

MySQL and the SQL Language

Student Workbook

Version 2.1

Table of Contents

Module 1 Introduction to Relational Databases	1-1
Section 1-1 Relational Databases	1-2
Databases and DBMS	1-3
Database Operations	1-4
Relational Databases	1-5
SQL	1-6
Databases: A Collection of Related Tables.....	1-7
Normalization: Minimizing Redundancy	1-9
Section 1-2 Working with MySQL.....	1-11
Installing MySQL Server and Understanding the Database	1-12
Working with MySQL Workbench	1-14
MySQL Navigator	1-1
Exploring a Database	1-2
Viewing Records	1-5
Executing Queries.....	1-6
Exercises	1-8
Module 2 The SQL Language and Querying Data	2-10
Section 2-1 Querying Data	2-11
The SQL Language	2-12
Using SELECT Statements to Query Data	2-13
Example: Simple Select	2-15
Example: Select *	2-16
Optional Clauses in a SELECT Statement.....	2-17
ORDER BY Clause	2-18
Examples: ORDER BY	2-19
WHERE Clause	2-21
Example: WHERE	2-22
Example: Complex WHERE	2-23
Using BETWEEN for Comparisons	2-24
Example: BETWEEN	2-25
Using LIKE for Comparisons.....	2-26
Example: LIKE	2-27
QUERYING FOR NULL VALUES	2-28
Example: Querying with NULL	2-29
SELECT DISTINCT	2-30
Example: DISTINCT	2-31
Comments	2-32
Exercises	2-33
Section 2-2 CodeWars.....	2-36
CodeWars Kata	2-37
Module 3 Additional Querying Features.....	3-1
Section 3-1 Aggregate Functions	3-2
Aggregate Functions	3-3
Example: COUNT()	3-4
Example: SUM()	3-5
Example: AVG()	3-6
Example: MIN() and MAX().....	3-7
Section 3-2 Grouping Results	3-8
GROUP BY clause.....	3-9
Example: GROUP BY	3-10
Renaming Computed Fields - Using AS	3-11
Example: AS keyword	3-12
HAVING clause	3-13
Example: HAVING.....	3-14
Exercises	3-15
Section 3-3 Nested Queries	3-17
Nested Queries.....	3-18
Example: Nested Query	3-19
Exercises	3-21
Section 3-4 CodeWars.....	3-22
CodeWars Kata	3-23
Module 4 Querying Multiple Tables (JOINS).....	4-1
Section 4-1 Querying Multiple Tables (JOINS).....	4-2

Querying Multiple Tables (JOINS)	4-3
Inner Joins	4-4
Example: INNER JOIN	4-6
Beyond Inner Joins	4-7
Outer Joins	4-9
Example: Outer Joins.....	4-10
Viewing the Database	4-11
Exercises	4-12
Section 4–2 CodeWars.....	4-14
CodeWars Kata	4-15
Module 5 Modifying the Data and the Database.....	5-1
Section 5–1 Inserting, Updating and Deleting Data	5-2
Inserting, Updating and Deleting Data	5-3
INSERT INTO Statement.....	5-4
Example: INSERT INTO	5-5
UPDATE Statement.....	5-6
Example: UPDATE	5-7
DELETE Statement	5-8
Example: DELETE	5-9
Exercises	5-10
Section 5–2 Creating a Table	5-12
Creating a New Table	5-13
Creating a New Table	5-14
Example: CREATE TABLE.....	5-15
Other Features.....	5-16
Using AI Tools for SQL Data Generation	5-17
Section 5–3 CodeWars.....	5-20
CodeWars Kata	5-21

Module 1

Introduction to Relational Databases

Section 1–1

Relational Databases

Databases and DBMS

- Databases are collections of organized data stored such that a program called a database management system (DBMS) can interact with it
- There are many DBMS systems out there, including:

MySQL

SQL Server

Oracle

IBM Db2

PostgresSQL

Access

MongoDb

- Modern DBMS fall into two broad categories:

SQL based databases (ex: MySQL) are called relational databases and manage their data in collections of related tables

NoSQL databases (ex: MongoDb) have an entirely different way of managing data (often resembling JSON documents)

- In this course, we will examine only relational databases and the MySQL DBMS

Database Operations

- When we store data in a database, there are four basic types of operations we perform with data

Query the data for answers to questions

Add new data

Change data

Delete data

- There is a common acronym for these four types of operation: CRUD

Create

Read

Update

Delete

Relational Databases

- With relational databases, the data is organized in around tables made up of rows and columns.

Each *column* represents a particular "property" we want to store

Each *row* represents one set of data and is often called a "record" or "entry" (or row!)

The screenshot shows a database table titled "Customers". The table has columns: CustomerId, FirstName, LastName, DateCreated, and ClientId. The data consists of 12 rows, each representing a customer. Row 1 is selected, showing Homer Simpson as the customer. The last row is a new entry with all fields empty except for the primary key CustomerId (11).

	CustomerId	FirstName	LastName	DateCreated	ClientId
1	Homer	Simpson	13/06/2014 3:33:37 PM		
2	Peter	Griffin	13/06/2014 9:09:56 PM		
3	Stewie	Griffin	13/06/2014 9:16:07 PM		
4	Brian	Griffin	13/06/2014 9:16:36 PM		
5	Cosmo	Kramer	13/06/2014 9:16:41 PM		
6	Philip	Fry	13/06/2014 9:17:02 PM		
7	Amy	Wong	13/06/2014 9:22:05 PM		
8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM		
9	Marge	Simpson	13/06/2014 9:22:37 PM		
10	Bender	Rodriguez	13/06/2014 9:22:52 PM		
11	Turanga	Leela	13/06/2014 9:23:37 PM		
*	(New)			15/06/2014 9:00:01 PM	

- When you create a table, you have to specify:

the names of each column

the type of data (int, string, etc) held in each column

- You can also set other properties for each column, such as whether it can be null

SQL

- Modern relational database management systems use a language called SQL to interact with the data

SQL stands for "Structured Query Language"

- Programmers are opinionated about how to pronounce SQL

The standard says that "Ess-cue-ell" is the correct way pronounce the language

Famous authors who originally wrote about the technology, including Jennifer Widom and Christopher Date, pronounce it "sequel"

- A command to fetch data written using SQL is called a "query" and might look like this:

Example

```
SELECT FirstName, LastName  
FROM Customers  
WHERE LastName = 'Griffin'
```

- The information returned from a query is called the "result" or a "result set".

Peter	Griffin
Stewie	Griffin
Brian	Griffin

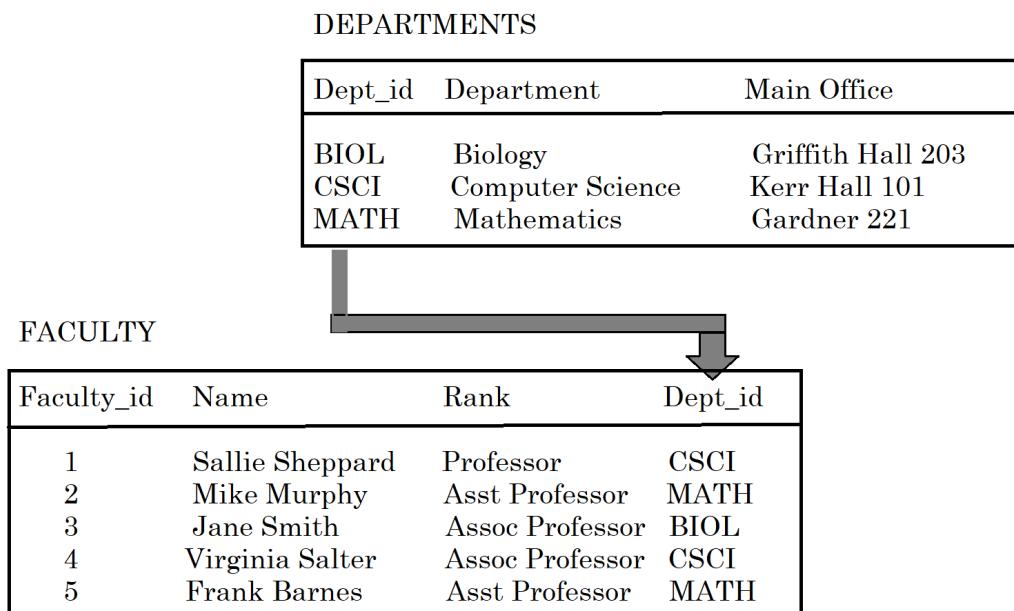
Databases: A Collection of Related Tables

- Relational databases typically consist of many tables
- Typically, each table contains one type of thing
ex: Products, stores, orders, etc
- Relationships between tables are established by having associated columns

The column that uniquely identifies a row is called the *primary key*

When a column in one table refers to the primary key of another table, it is called a *foreign key* and establishes a relationship between the two tables

- One-to-many relationships are represented as shown here:



- Many-to-many relationships are represented using a third table called an *intersection table*, as shown here:

PRODUCT

Product_id	Description	Price	Cost
1	Dragon Kite	9.99	2.76
2	Butterfly Kite	12.98	3.32
3	Box Kite Kit	29.99	11.00
4	100' string	1.00	.12
5	350' string	3.25	.29

STORE

Store_id	Address	City	State	Zip
1	111 Main St	Dallas	TX	75229
2	202 Broadway	Ft Worth	TX	76131
3	331 Straggle	Tulsa	OK	74133

STOCK

Product_id	Store_id	Qty_on_hand
1	1	7
1	2	0
1	3	2
2	1	5
2	2	7
2	3	0
3	1	1
3	2	5
3	3	3
...		

