**Databases and SQL for Data Science with Python**

# Getting started with SQL

**What is SQL?**

SQL is a powerful language that's used for communicating with databases.

Every application that manipulates any kind of data needs to store that data somewhere; whether it's big data, or just a table with a few simple rows for government, or a small startup, or a big database that spans over multiple servers or a mobile phone that runs its own small database.

SQL stands for Structured Query language (SQL) pronounced as "S-Q-L" or sometimes as "See-Quel". It helps in communicating with Relational Databases and perform tasks such as:

* Insert data on a database
* Search & retrieve data from a database
* Update data on a database

**Why SQL?**

According to Dataquest blog:

1. SQL is the most universal and commonly used database language.
2. It’s in high demand because so many companies use it.
3. SQL is still the most popular language for data work in 2021.

**What is Data and What is a Database?**

Data is one of the most critical assets of any business. It is used and collected practically everywhere. Data is a collection of facts in the form of words, numbers, or even pictures.

Your bank stores data about you, your name, address, phone number, account number et cetera. Your credit card company and your paypal accounts also store data about you. Data is important; so, it needs to be secure, and it needs to be stored and accessed quickly.

Database is a program that stores data. It is a repository of data, where data can be secured and accessed quickly. A database also provides the functionality for adding, modifying, and querying that data.

There are different kinds of databases of different requirements. The data can be stored in various forms.

In a relational database, you can form relationships between tables.

A diagram of a database

Description automatically generated

In a relational database, data is stored in a tabular form. The data is organized in tables like in a spreadsheet, which is columns and rows.

**DBMS and RDBMS**

A set of software tools for the data in the database is called a database management system or DBMS for short.  
  
RDBMS refers to relational database management system. RDBMS serves as the backbone of applications in many industries including banking, transportation, health, and so on.

**Basic SQL Commands**

In SQL, the statement is called a query. The output we get from executing this query is called a result set or a result table. Conditions are called predicates.

There are five simple commands in SQL. Please click below to know more about them.

Create: Create Table statement is used to create a new table in database

Insert: Insert data to populate the table

Select: Select data from the table

Update: Update data in the table

Delete: Delete data from the table

**Comparison operators supported by RDBMS**

When running complex queries at a later point in the course, we will make use of some operators. Below are the operators that are supported by RDBMS.

|  |  |
| --- | --- |
| Equal to | = |
| Greater than | > |
| Lesser than | < |
| Greater than or equal to | >= |
| Lesser than or equal to | <= |
| Not Equal | <> |

In the following blocks, we will learn the syntax of all the 5 basic commands of SQL that we introduced ourselves to above.

**Lets practice our Learning**

In the past lesson, we introduced ourselves to SQL and how it is a powerful tool when it comes to data management and alteration. We also got to know of the five basic commands that help us deal with data using SQL.

In this lesson, we will apply our learning by solving a set of problems where you will write the code for each of the problems.

**How to?**

You can run the codes using Google Colab or can download the DB browser for SQL. The instructional guidelines are attached below.

The datasheets and the activity sheets required for the SQL basics exercises are attached below:

A screenshot of a computer

Description automatically generated

The datasheets and the activity sheets required for the Where clause exercises are attached below:

A screenshot of a login box

Description automatically generated

# Introduction to Relational Databases and Tables

**Entity relationship (ER) data model**

An entity relationship model, also called an entity-relationship (ER) diagram, is a graphical representation of entities and their relationships to each other. ERD that represents entities called tables and their relationships.

The building blocks of an ER diagram are entities and attributes.

Suppose we design NPower Canada’s database. In this database, the participants will be an entity with attributes like id, name, age, etc.

A diagram of a group of people

Description automatically generated

**Relationships between tables**

Relationships are meaningful associations between tables that contain related information — they’re what make databases useful. Without some connection between tables in a database, you may as well be working with disparate spreadsheet files rather than a database system.

