the EMPLOYEES table name to INVESTORS, use the following statement:[\[60\]](#)

```
ALTER TABLE EMPLOYEES RENAME INVESTORS;
```

Your table is now called INVESTORS.

Modifying Column Attributes

A column's attributes refer to the properties and behaviors of data entered in a column. Normally, you set the column attributes when you create the table. However, you may still change one or more attributes using the ALTER TABLE command.

You may modify the following:

- Column name
- Column Data type assigned to a column
- The scale, length, or precision of a column
- Use or non-use of NULL values in a column

Renaming Columns

You may want to modify a column's name to reflect the data it contains. For instance, since you renamed the EMPLOYEES database to INVESTORS, the SALARY column will no longer be appropriate. You can change the column name to something like CAPITAL. Likewise, you may want to change its data type from DECIMAL to an INTEGER TYPE with a maximum of ten digits.

To do so, enter the following statement:

```
ALTER TABLE INVESTORS CHANGE SALARY CAPITAL INT(10);
```

[\[61\]](#) The result is the following:

ID	FIRST_NAME	LAST_NAME	POSITION	CAPITAL	ADDRESS
1	Robert	Page	Clerk	5000	282 Patterson Avenue, Illinois
2	John	Malley	Supervisor	7000	5 Lake View New York
3	Kristen	Johnston	Clerk	4000	25 Jump Road Florida
4	Jack	Burns	Agent	5000	5 Green Meadows California
5	James	Hunt	NULL	7500	NULL

5 rows in set (0.01 sec)

Deleting a Column

At this point, the Position column is no longer applicable. You can drop the column using the following statement:

```
ALTER TABLE INVESTORS
```

```
DROP COLUMN Position;
```

Here's the updated INVESTORS table: [\[62\]](#)

ID	FIRST_NAME	LAST_NAME	CAPITAL	ADDRESS
1	Robert	Page	5000	282 Patterson Avenue, Illinois
2	John	Malley	7000	5 Lake View New York
3	Kristen	Johnston	4000	25 Jump Road Florida
4	Jack	Burns	5000	5 Green Meadows California
5	James	Hunt	7500	NULL

5 rows in set (0.00 sec)

Adding a New Column

Since you're now working on a different set of data, you may decide to add another column to make the data on the INVESTORS table more relevant. You can add a column that will store the number of stocks owned by each investor. You may name the new column STOCKS. This column will accept integers up to 9 digits.

You can use the following statement to add the STOCKS column:

```
ALTER TABLE INVESTORS ADD STOCKS INT(9);
```

The following is the updated INVESTORS table: [\[63\]](#)

ID	FIRST_NAME	LAST_NAME	CAPITAL	ADDRESS	STOCKS
1	Robert	Page	5000.00	282 Patterson Avenue, Illinois	NUL
2	John	Malley	7000.00	5 Lake View New York	NUL
3	Kristen	Johnston	4000.00	25 Jump Road Florida	NUL
4	Jack	Burns	5000.00	5 Green Meadows California	NUL
5	James	Hunt	7500.00	NULL	NUL

Modifying an Existing Column without Changing its Name

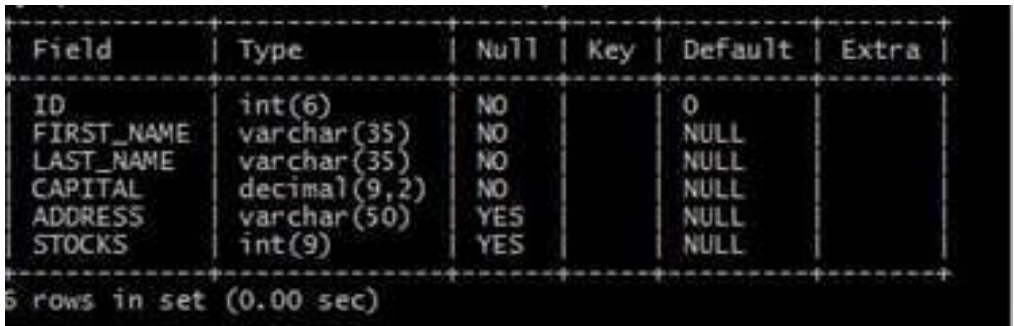
You may also combine the ALTER TABLE command with the MODIFY keyword to change the data type and specifications of a table. To demonstrate this, you can use the following statement to modify the data type of the column CAPITAL from an INT type to a DECIMAL type with up to 9 digits and two decimal numbers.

```
ALTER TABLE INVESTORS MODIFY CAPITAL DECIMAL(9,2) NOT NULL;
```

By this time, you may be curious to see the column names and attributes of the INVESTORS table. You can use the 'SHOW COLUMNS' statement to display the table's structure. Enter the following statement:

```
SHOW COLUMNS FROM INVESTORS;
```

Here's a screenshot of the result: [\[64\]](#)



The screenshot shows the output of the 'SHOW COLUMNS FROM INVESTORS;' command in a MySQL terminal. The output is a table with 6 columns: Field, Type, Null, Key, Default, and Extra. It lists 6 rows of data for the columns ID, FIRST_NAME, LAST_NAME, CAPITAL, ADDRESS, and STOCKS. At the bottom, it indicates '6 rows in set (0.00 sec)'.

Field	Type	Null	Key	Default	Extra
ID	int(6)	NO		0	
FIRST_NAME	varchar(35)	NO		NULL	
LAST_NAME	varchar(35)	NO		NULL	
CAPITAL	decimal(9,2)	NO		NULL	
ADDRESS	varchar(50)	YES		NULL	
STOCKS	int(9)	YES		NULL	

6 rows in set (0.00 sec)

Rules to Remember when Using ALTER TABLE

- Adding Columns to a Database Table

When adding a new column, bear in mind that you can't add a column with a NOT NULL attribute to a table with existing data. You will generally specify a column to be NOT NULL to indicate that it will hold a value. Adding a NOT NULL column will contradict the constraint if the existing data don't have values for a new column.

- Modifying Fields/Columns
 1. You can easily modify the data type of a column
 2. .You can increase the number of digits that numeric data types hold but you will only be able to decrease it if the largest number of digits stored by a table is equal to or lower than the desired number of digits.
 3. You can increase or decrease the decimal places of numeric data types as long as they don't exceed the maximum allowable decimal places.

If not handled properly, deleting and modifying tables can result to loss of valuable information. So, be extremely careful when you're executing the ALTER TABLE and DROP TABLE statements.

Deleting Tables

Dropping a table will also remove its data, associated index, triggers, constraints, and permission data. You should be careful when using this statement.

Here's the syntax:

```
DROP TABLE table_name;
```

For example, if you want to delete the INVESTORS TABLE from the xyzcompany database, you may use the following statement:

```
DROP TABLE INVESTORS;
```

The DROP TABLE command effectively removed the INVESTORS table from the current database.

If you try to access the INVESTORS table with the following command:

```
SELECT* FROM INVESTORS;
```

SQL will return an error, like this:

```
ERROR 1146 (42S02): Table 'xyzcompany.investors' doesn't exist
```

<https://docs.microsoft.com/en-us/sql/relational-databases/tables/delete-tables-database-engine?view=sql-server-2017>

Combining and joining tables

You can combine data from several tables if a common field exists between them. To do this, use the JOIN statement.

SQL supports several types of JOIN operations:

INNER JOIN

The INNER JOIN, or simply JOIN, is the most commonly used type of JOIN. It displays the rows when the tables to be joined have a matching field.

Here's the syntax: [\[65\]](#)

```
SELECT column n_name(s)
FROM table1
INNER JOIN table2
ON table1.column n_name=table2.column n_name;
```

In this variation, the JOIN clause is used instead of INNER JOIN.

```
SELECT column n_names(s)
FROM table1
JOIN table2
ON table1.column n_name=table2.column n_name;
```

LEFT JOIN

The LEFT JOIN operation returns all left table rows with the matching right table rows. If no match is found, the right side returns NULL.

[\[66\]](#) Here's the syntax for LEFT JOIN:

```
SELECT column n_name(s)
FROM table1
LEFT JOIN table2
ON table1.column n_name=table2.column n_name;
```

In some database systems, the keyword LEFT OUTER JOIN is used instead of LEFT JOIN. Here's the syntax for this variation: [\[67\]](#)

```
SELECT column n_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table2.column n_name=table2.column n_name;
```

RIGHT JOIN

This JOIN operation returns all right table rows with the matching left table rows.

The following is the syntax for this operation: [\[68\]](#)

```
SELECT column n_name(s)
FROM table1
RIGHT JOIN table2
ON table2.column n_name=table2.column n_name;
```

In some database systems, the RIGHT OUTER JOIN is used instead of LEFT JOIN. Here's the syntax for this variation:

```
SELECT column n_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table2.column n_name=table2.column n_name;
```

<http://www.sql-join.com/sql-join-types/>

FULL OUTER JOIN

This JOIN operation displays all rows when at least one table meets the condition. It combines the results from both RIGHT and LEFT join operations.

Here's the syntax:

```
SELECT column n_name(s)
```



```
FROM table1
FULL OUTER JOIN table2
ON table2.column_name=table2.column_name;
```

To demonstrate the JOIN operation in SQL, use the Branch_Sales and Branch_Location tales:

Branch_Sales Table

Branch	Product_ID	Sales
New York	101	7500.00
Los Angeles	102	6450.00
Chicago	101	1560.00
Philadelphia	101	1980.00
Denver	102	3500.00
Seattle	101	2500.00
Detroit	102	1450.00

Location Table

Region	Branch
East	New York City
East	Chicago
East	Philadelphia
East	Detroit
West	Los Angeles
West	Denver
West	Seattle

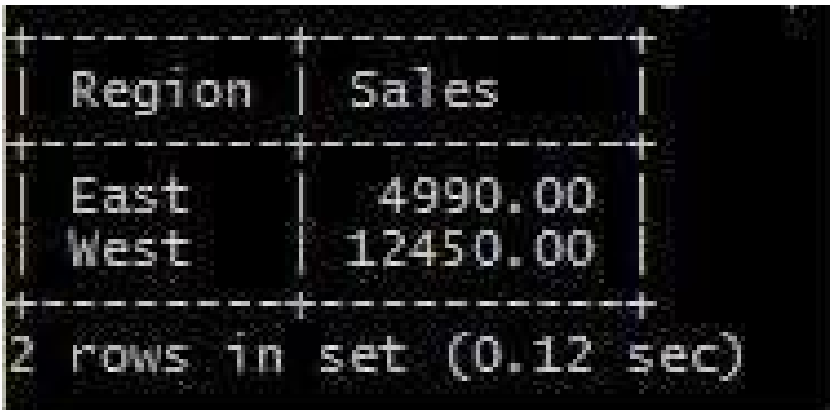
The objective is to fetch the sales by region. The Location table contains the data on regions and branches while the Branch_Sales table holds the sales data for each branch. To find the sales per region, you need to combine the data from the

Location and Branch_Sales tables. Notice that these tables have a common field, the Branch, which is the field that links the two tables.

The following statement will demonstrate how you can link these two tables by using table aliases:

```
SELECT A1.Region Region, SUM(A2.Sales) Sales
FROM Location A1, Branch_Sales A2
WHERE A1.Branch = A2.Branch
GROUP BY A1.Region;
```

This would be the result: [\[69\]](#)



Region	Sales
East	4990.00
West	12450.00

2 rows in set (0.12 sec)

In the first two lines, the statement tells SQL to select the fields ‘Region’ from the Location table and the total of the ‘Sales’ field from the Branch_Sales table. The statement uses table aliases. The ‘Region’ field was aliased as Region while the sum of the SALES field was aliased as SALES.

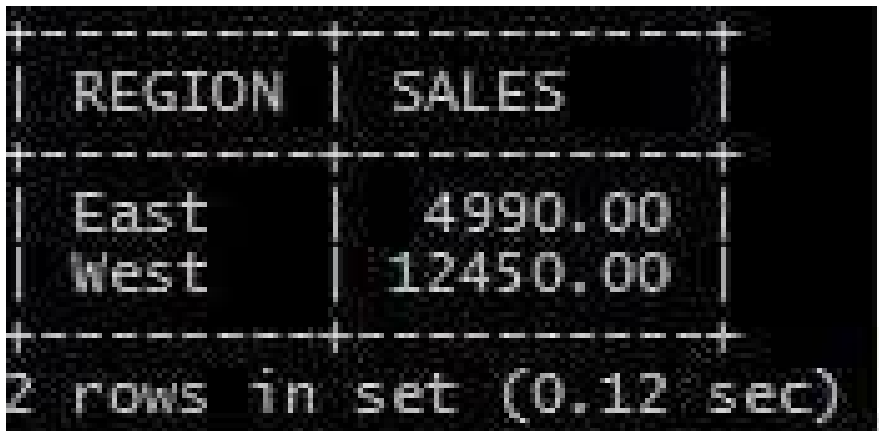
Table aliasing is the practice of using a temporary name for a table or a table column. Using aliases helps make statements more readable and concise. For example, if you opt not to use a table alias for the first line, you would have used the following statement to achieve the same result:

```
SELECT Location.Region Region,
SUM(Branch_Sales.Sales) SALES
```

Alternatively, you can specify a join between two tables by using the JOIN and ON keywords. For instance, using these keywords, the query would be:

```
SELECT A1.Region REGION, SUM(A2.Sales) SALES
FROM Location A1
JOIN Branch_Sales A2
ON A1.Branch = A2.Branch
GROUP BY A1.Region;
```

The query would produce an identical result: [\[70\]](#)



```
+-----+-----+
| REGION | SALES |
+-----+-----+
| East   | 4990.00 |
| West   | 12450.00 |
+-----+-----+
2 rows in set (0.12 sec)
```

Using Inner Join

An inner join displays rows when there is one or more matches on two tables. To demonstrate this, use the following tables:

Branch_Sales table

Branch	Product_ID	Sales
New York	101	7500.00
Philadelphia	101	1980.00
Denver	102	3500.00
Seattle	101	2500.00
Detroit	102	1450.00

Location_table

Region	Branch
East	New York
East	Chicago
East	Philadelphia
East	Detroit
West	Los Angeles
West	Denver
West	Seattle

You can achieve this by using the INNER JOIN statement.

You can enter the following:

```
SELECT A1.Branch BRANCH, SUM(A2.Sales) SALES
FROM Location A1
INNER JOIN Branch_Sales A2
ON A1.Branch = A2.Branch
GROUP BY A1.Branch;
```

[\[71\]](#) This would be the result:

BRANCH	SALES
Denver	3500.00
Detroit	1450.00
New York	7500.00
Philadelphia	1980.00
Seattle	2500.00
5 rows in set (0.00 sec)	

Take note that by using the INNER JOIN, only the branches with records in the Branch_Sales report were included in the results even though you are actually applying the SELECT statement on the Location table. The 'Chicago' and 'Los Angeles' branches were excluded because there are no records for these branches in the Branch_Sales table.

Using Outer Join

In the previous example, you have used the Inner Join to combine tables with common rows. The OUTER JOIN command is used for this purpose.

The example for the OUTER JOIN will use the same tables used for INNER JOIN: the Branch_Sales table and Location_table.

This time, you want a list of sales figures for all stores. A regular join would have excluded Chicago and Los Angeles because these branches were not part of the Branch_Sales table. Therefore, you want to do an OUTER JOIN.

The statement is the following:

```
SELECT A1.Branch, SUM(A2.Sales) SALES
FROM Location A1, Branch_Sales A2
```

WHERE A1.Branch = A2.Branch (+)

GROUP BY A1.Branch;

Please note that the Outer Join syntax is database-dependent. The above statement uses the Oracle syntax.

Here's the result: [\[72\]](#)

Branch	Sales
Chicago	NULL
Denver	3500.00
Detroit	1450.00
Los Angeles	NULL
New York	7500.00
Philadelphia	1980.00
Seattle	2500.00

When combining tables, be aware that some JOIN syntax have different results across database systems. To maximize this powerful database feature, it is important to read the RDBMS documentation.

LIMIT, TOP and ROWNUM Clauses

The TOP command helps us retrieve only the TOP number of records from the table. However, you must note that not all databases support the TOP command. Some will support the LIMIT while others will support the ROWNUM clause.

The following is the syntax to use the TOP command on the SELECT statement:

SELECT TOP number|percent columnName(s)

FROM tableName

WHERE [condition]

We want to use the EMPLOYEES table to demonstrate how to use this clause.

The table has the following data: [\[73\]](#)

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel42	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nicoh442	45	5000.00
6	Milly	mil342	32	2000.00
7	Grace	gra361	35	4000.00

The following query will help us fetch the first 2 rows from the table:

```
SELECT TOP 2 * FROM EMPLOYEES;
```

Note that the command given above will only work in SQL Server. If you are using MySQL Server, use the LIMIT clause as shown below:

```
SELECT * FROM EMPLOYEES
```

```
LIMIT 2;
```

```
mysql> SELECT * FROM EMPLOYEES
-> LIMIT 2;
+----+-----+-----+-----+-----+
| ID | NAME  | ADDRESS | AGE | SALARY |
+----+-----+-----+-----+-----+
| 2  | Mercy | Mercy32 | 25  | 3500.00 |
| 3  | Joel  | Joel42  | 30  | 4000.00 |
+----+-----+-----+-----+-----+
2 rows in set (0.37 sec)

mysql>
```

Only the first two records of the table are returned.

If you are using an Oracle Server, use the ROWNUM with SELECT clause, as shown below:

```
SELECT * FROM EMPLOYEES
```

```
WHERE ROWNUM <= 2;
```

ORDER BY Clause

This clause helps us sort our data, either in ascending or descending order. The sorting can be done while relying on one or more columns. In most databases, the results are sorted in an ascending order by default.

The ORDER BY clause uses the syntax below:

```
SELECT columns_list
```

```
FROM tableName
```

```
[WHERE condition]
```

In the ORDER BY clause, you may use one or more columns. However, you must ensure that the column you choose to sort the data is in the column list. Again, we will use the EMPLOYEES table with the data shown below: [\[75\]](#)

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel42	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nico442	45	5000.00
6	Milly	mil342	32	2000.00
7	Grace	gra361	35	4000.00

Now, we need to use the NAME and SALARY columns to sort the data in ascending order. The following command will help us achieve this:

```
SELECT * FROM EMPLOYEES
```

```
ORDER BY NAME, SALARY;
```

The query will return the following result: [\[76\]](#)


```
mysql> SELECT * FROM EMPLOYEES
-> ORDER BY NAME, SALARY;
```

ID	NAME	ADDRESS	AGE	SALARY
4	Alice	Alice442	31	2500.00
7	Grace	gra361	35	4000.00
3	Joel	Joel42	30	4000.00
2	Mercy	Mercy32	25	3500.00
6	Milly	mil342	32	2000.00
5	Nicholas	nicoh442	45	5000.00

```
6 rows in set (0.32 sec)

mysql>
```

We can also use the SALARY column to sort the data in descending order:

```
SELECT * FROM EMPLOYEES
```

```
ORDER BY SALARY DESC;
```

This is the result:

```
mysql> SELECT * FROM EMPLOYEES
-> ORDER BY SALARY DESC;
```

ID	NAME	ADDRESS	AGE	SALARY
5	Nicholas	nicoh442	45	5000.00
3	Joel	Joel42	30	4000.00
7	Grace	gra361	35	4000.00
2	Mercy	Mercy32	25	3500.00
4	Alice	Alice442	31	2500.00
6	Milly	mil342	32	2000.00

```
6 rows in set (0.05 sec)

mysql>
```

<https://www.tutorialspoint.com/sql/sql-top-clause.htm>

GROUP BY Clause

This clause is used together with the SELECT statement to group data that is related together, creating groups. The GROUP BY clause should follow the

WHERE clause in SELECT statements, and it should precede the ORDER BY clause.

The following is the syntax:

```
SELECT column_1, column_2
FROM tableName
WHERE [ conditions ]
GROUP BY column_1, column_2
ORDER BY column_1, column_2
```

Let's use the EMPLOYEES table with the data given below: [\[77\]](#)

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel42	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nicoh442	45	5000.00
6	Milly	mil342	32	2000.00
7	Grace	gra361	35	4000.00

If you need to get the total SALARY of every customer, just run the following command:

```
SELECT NAME, SALARY, SUM(SALARY) FROM EMPLOYEES
GROUP BY NAME;
```

The DISTINCT Keyword

This keyword is used together with the SELECT statement to help eliminate duplicates and allow the selection of unique records.

This is because there comes a time when you have multiple duplicate records in a table and your goal is to choose only the unique ones. The DISTINCT keyword can help you achieve this. This keyword can be used with the following syntax:

```
SELECT DISTINCT column_1, column_2,.....column_N
FROM tableName
```

WHERE [condition]

We will use the EMPLOYEES table to demonstrate how to use this keyword. The table has the following data: [\[78\]](#)

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel42	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nicoh442	45	5000.00
6	Milly	mil342	32	2000.00
7	Grace	gra361	35	4000.00

We have a duplicate entry of 4000 in the SALARY column of the above table. This can be seen after we run the following query:

```
SELECT DISTINCT SALARY FROM EMPLOYEES  
ORDER BY SALARY; \[79\]
```

```
mysql> SELECT SALARY FROM EMPLOYEES  
-> ORDER BY SALARY;  
+-----+  
| SALARY |  
+-----+  
| 2000.00 |  
| 2500.00 |  
| 3500.00 |  
| 4000.00 |  
| 4000.00 |  
| 5000.00 |  
+-----+  
6 rows in set (0.09 sec)  
mysql>
```

We can now combine the query with the DISTINCT keyword and see what the query returns:

[\[80\]](#)

```
mysql> SELECT DISTINCT SALARY FROM EMPLOYEES
-> ORDER BY SALARY;
+-----+
| SALARY |
+-----+
| 2000.00 |
| 2500.00 |
| 3500.00 |
| 4000.00 |
| 5000.00 |
+-----+
5 rows in set (0.07 sec)

mysql>
```

SQL Sub-queries

Until now, we have been executing single SQL queries to perform insert, select, update, and delete functions. However, there is a way to execute SQL queries within the other SQL queries. For instance, you can select the records of all students in the database with an age greater than a particular student. In this chapter, we shall demonstrate how we can execute sub-queries or queries-within-queries in SQL.

You should have tables labeled “Student” and “Department” with some records.

First, we can retrieve Stacy's age, store it in some variable, and then, using a "where" clause, compare the age in our SELECT query. The second approach is to embed the query that retrieves Stacy's age inside the query that retrieves the ages of all students. The second approach employs a sub-query technique. Have a look at Query 1 to see sub-queries in action.

Query 1

```
Select * From Student
```

```
where StudentAge >
```

```
(Select StudentAge from Student
```

```
where StudName = 'Stacy'
```

```
)
```

Notice that in Query 1, we've used round brackets to append a sub-query in the

“where” clause. The above query will retrieve the records of all students from the “Student” table where the age of the student is greater than the age of “Stacy”. The age of “Stacy” is 20; therefore, in the output, you shall see the records of all students aged greater than 20. The output is the following:

StudID	StudName	StudentAge	StudentGender	DepID
1	Alice	21	Male	2
4	Jacobs	22	Male	5
6	Shane	22	Male	4
7	Linda	24	Female	4
9	Wolfred	21	Male	2
10	Sandy	25	Female	1
14	Mark	23	Male	5
15	Fred	25	Male	2
16	Vic	25	Male	NULL
17	Nick	25	Male	NULL

Similarly, if you want to update the name of all the students with department name “English”, you can do so using the following sub-query:

Query 2

Update Student

Set StudName = StudName + ' Eng'

where Student.StudID in (

Select StudID

from Student

Join

Department

On Student.DepID = Department.DepID

where DepName = 'English'

)

In the above query, the student IDs of all the students in the English department have been retrieved using a JOIN statement in the sub-query. Then, using an UPDATE statement, the names of all those students have been updated by appending the string “Eng” at the end of their names. A WHERE statement has been used to match the student IDs retrieved by using a sub-query.

The IFNULL function checks if there is a Null value in a particular Table column. If a NULL value exists, it is replaced by the value passed as the second parameter to the IFNULL function. For instance, the following query will display 50 as the department ID of the students with a null department ID.

Now, if you display the Student’s name along with the name of their Department name, you will see “Eng” appended with the name of the students that belong to the English department.

Exercise 9

Task:

Delete the records of all students from the “Student” table where student’s IDs are less than the ID of “Linda”.

Solution

Delete From Student

where StudID <

(Select StudID from Student

where StudName = 'Linda'

)

SQL Character Functions

SQL character functions are used to modify the appearance of retrieved data. Character functions do not modify the actual data, but rather perform certain modifications in the way data is represented. SQL character functions operate on string type data. In this chapter, we will look at some of the most commonly used SQL character functions.

Note:

- Concatenation (+)

Concatenation functions are used to concatenate two or more strings. To concatenate two strings in SQL, the '+' operator is used. For example, we can join student names and student genders from the student column and display them in one column, like the following:

```
Select StudName '+' +StudentGender as NameAndGender  
from Student
```

- Replace

The replace function is used to replace characters in the output string. For instance, the following query replaces "ac" with "rs" in all student names.

```
Select StudName, REPLACE(StudName, 'ac', 'rs') as ModifiedColumn  
From Student
```

The first parameter in the replace function is the column whose value you want to replace; the second parameter is the character sequence which you want to replace, followed by the third parameter which denotes the character sequence you want to insert in place of the old sequence.

- Substring

The substring function returns the number of characters starting from the specified position. The following query displays the first three characters of student names.

```
Select StudName, substring(StudName, 1, 3) as SubstringColumn
```

From Student

- Length

The length function is used to get the length of values of a particular column. For instance, to get the length of names of students in the “Student” table, the following query can be executed:

```
Select StudName, Len(StudName) as NameLength  
from Student
```

Note that in the above query, we used the Len() function to get the length of the names; this is because in the SQL server, the Len() function is used to calculate the length of any string.

- IFNULL

The IFNULL function checks if there is a Null value in a particular Table column. If a NULL value exists, it is replaced by the value passed as the second parameter to the IFNULL function. For instance, the following query will display 50 as the department ID of the students with a null department ID.

```
Select Student.DepID, IFNULL(Student.DepID, 50)  
from Student
```

- LTRIM

The LTRIM function trims all the empty spaces from the values in the column specified as parameters to the LTRIM function. For instance, if you want to remove all the empty spaces before the names of the students in the “Student” table, you can use the LTRIM query, like the following:

```
Select Student.StudName, LTRIM(Student.StudName)  
from Student
```

- RTRIM

The RTRIM function trims all the proceeding empty spaces from the values in the column specified as parameters to the RTRIM function. For instance, if you want to remove all the empty spaces that come after the names of the students in the “Student” table, you can use the RTRIM query, like the following:


```
Select Student.StudName, RTRIM(Student.StudName)
```

```
from Student
```

SQL Constraints

Constraints refer to rules that are applied on the columns of database tables. They help us impose restrictions on the kind of data that can be kept in that table. This way, we can ensure that there is reliability and accuracy of the data in the database.

Constraints can be imposed at column level or at the table level. The column constraints can only be imposed on a single column, while the table level constraints are applied to the entire table.

NOT NULL Constraint

The default setting in SQL is that a column may hold null values. If you don't want to have a column without a value, you can specify this.

Note that NULL means unknown data rather than no data. This constraint can be defined when you are creating table. Let's demonstrate this by creating a sample table:


```
CREATE TABLE MANAGERS(  
    ID INT NOT NULL,  
    NAME VARCHAR (15) NOT NULL,  
    DEPT VARCHAR(20) NOT NULL,  
    SALARY DECIMAL (20, 2),  
    PRIMARY KEY (ID)  
);
```

Above, we have created a table named MANAGERS with 4 columns, and the NOT NULL constraint has been imposed on three of these columns. This means that you must specify a value for each of these columns with the constraint; otherwise, an error will be raised.

Note that we did not impose the NOT NULL constraint to the SALARY column of our table. It is possible for us to impose the constraint on the column even though it has already been created.

ALTER TABLE MANAGERS

MODIFY SALARY DECIMAL (20, 2) NOT NULL;



```
mysql> ALTER TABLE MANAGERS  
-> MODIFY SALARY DECIMAL (20, 2) NOT NULL;  
Query OK, 0 rows affected (0.23 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
mysql>
```

Now, the column cannot accept a null value.

Default Constraint

This constraint will provide a default value to the column if a value for the column is not specified in the INSERT INTO column.

Consider the example given below:

```
CREATE TABLE SUPPLIERS(  
    ID INT NOT NULL,  
    NAME VARCHAR (15) NOT NULL,  
    ADDRESS CHAR (20) ,  
    ARREARS DECIMAL (16, 2) DEFAULT 6000.00,  
    PRIMARY KEY (ID)  
);
```

Suppose you had already created the table but you needed to add a default constraint on the ARREARS column. You can do it like this:

```
MODIFY ARREARS DECIMAL (16, 2) DEFAULT 6000.00;
```

This will update the table, and a default constraint will be created for the column.

If you no longer need this default constraint to be in place, you can drop it by running the command given below:

```
ALTER TABLE SUPPLIERS  
    ALTER COLUMN ARREARS DROP DEFAULT;
```

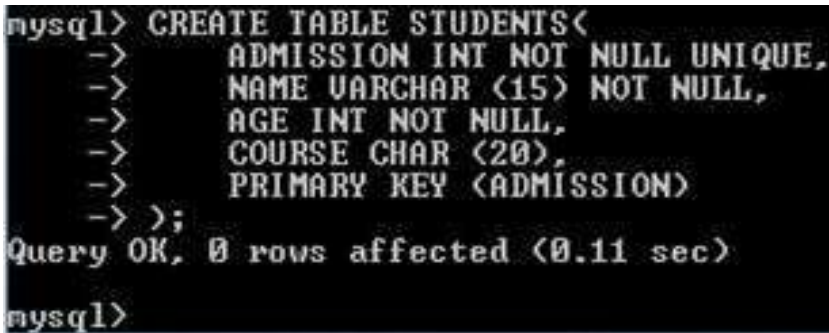
Unique Constraint

This constraint helps us avoid the possibility of having two or more records with similar values in one column. In the “employees” table, for example, we may need to prevent two or more employees from sharing the same ID.

The following SQL Query shows how we can create a table named “STUDENTS”. We will impose the UNIQUE constraint on the “ADMISSION” column:

```
CREATE TABLE STUDENTS(  
    ADMISSION INT NOT NULL UNIQUE,  
    NAME VARCHAR (15) NOT NULL,  
    AGE INT NOT NULL,  
    COURSE CHAR (20),  
    PRIMARY KEY (ADMISSION)  
);
```

[\[83\]](#)



```
mysql> CREATE TABLE STUDENTS(  
->     ADMISSION INT NOT NULL UNIQUE,  
->     NAME VARCHAR (15) NOT NULL,  
->     AGE INT NOT NULL,  
->     COURSE CHAR (20),  
->     PRIMARY KEY (ADMISSION)  
-> );  
Query OK, 0 rows affected (0.11 sec)  
mysql>
```

Properly created indexes enhance efficiency in large databases. The selection of fields on which to create the index depends on the SQL queries that you use frequently.

Perhaps you had already created the STUDENTS table without the UNIQUE constraint. The constraint can be added on the ADMISSION column by running the following command:

ALTER TABLE STUDENTS

MODIFY ADMISSION INT NOT NULL UNIQUE;

This is demonstrated below:

ALTER TABLE STUDENTS

ADD CONSTRAINT uniqueConstraint UNIQUE(ADMISSION, AGE);

The constraint has been given the name “uniqueConstraint” and assigned two columns, ADMISSION and AGE.

