**SQL Operations for Database Administration**

**CSC 210 Database Fundamentals**

**By Matthew Dunaway**

**Introduction**

The purpose of this report is to define operations and describe procedures needed to administer your own SQL database. First, I will describe the fundamentals of data and databases. Next, I will walk through the steps I took to understand which database was most useful for my needs. During this, I will cover the pros and cons of other database management systems. I will cover the setup and installation of MySQL and the basics of creating your first database. Database creation will cover database types, some properties inherent to each, and when is the best scenario to use them. Also in this section, I will talk about tables and why setting them up correctly is important to the integrity and longevity of your database. A large portion of this project will then be devoted to defining the steps of how to insert, update, delete, and retrieve data from your database. Finally, I will talk about deleting an entire table or database.

**Fundamentals of Databases**

We start with the basic concept of **data**. What is data and how is it used? The Oxford Dictionary says data is an unorganized set of facts used for reference or analysis. Data by itself does not contain any inherent meaning. Only when it is organized collectively or referred to as a group does it translate to **information**. Information that is understood and relatable to another context as significant would then be deemed **knowledge**. (University, 2021) Knowledge then applied in some form of action is then considered **wisdom**. (Sirma, 2021) Throughout this process, we start with data and add value to it through asking and answering questions. How is the raw data related to other raw data? What questions can I answer with the organized data? How can that answer to my future actions to get better results? Through this logical progression, we can see that the collection and storage of data is the single most important task advanced civilizations perform to facilitate a deeper understanding of the world we live in.

Data is stored and recalled in an organized representation called a **database**. A database can be stored in written format, in your memory, or on a computer. (Dane, 2018) Also, a database can contain various types of data. Stored data can be letters, numbers, whole sentences, and even pictures. A simple example would be a printed phonebook with names and numbers. A more complex example is your Facebook friend list organized by pictures, comments, advertisements, and their friends. For this reason, there are two types of database structures: flat file and structured databases. **Flat file databases** store text files in each line with delimiters (commas, semicolons) being the only way to separate data fields. **Relational databases** contain tables, made up of rows and columns, that organize data based upon inherent characteristics.

**Database Management Systems**

**Database management systems** are programs or software used for administering databases. Examples of these are MySQL, SQL Server, Mondo DB, and Maria DB. They serve as the link between the user and the database server. DBMS provide the platform for data input, retrieval, and display. DBMS also define, format, and structure the data that is stored based on the database model it is supporting. There are four types of database management systems: hierarchical, network, object oriented and non-relational. They differ based upon how the database is related or structured. (Invest, 2021)

**Hierarchical databases** use a ranking order to organize stored data forming a one-way relationship, or parent-child relationship. This is the folder model or tree structure model that has been used in data storage since the 1960’s. Child folders have only one parent, but parent folders can have many children. It has been very effective because of its ease of understanding and has seen widespread use across all academic, business, and enterprise organizations. Hierarchical models have serious drawbacks, nonetheless. To retrieve a child record, one must dig down into each cascading folder within the tree. Flexibility also becomes a structural issue with these types of databases. When a new table is added to the middle of the folder hierarchy, the entire database may need to be restructured. (Gilfillan, 2020) An example of this drawback would be a business that is shipping multiple orders with similar products in each order. If order A and order B has the same product, a database cannot represent the relationship accurately.

**Network databases** use a ranking order but allow for two-way relationships to be represented between multiple parent-child records. It was created to solve the shortcomings of the hierarchical database model’s inflexibility. (Gilfillan, 2015) Child records are called Members and parental records are called Owners. This allows the records to accurately reflect the relationship between similar products, in multiple orders. This model also allows for more fluid collaboration between multiple software development teams working on multiple projects in different departments. The downside of this database type is that the programmer must understand the network’s database core structure to implement it efficiently. This leads to more complex algorithms being written to create produce accurate query results. One limitation of this database type is that declarative queries cannot be used. (Preet, 2020)