Identifying the connections you’ll need between tables is part of the data modeling and schema design process — that is, the process of figuring out how your data fits together, and how exactly you should configure your tables and their fields. This process often involves creating a visual representation of tables and their relationships, known an entity relationship diagram (ERD), with different notations specifying the kinds of relationships. Those relationships between your tables can be:

* One-to-one
* One-to-many
* Many-to-many

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

If you have understood how these relationships work, lets try building ERDs by yourself. Attached below is a model activity. Review the database scenarios, build the ER models and convert it into tables. You can use any drawing tools, MS Word, MIRO etc to create these diagrams. The diagrams will guide you to create the tables as a next step. Try to answer the following:

* What are the relationships between the tables?

A white rectangle with black text

Description automatically generated

**Primary Key**

Primary key helps uniquely identify a record in the table. The primary key of a relational table uniquely identifies each tuple or row in a table, preventing duplication of data and providing a way of defining relationships between tables.

In the retail store example in the adjacent image, what should be the primary key for each table?

In the NPower Canada example in the image below, what should be the primary key for each table?

A diagram of a diagram

Description automatically generated

A diagram of a course

Description automatically generated

**Databases on a Cloud or Cloud Databases**

A cloud database is a database that is deployed, delivered, and accessed in the cloud. Cloud databases organize and store structured, unstructured, and semi-structured data just like traditional on-premises databases.

The advantages of using cloud databases are a many, but the primary ones being:

* Ease of use and access
* Scalability & Economics
* Disaster recovery

Relational Databases on Cloud:

Relational cloud databases consist of one or more tables of columns and rows and allow you to organize data in predefined relationships to understand how data is logically related. These databases typically use a fixed data schema, and you can use structured query language (SQL) to query and manipulate data. They are highly consistent, reliable, and best suited to dealing with large amounts of structured data. Some examples of relational cloud databases include:

* IBM Db2 on Cloud
* Databases for PostgreSQL on IBM Cloud
* Oracle Database Cloud Service
* Microsoft Azure SQL Database
* Amazon Relational Database Services, and more!

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Intermediate SQL

**Select Statement**

Do you remember the syntax of the Select Statement?

SELECT CustomerName, City FROM Customers;

SELECT \* FROM Customers

WHERE Country='Mexico';

Notice the Where clause above, it always requires a predicate; which we know is a condition that evaluates to true, false or unknown. But what if we dont know exactly what value the predicate is? This is where the LIKE operator comes in play. Is helps Where clause to search for a pattern in a column. The syntax is as follows:

SELECT column1, column2, ...  
FROM table\_name  
WHERE column LIKE pattern;

**LIKE operators with '%' and '\_' wildcards**

The percent sign is used to define missing letters. The percent sign is called a wildcard character.

|  |  |
| --- | --- |
| LIKE Operator | Description |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with 'a' |
| WHERE CustomerName Like '%a' | Finds any values that ends with 'a' |
| WHERE CustomerName Like '%0r%' | Finds any values that have 'or' in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_% | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

Just the way we used 'Like' Operator, we can also use other operators.

A screenshot of a computer code

Description automatically generated

**Sorting Results Set**

When you handle data using Excel, there is a built-in feature to sort the data that can help to

* display the result in alphabetical order
* display the result in descending  order
* display the result in descending  order for a specific column

But how can you achieve the same results when you use SQL? It is achieved with the ORDER BY Keyword.

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword. Look at the examples below and the syntax that follows:

* How to display the result in alphabetical order?
  + Select title from Book ORDER BY title
* How to display the result in descending  order?
  + Select title from Book Order BY title DESC
* How to display the result in descending  order for a specific column?
  + Select title from Book Order BY 2 DESC

**Grouping Results Set**

When handling data, there are instances when you want to eliminate duplicates from a result set and describe how to further restrict a result set. This is achieved by the GROUP BY statement.

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns. The syntax is as follows:

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)  
ORDER BY column\_name(s);

**Sub-Queries and Nested Selects**

A subquery is a SQL query nested inside a larger query.

* A subquery may occur in :
  + - A SELECT clause
  + - A FROM clause
  + - A WHERE clause

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery. A subquery is usually added within the WHERE Clause of another SQL SELECT statement.

You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL. A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select. The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

A nested SELECT is a query within a query, i.e. when you have a SELECT statement within the main SELECT.