Anytime you need to delete the constraint, run the following command combining the ALTER and DROP commands:

ALTER TABLE STUDENTS

DROP CONSTRAINT uniqueConstraint;

The constraint will be deleted from the two columns. For MySQL users, the above command will not work. Just run the command given below:

ALTER TABLE STUDENTS

DROP INDEX uniqueConstraint; [\[84\]](#)

```
mysql> ALTER TABLE STUDENTS
-> DROP INDEX uniqueConstraint;
Query OK, 0 rows affected (0.15 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Primary Key

The primary key constraint helps us identify every row uniquely. The column designated as the primary key must have unique values. Also, the column is not allowed to have NULL values.

Each table is allowed to only have one primary key, and this may be made up of single or multiple fields. When we use multiple fields as the primary key, they are referred to as a composite key. If a column for a table is defined as the primary key, then no two records will share same value in that field.

A primary key is a unique identifier. In the STUDENTS table, we can define the ADMISSION to be the primary key since it identifies each student uniquely. No two students should have the same ADMISSION number. Here is how the attribute can be created:

```
CREATE TABLE STUDENTS(  
    ADMISSION INT NOT NULL,  
    NAME VARCHAR (15) NOT NULL,  
    AGE INT NOT NULL,  
    PRIMARY KEY (ADMISSION)  
);
```

The ADMISSION has been set as the Primary Key for the table. If we describe the table, you will find that the field is the primary key, as shown below: [\[85\]](#)

```
mysql> desc Students;  
+-----+-----+-----+-----+-----+-----+  
| Field      | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| ADMISSION  | int(11)       | NO   | PRI | NULL    |      |  
| NAME       | varchar(15)   | NO   |     | NULL    |      |  
| AGE        | int(11)       | NO   |     | NULL    |      |  
+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.42 sec)  
  
mysql>
```

In the “Key” field above, the “PRI” indicates that the ADMISSION field is the primary key.

You may want to impose the primary key constraint on a table that already exists.

This can be done by running the command given below:

```
ALTER TABLE STUDENTS
```

```
ADD CONSTRAINT PK_STUDADM PRIMARY KEY (ADMISSION,  
NAME);
```

In the above command, the primary key constraint has been given the name “PK_STUDADM” and assigned to two columns namely ADMISSION and NAME. This means that no two rows will have the same value for these columns.

The primary key constraint can be deleted from a table by executing the command given below:

```
ALTER TABLE STUDENTS DROP PRIMARY KEY;
```

After running the above command, you can describe the STUDENTS table and see whether it has any primary key: [\[86\]](#)

```
mysql> DESC STUDENTS;
```

Field	Type	Null	Key	Default	Extra
ADMISSION	int(11)	NO		NULL	
NAME	varchar(15)	NO		NULL	
AGE	int(11)	NO		NULL	

```
3 rows in set (0.00 sec)  
  
mysql>
```

This shows that the primary key was dropped successfully.

Foreign Key

This constraint is used to link two tables together. Sometimes, it is referred to as a referencing key. The foreign key is simply a column or a set of columns that match a Primary Key in another table.

Consider the tables with the structures given below:

STUDENTS table:

```
CREATE TABLE STUDENTS(  
    ADMISSION INT NOT NULL,  
    NAME VARCHAR (15) NOT NULL,  
    AGE INT NOT NULL,  
    PRIMARY KEY (ADMISSION)  
);
```

FEE table:

```
CREATE TABLE FEE (  
    ID INT NOT NULL,  
    DATE DATETIME,  
    STUDENT_ADM INT references STUDENTS (ADMISSION),  
    AMOUNT float,  
    PRIMARY KEY (ID)  
);[87]
```

In the FEE table, the STUDENT_ADM field is referencing the ADMISSION field in the STUDENTS table. This makes the STUDENT_ADM column of FEE table a foreign key.

If we had created the FEE table without the foreign key, we could've still added it with the following command:

```
ALTER TABLE FEES  
ADD FOREIGN KEY (STUDENT_ADM) REFERENCES STUDENTS
```

(ADMISSION);

The foreign key will be added to the table. If you need to delete the foreign key, run the following command:

```
ALTER TABLE FEE
```

```
DROP FOREIGN KEY;
```

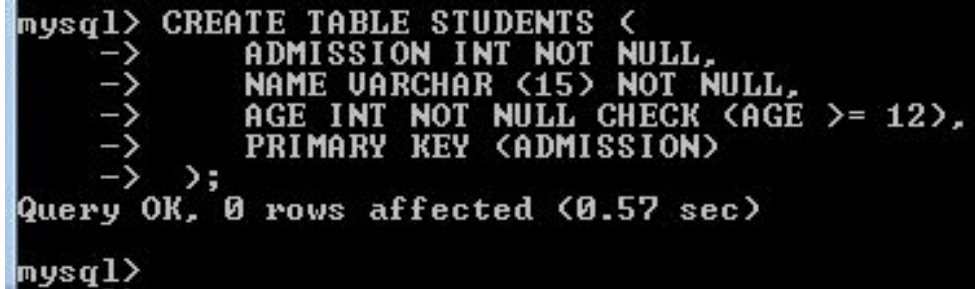
The foreign key will then be removed from the table.

CHECK Constraint

This constraint is used to create a condition that will validate the values that are entered into a record. If the condition becomes false, the record is violating the constraint, so it will not be entered into the table.

We want to create a table called STUDENTS and we don't want to have any student who is under 12 years of age. If the student doesn't meet this constraint, they will not be added to the table. The table can be created as follows:

```
CREATE TABLE STUDENTS (  
    ADMISSION INT NOT NULL,  
    NAME VARCHAR (15) NOT NULL,  
    AGE INT NOT NULL CHECK (AGE >= 12),  
    PRIMARY KEY (ADMISSION)  
); \[88\]
```



```
mysql> CREATE TABLE STUDENTS (  
->     ADMISSION INT NOT NULL,  
->     NAME VARCHAR (15) NOT NULL,  
->     AGE INT NOT NULL CHECK (AGE >= 12),  
->     PRIMARY KEY (ADMISSION)  
-> );  
Query OK, 0 rows affected (0.57 sec)  
mysql>
```

The table has been created successfully.

If you had already created the STUDENTS table without the constraint but then needed to implement it, run the command given below:

```
ALTER TABLE STUDENTS  
    MODIFY AGE INT NOT NULL CHECK (AGE >= 12);
```

The constraint will be added to the column AGE successfully.

It is also possible for you to assign a name to the constraint. This can be done

using the below syntax:

```
ALTER TABLE STUDENTS
```

```
ADD CONSTRAINT checkAgeConstraint CHECK(AGE >= 12);
```

<https://www.tutorialspoint.com/sql/sql-index.htm>

INDEX Constraint

An INDEX helps us quickly retrieve data from a database. To create an index, we can rely on a column or a group of columns in the database table. Once the index has been created, it is given a ROWID for every row before it can sort the data.

Properly created indexes enhance efficiency in large databases. The selection of fields on which to create the index depends on the SQL queries that you use frequently.

Suppose we created the following table with three columns:

```
CREATE TABLE STUDENTS (  
  ADMISSION INT NOT NULL,  
  NAME VARCHAR (15) NOT NULL,  
  AGE INT NOT NULL CHECK (AGE >= 12),  
  PRIMARY KEY (ADMISSION)  
);
```

We can then use the below syntax to implement an INDEX on one or more columns:

```
CREATE INDEX indexName  
  ON tableName ( column_1, column_2.....);
```

Now, we need to implement an INDEX on the column named AGE to make it easier for us to search using a specific age. The index can be created as follows:

```
CREATE INDEX age_idx  
  ON STUDENTS (AGE); \[89\]
```

```
mysql> CREATE INDEX age_idx  
->      ON STUDENTS (AGE);  
Query OK, 1 row affected (0.20 sec)  
Records: 1  Duplicates: 0  Warnings: 0  
  
mysql>
```

If the index is no longer needed, it can be deleted by running the following command:

ALTER TABLE STUDENTS

DROP INDEX age_idx; [\[90\]](#)

```
mysql> ALTER TABLE STUDENTS
-> DROP INDEX age_idx;
Query OK, 1 row affected (0.13 sec)
Records: 1 Duplicates: 0 Warnings: 0
mysql>
```

ALTER TABLE Command

SQL provides us with the ALTER TABLE command that can be used for addition, removal and modification of table columns. The command also helps us to add and remove constraints from tables.

Suppose you had the STUDENTS table with the following data: [\[91\]](#)

ADMISSION	NAME	AGE
3420	NICHOLAS	10
1234	john	32
3456	mercy	23

ALTER TABLE STUDENTS ADD COURSE VARCHAR(10); [\[92\]](#)

```
mysql> ALTER TABLE STUDENTS ADD COURSE VARCHAR(10);
Query OK, 3 rows affected (0.15 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql>
```

When we query the contents of the table, we get the following: [\[93\]](#)

ADMISSION	NAME	AGE	COURSE
3420	NICHOLAS	10	NULL
1234	john	32	NULL
3456	mercy	23	NULL

This shows that the column has been added and each record has been assigned a NULL value in that column.

To change the data type for the COURSE column from VarChar to Char, execute the following command:

ALTER TABLE STUDENTS MODIFY COLUMN COURSE Char(1); [\[94\]](#)

```
mysql> ALTER TABLE STUDENTS MODIFY COLUMN COURSE Char(1);
Query OK, 3 rows affected (0.13 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

We can combine the ALTER TABLE command with the DROP TABLE command to delete a column from a table. To delete the COURSE column from the STUDENTS table, we run the following command:

ALTER TABLE STUDENTS DROP COLUMN COURSE; [\[95\]](#)

```
mysql> ALTER TABLE STUDENTS DROP COLUMN COURSE;
Query OK, 3 rows affected (0.18 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

We can then view the table details via the describe command to see whether the column was dropped successfully: [\[96\]](#)

```
mysql> desc students;
```

Field	Type	Null	Key	Default	Extra
ADMISSION	int(11)	NO	PRI	NULL	
NAME	varchar(15)	NO		NULL	
AGE	int(11)	NO		NULL	

```
3 rows in set (0.03 sec)
```

```
mysql>
```

The above figure shows that the column was dropped successfully.

CONCLUSION AND NEXT STEPS

I hope that you enjoyed the book and were able to get as much information out of it as possible. If you like, feel free to go back to any other sections of the book and bounce around to sharpen your skills.

Product Review and Feedback

Rather than just giving you a standard conclusion, I'd like to give you some suggestions on the next steps that you can take to continue learning SQL. But first, if you have some feedback, hop over to the Amazon product page for this book and leave an honest review, or email me directly at jaschulz0705@gmail.com.

More Database Samples

If you're interested in working with more databases, then I have a few links that you will be interested in. You can download the Chinook and Northwind databases from the following links and begin working with those.

Go ahead and [click this link to download the Chinook database](#). Once you're there, click the arrow on the top-right to download a zip file. Once you've done that, extract the zip file to a location that is easy to access on your computer. Now, open the folder and drag and drop either the `Chinook_SqlServer.sql` file or `Chinook_SqlServer_AutoIncrementPKs.sql` into SSMS and click the 'Execute' button. You'll then have a full Chinook database.

In addition to the Chinook database, you can [click this link to download the Northwind database](#). Once here, click the arrow on the top-right to download another zip file. Then, extract the zip file from here and place it with the rest of your SQL backup files. Then, restore the database since it's a `Northwind.bak` file (database backup file). You can also refer to the previous sections for database restores and the fundamentals of SSMS to assist with these tasks!

Keep Learning

Dig into the AdventureWorks, Company_Db, Chinook and Northwind databases by running queries and understanding the data. Part of learning SQL is understanding the data within a database, too.

To protect databases, SQL has a security scheme that lets you specify which database users can view specific information from. This scheme also allows you to set what actions each user can perform. This security scheme (or model) relies on authorization identifiers. As you've learned in the second chapter, authorization identifiers are objects that represent one or more users that can access/modify the information inside the database.

More References

MTA 98-364 Certification

You can find the information for the certification here: <https://www.microsoft.com/en-us/learning/exam-98-364.aspx>. Once you're there, be sure to expand on the topics in the 'Skills Measured' section, as your database knowledge in these areas will be put to the test.

That's all for now.

REFERENCES

1keydata.com. (2019). *SQL - CREATE VIEW Statement | IKeydata*. [online] Available at: <https://www.1keydata.com/sql/sql-create-view.html> [Accessed 3 Feb. 2019].

Chartio. (2019). *How to Alter a Column from Null to Not Null in SQL Server*. [online] Available at: <https://chartio.com/resources/tutorials/how-to-alter-a-column-from-null-to-not-null-in-sql-server/> [Accessed 3 Feb. 2019].

Docs.microsoft.com. (2019). *Primary and Foreign Key Constraints - SQL Server*. [online] Available at: <https://docs.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-2017> [Accessed 3 Feb. 2019].

Sites.google.com. (2019). *DDL Commands - Create - Drop - Alter - Rename - Truncate - Programming Languages*. [online] Available at: <https://sites.google.com/site/prgimr/sql/ddl-commands---create---drop---alter> [Accessed 3 Feb. 2019].

Techonthenet.com. (2019). *SQL: UNION ALL Operator*. [online] Available at: https://www.techonthenet.com/sql/union_all.php [Accessed 3 Feb. 2019].

1keydata.com. (2019). *SQL - RENAME COLUMN | IKeydata*. [online] Available at: <https://www.1keydata.com/sql/alter-table-rename-column.html> [Accessed 3 Feb. 2019].

Docs.microsoft.com. (2019). *Create Check Constraints - SQL Server*. [online] Available at: <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-check-constraints?view=sql-server-2017> [Accessed 3 Feb. 2019].

Docs.microsoft.com. (2019). *Create Primary Keys - SQL Server*. [online] Available at: <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-primary-keys?view=sql-server-2017> [Accessed 3 Feb. 2019].

Docs.microsoft.com. (2019). *Delete Tables (Database Engine) - SQL Server*. [online] Available at: <https://docs.microsoft.com/en-us/sql/relational-databases/tables/delete-tables-database-engine?view=sql-server-2017> [Accessed 3 Feb. 2019].

Docs.microsoft.com. (2019). *Modify Columns (Database Engine) - SQL Server*. [online] Available at: <https://docs.microsoft.com/en-us/sql/relational-databases/tables/modify-columns-database-engine?view=sql-server-2017> [Accessed 3 Feb. 2019].

query?, H., M., D., K., R., Singraul, D., Singh, A. and kor, p. (2019). *How to change a table name using an SQL query?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/886786/how-to-change-a-table-name-using-an-sql-query> [Accessed 3 Feb. 2019].

SQL Joins Explained. (2019). *SQL Join Types — SQL Joins Explained*. [online] Available at: <http://www.sql-join.com/sql-join-types/> [Accessed 3 Feb. 2019].

Techonthenet.com. (2019). *SQL: ALTER TABLE Statement*. [online] Available at: https://www.techonthenet.com/sql/tables/alter_table.php [Accessed 3 Feb. 2019].

Techonthenet.com. (2019). *SQL: GROUP BY Clause*. [online] Available at: https://www.techonthenet.com/sql/group_by.php [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL DEFAULT Constraint*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-default.htm> [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL INDEX Constraint*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-index.htm> [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL INNER JOINS*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-inner-joins.htm> [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL LIKE Clause*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-like-clause.htm> [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL NOT NULL Constraint*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-not-null.htm> [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL TOP, LIMIT or ROWNUM Clause*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-top-clause.htm> [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL INNER JOIN Keyword*. [online] Available at: https://www.w3schools.com/sql/sql_join_inner.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL LEFT JOIN Keyword*. [online] Available at:

https://www.w3schools.com/sql/sql_join_left.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL NOT NULL Constraint*. [online] Available at: https://www.w3schools.com/sql/sql_notnull.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL NOT NULL Constraint*. [online] Available at: https://www.w3schools.com/sql/sql_notnull.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL PRIMARY KEY Constraint*. [online] Available at: https://www.w3schools.com/sql/sql_primarykey.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL RIGHT JOIN Keyword*. [online] Available at: https://www.w3schools.com/sql/sql_join_right.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL UNION Operator*. [online] Available at: https://www.w3schools.com/sql/sql_union.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL UNION Operator*. [online] Available at: https://www.w3schools.com/sql/sql_union.asp [Accessed 3 Feb. 2019].

W3schools.com. (2019). *SQL UNIQUE Constraint*. [online] Available at: https://www.w3schools.com/sql/sql_unique.asp [Accessed 3 Feb. 2019].

www.tutorialspoint.com. (2019). *SQL Primary Key*. [online] Available at: <https://www.tutorialspoint.com/sql/sql-primary-key.htm> [Accessed 3 Feb. 2019].

SQL

*The Ultimate Intermediate Guide to Learning SQL Programming
Step by Step*

INTRODUCTION

Structured Query Language (SQL) is a programming language whose major focus is on searches that you perform in the database. It deals with data on *Relational Database Management Systems*, or RDBMS. Everything that you do in the database will be determined by the data that you regulate in the system with which you are working.

This programming language was created to be used with math, such as relational calculus and algebra. The definitions that you find for the data that you use and manipulate will be done inside of SQL. You will also be able to manipulate the data through searching, removing, updating, and even inserting it. This applies to any data with which you need to work. Many people have described SQL as a language that is declarative, since you are able to use procedural elements.

Prepared by Mr. Edgar F. Codd, SQL is a language that adheres to a relational structure that is built on rules (Hosch, 2019). SQL became part of the American National Standards Institute (ANSI) in 1986, and the following year it became part of the International Organization for Standardization (ISO). Since then, improvements have been made so that the programming language can work with larger sets of data. You have to remember that SQL runs on code that is not going to be 100 percent transferable between the various databases with which you are going to be working. You are going to have to go in and change the code so that it fits the requirements that are set in place for that particular database.

When you choose to learn SQL, you will end up furthering yourself in your career. Ultimately, you are going to know how to fix things without relying on someone else to fix them for you. The more knowledge that you possess about SQL, the more qualified you will be, which opens you up to more opportunities at the best companies.

There are many ways to learn SQL and do your best to be proficient in it. Technically speaking, SQL is way more stable than Excel, and many people find it more convenient and efficient when making reports and searching for data. In addition, the geniuses with advanced SQL knowledge are in high demand by many companies. Hence, learning SQL will help you earn money – BIG money! In fact, those that work with SQL not only have unique knowledge of this programming language, they can also earn up to \$97,000 a year! This number can climb even higher depending on what level you are working at. This paycheck is double what

the average household makes in a single year. However, you need to be hired first, of course. Thus, being a professional with practical knowledge of SQL will give you a higher chance of getting hired by big companies that want to utilize your SQL skills.

CHAPTER 1: THE SIMPLE BASICS OF SQL

When you look at the structure of any business, you will see that it generates, holds, and then uses data. Because of the different ways that the company needs to handle this data, they will need to find some method of storing the information. In the traditional methods, known as the Database Management System (DBMS), firms or companies could have all of their data contained in one place to help them out.

These systems are pretty simple to use, but the evolution of modern technology has forced some changes. Even the most basic of data management systems has changed, and now they are more powerful than before. This can be an advantage to some companies that have a large amount of data to keep track of, or that need to be careful with some sensitive information.

Out of all this, a new breed of data management, called the Relational Database Management System (RDBMS), has been implemented. The derivation came from the traditional DBMS, but it has more to do with the web, as well as with server and client technologies. This basically means that these help various companies with the management of data.

One of these new relational databases that helps store data in an easy-to-use method while also keeping it all safe is SQL. This guidebook is going to take the time to explore SQL and help you gain the knowledge to work for any size business that you choose.

What Is SQL?

The first question that you may have is what exactly is SQL (“Introduction to SQL”)? SQL (Structured Query Language) is the basic language that is used in order to interact with different databases. The initial version of SQL was developed in the 1970s by IBM. It continued to progress until 1979, when Relational Software, Inc. released a new prototype and published one of the first SQL tools in the world. This tool was called Oracle, and it gained so much success that the company was renamed the Oracle Corporation after its flagship product. Even today, Oracle Corporation is one of the leaders thanks to its use of the SQL programming language.

Working on the Databases

Within this language, databases are groups of information. Some will consider these databases as organized mechanisms that are able to store information, so that the user is able to access the data effectively and efficiently to avoid issues.

You often use databases without even realizing it. For example, a phone book can be considered a database because it contains a lot of information about people, such as physical addresses, phone numbers, and names, all in one place. In addition, it is in alphabetical order to make it easier for everyone to easily find the information, or the pieces of data, that you need.

Relational Database

A relational database is segregated into logical units or tables. The tables can then be interconnected inside of your database to make the most sense for what you are working on. These databases can enable you to break up data into smaller units. This makes it easier to maintain the database and optimize it for all of the different uses in your organization.

The relational database is important because it helps keep everything together, but splits it up enough so that the pieces you deal with are a bit smaller. The server goes through all of the parts in order to find what you need because it is easier to go through smaller pieces than bigger ones. This is why so many systems have chosen to move over to a relational database for their customers.

Client and Server Technology

For some time, most of the computers that were being used in the industry were mainframe computers. These are machines that hold a robust and large system that is great at processing and storing information. The users interact with the mainframe computers using what is known as a “dumb terminal,” a terminal that doesn’t think on its own. In order to perform the right functions, the dumb terminals rely on the processor, memory, and storage inside of the computer.

There is not necessarily anything wrong with using this kind of setup, and there are still quite a few companies that use this option to run their businesses. It is effective and helps them get the information in the right place. However, there is a

better solution that is faster and will do the job even better.

The client/server systems use a different method to get the results. The main computer, or the server, can be accessed by the user through a network; you usually have either a LAN or a WAN network that helps you access the network. This way, users have the ability to access the server with their desktop computer or another server, rather than having to go through the dumb terminals. Every computer, which is known as the client in this system, has access to this system, which enables interaction between the server and any clients.

The big difference that you find between the client/server environments and the mainframe is that the former allows the user's computer to think on its own. The computer will be able to run its own applications rather than having to rely on the server all the time. Because of these great features, many businesses are switching over to the client/server environment to make things easier.

Internet-Based Database Systems

For the most part, the different database systems are moving towards the idea of integrating with the Internet. The users can access this database through the Internet, which means that the users are able to check out the database of the company simply by being on their own web browsers. Customers, or anyone who is using the data, can make changes to their accounts, check the status of any transactions, check inventories, pay online, and purchase items. All of this can be done from their own web browsers, which makes the transactions smoother.

In order to see these databases, you just need to go on a web browser of your choice, go to the company's website, log in to your account if required, and then search for any of the information that you need. Many of the businesses that you encounter require their customers to create an account with them before being able to go through the different steps. This is mostly for security reasons, like protecting customer payment information and more. As a customer, you can usually create your account for free, but it can help to protect your information.

Of course, there are a lot of things that happen behind the scenes when the customer gets onto a database that is Internet-based. While the customer will be able to access some basic information, such as their payment information and what they have ordered, there are a lot of things that the server has to put together in order to make this information correctly appear on the screen.

For example, you may find that the web browser will go through and execute SQL in order to find the data that the user is requesting. The SQL will be utilized to reach the database that the customer has put in place, such as a list of clothing or food that they are trying to sell, and then the SQL will give this information back to the website server before relaying that data to the web browser of the user.

This all takes some time to work together, even though it often takes just a few seconds for it show up on the screen. All of the things that were listed above have to occur simply for a search result to come back, or for you to be able to look through your payment information. You may see a simple expression come up on the screen after you do a quick search, but there is so much that is going to go on in the background.

SQL helps you make all of this possible. It is the best way for you to store your information and ensure that you get the right information to the user when they need it the most. We will discuss some of the ways that this is possible when you decide to create a database online and choose to use SQL to get it all in place for the user.

CHAPTER 2: INSTALLING AND CONFIGURING MYSQL

While almost all of the SQL queries presented here are general, it will eventually be easy for you to adjust to whatever type of SQL the server may use.

Before you can perform any SQL task on your computer, you first have to download SQL software.

Many options are available. You can utilize the free MySQL databases software. Hence, we will be focusing on how to download this application (“Chapter 5 Installing MySQL on Microsoft Windows”).

What Is MySQL?

MySQL is a tool (database server) that uses SQL syntax to manage databases. It is a Relational Database Management System (RDBMS) that you can use to facilitate the manipulation of your databases.

In the case where you are managing a website using MySQL, ascertain that the host of your website supports MySQL, too.

Here's how you can install MySQL on Microsoft Windows. We will be demonstrating with Windows because it is the most common operating system used on computers.

How to Install MySQL on Microsoft Windows on Your Computer

Step 1 – Visit the MySQL Website

Go to <https://dev.mysql.com/downloads/installer/> and browse through the applications to select MySQL. Make sure that you obtain the MySQL from its genuine website to prevent downloading viruses, which can be harmful to your computer.

Step 2 – Select the Download Option

Next, click on the Download option. This will bring you to the MySQL Community Server, and to the MySQL Community Edition. Click Download.



Step 3 – Choose Your Windows Processor Version

Choose your Windows processor version by reading through the details given on the page. Choose from the Other Downloads label. You can choose either the 32-bit or 64-bit.

Click the Download button for the Windows (x86, 32-bit), ZIP Archive or the Windows (x86, 64-bit), ZIP Archive, whichever is applicable to your computer.



Step 4 – Register on the Site

Before you can download your selected version, you will need to register by answering the sign in form for an Oracle account.

There is no need to reply to the questions that are optional. You can also click on the No Thanks button.

There is another option of just downloading the server without signing up, but you will not get to enjoy some freebies like the ability to download some white papers and technical information, faster access to MySQL downloads, and other services.

Step 5 – Sign in to Your MySQL Account

After registering, you can now sign in to your new account. A new page will appear where you can select your area through the displayed images of flags. Afterwards, you can click the download button and save it in your computer.

This could take several minutes.

Step 6 – Name the Downloaded File

After downloading the MySQL file, you can name it and save it on your Desktop or your C: drive, whichever you prefer.

Step 7 – Install Your MySQL Server

Click the file to open it and then press Install to install MySQL on your computer. This will open a small window where your computer will ask if you want to open and install the program. Just click the OK button.

Step 8 – Browse Your MySQL packages

The MySQL Enterprise Server page will appear, giving you some information about what your MySQL package contains.

There are packages offered for a small fee, but since we're just interested in the community server, just click Next until you reach the Finish button.

Step 9 – Uncheck the Box Register the MySQL Server Now

On set-up completion by the Wizard, a box that asks you to configure and register

your MySQL server will appear. Uncheck the Register the MySQL Server Now box, and check the small box that says Configure the MySQL Server Now.

Then, click Finish.

Step 10 – Click Next on the Configuration Wizard Box

A new box will appear, and you just have to click Next.

Step 11 – Select the Type of Configuration

A box that asks you to select your configuration type will appear. Tick the small circle for the Detailed Configuration. Click the Next button.

Step 12 – Select the Server Type

You will encounter three choices: The Developer Machine, the Server Machine, and the Dedicated MySQL Server Machine.

Select the Server Machine because it has medium memory usage, which is ideal for someone who is interested in learning more about MySQL.

The Developer Machine uses minimal memory and may not allow you to maximize the usage of your MySQL.

On the other hand, the Dedicated MySQL Server Machine is for people who work as database programmers, or who are full-time MySQL users. It uses all of the available memory in your computer, so we don't recommend it here.

Step 13 – Select the Database Usage

For database usage, there are three choices: Multifunctional Database, Transactional Database Only, and Non-Transactional Database Only. Choose the Multifunctional Database because you are using this for more general purposes.

The Transactional and Non-Transactional are used for more specific purposes.

Click Next at the bottom of the display box.

Step 14 – Choose the Drive Where the InnoDB Datafile Will Be Located

You may want to consider avoiding the default settings. Instead, you can select

which drive from your computer you want to store your InnoDB datafile on. Choose the drive you prefer and then click Next.

Step 15 – Set the Number of Concurrent Connections to the Server

This will indicate the number of users that will be connecting simultaneously to your server. The choices are: Discussion Support (DSS)/OLAP, Online Transaction Processing (OLTP), and Manual Setting.

You should select DSS/OLAP since there is no need for a high number of concurrent connections. OLTP is necessary for highly loaded servers, while the Manual Setting can be a hassle to set more than once.

After setting this, click Next.

Step 16 – Set the Networking Options

Enable TCP/IP Networking by checking off the small box before it. Below it, add your port number and then check the small box labeled Enable Strict Mode to set the server's SQL mode.

Click Next.

Step 17 – Select Your Default Character Set

Select the Standard Character Set since it is the default for English, and works well with other related Western European languages.

The other two choices, Manual Default Character Set and Best Support for Multilingualism, are best for those who speak a language other than English.

Tick the small circle before the Standard Character Set and click Next.

Step 18 – Set the Windows Options

Tick the two choices displayed, which are Install as Windows Server, and Include Bin Directory in Windows Path. These will allow you to work with your MySQL from your command line.

Selecting Install as Windows Server will automatically display the Service

Name. The small box below the Service Name must be checked, too.

Click Next.

Step 19 – Set the Security Options

Set your password. The box will indicate where you can type it.

Click Next.

Step 20 – Execute Your Configurations

All you have to do is click Execute and your computer will configure itself based on your specifications.

Once configuration is complete and all the boxes are checked, click Finish.

Step 21 – Set Verification Process

Go to the Start Menu, type cmd, and hit Enter. This will take you to the command panel.

Type the following and hit Enter:

```
mysql -u root -p
```

Notice that there is a space between MySQL and the dash symbol, between u and root, and between root and the dash symbol.

The command panel will ask for your password. Type your password and hit Enter.

A MySQL prompt will appear. You can type any SQL command to display the databases. Remember to add the semicolon at the end of your SQL statement. Close your command panel for now.

Using your MySQL can motivate you to learn more about other related applications, such as PHP and similar products.

However, it is important that you learn the basics of SQL first.

CHAPTER 3: THE SQL SERVER

Here is a guide to install and setup the Oracle Database 11g Express Edition release 2, and SQL Developer version 4 (“Creating and Configuring an Oracle 11g Database”). Both are available for download on the Oracle website for free.

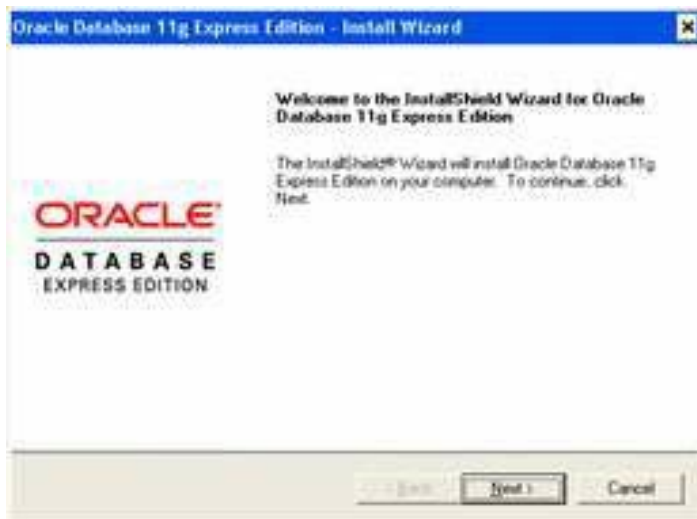
Installing Oracle Database 11g Express Edition

Go to <http://www.oracle.com/technetwork/indexes/downloads/index.html>

Locate and download the Windows version of the Oracle Database Express Edition (XE). You should consent to the License Agreement. If you don't already have an account, create one; it's free.

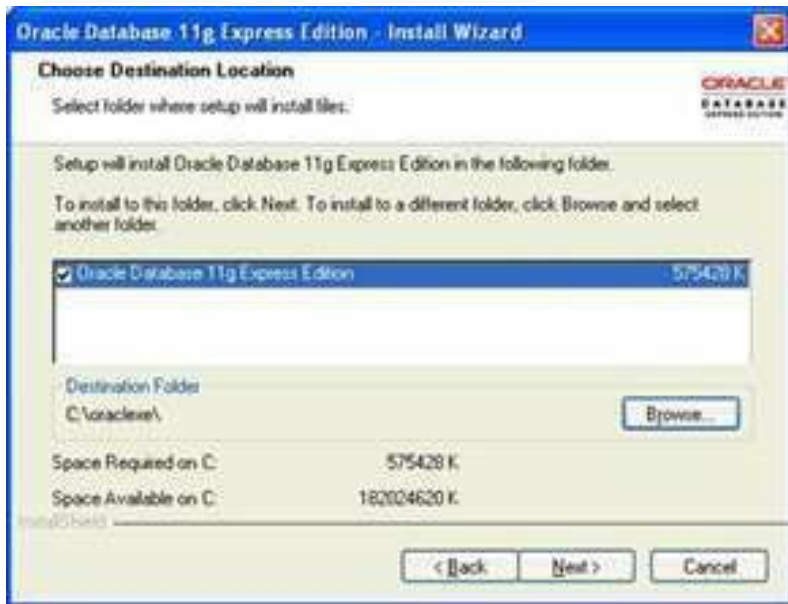
Unzip the downloaded file to a folder on your local drive, then double-click the **setup.exe** file.

You will see the Welcome window.



Click Next, accept the agreement on the License Agreement window, and then click Next again.

You will then see the Choose Destination Location window.



Accept the destination folder shown, or click the Browse button to choose a different folder for your installation, then click Next.

On the prompt for port numbers, accept the defaults, then click Next.

At the Passwords window, enter a password of your choice and confirm it, then click Next. The SYS and SYSTEM accounts created during this installation are for the database operation and administration, respectively. Make note of the password; you will use the SYSTEM account and its password for creating your own account, which you use for trying the examples.

Oracle Database 11g Express Edition - Install Wizard

Specify Database Passwords

ORACLE
DATABASE
EXPRESS EDITION

Enter and confirm passwords for the database. This password will be used for both the SYS and the SYSTEM database accounts.

Enter Password

Confirm Password

InstallShield

< Back Next > Cancel

The Summary window will be displayed. Click Install.

Oracle Database 11g Express Edition - Install Wizard

Summary

Review settings before proceeding with the installation.

ORACLE
DATABASE
EXPRESS EDITION

Current Installation Settings:

Destination Folder: C:\oraclexe\
Oracle Home: C:\oraclexe\app\oracle\product\11.2.0\server\
Oracle Base: C:\oraclexe\
Port for 'Oracle Database Listener': 1521
Port for 'Oracle Services for Microsoft Transaction Server': 2030
Port for 'Oracle HTTP Listener': 8080

InstallShield

< Back **Install** Cancel

Finally, when the Installation Completion window appears, click the Finish button.

Your Oracle Database XE is now installed.

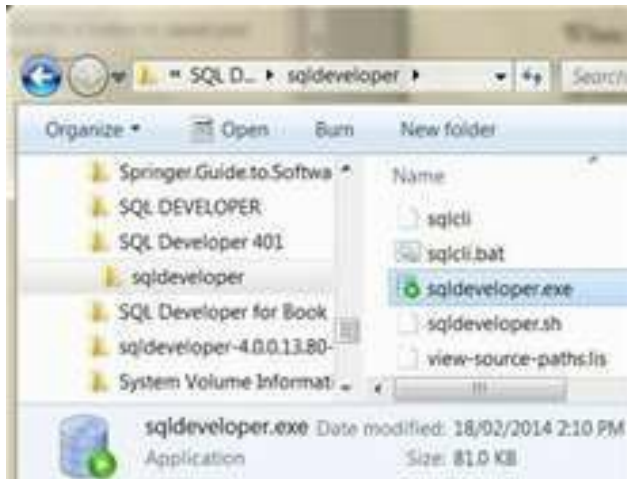
Installing SQL Developer

Go to <http://www.oracle.com/technetwork/indexes/downloads/index.html>

Locate and download the SQL Developer. You will need to consent to the License Agreement. If you don't have one, create an account; it's free.

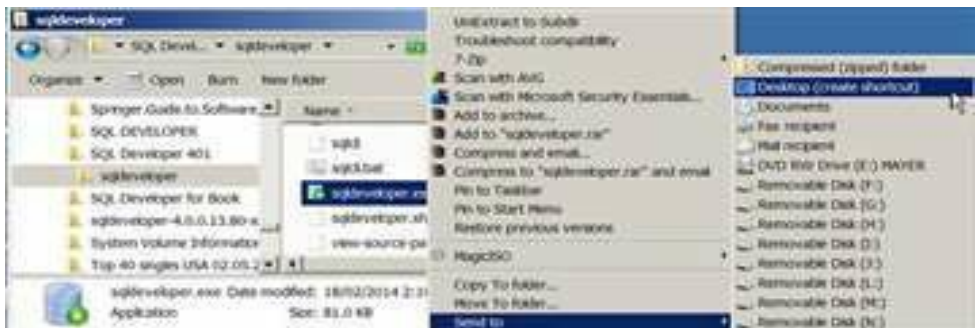
Unzip the downloaded file to a folder of your choice. Note the folder name and its location; you will need to know them to start your SQL Developer.

Once unzipped, locate the **sqldeveloper.exe** file and open (double-click) it.



SQL Developer will start once you open this file.

You can create a Desktop shortcut if you wish.



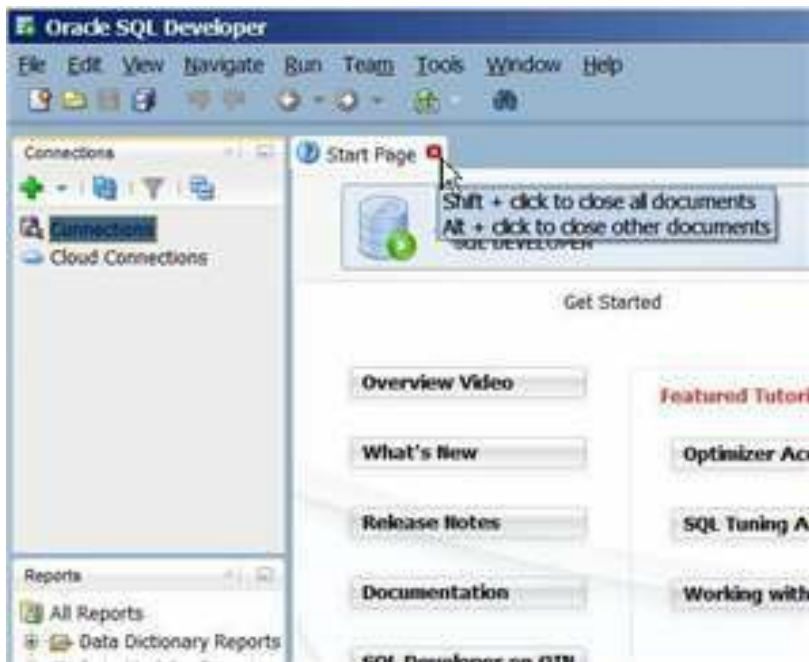
You can then start your SQL Developer by double-clicking the shortcut.



Your initial screen should look like the following. If you don't want to see the Start Page tab the next time you start SQL Developer, uncheck the Show on Startup box on the bottom left of the screen.



For now, close out of the Start Page tab by clicking the red X.

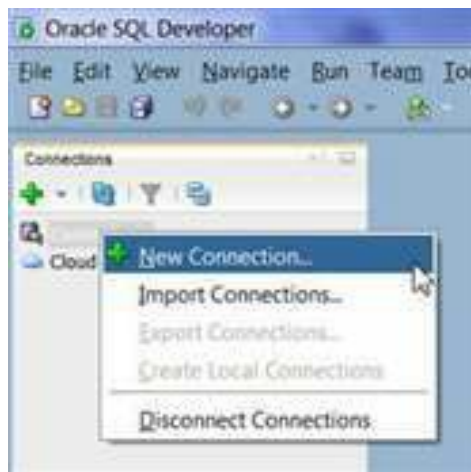


Creating a System Connection

To work with a database from SQL Developer, you need to have a connection (Otey, 2015).

A connection is specific to an account. We will use the SYSTEM account to create your own account, so you first have to create a connection for the SYSTEM account.

To create a connection, right-click the Connection folder.



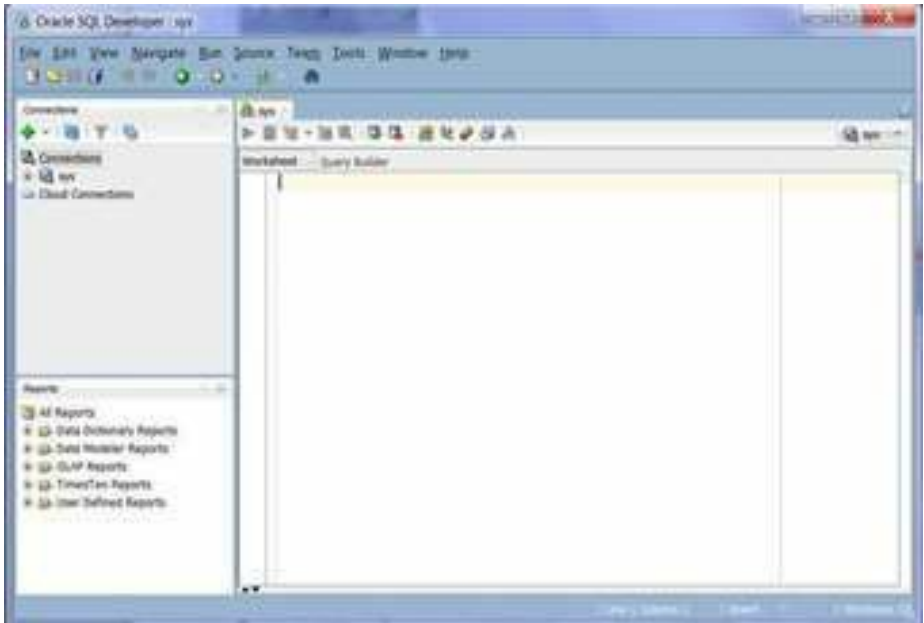
On the New/Select Database Connection window, enter a Connection Name and Username as shown. The Password is the same as that of the SYSTEM account you entered during the Oracle database installation. Check off the Save Password box.



When you click the Connect button, the *system* connection you have just created should be available on the Connection Navigator.



A Worksheet will open for the system connection. The Worksheet is where you type in source codes.

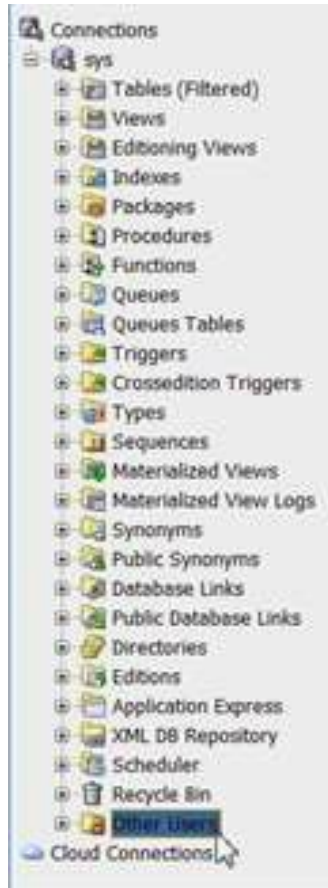


Creating a Database Account

You will use your own database account (user) to try the examples outlined in this book.

To create a new account, expand the system connection and locate the Other Users

folder at the bottom of the folder tree.



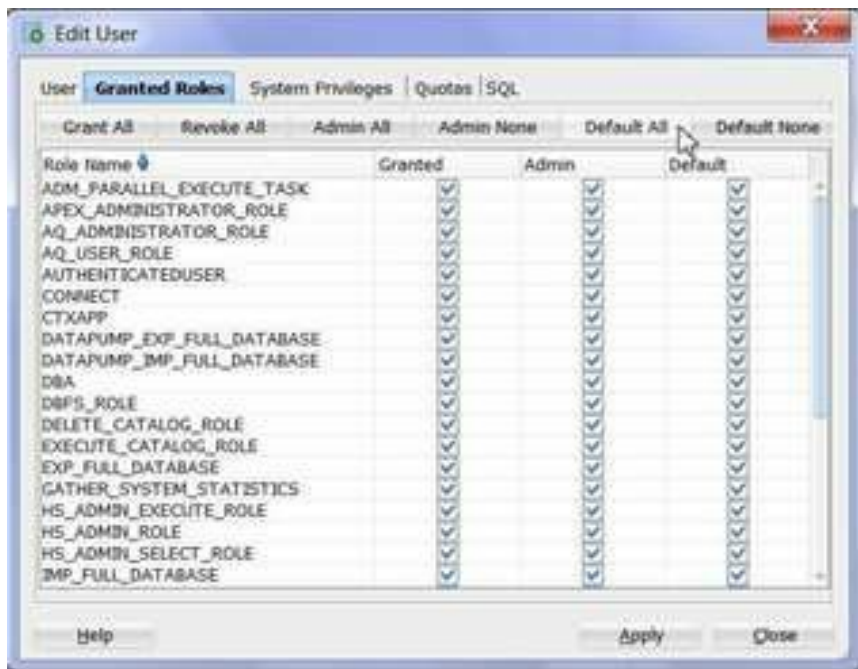
Right click and select Create User.



Enter a username and password of your choice, then click the Apply button. Close the success window that pops up afterwards.

A screenshot of the 'Create User' dialog box. The 'User' tab is selected, showing fields for 'User Name' (containing 'DJONG'), 'New Password' (masked with dots), and 'Confirm Password' (masked with dots). There are three checkboxes: 'Password Expired (user must change next login)', 'Account is Locked', and 'Edition Enabled', all of which are unchecked. Below these are two dropdown menus for 'Default Tablespace' and 'Temporary Tablespace'. At the bottom, there are buttons for 'Help', 'Apply', and 'Close'.

On the Granted Roles tab, select the Grant All, Admin All, and Default All buttons, then click Apply. Close the success window and the Edit User window, as well.

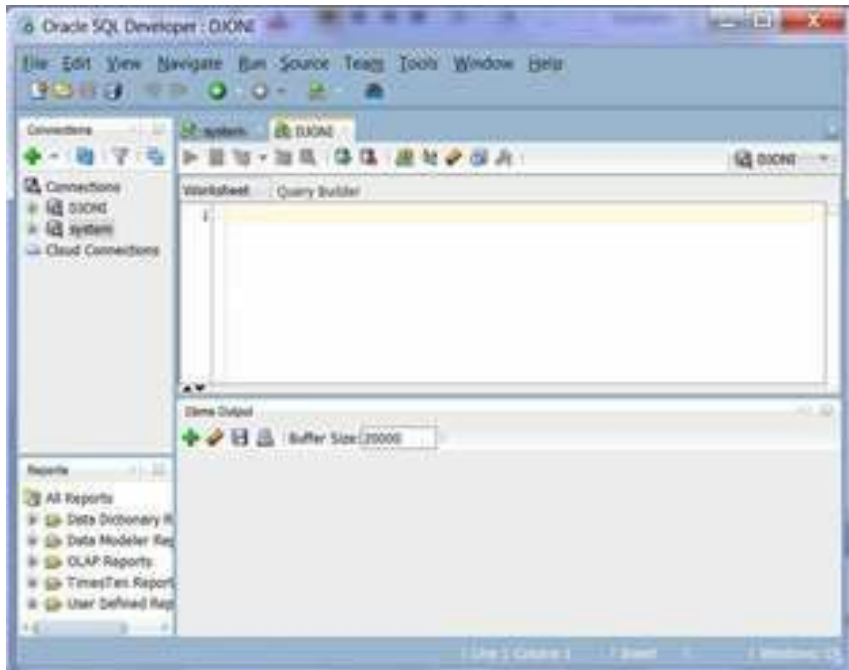


Creating Your Account Connection

Similar to when you created a system connection earlier, now you will create a connection for your account.

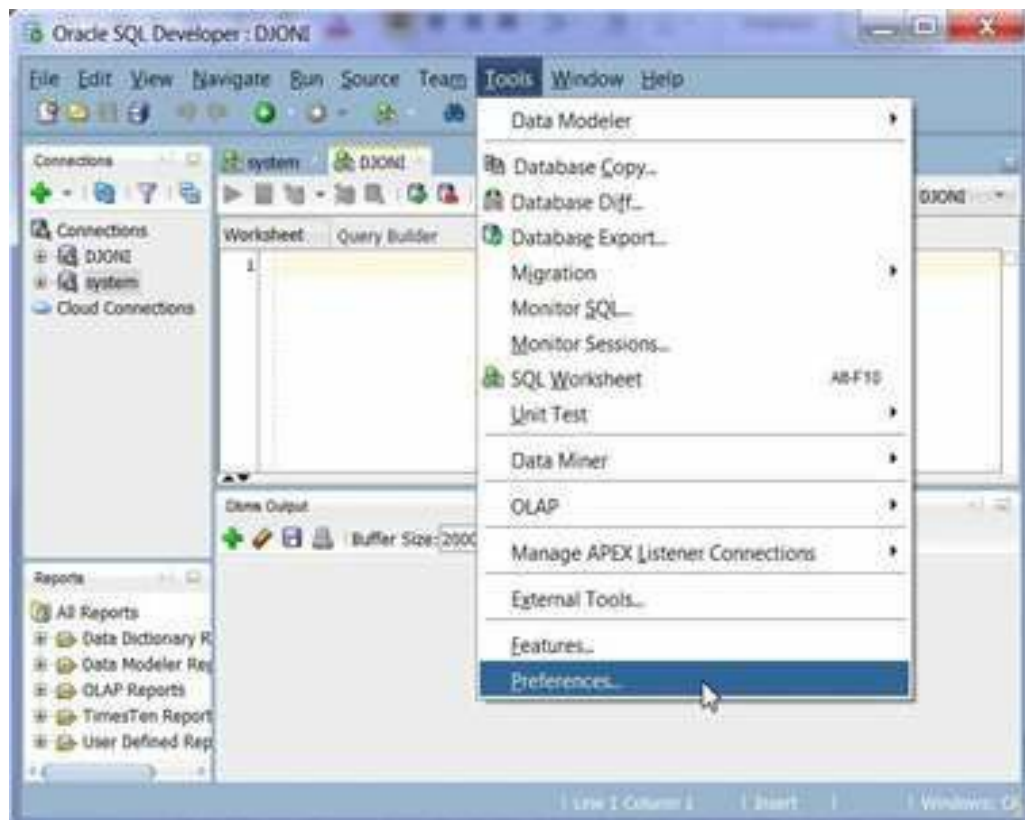


Click the Connect button. A worksheet for your connection is opened (for our example, this is labeled *DJONI*).

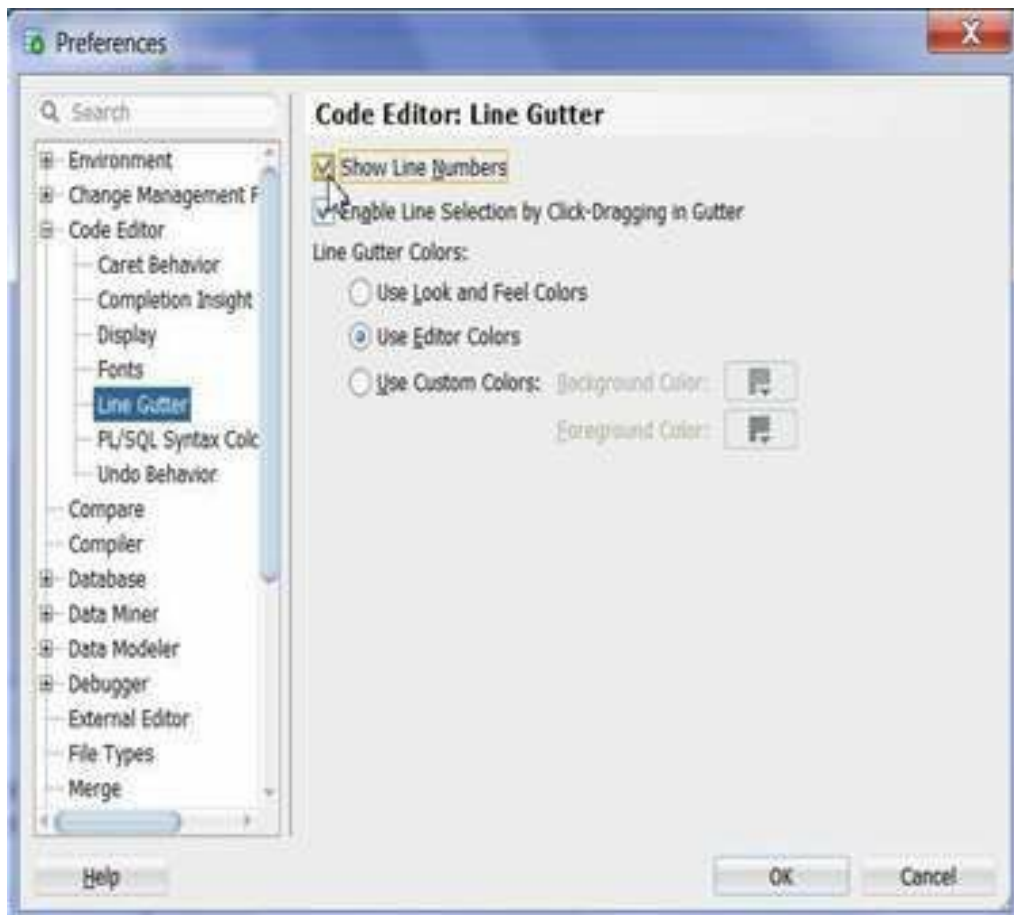


Showing Line Numbers

In describing the book examples, we sometimes refer to the line numbers of the program; these are line numbers you will locate on the worksheet. If you want to view the line numbers, click Preferences from the Tools menu.

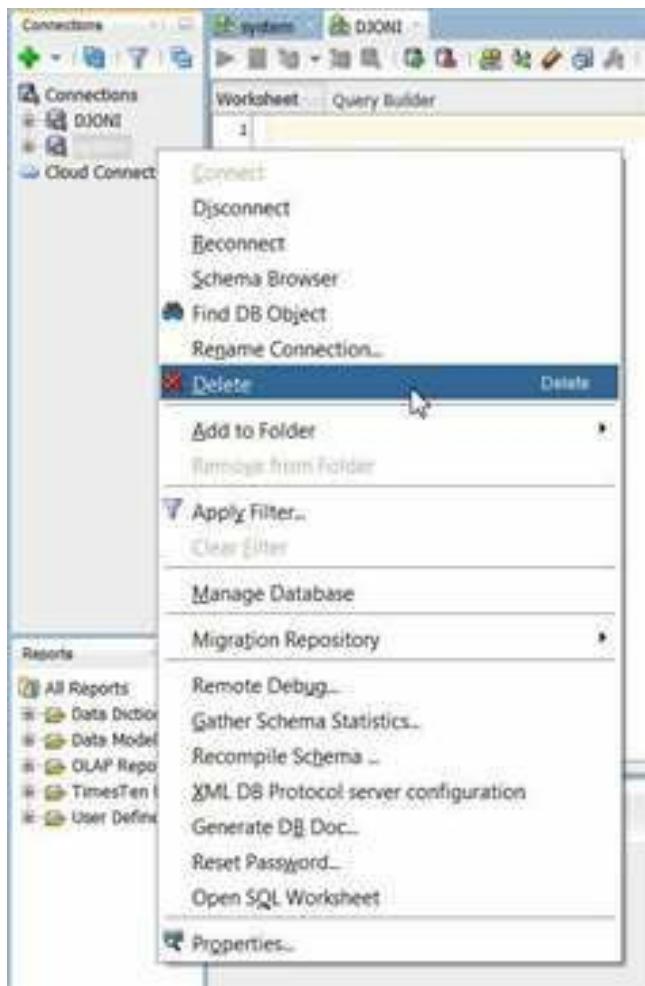


Select Line Gutter, then check off Show Line Numbers. Your Preferences need to match what is shown below. Click OK.

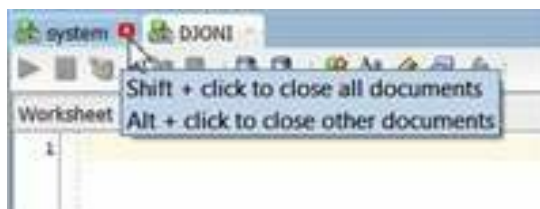


Deleting the System Connection

Delete the *system* connection, so you don't use this account mistakenly. Click Yes to confirm the deletion. Your SQL Developer is now set.



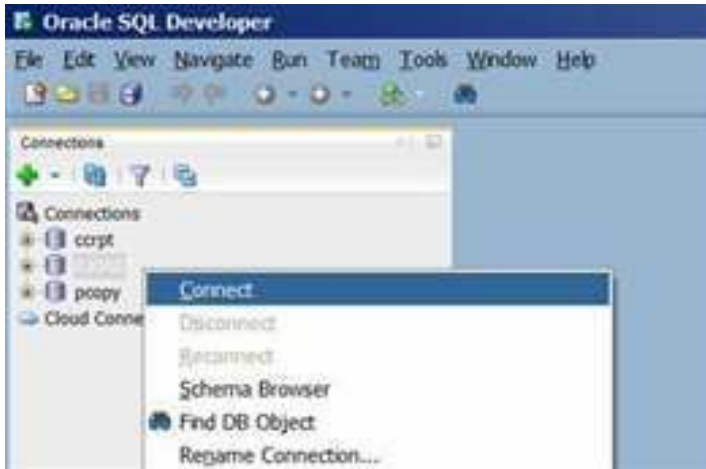
Close the *system* worksheet.



Using SQL Developer

The worksheet is where you enter SQL statement and PL/SQL source code.

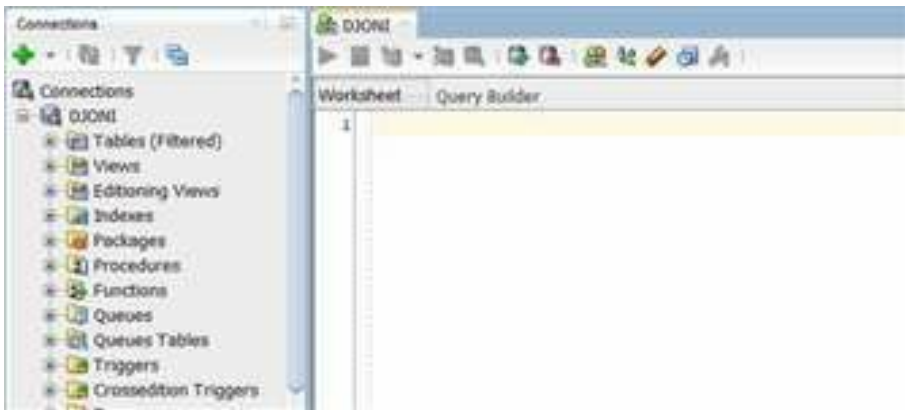
Start your SQL Developer if you have not already done so. To open a worksheet for your connection, click the + (folder expansion) or double-click the connection name. Alternatively, right-click the connection and click Connect.



Note that the worksheet's name (tab label) will be the one denoting your connection's name.

You can type source code on the worksheet.

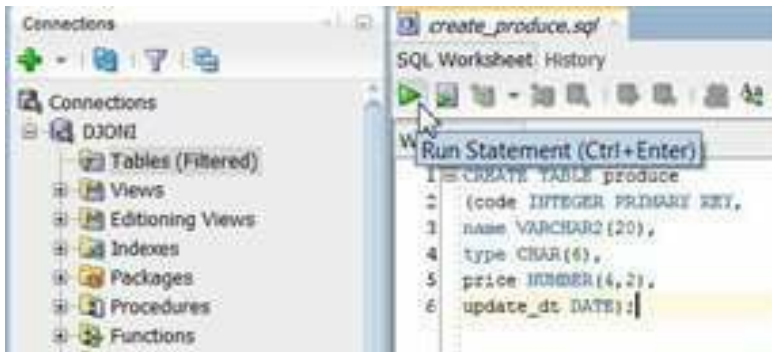
Appendix A has the source code of all the book examples. Instead of typing, you can copy a source code and paste it on the worksheet.



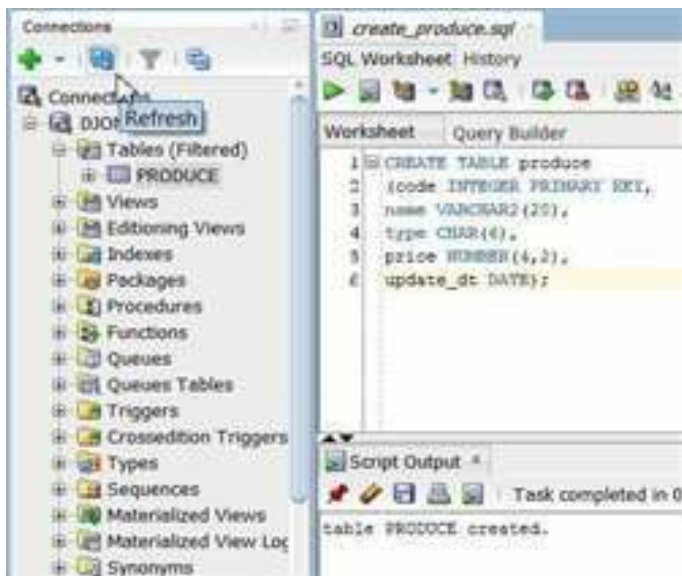
Running an SQL Statement

Some of the book examples use a table named *produce*. Type in the SQL CREATE TABLE statement shown below to create the table (you might prefer to copy the *create_produce.sql* listing from Appendix A and paste it in the open worksheet).

You run an SQL statement that is already in a worksheet by pressing the Run Statement button.



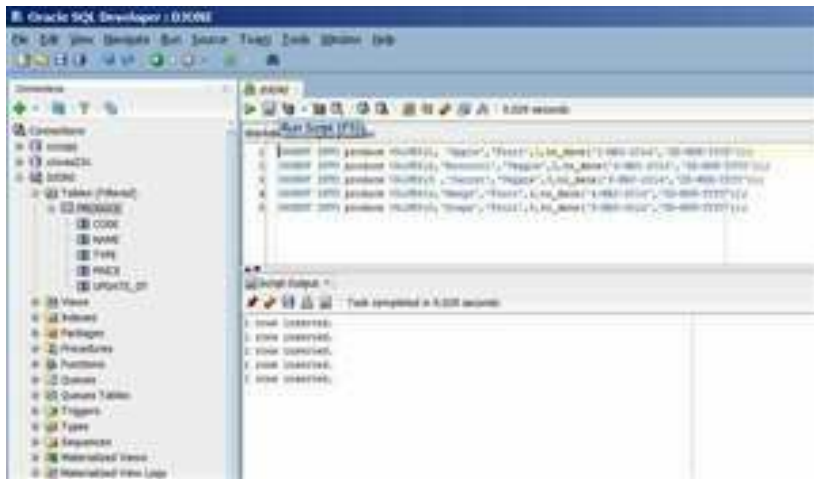
The Script Output panel confirms that the table has been created, and you should see the *produce* table in the Connection Navigator under your connection folder. If you don't see the newly created table, click Refresh.



Inserting Rows

To show an example of running multiple SQL statements in SQL Developer, the following five statements insert five rows into the *produce* table. Please type the statements, or copy them from *insert_produce.sql* in Appendix A. You will use these rows when you try the book examples.

Run all statements by clicking the Run Script button, or Ctrl + Enter (press and hold Ctrl, then click Enter button).

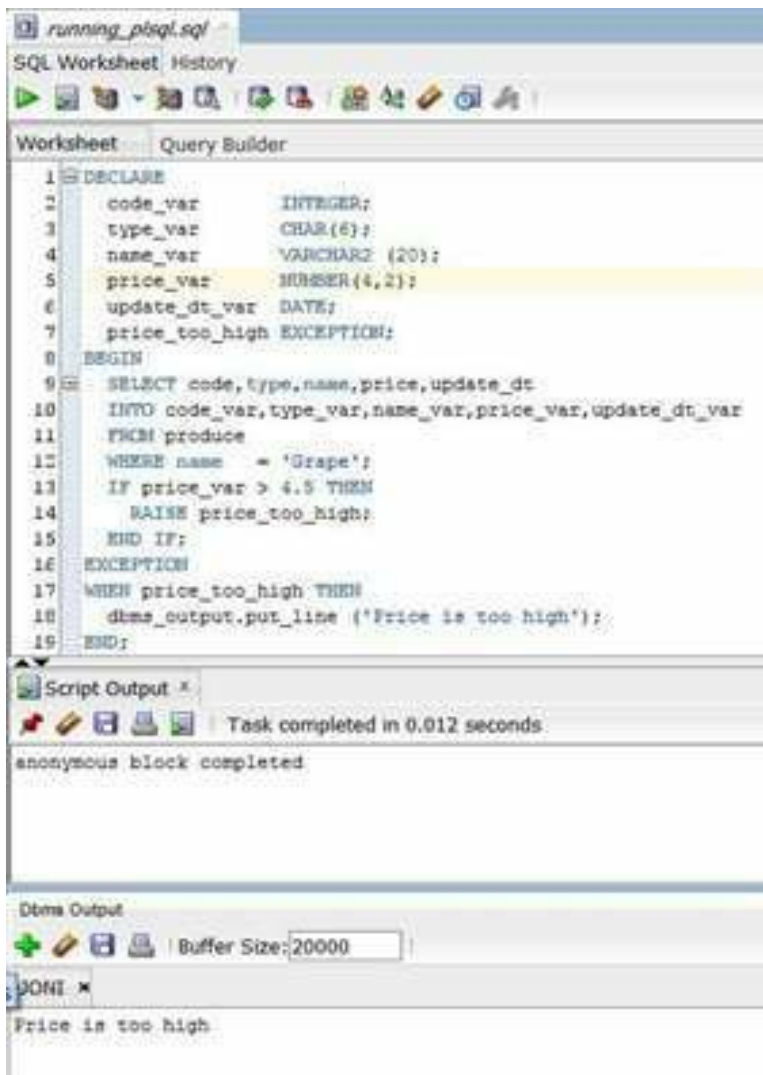


Running a PL/SQL Program

To learn how to run a PL/SQL program, type the following PL/SQL program, or copy it from *running_plsql.sql* in Appendix A.

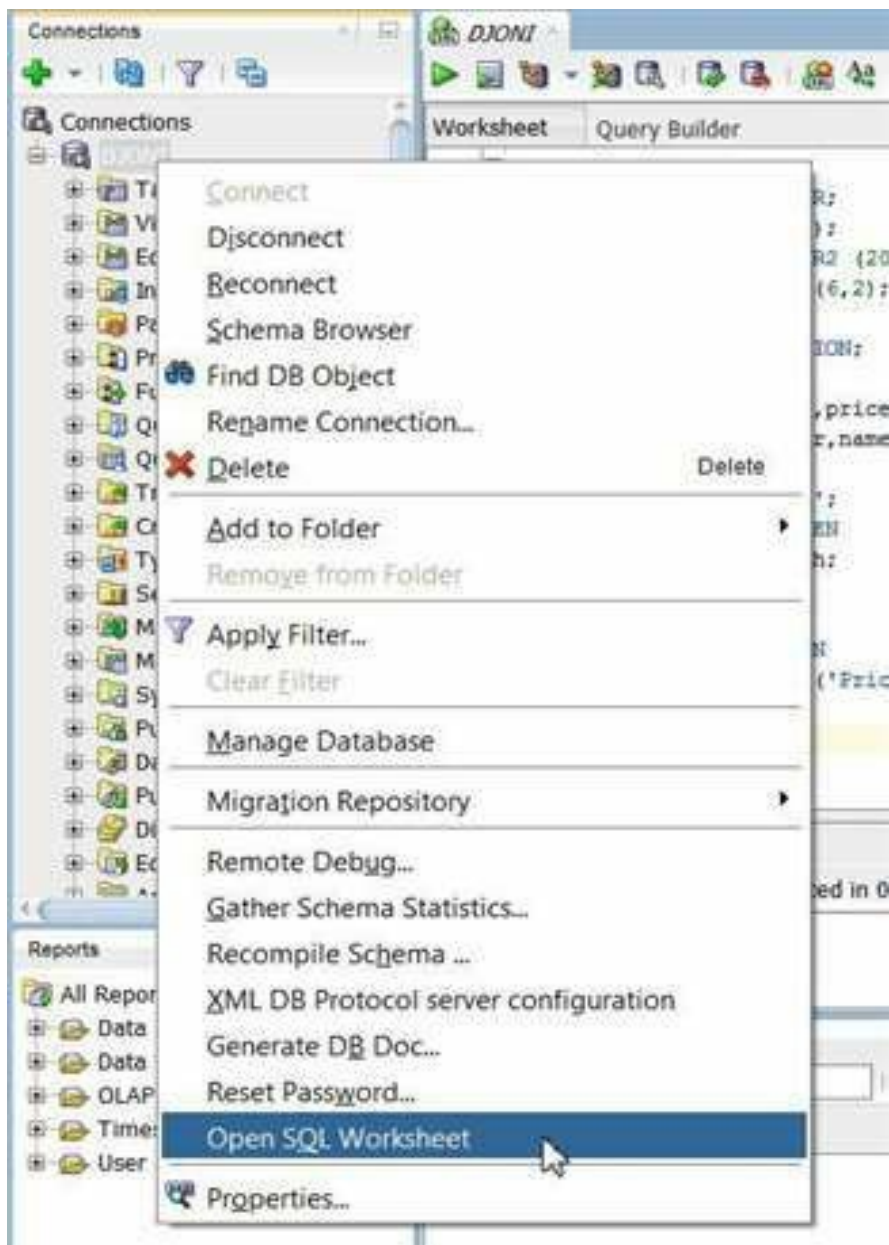
You have not learned anything about PL/SQL programming yet, so don't worry if you don't know what this program is all about.

To run the program, click the Run Script button or press F5.

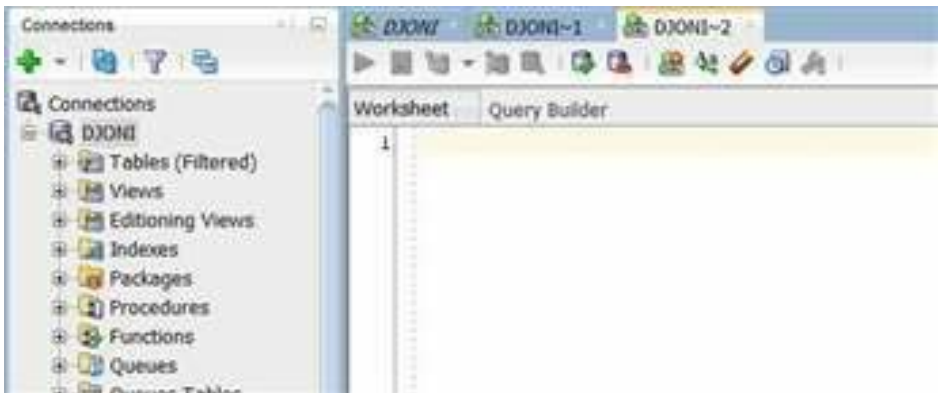


Multiple Worksheets for a Connection

Sometimes you should ensure that you have two or more programs on different worksheets. You are allowed to open multiple worksheets for a connection by right-clicking the connection and selecting Open SQL Worksheet.

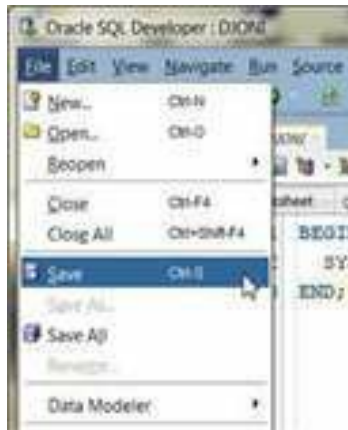


The names of the next tabs for a connection have sequential numbers added.

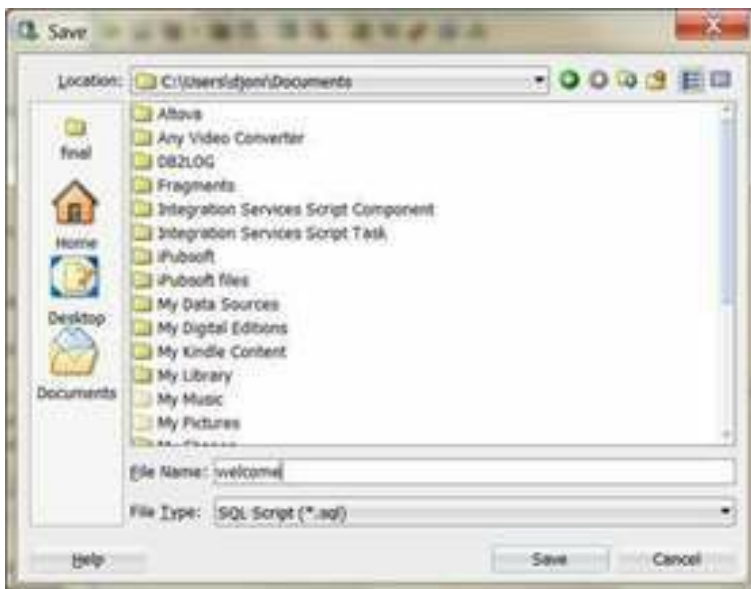


Storing the Source Code

You can store a source code into a text file to reopen later by selecting Save from the File menu.

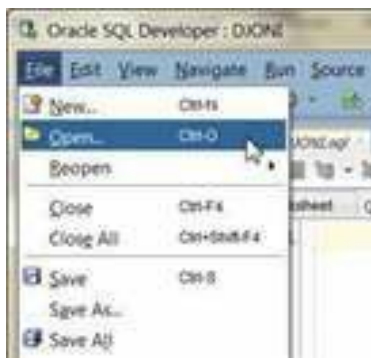


Choose the location where you wish to store the source code, give the file a name, and then select the Save option.

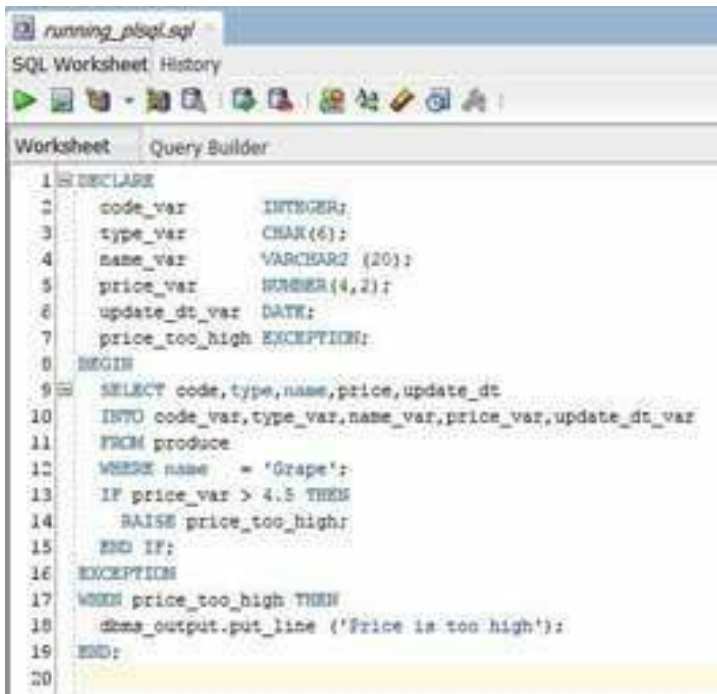


How to Open a Source Code

Open a source code by selecting Open or Reopen from the File menu and then choosing the file that contains the source code.



The source code will be opened on a new worksheet. The tab of the worksheet has the file's name. The following is the worksheet opened for the source code stored as a file named *running_plsql.sql*.



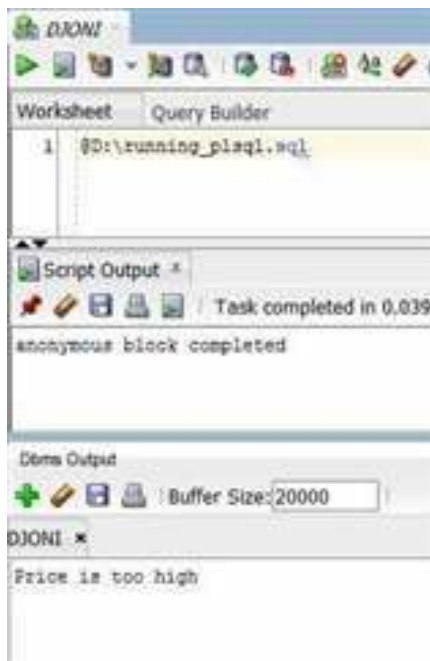
```
1 DECLARE
2     code_var      INTEGER;
3     type_var      CHAR(6);
4     name_var      VARCHAR2 (20);
5     price_var     NUMBER(4,2);
6     update_dt_var DATE;
7     price_too_high EXCEPTION;
8 BEGIN
9     SELECT code,type,name,price,update_dt
10    INTO code_var,type_var,name_var,price_var,update_dt_var
11    FROM produce
12   WHERE name = 'Grape';
13   IF price_var > 4.5 THEN
14       RAISE price_too_high;
15   END IF;
16 EXCEPTION
17 WHEN price_too_high THEN
18     dbms_output.put_line ('Price is too high');
19 END;
20
```

Storing the Listings in Appendix A in Files

As an alternative to copying and pasting, you can store each of the listings in a file and open the file. Note that you must store each program source code into a file.

Running SQL or PL/SQL from a File

You can execute a file that contains an SQL statement or a PL/SQL program without opening it on the worksheet, as shown below.



Clearing a Worksheet

To clear a Worksheet, click its Clear button.



Displaying Output

Most of the book examples use the Oracle-supplied *dbms_output.put_line* procedure to display some outputs. For readers learning PL/SQL, the displayed output gives an instant feedback of what happens in the running program. Real-life programs might not need to display any output.

The *dbms_output.put_line* procedure has the following syntax:

```
dbms_output.put_line (parameter);
```

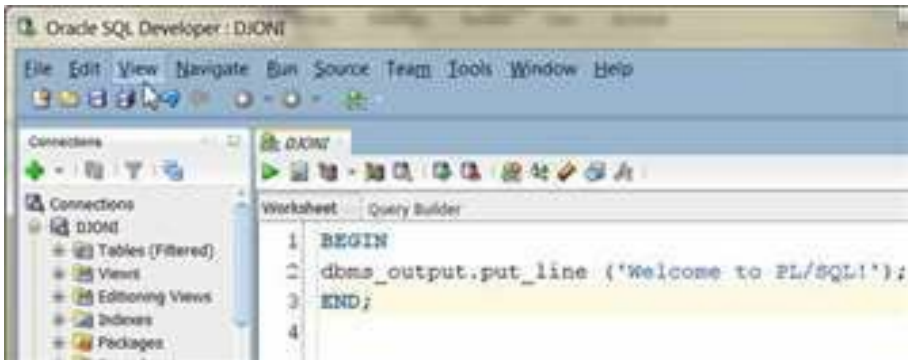
The value of the parameter must evaluate to a string literal (value).

When the procedure is executed in SQL Developer, the string literal is displayed on the DBMS_OUTPUT.

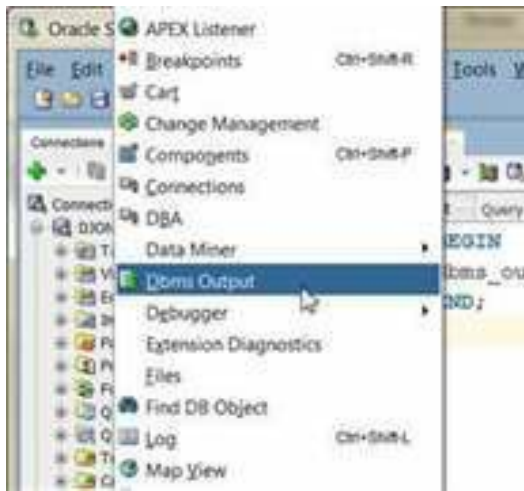
To see the output, before you run the example, make sure you already have a DBMS_OUTPUT panel opened for the connection you use to run the program. If your DBMS_OUTPUT is not ready, set it up as follows:

Assume you need to execute the program as shown here.

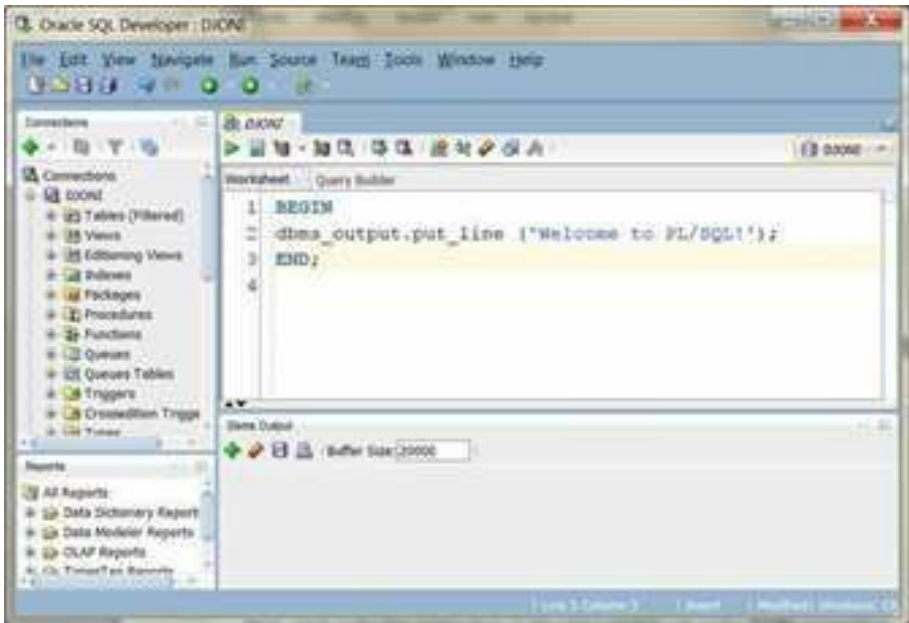
Click the View menu.



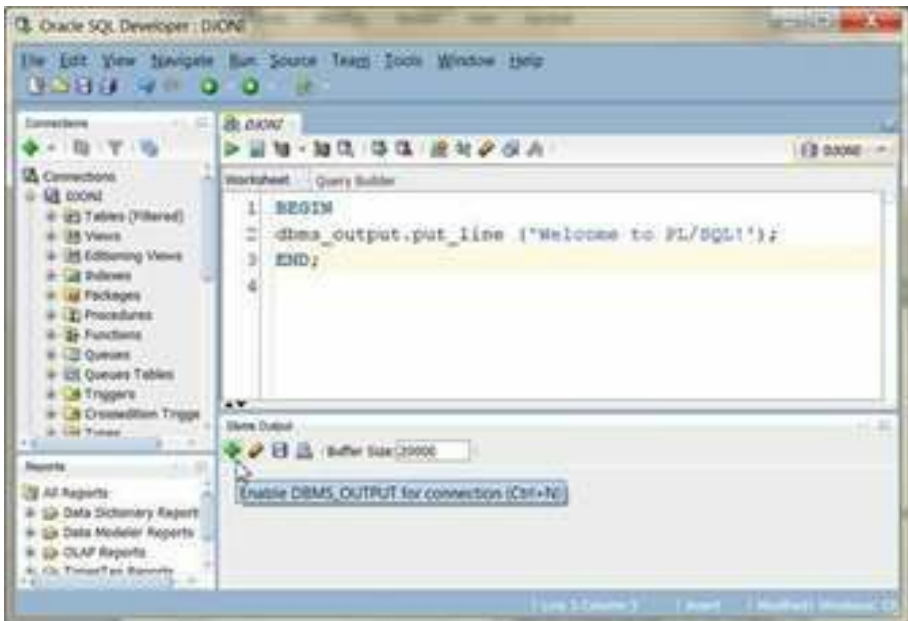
Next, select Dbms Output.



The DBMS_OUTPUT panel is now opened.



To display an output, you will set up the DBMS_OUTPUT panel for the connection you use to execute the program. Press the + button located on the DBMS_OUTPUT panel.



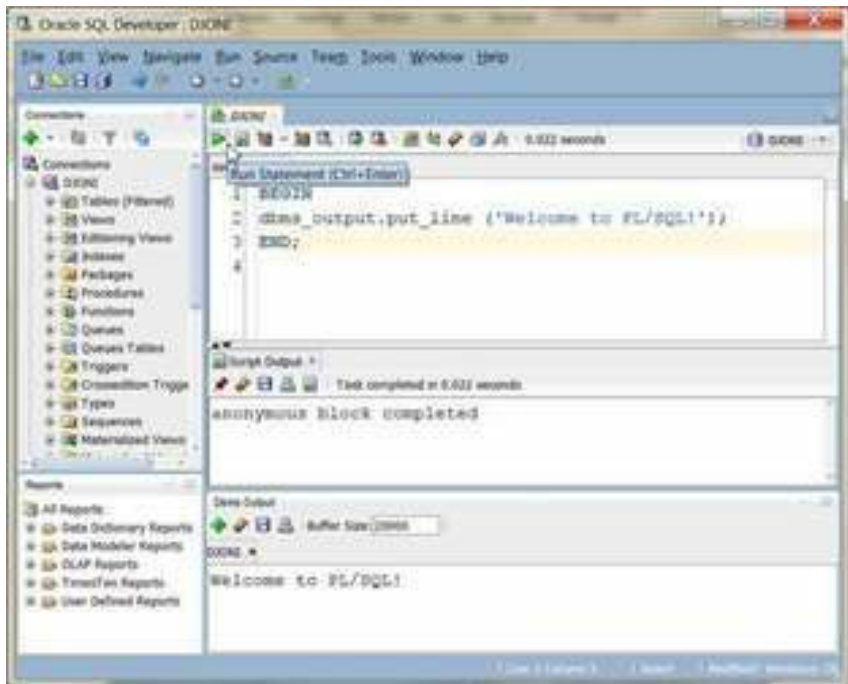
On the pop-up window, select the connection, and then press OK. As an example, we'll select the DJONI connection to run the PL/SQL program.



The DBMS_OUTPUT now has the tab for the DJONI connection.

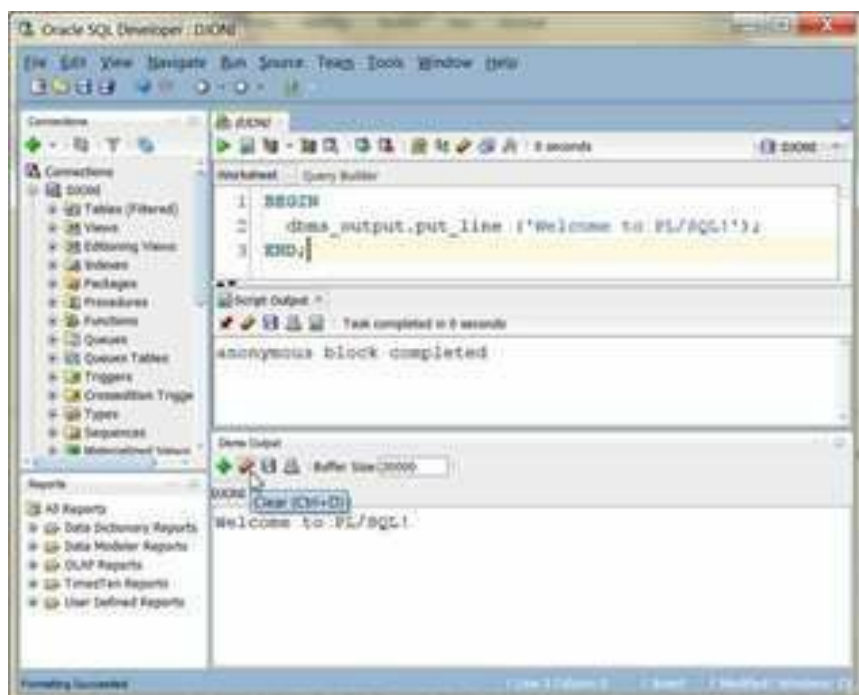


Now, run the program by pressing the Run Statement button. The DBMS_OUTPUT panel displays the “Welcome to PL/SQL!” greeting. The message on the Script Output panel shows the result of running the program; in this case it indicates that the program is completed successfully. It would show an error message if the program is having a problem.



Clearing DBMS_OUTPUT

To see a display output from a program, you might want to erase the output from a previous program. To clear a DBMS_OUTPUT, click its Clear button.



CHAPTER 4: DATA TYPES

Data types are specific for the object you are using. Regardless of what you are using, all expressions, variables, and rows have some sort of data type that operates effectively with SQL (“SQL Data Types”, 2001).

We use data types when creating tables. It is necessary that you pick one data type for that row in the table based on what you need that row to do:

- **Byte:** a set of numbers between 0 and 255. You only have a single byte of storage for these figures.
- **Singles:** a floating-point number that is associated with a decimal. Singles contain up to 4 bytes of storage.
- **Currency:** has up to 4 decimal points for currency, and can go up to 15 places in whole numbers. There are around 8 bytes of storage for currency.
- **Long:** has 4 bytes of storage and falls between -2,147,483 and 2,147,483,647.
- **Memo:** holds larger amounts and can go up to 65,536. Memos can be searched but not sorted.
- **Text:** letters and numbers that are entered SQL. It can handle up to 255 characters.
- **Date and Time:** with 8 bytes of storage, the date and time are going to be obvious.
- **Double:** just like singles, a double has two floating point numbers and contains 8 bytes of storage.
- **Integer:** you can store numbers using this data type. The numbers fall between -32,768 and 32,767. You have 2 bytes of data that can be stored here.
- **Lookup Wizard:** this is your options list so that you can choose what it is that you want to do. This has 4 bytes of storage.
- **Auto Number:** the fields are given their own record for data with its own number. It is going to begin with 1 unless you state otherwise. 4 bytes of data are saved for the auto number.
- **Hyperlink:** you can send the user to a web page if it is necessary.
- **Yes and No:** the logical fields that require truth, like yes or no, or even true or false. The true and false truth are equal to -1 and 0. If no value is

allowed, then that field is going to be null. The storage is 1 bit.

- **OLE Object:** pictures and other multimedia are placed in this data type. It is known as a blob, which stands for large binary objects. You have the most significant storage amount here, 1 gigabyte.

Categories of Data Types

Data types fall into three main categories: date and time, character, and number.

Character

- **VARCHAR(size)**: the length of the string that is stored can hold all the data types listed above. The size is based on whatever the number in the parentheses is. Normally, you convert texts.
- **CHAR(size)**: the character string is a fixed length that can be one of the data types that were previously discussed.
- **BLOB(size)**: there are 65,535 bytes of data held in a blob.
- **TINYTEXT**: you only put 255 characters in the small text.
- **LONGTEXT**: this holds the largest number of characters.
- **TEXT(size)**: this is the standard character text that holds up to 65,535 characters.
- **MEDIUMTEXT**: this holds around 16 million characters at most.
- **SET(x, y, z, ...)**: this is like an ENUM type, but it only holds up to 64 items. However, you can store more than one data type in it.
- **MEDIUMBLOB**: this holds up to 16 million bytes of data.
- **LOBLOB**: this holds the most bytes of data, hence, why it is called a long blob.
- **ENUM(x, y, z, ...)**: every possible value is going to be listed in an ENUM. It can hold up to 65,535 values. Anything that is not inside of the list will be shown as a blank value. The values do not have to be put in order when they are entered; they can be sorted later.

Number

- **FLOAT(size, d)**: a tiny number that uses a decimal that can float. The size of the parameters is the maximum amount of values that can be put into the data type. The d parameter is the largest amount of numbers that can be to the right of that decimal.
- **TINYINT(size)**: you can have a range of -128 to 127 for signed values, or 0 to 255 for unsigned values.
- **BIGINT(size)**: you can have a range of 0 to

18,446,744,073,709,551,615 for unsigned values, or between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 for signed values.

- SMALLINT(size): you can hold up to 32,767 signed values, or 65,535 unsigned.
- INT(size): holds a set value of signed and unsigned values.
- MEDIUM(size): holds up to 16 million unsigned values.
- DECIMAL(size, d): you can store decimals inside of a fixed point that is inside of the size parameters. The parameters are set by the maximum number of digits that are found to the right of the decimal.
- DOUBLE(size, d): this is similar to DECIMAL(size, d), but the number of digits that are to the right side of the decimal is set by the parameter d.

Date and Time

- DATE: dates are entered in the year, month, day format (YYYY-MM-DD).
- YEAR: the year is either inputted in two-digit form or four-digit form. How it is entered depends on what you are doing.
- DATETIME(): a combination of the date and time. The date is recorded in the same format as DATE is, YYYY-MM-DD, and then the time is entered as hours, minutes, seconds (hh:mm:ss).
- TIME(): time is entered in the hours, minutes, seconds format (hh:mm:ss).
- TIMESTAMP(): the time stamp aligns with your current time zone so that you can see exactly when something was done. The format is the same as that of DATETIME().

Arithmetic

With SQL 92, you can do five different arithmetic functions that most other artificial dialects can do (“Arithmetic Operators”).

1. Multiplication (*)
2. Addition (+)
3. Modulo (%)
4. Subtraction (-)
5. Division (/)

When you use the modulo operator, you will get the remainder that was received from the utilization of the division function, whether it is an integer or not. You are not able to use modulo when you are working with ANSI SQL because it is not supported. However, many of the other databases support the modulo function.

While the above functions are great, they are not used very often. Instead, there are other functions that you will be using more. These arithmetic operations are not required for you to use SQL 92, but that does not mean that you should not make sure that they are available on the RDBMS that you have decided to use. If they are not supported by the RDBMS that you are using, you are going to end up frustrating yourself when you put in all the effort for them to not work. If you are using a major database, they will most likely work. Just be sure to check on the off chance you have chosen to work with one that it does not operate with.

Here are a few examples of functions using these arithmetic operations:

- MOD (a, b): the modulo function will give you an integer for the remainder that you get when a is divided by b.
- SQRT (a): you get the square root of the value you enter into the equation.
- SIGN (a): a will give you a 0, -1, or 1.
- ROUND (a): the value you input is rounded to the nearest whole number.
- ABS (a): you receive the absolute value from the integer that you input into the database.
- ROUND (a, b): the integer for a is rounded to the closest whole number, and value b tells it how many times it should be rounded.

- **CEILING (a) or CEIL (a)**: the smallest number of greater than or equal value to the value of a is returned to you.
- **POWER (a, b)**: a is raised to the value that is in place for b.
- **FLOOR (a)**: the largest number of less than or equal value to the value of a is returned to you.

CHAPTER 5: SQL SCRIPT BOUNDARIES

A restrict consists of the different rules that the SQL system enforces when it is dealing with the rows on your table. Essentially, your table is going to be limited to the diverse types of data that are allowed to go into the database. Restricts are used so that you can ensure that the database has accurate and reliable data in it. This is particularly helpful when your database is being used by other people for a job.

Restricts are not limited to just rows; they can be placed on an entire table as well. The restrictions you place on a row are only going to be enforced on the row in question. However, when you put them on the table, they will be implemented on the entire table.

There are just a few limitations that you use most often when you are working with restricts (Jones, Plew, & Stephens, 2008).

- **Not NULL:** the row cannot have any null values.
- **Index:** the index restricts will not only construct, but will also retrieve data that you have in your database.
- **Default Restrict:** if you do not set a default value, then one will be provided for that row.
- **Check Restrict:** this limitation will make sure that any conditions placed on the row are met.
- **Unique Restrict:** there will not be any values in your row that are the same; each value is going to be unique.
- **Foreign Key:** records in other database tables are going to be identified with the key.
- **Primary Key:** in the table that you are currently working in, each record is going to be identified.

When you are creating your table, you will have the opportunity to put your restricts into the script so that they are done from the beginning. If you forget to do that, you can use the modify table statement so that the restricts can be constructed even though your table is already in place on your database.

Dropping Restricts

Even though your restrict has been put into place, you have the choice to drop the restrict by using the command that you use to modify your table. From there, you will choose the drop restrict option.

Depending on the restrict, you will be provided with a shortcut that you can use to drop that restrict. This will also depend on which application you use. As mentioned at the beginning of this book, not every command is going to work in the programs that you use as you construct your SQL script.

Along with shortcuts, some implementations let you disable restricts. This is not going to remove the restriction from the database permanently, but, instead, disable it so that it can be used in that database later when you need it.

Integrity Restricts

Integrity restricts are used when you want to be certain that your data is accurate and consistent throughout the entire database. The integrity of your data is going to be checked against a different database than what you are using thanks to the concept that has been put into SQL for referential integrity. Besides, how are you going to know if your data has integrity if you lack the appropriate mechanisms to check it?

While multiple integrity restricts can be used when using referential integrity, you are primarily going to use the foreign key and primary key, along with the other restricts.

People believe that their information is protected if you can control who can view, create, modify, or delete data. It is true that your database is protected from most threats, but a hacker can still access confidential information using some indirect methods.

A database has referential integrity if it is designed correctly. This means that the data in one table will always be consistent with the data in another table. Database developers and designers always apply constraints to tables that restrict what data can be entered in the database. If you use databases with referential integrity, users can create new tables that use the foreign key in a confidential table. This will allow them to source information from that table. This column will then serve as the link through which anybody can access confidential information.

Let us assume that you are a Wall Street stock analyst, and many people trust your assessment about which stock will give them the best returns. When you recommend a stock, people will always buy it, and this increases the value of the stock. You maintain a database called `FOUR_WAYS` that has the information and all your analyses. The top recommendations are in your newsletter, and you restrict user access to this table. You will identify a way to ensure that only your subscribers can access this information.

You are vulnerable when anyone other than you creates a table that uses the same name for the stock field as the foreign key. Let us look at the following example.

```
CREATE TABLE HOT_STOCKS
```

```
(  
Stock CHARACTER (30) REFERENCES FOUR_WAYS  
);
```

The hacker can insert the name of the stock in the New York Stock Exchange, NASDAQ, and American Stock Exchange into that table. These inserts will tell the hacker which stocks he or she entered that match the stock that is against your name. It will not take the hacker too long to extract the list of stocks that you own.

You can protect the database from such hacks by using the following statement:

```
GRANT REFERENCES (Stock) ON FOUR_WAYS TO SECRET_HACKER;
```

You should never grant people privileges if you know that they will misuse them. People never come with a trust certificate, and you would never lend your car to someone you do not trust, for example. The same can be said about giving someone REFERENCE privileges to an important database.

The previous example explains why it is important that you maintain control of the REFERENCES privilege. Here are some explanations on why you should carefully use REFERENCES:

- If another user were to specify a constraint in the HOT STOCKS by using the RESTRICT option, the DBMS would not allow you to delete a row from that table. This is because the referential constraint is being violated.
- The first person will need to drop or remove the constraints on a table if you want to utilize the DROP command to do away with your table.

Simply put, it is never a good idea to let someone else define the constraints for your database, since this will introduce a security breach. This also means that the user may sometimes get in your way.

Delegating Responsibility

To maintain a system that is secure, you should restrict the access privilege that you grant to different users. You should also decide which users can access the data. Some people will need to access the data in the database to carry on with their work. If you do not give them the necessary access, they will constantly badger you and ask you to give them that information. Therefore, you need to know how your database security will be maintained. You can use the WITH GRANT OPTION clause to manage database security. Let us consider the following examples:

```
GRANT UPDATE
```

```
ON RETAIL_PRICE_LIST
```

```
TO SALES_MANAGER WITH GRANT OPTION
```

The statement is similar to the GRANT UPDATE statement that allows the sales manager to update the retail price list. This statement also gives the manager the right to grant any update privileges to people he or she trusts. If you use this version of the GRANT statement, you should trust the fact that the grantee will use the privilege wisely. You should also trust the fact that the grantee will grant the privilege to only the necessary people.

