SQL

THE ULTIMATE INTERMEDIATE GUIDE TO LEARN
SQL PROGRAMMING STEP-BY-STEP

MARK REED

SQL

The Ultimate Intermediate Guide to Learning SQL Programming Step by Step

TABLE OF CONTENTS

INTRODUCTION

A database stores information in a structured way that is pertinent to the type of data it is storing. This data needs to be easily accessible to pull various reports and update, delete, modify, or add to the data stored.

One of the first databases was established in 1880. It was done with punch cards to organize a census in the USA. Charles Bachman was one of the first to take this concept and automate it in a system called the Integrated Data Store (IDS) in the 1960s (Hosch. n.d.).

By the 1970s, databases were well known, as were their usefulness and limitations. The database system was an excellent way of storing the data. However, accessing the data and manipulating it was cumbersome, time-consuming, and costly. Computer systems were also not as sophisticated as they are today.

Businesses started to rely more and more on technology. How companies processed and manipulated their data became more sophisticated. With historical databases, this would mean costly reworks of database systems that could take months.

Edgar Codd, who was working for IBM at the time, developed the relational data model in 1970. This model was introduced in a seminal paper he wrote. The paper was titled "*A Relational Model of Data for Large Shared Data Banks*," and it described a new way of structuring data (Hosch, n.d.). Codd's idea was derived from a branch of mathematics called **Set Theory**.

Oracle, Sybase Inc., and other Silicon Valley companies were already putting Edgar Codd's relational data model into practice long before IBM released SQL/DS. SEQUEL (Structured English Query Language), which was the original name of SQL, was developed in the early 1970s by Raymond Boyce and Donald Chamberlin.

It was not until 1981 that IBM released its first relational database called SQL/DS (Sequel Database system). In 1983 IBM released its second database management software family called SQL/DB2. The SQL/DB2 software was one of IBMs most successful software products ever. To date, DB2 products are still used for mission-critical systems through various industries (*Relational Database*, n.d.)

SQL

A standard relational database is a distributed system that constantly runs in the background, usually on a server. This is the program that interprets all the data files collected by a system. As it runs in the background it is known as the "back-end" of the system.

To update, delete, run reports, or manipulate that data, you need some sort of interpreter that can send those requests to the database from a client computer. Client software is installed on the client computer to send these requests or statements to the database. These requests are written in Structured Query Language (SQL) for the server to process and return the desired results.

SQL was initially called Structured English Query Language as it uses basic English sentences for writing queries. This makes it not only a powerful relational database tool but also not that hard to learn or use because of the common words it uses, such as select, update, insert, etc. Each statement takes very few lines of script to declare what needs to be done.

CHAPTER 1:

BASICS OF SQL

In today's environment, there are vast quantities of data that are collected nearly every second of the day. To be able to make this data useful it needs to be organized, edited, and structured in certain ways depending on the needs of the organization using it.

To understand how data can be manipulated using SQL, you first need to understand how the process all fits together. This chapter briefly outlines all the parts that fit together to create a workable database system.

Data and Databases

Data

Data is a collection of values, facts, and figures that has been gathered for some purpose. For instance, every ten years or so local or national governments collect data on residents within their cities or burroughs—census data. This data is gathered to help with various demographic conditions, city planners, and to aid government departments in understand various economic or social structures.

Data is only considered viable if it is meaningful. Meaningful data can come in various forms and come from different sources such as:

- Agricultural data
- Blogs
- Cultural data
- Educational data
- Environmental data
- Financial data
- Geographical data
- Meteorological data
- Scientific data
- Social media
- Statistical data
- Transport data

Databases

A database is a library, or storage area, where the collected data is stored. A database is bidirectional in that it can receive input and return the requested information. When data is imputed from a user, it is processed, converted, and then used as raw data/schema.

People use databases nearly every day without even realizing they're using them. For example, when you store a new contact on your phone, you are using a database. Here you would store information such as the person's name, surname, work, phone number(s), maybe even their physical addresses. When you look up that person's number, you are scrolling through a phonebook database to find and retrieve a phone number.

There are different types of databases:

- Cloud database
- Centralized database
- Distributed database
- Graph database
- NoSQL database
- Object-oriented database
- Operational database
- Relational database

Relational Database

Simply put, a relational database is a database that stores information and organizes it as a collection of tables with related data points. These related data points are connected by a unique ID called the Primary Key or Key.

Relational databases store information in tabular form which makes data more flexible to store, access, and manipulate. To better understand this, take a look at the tables below for a simple name, phone number, and address book:

Simple Name, Address, & Phone Number Database

Database Tables   **Database Relationships**

The phone numbers and address lists are joined to the names table by the User ID key. The above is a very basic example. Another example is that of a customer list and ordering system. Each customer will get a unique ID Key. This is the key that will be used to identify which customer is placing the order. The ID key is what forms the relationship bond between the two tables.

There are many relational database management programs on the market today. Some of the more popular relational databases on the market are:

- Azure SQL
- IMB DB2
- Informix
- MariaDB
- Microsoft Access
- MongoDB
- MySQL
- Oracle
- PostgreSQL
- SQLite
- SQL Server

Web/Cloud-Based Database Systems

A web or cloud database is a database that is accessed through the Internet. More and more companies as well as private individuals are moving towards web-based systems and applications.

Some of the top cloud-based relational databases include:

- Amazon Relational Database
- Couchbase
- DataStax

- Google Cloud SQL
- IBM Db2 on Cloud
- Microsoft Azure SQL Database
- MongoDB Atlas
- Oracle Database Cloud

Client/Server Technology

Historically, data was stored on large systems known as mainframe computers. To access the information on these systems the user would use a dumb terminal. Most of these terminals had a small screen and keyboard. They were heavy and the screen was black with either green or blueprint. They relied on the mainframe being accessible and drew all the information from it. If the mainframe was down, the terminals would not work.

Although there are still mainframes around, file servers have become the normal system storage solutions. A file server is either one powerful computer or a data center of powerful computers. These are the systems that not only store files, they also run various programs and allow for shared resources.

The end-user that needs to access the network does so through a network client. This is a desktop PC, a laptop, notebook, tablet, etc. For each program/database there will be the backend, which runs on the server, and the front-end, which runs on the client. The client accesses the data on the server by making a service request to the server. The server will process the request and respond. This is a client-server relationship.

The client is reliant on the server to manage, process, and provide information. However, most programs these days are resilient and allow the client to run independently from the server. If the server is down for any reason, depending on the system, the client may still be able to work. When the server is restored the information will update from the client to the server.

There are some applications, like web browsers, or web/cloud-based databases that are reliant on the back-end servers being up.

The Elements of an SQL Database

An SQL database consists of the following elements:

Tables

Tables or datasheets are much like a spreadsheet that holds specific related information. A database is made up of many different, but related tables. For example, in a simple ordering system you could have the following tables:

- Customer table — This table would hold all the customer's information.
- Stock Items table — This table would hold all the items held in stock.
- Orders table — This would hold all the customer's orders of the stock being sold.
- Invoices table — This would hold the invoices that were generated for the customers.
- Payments table — This would hold the payments that were made by the customers.

A table has to have a unique name within a database.

Each table will have a specific or set number of columns. Each column will have a unique identifying name that relates to the data and data type being stored in it.

Cells are an **attribute**. Each attribute will be defined by the name of the column and the data type to be stored in the column.

Each row will have a unique row ID or key that is used to identify the record or data in that row.

A row in a database is called a **tuple**. Each table will have an undefined number of rows or records that can and will grow as more records are added to the database.

Where the column and row meet is called a cell.

Fields

Each table will have several columns depending on the type of data the table was being used to collect. For instance, in the simple ordering system:

- Customer table you may have columns/fields for:
  - First Name
  - Last Name

- Telephone
- Address
- Stock Items you have columns/fields for:
  - Stock code
  - Item
  - Description
  - Price

Fields, also known as attributes, get set up to accommodate the type of data that is to be stored in the field. For instance:

- Stock codes would be an alphanumeric field
- Names would be a text field
- Quantity would be a numeric field
- Price would be a currency

There are many different types of fields, and each would be set up to not only hold but manage and manipulate the data. Names would need to be set as text to be able to effectively sort through them. Numbers have a few purposes such as codes, quantities, percentages, and currency. Each of these is used and calculated in a certain way and needs to be established when the database is set up.

Records

A table or datasheet also has a set of rows. These rows are the records or tuples and are what group the fields together. Each row will make up a collection of related information about the row's unique ID or key. For example, in a customer table, a single row would contain the customer's customer code, first name, last name, address, contact number, and so on.

Database Schemas

Every database contains what is called an object ownership group. These groups are known as a schema. This is the blueprint of the logical view as to how the database is structured and the relationships between the data.

Schemas are also used to form data constraints which help to authorize or prevent access for certain users or groups to various database tables, fields, and records. It is like a file that contains pertinent information with regard to certain parts of a database.

SQL Server

The actual SQL Server is a database engine that processes transactions such as creating databases, creating and manipulating tables, and handling the data contained in them.

It is also a Relationship Database Management System that manages the databases that are housed within its system.

SQL Server not only manages the data on the system, it also ensure that the database is:

- Secure
- Performs well
- Ensures data integrity
- Ensures data reliability
- Allows multiple concurrent access to the database

To use an SQL database, you need to have an instance of the SQL server running on the system that houses the SQL database. There can be more than one instance of an SQL server running on a system at one time.

Each SQL Server instance can contain one or more databases. Each database running on an SQL server will have a set of stored procedures, views, tables, and records. To access the SQL database, a user would need to log in with a user ID.

Each user ID will be granted access according to the specific task they need to perform within the database. Some databases do not require user logins. There are many programs or management tools that can be used to access SQL databases. These tools include:

- Azure Data Studio

- DataGrip
- DCIM
- RazorSQL
- SQL Server Management Studio (SSMS).
- Squirrel SQL

You can manage the SQL Server engine using management tools such as:

- SQL Server Management Studio (SSMS)
- Business Intelligence Developer
- Configuration Management
- SQL Profiler

CHAPTER 2:

INSTALLING AND CONFIGURING MYSQL

For this book, you will be using MySQL. MySQL is a relational database system that uses SQL syntax to facilitate the manipulation and management of SQL databases. It is one of the most popular RDBMS systems because it is free to download, easy to configure, and easy to use.

Downloading, Installing, and the Initial Setup of MySQL on a Microsoft Windows Computer System

From the MySQL website www.mysql.com select the "DOWNLOADS" menu option.

From the "DOWNLOADS" page select MySQL Community (GPL) Downloads, which you will find right at the bottom of the page.

On the next page select the "MySQL Community Server" option.

From the "Select Operating Systems" option, select the "Microsoft Windows" operating system from the drop-down selection box.

Click on the "MySQL Installer for Windows."

On the next page choose and "Download" the first installer on the page "Windows (x86, 32-bit), MSI Installer".

On the next page, there will be a few extra options to either "Login" or "Sign Up." Click on "No thanks, just start my download" near the bottom of the page.

The installer will download it.

Once it has downloaded, click on the "mysql-installer-web-community-8.0.21.0.msi" file in the "Downloads" folder.

If a pop-up box appears that asks "Do you want to allow this app to make changes," click on "Allow."

You may also be prompted to run an upgrade. Allow the system to run an upgrade.

The next screen is "Choosing a Setup Type."

It defaults to "Developer Default." Leave it at this option and click the "Next" button.

NOTE:

- If you get a screen that lists some MySQL install instances with an error about "Microsoft Visual C++ Redistributable 2019 not installed," exit the installation.
- Go to the following website: https://www.itechtics.com/microsoft-visual-c-redistributable-versions-direct-download-links/#6-microsoft-visual-c-redistributable-2019.
- Download the version the machine is asking for and restart the MySQL Community installation.

The "Check Requirements" screen has an option to connect to Python. Leave this option unchecked and click on the "Next" button. It is good to note that at this screen it may list some other packages such as "MySQL For Excel" or "MySQL for Visual Studio." Do not install any of these packages.

You will get a pop-up box that says, "One or more product requirements have not been satisfied." Click on "Yes."

The "Installation" screen will have a whole list of products that need to be installed on your system.

Click "Execute."

This will take a few minutes to install, and each application that is being installed will have a progress count next to it. Let the installation finish.

The installation is complete when all the products to be installed have a green checkmark next to them.

Click on "Next."

This takes you to the "Product Configuration" screen.

This screen lists products that have been installed that are now ready to be configured.

Click on "Next."

The next screen is called either "Group Replication" or "High Availability."

It will have the option to choose a "Standalone MySQL Server / Classic MySQL Replication" or "InnoDB Cluster."

Leave it on the default option which is the "Standalone MySQL Server / Classic MySQL Replication" option and click "Next."

At the "Type of Networking" options screen leave all the settings as they are (on the default settings) and click "Next."

At the "Authentication Method" screen leave the choice as the default method for authentication, "Use Strong Password Encryption for Authentication (RECOMMENDED)" and click on "Next."

At the "Accounts and Roles" screen you will need to set a password for the "root" user. Choose a strong password that you will remember. The "root" user is the admin user for MySQL. When you have created a password, leave the rest of the options as default and click "Next."

At the "Windows Service" screen you will leave all the default options and click on "Next."

At the "Apply Configuration" screen click on the "Execute" button and the system will apply all the settings. This will take a few minutes.

When the configuration is done there will be green check marks next to all the configuration steps. Click on "Finish."

At the "Product Configuration" screen click "Next."

At the "MySQL Router Configuration" screen, leave all the default settings and click "Finish."

At the "Connect To Server" screen, leave all the settings as the default settings. You do need to type in the root password you have just created in the "Password" box. Click on the "Check" box to see if it is correct. If you get a green check next to the "Check" button, click "Next."

At the "Apply Configuration" screen, click on "Execute."

When the configuration has successfully finished, click on "Finish."

At the next "Product Configuration" screen click on "Next."

The "Installation Complete" screen will appear and there should be two checked options on the screen which are "Start MySQL Workbench after Setup" and "Start MySQL Shell after Setup." Leave both of these checked and click on "Finish."

Two screens will open:

- A command prompt — In this tutorial, you will not be working with the command prompt to execute instructions so you can close this screen.
- "Welcome to MySQL Workbench" — You will be using MySQL Workbench, so you will need to leave this screen open for now.

On the "Welcome to MySQL Workbench" screen you should see MySQL Connections near the bottom of the page.

There should be a default connection:

> Local instance MySQL80
>
> root
>
> localhost: 3306

If you do not have the above or similar you can create it by:

> Next to "MySQL Connections" click on the + icon.
>
> You will be taken to the "Setup New Connection" screen.
>
> > - Connection Name: Local Connection (this is a name you can set up)
> > - Connection Method: Standard (TCP/IP) (this is the default connection type)
> > - Hostname: 127.0.01 (this is the localhost address)
> > - Port: 3306 (this is the default SQL port)
> > - Username: root
> > - Password: you will need to click on the "Store in Keychain" and type in the password you set up during the installation.
> > - Default Schema: Leave this blank

Test the connection by clicking on the "Test Connection" button at the bottom right-hand side of the screen.

You should see the "Successfully made the MySQL connection" screen.

Click on "OK" on the pop-up screen and again on the bottom right-hand side of the screen.

The next screen will be the "Welcome to MySQL Workbench" screen where you should now see the connection you set up under "MySQL Connections."

Whenever you open MySQL Workbench you will see this connection. To access the local MySQL server you will need to click on this connection.

The next screen will be the MySQL Workbench, and you will have completed setting up MySQL.

Downloading, Installing, and the Initial Setup of MySQL on a Mac Computer System

From the MySQL website www.mysql.com and select the "DOWNLOADS" menu option.

From the "DOWNLOADS" page select MySQL Community (GPL) Downloads, which you will find right at the bottom of the page.

On the next page click on "MySQL Community Server" option.

From the "Select Operating Systems" option, select the "macOS" operating system from the drop-down selection box.

Download the first option in the list "macOS 10.15 (x86, 64-bit), DMG Archive".

On the next page, there will be a few additional extra options to either "Login" or "Sign Up". Click on "No thanks, just start my download" near the bottom of the page.

The .dmg file will download.

Once the .dmg file has downloaded, open the directory where the file has been downloaded to.

Double-click on the "mysql-8.0.15-macos-x86_64.pkg" file which will launch the installation.

Click on "Continue" when the warning/installation box appears on the screen.

Click on "Continue" over the next few screens until you get to the Licence agreement.

Click "Agree" to the license.

Click "Continue" once again on the Installation type screen.

This will begin the installation of MySQL.

If you get a message "Installer is trying to install new software" and asks for a password, you must enter the password you use to access your computer.

The software will install.

You will get to the "Configure MySQL Server" screen. Here you will need to choose the first option "Use Strong Password Encryption". Click "Next".

In the "Please enter a password for the "root" user" you will need to enter a memorable password. This is the root user password, so make sure you can remember it. When you have entered your password click "Continue".

MySQL will continue to install and start the initial database setup.

You may need to enter your macOS password one more time.

When the "The installation was completed successfully" screen appears, click on the "Close" button.

Once the MySQL Community Server has downloaded you will need to download MySQL Workbench. This is the API interface software that allows you to communicate with the MySQL Community Server.

Go back to the "DOWNLOADS" page and select MySQL Community (GPL) Downloads once again.

On the next page select "MySQL Workbench".

From the "Select Operating Systems" option, select the "macOS" operating system from the drop-down selection box.

Download the first option which is "macOS (x86, 64-bit), DMG Archive".

On the next page, there will be a few extra options to either "Login" or "Sign Up". Click on "No thanks, just start my download" near the bottom of the page.

The .dmg file will download.

Once the .dmg file has downloaded, open the directory where the file has been downloaded to.

Double-click on the "mysql-workbench-community-8.0.15-macos-x86_6.dmg" file which will launch the installation.

This screen will be a bit different. You get to the "MySQL Workbench 8.0" screen. Beneath the title, there will be instructions "Drag the MySQL Workbench icon to the Applications folder".

Drag the MySQL Workbench Icon from the left side (in the small black box) to the blue box with the Application folder in it.

This will copy the file into the Applications folder.

Once the system has finished copying over the application, press the command spacebar to search for "MySQL Workbench".

Open the application.

You may get a warning box "MySQLWorkbench is an app downloaded from the Internet. Are you sure you want to open it?"

Click "Open".

On the next screen, you should see MySQL Connections near the bottom of the screen. There should be a default connection.

> Local instance 3306
>
> root
>
> localhost: 3306

If you do not have the above or similar you can create it by:

> Next to "MySQL Connections" click on the + icon
>
> You will be taken to the "Setup New Connection" screen
>
> - Connection Name: Local Connection (this is a name you can set up)
> - Connection Method: Standard (TCP/IP) (this is the default connection type)
> - Hostname: 127.0.01 (this is the localhost address)
> - Port: 3306 (this is the default SQL port)
> - Username: root
> - Password: you will need to click on the "Store in Keychain" and type in the password you set up during the installation.

- Default Schema: Leave this blank

Test the connection by clicking on the "Test Connection" button at the bottom right-hand side of the screen.

You should see the "Successfully made the MySQL connection" screen.

Click on "OK" on the pop-up screen and again on the bottom right-hand side of the screen.

The next screen will be the "Welcome to MySQL Workbench" screen where you should now see the connection you set up under "MySQL Connections".

Whenever you open MySQL Workbench you will see this connection. To access the local MySQL server you will need to click on this connection.

The next screen will be the MySQL Workbench and you will have completed setting up MySQL.

CHAPTER 3:

GETTING STARTED WITH SQL

For this tutorial, the exercises have been done on a computer running the Windows Operating system. As you are by now aware, there are many versions of SQL Server. For this book, we will be using MySQL and running most of the statements through MySQL Workbench.

Open MySQL Workbench.

Click on the MySQL Connection to access the database.

Your screen should look similar to the example below.
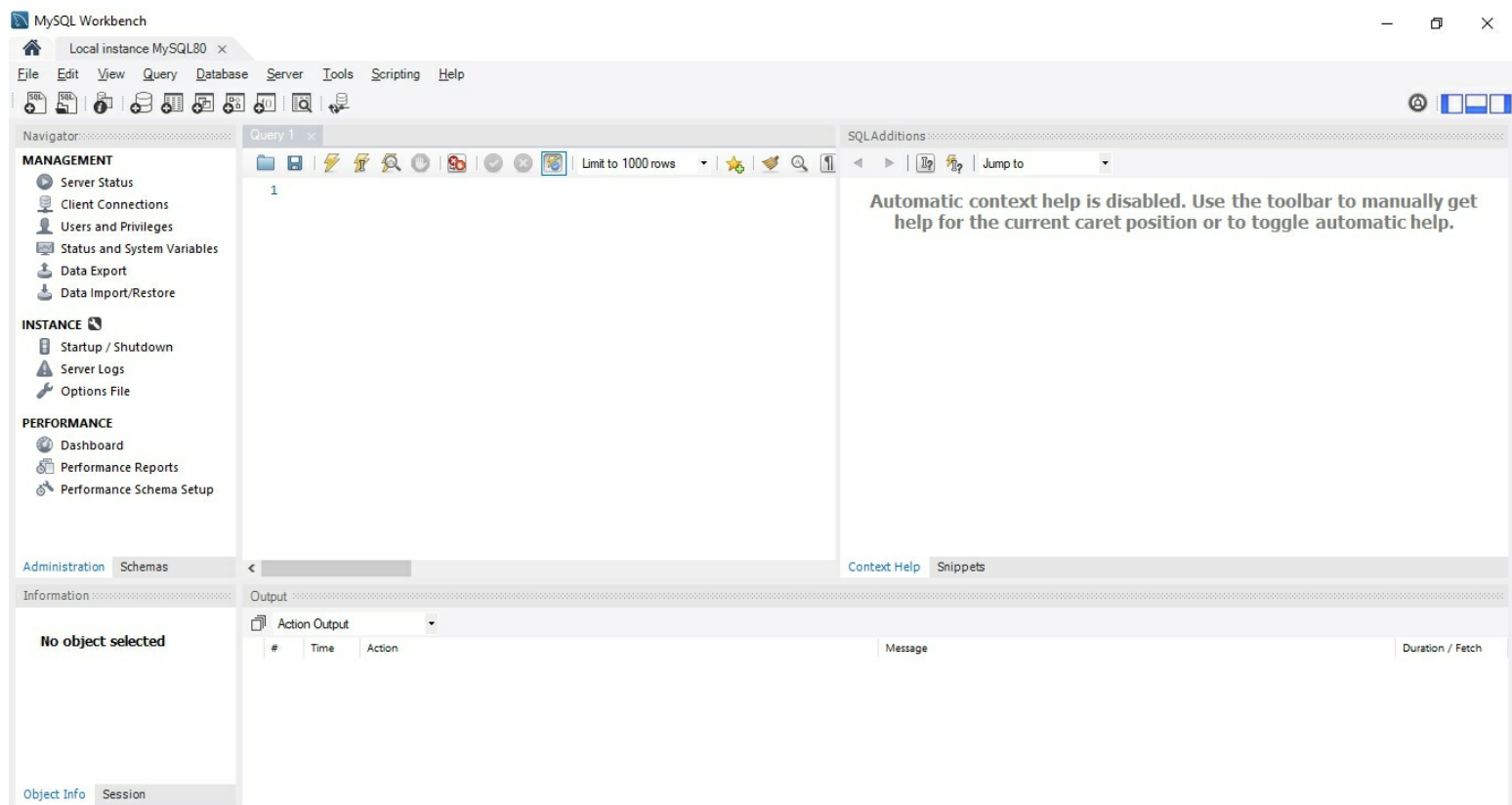
MySQL Screen

On the top left-hand corner of the screen is the "Home" icon which is represented by a house. If you click on the "House" icon it will take you back to the Welcome screen.