Normalization: Minimizing Redundancy

- Reducing redundancy is one of the goals of designing tables

When you duplicate data, then changes to data become complicated because you have to make changes multiple places

The process of designing a database that reduces redundancy is called *normalization*

- There's no "one answer" to database design

It is a matter of give and take and what is right for the situation

- For example, how do you represent Customers and Addresses?
- One organization might include customers and their addresses in a single table

CustomerID	CompanyName	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin	NULL	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.	NULL	05021	Mexico
ANTON	Antonio Moreno Taquería	Mataderos 2312	México D.F.	NULL	05023	Mexico
AROUT	Around the Horn	120 Hanover Sq.	London	NULL	WA1 1DP	UK
BERGS	Berglunds snabbköp	Berguvsvägen 8	Luleå	NULL	S-958 22	Sweden
BLAUS	Blauer See Delikatessen	Forsterstr. 57	Mannheim	NULL	68306	Germany

- Another organization might be aware that multiple customers live at the same address and separate address info into one table and customer info into another
- A row in the Customer table is linked to a specific row in the Address table using a key

Address Table

	addressId	addressLine1	addressLine2	city	state	zip
►	1	222 SW Main Street		Tulsa	OK	55422
	2	543 North Washington Avenue		Portland	OR	97204
	3	998 Blair Blvd.	Suite 34	New York	NY	22001

Customer Table

	customerId	firstName	lastName	addressId
►	1	Jimmy	Stewart	1
	2	Shelly	Duvall	3
	3	Robert	Duvall	3
	4	Fubar	Wilco	2

Section 1–2

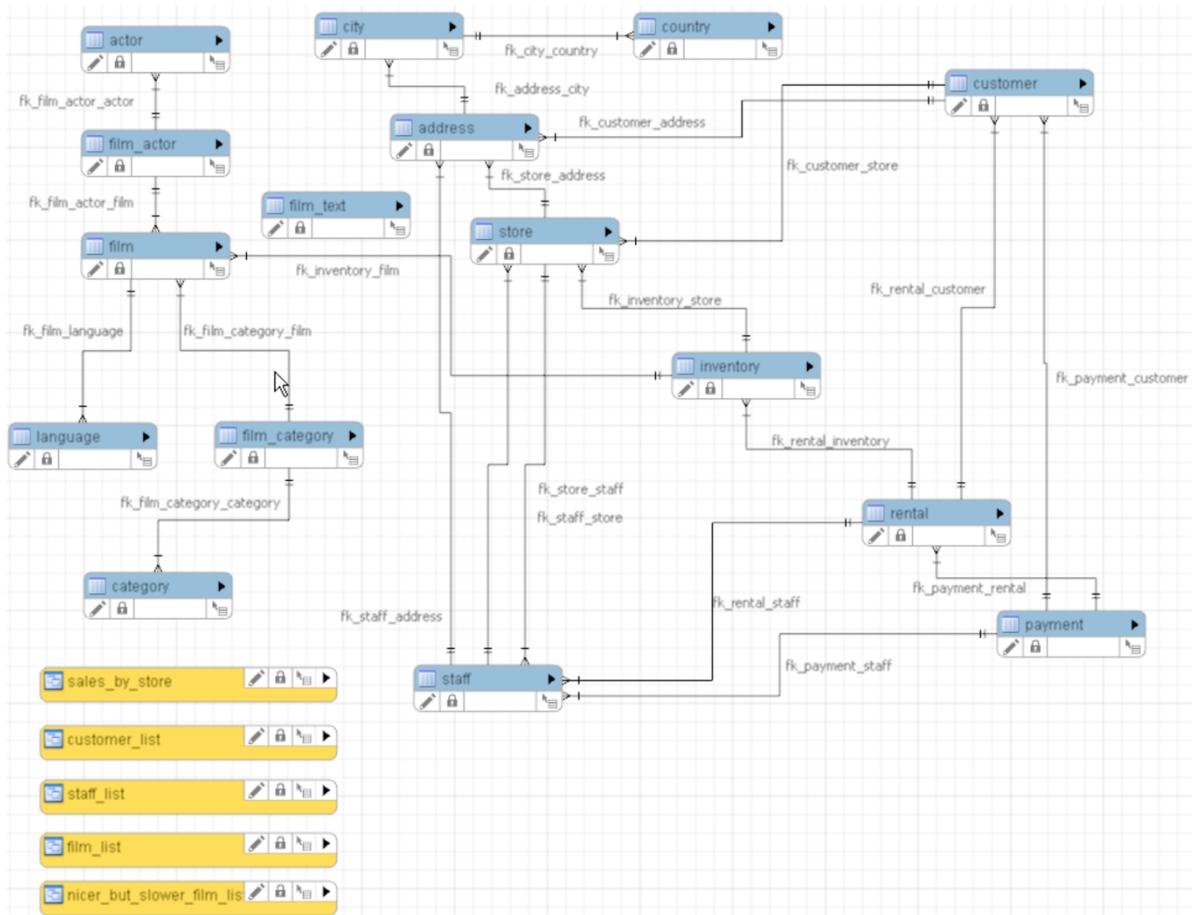
Working with MySQL

Installing MySQL Server and Understanding the Database

- The examples in this section of the course often use a sample database called sakila
 - It is a database for a company that sells movies

You can read about the schema here:

<https://dev.mysql.com/doc/sakila/en/>

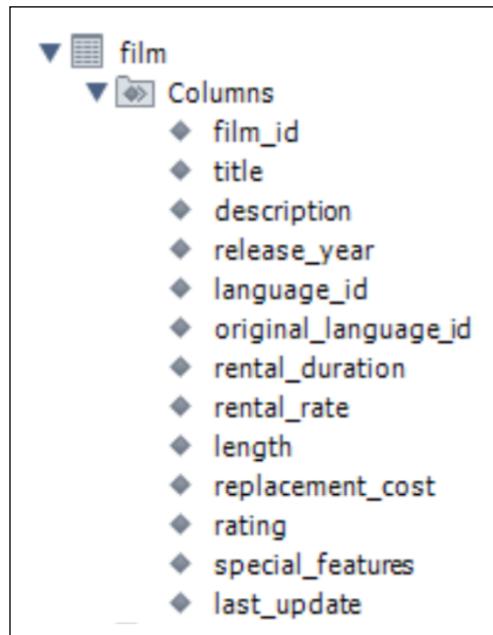


- Many of our early queries will use the film table

The film table is a list of all films that potentially might in stock in the stores

The actual in-stock copies of each film are represented in the inventory table.

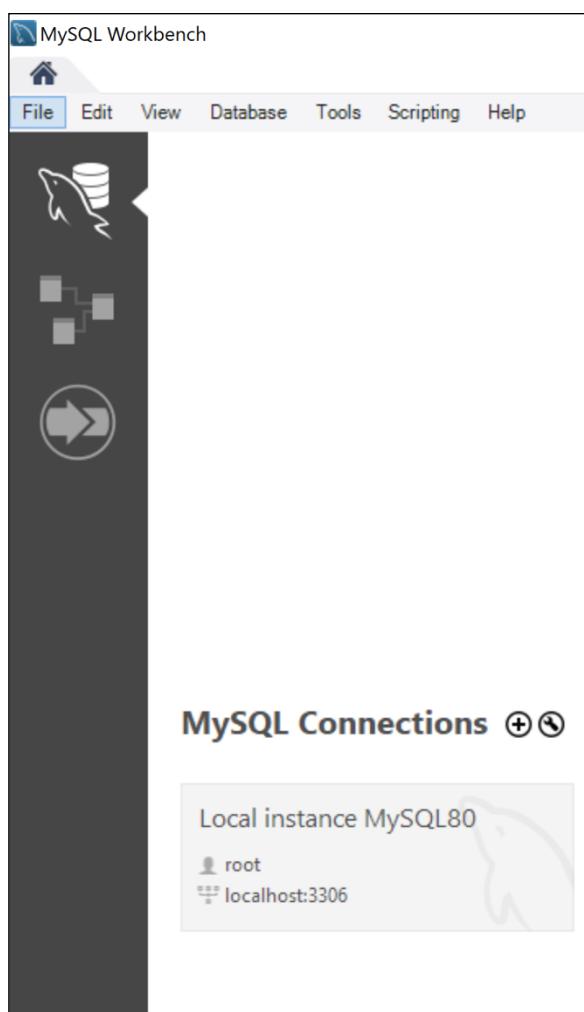
- The film table contains the following columns:



Working with MySQL Workbench

- You can run queries against MySQL databases using the MySQL Workbench
- Once you launch it, you should select your instance

Yours may look a little different, but should have your MySQL instance listed on the page



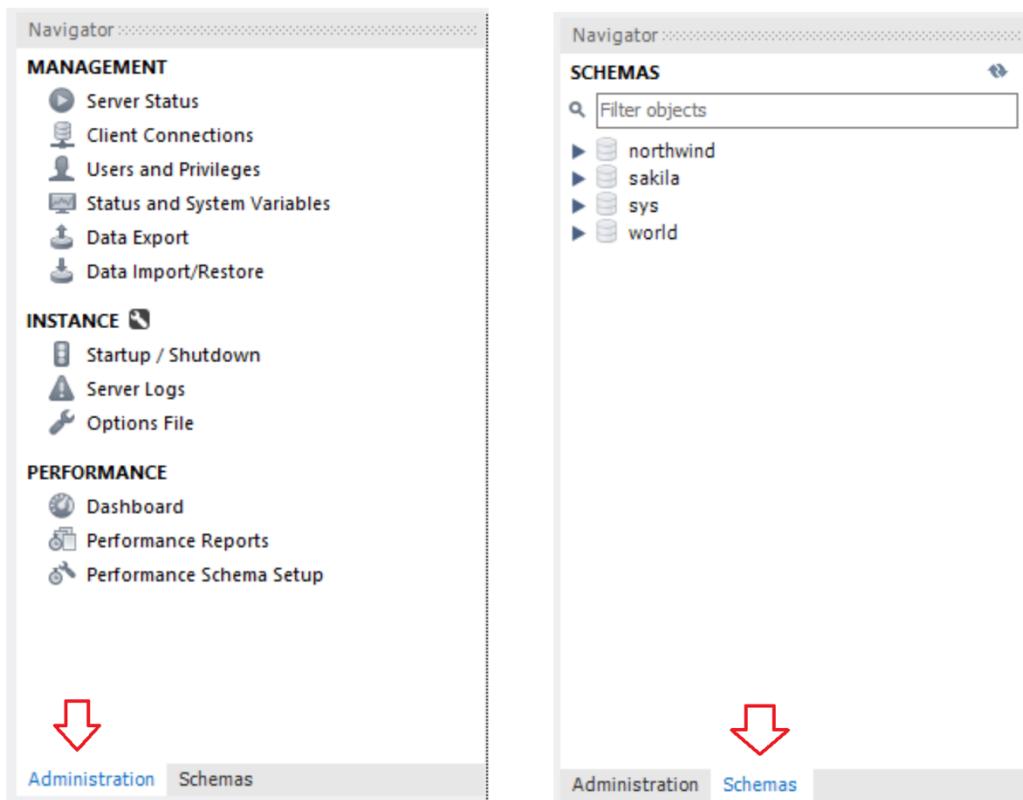
- When you select it, it will prompt you for your MySQL credentials