**Object-oriented databases** organize data as a person would describe the aspects of an object in the real world. (Preet, 2020) They use object-oriented programming language, for example Smalltalk or Java; combined with using a relational organizational structure to store data. Instead of relational data being split up into tables, all related data about an object is saved together in a block. An example of this would be an 18” gear chain. The chain is the primary object. Its attributes of: steel construction, 18 inches in length, and 0.5-inch linkage gaps would be saved as specific detailed data about this object. Since this bike chain can be a replacement part for many bike models, scooters, and other machines, the database would use pointers to represent this relationship. This example highlights the ability of object-oriented databases to represent and solve real world problems. The drawbacks to this model comes to light when objects start to have too many relationships or become attribute heavy. The complexity of storing and retrieving data can also cause performance issues. (Dancuk, 2021)

**Non-relational databases** (NoSQL databases) store data in a document format, grouping data such as graphs, pictures, and paragraphs all together. This data is largely unstructured and is used to track customer, business trends and storing massive datasets in a non-tabular format for future queries. (Mongo, 2021) The advantage of using this database model is increased server performance because the data set is not highly structured. Conversely, having a lack of data structure with this database type causes problems with accurate queries and drawing conclusions from relatable attributes.

**Relational databases** (SQL databases) use similar characteristics of data to organize it into tables, columns, and rows. The table stores the data of a specific attribute of an entity. There can be multiple tables related to one entity. These tables are made of intersecting columns and rows. The columns describe the value of the data contained. The rows contain the data of an instance. When queried, the DBMS will return results in sets of rows. Relational databases are the most used because of their ability to store most data types, query complex sets of data, and the ability of multiple users to work on one data set simultaneously. Presently, SQL is the most popular database language for relational databases. (Fatima, 2021) We will be using My SQL as our database management system because of its ease of use and open-source availability.

**SQL** stands for **Structured Query Language** and is a database language used for administering databases. SQL was created by IBM in the 1970’s and was originally named SEQUEL. (Hamedani, 2019) Due to copywrite infringements it was renamed SQL. The characteristics of the language are short, declarative, English sentences meant to have a well-defined syntax, thus, easy to learn. (Elmasri and Navathe, 2020) SQL has the feature of being portable. With its ubiquitous syntax, it can perform administrative duties for both proprietary and open-source database management systems. Out of 47 statements I used for a SQL assignment, 43 were used across a majority of the most common database management systems. An example of a non-ubiquitous statement in a database query is **CONCATENATE**. MySQL uses **CONCAT()** where SQL Server uses **+** but the string placement does not change. Another great feature of SQL is its use of virtual tables for multiple views. With this, you can create a virtual table using statements to create new columns of data on a virtual table without compromising the integrity of the original table. An example of this would be trying to find the total cost of items you ordered. You have the quantity of the items ordered and the cost per item ordered in columns, but not a column for the total cost. SQL allows you to create that information virtually using the query (item\_price\*quantity) and represent that data graphically in your returned results.

**Installation and Setup of MySQL DBMS**

Before creating my database, I had to select a database management system so I could interact with it. I watched a few videos describing the various types and decided on MySQL because of its ease of use and open-source availability. I also selected it because my database was to be created based on the CSC210 SQL database assignment, which used relational tables and data. There are also several great tutorials on YouTube for beginners creating their first database. Our CSC210 Moodle page had a link for the MySQL download tutorial so that is where I started. I was directed to mysql.com/downloads and selected the MySQL Community Edition 8.0.25.0.msi because it was free and was recommended for beginners. I downloaded the MySQL Installer for Windows X86 32 Bit to match my computers operating system specifications. MySQL installer asked me for an Oracle account, but I did not have one, so I started the download without making an account. I was then prompted to choose a setup type and I chose the Developer Default because it had MySQL Server, Router, Workbench, and Visual Studio. I did not choose to run the Connector/Python. The download then started. It contained 11 products and took about 10 minutes. The subsequent installation took over 20 minutes. Product Configuration started next and included MySQL Server, MySQL Router, and Samples and Examples.

**Establishing a Connection to your SQL database system**

In the configuration stage, I used the default settings in MySQL Router to set up a connection to InnoDB servers, where my database would be stored. In Authentication Method, I set up a secure username and password. I also noted the encryption was SHA-256. After this, Final Configuration was finished. I connected to the InnoDB server cluster using my new username and password.