Next to the "House" icon, there is the "Local instance MySQL80" tab. This is the instance you are currently working in.

Below these tabs are the menu and quick access ribbon that will change to reflect the chosen menu option.

Below the menu ribbon on the left is the Navigation panel.

Below the menu ribbon in the middle is the Query Editor where you will be writing your queries.

Below the menu ribbon on the far right is the SQL additions screen.

At the bottom of the screen on the left, there is the Administration tab. This is for the administration of the database(s). You select this tab to start and stop the server, run various health checks, etc.

Schemas list the current databases that are running in the currently selected MySQL server instance. If you had to take a look at the Schemas now you would see the "sys" database. This is the internal database of the system. You may have two other test databases on the system, namely, "Sakila" and "World".

The Query tab or Instance tab is where you will be writing and executing queries.

Working With MySQL Databases

Creating a Database with MySQL Workbench

On the top right-hand corner of the screen, you will see three boxes each with a portion of the box colored in. These boxes hide various panels on the screen. This is handy to keep the screen clean and not confuse you.

Hide the bottom and right-hand panel on the screen. This will make the screen easier to use while you navigate SQL.
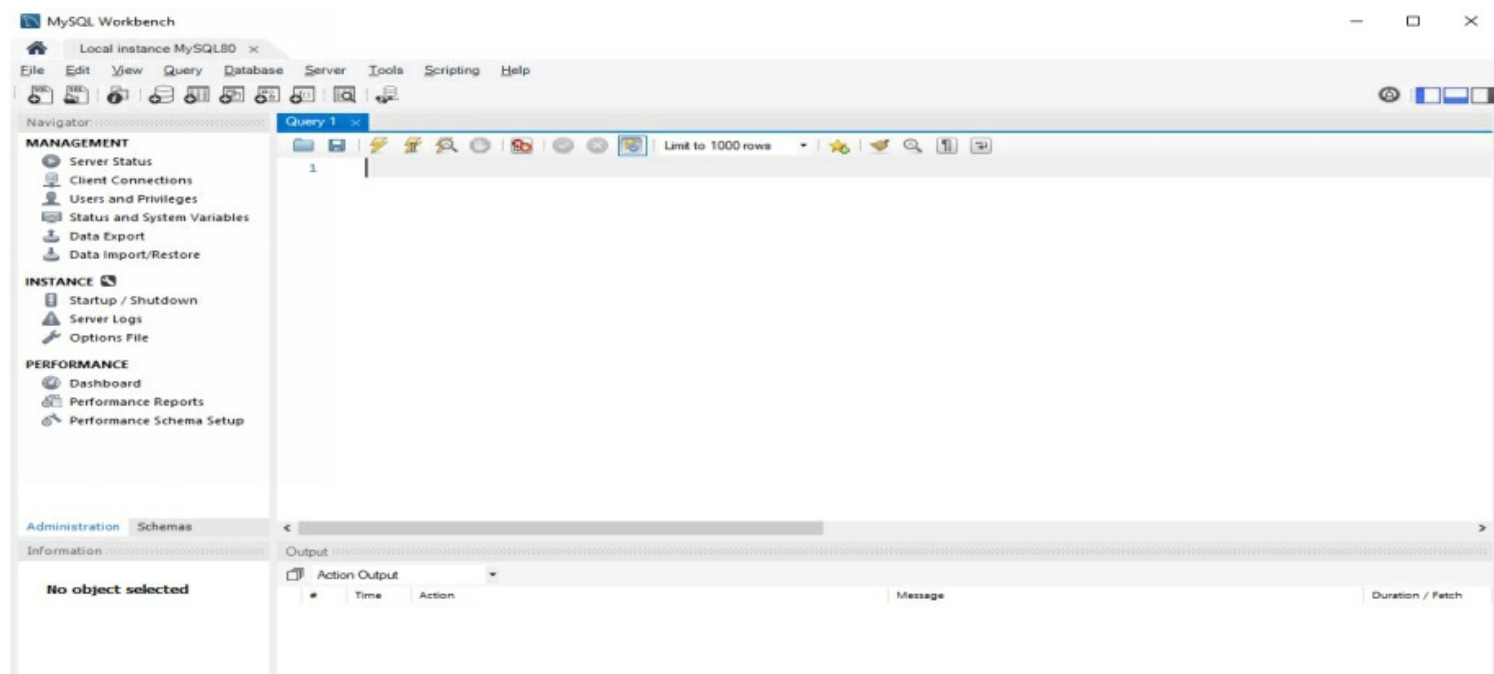
To create a database, start in the "Queries" tab and type the following:

create database test1;

This will create a database called test1. Each statement in SQL needs to end with or be separated by a semicolon (;). This allows for multiple statements to be executed within the same call to the SQL Server.

Click on the lightning bolt icon above the statements. The lightning bolt is what executes a selected script, or if there is no particular script selected it will execute all the statements.
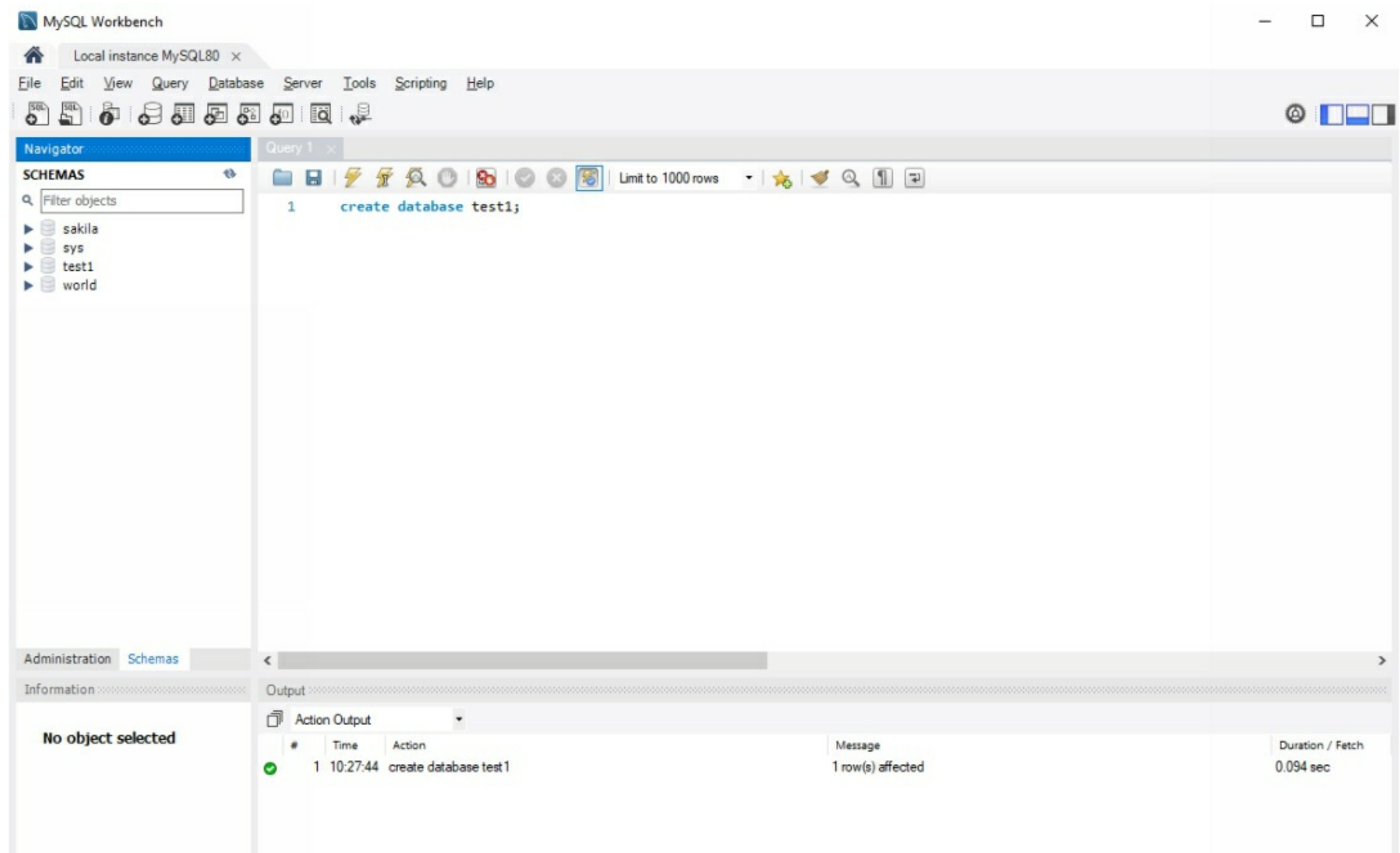
Make sure you have the "Schemas" tab selected.

Below the statement portion of the screen, you will find the "Output" section. This shows whether or not the execution of the statement was successful.

Refresh the "Schemas" tab and you should now see the test1 database listed.

Deleting a Database with MySQL Workbench

If you want to delete a database in MySQL, click on a new row in the "Queries" tab and type the following:

drop database test1;

Highlight the statement and then click on the Lightning bolt icon above the statement window. When you have a statement selected, when you click on the execute icon only that statement will be executed.

Working With Tables

In the chapter above you were introduced to the concept of tables. In this section, you are going to work with basic

table concepts.

Before moving on to the next section, create a database called School1.

If you have more than one database you will need to set the database you are working on as the default schema.

Highlight the database and right-click.

Choose "Set as Default Schema".

School1 should now be highlighted.

Types of Storage Engines Supported by SQL

When you create a database you want it to be optimized for the best performance, reliability, and stability. The handling of various SQL operations is done by an underlying SQL software component called a storage engine.

Usually, the default method is sufficient but at times, depending on the type of data and tables, you could run into performance issues.

There are a few different storage engine types each with their pros and cons. Some of these storage engines to consider are:

Archive

Archive is best for the storage of historical data that is seldom accessed or referenced. Archive is not designed for everyday database access.

- The tables are not indexed.
- Compression is done upon insert.
- Has no transaction support.
- It is best for the archiving and retrieval of historical archived data.

Blackhole

This engine is best in a testing environment where there is no need to store data.

- Blackhole will accept data.
- Blackhole does not store data.
- Useful if data is stored in a different location to the local database (distributed database environment).
- Useful for testing environments.
- Has a lot of similarities to UNIX/ dev/null.

CSV

If data needs to be sent to other applications such as Excel, the CSV storage engine stores data tables in a comma-delimited format text file(s). It is important to note that CSV tables do not get indexed. A way to get around this is to initially have an InnoDB storage engine and switch to CSV when you are ready to send the files to the program in CSV format.

- An excellent engine for sharing files to programs that are compatible with CSV format.
- CSV file formats are not indexed in SQL.

InnoDB

This is the default storage engine for MySQL and will create InnoDB tables as the default. InnoDB offers the following values to a database:

- It offers row-level locking.
- It offers integrity constraints for FOREIGN KEY.
- With the use of non-locking reads, multi-user concurrency is increased.
- It is fully compliant with ACID.
- It has commit, crash-recovery, and rollback.
- It gives overall excellent performance for nearly every type of database and can be used for the majority of applications.

MyISAM

ISAM has been around for a long time and is known for its speed.

- MyISAM is best suited for Data warehousing applications.

- MyISAM has no transaction support.
- It does offer full-text search indexes.
- MyISAM offers table-level locking.
- MyISAM performs best on read-heavy applications.
- MyISAM is not ACID compliant.

NDB

- NDB is also known as NDBCLUSTER.
- NDB is used for clustered environments.
- NDB is best suited for distributed computing environments.
- NDB operates well where an environment requires high-redundancy.
- NDB offers one of the highest uptimes and high availability.

Creating a Table

The first table to be created for the School1 database will be the Student table.

Type the following into the Query tab for the School1 database using MySQL Workbench.

> *create table Students(*
>
> *id int not null auto_increment,*
>
> *sname varchar(25),*
>
> *fname varchar not null (35),*
>
> *address varchar (45),*
>
> *marks int,*
>
> *primary key (id)*
>
> *);*

Highlight the entire statement and execute it.

Refresh the Schema and you will notice that Tables will now show the newly created "Students" table.

Adding Data to a Table

In this section, you are going to insert some data into the table so you have some information stored in the database.

Ensure you have the School1 database selected. You are going to be using the "**INSERT**", "**INTO**", and "**VALUES**" statements to insert data into the table. Starting in a new row in the Query tab type the following:

> *insert into students (sname, fname, address, marks)*
>
> *values ('brent', 'charles', '2 data road', 455);*

Highlight both lines and execute the statement.

To see if the data has been inserted into the table, use the "**SELECT**" query by typing in the following in a blank Query tab line:

> select * from students;

You should see a table similar to the example below showing the data you inserted into the table.
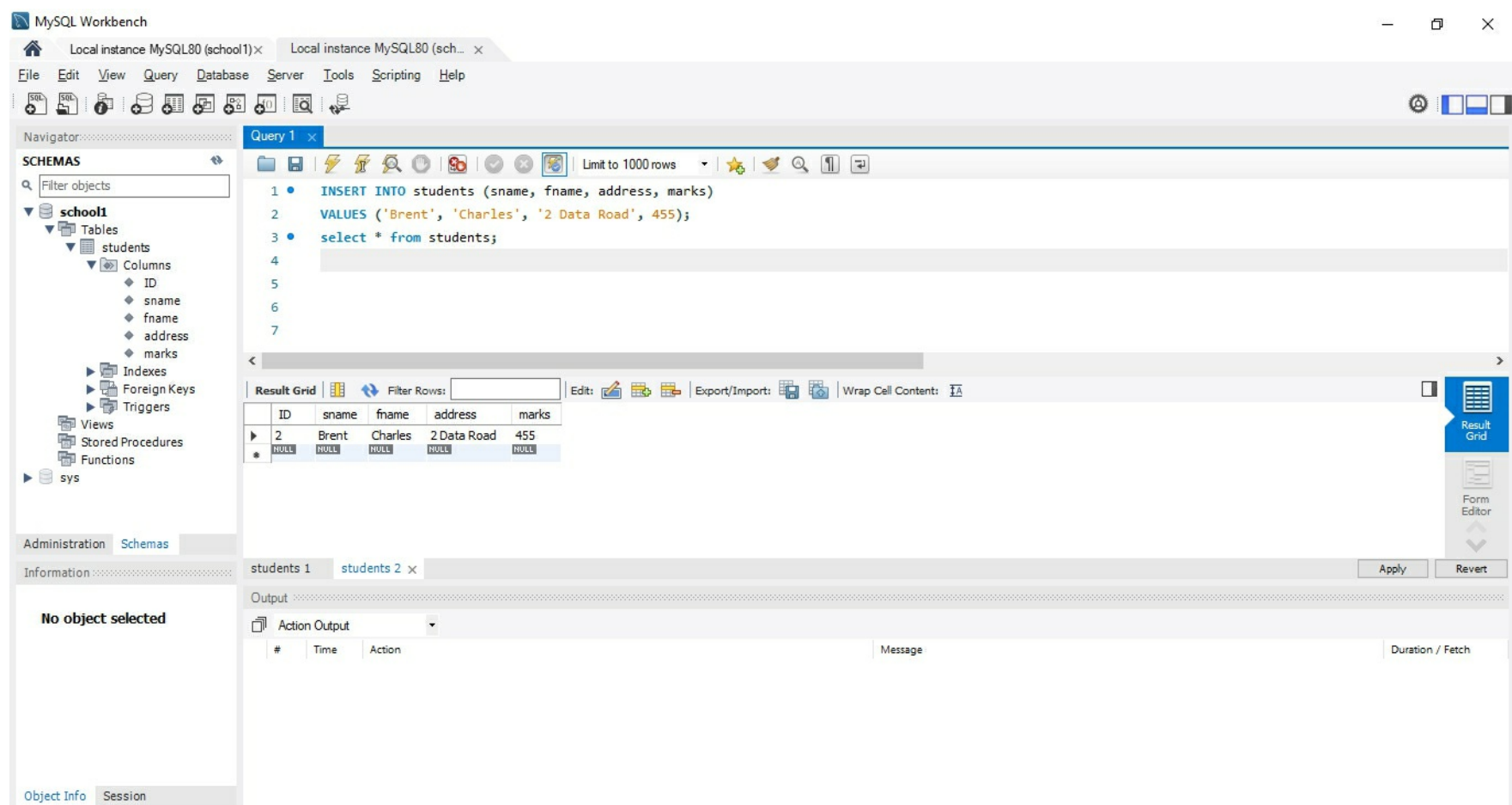
You will notice the green checkmarks in the Output section which shows that the statement ran through correctly and without error.

You will also notice that the ID column automatically added an ID number. This is because it was set to auto_increment when you set up the ID column. You can leave off auto_increment if you need to insert your ID into the field.

Exercise 1

Using the statements you have learned, complete the following tasks:

Delete the **School1** database.

> *drop database school1;*

Create a database called **Shop1**

> *create database Shop1;*

Set the **Shop1** database as the **default** schema.

Create the following **tables** and corresponding **fields**:

Customers

> **Fields**
>> ○ Cust_ID — auto_increment NOT NULL
>>
>> ○ Sname — varchar (35)
>>
>> ○ Fname — varchar (35)
>>
>> ○ Address — varchar (50)
>>
>> ○ Phone — varchar (20)

MySQL Statement

> *create table Customers(*
>
> *cust_id int not null auto_increment,*
>
> *sname varchar (35),*
>
> *fname varchar not null (35),*
>
> *address varchar (50),*
>
> *phone varchar (20),*
>
> *primary key (Cust_ID)*
>
> *);*

> **Data to Populate the Fields**

insert into customers (sname, fname, address, phone)

values ('barnes', 'jackie', '2 jupiter rd', '555-6565656');

insert into customers (sname, fname, address, phone)

values ('carter', ''john', '5 milky way', '555-7777777');

insert into customers (sname, fname, address, phone)

values ('pearce', 'sam', '15 uphill way', '555-8689894');

Products

Fields

- ○ Prod_ID — auto_increment NOT NULL
- ○ Product — varchar(45)
- ○ Descript — varchar(50)
- ○ Price — decimal (19, 2)

MySQL Statement

create table products(

prod_id int not null auto_increment,

product varchar(35),

descript varchar not null (50),

unit int,

price decimal (19,2),

primary key (prod_id)

);

Data to Populate the Fields

insert into products (product, descript, unit, price)

values ('apples', 'red and green apples', 1, 1.20);

insert into products (product, descript, unit, price)

values ('pears', 'green pears', 1, 1.25);

insert into products (product, descript, unit, price)

values ('oranges', 'large oranges', 1, 1.30);

insert into products (product, descript, unit, price)

values ('bananas', 'imported bananas', 4, 3.10);

Viewing Data in a Table

To view data in an individual table you can use the navigation bar on the left by selecting the table you want to view. When the table is selected you will notice that if you hover over the table, three icons appear to the right of it. Click on the tiny grid with a lighting bolt on it and the data the table has been populated with will appear in the "Results grid". It should look similar to the image below:

You will also notice that you have been switched to a new tab with the name of the table on it and the select statement appears in the Query screen.

CHAPTER 4:

DATA TYPES

Different types of data are captured, and each type can be used to manipulate the captured data in certain ways. A store would have data relating to its products such as the types of products, the products, the amount they have in stock, and the price. If you were going to put the store's data into a database you would have three different data type fields, namely:

- Product — This would be a character data type
- Amount of stock on hand — This would be an integer data type
- Price of the product — This would be a currency or decimal data type

There are many different data types and each one has a specific purpose. The more advanced you get with database development, the more advanced working with database types become.

SQL Data Type Categories

SQL data types can be broken into different data type categories, and these categories and the data types they represent are:

Approximate Numeric Data Types

SQL defines numbers to be either approximate or exact. Approximate numeric data types are floating-point numeric data types.

The data types for this category are:

FLOAT

Starting at - 1.79E + 308 and ending at 1.79E + 308

REAL

Starting at - 1.79E + 308 and ending at 1.79E + 308

Binary Data Types

A binary data type includes images, PDF files, video files, Word files, Excel files, etc. They can be set to store either a variable-length or fixed-length field to store these types of files.

The data types for this category are:

BINARY

This data type has a maximum length of 8 000 bytes, fixed length.

IMAGE

This data type has a maximum length of 2 147 483 647 bytes, variable length.

VARBINARY

This data type has a maximum length of 8 000 bytes, variable length.

Character Strings Data Types

Character string data is used for normal text or alphanumeric text fields.

The data types for this category are:

CHAR

This data type can store a mix of most of the data types such as numeric data that is not used in a calculation, text, and alphanumeric text. This data type has a maximum length of 8 000 characters but is a fixed-length data type.

VARCHAR

This data type can store a mix of most of the data types such as numeric data that is not used in a calculation, text, and alphanumeric text. This data type has a maximum length of 8 000 characters but is variable-length data.

Date and Time Date Types

If you have worked in any type of database or spreadsheet program you will know that there is a certain format for

inserting the date and time. You can use a character string field, but that will not enable you to do date-based calculations. It is always best to set a date and time field up as such.

The data types for this category are:

DATE

The way the date is stored can be set by the regional settings on the system the database is running on. This data type will store the date in the standard format, ex. May 02, 2020.

DATETIME

This will store both the date and time with a date range from Jan 1, 1753, to Dec 31, 9999.

SMALLDATETIME

This data type has a smaller date range window starting from Jan 1, 1900, to Jun 6, 2079.

TIME

This data type will store the time and can be set through the regional settings on the system to display the time in a customized format. An example of how it stores the time is: 13:00.

Exact Numeric Data Types

Exact numeric data types are data types where the scale and the precision of the data value have to be preserved.

The data types for this category are:

BIGINT

This data type takes 8 bytes of storage and represents 64-bit integers. The data type can range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 for signed values, while unsigned values have a minimum value of 0 and a maximum value of 2^{64}-1.

BIT

This data type is used to store bit values which can range from 1 to 64.

DECIMAL

Decimal is used to define data that requires exact numeric data such as money. You have to define the number of digits and decimals, for example *decimals (6, 2)* which will show as 000000.00. The values for this field range from -999.99 to 999.99. If a calculation returns more decimal points than have been specified, the decimal will be automatically rounded up to the nearest ten.

INT

This data type stores integer values, and it uses 4 bytes of storage. It has a minimum value of -2147483648 for signed values (data that can be a negative value), and a maximum value of 2147483647 for signed values. Unsigned values have a minimum value of 0 and a maximum value of 4294967295.

MEDIUMINT

This data type stores integer values, and it uses 3 bytes of storage. It has a minimum value of -8388608 for signed values (data that can be a negative value) and a maximum value of 8388607 for signed values. Unsigned values have a minimum value of 0 and a maximum value of 25516777215.

NUMERIC

The same rules that apply to decimal apply to numeric, as this data type is used in the same way as the decimal data type is.