MySQL Navigator

- The **Navigator** window docked on the left of the MySQL Workbench window lets you:

perform administrative tasks

work with your databases



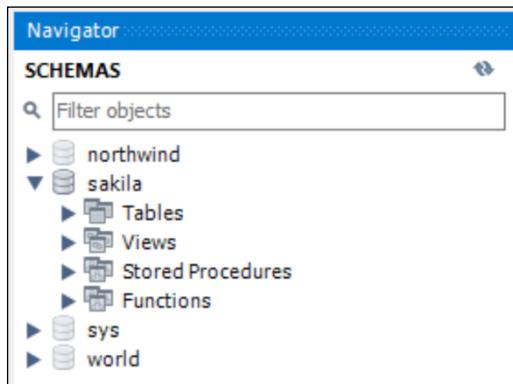
- When you switch to the schema tab, you will see your databases

You should have the Sakila and World databases installed as they are MySQL sample databases

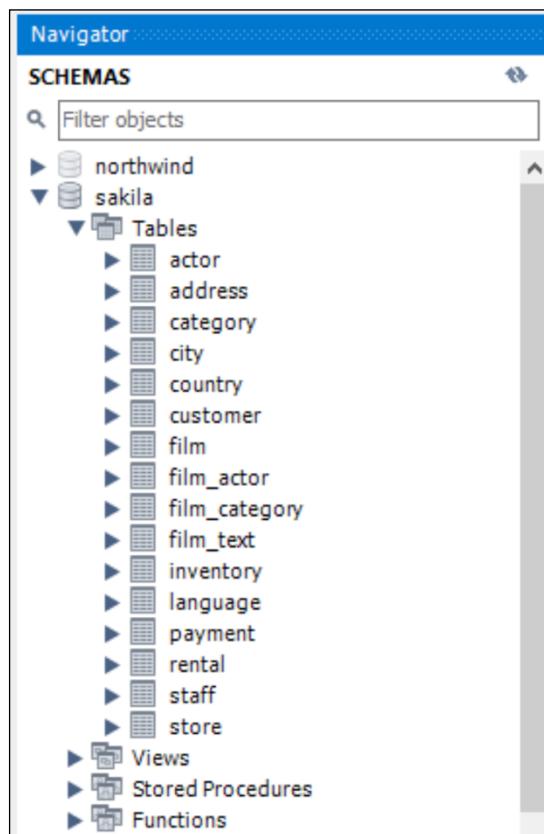
You will have to explicitly add other databases you want to interact with

Exploring a Database

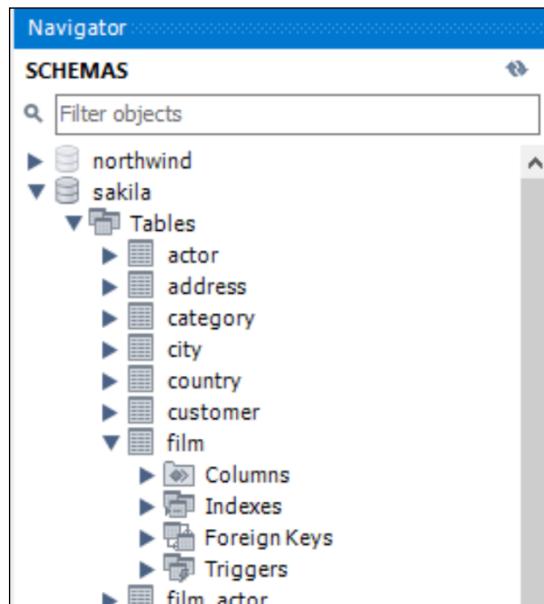
- If you drill into a database using **Navigator**, you can:
see the categories of elements in the database



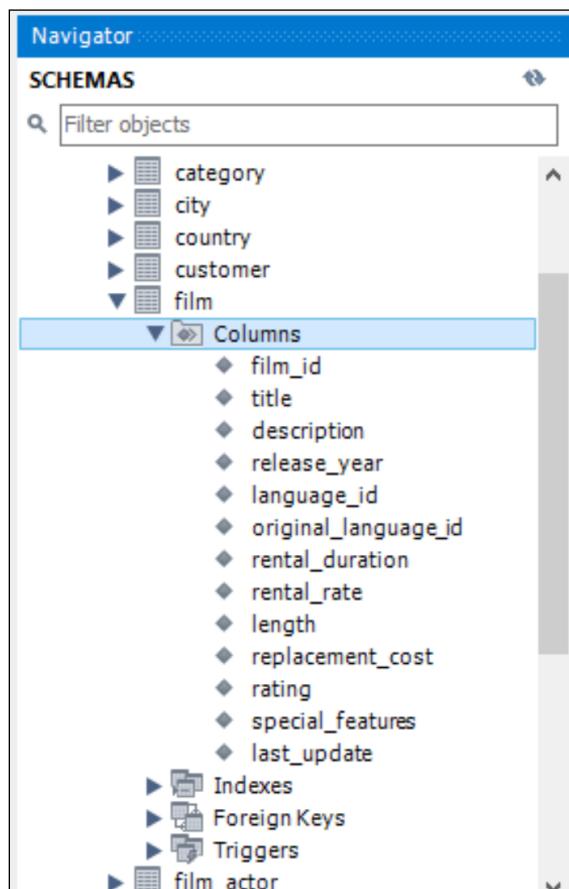
see all of the tables in the database



see the characteristics of a table



see the columns in a specific table



see the type of each column in a table

* when clicking on a table, look at the Information window

The screenshot shows the MySQL Workbench Navigator interface. In the left pane, under the 'SCHEMAS' section, the 'film' schema is selected. The right pane displays the 'Information' window for the 'film' table.

Table: film

Columns:

Column	Type
film_id	smallint UN AI PK
title	varchar(255)
description	text
release_year	year
language_id	tinyint UN
original_language_id	tinyint UN
rental_duration	tinyint UN
rental_rate	decimal(4,2)
length	smallint UN
replacement_cost	decimal(5,2)
rating	enum('G','PG','PG-13','R','NC-17')
special_features	set('Trailers','Commentaries','Deleted Scenes','Behind the Scenes')
last_update	timestamp

Viewing Records

- If you right-click on a table in the Navigator window, you can choose Select Rows - Limit 1000 to view data

It opens the Query tab, generates a SELECT statement for the data, and runs the query

The screenshot shows the MySQL Workbench interface with the 'film' database selected. The Query tab contains the SQL command: `SELECT * FROM sakila.film;`. The Result Grid displays 17 rows of film data. The columns are: film_id, title, description, release_year, language_id, original_language_id, rental_duration, and rental_rate. The data includes titles like 'ACADEMY DINOSAUR', 'ACE GOLDFINGER', and 'ALONE TRIP'. The rental rate for most films is 0.99, except for 'ALONE TRIP' which is 0.98. The right side of the interface has a sidebar with various tools and tabs: Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.

film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist ...	2006	1	NULL	6	0.99
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	NULL	3	4.99
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	NULL	7	2.99
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lum...	2006	1	NULL	5	2.99
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef An...	2006	1	NULL	6	2.99
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who...	2006	1	NULL	3	2.99
7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who ...	2006	1	NULL	6	4.99
8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Con...	2006	1	NULL	6	4.99
9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administ...	2006	1	NULL	3	2.99
10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjac...	2006	1	NULL	6	4.99
11	ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must ...	2006	1	NULL	6	0.99
12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef ...	2006	1	NULL	6	0.99
13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Croc...	2006	1	NULL	4	4.99
14	ALICE FANTASIA	A Emotional Drama of a A Shark And a Databas...	2006	1	NULL	6	0.99
15	ALIEN CENTER	A Brilliant Drama of a Cat And a Mad Scientist w...	2006	1	NULL	5	2.99
16	ALLEY EVOLUTION	A Fast-Paced Drama of a Robot And a Composer...	2006	1	NULL	6	2.99
17	ALONE TRIP	A Fast-Paced Character Studv of a Composer A...	2006	1	NULL	3	0.99