**Creating a Database on your System**

Once MySQL was configured and connected, I needed to create my first database. My plan was to use the data set given to us in the SQL assignment, named CSC210 Company Info. With this information, I opened MySQL Workbench tool and selected “ Create a new schema in selected server” icon. A new window will pop up requiring you to name the schema, or database. When naming the database, it is required to use lowercase letters and numbers. There can be no spaces in the name used. An underscore is substituted for spaces. My database name ended up being csc210\_company\_info. Then, it will prompt you to choose a character set used when one is not selected for inputted objects. I chose the default charset and default collation, as this was suggested in the SQL tutorial. Another window will then pop up asking you to review the SQL script about to run to create the database. It was **CREATE SCHEMA** followed by my database name. When you click execute, the SQL script operation will run. From there, you can look on the left of the MySQL Workbench window, in the Navigator tab and see your new database. The new database will look like a canister icon with a drop-down arrow beside it. If we expand the view, we can see Tables, Views, Stored Procedures, and Functions listed here. Since we have not added a table, that will be our next step.

**Creating a Table for your Database**

Next, I needed to create a table in my database. A table is like a spreadsheet with columns and rows that hold the data we are storing. I double clicked on the database I want to create a table in under the Schemas tab. Then I selected the icon “Create a new table in the active schema”. This will bring up a window asking you to name the table. The same naming rules outlined above apply as the database name, where you can only use lowercase letters with no spaces. I again chose the default settings on my Charset/Collation drop down. My table was named customers. Here, you can name your columns and decide whether they are going to contain character or numbers. You do this by choosing the storage type. The SQL script that ran to create columns was **CREATE TABLE**, followed by the column names. When I finished, there was a new drop-down icon for the table in the Schemas tab. When I looked at my columns, there was no data in them. This is represented by the **NULL** script in each cell.

**Inserting your Data**

When I was finished building my table and creating my columns, I needed to insert data into my rows. For this table, the rows represented my individual customer’s information. I selected the table icon with a lightning bolt on it under the customer’s table. This brought an interactive Result Grid. To insert the data, I selected the Form Editor underneath it that brought up cells to fill. When I was finished, I hit the Apply button on the bottom right of the tab. MySQL brought up a script window, that showed me the SQL script that was being executed. The operation was **INSERT INTO** the columns the **VALUES** put into the rows. The full SQL script is available at the end of the report. At this point, I had a complete table I could run queries and operations on.

**Updating your Data**

As you are updating the data in your tables, you must ask the question of whether you are updating a single row or multiple rows. I wanted to update a single customer record to reflect the customer 1000000003, Fun4All, was closing its location in Muncie, Indiana. The contact for this customer, Jim Jones, wanted the database to now reflect the store location in Olympia, Washington. I am going to use the **UPDATE, SET, and WHERE** statement to input the new values for the new customer information. The update indicates there is a record we are overwriting and specifies the table. In this case it is customers. Set is used to indicate the new values for the row. This will remain 1000000003 and Fun4All but 1 Sunny Place, Muncie, Indiana 42222 will be replaced with 7504 39th Avenue, Olympia, Washington 98503. If there is a value that will be replaced with no data, a **NULL** is used for the set value. If the column value is set to a certain value, use the **DEFAULT** statement. Where is used to specify the row being updated. MySQL works in safe mode and only allows updates to one row at a time. To update more than one row, you first unselect the “Safe Updates” selection on MySQL Workbench Preferences. Then, list the values on each row you are updating and specify the row numbers on the WHERE statement. Conversely, if you use a different SQL editor platform to update data, you may not have the “Safe Updates” restriction for updating multiple rows.

**Deleting your Data**

Deleting data is just as easy as updating data from a table. There are multiple variations to this operation and knowing what you are doing is critical before you execute any statement. You can delete a row, multiple rows, or an entire table. Always check the statement and the syntax of the statement you are about to execute to ensure you do not delete an entire table when you are just trying to delete a single row. In this task, I will be using the **DELETE FROM** statement with my table, customers. If I do not specify with a **WHERE** clause, I will delete my entire table. In this example, I want to delete The Toy Store from customers because Kim Howard chose another supplier. I will use **WHERE** cust\_id = 1000000005. This deletes the specified row.