SMALLINT

This data type stores integer values. It uses 2 bytes of storage. It has a minimum value of -32768 for signed values (data that can be a negative value) and a maximum value of 32767 for signed values. Unsigned values have a minimum value of 0 and a maximum value of 65535.

TINYINT

This data type stores integer values, uses 1 byte of storage, has a minimum value of -128 for signed values (data that can be a negative value) and a maximum value of 127 for signed values. Unsigned values have a minimum value of 0 and a maximum value of 255.

Default Data Type Values

A default value can be a literal constant or expression in SQL. Literal constants are known either as literals or constants and are values that cannot be altered during statement execution.

The example below creates a date column that takes the current date and subtracts 20 years:

date default (current_date - interval 20 year)

CHAPTER 5:

SQL STATEMENTS AND CLAUSES

One of the first things you need to understand when setting up an SQL database is the difference between an SQL Statement and SQL Clause.

SQL Statements

An SQL statement is the query or action part of the SQL command. For instance:

> *drop table school1;*

The SQL statement in the above example is "**DROP TABLE**" which will delete an entire table from the database.

The following are the most commonly used SQL statements:

ALTER DATABASE

This statement is used to add, delete, modify, or update file groups, files, and records associated with a database.

ALTER TABLE

This statement is used to add, delete, modify, or update tables in a database.

CREATE INDEX

This statement is used to fine-tune the database by creating an Index that allows for faster retrieval of the data in a database.

CREATE TABLE

This statement is used to create a new table in a database.

CREATE TABLE AS

This statement is similar to using the "File Save As" command in Word or Excel. It allows you to create a new table using the structure and data of an existing table.

DELETE

This statement is used to delete data/records from a table.

DROP DATABASE

This statement is used to delete an entire database from the SQL Server.

DROP INDEX

This statement is used to delete an index in a database.

DROP TABLE

This statement is used to delete an entire table from a database.

INSERT INTO

This statement is used to insert data into a table.

SELECT

This statement is used to retrieve certain or all records from a table.

SELECT LIMIT

This statement is used to retrieve a limited number of records from a table.

SELECT TOP

This statement is used similar to the SELECT LIMIT statement and will return only the top records in a given query.

TRUNCATE TABLE

This statement is used to delete all the records from a table without any rollback.

UPDATE

This statement is used to update the records in a table.

SQL Clauses

Some statements will be followed by a clause or condition of a query, for example:

select price from products where x = 2.30;

In the above example, the query is selecting a price from the product table that must equal 2.30. This example has the "**SELECT**" statement followed by the "**FROM**" and "**WHERE**" clauses/conditions.

The following are the most commonly used SQL clauses:

DISTINCT

This clause is used to retrieve distinctive/unique records from a table.

FROM

This clause is used to combine information or list information in tables.

GROUP BY

This clause is used to group information by one or more columns in a table.

HAVING

This clause is used to only return certain groups or rows in a query.

ORDER BY

This clause is used to sort information in a query, much like the sort by command in Word, Excel, etc.

WHERE

This clause is used to filter results by given criteria.

CHAPTER 6:

SQL EXPRESSIONS, FUNCTIONS, AND OPERATORS

To work with a database and the tables it houses, you need a way to manipulate the data to return the output required. You create a database to not only store, view, and retrieve the data, but to be able to manipulate the data to make it useful.

Take the "Shop1" database you created in an earlier chapter of this book. You created two tables, a customer table to hold customer records and a product table to hold product information. These tables can do more than just collect information about customers and products. The data can be manipulated to create even more useful information such as orders, invoices, and so on.

Expressions, functions, operators, and conditions allow you to perform specific tasks to create various outcomes.

Expressions

Expressions are usually called value expressions in SQL. An expression can be complex or extremely simple. It can contain statements, values, columns, operators, etc. It is even possible to combine multiple expressions into one expression, as long as the outcome of the expression reduces to a single value. An expression can be likened to a formula that is written as a query and as such can be used to query tables for specific sets of data.

There are five different kinds of expressions in SQL:

Conditional Value Expressions

A conditional value expression is one of the more complex types of expressions as the value of the expression depends on a condition. Although they are for advanced SQL, you should be aware of the following conditional expressions:

CASE

NULLIF

COALESCE

Datetime Value Expressions

The DateTime value expressions will always return date and time as these types of expressions are used to manipulate and perform date and time operations.

The following are common expressions of this type:

DATE

INTERVAL

TIME

TIMESTAMP

To write an expression that returns data for a month from today, it would look similar to the following:

current_data + interval '30' day

Interval Value Expressions

The interval value expression is used for scenarios such as subtracting one date from another date to get an interval time. For instance, subtracting your age from the current year will give you your birth year. The expression would look similar to the following:

date (current_date - interval 20 year) month to year

Numeric Value Expressions

Numeric value expressions require the data to be numeric values of which arithmetic operators can be used to formulate the result.

For example:

12 * 24

-109

25/5

The field name can be specified as long as the outcome equates to a numeric value, for example:

> price * qty
>
> price/5
>
> Oranges + Apples + Pears

String Value Expressions

String value expressions are used to concatenate expression values. The concatenate expression is referenced by two vertical lines — ||

For example, if you had the following expressions:

> *'grey'*
>
> *'goose'*

You can concatenate them into a single expression:

> *'grey' || ' ' || 'goose'*

Instead of two separate expressions the above will return:

> *'grey goose'*

Operators

An SQL operator is a reserved character, arithmetic operator, or reserved word that can be used to perform various arithmetic or comparison functions in a statement. There are different types of SQL operators, namely:

Arithmetic

Addition

The "+" symbol is used for addition.

Example:

> If variable a = 5 and variable b = 10
>
> $$a + b$$
>
> will return:
>
> *15*

Division

The "/" is used for division.

Example:

> If variable a = 25 and variable b = 5
>
> $$a / b$$
>
> will return:
>
> *5*

Modulus

The "%" symbol is used for modulus.

Example:

> If variable a = 100 and variable b = 10
>
> $$b \% a$$
>
> will return:
>
> *10*

Multiplication

The "*" symbol is used for multiplication.

Example:

> If variable a = 10 and variable b = 8
>
> $$a * b$$
>
> will return:
>
> *80*

Subtraction

The "-" symbol is used for subtraction.

Example:

If variable a = 50 and variable b = 15

a - b

will return:

35

Comparison

Greater Than

The ">" symbol is used for greater than.

Example:

If variable a = 25 and variable b = 5

(a > b)

will return:

True

Less Than

The "<" symbol is used for less than.

Example:

If variable a = 25 and variable b = 5

(a < b)

will return:

False

Equal

The "=" symbol is used for equal to.

Example:

If variable a = 25 and variable b = 25

(a = b)

will return:

True

Not Equal

The "!=" or "<>" symbols are both used to represent not equal to.

Example:

If variable a = 25 and variable b = 5

(a != b)

will return:

True

or

(a <> b)

will return:

True

Greater Than or Equal

The ">=" symbols are used for greater than or equal to.

Example:

If variable a = 25 and variable b = 5

(a >= b)

will return:

True

Less Than or Equal

The "<=" symbols are used for less than or equal to.

Example:

If variable a = 25 and variable b = 5

(a <= b)

will return:

Not Greater Than

The "!>" symbols are used for not greater than.

Example:

If variable a = 25 and variable b = 5

(b !> a)

will return:

True

Not Less Than or Equal

The "!<" symbols are used for not less than or equal to.

Example:

If variable a = 25 and variable b = 5

(a !< b)

will return:

True

Logical

ALL

Compares all values in one data set to a specified value.

AND

This operator is used with the WHERE clause to allow for multiple conditions in a statement.

ANY

Compares a value to similar values in a referenced list.

BETWEEN

This operator is used to return values that are between a minimum and maximum value.

EXISTS

This operator is used to search for an existing row in a table that has been specified that meets a specific criterion.

IN

This operator compares specified literal values in a given list.

LIKE

This operator uses wildcard operators to compare similar values in a table or list.

NOT

This operator negates the logical operator used alongside it. For example:

not in

not between

not unique

not exist

OR

This operator is used to string together multiple conditions when used with the WHERE clause in a statement. The value can be x or it can be y, for instance.

IS NULL

This operator compares NULL values with a value.

UNIQUE

This operator is used to search for duplicates in a table.

Functions

SQL has a set of built-in functions and can facilitate user-defined functions. Functions are used to manipulate and process data and are used throughout an SQL database.

Some of the more commonly used built-in functions in SQL are:

AVG

This function returns the average of an expression.

CONCAT

This function will concat string values.

COUNT

This function will count the number of rows in a database.

MAX

This function will return the maximum value of a given range of values.

MIN

This function will return the minimum value of a given range of values.

RAND

This function will generate a random number.

SQRT

This function will return the square root of a specified value.

SUM

This function will return the sum of a given range of values

CHAPTER 7:

WORKING WITH CONSTRAINTS

A constraint is a rule that is placed on an entire table, column, or selection of columns. It restricts what types of data may be added to the table or column.

To ensure the integrity of the data being stored in a cell, table, or database, the developer needs to assign certain constraints to it. There are two levels of constraints in SQL and these are:

- **Column level** constraints which will only affect a single column.
- **Table level** constraints which will affect the entire table.

Constraints can be set when the table is first created or after the table has been created.

Commonly Used SQL Constraints

CHECK

This constraint ensures that the data that is imputed into a cell/column meet and satisfy the criteria that have been set for that column.

An example of using the CHECK constraint would be in a database that has an age group selection. For a health and fitness database, you may want to set different exercise regimes per age group.

The CHECK constraint can be used to limit a column's value between two age groups such as 35 to 45.

Example:

```
create table exercises (
                cust_id int not null auto_increment primary key,
                name varchar (45) not null,
                age int not null check (age >=35 and age<= 45)
        );
```

The above example:

- Creates a table called exercises.
- It populates the table with the following fields:
    - **cust_id** — This is the primary key, the number is automatically inserted upon creation of a record. It will be created in the form of an integer data type. The field cannot be blank.
    - **name** — This is a variable character field that is set to forty-five characters long, the field cannot be blank.
    - **age** — This field is an integer field that cannot be blank and will only accept values from 35 up to and including 45.

DEFAULT

This constraint is used to insert or specify the default value(s) for a column. The value is automatically inserted into the column by the database engine.

Example:

```
create table product (
                prod_id int not null auto_increment primary key,
                product varchar (30) not null,
                price decimal (3.2) not null,
                qty int (5) not null default '4'
        );
```

The above example:

- Creates a table called product.
- It populates the table with the following fields:

- **prod_id** — This is the primary key, the number is automatically inserted upon the creation of a record. It will be created in the form of an integer data type. The field cannot be blank.
- **product** — This is a variable character field that cannot be blank.
- **price** — This field is a decimal field set to 3 digit and 2 decimal places. The field cannot be blank.
- **qty** — This field is an integer field that will assign the default quantity of the product. For the particular example, each product comes in a quantity of four which will automatically be assigned upon creation of a new record in the table.

FOREIGN KEY

This is a unique identifier that is used to identify a related record in a different table.

Example:

In Chapter 3 of this book under the "Table Exercise" section, you created the "shop1" database. You also created two tables for the shop1 database:

- Customers
- Products

Each of the above tables had a unique primary key:

- Customers table primary key = cust_id
- Products table primary key = prod_id

If you were going to have an ordering system you would need to create another table. For this example, the new table will be called orders.

For the ordering system to work, the orders table needs to pull information from the customers and the products table. The way the system does this is by referencing the primary keys from both tables which become known as a FOREIGN key to the orders table.

Take a look at the code below:

```
create table orders (

        order_id int not null auto_increment primary key,

        cust_id int,

        prod_id int,

        foreign key (cust_id) references customers (cust_id),

        foreign key (prod_id) references products (prod_id)

);
```

The above example:

- Creates a table called orders.
- Populates the table with the following fields:
  - **order_id** — This is the primary key, the number is automatically inserted upon the creation of a record. There can only ever be one primary key in a table.
  - **cust_id** — This is the column that references the cust_id in the customer's table.
  - **prod_id** — This is the column that references the prod_id in the products table.
  - **FOREIGN KEY** (cust_id) REFERENCES customers (cust_id) — This sets the first foreign key in the order table to reference the customer's table to link to the cust_id field in the table.
  - **FOREIGN KEY** (prod_id) REFERENCES products (prod_id) — This sets the first foreign key in the order table to reference the products table to link to the prod_id field in the table.

There can be more than one foreign key in a table. In MySQL Workbench the newly created order table will reflect two foreign keys and look similar to the image below.
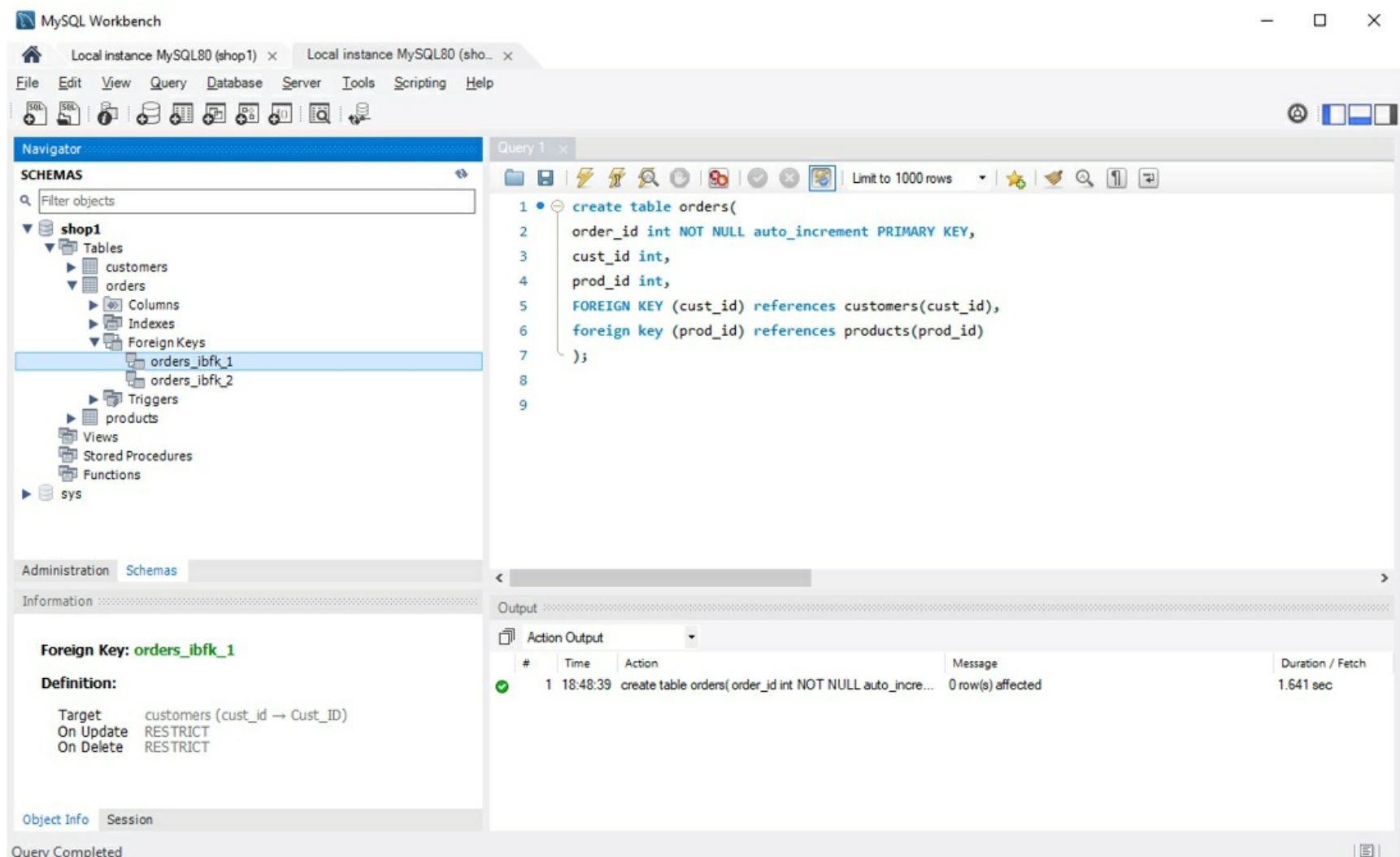
The relationship table will look similar to the diagram below.
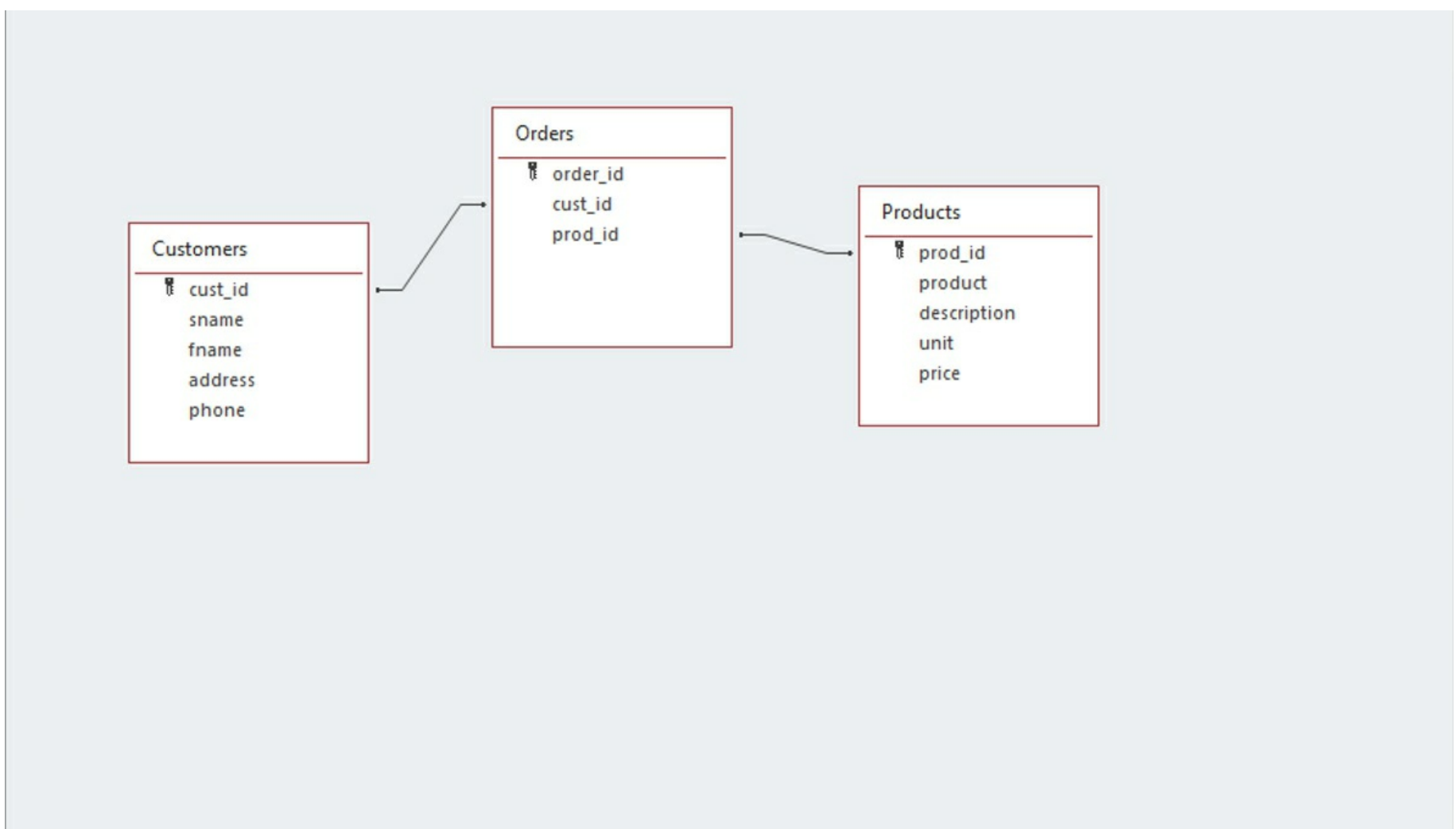
NOT NULL

When you see a NULL value in a table it means that the cell/record/column is blank. In other words, it does not contain any values or data, it has a NULL value. The NOT NULL constraint is used to ensure that the cell is not blank. It will create an error if no data is entered into it.

PRIMARY KEY

The primary key is the unique ID that is given to each record in every row of every table in a database. You can set the primary key to auto_increment which will allow the database engine to automatically generate the ID. You can also set it to be anything you want it to be and manually enter it.

For instance, your customer ID may be alphanumeric, ABC001. Always make sure the primary key field is set to NOT NULL to ensure a value is placed in the field.

There can only be one primary key column per table.

Adding Constraints

To add a constraint during the initial setup of a table the syntax would be similar to:

id int not null auto_increment,
***primary key** (id)*

Altering Constraints

To alter an existing constraint in a table the syntax would be similar to:

*Alter table students drop constraint id **primary key**;*

Exercise 2

Before continuing on to the next chapter you are going to create a new database called "hr" using some of the information you have learned so far.

The HR Database

Starting on a new query line, type the following:

create database hr;

Execute the command.

Starting on a new query line type the following:

create table salaries (
sal_id int not null auto_increment,
jobtitle varchar (45),
joblevel varchar (10),
salary int not null,
primary key (sal_id)
);

Execute the command lines.

Starting on a new query line type the following:

create table departments (

dept_id int not null auto_increment,

dept varchar (40) not null,

primary key (dept_id)

);

Execute the command lines.

Starting on a new query line type the following:

create table employees (

emp_id int not null auto_increment,

name varchar (55) not null,

age int not null,

primary key (emp_id)

);

Execute the command lines.

