Intro to SQL:

- SQL, Structured Query Language, is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL keywords are not case sensitive. E.g. select is the same as SELECT.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- SQL requires single quotes around strings.
- Below is a list of SQL commands/clauses listed in alphabetical order.

ALTER TABLE:

- The SQL ALTER TABLE command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table.
- There are many variations of the ALTER TABLE command:
    1. If we want to add a column, we use the command **ALTER TABLE table_name ADD column_name datatype;**
    2. If we want to remove a column, we use the command **ALTER TABLE table_name DROP COLUMN column_name;**
    3. If we want to change the data type of a column, we use the command **ALTER TABLE table_name MODIFY COLUMN column_name datatype;**
    4. If we want to add a not null constraint to a column, we use the command **ALTER TABLE table_name MODIFYcolumn_name datatype NOT NULL;**
    5. If we want to add a primary key constraint to a column(s), we use the command **ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);**

AVG:

- The AVG() function returns the average value of a numeric column.
- Syntax: **SELECT AVG(column_name) FROM table_name WHERE condition;**

CASE:

- The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause. If there is no ELSE part and no conditions are true, it returns NULL.
- Syntax:

```
CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN conditionN THEN resultN
        ELSE result
END;
```

COUNT:

- The COUNT() function returns the number of rows that matches a specified criteria.
- Syntax: **SELECT COUNT(column_name) FROM table_name WHERE condition;**

CREATE DATABASE:

- The CREATE DATABASE statement is used to create a new SQL database.
- Syntax: **CREATE DATABASE databasename;**

CREATE TABLE:

- The CREATE TABLE operator is used to create a new table.
- Syntax: **CREATE TABLE table_name (column1 datatype [arg1], column2 datatype [arg2], …, column(n) datatype [argn], integrity-constraint1, …, integrity-constraint(k));**
- The datatypes are the following:
    1. char(n): A fixed-length character string.
    2. varchar(n): A variable-length character string with max length n.
    3. int: An integer.
    4. numeric(p, d): A fixed point number with p digits of which d of the digits are to the right of the decimal point.
    5. real/double precision: Floating point and double precision floating point.
    6. float(n): A floating point with at least n digits of precision.
- The integrity constraints are the following:
    1. NOT NULL:
        - By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column. A NULL is not the same as no data, rather, it represents unknown data.
        - Specifies that this attribute may not have the null value. We list this constraint as an argument when defining the type of the attribute.
    2. PRIMARY KEY:
        - A primary key is a field in a table which uniquely identifies each row/record in a database table.
        - These attributes form the primary keys for the relation. Primary keys must be non-null and unique.
        - A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.
    3. FOREIGN KEY:
        - A foreign key is a key used to link two tables together. This is sometimes also called a referencing key.
        - A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
        - The values of these attributes for any tuple in the relation must correspond to values of the primary key attributes of some other tuple.
- The arguments are optional and are the following:
    1. AUTOINCREMENT:
        - Autoincrement allows a unique number to be generated automatically when a new record is inserted into a table.
    2. NOT NULL:
        - By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column. A NULL is not the same as no data, rather, it represents unknown data.
        - Specifies that this attribute may not have the null value. We list this constraint as an argument when defining the type of the attribute.

3. PRIMARY KEY:
    - You can declare a primary key by putting the words “PRIMARY KEY” beside the attribute name.

- E.g.
    1. **CREATE TABLE CUSTOMERS**
       ```
       (ID   INT                    NOT NULL,
        NAME VARCHAR (20)      NOT NULL,
        AGE  INT                    NOT NULL,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID));
       ```
    2. **CREATE TABLE department2**
       ```
       (dept_name   VARCHAR(20),
        building      VARCHAR(15),
        budget        NUMERIC(12,2),
        PRIMARY KEY (dept_name));
       ```
    3. **CREATE TABLE course**
       ```
       (course_id    VARCHAR(7),
        title           VARCHAR(50),
        dept_name   VARCHAR(20),
        credit         NUMERIC(2,0),
        PRIMARY KEY (course_id),
        FOREIGN KEY (dept_name) REFERENCES department);
       ```
    4. **CREATE TABLE course**
       ```
       (course_id    VARCHAR(7), AUTOINCREMENT PRIMARY KEY
        title           VARCHAR(50),
        dept_name   VARCHAR(20),
        credit         NUMERIC(2,0),
        FOREIGN KEY (dept_name) REFERENCES department);
       ```

CROSS JOIN:

- The CROSS JOIN operator joins every row from the first table with every row from the second table. I.e. The cross join returns a Cartesian product of rows from both tables.
- Syntax: **SELECT select_list FROM table1 CROSS JOIN table2;**

DELETE:

- The DELETE operator is used to delete existing records/tuples in a table.
- **Note:** The DELETE operator does not delete the table.
- There are 2 possible syntaxes for the delete operator:
    1. **DELETE FROM table_name;**
    2. **DELETE FROM table_name WHERE condition;**
- The first way deletes all tuples from the table where as the second way deletes the tuples that satisfy the condition.