```
GRANT ALL PRIVILEGES
```

```
ON FOUR_WAYS
```

```
TO BENEDICT_RENALD WITH GRANT OPTION;
```

If you misspell the name or give the wrong person access, you cannot guarantee the security of your database.

CHAPTER 6: FILTERS

In the previous book, we explained how to use the WHERE clause. In this book, we will go more in-depth on its practical use alongside other operators in filtering (“Filtering”).

Filtering is a very important aspect of SQL. When working on a particular section of a table, or retrieving some particular detail, it is important to use filters. It is because of this reason that, in SQL, all statements except the INSERT statement can use a WHERE clause to filter data. Remember, when retrieving all data, or modifying or inserting data, you are not applying any filters.

WHERE Clause

WHERE is the most widely used clause. It helps retrieve exactly what you require. The following example table displays the STUDENT STRENGTH in various Engineering courses in a college:

ENGINEERING_STUDENTS

ENGG_ID	ENGG_NAME	STUDENT_STRENGTH
1	Electronics	150
2	Software	250
3	Genetic	75
4	Mechanical	150
5	Biomedical	72
6	Instrumentation	80
7	Chemical	75
8	Civil	60
9	Electronics & Com	250
10	Electrical	60

Now, if you want to know how many courses have over 200 in STUDENT STRENGTH, then you can simplify your search by passing on a simple statement:

```
SELECT ENGG_NAME, STUDENT_STRENGTH
FROM ENGINEERING_STUDENTS
WHERE STUDENT_STRENGTH > 200;
```

ENGG_NAME	STUDENT_STRENGTH
Software	250

HAVING Clause

HAVING is another clause used as a filter in SQL. At this point, it is important to understand the difference between the WHERE and HAVING clauses. WHERE specifies a condition, and only that set of data that passes the condition will be fetched and displayed in the result set. HAVING clause is used to filter grouped or summarized data. If a SELECT query has both WHERE and HAVING clauses, then when WHERE is used to filter rows, the result is aggregated so the HAVING clause can be applied. You will get a better understanding when you see an example.

For an explanation, another table by the name of Dept_Data has been created, and it is defined as follows:

Field	Type	Null	Key	Default	Extra
Dept_ID	bigint(20)	NO	PRI	NULL	auto_increment
HOD	varchar(35)	NO			
NO_OF_Prof	varchar(35)	YES		NULL	
ENGG_ID	smallint(6)	YES	MUL	NULL	

Now, let's have a look at the data available in this table:

Where Dept_ID is set to 100.

Dept_ID	HOD	NO_OF_Prof	ENGG_ID
100	Miley Andrews	7	1
101	Alex Dawson	6	2
102	Victoria Fox	7	3
103	Anne Joseph	5	4

104	Sophia Williams	8	5
105	Olive Brown	4	6
106	Joshua Taylor	6	7
107	Ethan Thomas	5	8
108	Michael Anderson	8	9
109	Martin Jones	5	10

There are a few simple differences between the WHERE and HAVING clauses. The WHERE clause can be used with SELECT, UPDATE, and DELETE clauses, but the HAVING clause does not enjoy that privilege; it is only used in the SELECT query. The WHERE clause can be used for individual rows, but HAVING is applied on grouped data. If the WHERE and HAVING clauses are used together, then the WHERE clause will be used before the GROUP BY clause, and the HAVING clause will be used after the GROUP BY clause. Whenever WHERE and HAVING clauses are used together in a query, the WHERE clause is applied first on every row to filter the results and ensure a group is created. After that, you will apply the HAVING clause on that group.

Now, based on our previous tables, let's see which departments have more than 5 professors:

```
SELECT * FROM Dept_Data WHERE NO_OF_Prof > 5;
```

Look at the WHERE clause here. It will check each and every row to see which record has NO_OF_Prof > 5.

Dept_ID	HOD	NO_OF_Prof	ENGG_ID
100	Miley	7	1

	Andrews		
101	Alex Dawson	6	2
102	Victoria Fox	7	3
104	Sophia Williams	8	5
106	Joshua Taylor	6	7
108	Michael Anderson	8	9

Now, let's find the names of the Engineering courses for the above data:

```
SELECT e. ENGG_NAME, e.STUDENT_STRENGTH,
       d.HOD,d.NO_OF_Prof,d.Dept_ID
FROM ENGINEERING_STUDENTS e, Dept_Data d
WHERE d.NO_OF_Prof > 5
AND e.ENGG_ID = d.ENGG_ID;
```

The result set will be as follows:

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID
Electronics	150	Miley Andrews	7	100
Software	250	Alex Dawson	6	101
Genetic	75	Victoria Fox	7	102
Biomedical	72	Sophia	8	104

		Williams		
Chemical	75	Joshua Taylor	6	106
Electronics & Com	250	Michael Anderson	8	108

Next, we GROUP the data as shown below:

```
SELECT e.ENGГ_NAME, d.HOD,d.NO_OF_Prof,d.Dept_ID
FROM ENGINEERING_STUDENTS e, Dept_Data d
WHERE d.NO_OF_Prof> 5 AND e.ENGГ_ID = d.ENGГ_ID
GROUP BY ENGГ_NAME;
```

ENGГ_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID
Biomedical	72	Sophia Williams	8	104
Chemical	75	Joshua Taylor	6	106
Electronics	150	Miley Andrews	7	100
Electronics & Com	250	Michael Anderson	8	108
Genetic	75	Victoria Fox	7	102
Software	250	Alex Dawson	6	101

Let's see which departments from this group have more than 100 students:

```
SELECT e. ENGG_NAME, e.STUDENT_STRENGTH,  
       d.HOD,d.NO_OF_Prof,d.Dept_ID  
FROM ENGINEERING_STUDENTS e, Dept_Data d  
WHERE d.NO_OF_Prof > 5 AND e.ENGG_ID = d.ENGG_ID  
GROUP BY e.ENGG_NAME HAVING e.STUDENT_STRENGTH > 100;
```

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID
Electronics	150	Miley Andrews	7	100
Electronics & Com	250	Michael Anderson	8	108
Software	250	Alex Dawson	6	101

Evaluating a Condition

A WHERE clause can evaluate more than one condition where every condition is separated by the AND operator. Let's take a look at the example below:

```
SELECT e. ENGG_NAME, e.STUDENT_STRENGTH,  
       d.HOD,d.NO_OF_Prof,d.Dept_ID  
FROM ENGINEERING_STUDENTS e, Dept_Data d  
WHERE d.NO_OF_Prof > 5 AND  
e.ENGG_ID = d.ENGG_ID AND  
100 < e.STUDENT_STRENGTH < 250;
```

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID
Electronics	150	Miley Andrews	7	100
Software	250	Alex Dawson	6	101
Genetic	75	Victoria Fox	7	102
Biomedical	72	Sophia Williams	8	104
Chemical	75	Joshua Taylor	6	106
Electronics & Com	250	Michael Anderson	8	108

There is one thing you must understand with the WHERE clause. It is only when all conditions become true for a row that it is included in the result set. If even one condition turns out to be false, the row will not be included in the result set.

The result set will be different if you replace any or all AND operators in the

above statement with the OR operator. Say you need to find out which department has either fewer than 100 students OR fewer than 5 professors. Have a look at the following statement:

```
SELECT e. ENGG_NAME,e.STUDENT_STRENGTH,  
       d.HOD,d.NO_OF_Prof,d.Dept_ID,e.ENGG_ID  
FROM ENGINEERING_STUDENTS e, Dept_Data d  
WHERE e.ENGG_ID = d.ENGG_ID AND  
(e.STUDENT_STRENGTH < 100 OR d.NO_OF_Prof < 5);
```

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID
Genetic	75	Victoria Fox	7	102
Biomedical	72	Sophia Williams	8	104
Instrumentation	80	Olive Brown	4	105
Chemical	75	Joshua Taylor	6	106
Civil	60	Ethan Thomas	5	107
Electrical	60	Martin Jones	5	109

In the above statement, notice how the parentheses are placed. You should clearly define how the AND operator exists between two conditions, and how the OR operator exists between two conditions. You will learn more about the usage of parentheses in the upcoming section. For now, please understand how the outcome of the AND/OR operators are evaluated.

AND Operator

Condition	Outcome
Where True AND True	True
Where False AND True	False
Where True AND False	False
Where False AND False	False

OR Operator

Condition	Outcome
Where True OR True	True
Where False OR True	True
Where True OR False	True
Where False OR False	False

Usage of Parentheses

In the last example, we had three conditions, and we put one condition in parentheses. If the parentheses are missing, the results will be wrong and confusing. For a change, let's try and see what happens if this occurs:

```
SELECT e.ENGГ_NAME,e.STUDENT_STRENGTH,
       d.HOD,d.NO_OF_Prof,d.Dept_ID,e.ENGГ_ID
```

FROM ENGINEERING_STUDENTS e, DEPT_DATA d
WHERE e.ENGG_ID = d.ENGG_ID AND
e.STUDENT_STRENGTH < 100 OR d.NO_OF_Prof < 5;

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID	F
Genetic	75	Victoria Fox	7	102	3
Biomedical	72	Sophia Williams	8	104	5
Electronics	150	Olive Brown	4	105	1
Software	250	Olive Brown	4	105	2
Genetic	75	Olive Brown	4	105	3
Mechanical	150	Olive Brown	4	105	4
Biomedical	72	Olive Brown	4	105	5
Instrumentation	80	Olive Brown	4	105	6
Chemical	75	Olive Brown	4	105	7
Civil	60	Olive Brown	4	105	8
Electronics &	250	Olive	4	105	9

Com		Brown			
Electrical	60	Olive Brown	4	105	1
Chemical	75	Joshua Taylor	6	106	7
Civil	60	Ethan Thomas	5	107	8
Electrical	60	Martin Jones	5	109	1

One look at the table above and you know the results are all wrong and misleading. The reason is that the instructions are not clear to the data server. Now, think about what we want to accomplish; we want to know which department has fewer than 100 students OR fewer than 5 professors. Now, witness the magic of the parentheses. The parentheses convert three conditions to two, well-defined conditions.

WHERE e.ENGG_ID = d.ENGG_ID AND

(e.STUDENT_STRENGTH < 100 OR d.NO_OF_Prof < 5);

This follows the following format: Condition1 AND (Condition2 OR Condition3).

Now, this is how the condition will be calculated:

Cond1	Cond2	Cond3	Cond2 OR Cond3	Cond1 AND (Cond2 OR Cond3)
T	T	T	T	T
T	T	F	T	T
T	F	T	T	T

T	F	F	F	F
F	T	T	T	F
F	T	F	T	F
F	F	T	T	F
F	F	F	F	F

By putting $e.STUDENT_STRENGTH < 100$ OR $d.NO_OF_Prof < 5$ into parentheses, we convert it into one condition, and the first condition will consider the output of the parentheses for the final result. Out of eight possible conditions from the table above, there are only three scenarios where the final condition is True.

The NOT Operator

To understand the usage of the NOT operator, let's replace AND with AND NOT and find out what output we receive:

```
SELECT e. ENGG_NAME,e.STUDENT_STRENGTH,
       d.HOD,d.NO_OF_Prof,d.Dept_ID,e.ENGG_ID
```

```
FROM ENGINEERING_STUDENTS e, Dept_Data d
```

```
WHERE e.ENGG_ID = d.ENGG_ID AND NOT
```

```
(e.STUDENT_STRENGTH < 100 OR d.NO_OF_Prof < 5);
```

ENGG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID	
Electronics	150	Miley Andrews	7	100	1
Software	250	Alex Dawson	6	101	2
Mechanical	150	Anne Joseph	5	103	4

Electronics & Com	250	Michael Anderson	8	108	9
-------------------	-----	------------------	---	-----	---

Remember that our desired results find which department has fewer than 100 students or fewer than 5 professors. After applying the NOT operator before the parentheses, our results look for either more than 5 professors, or more than 100 students.

Is it possible to obtain the same results while using the NOT statement? Yes! Just replace ‘>’ with ‘<’ and replace OR with AND.

```
SELECT e.ENG_NAME,e.STUDENT_STRENGTH,
       d.HOD,d.NO_OF_Prof,d.Dept_ID,e.ENG_ID
```

```
FROM ENGINEERING_STUDENTS e, Dept_Data d
```

```
WHERE e.ENG_ID = d.ENG_ID AND NOT
```

```
(e.STUDENT_STRENGTH ≥ 100 AND d.NO_OF_Prof ≥ 5);
```

ENG_NAME	STUDENT_STRENGTH	HOD	NO_OF_Prof	Dept_ID	F
Genetic	75	Victoria Fox	7	102	3
Biomedical	72	Sophia Williams	8	104	5
Instrumentation	80	Olive Brown	4	105	6
Chemical	75	Joshua Taylor	6	106	7
Civil	60	Ethan Thomas	5	107	8
Electrical	60	Martin Jones	5	109	1

Here is the output for applying AND NOT:

Cond1	Cond2	Cond3	Cond2 AND Cond3	Cond1 AND NOT (Cond2 AND Cond3)
T	T	T	T	F
T	T	F	F	T
T	F	T	F	T
T	F	F	F	T
F	T	T	T	T
F	T	F	F	T
F	F	T	F	T
F	F	F	F	T

However, use the NOT operator only when required. It can aid in the legibility of the statement, but if you use NOT when it can be avoided, it could unnecessarily complicate things for the developer.

Sequences

A sequence refers to a set of numbers that has been generated in a specified order on demand. These are popular in databases. The reason behind this is that sequences provide an easy way to have a unique value for each row in a specified column. This section explains the use of sequences in SQL.

AUTO_INCREMENT Column

This provides you with the easiest way of creating a sequence in MySQL. You only have to define the column as `auto_increment` and leave MySQL to take care of the rest. To show how to use this property, we will create a simple table and insert some records into the table.

The following command will help us create the table:

```
CREATE TABLE colleagues
```

```
(  
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
    PRIMARY KEY (id),  
    name VARCHAR(20) NOT NULL,  
    home_city VARCHAR(20) NOT NULL
```

```
);
```

The command should create the table successfully, as shown below:

```
mysql> create database tuw;
-> ;
Query OK, 1 row affected (0.05 sec)

mysql> use tuw;
Database changed
mysql> CREATE TABLE colleagues
-> (
->   id INT UNSIGNED NOT NULL AUTO_INCREMENT,
->   PRIMARY KEY (id),
->   name VARCHAR(20) NOT NULL,
->   home_city VARCHAR(20) NOT NULL
-> );
Query OK, 0 rows affected (0.30 sec)

mysql>
```

We have created a table named colleagues. This table has 3 columns: id, name, and home_city. The first column is an integer data type while the rest are varchars (variable characters). We have added the auto_increment property to the id column, so the column values will be incremented automatically. When entering data into the table, we don't need to specify the value of this column. It will start at 1 by default then increment the values automatically for each record you insert into the table.

Let us now insert some records into the table:

```
INSERT INTO colleagues
VALUES (NULL, "John", "New Delhi");
```

```
INSERT INTO colleagues
VALUES (NULL, "Joel", "New Jersey");
```

```
INSERT INTO colleagues
VALUES (NULL, "Britney", "New York");
```

INSERT INTO colleagues

VALUES (NULL, "Biggy", "Washington");

The commands should run successfully, as shown below:

```
mysql> INSERT INTO colleagues
-> VALUES (NULL, "John", "New Delhi");
Query OK, 1 row affected (0.03 sec)

mysql>
mysql> INSERT INTO colleagues
-> VALUES (NULL, "Joel", "New Jersey");
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> INSERT INTO colleagues
-> VALUES (NULL, "Britney", "New York");
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> INSERT INTO colleagues
-> VALUES (NULL, "Biggy", "Washington");
Query OK, 1 row affected (0.01 sec)

mysql>
```

Now, we can run the select statement against the table and see its contents:

```
mysql> select * from colleagues;
+----+-----+-----+
| id | name   | home_city |
+----+-----+-----+
| 1  | John   | New Delhi |
| 2  | Joel   | New Jersey |
| 3  | Britney | New York |
| 4  | Biggy  | Washington |
+----+-----+-----+
4 rows in set (0.01 sec)

mysql> _
```

We see that the id column has also been populated with values starting from 1. Each time you enter a record, the value of this column is increased by 1. We have successfully created a sequence.

Renumbering a Sequence

You notice that when you delete a record from a sequence such as the one we have created above, the records will not be renumbered. You may not be impressed by this kind of numbering. However, it is possible for you to re-sequence the records. This only involves a single trick, but make sure to check whether the table has a join with another table or not.

If you find you have to re-sequence your records, the best way to do it is by dropping the column and then adding it. Here, we'll show how to drop the id column of the colleagues table.

The table is as follows for now:

```
mysql> select * from colleagues;
+----+-----+-----+
| id | name   | home_city |
+----+-----+-----+
| 1  | John   | New Delhi |
| 2  | Joel   | New Jersey |
| 3  | Britney | New York |
| 4  | Biggy  | Washington |
+----+-----+-----+
4 rows in set (0.01 sec)

mysql> _
```

Let us drop the id column by running the following command:

ALTER TABLE colleagues DROP id;

```
mysql> ALTER TABLE colleagues DROP id;
Query OK, 4 rows affected (0.40 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> _
```

To confirm whether the deletion has taken place, let's take a look at the table data:

```
mysql> select * from colleagues;
+-----+-----+
| name   | home_city |
+-----+-----+
| John   | New Delhi |
| Joel   | New Jersey |
| Britney | New York  |
| Biggy  | Washington |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

The deletion was successful. We combined the ALTER TABLE and the DROP commands for the deletion of the column. Now, let us re-add the column to the table:

ALTER TABLE colleagues

ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id);

The command should run as follows:

```
mysql> ALTER TABLE colleagues
-> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
-> ADD PRIMARY KEY (id);
Query OK, 0 rows affected (0.76 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

We started with the ALTER TABLE command to specify the name of the table we need to change. The ADD command has then been used to add the column and set it as the primary key for the table. We have also used the auto_increment property in the column definition. We can now query the table to see what has happened:

```
mysql> ALTER TABLE colleagues
->     ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
->     ADD PRIMARY KEY (id);
Query OK, 0 rows affected (0.76 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> select * from colleagues;
+----+-----+-----+
| id | name  | home_city |
+----+-----+-----+
| 1  | John  | New Delhi |
| 2  | Joel  | New Jersey |
| 3  | Britney | New York |
| 4  | Biggy  | Washington |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> _
```

The id column was added successfully. The sequence has also been numbered correctly.

MySQL starts the sequence at index 1 by default. However, it is possible for you to customize this when you are creating the table. You can set the limit or amount of the increment each time a record is created. Like in the table named colleagues, we can alter the table for the auto_increment to be done at intervals of 2. This is achieved through the code below:

```
ALTER TABLE colleagues AUTO_INCREMENT = 2;
```

The command should run successfully, as shown below:

```
mysql> ALTER TABLE colleagues AUTO_INCREMENT = 2;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

We can specify where the auto_increment will start at the time of the creation of the table. The following example shows this:

```
CREATE TABLE colleagues2
```



```
(  
    id INT UNSIGNED NOT NULL AUTO_INCREMENT = 10,  
    PRIMARY KEY (id),  
    name VARCHAR(20) NOT NULL,  
    home_city VARCHAR(20) NOT NULL  
);
```

From the above instance, we set the `auto_increment` property on the `id` column, and the initial value for the column will be 10.

CHAPTER 7: VIEWS

In this book, tables have been used to represent data and information. Views are like virtual tables, but they don't hold any data, and their contents are defined by a query. One of the biggest advantages of a view is that it can be used as a security measure by restricting access to certain columns or rows. You can also use views to return a selective amount of data instead of detailed data. A view protects the data layer while allowing access to the necessary data. It differs from a stored procedure because it doesn't use parameters to carry out a function.

Encrypting a View

You can create a view without columns that contain sensitive data, thus hiding the data you don't want to share. You can also encrypt the view definition, which returns data of a privileged nature. Not only are you restricting certain columns in a view, you are also restricting who has access to the view. However, once you encrypt a view, it is difficult to get back to the original view detail. The best precaution is to make a backup of the original view.

Creating a View

To create a view in SSMS, expand the database you are working on, right click on Views and select New View. The View Designer will appear, showing all the tables that you can add. Add the tables you want in the view. Now, select which columns you want in the view. You can now change the sort type for each column from ascending to descending, and can also give aliases to column names. On the right side of sort type there is Filter. Filter restricts what a user can and cannot see. Once you set a filter (e.g. sales > 1000) a user cannot retrieve more information than this view allows. In the T-SQL code there is a line stating TOP (100) PERCENT which is the default. You can remove it (also remove the order by statement) or change the value. Once you have made the changes, save the view with the save button and start the view with vw_ syntax. You can see the contents of the view if you refresh the database, open views, right click on the view, and select top 1000 rows.

Indexing a View

You can index a view just like you can index a table; the rules are very similar. When you build a view, the first index needs to be a unique clustered index. Subsequent non-clustered indexes can then be created. You need to have the following set to ON or OFF as indicated below:

SET ANSI_NULLS ON

SET ANSI_PADDING ON

SET ANSI_WARNINGS ON

SET CONCAT_NULL_YIELDS_NULL ON

SET ARITHABORT ON

SET QUOTED_IDENTIFIER ON

SET NUMERIC_ROUNDABORT OFF

Now, type the following: **CREATE UNIQUE CLUSTERED INDEX
_ixCustProduct**

ON table.vw_CusProd(col1,col2)

CHAPTER 8: TRIGGERS

Triggers in SQL are used to automatically perform an action on the database when a DDL or DML operation takes place. For example, picture a situation where you want to give a discount of 10 percent on the prices of all the examination records that will be inserted in future; you can do so using triggers. Another example is log maintenance. You will want that whenever a new record is inserted, or when an entry is made in the log table. You can perform such operations with triggers (“CS145 Lecture Notes (8) -- Constraints and Triggers”).

You will encounter two types of SQL triggers: AFTER trigger and INSTEAD OF trigger. The AFTER trigger is executed after an action has been performed. For instance, if you want to log the operation after an operation has been performed, you can use the AFTER trigger. On the other hand, if you want to perform a different operation in place of the actual event, you can use the INSTEAD OF trigger. In this chapter we will explore examples of both types of triggers:

Trigger Syntax

Trigger syntax is complex, so before we look at an actual example, let's first understand the syntax. Take a look at the following script:

```
CREATE [ OR ALTER ] TRIGGER [ schema_name.]name_of_trigger
ON {TABLE | VIEW}

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS

{

BEGIN

The SQL instructions/statements to execute...

END

}
```

Now, let's see what is happening in the above script, line by line. To come up with a trigger, the CREATE TRIGGER statement is used. To ALTER the existing trigger, we use the ALTER TRIGGER statement. Next the database scheme and the name of the trigger are specified. After that, the ON TABLE clause specifies the name of the table or view. Next, you have to mention the type of trigger by using the FOR clause. Here, you can write AFTER or INSTEAD OF. Finally, you write AS BEGIN and END statements. Within the BEGIN and END clause you include the SQL statements that you want executed when the trigger fires.

AFTER Trigger

Now let's see a simple example of the AFTER trigger. We will insert a new record in the Patients table of a hospital database. When the record is inserted, we will use an AFTER trigger to add an entry into the Patient_Logs table. We do not currently have a Patient_Logs table in the hospital database. Execute the following query to create this table:

```
USE Hospital;

CREATE TABLE Patient_Logs
(
id INT IDENTITY(1,1) PRIMARY KEY,
patient_id INT NOT NULL,
patient_name VARCHAR(50) NOT NULL,
action_performed VARCHAR(50) NOT NULL
)
```

Now, we have our Patient_Logs table. Let's create our AFTER trigger that will automatically log an entry in the Patient_Logs table whenever a record is inserted into the Patients table. Take a look at the following script:

```
USE Hospital;

GO -- begin new batch

CREATE TRIGGER [dbo].[PATIENT_TRIGGER]
ON [dbo].[Patients]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @patient_id INT, @patient_name VARCHAR(50)

    SELECT  @patient_id = INSERTED.id,      @patient_name =
```

```
INSERTED.name  
  
FROM INSERTED  
  
INSERT INTO Patient_Logs  
  
VALUES(@patient_id, @patient_name, 'Insert Operation')  
  
END
```

In the above script, we created a trigger named PATIENT_TRIGGER on the dbo database scheme. This is an AFTER-type trigger, and it fires whenever a new record is inserted into the Patients table. In the body of the trigger starting from BEGIN, we set variable NOCOUNT ON. This returns the number of affected rows. Next, we declared two variables: @patient_id and @patient_name. These variables will hold the id and name of the newly inserted patient record.

The INSERTED clause here is a temporary table that holds the newly inserted record. The SELECT statement within the script is targeted at selecting the ID and name from the INSERTED table and storing them in the patient_id and patient_name variables, respectively. Finally, the INSERT statement is used to insert the values of these variables into the Patient_Logs table.

Now, simply insert a new record in the Patients table by executing the following query:

```
USE Hospital;  
  
INSERT INTO Patients  
  
VALUES('Suzana', 28, 'Male', 'AB-', 65821479)
```

When the above query executes, the PATIENT_TRIGGER fires, which inserts a record in the Patient_Logs table, too. Now, if you select records from your Patient_Logs table, you will see your new log entry.

```
USE Hospital;  
  
SELECT * FROM Patient_Logs
```

The output will look like this:

ID	Patient_ID	Patient_Name	Action_Performed
1	14	Suzana	Insert Operation

INSTEAD OF Trigger

In the previous section, we explored the AFTER trigger, which is executed after an event has occurred. In this section, we will study the INSTEAD OF trigger. As mentioned earlier, the INSTEAD OF trigger operates when we want to execute an alternative action instead of carrying out the event that caused/triggered the action. For instance, consider a scenario where you want to update the brand names of different productions in a company. You also want to implement the condition that, if the brand exists, it should not be updated. You also want to keep track of all the efforts to update the brands. In this case, you can utilize the INSTEAD OF trigger to execute a script that checks whether the updated brand exists. You will log an entry in the table that states “brand cannot be updated.” Otherwise, update the brand and log an entry that states “brand updated.” The following example creates this INSTEAD OF trigger.

```
CREATE TRIGGER company.nw_brands
ON company.main_brands
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO company.brand_approvals (
        brand_name
    )
    SELECT
        i.brand_name
    FROM
        inserted i
    WHERE
        i.brand_name NOT IN (
            SELECT
```

brand_name

FROM

company.brands

);

END

The above trigger inserts a new brand name into the company.brand_approvals after determining that it is absent in company.brands.

CHAPTER 9: STORED ROUTINES AND FUNCTIONS IN SQL

Stored Procedures

Imagine that you have to execute a large, complex query that performs a specific task. If you need to perform that task again, you will have to write that large piece of script all over again. If you are executing your queries over a network, you will have to send large amounts of data over the network. This approach has several drawbacks. You have to execute complex scripts again and again, and you will end up consuming a large amount of bandwidth over the network.

A better approach is to utilize stored procedures. A stored procedure is a set of SQL statements grouped under a single heading. In this chapter, we will cover how to create, execute, modify, and delete stored procedures in the MS SQL Server (“SQL Stored Procedures for SQL Server”). However, before that, let’s take a look at some of the advantages of stored procedures.

Benefits Offered by Stored Procedures in SQL

Execution Speed

When a stored procedure is executed for the first time, it is optimized, parsed, compiled, and stored in the local database cache. The next time you carry out the stored procedure, the compiled version is executed from cache memory which greatly improves the execution speed of the stored procedures.

Less Network Bandwidth

Since stored procedures are compiled and stored in the cache memory, this means that the large, complex SQL script is only sent to the database over the network once. When you want to execute the stored procedure again, you only have to send the command to execute that stored procedure rather than the whole script. This reduces bandwidth utilization.

Reusability

Another advantage of stored procedures is their usability. A stored procedure executes a set of statements. Therefore, you can write the statements once and then

simply execute that stored procedure whenever you want to execute that particular set of statements.

Security

SQL stored procedures also foster security since you can restrict users from accessing stored procedures while still allowing them to execute other SQL statements.

Creating a Stored Procedure

It is very easy to create stored procedures in SQL. Take a look at the syntax outlined below:

```
CREATE PROCEDURE procedure_name  
AS  
BEGIN  
SQL statements here...  
END
```

To create a stored procedure, you have to use the `CREATE PROCEDURE` command followed by the name of the stored procedure. Then, you have to use the `AS` clause followed by `BEGIN` and `END` statements. The body, or the SQL statements, that the procedure will execute are contained within `BEGIN` and `END`.

Now, it is time to create a simple stored procedure named `spListPatients` which lists records from the `Patients` table arranged alphabetically by name:

```
USE Hospital;  
GO --- Begins New Batch Statement  
CREATE PROCEDURE spListPatients  
AS  
BEGIN  
SELECT * FROM Patients  
ORDER BY name
```

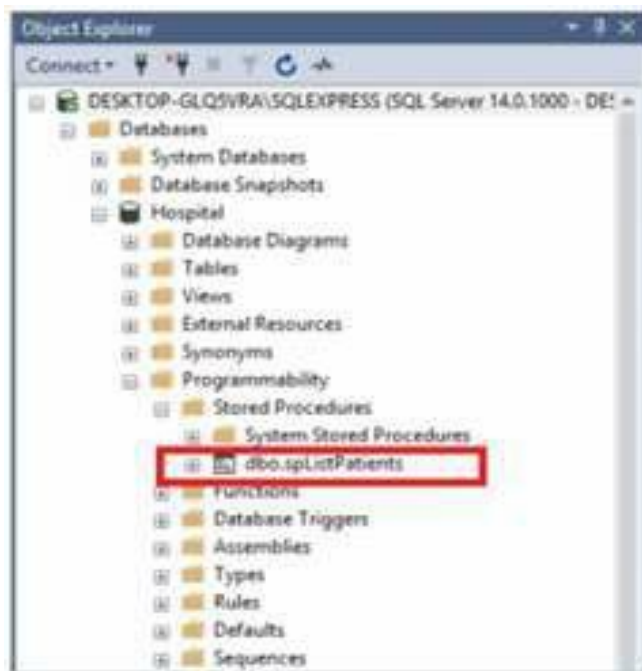
END

Take a look at the above script, here we started with the USE Hospital command. This tells the database server that the stored procedure should be stored in the hospital database. Next, we use the GO command to start a new batch statement. It is important to mention here that the CREATE PROCEDURE statement should always be the first statement in the batch. However, in this case, we have to write USE Hospital as the first statement, so we inserted the GO clause here as well.

To create the stored routine, go ahead and execute the above script. To see if the stored procedure has actually been created, go to the following path in your SQL Server management studio:

Object Explorer → Database → Hospital (your DB name) → Programmability → Stored Procedures

The following screenshot shows this:



Here, you can see your newly created stored procedure named “spListPatients.”

Executing a Stored Procedure

To execute a stored procedure, we use the following syntax:

```
EXECUTE stored_procedure_name
```

Yes, it is that simple! Now let us execute our spListPatients stored procedure that we created in the last section. Execute the following query:

```
EXECUTE spListPatients
```

We know that the spListPatients stored procedure retrieves records of all patients arranged alphabetically by name. The output is shown below:

ID	NAME	AGE	GENDER	BLOOD_GROUP	PHONE
8	Alex	21	Male	AB-	46971235
10	Elice	32	Female	O+	34169872
7	Frank	35	Male	A-	85473216
9	Hales	54	Male	B+	74698125
3	James	16	Male	O-	78452369
6	Julie	26	Female	A+	12478963
2	Kimer	45	Female	AB+	45686412
4	Matty	43	Female	B+	15789634
5	Sal	24	Male	O+	48963214
1	Tom	20	Male	O+	123589746

Inserting Records

We know how to create stored procedures for a select statement. In this section, we will see how to create a stored procedure for inserting records. For this purpose, we will create parameterized stored procedures. The following query creates a stored procedure that inserts one record in the Patients table:

```
USE Hospital;
```

GO --- Begins New Batch Statement

CREATE PROCEDURE spInsertPatient

(@Name VARCHAR(50), @Age int, @Gender VARCHAR(50),
@Blood_Group VARCHAR(50), @Phone int)

AS

BEGIN

INSERT Into Patients

Values (@Name, @Age, @Gender, @Blood_Group, @Phone)

END

Here, after the name of the stored procedure, spInsertPatient, we specified the list of parameters for the stored procedure. The parameters should match the table columns in which you are inserting the data. You can give any name and order to these parameters but the parameters' data types need to match the data types of the corresponding columns. For instance, the data type of the age column in the Patients table is int, so, therefore, the parameter data type that matches this column (@Age, in this case) must also be int.

Now, when you execute the above stored procedure, you also have to pass the values for the parameters. These values will be inserted in the database. It is important to note that the type of the values passed must also match the type of the parameters. Take a look at the following script:

EXECUTE spInsertPatient 'Julian', 25, 'Male', 'O+', 45783126

The above line executes the spInsertPatient stored procedure and inserts a new patient record in the Patients table where the name of the patient is Julian. If a new record has actually been inserted, execute the spListPatients stored procedure again. In the output, you will see your newly inserted record.

Updating Records

The stored procedure for updating records is similar in syntax to that of inserting records. In the case of updating records, we create a stored procedure with parameters that receive values for SET and WHERE conditions. Let us look at an example. The following stored procedure updates the age of a patient named

Julian;

USE Hospital;

GO --- Begins New Batch Statement

```
CREATE PROCEDURE spUpdatePatient (@PatientName Varchar(50),  
@PatientAgeint)
```

```
AS
```

```
BEGIN
```

```
UPDATE Patients
```

```
SET age= @PatientAge
```

```
WHERE name = @PatientName
```

```
END
```

The following script executes the spUpdatePatient stored procedure.

```
EXECUTE spUpdatePatient 'Julian', 30
```

When you execute the above stored procedure, the age of the patient named Julian will be updated to 30.

Deleting Records

To delete records using stored procedures, you have to write a parameterized stored procedure that accepts the value for filtering records as parameter. The following script creates a stored procedure that deletes the record of the patient by passing the patient's name as an argument to the parameter:

```
CREATE PROCEDURE spDeletePatient (@PatientName Varchar(50))
```

```
AS
```

```
BEGIN
```

```
DELETE FROM Patients
```

```
WHERE name = @PatientName
```

```
END
```

Now, while executing this stored procedure, you have to pass the name of the patient whose record will be deleted. The following script deletes the record of the patient named Julian.

```
EXECUTE spDeletePatient 'Julian'
```

Modifying a Stored Procedure

If you want to change the functionality of a stored procedure, you can modify it. For instance, if you want the spListPatients stored procedure to retrieve all the records from the Patients table in reverse alphabetical order, you can modify it by using the ALTER statement. Take a look at the following script:

```
USE Hospital;
```

```
GO --- Begins New Batch Statement
```

```
ALTER PROCEDURE spListPatients
```

```
AS
```

```
BEGIN
```

```
SELECT * FROM Patients
```

```
ORDER BY name DESC
```

```
END
```

Now, if you execute the spListPatients stored procedure, you will see all the records from the Patients table retrieved in reverse alphabetical order by name. The output is shown below:

ID	NAME	AGE	GENDER	BLOOD_GROUP	PHONE
1	Tom	20	Male	O+	123589746
5	Sal	24	Male	O+	48963214
4	Matty	43	Female	B+	15789634
2	Kimer	45	Female	AB+	45686412
6	Julie	26	Female	A+	12478963

3	James	16	Male	O-	78452369
9	Hales	54	Male	B+	74698125
7	Frank	35	Male	A-	85473216
10	Elice	32	Female	O+	34169872
8	Alex	21	Male	AB-	46971235

Deleting a Stored Procedure

To delete a stored procedure, the DROP PROCEDURE statement is used. The following script deletes spDeletePatient stored procedure.

```
DROP PROCEDURE spDeletePatient
```

Functions

Functions are similar to stored procedures, but they differ in that functions (or User Defined Functions - UDF) can execute within another piece of work – you can use them anywhere you would use a table or column. They are like methods, and are small and quick to run. You simply pass a function some information and it returns a result. You will learn about two function types: scalar and table-valued. The difference between the two is what you can return from the function.