Starting on a new query line type the following:

insert into departments (department) values ('IT Department');

insert into hr.departments (department) values ('Accounting Department');

insert into hr.departments (department) values ('Helpdesk Department');

insert into hr.departments (department) values ('HR Department');

insert into hr.departments (department) values ('Snr Management');

insert into hr.employees (name, age) values ('John Green', '35');

insert into hr.employees (name, age) values ('Sandy Smith', '28');

insert into hr.employees (name, age) values ('Terry Peach', '31');

insert into hr.employees (name, age) values ('Jack Sunny', '47');

insert into hr.employees (name, age) values ('Dru Red', '51');

insert into hr.employees (name, age) values ('Jane Web', '30');

insert into hr.'salaries (jobtitle, joblevel, salary) values ('Administration', 'Entry 1', '25000');
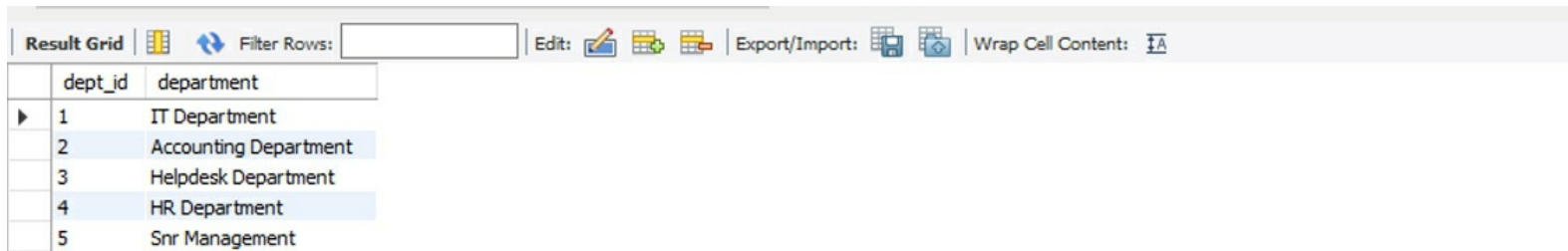
insert into hr.salaries (jobtitle, joblevel, salary) values ('Clerk 1', 'Entry 2', '28000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Clerk 2', 'Mid 1', '31000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Management', 'Mid 2', '35000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Engineer 1', 'Entry 3', '26000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Engineer 2', 'Mid 2', '32000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Engineer 3', 'Mid 3', '40000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Sn Manager 1', 'Snr 1', '43000');

insert into hr.salaries (jobtitle, joblevel, salary) values (''Director', 'Snr 2', '55000');

Execute all the "insert into" command lines

Starting a new query tab type the following:

*select * from hr.departments*

Execute the command and the "Results Grid" should look similar to the image below
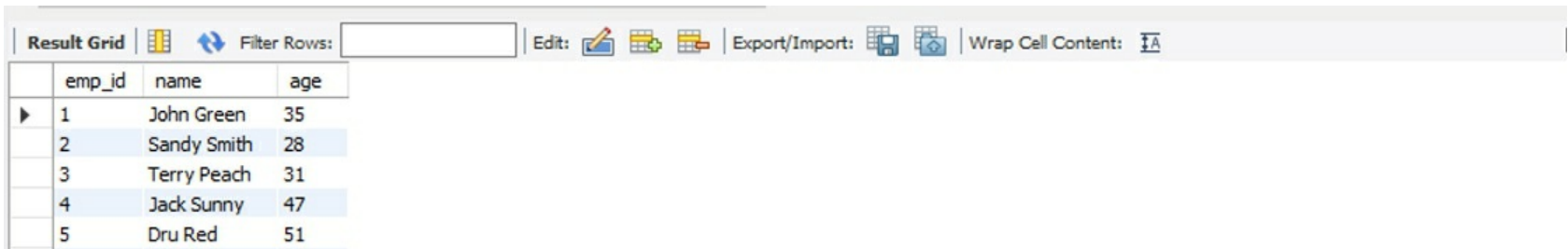
dept_id	department
1	IT Department
2	Accounting Department
3	Helpdesk Department
4	HR Department
5	Snr Management

Starting a new query tab type the following:

*select * from hr.employees*

Execute the command and the "Results Grid" should look similar to the image below
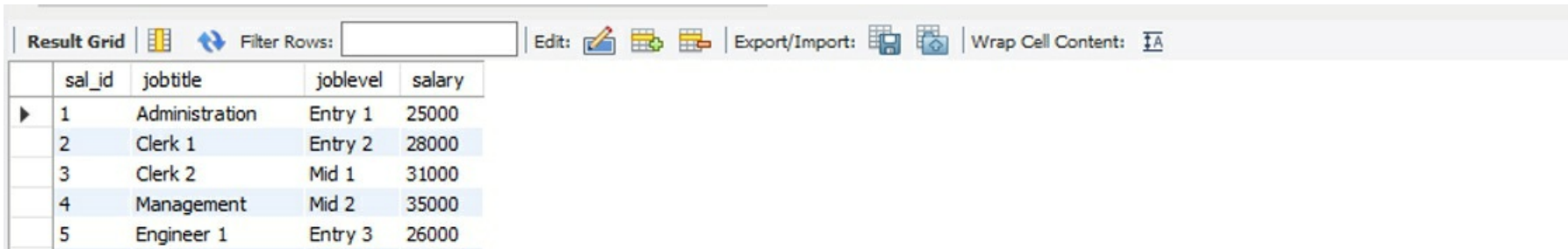
emp_id	name	age
1	John Green	35
2	Sandy Smith	28
3	Terry Peach	31
4	Jack Sunny	47
5	Dru Red	51

Starting a new query tab type the following:

*select * from hr.salaries*

Execute the command and the "Results Grid" should look similar to the image below

sal_id	jobtitle	joblevel	salary
1	Administration	Entry 1	25000
2	Clerk 1	Entry 2	28000
3	Clerk 2	Mid 1	31000
4	Management	Mid 2	35000
5	Engineer 1	Entry 3	26000

CHAPTER 8:

JOINS, UNIONS, ORDERING, GROUPING, AND ALIAS

In a database, you are going to want to combine the information from one or more tables into one table. You can do this using *Join* and *Union*, but it is important not to get the two confused. In this chapter, you are going to learn the basics of *join* and *union*, and how they work.

JOINS
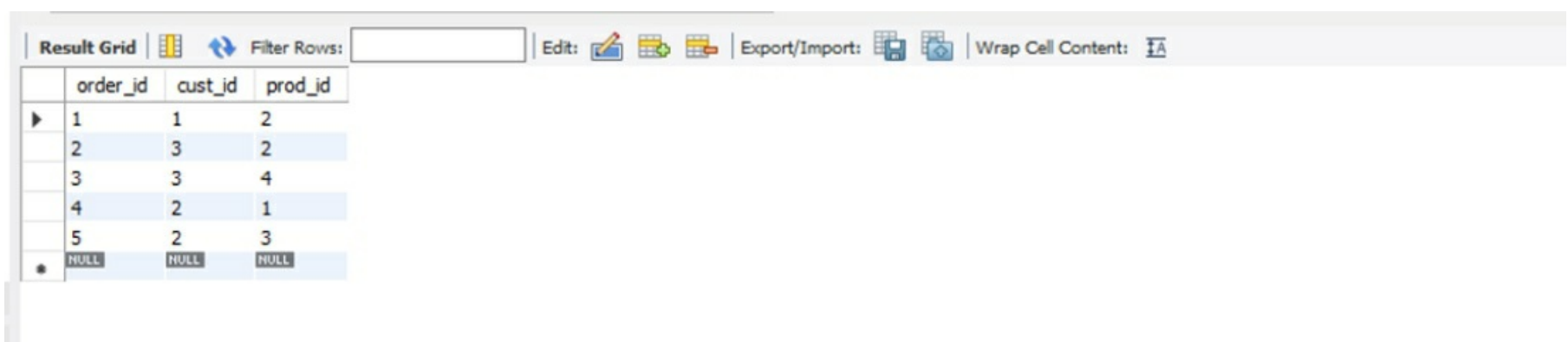
Consider the *shop1 database* created in an earlier chapter. There is the *products table* and the *customers table*. These tables collect valuable information about the store's products and customers. In Chapter 7 you created the *orders table* which references the *products table* and *customers table.* The information stored in the *orders table* does not show much information about either table except for ID codes.

Before you start working with joins, populate the *orders table* with some example data.

In a new "Query" line, insert the following data into the *orders table*:

> *insert into shop1.orders (cust_id, prod_id) values ('1', '2');*
>
> *insert into shop1.orders (cust_id, prod_id) values ('3', '2');*
>
> *insert into shop1.orders (cust_id, prod_id) values ('3', '4');*
>
> *insert into shop1.orders (cust_id, prod_id) values ('2', '1');*
>
> *insert into shop1.orders (cust_id, prod_id) values ('2', '3');*

Currently the *orders table* returns information similar to the table below:
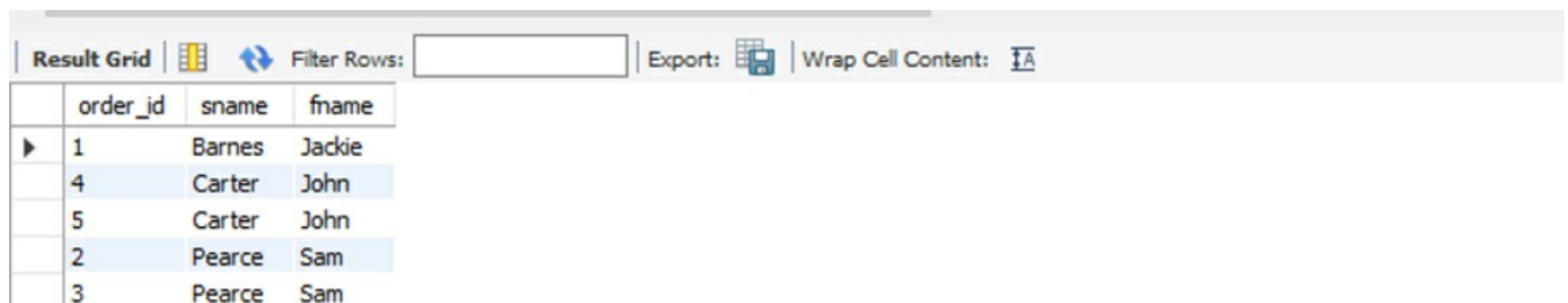

The *orders table* would not make much sense to someone who did not know all the customer and product identification numbers. When dealing with a large customer and product databases with long lists of customers and products, only seeing the codes is not going to be useful.

The information that gets filtered into the database can be selected by joining tables. For this exercise use the *shop1 database* to *join* information between the *orders table* and the *customers table*.

Using MySQL Workbench, set the *shop1 database* as the *default* database by typing the following into a new "Query" line:

> *select order_id, sname, fname*
>
> *from orders*
>
> *join customers on orders.cust_id=customers.cust_id*

Execute the script and the "Results Grid" should look similar to the one in the image below.

The *orders table* now reflects the customer's details and not just the *cust_id*.

Notice that there is no *cust_id* field. This field may be required, especially for reporting purposes.

If the *cust_id* field had been included in the following script, the script would not have run:

> select order_id, cust_id, sname, fname
>
> from orders
>
> join customers on *orders.cust_id=customers.cust_id*

Instead, it would have thrown out an error similar to the example below:

> "Error code: 1052 Column 'emp_id' in field list is ambiguous"

This error occurs because the *cust_id* field appears in both tables being joined. To get around this error and show the *cust_id* type the code as follows:

> select order_id, orders.cust_id, sname, fname
>
> from orders
>
> join customers on orders.cust_id=customers.cust_id

The *cust_id* field will now appear in the query.

	order_id	cust_id	sname	fname
▶	1	1	Barnes	Jackie
	4	2	Carter	John
	5	2	Carter	John
	2	3	Pearce	Sam
	3	3	Pearce	Sam

Types of Joins

There are different types of SQL joins, and each will return a different result when used in a table.

INNER JOIN
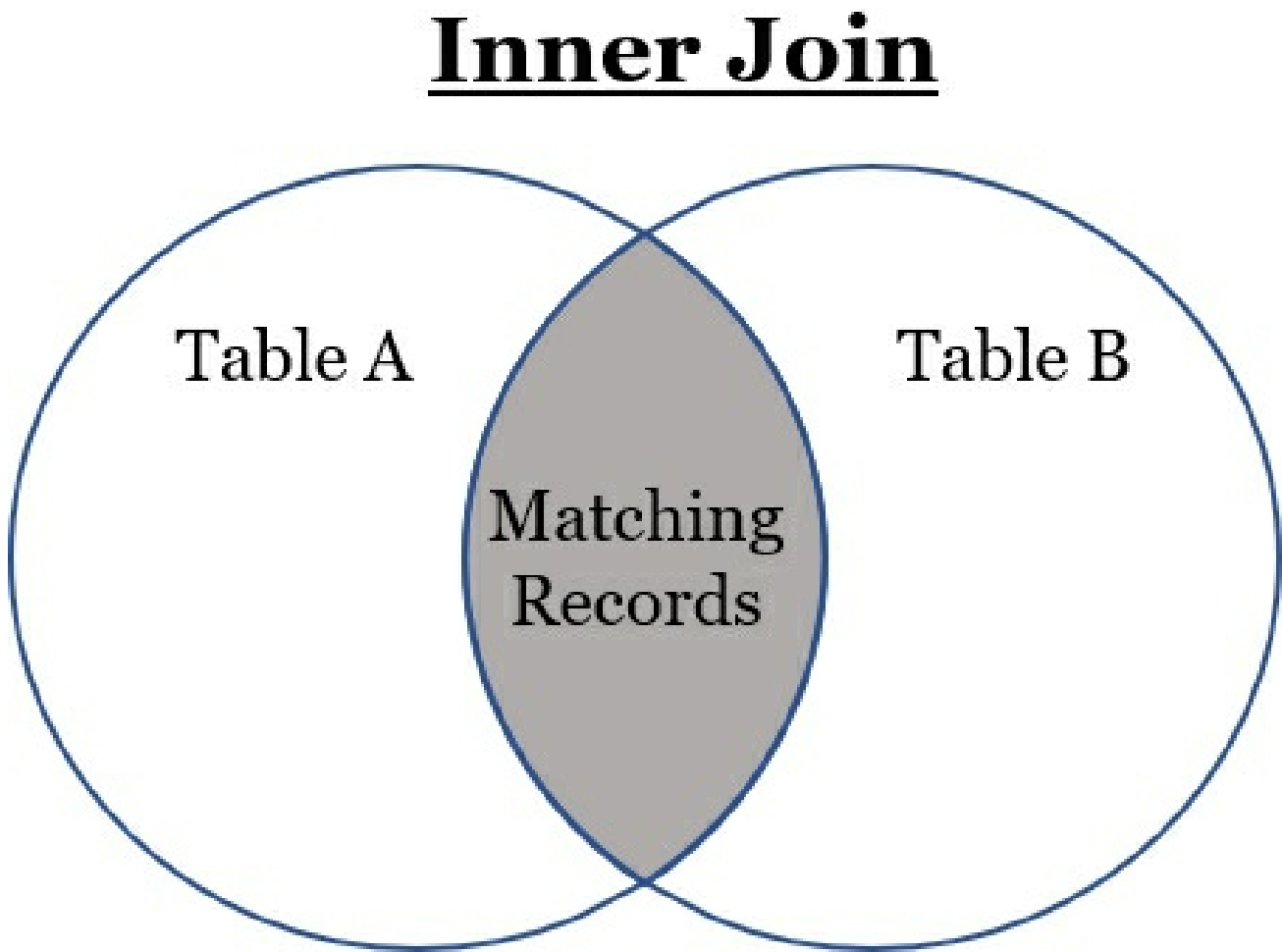
Join and *inner join* are the same function. If you are working in a database with different types of joins, using inner join makes the statement a lot clearer.

The *inner join* or *join* will only return records that are a match in both *Table A* and *Table B*.

Example:

Using the hr database created in Exercise 2, type the following in a new "Query" line:

> *create table workforce (*
> wf_id int not null auto_increment primary key,

emp_id int,
dept_id int,
sal_id int,
foreign key (emp_id) references employees (emp_id),
foreign key (dept_id) references departments (dept_id),
foreign key (sal_id) references salaries (sal_id)
);

Execute all the command lines.

Starting on a new "Query" line, type the following:
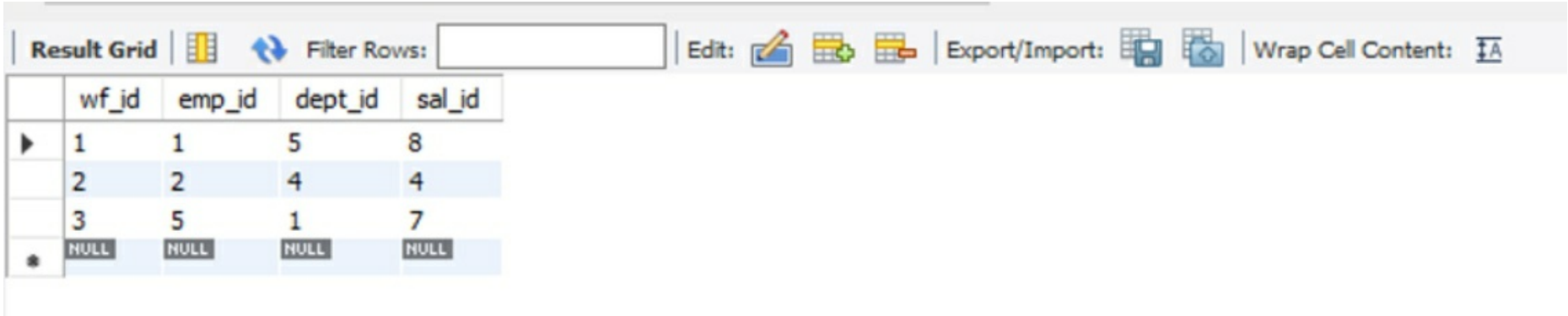
> *insert into workforce (emp_id, dept_id, sal_id) values ('1', '5', '8');*
> *insert into workforce (emp_id, dept_id, sal_id) values ('2', '4', '4');*
> *insert into workforce (emp_id, dept_id, sal_id) values ('5', '1', '7');*

Execute all the command lines.

Run the following query:

> *select * from hr.workforce*

Execute the command line and the results will be similar to the following image:
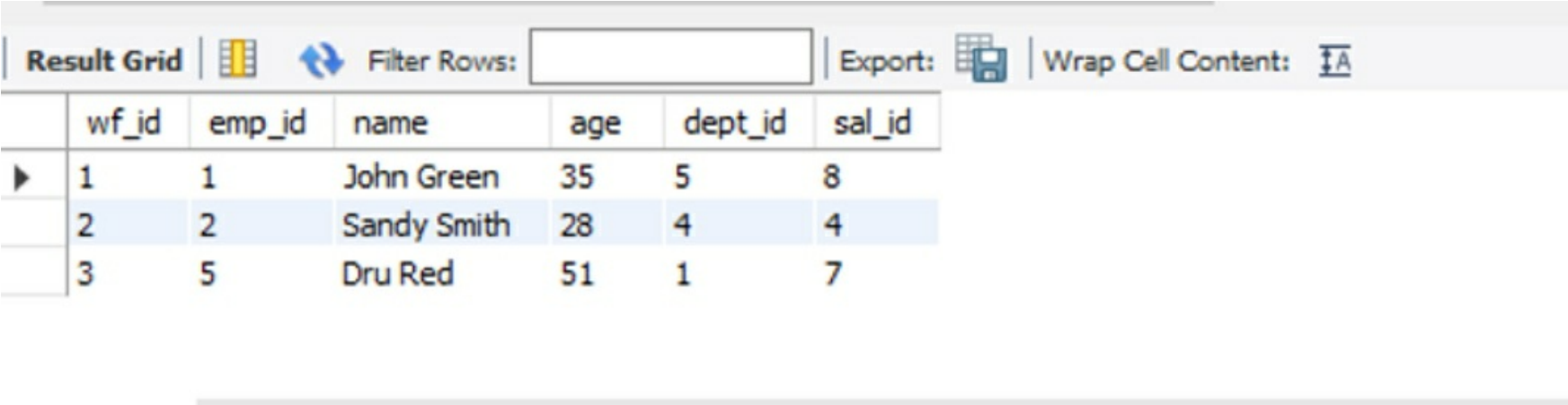
Using an *inner join* to populate the *workforce table* with the *emp_id, name,* and *age* from the *employees table*, type the following code in a new "Query" line:

> *select wf_id, employees.emp_id, name, age, dept_id, sal_id*
> *from workforce*
> *inner join employees on* **workforce.emp_id=employees.emp_id**

The "Results Grid" should show the *workforce table* as below:

The *inner join* returns only the records from the *workforce table* that match the records from the *employees table*.

FULL JOIN (OUTER)

Full Join
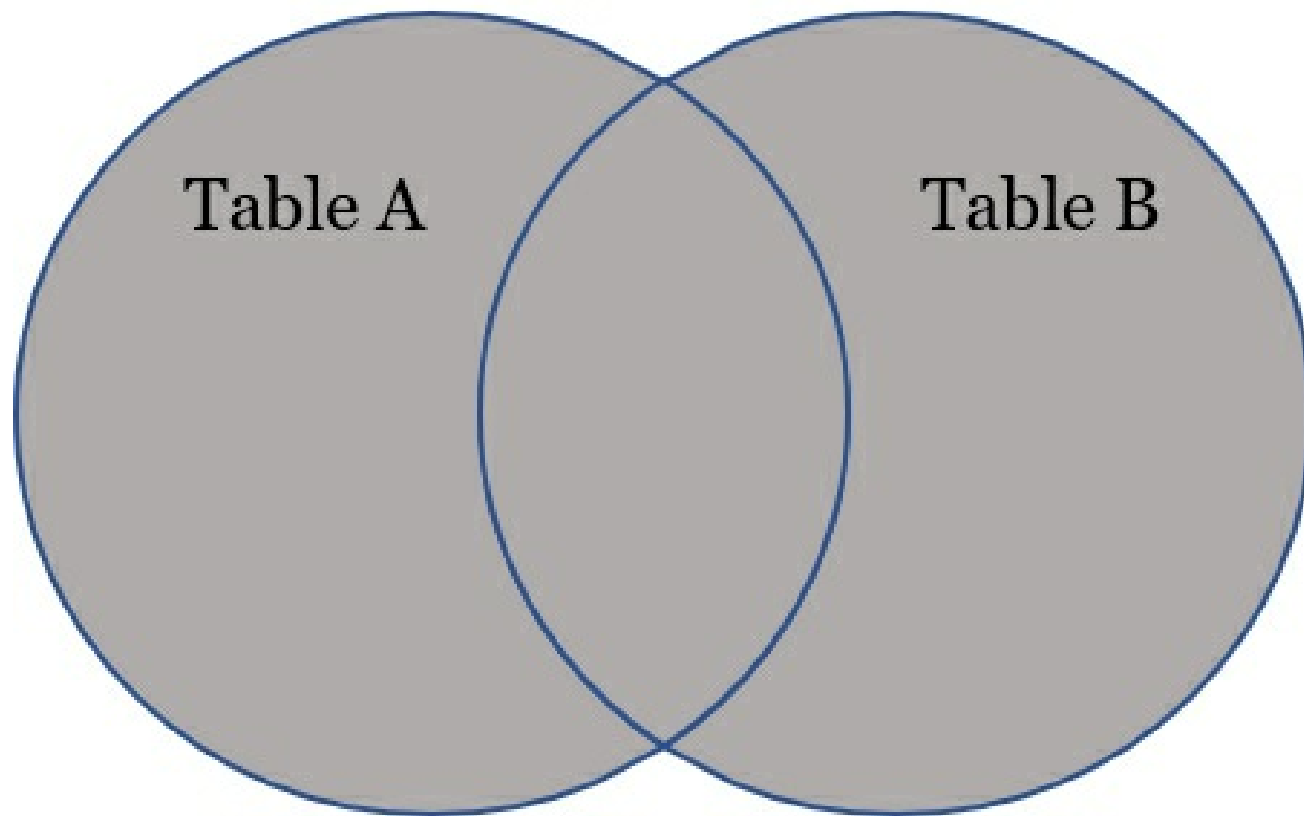
The *full join* is also known as the *outer* or *full outer join*. It will show all the records in both *Table A* and *Table B*, regardless of whether they match or not.

To use the *full join* to populate the *workforce table* with the *emp_id*, *name*, and *age* fields from the *employees table*, the code should look similar to the following:

> *select wf_id, employees.emp_id, name, age, dept_id, sal_id*
>
> *from workforce*
>
> *full join employees*

The *join* will return all the records in the *workforce table* and the *employees table*.