- To open additional Query tabs, click on the first toolbar icon



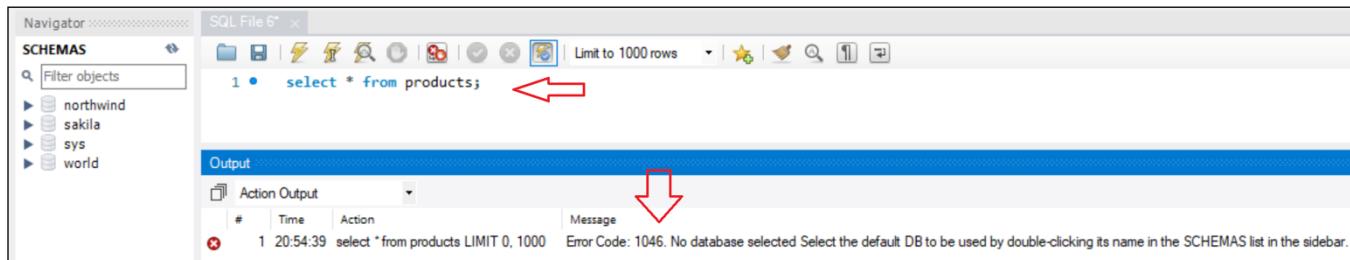
Executing Queries

- You can enter any query you want in the **Query tab** and then execute it



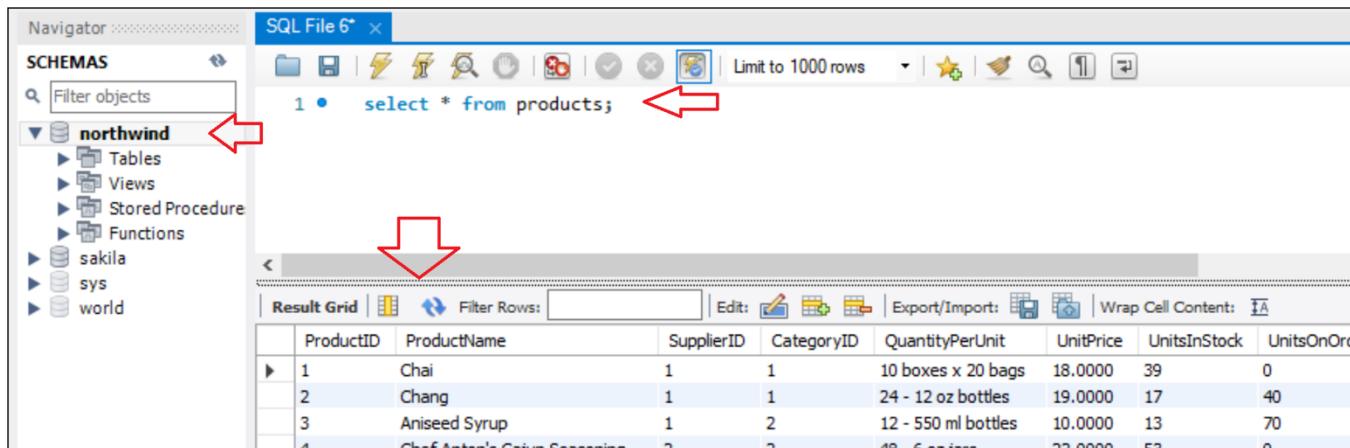
You execute it by specifying the icon on the Query tab just above where you enter the SQL statement

- However, because you may have several databases available, you have to select the database to run the query against

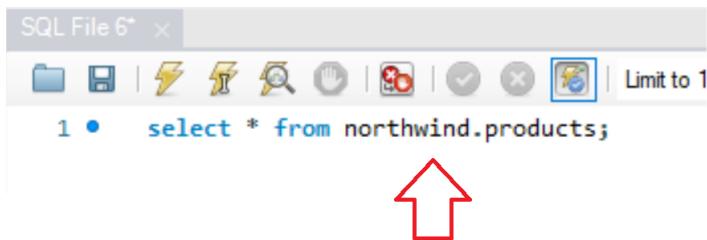


- One way to do this is to double-click the database schema in the **Navigator**

It will bold the database and become the default database that queries will be run against

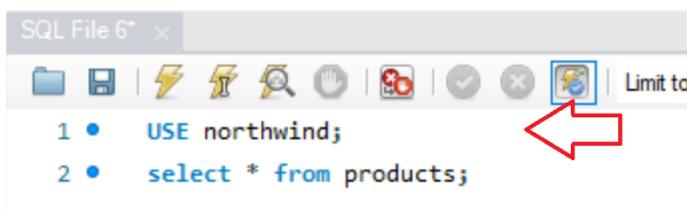


- Another way is to specify the database is to type it and a period in front of the table name



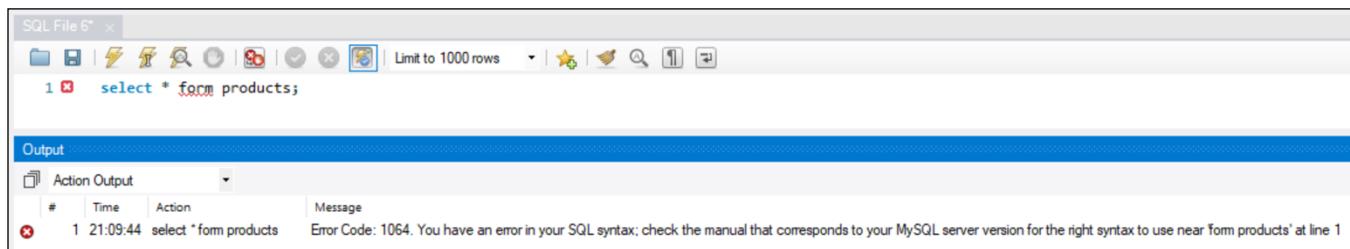
```
SQL File 6* ×
File | New | Open | Save | Run | Stop | Refresh | Limit to 1
1 • select * from northwind.products;
```

- Yet another way is to place a USE statement at the top of the script specifying the database



```
SQL File 6* ×
File | New | Open | Save | Run | Stop | Refresh | Limit to
1 • USE northwind;
2 • select * from products;
```

- If you have a syntax error in your query, the Output window will describe the error



```
SQL File 6* ×
File | New | Open | Save | Run | Stop | Refresh | Limit to 1000 rows | 
1 ✘ select * form products;
```

Output		
Action	Output	Message
1	21:09:44	select * form products Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'form products' at line 1

- For more information on how MySQL Workbench works, see:
<https://dev.mysql.com/doc/workbench/en/wb-sql-editor-query-panel.html>

Exercises

EXERCISE 1

In this exercise, you will install the Northwind database and then run some simple queries against it.

We will use the Microsoft Northwind database for many of the exercises and examples in this workbook. A script to install the database is included for you in the data files zip file.

Once you have installed your Northwind database, to see your new database in the Navigator window, you may have to refresh it. Right-click in the Navigator window and choose **Refresh All**.

Being able to use a database efficiently in your applications, requires that you know your data. You should be familiar with the database structure and the data in the database. Take the next several minutes to familiarize yourself with this database.

Northwind is a database for a small grocery store. Take a few minutes to examine the schema. Then answer the following questions by looking at the data in the tables or the table structure.

NOTE: These question should all be answered without the use of SQL Queries (since we have not learned about them yet). Think about how you would discover this information by just looking at the data in the tables.

1. What categories of products does Northwind Traders sell?
2. What is the name of a product that sounds interesting to you?
3. What are some of the countries in which we have customers?
4. What date was our first order placed? How did you determine that?
5. What date was our last order placed?
6. How many customers do we currently have?
7. Do we have any suppliers in South America? Europe? Africa?
8. What is our most expensive product?
9. What is our least expensive product?
10. We have a customer named "The Big Cheese" if I wanted to get details about their last order from us how would you find that information?
 - a. What tables would you use?

- b. How would you link the data in the tables you are using
- c. What information might I want to know about their order?
 - i. Order Date?
 - ii. Sales Total?
 - iii. Product Name?
 - iv. Is there other information that you might find interesting?

Module 2

The SQL Language and Querying Data

Section 2–1

Querying Data

The SQL Language

- When you want to interact with SQL databases, you use SQL
- SQL is a *declarative language*

JavaScript and Java are *procedural* languages

- The difference is that, in JavaScript, you must specify *how* to do things that you want -

Example: executing a group of statements multiple times with a `for` loop

- With SQL, you specify *what* you want.

All SQL databases have sophisticated execution optimizers that figure out the best way to produce your results based on what the database looks like when you execute your command

- Over the next several modules, we will see examples of, and practice, SQL

Using SELECT Statements to Query Data

- Much of the time, when we interact with SQL databases we will be looking for specific data to read
- In SQL, you do this with a SELECT statement
- The SELECT statement's minimum requirements are:

SELECT (a keyword)

the list of columns to be displayed (or * for all columns)

FROM (a keyword)

the table to get data from

Syntax

```
SELECT column1, column2, etc  
FROM table-name;
```

Syntax

```
SELECT *  
FROM table-name;
```

- **The case of the keywords SELECT and FROM isn't important**

It can be in uppercase, lowercase or mixed case

- **The statement, and all other SQL statements, must end with a semicolon**

Example: Simple Select

SAKILA DATABASE: The `film` table is shown below:

film	
▼	Columns
◆	film_id
◆	title
◆	description
◆	release_year
◆	language_id
◆	original_language_id
◆	rental_duration
◆	rental_rate
◆	length
◆	replacement_cost
◆	rating
◆	special_features
◆	last_update

QUERY: What are the titles of the films we sell and their ratings?

APPROACH: Use a SELECT statement, list the columns we want after the keyword SELECT, and list the film table after the keyword FROM.

Query

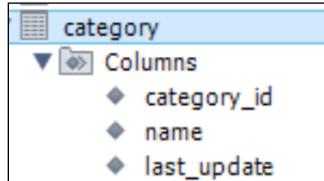
```
SELECT title, rating  
FROM film
```

RESULTS:

title	rating
ACADEMY DINOSAUR	PG
ACE GOLDFINGER	G
... there are many rows returned ...	

Example: Select *

SAKILA DATABASE: The `category` table is shown below:



QUERY: What are the categories of films we carry?

APPROACH: Use a SELECT statement and list all columns from the category table.

Query

```
SELECT *
FROM category
```

RESULTS:

```
+-----+-----+
| category_id | name           | last_update   |
+-----+-----+
| 1          | Action          | 2006-02-15 04:46:27 |
| 2          | Anumation       | 2006-02-15 04:46:27 |
+-----+-----+
| ... there are many rows returned ... |
+-----+-----+
```

Optional Clauses in a SELECT Statement

- The SELECT statement has several optional clauses, including:

ORDER BY specifies how to order the returned rows

WHERE specifies which rows to retrieve

AS can be used to provide an alias for a column or expression in the SELECT

GROUP BY groups rows based on a specified characteristic so that an aggregate function can be applied to each group

HAVING specifies which groups to include in the result

ORDER BY Clause

- ORDER BY lets you control the order of the data that is returned

ORDER BY is generally the last clause in the SELECT statement

- By default, it returns data in ascending order based on the column selected

SYNTAX

```
SELECT column1, column2, etc  
FROM table  
ORDER BY any-column;
```

- However, you can use the DESC keyword after the column name to specify rows should be returned in descending order

SYNTAX

```
SELECT column1, column2, etc  
FROM table  
ORDER BY any-column DESC;
```

- NOTE: In some situations, you may want to have more than one field you order by

For example, order by state and, within state, by city

Examples: ORDER BY

QUERY: What are the films we carry ordered by the length of the film (shortest first)?

APPROACH: Use an ORDER BY on the query to sort by the length column.

Query

```
SELECT film_id, title, length
FROM film
ORDER BY length;
```

RESULTS:

film_id	title	length
15	ALIEN CENTER	46
469	IRON MOON	46
... there are many rows returned ...		