Scalar Functions

A scalar function can only return a single value of the type defined in the RETURN clause. You can use scalar functions anywhere that the scalar matches the same data type that is being used in the T-SQL statements. When calling them, you can omit a number of the function's parameters. You need to include a return statement if you want the function to complete and return control to the calling code. The syntax for the scalar function is as follows:

```
CREATE FUNCTION schema_Name function_Name
-- Parameters
RETURNS dataType
AS
BEGIN
-- function code goes here
RETURN scalar_Expression
END
```

Table-Valued Functions

A table-valued function (TVF) lets you return a table of data rather than the single value that you get from a scalar function. You can use the table-valued function anywhere you would normally use a table, which is usually from the FROM clause in a query. With table-valued functions, it is possible to create reusable code framework in a database. The syntax of a TVF is as follows:

CREATE FUNCTION function_Name (@variableName)

RETURNS TABLE

AS

RETURN

SELECT columnName1, columnName2

FROM Table1

WHERE columnName > @variableName

Notes on Functions

A function cannot alter any external resource, like a table, for example. A function needs to be robust, and if there is an error generated inside it from either invalid data or logic, then it will stop executing and the control will return to the T-SQL that called it.

Cursors

First off, what is a cursor? A cursor is a mechanism that allows us to step through the rows returned by a SQL statement.

So far, we have been learning to use SELECT statements to retrieve information from our tables. These statements return all the rows that match the stated criteria. However, there is no way for us to step through the rows one at a time. If we want to do that, we have to use cursors.

The syntax for declaring a cursor is:

```
DECLARE cursor_name CURSOR FOR  
SELECT statement
```

This cursor declaration must be done after any variable declaration.

After declaring the cursor, we also need to declare a handler that defines what the cursor should do when it reaches the last row of the results returned by the SELECT statement. Normally, we want the cursor to set a variable to a certain value. For instance, we may want it to set a variable called `v_done` to 1.

The syntax is:

```
DECLARE CONTINUE HANDLER FOR  
NOT FOUND SET variable_name = value;
```

Next, we need to open the cursor using this statement:

```
OPEN cursor_name;
```

Once the cursor is open, we can use a loop to step through the rows. Within the loop, we need to retrieve the row that the cursor is currently pointing at and store the data into variables. The syntax for retrieving the row is:

```
FETCH cursor_name INTO variable_names;
```

After the row is retrieved, the cursor will automatically move to the next row. The LOOP statement will repeatedly fetch each row into the variables and process those variables until it reaches the end of the results.

When that happens, we leave the loop and close the cursor. The syntax for closing the cursor is:

```
CLOSE cursor_name;
```

To help clarify, let's look at an example.

Suppose we want to get the names and genders of all employees in our employees table and combine them into a single line of text. Here's how we do it:

```
DELIMITER $$
```

```
CREATE FUNCTION get_employees () RETURNS  
    VARCHAR(255) DETERMINISTIC
```

```
BEGIN
```

```
DECLARE v_employees VARCHAR(255) DEFAULT "";
```

```
DECLARE v_name VARCHAR(255);
```

```
DECLARE v_gender CHAR(1);
```

```
DECLARE v_done INT DEFAULT 0;
```

```
    DECLARE cur CURSOR FOR
```

```
SELECT em_name, gender FROM employees;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = 1;
```

```
OPEN cur;
```

```
employees_loop: LOOP
```

```
    FETCH cur INTO v_name, v_gender;
```

```
    IF v_done = 1 THEN LEAVE employees_loop;
```

```
ELSE SET v_employees = concat(v_employees, ', ', v_name, ': ',  
    v_gender);
```

```
END IF;
```

```
END LOOP;
```

```
CLOSE cur;
```

```
RETURN substring(v_employees, 3);
```

```
END $$
```

```
DELIMITER;
```

First, we declare a function called `get_employees`.

Within the function, we declare four local variables, `v_employees`, `v_name`, `v_gender`, and `v_done`, and set the default values of `v_employees` and `v_done` to "" and 0 respectively (using the `DEFAULT` keyword).

Next, we declare a cursor called `cur` for the following `SELECT` statement:

```
SELECT em_name, gender FROM employees;
```

We also declare a handler called `v_done` for this cursor.

After that, we open the cursor and declare a loop called `employees_loop` to work with the cursor.

Within the loop, we fetch the row that `cur` is currently pointing at and store the results into the `v_name` and `v_gender` variables. We need to store the results into two variables as we selected two columns from the `employees` table.

Next, we check if `v_done` equals 1.

`v_done` equals 1 when the cursor reaches the end of the table (i.e. when there is no more data for the cursor to fetch).

If `v_done` equals 1, we exit the loop. Otherwise, we use the `concat()` function to concatenate `v_employees` with `v_name` and `v_gender`, separated by a colon and comma.

The initial value of `v_employees` is an empty string (denoted by ""). When we first concatenate `v_employees` with comma, `v_name`, colon, and `v_gender`, we'll get, James Kpk: M

James Kpk and M are the name and gender of the first employee.

After we do this concatenation, we assign the concatenated result back to the `v_employees` variable.

The `LOOP` statement then repeats the same process for all the other rows in the table.

After looping, we close the cursor and return the value of `v_employees`. However, before we do that, we use the `substring()` function to remove the first comma from the string.

With that, the function is complete.

Call this function using the statement:

```
SELECT get_employees();
```

You'll get:

James Kpk: M, Clara Champ: F, Joyce Ytr: F, Jason CC: M, Prudence Nera: F,
Walker Waters: M as the result.

CHAPTER 10: HOW TO PIVOT DATA

Pivoting data is defined as converting your data, which is normally presented in rows, into column presentations.

Through the use of PIVOT queries, you can manipulate the rows and columns to present variations of the table that will help you in analyzing your table (Xue, 2018). PIVOT can present a column as multiple columns. You also have the option to use the UNPIVOT query. UNPIVOT will reverse what PIVOT does.

This is extremely useful in multidimensional reporting. You may need to utilize it when generating your numerous reports.

How Can You Compose Your PIVOT Query?

Step 1 – Ascertain That Your SQL Allows Pivot Queries

Is the version of your SQL server appropriate for PIVOT queries? If not, then you cannot accomplish these specific queries.

Step 2 – Determine What You Want Displayed in Your Results

Identify the column or data that you want to appear in your results or output page.

Step 3 – Prepare Your PIVOT Query Using SQL

Use your knowledge of SQL to compose or create your PIVOT query.

To understand more about PIVOT, let’s use the table below as a base table:

ProductSales

ProductName	Year	Earnings
RazorBlades1	2015	12,000.00
BarHandles1	2015	15,000.00
RazorBlades2	2016	10,000.00
BarHandles2	2016	11,000.00

Let’s say you want an output that will show the ProductName as the column headings. This would be your PIVOT query:

Example 1:

```
SELECT * FROM ProductSales
PIVOT (SUM(Earnings))
FOR ProductNames IN ([RazorBlades1], [BarHandles1],
[RazorBlades2], [BarHandles2]) AS PVT
```

With the PIVOT query above, your ProductSales table will now appear like this:

#ProductNamesPIVOTResults

--	--	--	--	--

ProductName	RazorBlades1	BarHandles1	RazorBlades2	BarHandles2
Year	2015	2015	2016	2016
Earnings	12,000.00	15,000.00	10,000.00	11,000.00

You can also manipulate the table based on your preferences.

Example 2:

If you want the years to be the column headings, you can write your PIVOT query this way:

```
SELECT * FROM ProductSales
PIVOT (SUM(EARNINGS))
FOR Year IN ([2015], [2016]) AS PVT
```

The resulting output would be the table below:

YearPivotResult

ProductName	2015	2016
RazorBlades1	12,000.00	NULL
BarHandles1	15,000.00	NULL
RazorBlades2	NULL	10,000.00
BarHandles2	NULL	11,000.00

There are even more ways for you to use your PIVOT query.

Your UNPIVOT query will work opposite from the way your PIVOT query does. It will convert the columns into rows.

Example:

Based off of our PIVOT query Example 1 shown above, you can generate this UNPIVOT query:

```
SELECT ProductName, Year, Earnings
FROM #ProductNamesPivotResult
UNPIVOT (Earnings FOR ProductName IN ([RazorBlades1], [BarHandles1],
[RazorBlades2], [BarHandles2]) AS UPVT
```

The output would be the original table:

#ProductSalesUnpivotResult

ProductName	Year	Earnings
RazorBlades1	2015	12,000.00
BarHandles1	2015	15,000.00
RazorBlades2	2016	10,000.00
BarHandles2	2016	11,000.00

CHAPTER 11: CLONE TABLES

CLONE TABLES are identical tables that you can create to perform particular SQL tasks. These CLONE TABLES have exactly the same format and content as the original table, so you won't have problems practicing on them first.

Tables that are retrieved by using CREATE TABLE SELECT may not have the same indexes and other values as the original, so CLONE TABLES are the best in this regard (“SQL Clone Tables”, 2018).

How to Create a Clone Table

You can do this by following these steps in MySQL:

Step 1 – Retrieve the Complete Structure of Your Selected Table

Obtain a CREATE TABLE query by displaying the CREATE TABLE keywords.

Step 2 – Rename the Table and Create Another One

Change the table_name to your CLONE TABLE name. After you submit the query, you will have two identical tables.

Step 3 – To Retain the Data in the Table, Use the Keywords INSERT INTO and SELECT

Purposes of Clone Tables

- To create sample tables that you can work on without being worried about destroying the whole table.
- To act as practice tables for beginners, so that the tables in the databases are safe and protected.
- To feature new interfaces for new users.
- To ensure that the integrity of the tables in your databases is protected from new users.

CLONE TABLES can be very useful, if utilized properly. Learn how to use them to your advantage.

CHAPTER 12: NORMALIZATION OF YOUR DATA

With SQL, normalization is the process of taking your database and breaking it up into smaller units. Developers will often use this procedure as a way to make your database easier to organize and manage. Some databases are really large and hard to handle. If you have thousands of customers, for example, your database could get quite large. This can make it difficult to get through all the information at times because there is just so much that you have to filter through to get what you want.

However, with database normalization, this is not as big of an issue. The developer will be able to split up some of the information so that the search engine is better able to go through the database without taking as long or running into as many issues. In addition, using database normalization will help to ensure the accuracy and the integrity of all the information that you place into the database.

How to Normalize the Database

Now that you understand a number of the reasons for choosing to do database normalization, it is time to work on the actual process. The process of normalization basically means that you are going to decrease any of the redundancies that are inside the database (“What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples”). You will be able to use this technique any time that you want to design, or even redesign, your database. Some of the tools that you need and the processes that you should learn to make this happen include:

Raw Databases

Any database that hasn’t gone through the process of normalization can contain multiple tables that have the same information inside of them. This redundant information can slow down the search process and can make it difficult for your database to find the information that you want. Some of the issues that you could have with a database that hasn’t gone through normalization include slow queries, inefficient database updates, and poor security.

This is all because you have the same information in your database several times, and you haven’t divided it into smaller pieces to make things easier in your search. Below is an example of a database that needs some help with normalization:

COMPANY_DATABASE	
emp_id	cust_id
last_name	cust_name
first_name	cust_address
middle_name	cust_city
address	cust_state
city	cust_zip
state	cust_phone
zip	cust_fax
phone	ord_num
pager	qty
position	ord_date
date_hire	prod_id
pay_rate	prod_desc
bonus	cost
date_last_raise	

As you can see, there are quite a few points where the same information is asked several times. While this is a small table, some of the bigger ones could have more issues that will drastically slow down retrieval of results and end up causing a mess. Using the process of normalization on your original database can help cut out some of this mess and make things more efficient for both you and the user to find the information needed.

Logical Design

Each of the databases that you are working on need to be created as well as designed with the end users in mind. You can make a fantastic database, but if the users find that it is difficult to navigate, you have just wasted a lot of time in the process. Logical model, or logical design, is a process where you can arrange the data into smaller groups that are easier for the user to find and work with.

When you are creating the data groups, you must remember that they should be manageable, organized, and logical. Logical design will make it easier for you to reduce, and in some cases even eliminate, any of the data repetition that occurs in your database.

The Needs of the End User

When designing the database, it is important to keep the end users' needs in mind. The end users are those who will be using the database that you develop, so you will need to concentrate on creating a database that is beneficial to the customer.

In general, you will want to create a database that is user friendly, and if you are able to add in an intuitive interface, this can be helpful, too. Good visuals are a way to attract the customer, but you need to have excellent performance as well or the customer may get frustrated with what you are trying to sell them on the page.

When you are working on creating a new database for your business, some of the questions that you should answer to ensure that you are making the right database for your customers include:

- What kind of data will I store on here?
- How will the users be able to access the data?
- Do the users need any special privileges to access the data?
- How can the users group the data within the database?
- What connections will I make between the data pieces that I store?
- How will I ensure the integrity and accuracy of the data?

Data Repetition

When creating your database, you need to ensure that the data is not repetitive. You need to work on minimizing this redundancy as much as possible. For example, if you have the customer's name in more than one table, you are wasting a lot of time and space in the process because this duplicated data is going to lead to inefficient use of your storage space.

On top of wasting the storage space, this repetitive entry will also lead to confusion. This happens when one table's data doesn't match up or correlate with others, even when the tables were created for the same object or person.

Normal Forms

A normal form is a method of identifying the levels or the depth that you will require to normalize the database. In some cases, it just has to be cleaned up a little bit, but other times there will be excessive work required to ensure that the table looks clean and organized. When you are using a normal form, you can establish the level of normalization required to perform on the database. There

are three forms that you will use for normalizing databases: the first form, the second form, and the third form.

Every subsequent form that you use will rely on the techniques that you used on the form preceding it. You should ensure that you have used the right form before you move on. For example, you cannot skip from the first form to the third form without doing the second form.

1. **First Form** — The goal of using this form is to take the data and segregate it into tables. Once the tables are designed, the user will be able to assign a primary key to each individual table or groups of tables. To attain this first form, you will divide up the data into small units, and each one needs to have a primary key and a lack of redundant data.
2. **Second Form** — The goal of using the second form is to find the data that is at least partially reliant on those primary keys. Then, this data can be transferred over to a new table as well. This will help to sort out the important information and leave behind redundant information or other unnecessary things.
3. **Third Form** — The goal of the third form is to eliminate the information that does not depend on any of your primary keys. This will help to dispose of the information that is in the way and slowing down the computer, and you will also discard the redundant and unneeded information along the way.

Naming Conventions

When you are working on the normalization process, you need to be particular and organized with your naming conventions. Make use of unique names that will enable you to store and then later retrieve your data. You should choose names that are relevant to the information that you are working on in some way, so that it is easier to remember these names in the future. This will help keep things organized and avoid confusion in the database.

Benefits of Normalizing Your Database

We have spent some time talking about normalization in your database, but what, exactly, are the benefits? Why should you go through this whole process simply to clean out the database that your customers are using? Would it still work just fine

to leave the information as is and let the search sift through the redundancies and other information that you don't need? Here are some of the beneficial reasons why you should utilize normalization and avoid letting your database become inefficient:

- Keeps the database organized and easier to use
- Reducing repetitive and redundant information
- Improves security for the whole system and its users
- Improves flexibility in your database design
- Ensures consistency within the database

It may seem like a hassle to go through and normalize the database, but it truly makes the whole experience better. Your customers will have an easier time finding the information that they want, the searching and purchasing process will become more streamlined, and your security will be top of the line.

Despite the many benefits, there is one downside to normalization. This process does reduce the performance of the database in some cases. A normalized database will require more input/output, processing power, and memory to get the work done. Once normalized, your database is going to need to merge data and find the required tables to get anything done. While this can help make the database system more effective, it is still important to be aware of it.

Denormalization

Another process that you should be aware of is denormalization. This allows you to take a normalized database and change it to ensure that the database has the capability of accepting repetition. The process is important in some instances in order to increase how well the database is able to perform. While there are some benefits to using the normalization process, it is bound to slow down the database system simply because it is working through so many automated functions. Depending on the situation, it could be better to have this redundant information rather than work with a system that is too slow.

Normalization of your database has many great benefits and it is pretty easy to set it all up. You just need to teach the database to get rid of information that it finds repetitive or that could be causing some of the issues within your system. This can help to provide more consistency, flexibility, and security throughout the whole

system.

Database Normal Forms

We briefly touched in the topic of normal forms, and in this section, we will look at them in more detail. To normalize a database, you need to ensure that a certain normal form is achieved. A database is said to be in a particular normal form if it adheres to a specific set of rules. A database can have six normal forms, which are denoted as 1NF, 2NF, 3NF, 4NF, 5NF, and 6NF. The higher the normal form, the more a database is normalized. Most of the real-world databases are in third normal form (3NF). You have to start with 1NF and work your way up to the higher normal forms. In this chapter we will delve into the different types of normal forms.

First Normal Form (1NF)

A database in 1NF adheres to the following rules:

1. Atomic Column Values

All the columns in the table should contain atomic values. This means that there should be no column that contains more than one value. The following table, which contains multiple names in the PatientName column, does not have atomic column values

PatientName	DepName
Jane, Elizabeth	Alan, Cardiology
Mark, Royce	Pathology

A downside to the table shown above is that you cannot perform CRUD operations on this database. For instance, you cannot delete the record of Jane alone; you would have to delete all the records in the Cardiology department. Similarly, you cannot update the department name for Mark without also updating it for Royce.

2. No Repeated Column Groups

Repeated column groups are group of columns that have similar data. In the following table, the PatientName1, PatientName2, and PatientName3 columns are considered repeated columns since all of them serve to store names of patients.

PatientName1	PatientName2	PatientName3	DepName
Jane	Alan	Elizabeth	Cardiology
Mark	Royce		Pathology

This approach is also not ideal since, if we have to add more patient names, we will have to add more and more columns. Similarly, if one department has fewer patients than the other, the patient name columns for the former will have empty records, which leads to a huge waste of storage space.

3. Unique Identifier for Each Record

Each record in the table must have a unique identifier. A unique identifier is also known as a primary key and the column that contains the primary key is called the primary key column. The primary key column must have unique values. For instance, in the following table, PatientID is the primary key column.

PatientID	PatientName	PatientAge	PatientGender
1	Jane	45	Female
2	Mark	54	Male
3	Alan	65	Male
4	Royce	21	Male
5	Elizabeth	27	Female

If a database adheres to all of these conditions, it is considered to be in first normal form. You need to check the finer details to be able to establish this aspect.

Second Normal Form (2NF)

For a database to be in second normal form, it must adhere to the following three conditions:

1. Adheres to All the Conditions That Denote First Normal Form
2. No Redundant Data in Any Column Except the Foreign Key Column
3. Tables Should Relate to Each Other via Foreign Keys

Take a look at the following table:

PatientID	Patient Name	Patient Age	Patient Gender	DepName	DepHead	NoOfBeds
1	Jane	45	Female	Cardiology	Dr. Simon	100
2	Mark	54	Male	Pathology	Dr. Greg	150
3	Alan	65	Male	Cardiology	Dr. Simon	100
4	Royce	21	Male	Pathology	Dr. Greg	150
5	Elizabeth	27	Female	Cardiology	Dr. Simon	100

The above table is in 1NF since the column values are atomic. There is no existence of repeated column sets, and there is a primary key that can identify each of the records. The primary key is PatientID.

Some columns, like the last three columns, contain redundant values. Therefore, it is not in 2NF yet. The redundant columns should be grouped together to form a new table. The above table can be divided into two new ones: The Patient Table and the Department Table. Patient Table will contain the following columns: PatientID, PatientName, PatientAge, and PatientGender. The Department Table will have an ID column, and the DepName, DepHead, and NoOfBeds columns.

Department Table

ID	DepName	DepHead	NoOfBeds
1	Cardiology	Dr. Simon	100

2	Pathology	Dr. Greg	150
---	-----------	----------	-----

Now that we are creating two tables, the third condition of 2NF requires that we create a relationship between the two tables using a foreign key. Here, we know that one department can have many patients. This means that we should add a foreign key column to the Patient table that refers to the ID column of the Department table. The Department table will not change from what is shown above, and the updated Patient table will look like this:

Patient Table

PatientID	PatientName	PatientAge	PatientGender	DepID
1	Jane	45	Female	1
2	Mark	54	Male	2
3	Alan	65	Male	1
4	Royce	21	Male	2
5	Elizabeth	27	Female	1

In the above table, the DepID column is the foreign key column, and it references the ID column of the Department Table.

Third Normal Form (3NF)

You need to be careful and know when a database is in 2NF. To be considered 3NF, it must adhere to the following conditions:

1. Should Satisfy All the Rules of the Second Normal Form
2. All Columns in the Tables Fully Depend on the Primary Key Column

Take a look at the following table:

PatientID	Patient Name	Patient Age	Patient Gender	DepName	DepHead	NoOfBeds
1	Jane	45	Female	Cardiology	Dr. Simon	100

2	Mark	54	Male	Pathology	Dr. Greg	150
3	Alan	65	Male	Cardiology	Dr. Simon	100
4	Royce	21	Male	Pathology	Dr. Greg	150
5	Elizabeth	27	Female	Cardiology	Dr. Simon	100

In the above table, the DepHead and NoOfBeds columns are not fully dependent on the primary key column, PatientID. They are also dependent on the DepName column. If the value in the DepName column changes, the values in the DepHead and NoOfBeds columns also change. A solution to this problem is that all the columns that depend on some column other than the primary key column should be moved into a new table with the column on which they depend. After that, the relation between the two tables can be implemented via foreign keys as shown below:

Patient Table

PatientID	PatientName	PatientAge	PatientGender	DepID
1	Jane	45	Female	1
2	Mark	54	Male	2
3	Alan	65	Male	1
4	Royce	21	Male	2
5	Elizabeth	27	Female	1

Department Table

ID	DepName	DepHead	NoOfBeds

1	Cardiology	Dr. Simon	100
2	Pathology	Dr. Greg	150

Boyce-Codd Normal Form (BCNF)

This normal form, also known as 3.5 Normal Form, is an extension of 3NF. It is considered to be a stricter version of 3NF, in which records within a table are considered unique. These unique values are based upon a composite key, which is created by a combination of columns. Though, this does not always apply or need to be applied for every table because sometimes the data in a table does not need to be normalized up to BNCF.

Fourth Normal Form (4NF)

For this step, the previous form, BCNF, must be satisfied. This particular form deals with isolating independent multi-valued dependencies, in which one specific value in a column has multiple values dependent upon it. You'd most likely see this particular value several times in a table.

Fifth Normal Form (5NF)

This is the last step in normalization. The previous normal form, 4NF, must be satisfied before this can be applied. This particular form deals with multi-valued relationships being associated to one another, and isolating said relationships.

CHAPTER 13: SECURITY

Databases are containers that maintain all kinds of data, corporate secrets, sensitive employee information, and many other types of information that need proper security and access control. Many companies today are using Microsoft's Active Directory to manage users and sort them into access profiles using a group policy process. Through this, employees of a company are assigned group permissions based on their job title, and within those group permissions, more individualized permissions are created depending on an employee's rank within a group. SQL does interact with Active Directory for access control, but it does not provide the internal security regarding authorization. The SQL application itself provides these services.

Components of Database Security

There are four main components of database security: authentication, encryption, authorization, and access control.

Authentication

Authentication pertains to validating whether a user has permission to access any resources of the system. The most common method of authentication by far is the username and password, which verify the credentials of a potential user. Single sign-on systems also exist, which use certificate authentication that the user will not have a chance to interact with directly. The end user's system is prepared with information that provides authentication automatically without prompt.

Encryption

Corporations go to great lengths to ensure that system access is authenticated and appropriately authorized. Encryption strengthens this access control by scrambling data into indecipherable gibberish to any potential interceptors of the transmitted data. Microsoft SQL Server uses RSA encryption to protect data. RSA is a data encryption algorithm that uses a layered hierarchical structure along with key management to secure data.

Authorization

Authorization is the process that determines what resources within a system an authenticated user can access. Once a client has provided acceptable credentials, the next step is to decide which entities the subject has permission to access or modify.

Access Control

Lastly, SQL uses change tracking to maintain a log of all the actions of potentially unauthorized users. It is also possible to track the activities of all authorized users, but that isn't a part of the security functionality of change tracking. Power-users or super-users with elevated permissions may have access to all systems, but that does not authorize them as users of the database. Tracking changes

protects a system from operations performed by users with elevated credentials.

Three-Class Security Model

SQL uses a security model comprised of three classes that interplay with each other: principals, securables, and permissions. Principals have permission to access specific objects. Securables refer to resources within a database that the system regulates access to. Permission refers to the right to view, edit, or delete securables, and this access is pre-defined.

This security model primarily belongs to Microsoft SQL server, but there are equivalents within other SQL management products like MySQL, DB2, or 11G. The theories behind modeling access control are widespread across the IT industry and cover much more than database security. These same principals are behind nearly all enterprise applications that house sensitive data.

Security is a profound subject, and there are comprehensive books available on securing enterprise resources. The primary elements covered briefly above are an excellent base to expand upon the ideas of access roles and schema related to security.

Schemas

In relation to security, the schema defines ownership of resources and identifies principals based on the user's relation to the owner. Microsoft defines a schema as “a group of database objects under the ownership of a single individual and together they create what is known as single namespace.” The single namespace refers to a limitation that does not allow two tables that are contained in one schema to contain similar names. Data consistency and referential ease are the guiding ideas behind SQL's design, and this is the reason behind the limitation. Principals can be for a single user or single login, or a group principal. Multiple users sharing one role are grouped using group policies, and all can cooperatively own a schema or many schemas. Transferring a schema from one principal to another is possible and does not require renaming unless the new owner maintains a schema that would create a duplicate name. There are T-SQL statements for managing schema, but a majority of this work belongs to database administrators, not the principals of a database.

Server Roles

Roles are another layer of security for identifying access dependent upon title and responsibilities. There are many different kinds of roles available. SQL comes with fixed server roles and fixed database roles which provide implicit permissions. It is also possible to create customized application roles, user-defined server roles, and user-defined database roles. The following is a list of Microsoft's fixed server roles:

sysadmin	Members of the sysadmin fixed server role can perform any activity in the server.
serveradmin	Members of the serveradmin fixed server role can change server-wide configuration options and shut down the server.
securityadmin	Members of the securityadmin fixed server role manage logins and their properties. They can GRANT, DENY, and REVOKE server-level permissions. They can also GRANT, DENY, and REVOKE database-level permissions if they have access to a database. Additionally, they can reset passwords for SQL Server logins. IMPORTANT: The ability to grant access to the Database Engine and to configure user permissions allows the security admin to assign most server permissions. The securityadmin role should be treated as equivalent to the sysadmin role.
processadmin	Members of the processadmin fixed server role can end processes that are running in an instance of SQL Server.
setupadmin	Members of the setupadmin fixed server role can add and remove linked servers by using Transact-SQL statements. (sysadmin membership is needed when using Management Studio.)
bulkadmin	Members of the bulkadmin fixed server role can run the BULK INSERT statement.
diskadmin	The diskadmin fixed server role is used for managing disk files.
dbcreator	Members of the dbcreator fixed server role can create, alter, drop, and restore any database.

Roles are very important to database security. All the other security functions are built on top of roles. Authentication determines who can access databases; authorization determines which principals have access to which schema. Encryption protects data from interception by those external to the company, as well as potential internal hazards. Sensitive data that leaks internally or is intercepted unintentionally is more dangerous than external threats. Roles maintain the minutiae of which users are allowed to perform any given operation within a database. Microsoft teaches security principals that provide a user with the least amount of access possible to perform their duties. This theory prevents users from having access to resources that they are not trained to use. The same guiding ideas are used with SQL. The purpose of roles is to limit the amount of access an employee has to a database that does not pertain to their specific responsibilities within the schema. Owners are considered “sysadmins,” but all other users of a database have specialized functions based on their training and expertise. Database administrators are usually part of the IT department, but database

owners are rarely within the same department as the database administrators. Thus, there is a level of autonomy that is necessary because administrators are responsible for maintaining the structure and consistency of dozens, and even hundreds, of databases. These rules and roles are all for the end goal of keeping consistent and referential data.

There are a number of different server roles available within the SQL Server that you can use to manage permissions at the instance level (“Server-Level Roles”, 2017). Server roles are like groups in Windows and help you manage permissions on a server. They are:

- sysadmin – any user of this group can perform any action within the instance of SQL Server
- bkladmin – allows users to import data from a file using BULK INSERT statement
- dcreator – allows members to create new databases within the instance
- diskadmin – allows members to manage backup devices in SQL Server
- processadmin – members are allowed to kill running processes or stop instances
- public – all logins added to this group are used for internal operations like authentication
- securityadmin – members can manage logins at the instance level, e.g. assign permissions
- serveradmin – combines both diskadmin and processadmin, and can shut down the instance (i.e. the option of no checkpoint)
- setupadmin – members can create and manage linked servers

You can also create your own server roles with a specific number of permissions. To add a new server role, right click on Security in the object browser, click New and then click Server Role. Enter a server role name and then tick the boxes for explicit permissions.

Logins

The security hierarchy starts at the Windows domain (or all accounts) and flows down to Windows groups and Windows users, down to the local server, down to the database, and, finally, right down to the object (e.g. a table) level. The model, as stated earlier, is based on three concepts: principals – entities which request

that permissions are granted/denied/revoked, securables – objects which can have permissions granted on them, and permissions – which define what you are allowed to do. Essentially, you permit a permission on a securable to a principal.

You can create a group that groups of users belong to and have a designated level of access to the database. Whenever you originally create a new database, only the database owner or system administrator has complete access to it. It's up to them to create groups and specify access.

Mixed Mode Authentication

In addition to Windows authentication to login, there is also mixed mode authentication where you can connect to the database through a secondary tier of authentication. These users and passwords are stored on the database. Also, when you enable mixed mode authentication, a special user called an SA (or System Administrator) is created who has complete administrator rights. This can be a security risk, as hackers will try to target this password first. The best option is to rename this account and ensure it has a very strong password. To rename the account, create a new query and execute the following:

```
ALTER LOGIN sa WITH NAME = SQLSERVERADMINSA
```

You can give the SA account any name of your choosing, but this simple step will really help the database security. The best practice is to stick to Windows authentication. You can change to Windows authentication by right-clicking the server in object explorer in SSMS, and then selecting Properties. You can choose the server authentication here and restart the SQL Server to bring the changes into effect.

Database Roles

While server roles manage permissions at the server level, database roles can group principals together to assign common permissions. The main database roles are:

- db_accessadmin – can add and remove database users from the database
- db_backupoperator – gives users the permissions they need to back up

the database

- db_datareader – can run SELECT statements against any table in the database
- db_datawriter – can perform Data Manipulation Language statements against any table in the database
- db_denydatareader – denies the SELECT permission against every table in the database
- db_denydatawriter – denies its members the permissions to perform DLM statements against every table in the database
- db_ddladmin – able to run CREATE, ALTER, and DROP statements against any object in the database
- db_owner – can perform any action within the database that has not been specifically denied
- db_securityadmin – can grant, deny, and revoke a user's permissions to securables; can add or remove role memberships, with the exception of the db_owner role

To create a database role in SSMS, click the + on the left of your database, then right-click on Security → New → Database Role.

Encryption

Encryption is another option you can utilize to ensure the security of the database, and while it adds another layer of security to your database, it does not replace the need for controlling access to the database. There are some negative effects to encryption, like a reduction in performance, so the best option is to implement encryption when the need arises. It is not a mandatory task for a DBA to perform on each and every database.

There are several ways to encrypt data. The weakest form is to use a symmetric key that employs the usage of the same key aimed to perform encryption and decryption. This has the least performance overhead.

An asymmetric key includes the usage of a pair of keys, one for encryption and the other for decryption. To encrypt the data, you use the private key, and to decrypt the data you use the public key. This method is strong but the keys are slow.

A certificate is issued by a recognized Certificate Authority. It uses an asymmetric key and provides a digitally signed statement that is associated with an individual or device.

Master Keys

SQL Server uses service master keys as the root of SQL Server encryption. A master key is automatically created once a new instance is built, and you can use it to encrypt database master keys, credentials, and linked server passwords. You can authorize a user to unlock subsequent keys. The key is stored in the Master database and there is always only one per instance. To generate a new master key, execute the following in a new query:

```
ALTER SERVICE MASTER KEY REGENERATE
```

It's important that you make a backup (and choose a suitable password). You can do so with the following:

```
BACKUP SERVICE MASTER KEY
```

```
TO FILE = 'c:\sqlServerKeys\serviceMasterKey'
```

```
ENCRYPTION BY PASSWORD = 'Password1'
```

To restore to a previous master key, you use the following:

RESTORE SERVICE MASTER KEY

FROM FILE = 'c:\sqlServerKeys \serviceMasterKey'

DECRYPTION BY PASSWORD = 'Password1'

There are database master keys that should also be backed up. To create a database master key, you use the following:

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Password1'

To back up a database master key you use the following:

BACKUP MASTER KEY TO FILE = 'c:\sqlServerKeys \serviceMasterKey'

ENCRYPTION BY PASSWORD = 'Password1';

To restore a database master key, you use:

RESTORE MASTER KEY

FROM FILE = 'c:\sqlServerKeys \serviceMasterKey'

DECRYPTION BY PASSWORD = 'Password1'

ENCRYPTION BY PASSWORD = 'Password1'

Transparent Data Encryption (TDE)

Data pages and log files of a database can be encrypted by storing the key (or database encryption key) within the boot record of the database. This works by encrypting pages before writing them to the disk, and they are decrypted before being read into memory. This approach doesn't cause unnecessary bloat, and the performance overhead is significantly less than encrypting individual cells in a database. It is also transparent to other applications, so developers are not required to alter their code.