LEFT JOIN

Left Join
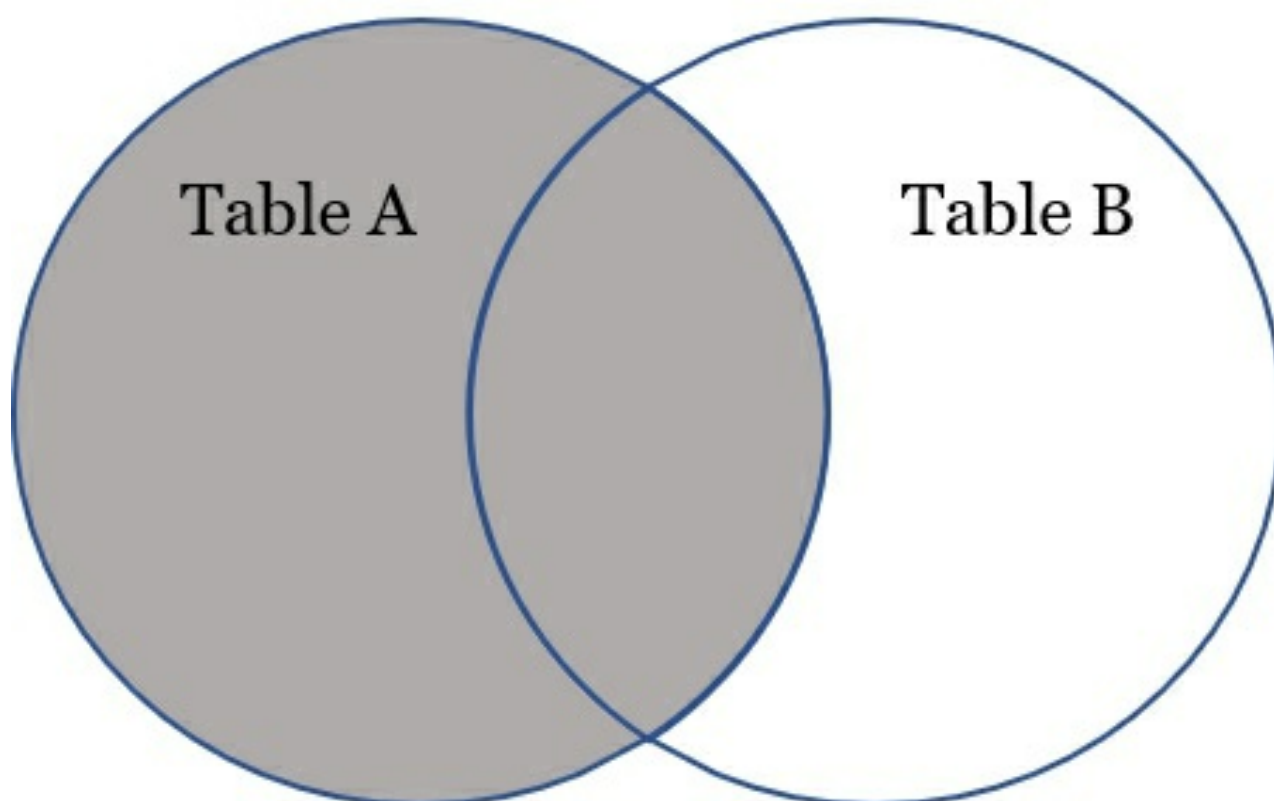
The *left join* will show all the records in *Table A* regardless of whether they match the records in *Table B*.

To use the *left join* to populate the *workforce table* with the *emp_id*, *name*, and *age* fields from the *employees table*, the code should look similar to the following:

> *select wf_id, employees.emp_id, name, age, dept_id, sal_id*
>
> *from workforce*

The *join* should have returned all the records from the *workforce table* and only the matching records from the *employees table.*

RIGHT JOIN
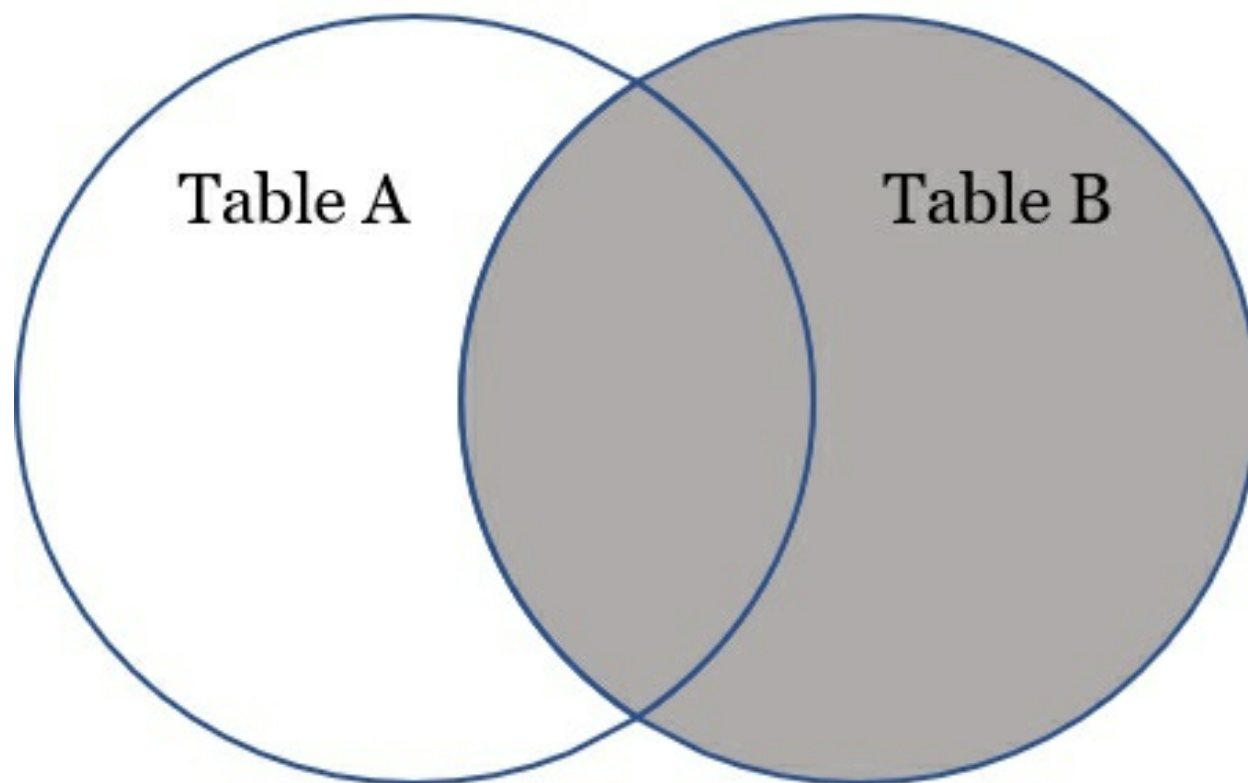
The *right join* will show all the records in *Table B* regardless of whether they match the records in *Table A.*

To use the *right join* to populate the *workforce table* with the *emp_id, name,* and *age* fields from the *employees table,* the code should look similar to the following:

> *select wf_id, employees.emp_id, name, age, dept_id, sal_id*
>
> *from workforce*
>
> *left join employees on employees.emp_id=workforce.emp_id*

The *join* returns all the records from the *employees table* and only the matching records from the *workforce table.*

Joining More Than Two Tables

Consider the *hr database* for instance. It is not only the *employee table* information that is required to make the *workforce table* usable. What about the information from the *departments table* and the *salaries table*?

In the examples above only the *dept_id* and *sal_id* information is reflected in the *workforce table* query. More than two tables can be joined together using the various SQL joins.

To use the *right join,* to populate the *workforce table* with the *emp_id, name,* and *age* fields from the *employees table,* the code should look similar to the following:

> *select wf_id, employees.emp_id, name, age, departments.dept_id, department, salaries.sal_id, jobtitle, joblevel, salary*
>
> *from workforce*
>
> *inner join employees on employees.emp_id=workforce.emp_id*
>
> *inner join departments on departments.dept_id=workforce.dept_id*
>
> *inner join salaries on salaries.sal_id=workforce.sal_id*

Execute all the command lines.

The *join* returns all the records from the matching records from the *employees table, departments table,* and the *salaries table* that match the records listed in the *workforce table.*

Try using each of the different types of joins to join the tables in the *hr database.*

UNION

It is easy to get confused with *join* and *union* as they are both used to join information. The difference between these two types of joins is that *join* is used to filter and combine data from two or more different tables. The *union* clause is used to join the results of two or more queries without returning any duplicate information.

Example:

If certain information was required from the *employees table* such as a criterion for a certain age group, the *select* statement would be similar to:

select age from employees where age >= 30 and age <=40;

Type the query above into a new "Query" line and execute all the command lines.

The "Results Grid" will list all the employees in the *employees table* that are within the set query age parameters.

From the workforce table, information needs to be extracted to find what jobs earn between $25,000 and $35,000. The *select* statement would be similar to:

select salary from salaries where salary >= 21000 and salary <=32000;

Type the query above into a new "Query" line and execute all the command lines.

The "Results Grid" will list all the job titles in the *salaries table* that are within the set query salary parameters.

If you had to create a *union* between the two queries above:

select age from employees where age >= 30 and age <=40;

union

select salary from salaries where salary >= 21000 and salary <=32000;

The above union would combine the two queries with the information specified in both select statements. However, the above example is a basic example of what the *union* clause does.

There are a few rules that apply when using the *union* clause:

- The data types must be the same in each *select* statement.
- The data must follow the same order in each *select statement.*
- There must be the same number of selected columns in each select statement.
- There must be the same number of column expressions in each select statement.
- There does not need to be the same amount of records in each select statement.

Example:

Create a small customer and order database as follows:

create database orders1

Set the *orders1 database* as the default schema in MySQL Workbench.

Create tables for the *orders1 database*.

In a new "Query" line type the following:

```
create table customer (
    cust_id int not null auto_increment,
    custname varchar (55),
    address varchar (55),
    olimit dec (10. 2),
    primary key (cust_id)
    );
```

Execute the script.

In a new "Query" line type the following:

insert into customer (custname, address, olimit) values ('Henery', 'address 1', '2300.34');

insert into customer (custname, address, olimit) values ('Janet', 'address 2', '2000.00');

insert into customer (custname, address, olimit) values ('John', 'address 3', '4500.50');

insert into customer (custname, address, olimit) values ('Jessie', 'address 3', '5200.00');

insert into customer (custname, address, olimit) values ('Daisy', 'address 4', '3500.00');

Execute the script.

In a new "Query" line type the following:

create table orders (

ord_id int not null auto_increment primary key,

cust_id int,

oddate date,

amount decimal (10,3),

ostatus varchar (25),

foreign key (cust_id) references customer (cust_id)

);

Execute the script.

In a new "Query" line type the following:

insert into orders (cust_id, oddate, amount, ostatus) values ('1', '2020-03-12', '100.50','Pending');

insert into orders (cust_id, oddate, amount, ostatus) values ('3', '2020-07-10', '1000.00','Shipped');

insert into orders (cust_id, oddate, amount, ostatus) values ('4', '2020-07-15', '500.50','Complete');

insert into orders (cust_id, oddate, amount, ostatus) values ('4', '2020-09-19', '2300.00','Processing');

Execute the script.

In a new "Query" line type the following query:

select customer.cust_id, custname, address, oddate, amount from customer

left join orders on customer.cust_id=orders.cust_id

Execute the script and take note of the results the query returned.

In a new "Query" line type the following query:

select customer.cust_id, custname, address, oddate, amount from customer

right join orders on customer.cust_id=orders.cust_id

Execute the script and take note of the results the query returned.

Now create a union between the two queries. In a new "Query" line type the following query:

select customer.cust_id, custname, address, oddate, amount from customer
left join orders on customer.cust_id=orders.cust_id
union
select customer.cust_id, custname, address, oddate, amount from customer
right join orders on customer.cust_id=orders.cust_id

Execute the script and take note of the results the query returned. You will notice that there is now no repeated information in the table.

The more advanced you become in SQL, the more you will be able to refine the *union* clause.

Order

The *order* clause is used much like the "Sort By" command in programs such as MS Word, MS Excel, and various databases. The *order* clause is used to sort data in the tables by selected columns in ascending or descending order.

Using the *customer table* in the *order1 database* run the following scripts:

*select * from customer*

order by custname

Execute the script.

You will notice that the table has been sorted alphabetically from a to z (ascending) order.

Using the *customer table* in the *order1 database* run the following scripts:

*select * from customer*

order by custname, olimit

Execute the script.

You can sort data in a table by more than one column.

Try the following script:

*select * from customer*

order by olimit, custname

Execute the script.

You select the first column the data is to be sorted by, by referencing that column name first.

You can sort the table in descending order:

Try the following script:

*select * from customer*

order by custname desc;

Execute the script.

Now the order of the records runs from z to a. The same rules that apply to sort the columns in ascending order (default) apply to sort the columns in descending order.

Group By

The *group by* clause is used to group identical data. For instance, if you look at the *orders table* in the *order1 database,* some customers have placed more than one order.

To get the total amount a customer has spent on orders, the *group by* clause can be used.

Try the following example to give you a better idea of how this clause works.

> *select ord_id, customer.cust_id, custname, sum(amount) from orders*
>
> *join customer on customer.cust_id = orders.cust_id*
>
> *group by cust_id;*

Execute the script.

The results of the above script will show all the customers who have placed orders and total the amount of their orders.

Alias

There are times when you need to shorten the name of a table or column. The following script is a prime example. You can shorten the fields highlighted in bold below to make working with the table a lot simpler.

> *select **customer**.cust_id, **customer**.custname, **orders**.amount*
>
> *join customer on customer.cust_id = orders.cust_id*

Using an alias for the script above would reduce the syntax to the following:

> *select ord_id cust.cust_id, custname, sum(amount)*
>
> *from customer as cust, orders as o*
>
> *where cust.cust_id = o.cust_id;*

CHAPTER 9:

STORED PROCEDURES

As you have been working through the various chapters of this book you have been writing quite a few scripts. You may have noticed some of the scripts have recurring code and some scripts are exactly the same as other scripts that you had written before.

When you are writing quick small scripts as you have been doing in the exercises above, it is not that time-consuming. But what if you had complex queries that performed various tasks on a weekly or daily basis? Some complex scripts can carry on for many lines. When you are writing scripts such as those, it is very easy to miss the odd comma, underscore, brackets, and so on. Once the script is perfect, you are going to want to have a way to store the script so you can use it again and again.

The Advantages of Using Stored Procedures

There are a few benefits to using SQL stored procedures. Some of these benefits are:

- Stored procedures increase the execution speed of the script. Execution is faster because stored procedures are compiled, optimized, parsed, and stored in the database cache.
- Executing a stored procedure is less bandwidth-hungry if you are using a network because the procedure is executed from the local database cache.
- Stored procedures can be used over and over again without having to rewrite complex scripts.
- Stored procedures offer great security and have tighter restrictions and constraint options.

Creating Stored Procedures

Using the following script you used in the *group by* section of the previous chapter, you are going to create a stored procedure.

In a new "Query" line type the following:

select ord_id, customer.cust_id, custname, sum(amount) from orders

    join customer on customer.cust_id = orders.cust_id

    group by cust_id

    order by custname;

If you were using the *orders database* in a real-life situation, the order table would be large and would more than likely grow per day. The above script is one that would most likely be used for an invoicing procedure.

On a new "Query" line create a stored procedure as follows:

```
delimiter $$
use `orders1`$$
create procedure `totalcustorder`()
begin
select ord_id, customer.cust_id, custname, sum(amount) from orders
join customer on customer.cust_id = orders.cust_id
group by cust_id
order by custname;
end;$$

delimiter ;
```

Execute the script.

In the "Navigation Panel" of MySQL, under the "Schemas Tab", refresh the "Schemas".

Expand the *orders database*. Below the database, there are the "Tables", "View", and "Stored Procedures".

Expand the "Stored Procedures". You will see the newly created *totalcustorder* procedure.

Executing Stored Procedures

There are two ways of executing stored procedures.

MySQL "Navigator Panel"

Expand the "Stored Procedures" in the navigations panel.

Hover over the *stored procedure* that is to be executed. Click on the lightning bolt icon that appears upon hover.

Use a Call Function

You can write a small script:

call orders1.totalcustorder();

CHAPTER 10:

VIEWS, INDEX, TRUNCATE, TOP, WILDCARDS, AND TRIGGERS

You can use views as virtual tables that do not hold any data with their contents being defined by a query. The advantage of using a view is that it adds a layer of security to the database by restricting access to certain columns or rows.

A view can also restrict data being viewed to a summary view instead of a detailed one. They can be used to protect the data layer while still allowing viewing access to it.

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, but 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 MySQL, expand the database you want to create the view for.

Right-click on Views.

Select New View.

The View Designer will appear.

Type the following in the script that is pre-written on the screen using the *shop1* database:

> *CREATE VIEW `customerview` AS select cust_id, sname, phone*
>
> *from shop1;*

Execute the script by clicking on the "Apply" button on the bottom right-hand side of the "Query" screen.

Refresh the "Schemas" navigation panel in MySQL.

Expand the *shop1* database.

Expand the *Views* beneath the *shop1* database.

Hover the mouse over the *customerview* you created.

Click on the small table with a lightning bolt icon and the "Result Grid" will bring up the view.

You can order columns for views and create views that pivot tables (see the chapter in this book on how to Pivot tables in MySQL). Creating views in SQL allows you to limit what certain users can and cannot see right down to the data records in a table.

Index

Indexing data searches speeds up the data lookup and access to the tables/records that have been indexed. They work much like the Index or TOC in a book. Instead of flipping through hundreds or thousands of pages looking for information, you can refer to the book's index to find the page you are looking for. This cuts down the time and effort of having to scan through all those pages.

In a database, creating an index will speed up a search using the select query and the where clause. But, you have to be careful with indexes because they may speed up lookup but they can have a negative impact on updating and inserting information into a table. But creating or dropping indexes will not affect the data in tables and is relatively easy to do.

There are different types of indexes that you can create within a database and these are:

Table Indexes

This creates an index for an entire table.

To create an index on a table you will use the *create index* command and the statement will look similar to the following statement:

> *create index testindex on customers;*

This will create an index for the entire *customers* table in the *shop1* database.

Single-Column Indexes

You can create an index for a single column within a database table by typing in a statement such as the one below:

create index testindex2 on customers (sname);

This will create an index for the sname column in the customers table of the *shop1* database.

Composite Index

You can also create an index that will index more than one column within a table. This is called a composite index and the statement to create this index is as follow:

create index testindex3 on customers (sname, fname);

The above statement will index both the sname and the fname columns of the customer table in the shopt1 database.

Implicit Index

The moment you set up a table in a database it will automatically create an index for the new object. Unique constraints and primary keys will automatically be indexed by the SQL engine as an implicit index.

Unique Index

Unique indexes test for the integrity of data and do not allow any duplicate entries to be entered into a table. A unique index can be created by using the following SQL statement:

create unique index testindex4 on products (products);

This will create a unique index on the products table and ensure there are no duplicate product names in the products table of the shop1 database.

Deleting an Index

You can delete an index when it is no longer useful or is taking up to many resources. To do this, you use the *drop index* statement. This can be done by using the following statement:

alter table customers

drop index testindex3;

Indexing Tips

There are times when using an index is necessary and can greatly enhance the performance of a search through millions of database records. But there are also times when indexes should not be used.

Here are a few tips on when to use or not use database indexes:

- Indexes are necessary when searching through large amounts of data.
- Indexes are not necessary when the database tables are small.
- Avoid using indexes on data that is not searched frequently.
- If columns in a table contain NULL values it is not a good idea to use indexes.
- Tables and columns that are used for large batch inputs, updates, and modifications should not be indexed.

Truncate

The truncate command is used to completely delete all the data from a table. This differs from the *drop* table command in that *truncate* only deletes the data and not the table itself.

This is useful when you have obsolete data in a table but want to keep the structure of the table, or you have archived the current data in a table and need to use it for more recent data.

The syntax to truncate data from a table is as follows:

truncate table customers;

You can check that all the records have been removed from the table by running the following:

*select * from customers;*

Top

SQL uses the *top* command to return the top *n* amount of records from a table. For example, say you had the following products with their sales data for the year:

Product	Yearly Sales
Apples	40000

Bananas	22000
Grapes	35000
Pears	48000
Watermelon	55000

You could write a query that will return the top products in the products table. To do this you would use the following statement in SQL:

select top 2 * from products;

The result of the query would return:

Product	Yearly Sales
Apples	40000
Bananas	22000

MySQL does not support the *top* command. Instead, you will need to use the *limit* statement in a similar query to the one below:

select * from products limit 3;

The above statement will return the same result as using the *top* statement in SQL does.

Wildcards

SQL supports the use of wildcards to make selecting certain data easier. For instance, when you add the "*" in the select statement you are using a wild card, as the asterisks tell SQL to select "all" the columns in that particular table without having to type them out individually.

Besides the asterisk, there are two other common wildcards supported by SQL:

The Underscore Operator "_"

The underscore operator is used to mark place holders to create a match in a search for one character example:

select * from customers where fname like _ndy

The above statement will return all the customer names that end with "ndy" and have one character in front of the search criteria, for instance, Andy.

select * from customers where fname like sh___

The above statement will feature all the customers' names that start with "sh" and have three characters at the end of the name, for instance, Shane, Shawn, Shaye, etc.

The Percentage Operator "%"

The percentage operator works a lot like the asterisks but is used to match one or more characters in a search:

select * from customers where fname like %nd%

The above statement will list all the customers' names that have an "nd" in them in any position of the word, such as Sandy, Andy, Mandy, etc. Some databases use the * operator in place of the %.

select * from customers where fname like %am

The above statement will only return names that have "am" as the last two letters such as Pam, Sam, William, etc.

Triggers

Triggers can be used to automatically execute actions within an SQL database during a DDL or DML operation.

An example of a database trigger is setting the database maintenance to run after a set period of time. Another example would be setting up notifications when certain conditions are met. Retail stores use them to run future promotions or to give certain discounts for the one-hundredth user, and so on.

SQL offers two types of triggers which are:

AFTER Trigger

The AFTER trigger will be executed after an event or action has been performed and meets the trigger's conditions (for instance, when a certain database threshold has been exceeded, trigger an event).

INSTEAD OF Trigger

This trigger will perform an action that is different from the actual event.

Trigger Syntax

You can create a trigger in MySQL by expanding the Schema navigation view of the MySQL database. For this example, you can use the *shop1* database, so make sure it is set as the default schema.

Type the following in the script that is pre-written on the screen using the *shop1* database:

```
create table totalaccount (
id int auto_increment not null,
tamount decimal (10,2),
Primary key (id)
);
create trigger ins_sum before insert on totalaccount
for each row set @sum = @sum + new.tamount;
```

Execute the script.

Refresh the "Schemas" navigation panel in MySQL.

Expand the *shop1* database.

Expand the *Triggers* beneath the *shop1* database.

You will now see the trigger you created.

This will sum the amount of the tamount column when data is entered into it.

CHAPTER 11:

PIVOTING TABLES IN MYSQL

Records in an SQL database are presented in rows. To *pivot* data in SQL means to rotate the data records and convert them into columns. This is handy for use in data analysis and reporting, especially for visualizing trends and multidimensional reports.

Create a Products Sales Database

Open a new MySQL session with a new "Query" tab and type the following script:

> *create database sales*

Set the *sales database* as the default schema in MySQL Workbench.