**Delete a Table in your Database or an Entire Database**

Although we covered how to not accidentally delete a table in the previous section, it is necessary to purposefully outline the procedure to delete a table from a database. For this example, I created a new table named new\_table to delete and filled the columns and rows. Next, I right clicked the table and selected Drop Table. MySQL Workbench asks you to confirm the **DROP TABLE** and warns you the table will be deleted permanently. I chose yes and my table was permanently erased. Similarly, I wanted to delete the database the deleted table was in. I right clicked over it and chose DROP SCHEMA. The same warning appeared, and I chose to review the SQL script. It showed DROP DATABASE new\_schema. I chose “Execute” and it deleted my database.

**Retrieving Data/ Query Statements**

Retrieving data from your database and table is a function of the SQL query. We use statements to execute this in the MySQL Workbench query tab. First, select the table from the database you are querying by double clicking the table. This will execute the SELECT \* FROM customers. This starts your query in the customer’s table. The **SELECT** statement is used to get data from a table or database. (W3, 2021) I used SELECT to include **all (\*)** columns here, but specific queries will use the statement to select the field name(s) to call data from. **FROM** is used to indicate the table you are selecting data from. In my customers table I want to show all my customers name in a query one time. Fun4All shows up twice. To create this query, we need to use the **SELECT DISTINCT** statement with cust\_name. If I add other fields to this query though, I will return both records for Fun4All because other data in these two records are different.

The **WHERE** statement is used to filter rows using specific conditions. Most queries can be performed using syntax called **operators**. Operators are commonly math symbols and can represent and define query parameters quite accurately. Some operators of note are **=, <, >, <>, BETWEEN, LIKE, and IN**. For example, I can use a broad search query of SELECT \* FROM customers and then narrow the search using the condition of WHERE cust\_name = 'Fun4All';. This returns my two Fun4All records. Some other operators that add Boolean functionality to your query are the **AND, OR**, **NOT** operators. The AND operator can stack conditions that are true together in the WHERE clause. Conversely, the NOT operator can query for false conditions. The OR operator can find the true condition in the conditions you list.

The **ORDER BY** statement is used to order your results by ascending (**ASC**) or descending (**DESC**). Ordering can be done on characters alphabetically or numerically so organizing query data to locate trends or present lists is made easy. For this example, I created a new table called orders. The **PRIMARY KEY** is order\_num column. The primary key is used to identify which column is relational and serves as the identifier for records/ rows. I ordered my query by ascending cust\_id. I did not have to use ASC because it is implicit. If using descending order, it must be included. You can also order by multiple columns. The first ordered column will be used for ordering. If there are two or more of the same data in the first column, ordering will proceed to the next queried column requested.

When you create a table, sometimes there is no data to input into some parts of the record. The data field maybe optional or not available at that time. For situations like this, we fill these cells with **NULL** values. The NULL value is not a zero value. It is a placeholder. NULL values can be queried and updated with data. For my database, I had optional fields, such as cust\_email, that contained NULL values. If I had a list of NULL emails I needed to update, I could perform a query to retrieve and update that data. It would be SELECT cust\_id, cust\_email FROM customers WHERE cust\_email IS NULL. I would get an amended table with all the records that needed to be updated. Conversely, if I needed a list of all customers with email addresses for a new advertising campaign, I would use SELECT cust\_id, cust\_email FROM customers WHERE cust\_email **IS NOT NULL**. This would return a list of customer IDs and emails to send advertisements to.

**LIKE** is used as a condition when you are querying data that starts with, contains, or ends with certain characters or numbers. If you wanted the find every record that starts with the letter a, you would write a statement with LIKE **‘a%’**. If the data ends with a, you would use ‘%a’. LIKE **‘%a%’** would find an a in any position. In my customers table, I wanted to find any record that had Jim Jones’ email so I could check the cust\_name. In this example, I would use the statement, SELECT cust\_id, cust\_name, cust\_email FROM customers WHERE cust\_email Like 'jjones%'. You can also use LIKE with an underscore to enumerate the letters searched for. The example would look like **‘\_or%’** to find a word with o and r in the second and third position. The combinations of LIKE operators are numerous and will enable you to query any specific character or number you are looking for.