QUERY: What are the films we carry ordered by the length of the film (longest first)?

APPROACH: Use an ORDER BY on the query to sort by the length column and used a descending sort

Query

```
SELECT film_id, title, length
FROM film
ORDER BY length DESC;
```

RESULTS:

film_id	title	length
141	CHICAGO NORTH	185
182	CONTROL ANTHEM	185
... there are many rows returned ...		

QUERY: What are the films we carry ordered by the length of the film (longest first), but if multiple films have the same length then order by film_id?

APPROACH: Use an ORDER BY on the query to sort by the length column and used a descending sort

Query

```
SELECT film_id, title, length
FROM film
ORDER BY length DESC, film_id;
```

RESULTS:

film_id	title	length
141	CHICAGO NORTH	185
182	CONTROL ANTHEM	185
... there are many rows returned ...		

WHERE Clause

- The WHERE clause limits the rows returned to those that match a specified condition
- The condition is specified using the comparison operators = <> > >= < <=

SYNTAX

```
SELECT column1, column2, etc  
FROM table  
WHERE expression
```

- You can build a complex condition using the logical operators AND, OR, and NOT
- You can use parentheses to control order of precedence
- NOTE: String comparisons in MySQL are typically case insensitive

Example: WHERE

QUERY: What films have a "PG" rating?

APPROACH: Use a WHERE clause and specify a condition that the rating column must be "equal to" the string "PG".

Query

```
SELECT film_id, title, rating  
FROM film  
WHERE rating = "PG";
```

RESULTS:

film_id	title	rating
1	ACADEMY DINOSAUR	PG
6	AGENT TRUMAN	PG
... there are many rows that match ...		

Example: Complex WHERE

QUERY: What films have a "PG" rating that run between 90 and 120 minutes? List the results in descending order by length. If two or more films have the same length, then list them in alphabetical order by title.

APPROACH: Use a WHERE clause and specify a condition that the rating column must be "equal to" the string "PG" and that the value in the rating column must be between 90 and 120.

Query

```
SELECT film_id, title, rating, length
FROM film
WHERE rating = "PG" AND (length >= 90 AND length <= 120)
ORDER BY length DESC, title;
```

RESULTS:

film_id	title	rating	length
477	JAWBREAKER BROOKLYN	PG	118
645	OTHERS SOUP	PG	118
... there are many rows that match ...			

Using BETWEEN for Comparisons

- You can use the BETWEEN operator in a WHERE clause to specify a range of values

Specify the minimum and maximum values separated by the word AND

* NOTE: The values are inclusive in the range.

Example

```
WHERE length BETWEEN 89 AND 91
```

Example: BETWEEN

QUERY: What films have a running length in the range 89-91 minutes?

APPROACH: Use BETWEEN to specify a range of values for length.

```
SELECT film_id, title, length
FROM film
WHERE length BETWEEN 89 AND 91
ORDER BY title;
```

RESULTS:

film_id	title	length
28	ANTHEM LUKE	91
57	BASIC EASY	90
... there are many rows that match ...		

Using LIKE for Comparisons

- The standard relational operators in a WHERE clause work well for numeric comparisons
- The LIKE allows you to perform custom string comparison

Partial string match using the percent sign (%) as a wildcard match

Example

```
WHERE title LIKE "Sit%"
```

means the value **starts with** "Sit" (or even "sit" since SQL doesn't care about case)

```
WHERE title LIKE "%sit"
```

means the value **ends with** "sit"

```
WHERE title LIKE "%sit%"
```

means **contains** "sit" anywhere in the string

Example: LIKE

QUERY: What films titles start with the word "Theory"?

APPROACH: Use LIKE with a wildcard value of "Theory%" in the query.

```
SELECT film_id, title, rating
FROM film
WHERE title LIKE "Theory%";
```

RESULTS:

film_id	title	rating
886	THEORY MERMAID	PG-13

QUERY: What films have the phrase "sec" anywhere in their title?

APPROACH: Use LIKE with a wildcard value of "%sec%" in the query.

```
SELECT film_id, title, rating
FROM film
WHERE title like "%sec%"
ORDER BY title;
```

RESULTS:

film_id	title	rating
231	DINOSAUR SECRETARY	R
461	INSECTS STONE	NC-17
... there are many rows that match ...		

QUERYING FOR NULL VALUES

- Some tables will allow **NULL** for some fields

This does complicate queries a tiny bit

- For example:

if you are querying the film table for all films that have an original_language_id of 1, films that have a **NULL** for that column won't be included

But if you are querying for all films that DO NOT have an original_language_id of 1, films that have a **NULL** for that column *still won't be included*.

- You must use a specific test to match a column to **NULL**

Syntax

```
WHERE column-name IS NULL
```

```
WHERE column-name IS NOT NULL
```

Example: Querying with NULL

QUERY: Find all films that don't have a value for original_language_id

```
SELECT film_id, title  
FROM film  
WHERE original_language_id IS NULL;
```

QUERY: Find all films that do have a value for original_language_id

```
SELECT film_id, title  
FROM film  
WHERE original_language_id IS NOT NULL;
```

SELECT DISTINCT

- The **SELECT DISTINCT** statement returns only distinct (unique) values
- When applied to a row, it returns a row the totality of all the fields is different from others returned

Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

- When applied to a column, it returns only unique values for the column

Syntax

```
SELECT DISTINCT(column1)
FROM table_name;
```

Example: DISTINCT

QUERY: Find all of the unique prices we rent films for

```
SELECT DISTINCT(rental_rate)
FROM film;
```

RESULTS:

rental_rate
0.99
4.99
2.99

Comments

- You can add both single-line and multi-line comments to a SQL script
- Single line comments start with --

The rest of the line after the -- is ignored

Example

```
--Select all films that run 1.5 hrs or less
SELECT film_id, title, length
FROM film
WHERE length <= 90;    -- 90 minutes is 1.5 hrs
```

- Multi-line comments start with /* and end with */

All text between /* and */ is ignored

Example

```
/*
  Select all films that run 1.5 hrs or less
*/
SELECT film_id, title, length
FROM film
WHERE length <= 90;
```

Exercises

Create a new folder in your pluralsight directory and name it `workbook-7`. You will do all of this week's exercises in this folder.

EXERCISE 1

W3Schools not only has CSS, Bootstrap, and JavaScript material, it had a fairly detailed section on SQL. Take a few minutes to do the following practice exercises:

SQL Select: https://www.w3schools.com/sql/exercise.asp?filename=exercise_select1

SQL Order By: https://www.w3schools.com/sql/exercise.asp?filename=exercise_orderby1

SQL Where: https://www.w3schools.com/sql/exercise.asp?filename=exercise_where1

SQL Like: https://www.w3schools.com/sql/exercise.asp?filename=exercise_like1

SQL Between: https://www.w3schools.com/sql/exercise.asp?filename=exercise_between1

EXERCISE 2

Create a new project folder named `basic-queries` in the `workbook-7` directory. Create any `.sql` files directly in the `basic-queries` folder.

Use the MySQL Workbench and the Northwind database to answer the following questions

Create a new `.sql` file for each query and name each file with the question number, i.e. `02_product_list.sql`. (This approach helps to organize and more easily find your queries later)

Alternatively, you may want to put all SQL statements in the same `.sql` script with comments before each query and then only run the selected query by pressing the 2nd lightning bolt.

1. What is the name of the table that holds the items Northwind sells?
2. Write a query to list the product id, product name, and unit price of every product.
3. Write a query to list the product id, product name, and unit price of every product. Except this time, order them in ascending order by price.
4. What are the products that we carry where the unit price is \$7.50 or less?

5. What are the products that we carry where we have at least 100 units on hand? Order them in descending order by price.
6. What are the products that we carry where we have at least 100 units on hand? Order them in descending order by price. If two or more have the same price, list those in ascending order by product name.
7. What are the products that we carry where we have no units on hand, but 1 or more units of them on backorder? Order them by product name.
8. What is the name of the table that holds the types (categories) of the items Northwind sells?
9. Write a query that lists all of the columns and all of the rows of the categories table? What is the category id of seafood?
10. Examine the Products table. How does it identify the type (category) of each item sold? Write a query to list all of the seafood items we carry.
11. What are the first and last names of all of the Northwind employees?
12. What employees have "manager" in their titles?
13. List the distinct job titles in employees.
14. What employees have a salary that is between \$200 0 and \$2500?
15. List all of the information about all of Northwind's suppliers.
16. Examine the Products table. How do you know what supplier supplies each product? Write a query to list all of the items that "Tokyo Traders" supplies to Northwind

Commit and push your code!

(OPTIONAL) EXERCISE 3

The sqlbolt.com web site has some nice tutorial and practice exercises that build up to more complex situations.

Anytime you finish an exercise early, go there and start working your way through the exercises.

Section 2–2

CodeWars

CodeWars Kata

- Complete these Katas for additional SQL practice

<https://www.codewars.com/kata/582365c18917435ab3000020/sql>

<https://www.codewars.com/kata/59440034e94fae05b2000073/sql>

Module 3

Additional Querying Features

Section 3–1

Aggregate Functions

Aggregate Functions

- **SQL has functions that can be used to perform calculations on values from groups of rows, including**

The COUNT() function counts the number of occurrences of a value in a specified column

* It can also be used to count the number of rows

The SUM() function adds up values in the specified column

The AVG() function returns the average value from a specific column

- **SQL also functions that can be used to find the minimum or maximum value of a column**

The MIN() function finds the minimum value

The MAX() function finds the maximum value

Example: COUNT()

QUERY: How many films are in the films table?

APPROACH: Use the COUNT() function to count the number of rows in the film table.

```
SELECT COUNT(*)  
FROM film;
```

RESULTS:

COUNT(*)
1000

QUERY: How many distinct ratings are represented in the films table?

APPROACH: Use the COUNT() function combined with DISTINCT to count the number of ratings in the film table.

```
SELECT COUNT(DISTINCT(rating))  
FROM film;
```

RESULTS:

COUNT(DISTINCT(rating))
5

Example: SUM()

QUERY: If I wanted to watch all of the movies in the film catalog, how long would it take?