Create tables for the sales *database*.

In a new "Query" line type the following:

> *create table prodsales (*
>
> > *id int not null auto_increment,*
> >
> > *product varchar (45),*
> >
> > *syear year,*
> >
> > *sales decimal (10, 2),*
> >
> > *primary key (id)*
> >
> > *);*

Execute the script.

In a new "Query" line type the following:

> *insert into prodsales (product, syear, sales) values ('Razor Blades', '2015', '23000');*
>
> *insert into prodsales (product, syear, sales) values ('Razor Blades','2015', '25000');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2015', '23800');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2015', '20800');*
>
> *insert into prodsales (product, syear, sales) values ('Razor Blades','2016', '30000');*
>
> *insert into prodsales (product, syear, sales) values ('Razor Blades','2016', '32000');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2016', '35000');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2016', '37000');*
>
> *insert into prodsales (product, syear, sales) values ('Razor Blades', '2017', '43000');*
>
> *insert into prodsales (product, syear, sales) values ('Razor Blades', '2017', '45000');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2017', '46000');*
>
> *insert into prodsales (product, syear, sales) values ('Razors', '2017', '48000');*

Execute the script.

Query the *prodsales table:*

> *select * from prodsales*

Execute the script.

The *prodsales table* currently looks like the table below.

Prodsales SQL Table		
product	**syear**	**sales**

Razor Blades	2015	23000.00
Razor Blades	2015	25000.00
Razors	2015	23800.00
Razors	2015	20800.00
Razor Blades	2016	30000.00
Razor Blades	2016	32000.00
Razors	2016	35000.00
Razors	2016	37000.00
Razor Blades	2017	43000.00
Razor Blades	2017	45000.00
Razors	2017	46000.00
Razors	2017	48000.00

How to PIVOT Data in MySQL

MySQL does not support *pivot*, nor do a few other RDBMS databases such as Microsoft SQL Server and SQLite. However, like MySQL, some functions can simulate the functionality of *pivot.*

In MySQL, the *case function* can be used to *pivot* tables. The *case function* is a conditional statement. It will check to see if a condition is met before returning a result. It uses the same type of format as the "if … then … else … and is usually followed by an aggregate function or two such as *when* and *then*.

A simple explanatory example of using the *case* statement would be to consider a simple customer address log where you want to group customers by the towns they live in.

This simple script would look something like:

> *select residents, town, state, zip*
> *from states*
> *order by (**CASE WHEN** town is NULL **THEN** zip **ELES** state **END**);*

The above script selects residents and sorts them by either their town, zip, or state.

In the next section, you are going to use the *case* function to pivot data using the *prodsales table* from the *sales* database you have created.

If you look at the data collected in the above prodsales table representation, you can see there are numerous ways you could manipulate and report on that data. One such report would be to view the number of sales per product per year as per the table below.

Pivoted Prodsales SQL Table			
product	**2015**	**2016**	**2017**
Razor Blades	47000.00	62000.00	88000.00
Razors	44600.00	72000.00	94000.00

You could extract the information with the following query:

> *Select product, sum(sales), syear*
> *from prodsales*
> *group by syear;*

But this would still return the output view in a column view.

Select a new "Query" tab and start the following script to pivot the data to read as shown in the table above.

> *SELECT*

product,

sum(case when syear = '2015' then sales else 0 end) as '2015',

sum(case when syear = '2016' then sales else 0 end) as '2016',

sum(case when syear = '2017' then sales else 0 end) as '2017'

FROM prodsales

GROUP BY product

Execute the script and the data will be presented with only the two products, each showing the sales per year.

As good practice and to keep your SQL skills fresh, save the pivot table script as a procedure that you can reference and use again.

CHAPTER 12:

CLONE TABLES

Clone tables are identical tables that you can create to perform SQL tasks. *Clone tables* have exactly the same format and content as the original table they are cloned from. Using a *clone table,* you can ensure that the new table has the exact same indexes, constraints, etc. that the original table has.

Why Use a Clone Table?

- To create identical tables to work with for updates, fixes, etc. instead of working on live tables and data.

- To act as practice tables for beginners, so that the tables in the live databases are safe and protected.

- To feature new interfaces for new users.

Using Clone Table

Using the *sales database*, you will clone the *prodsales* table created in the previous chapter and populate the *cloned table* with the data of the original table.

Open a new "Query" tab in MySQL. You will use the same script you used to create the original *prodsales table.* Type the following into a new "Query" line:

create table prodsales_clone (

    id int not null auto_increment,

    product varchar (45),

    smonth varchar (25)

    syear year,

    sales decimal (10, 2),

    primary key (id)

    );

Execute the script. You now have two identical *prodsales tables* in the *sales database* to work with.

Populate the *prodsales_clone* table with the exact same data as the original table by typing the following into a new "Query" line in MySQL:

    insert into prodsales_clone (product, smonth, syear, sales)

    select product, smonth, syear, sales

    from prodsales;

Execute the script.

Query the *prodsales_clone table:*

    *select * from sales.prodsales_clone;*

Execute the script and the "Result Grid" will show the cloned table populated with the exact same data as the original *prodsales* table.

CHAPTER 13:

SECURITY

Databases hold all kinds of sensitive data including employee records, company blueprints, and so on. Due to the nature of company data, systems need high-end, functional, and reliable security which include controlled access to systems. Many companies today use Microsoft's Active Directory to manage users and sort them into access profiles using processes such as group policies.

Group policies are used to assign employees various permissions based on their job title, security level access, etc. by using group level, directory level, and even file-level permissions for more individualized permissions.

Active Directory is compatible with SQL but only for access control and does not provide internal security for authorization. These types of security servers are provided by the applications that run SQL, such as MySQL, SMSS, Microsoft Sequel server, and so on.

Components of Database Security

There are four main components of database security and these are:

Authentication

Authentication pertains to validating a user's permission to access various systems resources, directories, files, and applications.

The most common method of authentication is simply a username and password. A username and password will verify the credentials of a person trying to access the system.

Single sign-on is available which uses secure certificate authentication. This means that the user can only sign onto the system on a certain device that has been set up with the necessary security certificates.

Encryption

Most 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 symbols which are hard to interpret if not correctly authenticated.

Microsoft SQL Server uses RSA data encryption. RSA is an algorithm that uses key management and a layered hierarchical structure to encrypt data. It makes use of SSL certificate key infrastructure which uses both Public and Private Keys.

Authorization

The Authorization process is what determines a systems user's authentication type, in other words, what they can and cannot access.

Once a user has logged onto the system their access policies will prevent them from being able to access the information they do not have permissions to access. If the client is dealing with information, authorization can be restricted right down to the file level. Rights can be limited to being able to modify, update, delete, create, etc.

Access Control

Change tracking is used to maintain and keep a log of a user's use of the system. SQL user change tracking also makes it possible to track the activities of all authorized users.

Tracking can track the day, time, and even what was accessed per workstation or user ID.

Tracking changes helps to prevent a user with top-level access from causing damage to the system.

Three-Class Security Model

SQL uses a security model consisting of three classes. All three SQL security classes interact with each and have the same basic security functions, namely:

- Principals — Principals can only access certain specified objects in the database.
- Securables — Securables are system regulated resources that run within the database.
- Permissions — Permission refers to the right to view, edit, or delete securables. This access is pre-defined.

Although this security model is a Microsoft SQL Server-based idea, other SQL Management systems such as

MySQL, DB2, and MongoDB have a similar security structure.

Security modeling is a lot more widespread across the IT industry. It incorporates networks, cloud computing, and personal computers or devices right down to mobile phones.

Security covers a wide area of SQL and any system that houses sensitive data. It is not a subject that can be covered in a chapter, but rather an entire course on its own.

Schemas

In relation to security, the schema defines ownership of resources in a database system. Schemas identify principles based on the client's level of authorization.

According to Microsoft, a schema is "a group of database objects under the ownership of a single individual and together they create what is known as a 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.

Principals are used for either group, single login, or a single user.

Multiple users sharing one role are grouped using group policies, and all can cooperatively own a schema or many schemas.

It is possible to share or transfer a schema between principals without renaming the schema.

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 offer a database system another layer of security. This layer identifies access based on a user's systems credentials such as responsibilities and access levels.

Although there are many different kinds of roles, SQL comes with both fixed database and server roles, each of which provides implicit permissions.

SQL offers the option to customize roles such as user-defined server roles, application roles, and user-defined database roles.

A few Fixed Server Roles are:

- **sysadmin** —The role of the sysadmin user holds the highest form of access and can perform any SQL database or SQL Server action.
- **blkadmin** — This role permits users to run the *bulk insert statement.*
- **dcreator** — This role permits users to create a new database within an SQL Server instance.
- **diskadmin** — Users allocated to this role manage SQL Server disk files.
- **processadmin** — This role allows users to terminate SQL Server processes.
- **public** — This role is used for user authentication to the SQL database.
- **securityadmin** —The securityadmin role grants users permission to assign or revoke user rights to the database in an SQL Server instance.
- **serveradmin** — The serveradmin role can start and stop the SQL Server as well as assign various configuration rights.
- **setupadmin** – Users assigned this role can use T-SQL statements to remove or add new linked servers and they must have sysadmin rights to perform this role.

Roles play an important role in securing a database and database engine. SQL security functions are mostly built on top of server roles as the authentication upon login determines the access permissions principals have to various schemas.

Data encryption protects information from potential hackers and other unauthorized access or login attempts. It also plays a vital role in protecting company data from internal threats or for information being accessed by users that should not be accessing it.

Roles govern exactly what users can access or what tasks they can perform within a database or Database server instance. SQL security practices are to allocate a user the barest minimum rights they need to effectively perform their duties.

This stops a user from having rights to the wrong information and access to valuable resources that they are either not trained to use or may be holding an unnecessary license for.

Sysadmins are considered to be database or schema owners.

Members of an organization's IT department are usually tasked or assigned the role of database administrators. There are different levels and departments of both database administrators and database owners. The level depends on the assigned task within the organization.

Customized Server Roles

Server roles can also be created and customized to suit a given purpose within an organization.

Custom server roles can be created by:

- Selecting Security objects in the SQL server management program.
- Select "New"
- Select "Role"
- The name of the role can be anything, but it is usually given a name pertaining to the role and function of the role.
- The "Explicit Permissions" needs to be set. This is usually a checkmark box.
- The permissions are then set and the role is created.

Logins

The security hierarchy in an SQL database starts with the systems administrator or database owner that originally creates the database instance.

The SQL security model is based on the principles as covered earlier in this chapter:

- Principals – Have access to deny, granted, or revoke access at all levels.
- Securable – These are database/database engine objects that can be manipulated and as such need access level rights to be set.
- Permissions – Permissions are set to allow various access levels to data, database tables, and securables.

Groups

Groups can be set up to contain either various roles, access permission rights to specific files, databases, tables, or database functions.

Groups make granting various access and roles to a specific group of people a lot easier. For instance, a manager of an accounting department may require access to the same roles as their teams but may also need to have access to resources. These can all be set in a group, making it easier to manage all the users' access and for granting another user the same access requirements more efficiently.

It is usually the function of the database owner or administrator to create and manage groups.

Mixed Mode Authentication

Mixed mode authentication gives an extra layer of security to authentication of a user login. Mixed mode authentication means logging into the database has two-tier authentication and passwords are stored on the database.

The "Systems Administrator" or "SA" user is enabled when mixed mode authentication is set. The SA has the highest level of access rights on the database and this user is barely used or checked; it can become a security risk. The SA is usually the first ID targeted by hackers.

SQL best practice is to rename the SA and ensure it has a strong password that is changed regularly.

Find the name of the SA or root user by typing in the following:

select user, host from mysql.user;

This will return a table similar to the following in the "Result Grid".

User	Host
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

The SA or root user can be renamed in MySQL by typing the following into a new "Query" line:

rename user 'root'@'localhost' to 'newroot'@'localhost'

flush privileges;

mysql -u newroot -p

To change the password of the SA or root user:

*update mysql.user SET Password=PASSWORD('**thisismynewpassword**') where user='root',*

flush privileges

After the root password has been changed you will need to completely log out and shut MySQL down. Then log in again with the new password.

Database Roles

Permissions managed at the server level are called server roles. Permissions managed at the database are called database roles. Database roles can assign common permissions to principles by grouping them.

The most common database roles are:

- db_accessadmin — Manages user access by being able to add and remove users a database instance.
- db_backupoperator — This user role is used to backup the database.
- db_datareader — This user role is permitted to use the *select* statement in permitted database instances.
- db_datawriter — This user role is permitted to execute Data Manipulation Language (DML) statements on permitted database and table instances.
- db_denydatareader — This user role is used to deny permission to run the *select* statement throughout all database instances.
- db_denydatawriter — This user role is used to deny permissions to perform DML statements in database instances.
- db_ddladmin — This user role can *drop, alter,* and create objects in a database.
- db_owner — This is the user role that has full access to the entire database.
- db_securityadmin – This user role is used to set permission to the database securables.

Customized Database Roles

Like server roles, database roles can also be created and customized as and when required to suit a given purpose within an organization.

Custom database roles can be created by:

- Selecting Security objects in the SQL server management program.
- Select "New"
- Select "Database Role"
- The name of the role can be anything but should be something pertaining to the role and function of the role.
- The "Explicit Permissions" needs to be set; this is usually a checkmark box.
- The permissions are then set and the role is created.

Encryption

Encryption can be used to enhance database security. Encryption is just another layer of security and does not replace the main security control method for an SQL database.

Encryption offers the following benefits:

- It can be easy to implement.
- It does not have an impact on any applications or scripting.
- It offers protection for system backups.
- Allows for more useful and efficient audit trails.

While encryption enhances the system's security, it can have a few negative impacts on the database system as well. Some of the negative impacts of implementing encryption are:

- Encryption can have a negative impact on the system's performance.
- There is no encryption protection at the application level which leaves data vulnerable to inside attacks.
- There are usually large amounts of encryption keys to manage that can become costly and add to an overburdened database administration team.
- Encryption has been known to be disruptive to DB functions such as indexing, merge, and search.

Master Keys

SQL Server root encryption uses service *master keys*. When a new SQL server instance is created a new *master key* is automatically generated. A *master key* is to be used to encrypt database master keys and is linked to various server passwords.

There can only be one *master key* per server instance and the key gets stored in the *master database.*

You can generate a new *master key* should you need. The script to generate a new SQL master key is as follows:

alter service master key regenerate

SQL best practice is to create a backup of the *master key* and to create a password for it. To backup the *master key* and set a password the script would look similar to 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 have a strong password that is changed regularly. To create a database master key, you use the following script:

create master key encryption by password = 'Password1'

To back up the database *master key* you can use the following script:

backup master key to file = 'c:\ sqlServerKeys \serviceMasterKey'

encryption by password = 'Password1';

If you need to use the backup *master key*, you can run a restore procedure such as the following one:

restore master key

from file = 'c:\ sqlserverkeys \servicemasterkey'

decryption by password = 'Password1'

encryption by password = 'Password1'

Transparent Data Encryption (TDE)

If you store the data by encryption key in the boot record of a database, you can encrypt aspects of the database such as log files and data pages. To do this, pages need to be encrypted before they are written to disk, then before they are written to memory they get decrypted. This is one of the most efficient ways of encryption as it does not put unnecessary strain on the system or cause bloat.

Because the encryption method is transparent across all systems, this encryption method does not require extra coding.

Once a database master key is created a security certificate is also created, which can only be encrypted by the master key. A database is still accessible once TDE has been implemented. But some statements are affected and will **not** be accessible. These are:

- Drop a file
- Drop a filegroup
- Drop a database
- Detach a database
- Take a database offline
- Set a database as read-only

The following script can be used to create a service master and create a server certificate using the master key.

In a new "Query" tab, type the following on a new "Query" line:

```
use master
create master key encryption by password = 'Password1'
create certificate TDECertificate with subject = 'Certificate For TDE'
```

This key can be used to encrypt the database it was created in.

You can also use the *database encryption wizard* to create a TDE certificate.

CHAPTER 14:

SQL INJECTIONS

Hackers are the cat burglars of the Internet, and they are always on the lookout for new targets. They find vulnerabilities in systems and exploit those vulnerabilities to gain access to a system. Once in a system, they can destroy data, steal data, and so on as they have access to credit card information, bank details, and other sensitive data.

Hackers access vulnerabilities in a SQL database by using a method called "SQL Injection". This is the most widely used method for unauthorized users to gain access to an SQL system. It is called a code injection technique because it injects malicious code through a web page into SQL statements.

A user can inject this malicious code into the system at a login screen. The user will use an SQL statement in place of a username or userid to login with. This statement passes into the system and is run through the database, unknowingly unleashing the code.

The statement below is a common authentication statement that allows users to login to a system:

select * from users where username='username' and password='password'

There is nothing wrong with the above code and it will perform the task and return the desired results. For instance, the login screen would give you the following details to fill in:

*Enter Username: **Mike***

*Enter Password: **Rainyday***

Once the user enters the above details the query run on the system looks like this:

select * from users where username='Mike' and password='Rainyday'

This simple code will check for the user name Mike. If it finds the username SQL presumes the login is successful; if the username is not found the login is unsuccessful. Here the hacker can subvert the application logic by removing the password check after the *where* clause simply by using the SQL comments symbol "--".

The unauthorized user could use the following to gain full access to the SQL database:

*Enter Username: **root --***

Enter Password:

The "--" in SQL is a line comment statement and when it is entered everything after "--" is ignored. Simply by adding a common administrator name and then "--" after the name, SQL was told to ignore the password and allow access. Most SQL databases have an administrator, SA, or root user with full administrative rights to the database. These are the first userids a hacker will exploit.

Another common way of gaining unauthorized access is using the concept of 1=1 is always true. If you use the same username and password script you used above and entered the following user credentials:

*Enter Username: **'jjj or' '1=1'***

*Enter Password: **'jjj or' '1=1'***

The SQL statement would run as:

select * from users where username='' jjj or '' ''1=1'' and password='' jjj or '' ''1=1''

The above SQL statement reflects as a valid statement in SQL as x=x is always true. Thus the above username and password will gain the unauthorized access. This access will allow them to view all the users and their passwords in the SQL user table, which in turn allows the hacker access to valid users and all their details along with it.

Due to the nature and size of data out there today, the above attack can also render the database inaccessible to its valid users with a denial of service (DoS).

The above scenario is an attack on an SQL database table for a *select* query. Imagine what an unauthorized user can do with an *update* or *delete* query. They could inject all kinds of data into the existing data and even destroy all the records in a table.

Preventing an SQL Injection

PHP and Perl offer a function of "escaped characters" which enable the use of special characters. MySQL uses a PHP extension to ensure SQL scripts can handle the MySQL version of the escaped characters by providing the "mysql_real_escape_string()".

This string will get the data being entered and then "escape" the character to ensure that SQL sees the data being

entered as text and not as code. Using this string is outside the scope of this course and is for more advanced SQL users. Here's an example of how the code would look:

if (get_magic_quotes_gps()) {$userid = stripslases($userid);}

$userid = mysql_real_escape_string($userid);

*mysql_query(Select * from customers where name='{$useried}' ");*

Hacking Scenario

If you want to find vulnerable websites, Google is one of the best hacking tools in the world to do so. With the use of the *inurl:* command it is quite simple to find vulnerable websites.

Try it. You can copy and paste one of the commands below into the Google website browsers search bar, press enter, and you will get a list of websites:

inurl:index.php?id=
inurl:shop.php?id=
inurl:article.php?id=
inurl:pageid=

You can go through each of them and find their vulnerabilities.

CHAPTER 15:

FINE-TUNING

Investing time and effort into learning all there is to know about a database and how it works is the best way to learn about the database and the data it houses. This is important because it adds awareness to help you fine-tune and keep the database functional, effective, and useful.

By closely examining the database you will be able to understand various data trends, find vulnerabilities, and keep the database current. To confidently and effectively manage and administer an SQL database, you need to ensure you are equipped with the right credentials and knowledge to do so.

Helpful tips for learning the ins and outs of the database:

- Work with the "Third Normal Form" (3NF) design when working with an SQL database. The 3NF is a database schema that is used to design relational databases. To avoid anomalies of data, any data duplication, and ensure the referential integrity of the data, the 3NF approach uses normalizing principles.

- Wildcards should only be used when necessary. For example, using the * will load all the information from the chosen fields. In a small database, this is not that significant. In a large database, this could mean hours of loading time and a screen full of data to sift through.

- Only *index* tables that are frequently used. Indexes take up both memory and disk space.

- Large tables that are accessed frequently should be indexed to cut down on complex large search time. If there are no indexes a full index is run when a table is referenced. For smaller indexes, create table indexes for specific rows.

- Be careful when using equality operators, especially when you are using real numbers, times, and dates. It is likely there will be subtle differences that are not that obvious to spot, and this can cause errors. The equality operators make it nearly impossible to get an exact data match.

- Use pattern matching sparingly.

- Make sure tables are optimized for peak performance and the structure makes sense.

- Avoid the OR operator, unless absolutely necessary, as it will not only slow down searches but also add to database security vulnerabilities. Try not to use it in extra-large databases as it adds too many layers to a comparison query.

- Run regular backups. Backups should be kept at a safe site off location and in the cloud if this facility is available.

- Run regular maintenance checks on the database and ensure that the system has enough room for the database to grow. If you run out of space you run the risk of a database crash which could cause loss of data, data corruption, and application downtime.

- Defragment the database at least once a week to ensure data integrity and that the database runs smoothly.

SQL Tuning Tools

There are a few SQL optimization tools that can be used to help fine-tune SQL and they are broken into built-in tools and application fine-tuning tools.

SQL Built-In Fine Tuning Tools include:

- **TKProf** — This tool is used to measure database performance over a specified period of time. This time is based on how long an SQL statement takes to process.

- **EXPLAIN PLAN** — This tool is used to track the path of a statement to verify the statement's credibility and efficiency.

- **SQL*Plus Command** — This command is used to measure the elapsed time between SQL search passes.

Some of the most popular application tools are:

- SQL Query Tuner for SQL Diagnostic Manager

- SQL Server Management Studio
- EverSQL
- SolarWinds Database Performance Analyzer

CHAPTER 16:

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.

Download the latest version of SSMS from the following link:

> https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

Once the SSMA-Setup_ENU.exe has downloaded, double click the file in the Downloads folder.

Keep the default settings and let it install.