**CONCLUSION**

In conclusion, I learned the fundamentals of database theory. Data, information, knowledge, and wisdom can be organized and attained using relational databases. Relational databases can be built using a SQL database manager. I have outlined the procedures needed to install MySQL and create a database. I included steps and statements used to update, delete, and query data in the database. I included statements and operators that I commonly used in my project and assignment. There are many more I did not cover. To ensure these steps are repeatable, I have included the SQL script for the step I took in MySQL Workbench. Finally, I attempted to run a query from Visual Studio on the MySQL database. The attempt was unsuccessful due to an incompatibility of one of the Connector tools included in the MySQL software download. I will continue searching tutorials to troubleshoot this issue.

**Database SQL Script**

1. CREATE SCHEMA ‘csc210\_company\_info’;
2. CREATE TABLE ‘csc210\_company\_info’, ‘customers’ (‘cust\_id’ INT NOT NULL, ‘cust\_name’ VARCHAR(45) NULL, ‘cust\_address’ VARCHAR(45) NULL, ‘cust\_city’ VARCHAR(45) NULL cust\_state’ VARCHAR(45) NULL, ‘cust\_zip’ INT NOT NULL, ‘cust\_country’ VARCHAR(45) NULL, ‘cust\_contact’ VARCHAR(45) NULL, ‘cust\_email’ VARCHAR(45) NULL), PRIMARY KEY (‘cust\_id’, ‘cust\_zip’);
3. INSERT INTO `csc210\_company\_info`.`customers` (`cust\_id`, `cust\_name`, `cust\_address`, `cust\_city`, `cust\_state`, `cust\_zip`, `cust\_country`, `cust\_contact`, `cust\_email`) VALUES ('1000000001', 'Village Toys', '200 Maple Lane', 'Detroit', 'MI', '44444', 'USA', 'John Smith', 'sales@villagetoys.com');

INSERT INTO `csc210\_company\_info`.`customers` (`cust\_id`, `cust\_name`, `cust\_address`, `cust\_city`, `cust\_state`, `cust\_zip`, `cust\_country`, `cust\_contact`) VALUES ('1000000002', 'Kids Place', '333 South Lake Drive', 'Columbus', 'OH', '43333', 'USA', 'Michelle Green');

INSERT INTO `csc210\_company\_info`.`customers` (`cust\_id`, `cust\_name`, `cust\_address`, `cust\_city`, `cust\_state`, `cust\_zip`, `cust\_country`, `cust\_contact`, `cust\_email`) VALUES ('1000000003', 'Fun4All', '1 Sunny Place', 'Muncie', 'IN', '42222', 'USA', 'Jim Jones', 'jjones@fun4all.com');

INSERT INTO `csc210\_company\_info`.`customers` (`cust\_id`, `cust\_name`, `cust\_address`, `cust\_city`, `cust\_state`, `cust\_zip`, `cust\_country`, `cust\_contact`, `cust\_email`) VALUES ('1000000004', 'Fun4All', '829 Riverside Drive', 'Phoenix', 'AZ', '88888', 'USA', 'Denise L. Stephens', 'dstephens@fun4all.com');

INSERT INTO `csc210\_company\_info`.`customers` (`cust\_id`, `cust\_name`, `cust\_address`, `cust\_city`, `cust\_state`, `cust\_zip`, `cust\_country`, `cust\_contact`) VALUES ('1000000005', 'The Toy Store', '4545 53rd Street', 'Chicago', 'Il', '54545', 'USA', 'Kim Howard');

1. UPDATE `csc210\_company\_info`.`customers` SET `cust\_address` = '7504 39th Avenue', `cust\_city` = 'Olympia', `cust\_state` = 'WA', `cust\_zip` = '98503' WHERE (`cust\_id` = '1000000003') and (`cust\_zip` = '42222');
2. DELETE FROM customers WHERE cust\_id = 1000000005
3. CREATE TABLE `csc210\_company\_info`.`new\_table` (`idnew\_table` INT NOT NULL, `new\_tablecol` VARCHAR(45) NOT NULL, `new\_tablecol1` VARCHAR(45) NOT NULL, PRIMARY KEY (`idnew\_table`, `new\_tablecol`, `new\_tablecol1`));
4. INSERT INTO `csc210\_company\_info`.`new\_table` (`idnew\_table`, `new\_tablecol`, `new\_tablecol1`) VALUES ('1', 'A', '!');