APPROACH: Use the SUM() function to add up all the length values in the films table.

```
SELECT SUM(length)
FROM film;
```

RESULTS:

SUM(length)
115272

Example: AVG()

QUERY: What is the average cost to rent a "G"-rated film?

APPROACH: Use the AVG() function to find the average value in the rental_rate column of all films whose rating is "G".

```
SELECT AVG(rental_rate)
FROM film
WHERE rating = "G";
```

RESULTS:

AVG(rental_rate)
2.888876

Example: MIN() and MAX()

QUERY: How short is the shortest film? What about the longest?

APPROACH: Use the MIN() and MAX() function to examine the length.

```
SELECT MIN(length)
FROM film;
```

RESULTS:

MIN(length)

46

```
SELECT MAX(length)
FROM film;
```

RESULTS:

MAX(length)

185

Section 3–2

Grouping Results

GROUP BY clause

- The GROUP BY clause allows you to execute aggregate functions on "groups" of data created by the query

GROUP BY specifies the column(s) used to create the groups

The aggregate functions returns a value for EACH group created

Syntax

```
SELECT column1, column2, etc  
FROM table_name  
WHERE condition  
GROUP BY column_name(s)  
ORDER BY column_name(s);
```

Example: GROUP BY

QUERY: How many movies are available broken down by rating (G, PG, PG-13, etc)?

APPROACH: Use the GROUP BY clause to create groups of films by rating and then use the COUNT() function to count the number of rows in each group.

```
SELECT rating, COUNT(*)
FROM film
GROUP BY rating;
```

RESULTS:

rating	COUNT(*)
PG	194
G	178
NC-17	210
PG-13	223
R	195

QUERY: What is the average price to rent a movie broken down by rating (G, PG, PG-13, etc)??

APPROACH: Use the GROUP BY clause to create groups of films by rating and then use the AVG() function to calculate the average rental_rate of rows in each group.

```
SELECT rating, avg(rental_rate)
FROM film
GROUP BY rating;
```

RESULTS:

rating	AVG(rental_rate)
PG	3.051856
G	2.888876
NC-17	2.970952
PG-13	3.034843
R	2.9387818

Renaming Computed Fields - Using AS

- Computed fields don't have an official name in a SQL query

Example

```
SELECT rental_id, SUM(amount)
FROM sakila.payment
GROUP BY rental_id
ORDER BY rental_id;
```

	rental_id	SUM(amount)
▶	NULL	9.95
	1	2.99
	2	2.99
	3	3.99
	4	4.99
	5	5.99

- This can be a problem if you want to use it to order the results
- SQL provides the AS keyword to create an alias for the column name

You can use it for ordering or other purposes

Example

```
SELECT rental_id, SUM(amount) AS total_amount
FROM sakila.payment
GROUP BY rental_id
ORDER BY rental_id;
```

AS can also create an alias for a table name; we will see this in the next module

Example: AS keyword

QUERY: What is the average price to rent a movie broken down by rating (G, PG, PG-13, etc) and displayed in ascending order by average price?

APPROACH: Use the GROUP BY clause to create groups of films by rating and then use the `AVG()` function to calculate the `rental_rate` of rows in each group. Make sure to name the value returned by the `AVG()` function so that we can use it in the ORDER BY clause.

```
SELECT rating, AVG(rental_rate) AS avg_rate
FROM film
GROUP BY rating
ORDER BY avg_rating;
```

RESULTS:

rating	avg_rate
G	2.888876
R	2.938781
NC-17	2.970952
PG-13	3.034843
PG	3.051856

HAVING clause

- The HAVING clause is used with the GROUP BY clause
- It allows you to include only those groups that meet a specified condition

Syntax

```
SELECT column1, column2, etc  
FROM table_name  
WHERE row-condition  
GROUP BY column_name(s)  
HAVING group-condition  
ORDER BY column_name(s);
```

- If the WHERE clause is used to select rows, the HAVING clause is used to select groups

Example: HAVING

QUERY: What is the average rating for movies broken down by rating (G, PG, PG-13, etc)?
NOTE: I'm not interested in the rating if there are less than 200 films in the group.

APPROACH: Use the GROUP BY clause to create groups of films by rating and then use the COUNT() function to count the number rows in each group. Only display the groups that have at least 200 rows.

```
SELECT rating, COUNT(*)
FROM film
GROUP BY rating
HAVING COUNT(*) >= 200
ORDER BY rating;
```

RESULTS:

rating	COUNT(*)
NC-17	210
PG-13	223

Exercises

EXERCISE 1

Let's look back at W3Schools again to test ourselves with aggregate functions and GROUP BY. Take a few minutes to do the following practice exercises:

SQL Functions: https://www.w3schools.com/sql/exercise.asp?filename=exercise_functions1

SQL Group By: https://www.w3schools.com/sql/exercise.asp?filename=exercise_groupby1

EXERCISE 2

Create a new folder in the workbook-7 directory and call it aggregates-and-grouping. Add your .sql scripts to this folder.

In this exercise, will continue to execute queries against the Northwind database.

1. How many suppliers are there? Use a query!
2. What is the sum of all the employee's salaries?
3. What is the price of the cheapest item that Northwind sells?
4. What is the average price of items that Northwind sells?
5. What is the price of the most expensive item that Northwind sells?
6. What is the supplier ID of each supplier and the number of items they supply? You can answer this query by only looking at the Products table.
7. What is the category ID of each category and the average price of each item in the category? You can answer this query by only looking at the Products table.
8. For suppliers that provide at least 5 items to Northwind, what is the supplier ID of each supplier and the number of items they supply? You can answer this query by only looking at the Products table.
9. List the product id, product name, and inventory value (calculated by multiplying unit price by the number of units on hand). Sort the results in descending order by value. If two or more have the same value, order by product name.

Commit and push your code!

(OPTIONAL) EXERCISE 3

The sqlbolt.com web site has some nice tutorial and practice exercises that build up to more complex situations.

Anytime you finish an exercise early, go there and start working your way through the exercises.

Section 3–3

Nested Queries

Nested Queries

- Sometimes, you need to do a select to get one result and then use that result in another SELECT statement's WHERE clause

This is often called a subselect, inner query or nested query

- When you write subqueries, they must be enclosed within parentheses

They can't have ORDER BY clause

- A subquery usually has only one column in the SELECT clause

Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

Syntax

```
SELECT column1, column2, etc
FROM   table
WHERE  column_name OPERATOR (SELECT column_name
                               FROM table1
                               WHERE condition)
GROUP BY column_name
HAVING group-condition
ORDER BY column;
```

Example: Nested Query

QUERY: Which film(s) are the most expensive to replace?

APPROACH: Use the SQL `max()` function to find the largest `replacement_cost` in the `film` table and then use that maximum cost in a different query to select the film(s) that have that replacement cost.

```
SELECT film_id, title, replacement_cost
FROM film
WHERE replacement_cost = (SELECT MAX(replacement_cost)
                           FROM film);
```

RESULTS:

film_id	title	replacement_cost
34	ARABIA DOGMA	29.99
52	BALLROOM MOCKINGBIRD	29.99
... there were many, many rows that matched ...		

QUERY: Which film(s) are described as documentaries and how long do they run?

APPROACH: If we research the film_text table in the sakila database, we find it contains 3 columns named film_id, title, and description. We can run a query to find the films that have "documentary" in their descriptions. But the length of the film isn't available in film_text.

In this solution below, we keep the film_id values of the query that searches for documentaries and then use ANOTHER query against the film table to find all films in that 1st query's film_id list. Note that the where uses the keyword "in" rather than an "=" to match film_id values.

```
SELECT title, length
FROM film
WHERE film_id IN (SELECT film_id
                   FROM film_text
                   WHERE description LIKE "%documentary%");
```

RESULTS:

title	length
AFFAIR PREJUDICE	117
AFRICAN EGG	130
... many rows matched ...	

Exercises

EXERCISE 4

In this exercise, will continue to execute queries against the Northwind database.

Create a project folder named `nested-queries` in the `workbook-7` directory and add your `.sql` scripts to this folder.

1. What is the product name(s) of the most expensive products? HINT: Find the max price in a subquery and then use that value to find products whose price equals that value.
2. What is the order id, shipping name and shipping address of all orders shipped via "Federal Shipping"? HINT: Find the shipper id of "Federal Shipping" in a subquery and then use that value to find the orders that used that shipper.
3. What are the order ids of the orders that ordered "Sasquatch Ale"? HINT: Find the product id of "Sasquatch Ale" in a subquery and then use that value to find the matching orders from the `order details` table. Because the `order details` table has a space in its name, you will need to surround it with back ticks in the FROM clause.
4. What is the name of the employee that sold order 10266?
5. What is the name of the customer that bought order 10266?

Commit and push your code!

(OPTIONAL) EXERCISE 5

The sqlbolt.com web site has some nice tutorial and practice exercises that build up to more complex situations.

Anytime you finish an exercise early, go there and start working your way through the exercises.