In order to implement transparent data encryption, you first need to create a database master key (as shown above). Once you have a key, you are then in a position to create a certificate. You need to use the database master key to encrypt the certificate; using just a password would result in an error. Then you need to create the database encryption key object and encrypt this using the certificate.

Once you implement a TDE for a database, the database will continue to be accessible. However, maintenance operations are stopped and you won't be able

to drop a file, drop a filegroup, drop a database, detach a database, take a database offline, or set the database as read-only.

The following uses the master key, creates the service master key, and creates a server certificate. Right-click on a database, and in a new query, type and execute the following:

```
USE Master
```

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Password1'
```

```
CREATE CERTIFICATE TDECertificate WITH SUBJECT = 'Certificate For TDE'
```

You can now encrypt your database. If you right-click on the same database as you used above and select Tasks → Manage Database Encryption, you will get the database encryption wizard. You can select the Certificate for TDE that you just created in the Use Server Certificate drop down box and click OK.

Once a database is encrypted, you should establish that there are appropriate permissions associated with the SQL Server service account. Otherwise, you will receive an error when trying to drop a table, for example.

CHAPTER 14: SQL INJECTIONS

SQL injection is a special type of hacking method nowadays. By using this method, a hacker can access the database if the site is vulnerable, and get all details from the database (“SQL Injection”). The database could even be destroyed.

Whenever someone goes into a website, data is taken from their computer and sent through the site. You are able to take this same data and place it in the database for SQL. However, it is risky to do this due to the fact that you will be leaving yourself open to what is known as SQL injection, which can end up wiping out all of the hard work that you have put into your SQL script.

An injection typically occurs at the point in time that you ask a user to place some type of data into a prompt box before they are able to continue. However, you will not necessarily get the information that you want. Instead, you could end up getting a statement that runs through your database, and you won’t know that this has occurred.

Users cannot be trusted to give you the data that you are requesting, so you need to make sure that the data that they enter is looked at before it is sent to your database for validation. This is going to help secure your database from any SQL injection statements that may occur. Most of the time, you will use pattern matching to look at the data before you decide to send it to your main database.

Your function calls are used when you try to pull a particular record off of the database from the table requested when you are working with that title for that row. The data is usually going to end up matching what you have received from the user, so you are able to keep your database safe from SQL injection statements.

With MYSQL, queries are not allowed to be carried or stacked into a single call function. This helps in keeping calls from failing due to the queries being stacked.

Extensions such as SQLite, however, do allow for your queries to be stacked as you do your searches in that one string. This is where safety issues come into play with your script and your database.

How Do They Work?

Consider the statement given below, which shows how to authenticate a user in a web application:

```
SELECT * FROM users WHERE username='username'  
      AND password='password';
```

The username and password added in double quotes represent the username and the password entered by a user. If, for example, someone enters username alphas, and the password pass123, the query will be:

```
SELECT * FROM users WHERE username='alphas' AND password='pass123';
```

Suppose the user is an attacker, and instead of entering a valid username and password in the fields, he enters something such as ' OR 'x'='x'.

In such a case, the query will evaluate to the following:

```
SELECT * FROM users WHERE username="' OR 'x'='x'  
      AND password="' OR 'x'='x';
```

The above shows that the statement has tested to a valid SQL statement. The expression WHERE 'x'='x' will always test to be true. This means that the above query will return all the rows in the user's table. With this, the attacker will then be able to log into the database and do whatever they want to do. This shows how easy it is for an attacker to gain access to the database by the use of a dirty yet simple trick.

The user's table may also be large and loaded with millions of records. In such a case, the above query can result in a denial of service (DoS) attack. This is because the system resources may be overloaded, making the application unavailable to the users.

In the above case, we have seen what an SQL injection can do to your database table for a select query. This can even be dangerous in a case where you have a DELETE or UPDATE query. The attacker may delete all the data from your table or change all its rows permanently.

Preventing an SQL Injection

Escaped characters are meant to be handled inside of the artificial dialects such as Perl or PHP. However, with MySQL, there is an extension that enables you to use PHP so that the proper function is used with the escaped characters. This extension makes these characters unique to just MySQL.

LIKE Quandary

When dealing with a LIKE quandary, you have to have an escape method in order to change the characters that the user inserts into your prompt box that turn out to be literals. The addslashes function allows you to ensure that you are specifying the range of characters that is needed by the system in order to escape.

Hacking Scenario

Google is one of the best hacking tools in the world through the use of the inurl: command to find vulnerable websites.

For example:

inurl:index.php?id=

inurl:shop.php?id=

inurl:article.php?id=

inurl:pageid=

To use these, copy one of the above commands and paste it in the Google search engine box. Press Enter.

You will get a list of websites. Start from the first website and check each website's vulnerability one by one.

CHAPTER 15: FINE-TUNE YOUR INDEXES

When you are using SQL, you will want to become an expert on your database. This, however, is not going to happen overnight. Just like with learning how to use the system, you will need to invest time and effort to learn the important information and have the proper awareness of how the database works. You should also ensure that you are equipped with the right education targeted at working with the database because you never know what may happen in future.

In order to make your education more streamlined when you are learning the ins and outs of the database, here are some helpful insights:

1. When using the database, you need to work with the 3NF design.
2. Numbers compare to characters differently, and you could end up downgrading your database's performance, so do not change numbers unless you absolutely have to!
3. With the choose statement, you will only have data to display on the screen. Ensure that you avoid asterisks with your choose statement searches to avoid loading data that you do not need at that time.
4. Any records should be constructed carefully, and only for the tables that require them. If you do not intend to utilize the table as much, then it does not need to have an index. Essentially, you should attempt to save space on the disk, and if you are creating an index for each table, you are going to run out of room.
5. A full table scan happens when no index can be found on that table. You can avoid doing this by creating an index specifically for that row rather than the entire table.
6. Take precautions when using equality operators, especially when dealing with times, dates, and real numbers. There is a possibility that differences will occur, but you are not necessarily going to notice these differences right away. Equality operators make it almost impossible to get exact matches in your queries.
7. Pattern matching can be used, but use it sparingly.
8. Look at how your search is structured, as well as the script that is being used with your table. You can manipulate the script of your table to have a faster response time, as long as you change everything about the table and not just part of it.

9. Searches will be performed regularly on the SQL. Stick to the standard procedures that work with a large group of statements rather than small ones. The procedures have been put into place by the database before you even get the chance to use it. The database is not like the search engine though; the procedure is not going to be optimized before your command is executed.
10. The OR operator should not be used unless necessary. This operator will slow your searches.
11. Remove any records you currently have to allow you to optimize larger batches of data. It is an excellent idea to think of the history of the table in millions of different rows, and you are probably going to require multiple records to cover the entire table, which is going to take up space on the disk. While records will get you the information you want faster, after the batch has been loaded, the index is going to slow the system down due to the fact that it is now in the way.
12. Batch compromises require that you use the commit function. You should use this function after you construct a new record.
13. Databases have to be defragmented at least once a week to ensure everything is working properly.

SQL Tuning Tools

The Oracle program grants you access to various tools to use if you want to tune your database or its performance. Two of the most popular tools it offers are:

1. TKProf — lets you measure how the database performs over a certain period of time based on every statement you enter into SQL to be processed
2. EXPLAIN PLAN — shows you the path that is followed to ensure that statements are carried out as they are meant to be

Use SQL*Plus Command to measure the time that passes between each search on your SQL database.

CHAPTER 16: DEADLOCKS

In most cases, multiple users access database applications simultaneously, which means that multiple transactions are being executed on a database in parallel. By default, when a transaction performs an operation on a database resource such as a table, it locks the resource. During that period, no other transaction can access the locked resource. Deadlocks occur when two or more processes try to access resources that are locked by the other processes participating in the deadlock (Steynberg, 2016).

Deadlocks are best explained with the help of an example. Consider a scenario where some transactionA has performed an operation on tableA and has acquired a lock on the table. Similarly, there is another transaction named transactionB that is executing in parallel and performs some operation on tableB. Now, transactionA wants to perform some operation on tableB, which is already locked by transactionB. In addition, transactionB wants to perform an operation on tableA, but that table is already locked by transactionA. This results in a deadlock since transactionA is waiting on a resource locked by transactionB, which is waiting on a resource locked by transactionA.

For the sake of this chapter, we will create a dummy database. This database will be used in the deadlock example that we shall in next section. Execute the following script:

```
CREATE DATABASE dldb;
```

```
GO
```

```
USE dldb;
```

```
CREATE TABLE tableA
```

```
(  
id INT IDENTITY PRIMARY KEY,  
patient_name NVARCHAR(50)  
)
```

```
INSERT INTO tableA VALUES ('Thomas')
```

```
CREATE TABLE tableB
```

```
(  
id INT IDENTITY PRIMARY KEY,  
patient_name NVARCHAR(50)  
)  
  
INSERT INTO table2 VALUES ('Helene')
```

The above script creates a database named “dldb.” In the database, we create two tables, tableA and tableB. We then insert one record into each table.

Deadlock Analysis and Prevention

In the above section, a deadlock was generated. We understand the processes involved in the deadlock. In real-world scenarios, this is not the case. Multiple users access the database simultaneously, which often results in deadlocks. However, in such cases we cannot tell which transactions and resources are involved in the deadlock. We need a mechanism that allows us to analyze deadlocks in detail so that we can see what transactions and resources are involved, and determine how to resolve the deadlocks. One such way is via SQL Server error logs.

Reading Deadlock Info via SQL Server Error Log

The SQL Server provides minimal information about the deadlock. You can get detailed information about the deadlock via SQL Server error log. However, to log deadlock information to the error log, first you have to use a trace flag 1222. You can turn trace flag 1222 on global as well as session level. To turn on trace flag 1222, execute the following script:

```
DBCC Traceon(1222, -1)
```

The above script turns trace flag on global level. If you do not pass the second argument, the trace flag is turned on session level. To see if the trace flag is actually turned on, execute the query below:

```
DBCC TraceStatus(1222)
```

The above statement results in the following output:

TraceFlag	Status	Global	Session
1222	1	1	0

Here, Status value 1 shows that trace flag 1222 is on. The 1 in the Global column implies that the trace flag has been turned on globally.

Now, try to generate a deadlock by following the steps that we performed in the last section. The detailed deadlock information will be logged in the error log. To view the SQL Server error log, you need to execute the following stored procedure:

executesp_readerrorlog

The above stored procedure will retrieve a detailed error log. A snippet of this is shown below:

458	2017-11-01 15:51:47.810	spid58	Parallel redo is started for database 'test' with work...
459	2017-11-01 15:51:47.860	spid58	Parallel redo is shutdown for database 'test' with wo...
460	2017-11-01 15:51:55.320	spid13s	deadlock-list
461	2017-11-01 15:51:55.320	spid13s	deadlock victim=process1fcf9514ca8
462	2017-11-01 15:51:55.320	spid13s	process-list
463	2017-11-01 15:51:55.320	spid13s	process id=process1fcf9514ca8 taskpriority=0 log...
464	2017-11-01 15:51:55.320	spid13s	executionStack
465	2017-11-01 15:51:55.320	spid13s	frame procname=ad hoc line=2 stmtstart=58 stmt...
466	2017-11-01 15:51:55.320	spid13s	unknown
467	2017-11-01 15:51:55.320	spid13s	frame procname=ad hoc line=2 stmtstart=4 stmt...

Your error log might be different depending upon the databases in your database. The information about all the deadlocks in your database starts with log text “deadlock-list.” You may need to scroll down a bit to find this row.

Let’s now analyze the log information that is retrieved by the deadlock that we just created. Note that your values will be different for each column, but the information remains the same.

ProcessInfo	Text
spid13s	deadlock-list
spid13s	deadlock victim=process1fcf9514ca8
spid13s	process-list
spid13s	process id=process1fcf9514ca8 taskpriority=0 logused=308 waitre waittime=921 ownerId=388813 transactionname: XDES=0x1fcf8454490 lockMode=X schedulerid=3 kpid=1968 trancount=2 lastbatchstarted=2019-05-27T15:51:54.380 lastattention=1900-01-01T00:00:00.377 clientapp=Microsoft hostname=DESKTOP-GLQ5VRA hostpid=968 loginname=DESK (2) xactid=388813 currentdb=8 lockTimeout=4294967295 clientop
spid13s	executionStack

spid13s	frame procname=adhoc line=2 sqlhandle=0x0200000014b61731ad79b1eec6740c98aab3ab91bd3
spid13s	unknown
spid13s	frame procname=adhoc line=2 sqlhandle=0x0200000080129b021f70641be5a5e43a1ca1ef67e972
spid13s	unknown
spid13s	inputbuf
spid13s	UPDATE tableA SET patient_name = 'Thomas - TransactionB'
spid13s	WHERE id = 1
spid13s	process id=process1fcf9515468 taskpriority=0 logused=308 waitr waittime=4588 ownerId=388767 transactionname: XDES=0x1fcf8428490 lockMode=X schedulerid=3 kpid=11000 trancount=2 lastbatchstarted=2019-05-27T15:51:50.710 lastattention=1900-01-01T00:00:00.710 clientapp=Microsoft hostname=DESKTOP-GLQ5VRA hostpid=1140 loginname=DESK (2) xactid=388767 currentdb=8 lockTimeout=4294967295 clientop
spid13s	executionStack
spid13s	frame procname=adhoc line=1 sqlhandle=0x02000000ec86cd1dbe1cd7fc97237a12abb461f1fc27c
spid13s	unknown
spid13s	frame procname=adhoc sqlhandle=0x020000003a45a10eb863d6370a5f99368760983cacbf
spid13s	unknown

spid13s	inputbuf
spid13s	UPDATE tableB SET patient_name = 'Helene - TransactionA'
spid13s	WHERE id = 1
spid13s	resource-list
spid13s	keylockhobtid=72057594043105280 objectname=dlldbo.tableAindexname=PK__tableA__3213E83F associatedObjectId=72057594043105280
spid13s	owner-list
spid13s	owner id=process1fcf9515468 mode=X
spid13s	waiter-list
spid13s	waiter id=process1fcf9514ca8 mode=X requestType=wait
spid13s	keylockhobtid=72057594043170816 objectname=dlldbo.tableBindexname=PK__tableB__3213E83F associatedObjectId=72057594043170816
spid13s	owner-list
spid13s	owner id=process1fcf9514ca8 mode=X
spid13s	waiter-list
spid13s	waiter id=process1fcf9515468 mode=X requestType=wait

The deadlock information logged by the SQL server error log has three main parts.

1. The Deadlock Victim

As mentioned earlier, to resolve a deadlock, the SQL server selects one of the processes involved in the deadlock as a deadlock victim. Above, you can see that

the ID of the process selected as the deadlock victim is process1fcf9514ca8. You can see this value highlighted in gray in the log table.

2. Process List

The process list is the list of all the processes involved in a deadlock. In the deadlock that we generated, two processes were involved. In the processes list you can see details of both of these processes. The ID of the first process is highlighted in red, and the ID of the second process is highlighted in green. Notice that, in the process list, the first process is the process that has been selected as deadlock victim, too.

Apart from the process ID, you can also see other information about the processes. For instance, you can find login information of the process, the isolation level of the process, and more. You can even see the script that the process was trying to run. For instance, if you look at the first process in the process list, you will find that it was trying to update the patient_name column of tableA when the deadlock occurred.

3. Resource List

The resource list contains information about the resources taking place in the event of the deadlock. In our example, tableA and tableB were the only two resources during the deadlock. The tables are highlighted in blue in the resource list of the log in the table above.

Some Tips for Avoiding Deadlock

From the error log, we can get detailed information about the deadlock. However, we can minimize the chance of deadlock occurrence if we follow these tips:

- Execute transactions in a single batch and keep them short
- Release resources automatically after a certain time period
- Sequential resource sharing
- Don't allow user to interact with the application when transactions are being executed

CHAPTER 17: DATABASE ADMINISTRATION

Once you have your database up and running with tables and queries, it is up to you to keep the production database running smoothly. The database will need regular checks to ensure that it continues to perform as originally intended. If a database is poorly maintained, it can easily result in a connected website performing poorly, or it could result in down time or even data loss. There is usually a person, known as a Database Administrator or DBA, designated to look after the database. However, it is usually someone who is not a DBA who needs help with the database.

There are a number of different tasks that you can perform when carrying out maintenance, including the following:

- **Database Integrity:** When you check the integrity of the database, you are running checks on the data to make sure that both the physical and logical structure of the database is consistent and accurate.
- **Index Reorganization:** Once you start to insert and delete data on your database, there is going to be fragmentation (or a scattering) of indexes. Reorganizing the index will bring everything back together again and increase speed.
- **Rebuild Index:** You don't have to perform an index reorganization; you can drop an index and then recreate it.
- **Database Backup:** One of the most important tasks to perform. There are a number of different ways in which you can back up the database. These include: full - backs up the database entirely, differential - backs up the database since the last full backup, and transaction log - only backs up the transaction log.
- **Check Database Statistics:** You can check the statistics of the database that are kept on queries. If you update the statistics, which can get out of date, you can help aid the queries that are being run.
- **Data and Log File:** In general, make sure the data and log files are kept separate from each other. These files will grow when your database is being used, and it's best to allocate them an appropriate size going forward (and not just enable them to grow).

Depending on your database, some tasks may be more useful than others. Apart

from database backup, which is probably mandatory if it's in production, you can pick through the other tasks depending on the state of the database.

For example, should the fragmentation of the database be below 30%, then you can choose to perform an index reorganization. However, if the database fragmentation is greater than 30%, then you should rebuild the index. You can rebuild the index on a weekly basis or more often, if possible.

You can run a maintenance plan on the SQL Server via its Server Agent depending on database requirements. It's important to set the times appropriately, not when your application is expected to be busy. You can choose a time, or you can run it when the server CPU is not busy. Choosing to run when the server is not busy is a preferred option for larger databases rather than selecting a particular time, as there may be no guaranteed time when the CPU will be idle. However, it is usually only a concern if your application is quite big and has a lot of requests.

When you do rebuild the indexes, it is important that you have the results sorted in tempdb. When using tempdb, the old indexes are kept until new ones are added.

Normally, rebuilding the indexes uses the fixed space that was allocated to the database. Therefore, if you run out of disk space, then you would not be able to complete the index rebuilding. It's possible to use the tempdb and not have to increase the database disk size. The database maintenance can be run either synchronously (at the same time) or asynchronously (once task has been completed). However, you should ensure the tasks are running in the right order.

Setting up a Maintenance Plan in SQL Server

To set up a maintenance plan in SQL Server, you must first get the server to show advanced options. This is achieved through executing the following code in SQL Server as a new query:

```
sp_configure 'show advanced options', 1
```

```
GO
```

```
RECONFIGURE
```

```
GO
```

```
sp_configure 'Agent XPs', 1
```

```
GO
```

```
RECONFIGURE
```

```
GO
```

The SQL Server will now display the advanced options. Left-click the + icon to the left of Management, which is on the left-hand side of SQL Server Management Studio. Now, left-click Maintenance Plans and then right-click Maintenance Plans. Select New Maintenance Plan Wizard.

Enter an appropriate maintenance plan name and description. From here, you can either run one or all tasks in one plan and have as many plans as you want. After you have assigned a name, choose Single Schedule and click Next.

You will see a number of options that you can pick for your maintenance, including:

- Checking your Database Integrity
- Shrinking the Database
- Reorganizing Index
- Rebuilding the Index
- Updating the Statistics
- Clean up History
- Executing SQL Server Agent Job
- Back Up – full, differential or transaction log
- Maintenance Cleanup Task

Select which you want to perform (in this example, select all). This wizard will bring you through each of the items you have selected to fine-tune them.

Once you select the items you want in your plan, click Next. You can now rearrange them in the order that you want them to complete. It's best to have Database Backup first in case of power failure, so select it and move it to the top of the list. Click Next.

Define Backup Database (Full) Task

This screen will give you the freedom to pick which full database backup you wish to perform on. The best practice is to keep one plan per database; select one database and select Next.

Define Database Check Integrity Task

The integrity task is an SQL Server command that aims at inspecting the database's integrity to make sure that everything is stable. Select a database and click Next.

Define Shrink Database Task

You can now configure to shrink the database in order to free up space in the next screen. This will only shrink space if available, but should you need space in the future, you will have to allocate it again. However, this step will help backup speeds. Most developers don't use this feature that much. Click Next after selecting a database to shrink.

Define Reorganize Index Task

The next screen is the Define Reorganize Index Task screen. When you add, modify, and delete indexes you will, like tables, need to reorganize them. The process is the same as a hard disk, where you have fragmented files and space scattered across the disk. Perform this task once per week for a busy database. You can choose to compact large object, which compacts any index that has large binary object data. Click Next to proceed to the next screen.

Define Rebuild Index Task

This screen covers individual index rows and involves either reorganizing or reindexing. Doing both together in one plan is pointless. Depending on your fragmentation level, pick one or the other. In this example, select your database and sort results in tempdb. Click Next to proceed.

Define Update Statistics Task

The update statistics task helps the developer keep track of data retrieval as its created, modified, and deleted. You can keep the statistics up-to-date by performing this plan. Statistics for both indexes and individual columns are kept. Select your database and click Next to proceed.

Define History Cleanup Task

You should now see this screen, which specifies the historical data to delete. You can specify a shorter time frame to keep the backup and recovery, agent job history, and maintenance place on the drop down. Click Next to proceed.

Define Backup Database (Differential) Task

This screen allows you to back up every page in the database that has been altered since the previous or last full backup. Select a database you wish to use and click Next.

Define Backup Database (Transaction Log) Task

The transaction log backup backs up all the log records since the last backup. You can choose a folder to store it in. Performing this type of backup is the least resource-intensive backup. Select a database and storage location and click Next.

Define Execute SQL Server Agent Job Task

The SQL Server Agent Job Task deals with jobs that are outside the wizard. For example, it can check for nulls, check whether the database meets specified standards, and more. Any jobs that are specified in SQL Server Agent Job Task

are listed here. Click Next to proceed.

Define Maintenance Cleanup Task

This screen defines the cleanup action of the maintenance task. This ensures that files are not taking up unnecessary space, and you can specify where to store them. You can also delete specific backup files. Click Next to proceed.

Report Options

The next screen covers where you want to store the report of the maintenance plan. Make a note of where you are going to store it. You need to have email setup on SQL Server in order to email it. Click Next to proceed.

Complete the Wizard

The final screen is a complete review of the wizard. You can review the summary of the plan and which options were selected. Clicking Finish ends the wizard and creates the plan. You should now see a success screen with the completed tasks.

Running the Maintenance Plan

Once you successfully complete the maintenance wizard, the next step is to run the plan you created. In order to get the plan to run, you need to have the SQL Server Agent running. It is visible two below Management on SQL Server Management Studio. You can left-click SQL Server Agent, then right-click and select Start.

Alternatively, you can press the Windows key and press the letter R, then type services.msc and hit Enter. Once Services appears, scroll down and look for the SQL Server Agent (MSSQLSERVER). This book instructed you to install SQL Server Express, but you can select the other versions like (MSSQLSERVER) if you installed that. Left-click it, then right-click it and select Start.

You can go back to SSMS and right-click on the maintenance plan you created under maintenance plans, then select Execute. This will now run your plan. Upon successful completion of the plan, click OK and close the dialogue box. You can view the reports by right-clicking the maintenance plan you created and selecting View History. On the left-hand side are all the different plans in SQL Server, while the results of the specific plan are on the right.

Emailing the Reports

A lot of DBAs like to get their database reports via email. What you need to do is set up a database mail before you can fire off emails, and then set up a Server agent to send the emails.

Configuring the Database Mail

The first step is to right-click Database mail in SSMS and select Configure Database Mail. A wizard screen will appear; click Next. Now select, the first choice—Set Up Database Mail—and click Next. Enter a profile name and, if you want, an optional description of the profile. Now, click on the Add button to the right.

This will bring you to an add New Database Mail Account – SMTP. You need to enter the SMTP details for an email account. You may want to set up a new email account for this service. You can search online for SMTP details, and Gmail works quite well (server name: smtp.gmail.com, port number 587, SSL required,

tick basic authentication, and confirm password). Click on OK. Click Next, and select Public; it is important to do this so it can be used by the rest of the database. Set it as Default Profile, click Next, and click Next again. You should now get a success screen. Click Close.

SQL Server Agent

To send off the database email, you need to set up a Server Agent. Start by right-clicking on SQL Server Agent → New → Operator. Give the operator a name like Maintenance Plan Operator, enter in the email address to which you want the report delivered, and click OK.

Now, right-click the maintenance plan that you have successfully executed and select Modify. The maintenance plan design screen will appear on the right-hand side, where you can see some graphics of the tasks completed in it. Now, click on Reporting and Logging; it is an icon situated on the menu bar of the design plan, to the left of Manage Connections.

The Reporting and Logging window will appear. Choose the option to Send Report to an Email Recipient, and select the maintenance plan operator you just created. The next time you run the plan, an email will be sent to the email address.

The running and maintenance of a database is an important job. Having the right plan for your database will ensure that it continues to work as originally designed, and you will be able to quickly identify and fix database errors or slowdowns early on.

Backup and Recovery

The most important task a DBA can perform is backing up the essential database in use. When you create a maintenance plan, it's important to have backup and recovery at the top of the maintenance list in case the job doesn't get fully completed. Firstly, it is important to understand the transaction log and why it is important.

The Transaction Log

Whenever a change is made to the database, be it a transaction or modification, it is stored in the transaction log. The transaction log is the most important file in an SQL Server database, and everything revolves around either saving it or using it.

Every transaction log can facilitate transaction recovery, recover all incomplete transactions, roll forward a restored file, filegroup, or page to a point of failure, replicate transactions, and facilitate disaster recovery.

Recovery

The first step in backing up a database is choosing a recovery option for the database. You can perform the three types of backups when the SQL Server is online, and even while users are making requests from the database at the same time.

When you perform the process of doing backup and restore in the SQL Server, you do so within the confines of the recovery model designed to control the maintenance of the transactional log. Such a recovery model is known to be a database property aimed at ensuring that all transactions are logged in a certain procedure.

There are three different recovery options: simple, full, and bulk-logged.

Simple Recovery

You cannot backup the transaction log when utilizing the simple recovery model. Usually, this model is used when updates are infrequent. Transactions are minimally logged and the log will be truncated.

Full Recovery

In the full recovery model, the transaction log backup must be taken. Only when the backup process begins will the transaction log be truncated. You can recover to any point in time. However, you also need the full chain of log files to restore the database to the nearest time possible.

Bulk-Logged Recovery

This model is designed to be utilized for short-term use when you use a bulk import operation. You use it along with the full recovery model whenever you don't need a recovery to a certain point in time. It has performance gains and doesn't fill up the transaction log.

Changing the Recovery Model

To change the recovery model, you can right-click on a database in SQL Server Management Studio and select Properties. Then, select Options and choose the recovery model from the drop-down box. Alternatively, you can use one of the following:

```
ALTER DATABASE SQLEbook SET RECOVERY SIMPLE
```

```
GO
```

```
ALTER DATABASE SQLEbook SET RECOVERY FULL
```

```
GO
```

```
ALTER DATABASE SQLEbook SET RECOVERY BULK_LOGGED
```

```
GO
```

Backups

There are three types of backup: full, differential, and transaction log. When Database Administrators set up a backup plan, they base their plan on two measures: Recovery Time Objective (RTO) and Recovery Point Objective (RPO). The RTO records the period to recover after a notification of a disruption in the business process. RPO measures the timeframe that might pass during a disruption before the data size that has been lost exceeds the maximum limit of the

business process.

If there was an RPO of only 60 minutes, you couldn't achieve this goal if your backup was set to every 24 hours. You need to set your backup plan based on these two measures.

Full Backup

When you create a full backup, the SQL Server creates a CHECKPOINT which ensures that any exiting dirty pages are written to disk. Then, the SQL Server backs up each and every page on the database. It then backs up the majority of the transaction log to ensure there is transactional consistency. What all of this means is that you are able to restore your database to the most recent point and recover all the transactions, including those right up to the very beginning of the backup.

Exercising this alone is the least flexible option. Essentially, you are only able to restore your database back to one point of time, which is the last full backup. Thus, if the database went corrupt two hours from midnight (and your backup is at midnight) your RPO would be 22 hours. In addition, if a user truncated a table two hours from midnight, you would have the same 22-hour loss of business transactions.

Transaction Log Backup

With the transaction log backup, the SQL Server backs up the data in the transaction log only, in other words, only the transactions that were recently committed to the database. The transaction log is not as resource-hungry and is considered important because it can perform backups more often without having an impact on database performance.

If you select Full Recovery mode, you can run both a full backup and a transaction log backup. You can also run more frequent backups since running the transaction log backup takes less resources. This is a very good choice if your database is updated throughout the day.

In scheduling transaction log backups, it's best to follow the RPO. For example, if there is an RPO of 60 minutes, then set the log file backups to 60 minutes. However, you must check the RTO for such a backup. If you had an RPO of 60 minutes and are only performing a full backup once a week, you might not be able to restore all 330 backups in the allotted time.

Differential Backup

To get around the problem mentioned above, you can add differential backups to the plan. A differential backup is cumulative, which mean a serious reduction in the number of backups you would need to recover your database to the point just before failure.

The differential backup, as its name suggests, backs up every page in the database that has since been modified since the last backup. The SQL Server keeps track of all the different pages that have been modified via flags and DIFF pages.

Performing a Backup

To back up a database, right-click the database in SSMS, then select Tasks → Backup. You can select what kind of backup to perform (full, differential, or transaction log) and when to perform the backup. The copy-only backup allows you to perform a backup that doesn't affect the restore sequence.

Restoring a Database

When you want to restore a database in SSMS, right-click the database, then select Tasks → Restore → Database. You can choose the database contained in the drop-down menu and keep the rest of the tabs populated.

If you click on Timeline, you can see a graphical diagram of when the last backup was created, which shows how much data was lost. You may have the option of recovering up to the end of a log, or a specific date and time.

The Verify Backup Timeline media button enables you to verify the backup media before you actually restore it. If you want to change where you are going to store the backup, you can click on Files to select a different location. You can specify the restore options that you want to use on the Options page. Either overwrite the existing database or keep it. The recovery state either brings the database online or allows further backups to be applied.

Once you click OK, the database will be restored.

Attaching and Detaching Databases

The method of attaching and detaching databases is similar to that of backups and restores.

Essentially, here are the details of this method:

- Allows you to copy the .MDF file and .LDF file to a new disk or server.
- Performs like a backup and restore process, but can be faster at times, depending on the situation.
- The database is taken offline and cannot be accessed by any users or applications. It will remain offline until it's been reattached.

So, which one should you choose? Though a backup is the ideal option, there are cases where an attachment/detachment of the database may be your only choice.

Consider the following scenario:

Your database contains many filegroups. Attaching those can be quite cumbersome.

The appropriate solution is to back up the database and then restore it to the desired destination, as it will group all of the files together in the backup process.

Based on the size of the database, the backup/restore process takes a long time. However, the attaching/detaching of the database could be much quicker if it's needed as soon as possible.

In this scenario, you can take the database offline, detach it, and re-attach it to the new destination.

As mentioned above, there are two main file groups when following the method of attaching databases. These files are .MDF and .LDF. The .MDF file is the database's primary data file, which holds its structure and data. The .LDF file holds the transactional logging activity and history.

However, a .BAK file that's created when backing up a database, groups all of the files together and you restore different file versions from a single backup set.

Consider your situation before taking either option, but also consider a backup and restore first before looking into the attach/detach method as your next option. Also, be sure to test it before you move forward with live data!

Attaching/Detaching the AdventureWorks2012 Database

Since you already attached this database, we'll have you detach it from the server. After that, you'll attach it again using SQL syntax.

Detaching the Database

In SQL Server, there's a stored procedure that will detach the database for you. This particular stored procedure resides in the "master" database. Under the hood, you can see the complexity of the stored procedure by doing the following:

1. Click to expand the Databases folder
2. Click on System Databases, then the "master" database
3. Click on Programmability
4. Click on Stored Procedures, then System Stored Procedures
5. Find `sys.sp_detach_db`, right-click it and select 'Modify' in SSMS to see its syntax

For this, you'll just execute the stored procedure as is.

Below is the syntax:

```
USE master
```

```
GO
```

```
ALTER DATABASE DatabaseName SET SINGLE_USER
```

```
WITH ROLLBACK IMMEDIATE
```

```
GO
```

```
EXEC master.dbo.sp_detach_db @dbname = N'DatabaseName',
```

```
@skipchecks = 'false'
```

```
GO
```

We'll expand a little on what is happening. You want to use the "master" database to alter the database you'll be detaching and set it to single user instead of multi-user.

Last, the value after `@dbname` allows you to specify the name of the database to be detached, and the `@skipchecks` set to false means that the database engine will update the statistics information, identifying that the database has been detached. It's ideal to set this as `@false` whenever detaching a database so that the system

holds current information about all databases.

Attaching Databases

Once you have detached your database, if you navigate to where your data directory is, you'll see that the AdventureWorks2012_Data.MDF file still exists – which it should since you only detached it and didn't delete it.

Next, take the file path of the .MDF file and copy and paste it in some place that you can easily access, like notepad. The location we use is C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA.

Now, go back into SSMS and click on the New Query button (if you're already connected). If you have not connected, then go ahead and connect to your instance.

Once you've connected to your instance and opened up a new query session, you'll just need to use the path of where the data file is stored. Once you have that, you can enter that value in the following SQL syntax examples in order to attach your database.

Below is the syntax for attaching database files and log files. Though in the following exercise, you'll be skipping attaching the log file completely, since you're not attaching this to a new server. Therefore, you may omit the statement to attach the log file.