You can insert a nested SELECT within the WHERE clause with comparison operators or the IN, NOT IN, ANY, or ALL operators. The second group of operators are used when your subquery returns a list of values.

* The IN operator checks if a certain value is in the table returned by the subquery.
* The NOT IN operator filters out the rows corresponding to the values not present in that table returned by a subquery.
* The ANY operator is used with comparison operators to evaluate if any of the values returned by the subquery satisfy the condition.
* The ALL operator is also used with comparison operators to evaluate if all values returned by the subquery satisfy the condition.
* **Lets practice our Learning**
* In the past two lessons, we introduced ourselves to ERD diagrams in SQL, relational databases and how to run queries on relational databases using database on a cloud. We also learnt about how to handle data using SQL when it comes to sorting results and grouping result sets.
* In this lesson, we will apply our learning by solving a set of problems where you will write the code for each of the problems.
* **How to?**
* You can run the codes using Google Colab or can download the DB browser for SQL.
* The datasheets and the activity sheets required for the exercises on String Patterns, Sorting and Grouping Built-in database functions are attached below:

A screenshot of a computer

Description automatically generated

The datasheets and the activity sheets for the exercises using Operators are attached below:

A white rectangular object with a black border

Description automatically generated

A screenshot of a computer

Description automatically generated

The datasheets and the activity sheets for the exercises using Alter, Drop, Truncate and Where clause are as follows:

A screenshot of a computer

Description automatically generated

# Accessing Databases using Python

**Popular SQL APIs**

An application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS. The application program begins its database access with one or more API calls that connect the program to the DBMS.

Some of the popular SQL APIs are as follows:

|  |  |
| --- | --- |
| Application or Database | SQL API |
| MySQL | MySQL C API |
| PostgreSQL | psycopg2 |
| IBM DB2 | ibm\_db |
| SQL Server | dblib.API |
| Database access for Microsoft Windows OS | ODBC |
| Oracle | OCI |
| Java | JDBC |

How to pass SQL statements to the DBMS?

A diagram of a diagram of a cursor

Description automatically generated

**How to access databases using Python?**

The user writes Python programs using a Jupyter notebook. There is a mechanism by which the Python code communicates with the DBMS. The Python code connects to the database using DB-API calls.

**Connecting to a Database Using ibm\_db API**

There are three steps to connect to the Database using ibm\_db API

1. Describe the ibm-db API (Give it a unique name)
2. List the credentials required to connect to a database
3. How to connect to an IBM DB2 database using Python code written on a Jupyter notebook?

**SQL Magic**

What is SQL Magic?

Jupyter notebooks have a concept of Magic commands that can simplify working with Python, and are particularly useful for data analysis. There are 2 types of magic commands:

* Cell magics: start with a double %% sign and apply to the entire cell
* Line magics: start with a single % (percent) sign and apply to a particular line in a cell

However, there are certain limitations to their use. Unlike DB-API, there are no explicit methods to close a connection and free up resources.

Lets look at an example for using SQL Magic:

You want to execute a query to select some data from a table and fetch its results, what can you write on Jupyter notebook cell?

- %sql select \*from Tablename

**Connection Methods**

The DB\_API connect constructor creates a connection to the database and returns a Connection Object, which is then used by the various connection methods.

* cursor() method, which returns a new cursor object using the connection.
* commit() method, which is used to commit any pending transaction to the database.
* rollback() method, which causes the database to roll-back to the start of any pending transaction.
* close() method, which is used to close a database connection.

**Let's Practice our Learning**

In the past lesson, we learnt how to access databases using Python and how API connections help in establishing that access.

In this lesson, we will apply our learning by solving a set of problems where you will write the code for each of the problems.

**How to?**

You can run the codes using Google Colab or can download the DB browser for SQL.

The datasheets, the activity sheets and the Python notebook required for the exercises on DB connection and queries are attached below:

A screenshot of a computer

Description automatically generated

# SQL Tips and Tricks

There are a few guides and tutorials in this lesson which will help you navigate through different versions of SQL. With these tutorials, you have access to a cheat code that will help you establish the SQL connection with your database and how to develop Python database applications with the SQLite database.

A screenshot of a computer

Description automatically generated