Section 3–4

CodeWars

CodeWars Kata

- Complete these Katas for additional SQL practice

<https://www.codewars.com/kata/594633020a561e329a0000a2/sql>

<https://www.codewars.com/kata/58111670e10b53be31000108/sql>

Module 4

Querying Multiple Tables (JOINS)

Section 4–1

Querying Multiple Tables (JOINS)

Querying Multiple Tables (JOINS)

- Sometimes the information you want must be created by combining data in more than one table

In this case, SQL provides a way for us to *join* two or more tables together to get the information we want

- A join is where you take a row from one table and *join* it to a row in another table based on some condition

The condition is usually matching a foreign key to a primary key

employee

id	first name	last name	pay	pay grade id
100001	Greg	Smith	32000.00	1
100002	Cindy	Jones	49000.00	3
100003	Nick	Schwartz	41000.00	2
100004	Ken	McCaskill	38000.00	2

pay_grade

id	description	min_val	max_val
1	ES05 Pay Grade	22000.00	37000.00
2	ES10 Pay Grade	33000.00	46000.00
3	ES15 Pay Grade	39000.00	57000.00
4	ES20 Pay Grade	52000.00	75000.00

- The result set is "new" rows containing columns from all of the tables in the join
- There are several different types of JOIN operations, including:

INNER JOIN (most common)

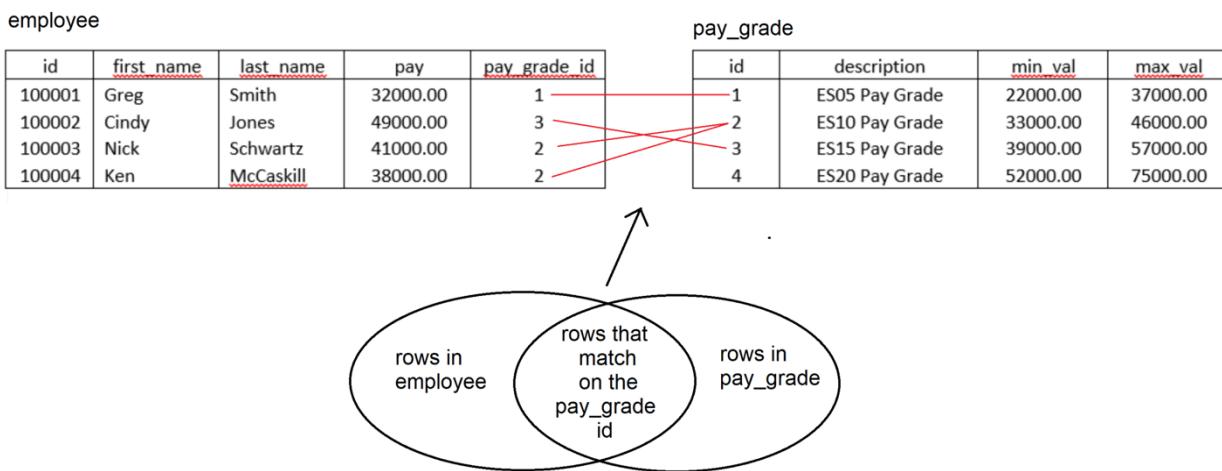
OUTER JOIN

Inner Joins

- With an INNER JOIN, a row in one table is joined with a row in another table based on a column match
- Only matched rows will be included in the result

If a row in the first table doesn't match any rows in the second table, it will be excluded

Similarly, if a row in the second table doesn't match a row in the first table, it will be excluded



- The syntax for doing an INNER JOIN is a bit more complex than a simple SELECT because you must specify both tables and the join condition

Syntax

```
SELECT column1, column2, ...
FROM table1
INNER JOIN table2
    ON table1.column-name = table2.column-name;
```

- You don't have to include the word INNER

A JOIN is considered to be an INNER JOIN unless specifically written as one of the other types you will learn about

Syntax

```
SELECT column1, columns2, ...
FROM table1
JOIN table2
ON table1.column-name = table2.column-name;
```

Example: INNER JOIN

QUERY: We want to list each employee, along with their pay_grade description and the min/max salary of that pay grade.

employee

id	first name	last name	pay	pay grade id
100001	Greg	Smith	32000.00	1
100002	Cindy	Jones	49000.00	3
100003	Nick	Schwartz	41000.00	2
100004	Ken	McCaskill	38000.00	2

pay_grade

id	description	min_val	max_val
1	ES05 Pay Grade	22000.00	37000.00
2	ES10 Pay Grade	33000.00	46000.00
3	ES15 Pay Grade	39000.00	57000.00
4	ES20 Pay Grade	52000.00	75000.00

APPROACH: Join the employee table to the pay_grade table and match employee.pay_grade_id to pay_grade.id

```
SELECT employee.id, first_name, last_name, description, min_val, max_val
FROM employee
JOIN pay_grade
ON employee.pay_grade_id = pay_grade.id;
```

RESULTS:

id	first_name	last_name	description	min_val	max_val
100001	Greg	Smith	ES05 Pay Grade	22000.00	37000.00
100002	Cindy	Jones	ES15 Pay Grade	39000.00	46000.00
100003	Nick	Schwartz	ES10 Pay Grade	33000.00	46000.00
100004	Ken	McCaskill	ES10 Pay Grade	33000.00	46000.00

We could have added the `WHERE`, `ORDER BY`, `GROUP BY`, and `HAVING` clauses if we wanted. And although this example does a JOIN on two tables, you can JOIN as many tables as you need to continue to add additional JOIN clauses.

Beyond Inner Joins

- The INNER JOIN puts data together from multiple tables based on common values in the tables.

A row is ONLY created in the result set when a row in one table and a row in the other table match a specified condition

- But this may not give us what we need
- For example, consider a commercial business

A company takes orders from customers

When an order is created for a customer, the customer id is stored in the order to identify which customer the order is for.

However, the company also allows cash orders and doesn't have a customer id for those orders

* This means the order table must allow NULLs

- If we try to join the order table to the customer table, these orders would be excluded

order			customer		
id	sold_date	customer_id	id	name	email
1	2021-05-21 10:02:00	104	101	Ezra Aiden	theater_guy@gmail.com
2	2021-05-21 11:13:45	102	102	Ian Auston	gamer05@yahoo.com
3	2021-05-21 12:06:13	NULL	103	Siddalee Grace	susa@gmail.com
4	2021-05-22 10:00:00	103	104	Elisha Aslan	gamer06@yahoo.com
5	2021-05-23 11:02:34	NULL			
6	2021-05-25 11:39:40	103			

An INNER JOIN would exclude cash customers in order where the customer_id is NULL

```
SELECT order.id, sold_date, name, email  
FROM `order`
```

```
JOIN customer
  ON order.customer_id = customer.id;
```

RESULTS:

id	sold_date	name	email
1	2021-05-21 10:02:00	Elisha Aslan	gamer06@yahoo.com
2	2021-05-21 11:13:45	Ian Auston	gamer05@yahoo.com
4	2021-05-22 10:00:00	Siddalee Grace	susa@gmail.com
6	2021-05-25 11:39:40	Siddalee Grace	susa@gmail.com

- Orders where the customer_id is NULL do NOT show, but we wanted all orders!

Outer Joins

- An **OUTER JOIN** is used when you might want rows in one table that don't match rows in the other table

You must specify which of the two tables isn't required to have matching data in its JOIN column

- You do this by creating a **LEFT OUTER JOIN** or a **RIGHT OUTER JOIN**

The first table listed in the JOIN is considered to be on the left and the second is considered to be on the right

- For a **LEFT OUTER JOIN**, the first table doesn't require matching data to be included

Selected columns without a matching value will be NULL

- For a **RIGHT OUTER JOIN**, the second table doesn't require matching data to be included

Selected columns without a matching value will be NULL

- NOTE: the word **OUTER** isn't actually required but is often included for readability

Example: Outer Joins

order

id	sold_date	customer_id
1	2021-05-21 10:02:00	104
2	2021-05-21 11:13:45	102
3	2021-05-21 12:06:13	NULL
4	2021-05-22 10:00:00	103
5	2021-05-23 11:02:34	NULL
6	2021-05-25 11:39:40	103

customer

id	name	email
101	Ezra Aiden	theater_guy@gmail.com
102	Ian Auston	gamer05@yahoo.com
103	Siddalee Grace	susa@gmail.com
104	Elisha Aslan	gamer06@yahoo.com

orders 3 and 5 are included
in a LEFT OUTER JOIN

QUERY: What orders were sold when?

```
SELECT order.id, sold_date, name, email
FROM order
LEFT JOIN customer
    ON order.customer_id = customer.id;
```

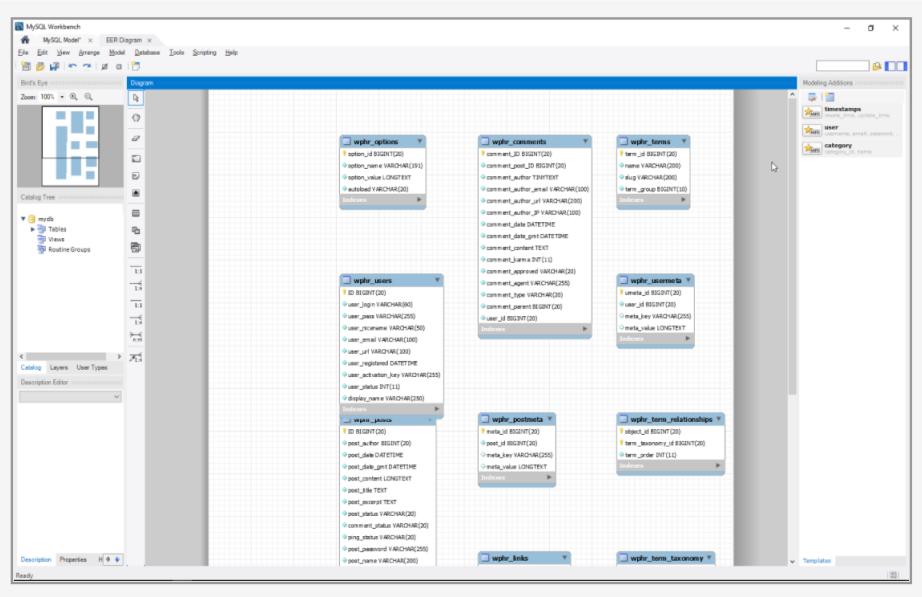
RESULTS:

id	sold_date	name	email
1	2021-05-21 10:02:00	Elisha Aslan	gamer06@yahoo.com
2	2021-05-21 11:13:45	Ian Auston	gamer05@yahoo.com
3	2021-05-21 12:06:13	NULL	NULL
4	2021-05-22 10:00:00	Siddalee Grace	susa@gmail.com
5	2021-05-23 11:02:34	NULL	NULL
6	2021-05-25 11:39:40	Siddalee Grace	susa@gmail.com

Viewing the Database

- When there are many tables involved and your queries are complex, it can be helpful to have a visual representation of the database

You can rearrange and connect the tables using the options on the left



- See the following for the process to create one:
<https://www.inmotionhosting.com/support/web-site/create-diagram-in-mysql-workbench/>

Exercises

EXERCISE 1

Let's start by checking out the exercises at W3Schools.

SQL Join: https://www.w3schools.com/sql/exercise.asp?filename=exercise_join1

EXERCISE 2

Now take a few minutes to look at this great visual diagram of the different types of joins:

<https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins>

EXERCISE 3

Create a new folder called `joins` to put your `.sql` files for this exercise.

Let's continue working with Northwind.