```
CREATE DATABASE DatabaseName ON
```

```
(FILENAME = 'C:\SQL Data Files\DatabaseName.mdf'),
```

```
(FILENAME = 'C:\SQL Data Files\DatabaseName_log.ldf') FOR ATTACH
```

In the above example, we are calling out the statement to attach the log file if one is available. However, if you happen to not have the .LDF file and only the .MDF file, then that's fine, too. You can just attach the .MDF file and the database engine will create a new log file and start writing activity to that particular log.

CHAPTER 18: WORKING WITH SSMS

Downloading SQL Server Management Studio (SSMS)

SQL Server Management Studio is Microsoft's interface for interacting with SQL databases. It's free and is a great tool for learning and managing database servers.

At this time, Microsoft has versions of 17.0 and up ("Release notes for SQL Server Management Studio (SSMS) - SQL Server"). We've used version 16.5.3 and have had a really good experience with it. Here is the link to that, as well as versions 17.0 and beyond, if you want to try them.

Version 17.0+:

<https://docs.microsoft.com/en-us/sql/ssms/release-notes-ssms?view=sql-server-2017#download-ssms-170>

Version 16.5.3:

<https://docs.microsoft.com/en-us/sql/ssms/release-notes-ssms?view=sql-server-2017#download-ssms-1653>

All you need to do is click the link to download the version that you'd like and it will start downloading. With earlier versions, use the panel on the right side of the browser to cycle through the versions and choose the one you'd like.

Once it has downloaded, go ahead and run the installer. Both versions run through the standard installation process. You can keep the default settings and let it install.

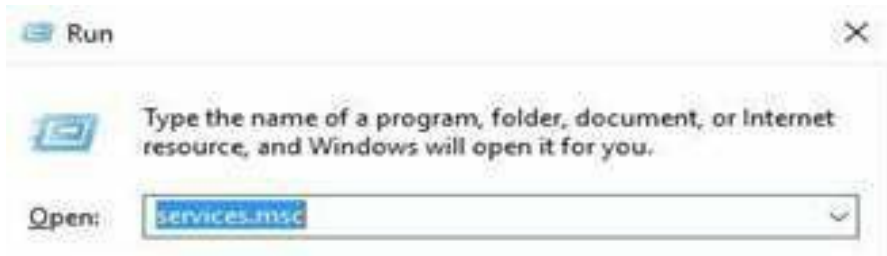
After it's finished, you're almost ready to connect!

Starting the Database Engine Services

After you have finished installing your instance, it's almost time to log in!

The first step is to check and ensure that the database service is in a good state and running. The service can be found under the list of typical services running under your Windows OS. There are a couple of ways for you to locate the service:

Option 1: Press the Windows key + R together to bring up the Run command. Once it comes up, enter **services.msc** to bring up the list of services.



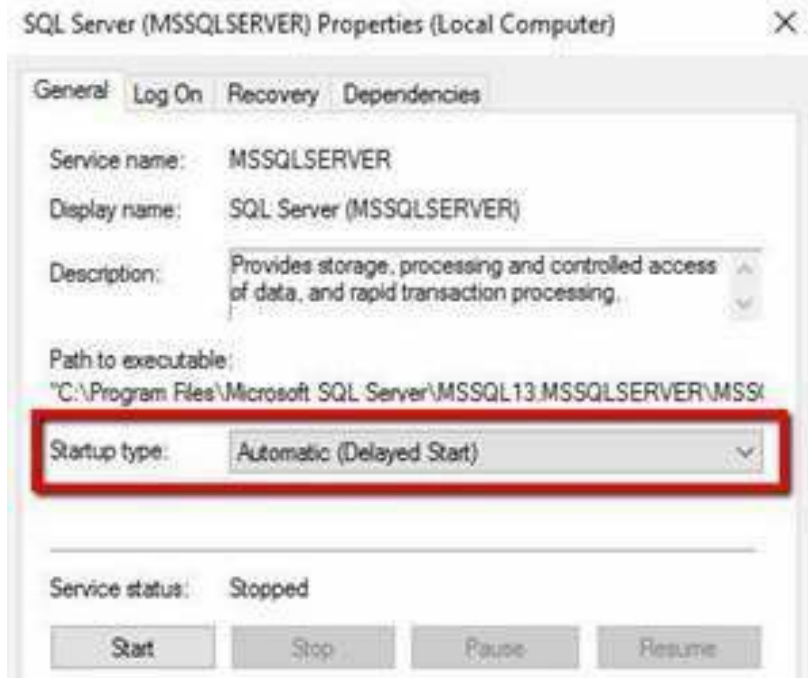
Once here, scroll down to find the service SQL Server. By default, it will not be running.



To connect to the database engine, the service will need to be running. You can click to “Start” the service, as shown above on the left, and then you’ll be ready to go.

By default, this service is set up to run manually so that it doesn’t consume a lot of resources on your computer when it’s on. You can keep it on manual if you’d like, though you should remember that you will need to start it manually every time your computer comes back on after it’s been turned off and there is a need to connect to the SQL Server.

Alternatively, you can right-click SQL Server service, go to Properties, and change the Startup Type to either Automatic or Automatic (Delayed Start). This will let it run automatically and you won’t need to stop/start when you want to connect to SQL. **Make sure that you start this service, too!**

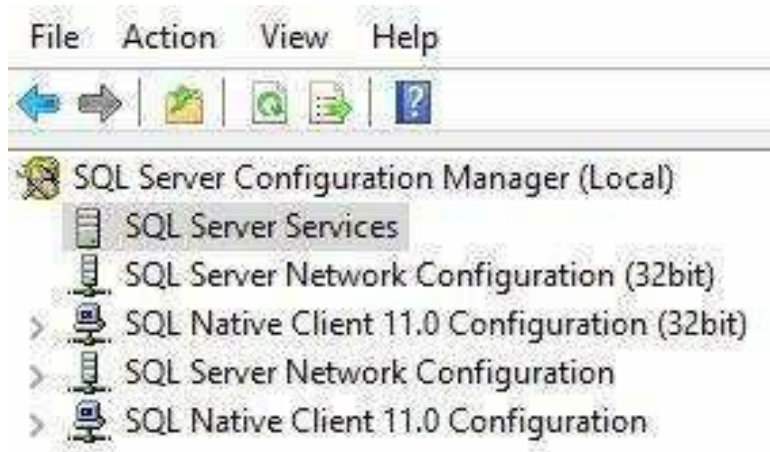


Option 2: Go to the Start menu and look for SQL Server 2016 Configuration Manager.



SQL Server 2016 Configuration Manager
Desktop app

Click the SQL Server Services on the left-hand side.



Then, look for the SQL Server service.

Name	State	Start Mode
SQL Full-text Filter Daemon Launcher (MSSQLSERVER)	Running	Manual
SQL Server Launchpad (MSSQLSERVER)	Running	Automatic
SQL Server (MSSQLSERVER)	Running	Automatic
SQL Server Reporting Services (MSSQLSERVER)	Running	Automatic
SQL Server Browser	Stopped	Other (Boot, System, Disabl...
SQL Server Agent (MSSQLSERVER)	Stopped	Automatic

As explained above, it won't be running by default so that it doesn't consume a lot of resources on your machine. Go ahead and start it up and then you're ready to connect!

Connect to SQL Server with SSMS

Here's where you can really start getting in and creating your database and data. Search for SQL Server Management Studio on your local computer and open it.

When the main screen finally opens, you'll want to ensure that you have the following configured:

- Server Type: Database Engine
- Server Name: (The name of your computer)
- Authentication: Windows Authentication

Here's what mine looks like:



In order to obtain your server name, open the Control Panel and navigate to this path: **Control Panel\All Control Panel Items\System**. Then, look for what's displaying in the Computer Name field. The Computer Name alone is all that you'll need – you won't need the Full Computer Name.

Since you're using Windows Authentication, you won't need to enter a password. Just click connect and you will be connected to your database engine.

The Basics and Features of SSMS

Managing Connections

There are multiple ways to manage connections to servers within SSMS. The connection near the top arrow gives you access to manage the database servers within the current query. The connection near the bottom arrow gives you access to control connections to many database servers in SSMS.



Choosing Your Database

Within SSMS, you have the ability to choose the databases you'd like to use, depending on which database server you're connected to. All you have to do is choose from the drop-down list and click the desired database.



New Query Window

In order to run queries against your database, you must open a query session. You can achieve this by using the handy New Query button shown below, which will open a query to your current database connection. You can also see that the

shortcut is Ctrl + N.



The second way to do this is by using the Object Explorer window and finding the database that you want to use. Simply right-click the database and choose New Query.



Executing Statements

There are several ways to execute or run SQL statements in SSMS. First, we'll expand on how you can run these statements.

You can highlight certain batches of code in order to run them. Once you've highlighted the desired code, you can execute it in any of the following ways.

However, if you do not highlight any code at all, you'll be executing all code within your current query window.

Here's the first option to execute SQL statements:



The other option is to press F5 or use Ctrl + E on your keyboard.

IntelliSense

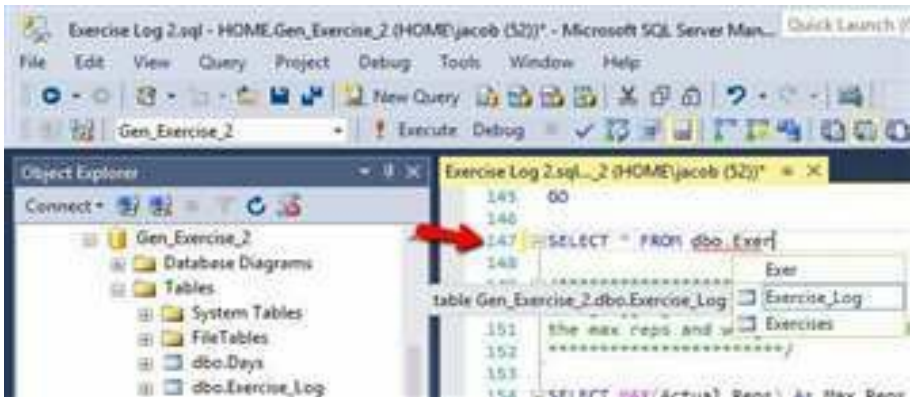
This feature is one of the greatest features of SSMS. Instead of having to remember table names, column names, etc., SSMS has integrated a feature that populates objects within the database based on what you type.

To turn it on, simply click the button as shown below. You can also hover over the button with your mouse in SSMS to see the keyboard shortcuts to enable/disable it.



Below is an example of how IntelliSense works. You'll begin typing an object name in your database and it will populate a list of relative results for you.

You can see that we're using the Gen_Exercise_2 database and writing a SELECT statement on line 147 to pull data from the Exercise_Log table. You can also see that the table exists in the list of tables on the left-hand side, and IntelliSense is providing us with some suggestions.



Sometimes, IntelliSense doesn't work as well as it should, or it stops working completely. If this is the case, then you can refresh the cache by going to Edit → IntelliSense → Refresh Local Cache.

Note that the keyboard shortcut Ctrl + Shift + R also refreshes the IntelliSense cache.

Results Presentation

You can change the way that the result set is presented to you. The most common ones are Results to Grid and Results to Text. You can control each option in the below image. The one on the right is the grid option, and the one on the left is the text option.



The keyboard shortcut for text results is Ctrl + T, while the grid option is Ctrl + D.

Below is an example of what the grid option looks like:



	Day	Exercise_Name	Rep_Goal	Actual_Reps	Weight_Used	Date_Performed
1	Monday	Bicep Curls	5	7	25.00	2017-12-05
2	Tuesday	Chest Press	5	6	35.00	2017-12-05
3	Wednesday	Squats	8	9	45.00	2017-12-05
4	Thursday	Calf Raises	6	8	30.00	2017-12-05
5	Friday	Shoulder Press	11	12	40.00	2017-12-05
6	Saturday	Back Rows	8	9	50.00	2017-12-05

Below is an example of what the text option looks like:

Results				
Day	Exercise_Name	Rep_Goal	Actual_Reps	Weight_Used
Monday	Bicep Curls	5	7	25.00
Tuesday	Chest Press	5	6	35.00
Wednesday	Squats	8	9	45.00
Thursday	Calf Raises	6	8	30.00
Friday	Shoulder Press	11	12	40.00
Saturday	Back Rows	8	9	50.00
Sunday	Glute Bridge	9	12	60.00

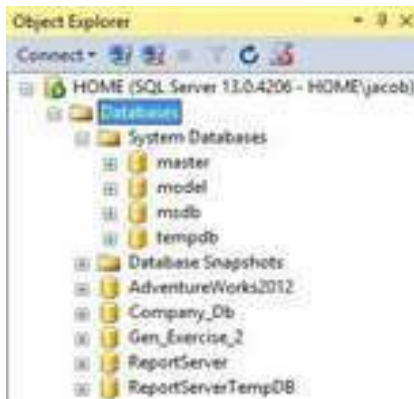
The last keyboard shortcut for the results is Ctrl + R, which toggles the ability to show/hide the result set. That way, if you want to keep your query window clear while you write code, just use the keyboard shortcut.

Object Explorer

This is one of the tools that you will use quite often. The object explorer interface allows you to dig deeper into the information of the databases, tables, views, users, stored procedures, and so much more.

Databases

You can click to expand the SQL Server system databases, along with all of the databases created by a user.



Here's a brief summary of each SQL Server system database:

- **Master** – holds all of the system/server configurations and their information. Make backups of this frequently, as you want to retain as much of this data as possible.
- **Model** – A copy of this is used when you create your own custom database.
- **Msdb** – This manages and holds the configuration for SQL Server Agent Scheduled Jobs and other automated tasks.
- **Tempdb** – Holds temporary data and is used for running queries and stored procedures. The information in this database will only be retained while the session is open. Once you open a new session, all of the data is gone.

You may also notice that you have ReportServer and ReportServerTempDB. The reason that these two exist here is because you installed the Reporting Services – Native feature during the installation of SQL Server Express.

Here's what they do:

- **ReportServer** – The main database that holds the data for custom reports, as well as any scheduled jobs and notifications.
- **ReportServerTempDB** – A temporary database that holds a user's session information and a local cache for the reporting instance.

CHAPTER 19: REAL-WORLD USES

We have seen how we can use SQL in isolation. For instance, we went through different ways to create tables and what operations you can perform on those tables to retrieve the required answers. If you only wish to learn how SQL works, you can use this type of learning, but this is not how SQL is used.

The syntax of SQL is close to English, but it is not an easy language to master. Most computer users are not familiar with SQL, and you can assume that there will always be individuals who do not understand how to work with SQL. If a question about a database comes up, a user will almost never use a `SELECT` statement to answer that question. Application developers and systems analysts are probably the only people who are comfortable with using SQL. They do not make a career out of typing queries into a database to retrieve information. They instead develop applications that write queries.

If you intend to reuse the same operation, you should ensure that you never have to rebuild that operation from scratch. Instead, write an application to do the job for you. If you use SQL in the application, it will work differently.

SQL in an Application

You may believe that SQL is an incomplete programming language. If you want to use SQL in an application, you must combine SQL with another procedural language like FORTRAN, Pascal, C, Visual Basic, C++, COBOL, or Java. SQL has some strengths and weaknesses because of how the language is structured. A procedural language that is structured differently will have different strengths and weaknesses. When you combine the two languages, you can overcome the weaknesses of both SQL and the procedural language.

You can build a powerful application when you combine SQL and a procedural language. This application will have a wide range of capabilities. We use an asterisk to indicate that we want to include all the columns in the table. If this table has many columns, you can save a lot of time by typing an asterisk. Do not use an asterisk when you are writing a program in a procedural language. Once you have written the application, you may want to add or delete a column from the table when it is no longer necessary. When you do this, you change the meaning of the asterisk. If you use the asterisk in the application, it may retrieve columns which it thinks it is getting.

This change will not affect the existing program until you need to recompile it to make some change or fix a bug. The effect of the asterisk wildcard will then expand to current columns. The application could stop working if it cannot identify the bug during the debugging process. Therefore, when you build an application, refer to the column names explicitly in the application and avoid using the asterisk.

Since the replacement of paper files stored in a physical file cabinet, relational databases have given way to new ground. Relational database management systems, or RDBMS for short, are used anywhere information is stored or retrieved, like a login account for a website or articles on a blog. Speaking of which, this also gave a new platform to and helped leverage websites like Wikipedia, Facebook, Amazon, and eBay. Wikipedia, for instance, contains articles, links, and images, all of which are stored in a database behind-the-scene. Facebook holds much of the same type of information, and Amazon holds product information and payment methods, and even handles payment transactions.

With that in mind, banks also use databases for payment transactions and to manage the funds within someone's bank account. Other industries, like retail, use

databases to store product information, inventory, sales transactions, price, and so much more. Medical offices use databases to store patient information, prescription medication, appointments, and other information.

To expand further, using the medical office for instance, a database gives permission for numerous users to connect to it at once and interact with its information. Since it uses a network to manage connections, virtually anyone with access to the database can access it from just about anywhere in the world.

The digital database helps leverage mobile applications and provides new opportunities for software, or any other platforms that use databases on a daily basis, to be developed. One app that comes to mind would be an email app, as it's storing the emails on a server somewhere in a data center and allowing you to view and send emails.

These types of databases have also given way to new jobs and have even expanded the tasks and responsibilities of current jobs. Those who are in finance, for instance, now have the ability to run reports on financial data; those in sales can run reports for sales forecasts, and so much more!

In practical situations, databases are often used by multiple users at the same time. A database that can support many users at once has a high level of concurrency. In some situations, concurrency can lead to loss of data or the reading of non-existent data. SQL manages these situations by using transactions to control atomicity, consistency, isolation, and durability. These elements comprise the properties of transactions. A transaction is a sequence of T-SQL statements that combine logically and complete an operation that would otherwise introduce inconsistency to a database. Atomicity is a property that acts as a container for transaction statements. If the statement is successful, then the total transaction completes. If any part of a transaction is unable to process fully, then the entire operation fails, and all partial changes roll back to a prior state. Transactions take place once a row, or a page-wide lock is in place. Locking prevents modification of data from other users taking effect on the locked object. It is akin to reserving a spot within the database to make changes. If another user attempts to change data under lock, their process will fail, and an alert communicates that the object in question is barred and unavailable for modification. Transforming data using transactions allows a database to move from one consistent state to a new consistent state. It's critical to understand that transactions can modify more than one database at a time. Changing data in a primary key or foreign key field

without simultaneously updating the other location, creates inconsistent data that SQL does not accept. Transactions are a big part of changing related data from multiple table sources all at once. Transactional transformation reinforces isolation, a property that prevents concurrent transactions from interfering with each other. If two simultaneous transactions take place at the same time, only one of them will be successful. Transactions are invisible until they are complete. Whichever transaction completes first will be accepted. The new information displays upon completion of the failed transaction, and at that point, the user must decide if the updated information still requires modification. If there happened to be a power outage and the stability of the system fails, data durability would ensure that the effects of incomplete transactions rollback. If one transaction completes and another concurrent transaction fails to finish, the completed transaction is retained. Rollbacks are accomplished by the database engine using the transaction log to identify the previous state of data and match the data to an earlier point in time.

There are a few variations of a database lock, and various properties of locks as well. Lock properties include mode, granularity, and duration. The easiest to define is duration, which specifies a time interval where the lock is applied. Lock modes define different types of locking, and these modes are determined based on the type of resource being locked. A shared lock allows the data reads while the row or page lock is in effect. Exclusive locks are for performing data manipulation (DML), and they provide exclusive use of a row or page for the execution of data modification. Exclusive locks do not take place concurrently, as data is being actively modified; the page is then inaccessible to all other users regardless of permissions. Update locks are placed on a single object and allow for the data reads while the update lock is in place. They also allow the database engine to determine if an exclusive lock is necessary once a transaction that modifies an object is committed. This is only true if no other locks are active on the object in question at the time of the update lock. The update lock is the best of both worlds, allowing reading of data and DML transactions to take place at the same time until the actual update is committed to the row or table. These lock types describe page-level locking, but there are other types beyond the scope of this text. The final property of a lock, the granularity, specifies to what degree a resource is unavailable. Rows are the smallest object available for locking, leaving the rest of the database available for manipulations. Pages, indexes, tables, extents, or the entire database are candidates for locking. An extent is a

physical allocation of data, and the database engine will employ this lock if a table or index grows and more disk space is needed. Problems can arise from locks, such as lock escalation or deadlock, and we highly encourage readers to pursue a deeper understanding of how these function.

It is useful to mention that Oracle developed an extension for SQL that allows for procedural instruction using SQL syntax. This is called PL/SQL, and as we discussed at the beginning of the book, SQL on its own is unable to provide procedural instruction because it is a non-procedural language. The extension changes this and expands the capabilities of SQL. PL/SQL code is used to create and modify advanced SQL concepts such as functions, stored procedures, and triggers. Triggers allow SQL to perform specific operations when conditional instructions are defined. They are an advanced functionality of SQL, and often work in conjunction with logging or alerts to notify principals or administrators when errors occur. SQL lacks control structures, the for looping, branching, and decision making, which are available in programming languages such as Java. The Oracle corporation developed PL/SQL to meet the needs of their database product, which includes similar functionality to other database management systems, but is not limited to non-procedural operations. Previously, user-defined functions were mentioned but not defined. T-SQL does not adequately cover the creation of user-defined functions, but using programming, it is possible to create functions that fit neatly within the same scope as system-defined functions. A user-defined function (UDF) is a programming construct that accepts parameters, performs tasks capable of making use of system defined parameters, and returns results successfully. UDFs are tricky because Microsoft SQL allows for stored procedures that often can accomplish the same task as a user-defined function. Stored procedures are a batch of SQL statements that are executed in multiple ways and contain centralized data access logic. Both of these features are important when working with SQL in production environments.

CONCLUSION

Thank you for making it through to the end of this book. You have learned many techniques that apply to SQL. The next step is implementing all the new techniques and ensuring you improve on your SQL mastery.

Good luck, and enjoy the journey!

REFERENCES

- CS145 Lecture Notes (8) -- Constraints and Triggers. (n.d.). Retrieved from <http://infolab.stanford.edu/~ullman/fcdb/jw-notes06/constraints.html>
- Guru99. (n.d.). What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples. Retrieved from <https://www.guru99.com/database-normalization.html>
- Hosch, W. L. (2019, April 14). Edgar Frank Codd. Retrieved from <http://www.britannica.com/biography/Edgar-Frank-Codd>
- Jones, A. D., Plew, R., & Stephens, R. (2008, July 1). Informit. Retrieved from <http://www.informit.com/articles/article.aspx?p=1216889&seqNum=4>
- Microsoft. (n.d.). Release notes for SQL Server Management Studio (SSMS) - SQL Server. Retrieved from <https://docs.microsoft.com/en-us/sql/ssms/release-notes-ssms?view=sql-server-2017#previous-ssms-releases>
- Microsoft. (n.d.). Server-Level Roles - SQL Server. Retrieved from <https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-2017>
- MySQL. (n.d.). Chapter 5 Installing MySQL on Microsoft Windows. Retrieved from <https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/windows-installation.html>
- Oracle. (2016, December 23). Fusion Middleware WebCenter Sites: Installing and Configuring Supporting Software. Retrieved from https://docs.oracle.com/cd/E29542_01/doc.1111/e29751/db_oracle_11.htm#WB0
- Otey, M. (2015, July 31). Setting up SQL Server 2014 and Oracle 12c Linked Servers. Retrieved from <https://logicalread.com/sql-server-2014-and-oracle-12c-linked-servers-mo01/#.XOvpVohKjIU>
- Snowflake Inc. (n.d.). Arithmetic Operators. Retrieved from <https://docs.snowflake.net/manuals/sql-reference/operators-arithmetic.html>
- SQL Data Types. (2001, September 26). Retrieved from http://www.cs.toronto.edu/~nn/csc309-20085/guide/pointbase/docs/html/htmlfiles/dev_datatypesandconversionsFIN.html
- Steynberg, M. (2016, August 16). What is a SQL Server deadlock? - SQL Shack. Retrieved from <https://www.sqlshack.com/what-is-a-sql-server-deadlock/>

TekSlate. (2018, September 21). SQL Clone Tables. Retrieved from <https://tekslate.com/sql-clone-tables>

The Carpentries. (n.d.). Filtering. Retrieved from <https://swcarpentry.github.io/sql-novice-survey/03-filter/>

Vishwakarma, Y. (2015, December 29). Basics of Database Administration in SQL Server : Part 1. Retrieved from <https://www.c-sharpcorner.com/UploadFile/168aad/basics-of-database-administration-in-sql-server-part-1/>

W3schools.com. (n.d.). SQL Injection. Retrieved from https://www.w3schools.com/sql/sql_injection.asp

W3schools.com. (n.d.). Introduction to SQL. Retrieved from https://www.w3schools.com/sql/sql_intro.asp

W3schools.com. (n.d.). SQL Stored Procedures for SQL Server. Retrieved from https://www.w3schools.com/sql/sql_stored_procedures.asp

Xue, M. (2018, November 1). SQL Pivot: Converting Rows to Columns - The Databricks Blog. Retrieved from <https://databricks.com/blog/2018/11/01/sql-pivot-converting-rows-to-columns.html>

[1] <https://sites.google.com/site/prgimr/sql/ddl-commands---create---drop---alter>

[2] https://www.w3schools.com/sql/sql_primarykey.asp

[3] <https://www.1keydata.com/sql/sql-create-view.html>

[4] <https://www.1keydata.com/sql/sql-create-view.html>

[5] https://www.w3schools.com/sql/sql_join_inner.asp

[6] https://www.w3schools.com/sql/sql_join_right.asp

[7] https://www.w3schools.com/sql/sql_join_left.asp

[8] https://www.w3schools.com/sql/sql_union.asp

[9] https://www.w3schools.com/sql/sql_union.asp

[10] https://www.techonthenet.com/sql/union_all.php

[11] https://www.w3schools.com/sql/sql_notnull.asp

[12] https://www.w3schools.com/sql/sql_unique.asp

[13] https://www.w3schools.com/sql/sql_unique.asp

- [14] <https://chartio.com/resources/tutorials/how-to-alter-a-column-from-null-to-not-null-in-sql-server/>
- [15] <https://www.tutorialspoint.com/sql/sql-primary-key.htm>
- [16] <https://www.tutorialspoint.com/sql/sql-primary-key.htm>
- [17] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-2017>
- [18] <https://docs.faircom.com/doc/esql/32218.htm>
- [19] <https://stackoverflow.com/questions/7573590/can-a-foreign-key-be-null-and-or-duplicate>
- [20] <https://stackoverflow.com/questions/7573590/can-a-foreign-key-be-null-and-or-duplicate>
- [21] https://www.w3schools.com/sql/sql_check.asp
- [22] <https://stackoverflow.com/questions/11981868/limit-sql-server-column-to-a-list-of-possible-values>
- [23] <https://www.tutorialspoint.com/sql/sql-primary-key.htm>
- [24] <https://www.tutorialspoint.com/sql/sql-primary-key.htm>
- [25] <https://stackoverflow.com/questions/11981868/limit-sql-server-column-to-a-list-of-possible-values>
- [26] <https://stackoverflow.com/questions/11981868/limit-sql-server-column-to-a-list-of-possible-values>
- [27] <https://docs.microsoft.com/en-us/biztalk/core/step-2-create-the-inventory-request-schema>
- [28] https://www.techonthenet.com/oracle/schemas/create_schema.php
- [29] <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-schema-transact-sql?view=sql-server-2017>
- [30] <https://www.postgresql.org/docs/9.3/sql-createschema.html>
- [31] https://www.techonthenet.com/sql/tables/create_table2.php
- [32] https://www.w3schools.com/sql/sql_wildcards.asp
- [33] https://www.w3schools.com/sql/sql_like.asp
- [34] https://www.w3schools.com/sql/sql_like.asp
- [35] https://www.w3schools.com/sql/sql_where.asp
- [36] https://www.w3schools.com/sql/sql_orderby.asp
- [37] https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6014.htm
- [38] https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6014.htm
- [39] <https://stackoverflow.com/questions/2578194/what-is-ddl-and-dml>
- [40] https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6014.htm
- [41] <https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-detach-and-attach-sql-server?view=sql-server-2017>
- [42] <https://www.tutorialspoint.com/sql/sql-like-clause.htm>

[43] <https://www.tutorialspoint.com/sql/sql-like-clause.htm>

[44] <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

[45] <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

[46] <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

[47]

[48] <https://docs.microsoft.com/en-us/sql/t-sql/functions/round-transact-sql?view=sql-server-2017>

[49] [h](#)

ID	EMP_NAME	SALES	BRANCH
1001	ALAN MARSCH	3000.00	NEW YORK
1098	NEIL BANKS	5400.00	LOS ANGELES
2005	RAIN ALONZO	4000.00	NEW YORK
3008	MARK FIELDING	3555.00	CHICAGO
4356	JENNER BANKS	14600.00	NEW YORK
4810	MAINE ROD	7000.00	NEW YORK
5783	JACK RINGER	6000.00	CHICAGO
6431	MARK TWAIN	10000.00	LOS ANGELES
7543	JACKIE FELTS	3500.00	CHICAGO
8934	MARK GOTH	5400.00	AUSTIN

10 rows in set (0.00 sec)

<https://docs.microsoft.com/en-us/sql/t-sql/functions/round-transact-sql?view=sql-server-2017>

[50] <https://docs.microsoft.com/en-us/sql/t-sql/functions/round-transact-sql?view=sql-server-2017>

[51] <https://docs.microsoft.com/en-us/sql/t-sql/functions/round-transact-sql?view=sql-server-2017>

[52] <https://docs.microsoft.com/en-us/sql/t-sql/functions/round-transact-sql?view=sql-server-2017>

[53]

[54] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[55] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[56] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[57] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[58] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[59] <https://stackoverflow.com/questions/1475589/sql-server-how-to-use-an-aggregate-function-like-max-in-a-where-clause>

[60] <https://stackoverflow.com/questions/886786/how-to-change-a-table-name-using-an-sql-query>

[61] <https://www.1keydata.com/sql/alter-table-rename-column.html>

[62] <https://www.1keydata.com/sql/alter-table-rename-column.html>

[63] <https://www.1keydata.com/sql/alter-table-rename-column.html>

[64] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/modify-columns-database-engine?view=sql-server-2017>

[65] <https://www.tutorialspoint.com/sql/sql-inner-joins.htm>

[66] <http://www.sql-join.com/sql-join-types/>

[67] <http://www.sql-join.com/sql-join-types/>

[68] <http://www.sql-join.com/sql-join-types/>

[69] <http://www.sql-join.com/sql-join-types/>

[70] <http://www.sql-join.com/sql-join-types/>

[71] <http://www.sql-join.com/sql-join-types/>

[72] <http://www.sql-join.com/sql-join-types/>

[73] <https://www.tutorialspoint.com/sql/sql-top-clause.htm>

[74] <https://www.tutorialspoint.com/sql/sql-top-clause.htm>

[75]

ID	NAME	ADDRESS	AGE	SALARY
2	Mercy	Mercy32	25	3500.00
3	Joel	Joel42	30	4000.00
4	Alice	Alice442	31	2500.00
5	Nicholas	nicoh442	45	5000.00
6	Milly	mil342	32	2000.00
7	Grace	gra361	35	4000.00

<https://www.tutorialspoint.com/sql/sql-top-clause.htm>

[76] <https://www.tutorialspoint.com/sql/sql-top-clause.htm>

[77] https://www.techonthenet.com/sql/group_by.php

[78] https://www.techonthenet.com/sql/group_by.php

[79] https://www.techonthenet.com/sql/group_by.php

[80] https://www.techonthenet.com/sql/group_by.php

[81] <https://www.tutorialspoint.com/sql/sql-not-null.htm>

[82] <https://www.tutorialspoint.com/sql/sql-default.htm>

[83] <https://www.tutorialspoint.com/sql/sql-default.htm>

[84] <https://www.tutorialspoint.com/sql/sql-default.htm>

[85] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-primary-keys?view=sql-server-2017>

[86] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-primary-keys?view=sql-server-2017>

[87] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-primary-keys?view=sql-server-2017>

[88] <https://docs.microsoft.com/en-us/sql/relational-databases/tables/create-check-constraints?view=sql-server-2017>

[89] <https://www.tutorialspoint.com/sql/sql-index.htm>

[90] <https://www.tutorialspoint.com/sql/sql-index.htm>

[91] https://www.techonthenet.com/sql/tables/alter_table.php

[92] https://www.techonthenet.com/sql/tables/alter_table.php

[93] https://www.techonthenet.com/sql/tables/alter_table.php

[94] https://www.techonthenet.com/sql/tables/alter_table.php

[95] https://www.techonthenet.com/sql/tables/alter_table.php

[96] https://www.techonthenet.com/sql/tables/alter_table.php