Starting the Database Engine Services

Login

Ensure that the SSMS instance is running on your system by:

- Using the Windows key + R
- Type services.msc in the Run command box.
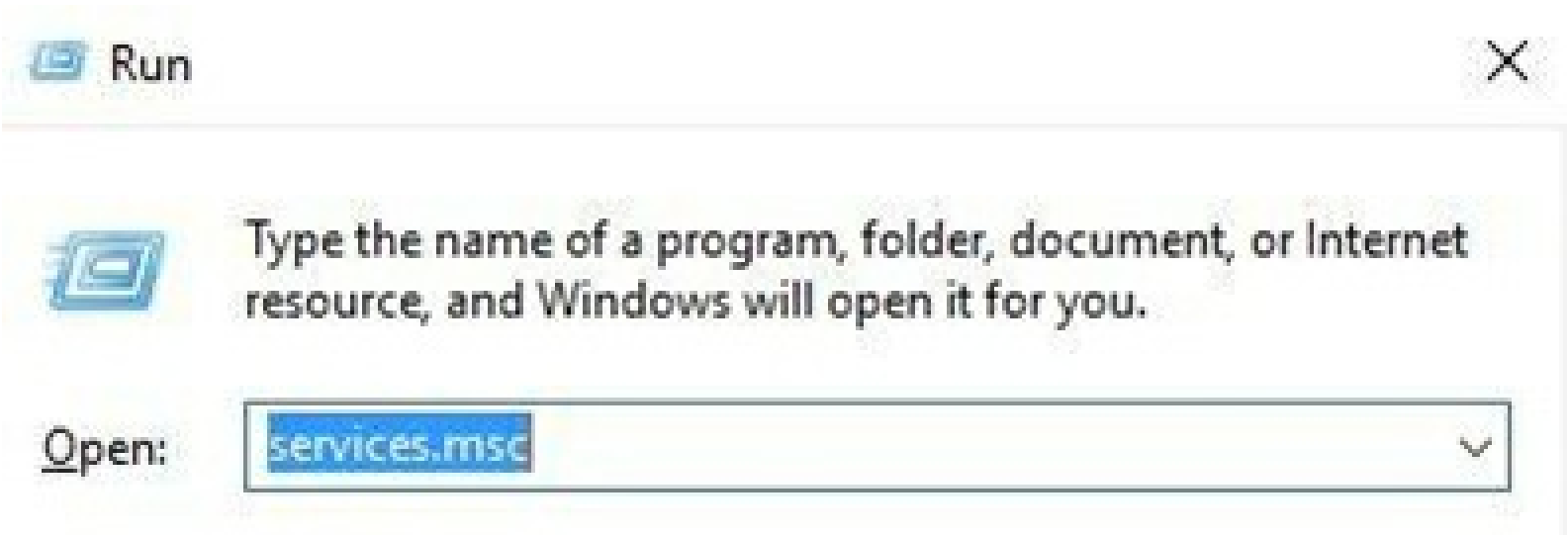
- This will bring up a list of services currently running on the device.
- Scroll through the services to find SQL Server.
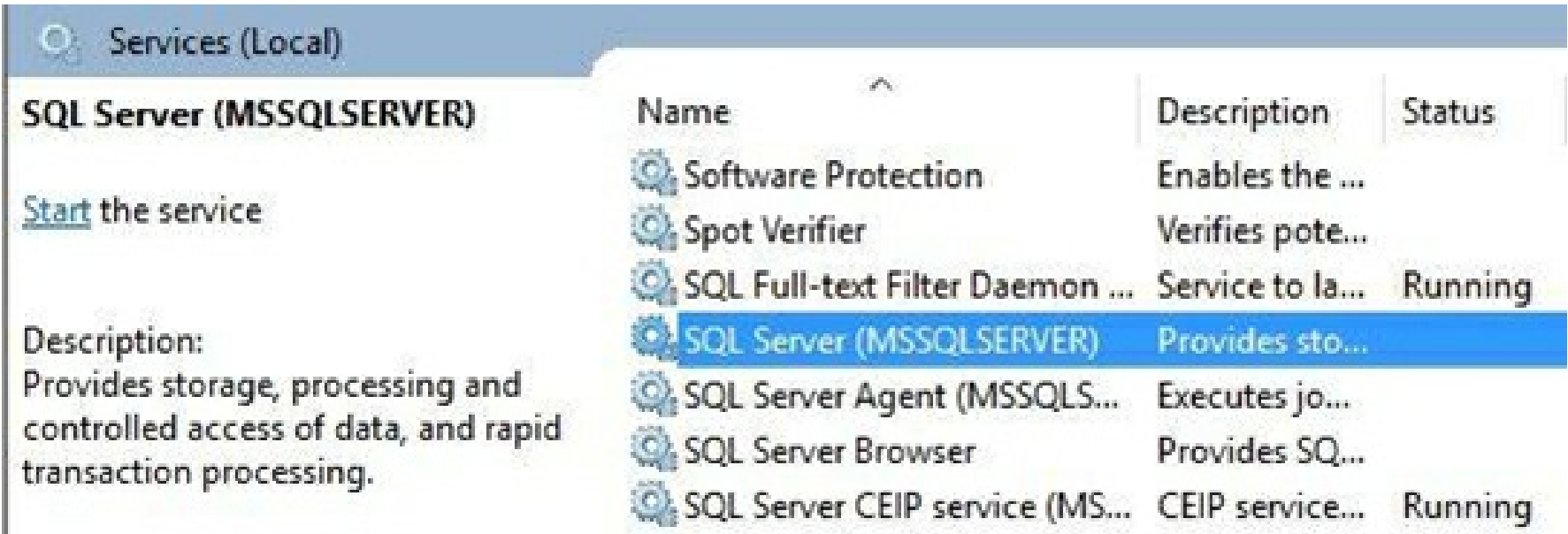
- When first installed, this service is not automatically started by default.
- Start the service if it is not currently running
- This service can consume a lot of computer resources. You can either leave it as a manual start or set it to automatically start whenever the computer is started.
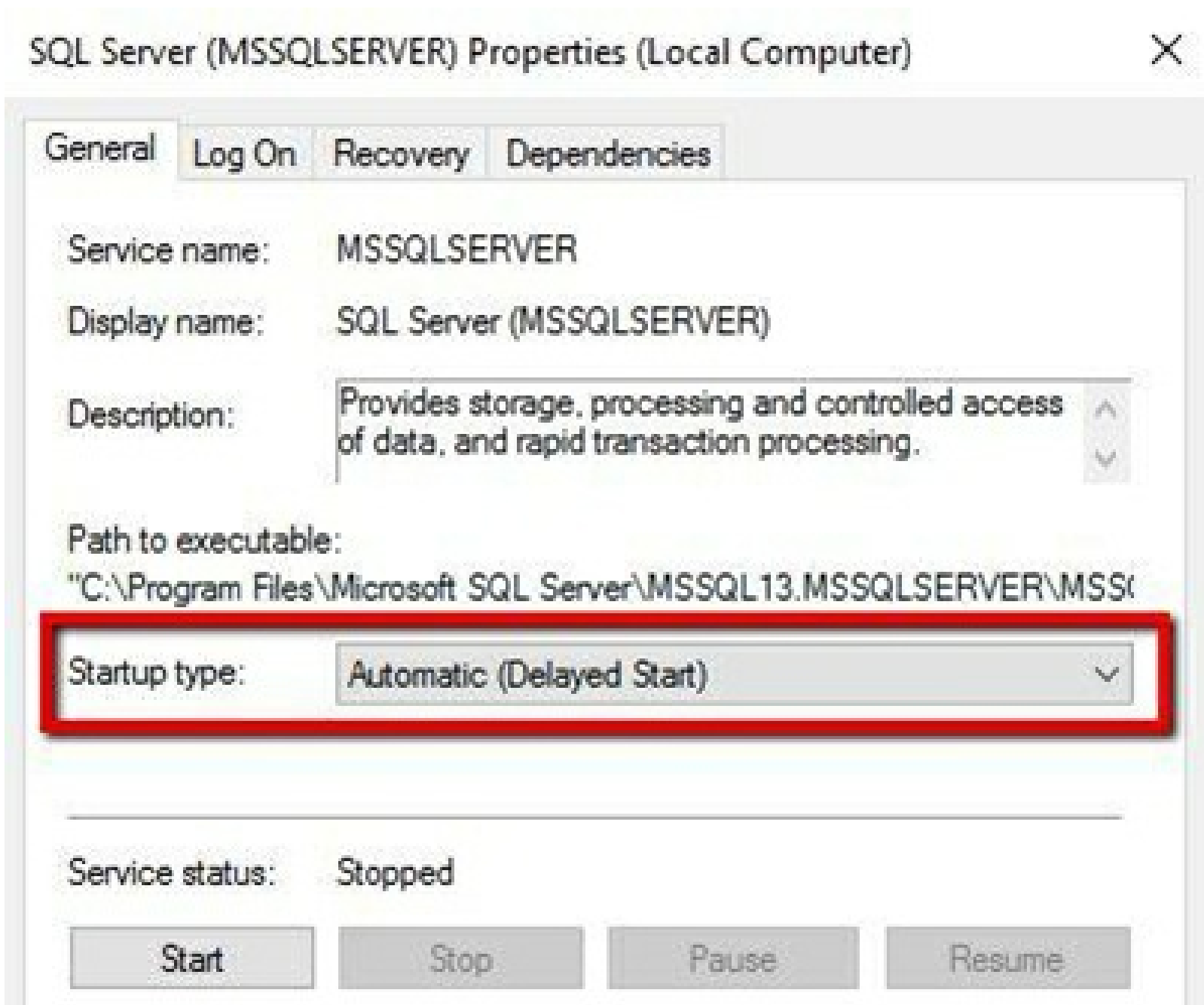
Connect to SQL Server with SSMS

- From the program, the menu finds SQL Server Management Studio.
- The following needs to be configured as follows:
    - Server Type: Database Engine
    - Server Name: (The name of your computer)
    - Authentication: Windows Authentication

You may not need to enter a password as the server will authenticate through the windows instance running on the device.

Click Connect to connect to the database engine.

The Basics and Features of SSMS

Managing Connections

Managing connections to servers within SSMS can be done in several ways.

The connection the red arrow points to at the top allows you access to manage the current SSMS server.

The connection the red arrow points to at the bottom gives you access to control connections to many database SSMS servers.

Choosing Your Database

You can choose the database to manage depending on the server instance in SSMS.

You do so by choosing a database from the list of available databases listed in the drop-down menu.

New Query Window

You will need to open a new query to run queries against the current session.

Use the "New Query" button for a new query. Alternatively, you can use the shortcut key "Ctrl + N".

Executing Statements

SSMS offers several easy to execute statements and they work much the same way as they do in MySQL.

- Highlight certain batches of code and execute the run command.
- If you do not select any code to execute and use the execute code command, all the code scripted on the open "Query" tab will be executed.

IntelliSense

Intellisense is like predictive text on a mobile phone. Instead of having to type out statement names, tables, or field names, it will list what it thinks you want to use. As soon as you type the first few letters it will give you a list of

options to choose from. This is handy as it cuts down on coding errors.

What is great about SSMS is that you can turn this option on and off as you need.

If for some reason, IntelliSense doesn't work as well as it should, or it stops working completely, it may mean you need to clean out the cache. To do this follow these directions:

Got to the "Edit" menu option

Choose "IntelliSense"

Choose Refresh Local Cache.

You can also use the keyboard shortcut Ctrl + Shift + R to refresh the IntelliSense cache.

Results Presentation

The way that the result set is presented can be changed to your preference from the provided options.

The most commonly used preferences are:

- Results to Grid
- Results to Text
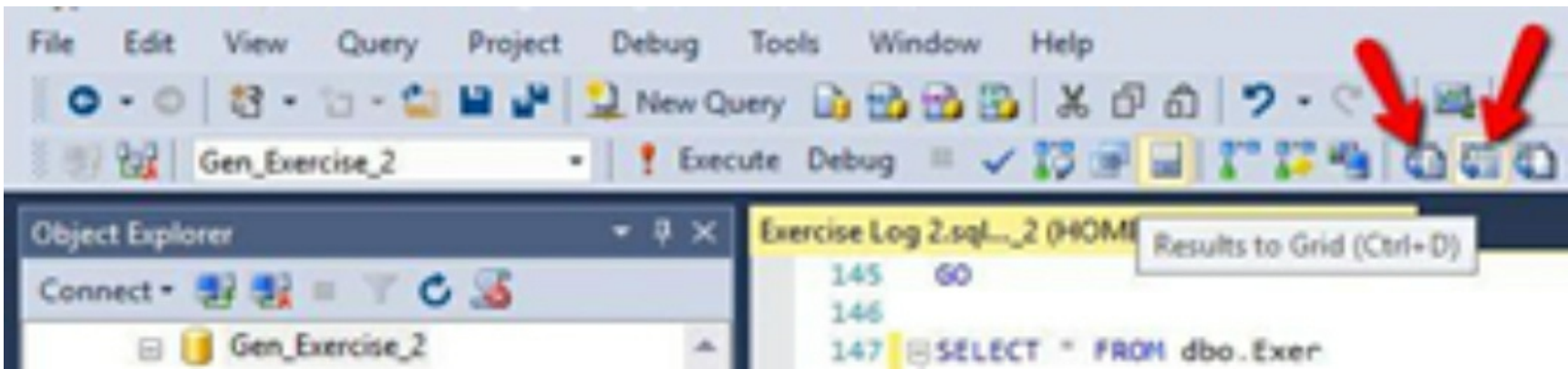
Keyboard shortcuts to change the presentation of the results are:

- Text results — Ctrl + T
- Grid option — Ctrl + D

Below is an example of the grid option:

Below is an example of the text option:

Ctrl + R toggles between show/hide the result set.

Object Explorer

This is one of the most used tools in SSMS.

The object explorer interface expands and contracts the visual presentation of the navigation panel to view databases, tables, views, users, stored procedures, and more.

Databases

The database view can be expanded to see all the objects beneath each database in a server instance. This will take you right down to the table view.

SQL Server system database summary:

- **Master** — Contains the system/server configurations. It is good practice to do regular backups of this object.
- **Model** – This is the template that is used whenever a new database is created.
- **Msdb** – This object holds the configuration for the SQL server and is used to manage the server configuration. It is used for scheduled jobs and various automated tasks.
- **Tempdb** –For the storage of temporary data files, running queries, and stored procedures. The information that exists and is used in this database is temporary and will be erased once the session ends.

There are also the ReportServer and ReportServerTempDB. These two objects represent the Reporting Services that are installed when SSMS is installed. These two services are used for:

- **ReportServer** – The main reporting server that houses custom reports, and is used to schedule jobs and notifications.
- **ReportServerTempDB** – This is a temporary reporting service that holds data in its caches until the session ends, at which time all the data is erased.

CHAPTER 17:

DATABASE ADMINISTRATION

Databases are not only about getting set up and working. They also need to be managed, maintained, modified, fine-tuned and kept updated. This means regular integrity, performance, resilience, and various other checks.

Organizations that run and maintain an in-house database usually have a designated person or team that performs these tasks. They are a database administration (DBAT) team or database administrator (DBA).

A DBA will perform a number of different database tasks which can include the following tasks:

- **Database Integrity Checks**

  Database integrity checks ensure the data structures are consistent and accurate.

- **Index Reorganization**

  Updating of database tables can start to make data unravel or become messy, especially if there is indexing involved. Regular reorganizing of database indexes needs to happen to ensure the database does not become bloated, full, or lag.

- **Rebuild Index**

  If an index is not functioning correctly it will need to be dropped and rebuilt.

- **Database Backup**

  Backups are one of the most important parts of any system. They should be done each night. There are a few different types of backups:

  - **Differential** — This will check for any changes in the data and back up those changes.
  - **Incremental** — This will look for changes and only back up changes that have taken place since the last backup.
  - **Full Backup** — This will backup the entire database regardless of whether or not there have been changes made.

- **Check Database Statistics**

  This is to keep the newest queries being run from getting mixed up with any old queries. All old outdated queries should be deleted or archived to protect the integrity of the data.

- **Data and Log File**

  Data and log files should be kept in order, with older logs that are not used either archived or deleted. The file sizes of the logs also need to be tracked as they can grow to quite a large size that takes up valuable space. Larger log files can also become hard to read and decipher. It is important to maintain smaller log files.

- **Defragmenting**

  The fragmentation of the database should stay consistently below 30%. If it is higher the database will need to be defragmented. It is good practice to check on this at least two to three times a month for very large databases.

Maintenance Plan

SQL Server Agent allows for the setting up of an automated maintenance plan. When setting up a maintenance plan, you need to get the maintenance schedule right to optimize the database and run it during off-peak usage times. This ensures that the CPU is not being over-utilized. It also does not slow it down which would impact any work being done on the database or database applications.

Setting up a Maintenance Plan in SQL Server

The first set up establishing an SQL server plan is to identify the SQL server instances advance options. To do this you can run the following script:

```
sp_configure 'show advanced options', 1
go
reconfigure
```

```
go
sp_configure 'agent xps', 1
go
reconfigure
go
```

Once the advanced options are displayed:

- Left-click the + icon to the left of the SSMS Management screen.
- Left-click Maintenance Plans.
- Right-click Maintenance Plans
- Choose New Maintenance Plan Wizard
    - Enter the name of the maintenance plan
    - Enter an appropriate description so anyone can use it
    - From this point, you can
        - Run a specific task
        - Run all tasks
- Choose Single Schedule then click the Next button.
- Design your maintenance plan around one of the following options available within SSMS:
    - Checking your Database Integrity
    - Shrinking the Database
    - Reorganizing Index
    - Rebuilding the Index
    - Updating the Statistics
    - Clean up History
    - Executing SQL Server Agent Job
    - Backup – full, differential, or transaction log
    - Maintenance Cleanup Task
- Once you have selected the tasks this plan must perform the Maintenance Plan Wizard will guide you through setting up and fine-tuning each task.
- When you have fine-tuned the plan click Next.
- Order the tasks in order of importance. It is always best practice to set Database Backup first.
- When you have ordered the tasks click Next and you are done.

Defining Tasks for the Maintenance Plan

The following is a list of handy tips for defining tasks within the Maintenance Plan for SSMS:

Full Database Backup Task

This option allows you to choose which full database backup to perform when running the task in the maintenance plan. The best practice is to keep one database per plan.

The best choice for this option is to select 1.

Define Database Check Integrity Task

This is an SQL Server command that runs and checks the integrity of the database, the tables, and the data populating the database.

Define Shrink Database Task

You can shrink (compact) the database. This option helps to keep the database from expanding and taking up unnecessary space. It also speeds up the amount of time it takes to back up the database.

This option will let you choose which database you wish to shrink.

Define Reorganize Index Task

Every time you add, modify, and delete indexes you will 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 at least once a week or more for a busy database.

Define Rebuild Index Task

You can either reindex or reorganize the index tables in a database.

Define Update Statistics Task

This option will help to keep the update statistics log current. It handles the statistics for both indexes and individual columns. You will need to select which database you want to keep the statistical logs for.

Define History Cleanup Task

This is the screen that will delete or keep various historical data which can include:

- Backup information
- Recovery tables
- Agent job history
- Various other historical data

Define Backup Database (Differential) Task

This option can be set to back up every page in the database that was altered since the last full backup. You will be prompted to select the database to backup.

Define Backup Database (Transaction Log) Task

The transaction log backup backs up all the log records since the last backup.

Define Execute SQL Server Agent Job Task

Any job that is not listed in the Maintenance Plan Wizard can be tasked here. For example:

- Check for null values.
- Check the database meets specified standards.
- Many jobs are defined here.

SQL Server Agent Job Tasks are all listed in this section of the Maintenance Plan.

Define Maintenance Cleanup Task

You will define the cleanup action of the maintenance task.

This will ensure that no unnecessary files or objects are taking up disk space.

Report Options

This part is to set up the Maintenance Plan reporting system. This report will send a summary of the maintenance performed and how the scheduled tasks ran.

You can either have the report saved to disk in a specific folder or have it emailed to a specific account or both.

Running the Maintenance Plan

To run the SQL Maintenance plan you first need to check that the SQL Server Agent is running.

Press the Windows key and press the letter R.

Type services.msc in the Run box and hit Enter.

At the services screen, scroll through the services and find the SQL Server Agent (MSSQLEXPRESS).

Back in SSMS, right-click on the maintenance plan you created under the Maintenance Plans section.

Click Execute to start running the Maintenance Plan.

When the plan has successfully completed, click OK.

Close the dialogue box.

The maintenance reports can be viewed by right-clicking the maintenance plan you created and selecting View History. You will see all the different maintenance plans in the SQL Server instance along with the results of the plans.

To send off the database maintenance plan report you can email it.

You will first need to up a Server Agent. To do this, right-click on SQL Server Agent.

Select New.

Select Operator.

Allocate an operatorname.

Enter in a valid email address to which you want the report sent.

Click OK and the report will be sent to the designated email.

Right-click the maintenance plan that has just been successfully run.

Select the Modify option.

The maintenance plan design screen will appear on the right-hand side.

Here you will see a graphical representation of the completed tasks.

Click on Reporting and Logging. You will find the icon for this situated on the menu bar of the design plan. This is on the left of Manage Connections.

When the Reporting and Logging screen appears, select the "Send Report to an Email Recipient".

Select the maintenance plan operator name to send the report to.

This sets the report up to be sent to the selected recipient each time the Maintenance Plan is run.

The running and maintenance of a database is an important job. You need to ensure you have the right plan set up for the best performance and optimization of the system you're working 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.

It is important to have an understanding of the transaction log and its relevance to running a successful database system.

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 one of the most important files to ensure the integrity of a database in an SQL Server instance. A lot of importance is set around the use of and saving of the transaction log.

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

Choosing a recovery option for the database is one of the first things that needs to be done.

When the SQL Server is online, and even while users are still using the database services, all three SQL database backup options can be used.

When a backup and restore are run on the SQL Server, they are run within the confines of the SQL database recovery mode.

SQL has three database recovery model options:

Simple Recovery

The simple recovery model does not make allowances for the transaction log to be backed up.

This model should only be used for databases with infrequent updates.

The log file will be truncated only allowing for minimally logged transactions.

Full Recovery

The transaction log is backed up during a full recovery model. However, the log will be truncated once the backup process begins.

You can recover to any point in time with the full recovery option. In this model, you will need all the log files pertaining to the specific back to successfully restore the data.

Bulk-Logged Recovery

This recovery option is only used for bulk import operations. It is used in conjunction with the full recovery model.

It must be noted that with this option the backup cannot be recovered to any point in time. Rather you can only recover to certain points in time.

The option does not fill up the transaction log and is fast.

Changing the Recovery Model

The recovery model can be changed in a few ways.

- You can change the model in the SQL Server Management Studio screen by right-clicking on the database you want to change the setting for.
- Select the Properties option.
- Select Options.
- Choose the recovery model from the drop-down box.
- Reset the model to the preferred choice.

Alternatively, you can use an SQL statement query to do so. There are a few you can choose from:

Simple Recovery Option

> *alter database sqlebook set recovery simple*
>
> *go*

Full Recovery Option

> *alter database sqlebook set recovery full*
> *go*

Bulk Recovery Option

> *alter database sqlebook set recovery bulk_logged*
> *go*

Backups

Database Administrators base their backup plans on the following two measures:

- **Recovery Time Objective (RTO)** — The RTO measures the recovery period time from after a disruption notification of the business process gets logged.
- **Recovery Point Objective (RPO)** —The RPO measures the elapsed time passed during a disruption but before the data size that has been lost exceeds the maximum limit of the business process.

There are three types of database backup processes to choose from in SQL and these are:

Full Backup

The SQL Server will create a CHECKPOINT when a full database backup is set. This checkpoint is set to make sure that existing dirty pages are written to disk. For those that don't know what dirty or clean pages are:

- A dirty page is a page that has been altered or added to since it was loaded into memory. In other words, the original file that exists on the disk and the one in the memory is no longer the same.
- A clean page is a page that has been read from disk but nothing has changed. It is still the same as its counterpart on the disk it was read from.

The entire database will then be backed up along with the majority of the transaction log to keep the transactional consistency intact. Thus the database can be restored to the most recent point in time. It will also recover all the transactions up to the time the backup started.