1. List the product id, product name, unit price and category name of all products. Order by category name and within that, by product name.
2. List the product id, product name, unit price and supplier name of all products that cost more than \$75. Order by product name.
3. List the product id, product name, unit price, category name, and supplier name of every product. Order by product name.
4. What is the product name(s) and categories of the most expensive products? HINT: Find the max price in a subquery and then use that in your more complex query that joins products with categories.
5. List the order id, ship name, ship address, and shipping company name of every order that shipped to Germany.
6. List the order id, order date, ship name, ship address of all orders that ordered "Sasquatch Ale"?

Commit and push your code!

(OPTIONAL) EXERCISE 4

The sqlbolt.com web site has some nice tutorial and practice exercises that build up to more complex situations.

Anytime you finish an exercise early, go there and start working your way through the exercises.

Section 4–2

CodeWars

CodeWars Kata

- Complete these Katas for additional SQL practice

<https://www.codewars.com/kata/5802e32dd8c944e562000020>

<https://www.codewars.com/kata/580918e24a85b05ad000010c>

<https://www.codewars.com/kata/58094559c47d323ebd000035>

Module 5

Modifying the Data and the Database

Section 5–1

Inserting, Updating and Deleting Data

Inserting, Updating and Deleting Data

- Up until now, most of the SQL we have shown has been used to query data
- However, you often need to make changes to the data in a database
- You do this using the SQL statements **INSERT**, **UPDATE**, and **DELETE**

INSERT INTO Statement

- If you want to add a new rows to a table, you use the INSERT INTO statement.
- There are two possible techniques:

In the first, you specify the table name and the values of the rows to be added

In the second, you specify the table's column names and the values of the rows to be added

* In this version, you could omit some columns and they would become NULL

Syntax

```
INSERT INTO table-name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Syntax

```
INSERT INTO table-name
VALUES (value1, value2, value3, ...);
```

- If you attempt to add a record with NULL in a non-nullable column, the INSERT statement will fail

Example: INSERT INTO

TASK: Add a new country to the sakila country table

STATEMENT (option 1):

```
INSERT INTO country(country_id, country, last_update)
VALUES(110, "Zimbabwe", NOW());
```

STATEMENT (option 2):

```
INSERT INTO country
VALUES(110, "Zimbabwe", NOW());
```

UPDATE Statement

- **Use the UPDATE statement to edit rows in a table**

It allows you to change the value in one or more columns of a row

- **Use the WHERE clause to specify which row(s) will have the changes made to them**

Syntax

```
UPDATE table-name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

- *Be careful with the UPDATE statement. If you forget the WHERE clause, all of the records in that table will be updated!*
- MySQL will generate an error if the update condition isn't based on a primary key

You can turn that off with the following code

Turn the safety setting back on again after your query

Example

```
SET SQL_SAFE_UPDATES=0;  
-- run the update w/o PK  
SET SQL_SAFE_UPDATES=1;
```

Example: UPDATE

TASK: Change the first and last name for the customer whose customer_id is 2.

```
UPDATE customer
SET first_name = 'PATTY', last_name = 'JOHNSTON'
WHERE customer_id = 2;
```

TASK: Change all PATTY first names to PATRICE.

```
SET SQL_SAFE_UPDATES=0;

UPDATE customer
SET first_name = 'PATRICE'
WHERE first_name = 'PATTY';

SET SQL_SAFE_UPDATES=1;
```

DELETE Statement

- To delete one or more rows from a table, use the DELETE statement

Syntax

```
DELETE FROM table-name  
WHERE condition;
```

- Be careful with the UPDATE statement. If you forget the WHERE clause, all of the records in that table will be updated!
- Deleting a record is a precarious action because the row you want to delete often is referenced in many tables!

For example, if you try to delete a specific film from the film table, you will now have a reference in inventory to a film_id that doesn't exist

- NOTE: Many companies don't actually delete records... Instead they have a column whose name might be "active" or something similar

You set active to true when the row is created and mark it "deleted" by using the UPDATE statement to set active to false.

This will allow the database to maintain referential integrity between related tables.

Example: DELETE

TASK: Delete all references to the payment whose payment_id is 100

```
DELETE FROM payment  
WHERE payment_id = 100;
```

Exercises

EXERCISE 1

Let's start by checking out the exercises at W3Schools.

SQL Insert: https://www.w3schools.com/sql/exercise.asp?filename=exercise_insert1

SQL Update: https://www.w3schools.com/sql/exercise.asp?filename=exercise_update1

SQL Delete: https://www.w3schools.com/sql/exercise.asp?filename=exercise_delete1

EXERCISE 2

Create a new folder called `insert-update-delete` to hold your `.sql` scripts.

Let's continue working with Northwind.

1. Add a new supplier.
2. Add a new product provided by that supplier
3. List all products and their suppliers.
4. Raise the price of your new product by 15%.
5. List the products and prices of all products from that supplier.
6. Delete the new product.
7. Delete the new supplier.
8. List all products.
9. List all suppliers.

Commit and push your code!

(OPTIONAL) EXERCISE 3

The sqlbolt.com web site has some nice tutorial and practice exercises that build up to more complex situations.

Anytime you finish an exercise early, go there and start working your way through the exercises.

Section 5–2

Creating a Table

Creating a New Table

- You can create tables (and even databases) using SQL
- The CREATE TABLE statement is used to create a table

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ...
);
```

- A list of MySQL data types can be found here:

<https://dev.mysql.com/doc/refman/8.0/en/datatypes.html>

- You can mark a field as a primary key using PRIMARY_KEY

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ...
    PRIMARY KEY (column1)
);
```

Creating a New Table

- You can mark fields as not being nullable

Syntax

```
CREATE TABLE table_name (
    column1 datatype NOT NULL,
    column2 datatype,
    column3 datatype NOT NULL,
    ....
    PRIMARY KEY (column1)
);
```

- You can also mark fields as being an auto-increment field

This happens frequently when you don't want the user to be responsible for generating unique primary keys

- * MySQL uses the AUTO_INCREMENT keyword to specify an auto-increment field
- * It must be associated with an integer datatype

Syntax

```
CREATE TABLE table_name (
    column1 INT NOT NULL AUTO_INCREMENT,
    column2 datatype,
    column3 datatype NOT NULL,
    ....
    PRIMARY KEY (column1)
);
```

Example: CREATE TABLE

TASK: Create a new table to track advertised sales

```
CREATE TABLE advertisements (
    AdId int NOT NULL,
    Title varchar(50) NOT NULL,
    MagicCode varchar(9),
    PercentOff float NOT NULL,
    PRIMARY KEY(AdId)
);
```

TASK: Create a new table to track advertised sales but use an auto-increment key

```
CREATE TABLE advertisements (
    AdId int NOT NULL AUTO_INCREMENT,
    Title varchar(50) NOT NULL,
    MagicCode varchar(9),
    PercentOff float NOT NULL,
    PRIMARY KEY(AdId)
);
```

Other Features

- There are many SQL and MySQL features that we aren't discussing, including:
 - dropping tables
 - defining constraints
 - creating and using views
 - creating and using stored procedures
 - working with triggers
- It never hurts a developer to learn more SQL!

Using AI Tools for SQL Data Generation

- Let's look at how AI tools like Claude and ChatGPT can help with some of the repetitive tasks you'll encounter when working with MySQL later in this course.
- **Automating Tedious Database Tasks**
 - When working with databases, you'll often need sample data to test your queries and applications
 - Creating this data manually can be extremely time-consuming
 - AI tools can help generate test data while you focus on learning database concepts
- **Appropriate Uses for Database Work**
 - Generate INSERT statements for test data
 - Instead of writing dozens of similar statements, describe the pattern and let AI generate them
 - **Example:** "I need 10 INSERT statements for a student's table with random names, IDs, and GPAs"
 - Generate INSERT statements for test data
 - Create sample schemas with constraints
 - Describe the entities and relationships you need, and get DDL statements to create them
 - Always review the generated SQL to understand how constraints and relationships are established

- **Format and clean up existing SQL**

- If you have working but messy SQL, AI can help make it more readable

- **Example: Generating Test Data**

I'm creating a books database for my Java application. Can you generate 5 INSERT statements for a 'books' table with columns for id (INT), title (VARCHAR), author (VARCHAR), publication_year (INT), and price (DECIMAL)?"

Reasoned about SQL INSERT statements for a few seconds >

Below are five sample SQL INSERT statements for a `books` table with the columns `id`, `title`, `author`, `publication_year`, and `price`:

```
sql Copy  
  
INSERT INTO books (id, title, author, publication_year, price)  
VALUES (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, 10.99);  
  
INSERT INTO books (id, title, author, publication_year, price)  
VALUES (2, '1984', 'George Orwell', 1949, 9.99);  
  
INSERT INTO books (id, title, author, publication_year, price)  
VALUES (3, 'To Kill a Mockingbird', 'Harper Lee', 1960, 12.99);  
  
INSERT INTO books (id, title, author, publication_year, price)  
VALUES (4, 'Pride and Prejudice', 'Jane Austen', 1813, 8.99);  
  
INSERT INTO books (id, title, author, publication_year, price)  
VALUES (5, 'The Catcher in the Rye', 'J.D. Salinger', 1951, 11.50);
```

These statements assume that the `books` table already exists in your database with the specified columns and appropriate data types. Adjust the values as needed for your application.



- **Important Guidelines:**

- Always understand the SQL you're using
- Don't just copy-paste generated SQL without reviewing it
- Make sure you understand how each statement works

- **Focus on learning database principles**

- Using AI to generate test data is fine
- Using AI to solve your database design assignments is not

- **Keep your data relevant**

- When asking for sample data, specify realistic constraints
- **Example:** "Generate customer data with ages between 18-80"

- **Remember**

- The goal is to save time on mechanical tasks, not to avoid learning
- As you gain experience, you'll likely write scripts to generate test data
- Using AI for test data generation is similar to using other tools like data generators or scripts
- Your instructor's guidelines always take precedence - if in doubt, ask!

Section 5–3

CodeWars

CodeWars Kata

- Complete these Katas for additional SQL practice

<https://www.codewars.com/kata/58111f4ee10b5301a7000175>

<https://www.codewars.com/kata/5811597e9d278beb04000038>