INSERT INTO `csc210\_company\_info`.`new\_table` (`idnew\_table`, `new\_tablecol`, `new\_tablecol1`) VALUES ('2', 'B', '@');

INSERT INTO `csc210\_company\_info`.`new\_table` (`idnew\_table`, `new\_tablecol`, `new\_tablecol1`) VALUES ('3', 'C', '#');

1. DROP TABLE `csc210\_company\_info`.`new\_table`
2. DROP DATABASE `new\_schema`;
3. SELECT \* FROM csc210\_company\_info.customers;
4. SELECT cust\_name, cust\_city, cust\_email FROM customers;
5. SELECT DISTINCT cust\_name FROM customers;
6. SELECT \* FROM customers WHERE cust\_name = 'Fun4All'
7. CREATE TABLE `orders` ( `order\_num` int NOT NULL, `order\_date` varchar(45) NOT NULL, `cust\_id` varchar(45) NOT NULL, PRIMARY KEY (`order\_num`) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci COMMENT='Add columns'
8. INSERT INTO `csc210\_company\_info`.`orders` (`order\_num`, `order\_date`, `cust\_id`) VALUES ('20009', '2012-02-08', '1000000001')
9. SELECT order\_num, cust\_id FROM orders ORDER BY order\_num, cust\_id;
10. SELECT order\_num, cust\_id FROM orders ORDER BY order\_num DESC, cust\_id;
11. SELECT cust\_id, cust\_email FROM customers WHERE cust\_email IS NULL;
12. SELECT cust\_id, cust\_email FROM customers WHERE cust\_email IS NOT NULL;
13. SELECT cust\_id, cust\_name, cust\_email FROM customers WHERE cust\_email Like 'jjones%';

**References**

1. University, O., 2021. data noun - Definition, pictures, pronunciation, and usage notes | Oxford Advanced American Dictionary at OxfordLearnersDictionaries.com. [online] <https://www.oxfordlearnersdictionaries.com/us/definition/american_english/data>
2. Elmasri, R. and Navathe, S., 2020. Fundamentals of database systems. 7th ed. Pearson.
3. Hamedani, M., 2019. MySQL Tutorial for Beginners. [online] Youtube.com. <https://www.youtube.com/watch?v=7S_tz1z_5bA>
4. Dane, M., 2018. SQL Tutorial. [online] Youtube.com. Available at: <https://www.youtube.com/watch?v=HXV3zeQKqGY>
5. Fatima, N., 2021. A Quick Overview of Different Types of Databases. [online] Astera.com. Available at: <https://www.astera.com/type/blog/a-quick-overview-of-different-types-of-databases/>
6. Preet, K., 2020. Basic Object Oriented Data Model. [online] GeeksforGeeks.com. Available at: <https://www.geeksforgeeks.org/basic-object-oriented-data-model/#:~:text=Object%20Oriented%20Data%20Model%20represents%20the%20real%20world%20problems%20easily.&text=An%20object%20is%20an%20abstraction,implementation%20details%20from%20the%20user.>
7. Mongo, D., 2021. What Is A Non-Relational Database?. [online] MongoDB.com. <https://www.mongodb.com/non-relational-database>
8. Invest, N. (2021). Benefits of databases. <https://www.nibusinessinfo.co.uk/content/benefits-databases>
9. Sirma, A. (2021). What is the Data Information Knowledge Wisdom Pyramid? | Ontotext. <https://www.ontotext.com/knowledgehub/fundamentals/dikw-pyramid/>
10. Gilfillan, I. (2020). Understanding the Hierarchical Database Model. <https://mariadb.com/kb/en/understanding-the-hierarchical-database-model/>
11. Gilfillan, I. (2015). Understanding the Network Database Model - Source. <https://mariadb.com/kb/en/understanding-the-network-database-model/+source/>
12. Preet, K. (2020). Difference between Network and Relational data model - GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-network-and-relational-data-model/>
13. Dancuk, M. (2021). What Is an Object-Oriented Database {Concepts, Examples, Pros and Cons}. <https://phoenixnap.com/kb/object-oriented-database#:~:text=Object%2Doriented%20databases%20are%20a,creating%20more%20manageable%20code%20bases>
14. W3, S., 2021. SQL Tutorial. [online] W3schools.com. <https://www.w3schools.com/sql/default.asp>