What this means in plain English is that you can essentially only completely restore the database back to one point of time. This will be the last full backup that was taken of the database.

The reason this is not the most ideal backup option is that there is a chance of hours of data loss. For example, if the full backup ran every 24-hours and there was database corruption during the day, you would have to restore the database back to the previous night's data. This could result in the loss of an entire day's worth of work.

Transaction Log Backup

The transaction log backup will only backup the data in the transaction log. Which means only the transactions that were updated or changed in any way in between backups.

Because only recently changed files are backed up, it makes this backup model less resource-hungry. Because it does not take up a lot of resources to run, nor does it have to lock files it is backing up, backups can be performed throughout the day.

Differential Backup

Differential backups are backups that can run through the course of the day as they do not require a lot of resources to run. These backups will only backup a file that has changed during the day.

If there is a database corruption, using the differential model, the data can be restored to a more recent point in time. For instance, if you set the backup to run every hour, it scans for the most recently committed files and will back them up. The only data you may lose if there is a database corruption is an hour's worth (between backups).

The SQL Server keeps track of all the different pages that have been modified via flags and DIFF pages.

Database Backup

Performing a Backup

To back up a database in SSMS start by right-clicking the database object in SSMS.

Select Tasks.

Select Backup.

Select the type of backup to perform (full, differential, or transaction log).

Select when to run the backups.

The copy-only backup is for running a backup that will not affect the restore sequence.

Restoring a Database

To restore a database in SSMS you need to start by right-clicking the database object.

Select Tasks.

Select Restore.

Select Database.

Select the database contained in the drop-down menu.

Keep the rest of the tabs populated.

If you choose the Timeline, you can see a graphical diagram of when the last backup was created. This shows how much data was lost.

There is an option of recovering up to the end of a log, or a specific date and time.

The Backup Timeline media button is there to help you verify the integrity of the backup media using it to restore the database.

Select if you want to restore the data to the original structure or a new location.

Specify the required restore options. You can choose too:

- Overwrite the existing database
- Keep the existing database as a copy.

The recovery state either brings the database online or allows further backups to be applied.

Click OK.

The database will start the restore which could take some time depending on how large it is.

Attaching and Detaching Databases

The method of attaching and detaching databases is similar to that of a backup and restore method.

Attaching and detaching a database is used for:

- Copying the .MDF file and .LDF file to a new disk or server.
- Completely taking the database offline makes sure that the database cannot be accessed by any users or applications.
- The database remains offline until it is reattached.

A database can be taken offline — this is how you detach it. Then you can re-attach a database to a new destination if need be. This is very handy if for some reason the database engine needs to come down. You can run another database engine and reattach the database to the new instance.

A database has two main file groups, these files are the:

- .MDF — This file group is the database's primary data file, which holds its structure and data.

- .LDF. — This file group is where all the transactional logging activity and history is stored in.

These are the two main database files and are the files you would move to a new location once the database has been detached.

There is another file to be aware of and that is the .BAK file. This is the backup file that gets created during a database backup. You may notice that there are different versions of this file due to the various backups that have been created.

Detaching the Database

While a database is operational it is considered to be attached.

SQL has a stored procedure for detaching a database which makes this procedure a lot easier for a DBA.

The detach procedure is held in the "master" database store.

Have a look at the complexity of the detach stored procedure by following these steps in SSMS:

- Click on the Databases folder in the navigation panel.
- Click on the System Databases beneath the database folder.
- Click on the "master" database.
- Click on the Programmability option.
- Click on Stored Procedures.
- Click on System Stored Procedures.
- Scroll down until you find sys.sp_detach_db.
- Right-click on this object.
- Select 'Modify' in SSMS.
- This will bring up the syntax for this procedure on the Query screen.
- You can execute this code to detach the database.

The SQL syntax for detaching a database is as follows:

```
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
```

Attaching Databases

When the database has been detached, to get it operational again it needs to be attached.

Navigate the data directory.

Find the .MDF file for the database

Connect to an instance of the SQL database.

In SSMS open a New Query option.

Type in the following SQL script to reattach the database files, note you have to reattach both the .MDF and .LDF files.

```
create database databasename on
(filename = 'c:\sql data files\databasename.mdf'),
(filename = 'c:\sql data files\databasename_log.ldf') for attach
```

It should be noted that you do not have to reattach the .LDF as once you start up the database a new log will be created if it cannot find the original one.

CHAPTER 18:

DEADLOCKS

In most cases, multiple users access database applications simultaneously. This will mean that multiple transactions are being executed at one time on a database instance. When a transaction executes an operation request on a database resource such as a table, that resource is locked by default. Thus no other transaction can access the locked resource.

A deadlock occurs when two or more processes access resources that are locked by another process.

If you consider a scenario:

- TransactionA performs an operation on TableA.

- TableA is then locked by TransactionA.

- TransactionB tries to execute a parallel operation on TableA after performing a task on TableB and acquiring a lock for TableB.

- While TransactionB awaits a response from TableA, TransactionA attempts a process on TableB.

- What you now have are two transactions sitting in a deadlock awaiting a response from locked resources.

Create a dummy database by executing the following script in a new "Query" tab:

```
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')
```

Execute the script to create the database and table. The code you executed will also create a deadlock.

Deadlock Analysis and Prevention

In real-world scenarios, multiple users access the database simultaneously, which often results in deadlocks.

To analyze these deadlocks, you need the proper tools to be able to find out what is causing the deadlocks so you can create a way of preventing them.

SQL Server error logs, log what transactions and resources are involved in any deadlocks that may have occurred throughout the database.

Reading Deadlock Info via SQL Server Error Log

The SQL Server only provides minimal information about any deadlocks that may have occurred through the database. To find more detailed information about the deadlocks, you will need to do so in the SQL Server error log.

To log deadlock information to the error log you need to turn on the *trace flag 1222*.

You can turn *trace flag 1222* on, on a global level, by executing the following script in a new "Query" tab:

```
DBCC Traceon(1222, -1)
```

You can see if the *trace flag* is on, by executing the following query in a new "Query" line:

```
DBCC TraceStatus(1222)
```

The above statement results in the following output:

TraceFlag	Status	Global	Session

1222	1	1	0

The "Status value 1" indicates that the flag 1222 is on.

The 1 in the Global column implies that the trace flag has been turned on globally.

Now execute the code above to insert records into each table.

First, you need to drop *tablea* and *tableb.*

Once the tables have been dropped rerun the code below:

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')

To view the SQL Server error log 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	Parallel redo is started for database test with worker pool size
459	2017-11-01 15:51:47	Parallel redo is shutdown for database test with worker pool size
460	2017-11-01 15:51:55	deadloc list
461	2017-11-01 15:51:55	deadlock victim=process 1fcd9514ca8
462	2017-11-01 15:51:55	process list
463	2017-11-01 15:51:55	process id=process 1fcd9514ca8 taskpriority=o log...
464	2017-11-01 15:51:55	execution Stack
465	2017-11-01 15:51:55	frame procname=adhoc line=2 stmtstart=58 stmt...
466	2017-11-01 15:51:55	unknown
467	2017-11-01 15:51:55	frame procname=adhoc line=2 stmtstart=4 stmt...

The above log is just an example; errors are different depending on the database application software being used. All the information in the log about the deadlocks will start with "deadlock-list."

Analyze the log information retrieved by the deadlock you created by rerunning the script to create *tablea* and *tableb*.

Take note that your values may differ per column, but the information should be the same as per the table below:

ProcessInfo	Text
spid13s	deadlock-list
spid13s	**deadlock victim=process1fcf9514ca8**
spid13s	process-list
spid13s	process id=process1fcf9514ca8 taskpriority=0 logused=308 waitresource=KEY: 8:72057594043105280 (8194443284a0) waittime=921 ownerId=388813 transactionname=transactionBlasttranstarted=2019-05-27T15:51:46.547 XDES=0x1fcf8454490 lockMode=X schedulerid=3 kpid=1968 status=suspended spid=57 sbid=0 ecid=0 priority=0 trancount=2 lastbatchstarted=2019-05-27T15:51:54.380 lastbatchcompleted=2019-05-27T15:51:54.377 lastattention=1900-01-01T00:00:00.377 clientapp=Microsoft SQL Server Management Studio - Query hostname=DESKTOP-GLQ5VRA hostpid=968 loginname=DESKTOP-GLQ5VRA\Mani isolationlevel=read committed (2) xactid=388813 currentdb=8 lockTimeout=4294967295 clientoption1=671090784 clientoption2=390200

spid13s	executionStack
spid13s	frame procname=adhoc line=2 stmtstart=58 stmtend=164 sqlhandle=0x0200000014b61731ad79b1eec6740c98aab3ab91bd31af4d0000000000000000000000000000000000000000
spid13s	unknown
spid13s	frame procname=adhoc line=2 stmtstart=4 stmtend=142 sqlhandle=0x0200000080129b021f70641be5a5e43a1ca1ef67e9721c9700000000000000000000000000000000000000000
spid13s	unknown
spid13s	inputbuf
spid13s	UPDATE tableA SET patient_name = 'Thomas - TransactionB'
spid13s	WHERE id = 1
spid13s	process id=process1fcf9515468 taskpriority=0 logused=308 waitresource=KEY: 8:72057594043170816 (8194443284a0) waittime=4588 ownerId=388767 transactionname=transactionAlasttranstarted=2019-05-27T15:51:44.383 XDES=0x1fcf8428490 lockMode=X schedulerid=3 kpid=11000 status=suspended spid=54 sbid=0 ecid=0 priority=0 trancount=2 lastbatchstarted=2019-05-27T15:51:50.710 lastbatchcompleted=2019-05-27T15:51:50.710 lastattention=1900-01-01T00:00:00.710 clientapp=Microsoft SQL Server Management Studio - Query hostname=DESKTOP-GLQ5VRA hostpid=1140 loginname=DESKTOP-GLQ5VRA\Mani isolationlevel=read committed (2) xactid=388767 currentdb=8 lockTimeout=4294967295 clientoption1=671090784 clientoption2=390200
spid13s	executionStack
spid13s	frame procname=adhoc line=1 stmtstart=58 stmtend=164 sqlhandle=0x02000000ec86cd1dbe1cd7fc97237a12abb461f1fc27e27800000000000000000000000000000000000000000
spid13s	unknown
spid13s	frame procname=adhoc line=1 stmtend=138 sqlhandle=0x020000003a45a10eb863d6370a5f99368760983cacbf48950000000000000000000000000000000000000000000
spid13s	unknown
spid13s	inputbuf
spid13s	UPDATE tableB SET patient_name = 'Helene - TransactionA'
spid13s	WHERE id = 1
spid13s	resource-list
spid13s	keylockhobtid=72057594043105280 dbid=8 objectname=dldb.dbo.tableAindexname=PK__tableA__3213E83F1C2C4D64 id=lock1fd004bd600 mode=X 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 dbid=8 objectname=dldb.dbo.tableBindexname=PK__tableB__3213E83FFE08D6AB id=lock1fd004c2200 mode=X associatedObjectId=72057594043170816
spid13s	owner-list
spid13s	owner id=process1fcf9514ca8 mode=X
spid13s	waiter-list
spid13s	waiter id=process1fcf9515468 mode=X requestType=wait

There are three main parts logged by the SQL server in the error log which are:

1. **The Deadlock Victim**

   One of the processes involved in a deadlock is flagged as the deadlock victim by the SQL server. In

the table above, you can see the ID of the process selected as the deadlock victim. The process is process1fcf9514ca8 and has been highlighted in bold to make it easier for you to see for this example.

2. **Process List**

The process list references all the processes that are involved in the deadlock. For the table and the deadlock generated by the example there are two processes involved.

The processes list shows all the details of both of these processes. The process that is listed first in the process list is the one that is selected as the deadlock victim.

The process list provides a host of information about the processes such as:

- The login information of the process.
- The isolation level of the process.
- The script that the process was trying to execute.
- And much more

3. **Resource List**

The resource list has information pertaining to the resources involved in the deadlock event. For instance, in the example above the resources that were involved in the event were *tablea* and *tableb.*

Tips for Avoiding Deadlock

- Execute transactions in a single batch and keep them short.
- Release resources automatically after a certain time period.
- Sequential resource sharing.
- Don't allow a user to interact with the application when transactions are being executed.

CHAPTER 19:

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.

Using the process of normalization on your original database can help cut out some of the 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 needs 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. A 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 can 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 user 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 at 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.

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 table or group 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.

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.

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 to increase how well the database can 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 on 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:

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, Alan, Elizabeth	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.

No Repeated Column Groups

Repeated column groups are a 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.

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:

- Adheres to All the Conditions That Denote First Normal Form
- No Redundant Data in Any Column Except the Foreign Key Column
- 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. The 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.

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:

- Should Satisfy All the Rules of the Second Normal Form
- 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 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 with one another and isolating said relationships

CHAPTER 19:

REAL-WORLD USES

Throughout this book, you had the first-hand experience of using SQL in a stand-alone environment. For instance, you created databases, tables, populated tables, pivoted tables, cloned tables, used stored procedures, etc. You also performed various operations on the tables to manipulate the data the way the data is viewed to achieve various outcomes.

SQL syntax uses basic English language and the basics of SQL is quite easy to learn and understand. As there are different SQL applications it can be quite a difficult language to master. Although SQL is a language you can write code in one SQL application and use it in another, some applications may not support various functions. Some applications may also vary slightly in how they use a statement, thus, SQL can, at times, seem like a difficult language to master.

In SQL you can reuse scripts, which ensures you only ever have to create a complex script once and never have to rebuild it from scratch. You can store the script as a s*tored procedure* and access it at any time you may need to use it. Triggers can also be set to automatically run the script for you.

SQL in an Application

You will find that SQL has a few weaknesses along with all its strengths. One of these weaknesses is that it is not a procedural language. SQL must be combined with a procedural language like Pascal, C, FORTRAN, COBOL, Visual Basic, Java, or C++ to be used in an application. By combining SQL with one of these procedural languages, can overcome some of the language limitations, especially when creating an application.

SQL working alongside a procedural language allows for the creation of powerful applications with a wide range of capabilities. RDBMS has long since replaced the physical storage cabinets with paper files. These paper files needed large filing facilities or archives in which to store them. Now relational databases are used anywhere information is stored or retrieved, without the need for a person to physically go retrieve a paper file from reams of files. With the use of an RDBMS database you can retrieve data anytime, from anywhere, and a lot faster.

With the rise of RDBMS databases, there are now large platforms such as Wikipedia, and other informational based sites that you get information from. There is not much you cannot find on the Internet today, and all these sites use some form of RDBMS.

Sites like Amazon, Facebook, and even Netflix also use a database back-end in which their user base, products, etc. are stored. Database systems run on the back-end of these sites and also handle transactions such as orders and payments.

Example of industry or organizations that use RDBMS are:

- Banks are a prime example of an organization that uses databases. They are used for payment transactions, to manage user funds, deposits, and so on.
- Retail industries are another prime candidate for using RDBMS to store product information, inventory, sales transactions, customer bases, etc.
- Medical practices utilize databases for patient information, prescription medication, appointments, and other information.
- Environmental organizations use databases to store information such as weather patterns, land corrosion, environmental pollution, and various other statistics.
- Governmental agencies use RDBMS to store population statistics, resource utilization, and various other statistics about areas, people. and so on.
- Many, many other organizations use RDBMS daily such as schools, military, and even NASA.

Instead of trying to list all the applications RDBMS covers in this day and age, there are not many places or organizations that are not using a form of RDBMS.

RDBMS is not only a way of maintaining, organizing, and using data at the organization level but it also allows for an organization's customer base to interact with it. Customers have access to their profile information, payment methods, order, deliveries, etc. They can also manage their own data. In the medical field, this also means a customer has access to their patient records, medication history, and quick access to their medical advisor.

A digital database can leverage mobile applications. This expands on the opportunities for RDBMS including new

software platforms that use databases daily. The use of mobile applications paired with RDBMS has seen exponential growth in the capturing of data. In turn, this has led to more development opportunities as the need for RDBMS is fast expanding to keep up with technology. All you have to do is think along the lines of email apps, chat apps, and social media apps that are run on mobile devices daily.

New types of databases mean a call for new types of development, which in turn means job opportunities. It is no longer only IT specialists who are becoming database developers. With programs that are becoming easier to use because of clever APIs and using basic English to write scripts, it is getting easier for anyone to learn to code. Python is a language that has great API's, and it is also not that difficult to learn. It is also free to download and use. As such, someone with a finance background could easily pick up the language and write a program to suit their needs. Incorporating an SQL database with a bit of learning time is just as cost-effective to do.

In a real-world situation, multiple users access a database at the same time. Some databases do not handle such a high level of concurrency and the result can be corrupted or loss of data. SQL uses transactions to control atomicity, consistency, isolation, and durability to enable it to manage multiple uses without instability, data corruption, or data loss.

T-SQL statements are a sequence of SQL transactions that combine logically and complete an operation that would otherwise introduce database inconsistency.

Atomicity is a property that mimics a container that stores transaction statements. A successful statement means that all transactions have been completed. The entire operation will fail at any time during the statement a condition is not met and it will not process any further. If the process fails SQL will automatically roll-back to its previous state before the transaction was executed. Only once a row, or a page-wide lock is in place will a transaction be executed. While the transaction is locked it will prevent the object from being updated or manipulated by another user. The object gets reserved for the user utilizing it until their transaction has been completed.

Any attempts to use the object by another instance will fail with a "data locked" warning. However, even when an object is locked, another user can work on it. Their position to update will be placed in a queue until the object becomes free.

Using transactions to transform data enables a database to efficiently move from one consistent state to a new one. Always keep in mind and understand that transactions can modify more than one database at a time. For instance, if you change data in a primary key or foreign key field, make sure to simultaneously update the data with these key effects. If this is not done there will be inconsistencies in the SQL data. Transactions are vital to being able to update multiple tables consecutively.

The transformation of transactions is what reinforces isolation and is a property that prevents concurrent interactions from creating interference between them. This means that simultaneous transactions can take place at the same time, only one of them will be successful, while the other will fail or be queued.

Until transactions are complete they remain invisible and the first transaction completed is the one that will be processed first. Historically, failed transactions would either be deleted or the user would decide to try run the process again or end it. Most systems will now queue the information until it can try to rerun it. Only after a specified number of failed attempts will the process fail outright. This process is what lends more durability and stability to transactions which are handy in the case of a sudden system outage or power failure. In a worst-case scenario, all that would need to be done is to rerun the script. The system will automatically roll-back to before the execution of the failed transaction.

Any transactions that have failed to complete are retained. Rollbacks are accomplished by the A database engine will use transactional logs to identify what the state of data was before running the failed transaction. It will use this reference to match to a previous state and roll it back to that point.

Database Locks

There are many variations of a database lock which include different locking properties that can achieve varied database lock results.

Some common properties of a database lock may include:

Mode

A lock mode is the type of database lock being applied. There are a few different types of lock modes including:

- Shared lock mode — While the record is locked this mode allows data reads.
- Exclusive lock mode — Used with DML to provide an exclusive page or row to allow for data modification. It will render all access to the data inaccessible regardless of user rights while the data is being modified.
- Update lock — An update lock is a single object that allows data reads while the record is locked. The

update lock will also review and determine the necessity of an exclusive lock. This is handy to make sure the exclusive lock releases a record after it has been committed. The update lock shares the function of both the exclusive lock and shared lock.

Granularity

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.

Duration

One of the simplistic database lock properties to define is the duration property. The duration property of a database lock simply specifies the duration of the object lock.

It is important to note that the locks listed above are page-level locks. There are more advanced locks but those are beyond the scope of this book.

PL/SQL

Oracle developed an extension for procedural instruction using SQL syntax called PL/SQL. As discussed at the beginning of this chapter, SQL is a non-procedural language and needs to be paired with a procedural language to create full applications. The PL/SQL extension from Oracle expands the scope of SQL's capabilities as a programming language.

SQL, with the use of PL/SQL code, now offers more functions, stored procedures, and triggers. Triggers are what enable SQL to perform specific transactions when specific conditions are met. These conditions are usually defined by the database administrator and work in conjunction with various notification alerts, such as alarms. PL/SQL was developed by Oracle to meet their particular RDBMS product requirements. PL/SQL came about to overcome the non-procedural limitations of SQL.

CONCLUSION

By now you should have quite an in-depth knowledge of SQL. Because the code is simple and uses basic English statements, the coding may only slightly differ across various SQL server programs.

One of SQL's greatest strengths is its reusable code that can be easily adapted to different environments. So do not be afraid to try your new-found SQL skills in an environment other than MySQL. There are a few SQL databases that you can download and try. These databases include the likes of:

- Oracle Database
- SQL Server Management Studio(SSMS)
- MongoDB
- Microsoft SQL Server
- PostgreSQL
- MariaDB
- SQLite

The more SQL servers you try out and work on, the more diverse your SQL skills will become. There are also various SQL certification paths for those that want to add SQL as a certified skill to their resume to boost their IT careers or their salaries.

The following are a few SQL Certifications from Microsoft to consider:

- Microsoft Technology Associate (MTA): Database Fundamentals SQL
- Microsoft Certified Solutions Associate (MCSA): Database Administration
- Microsoft Certified Solutions Associate (MCSA): Database Development
- Microsoft Certified Solutions Expert (MCSE): Data Management and Analytics

There are also certifications from other RDBMS programs such as:

- MongoDB
  - MongoDB Certified Developer
  - MongoDB Certified DBA
- MySQL
  - Database Administration
  - Database Developer

You cannot go wrong by improving your SQL skills or getting certified in a world where data has become a valuable commodity.

Thank you for choosing *SQL: The Ultimate Intermediate Guide to Learning SQL Programming Step by Step* to help you further your SQL knowledge.

REFERENCES

Beginner's Guide to MySQL Storage Engines. (2018, April 4). SiSense. https://www.sisense.com/blog/beginners-guide-to-mysql-storage-engines/

Edwards, B. (2016, October 13). *The Forgotten World of Dumb Terminals*. PC Magazine. https://www.pcmag.com/news/the-forgotten-world-of-dumb-terminals

Hosch, W. (n.d.). *Charles Bachman*. Britannica. https://www.britannica.com/biography/Charles-William-Bachman

Hosch, W. (n.d.). *Edgar Frank Codd*. Britannica. https://www.britannica.com/biography/Edgar-Frank-Codd

Learn SQL. (n.d.). Tutorialspoint. https://www.tutorialspoint.com/sql/

Network Hardware. (n.d.). BBC. https://www.bbc.co.uk/bitesize/guides/zh4whyc/revision/7

Relational Database. (n.d.). IBM. *https://www.ibm.com/ibm/history/ibm100/us/en/icons/reldb/*

Relational Database Definition. (n.d.). Omni-Sci. https://www.omnisci.com/technical-glossary/relational-database

SQL - Constraints. (n.d.). Tutorialspoint. https://www.tutorialspoint.com/sql/sql-constraints.htm#:~:text=Constraints%20are%20the%20rules%20enforced,can%20go%20into%20a%20table.&text=UNIQUE%20Constraint

SQL Injections. (n.d.) PortSwigger. https://portswigger.net/web-security/sql-injection

SQL Tutorial. (n.d.). w3schools. https://www.w3schools.com/sql/default.asp