DROP DATABASE:

- The DROP DATABASE statement is used to drop (delete) an existing SQL database.
- Syntax: **DROP DATABASE databasename;**

DROP TABLE:

- The DROP TABLE operator is used to delete an existing table.
- Syntax: **DROP TABLE table;**

EXCEPT:

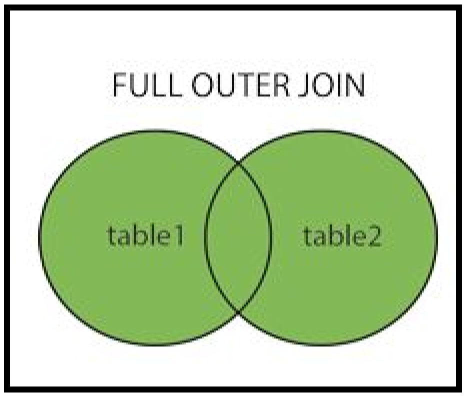
- The EXCEPT operator compares the result sets of two queries and returns the distinct rows from the first query that are not output by the second query. In other words, the EXCEPT subtracts the result set of a query from another.
- Syntax: **query1 EXCEPT query2;**

EXIST:

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns true if the subquery returns one or more records.
- Syntax: **SELECT column_name(s) FROM table_name WHERE EXISTS (SELECT column_name FROM table_name WHERE condition);**

FULL OUTER JOIN:

- The FULL OUTER JOIN keyword returns all records when there is a match in table1's or table2's table records.



- Syntax: **SELECT select_list FROM table1 FULL OUTER JOIN table2 ON join_predicate;**

GROUP BY:

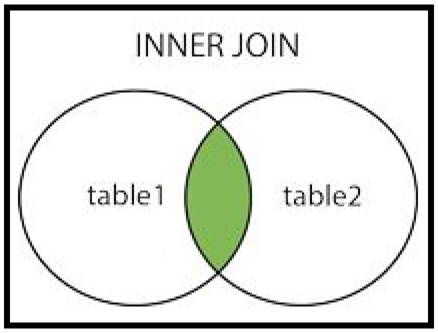
- The GROUP BY statement groups rows that have the same values into summary rows.
- The GROUP BY statement is often used with aggregate functions to group the result-set by one or more columns.
- Syntax: **SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s);**

HAVING:

- A HAVING clause in SQL specifies that an SQL SELECT statement should only return rows where aggregate values meet the specified conditions. It was added to the SQL language because the WHERE keyword could not be used with **aggregate functions**, which is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning. Examples of aggregate functions are avg, count, and sum.
- Syntax: **SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) HAVING condition**

INNER JOIN:

- The INNER JOIN keyword selects records that have matching values in both tables.



- Syntax: **select select_list from table1 inner join table2 on join_predicate;**

INSERT INTO:

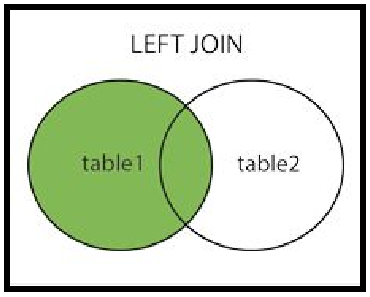
- The INSERT INTO operator is used to insert new records in a table.
- Syntax: **insert into table_name (col1, col2, …, coln) values (value1, value2, …, value(n));**
- **Note:** All primary key values will be updated automatically.

INTERSECT:

- The INTERSECT operator combines result sets of two or more queries and returns distinct rows that are output by both queries.
- Syntax: **query1 INTERSECT query2;**

LEFT JOIN:

- The LEFT JOIN keyword returns all records from table1, and the matched records from table2. The result is NULL from table2, if there is no match.
- Syntax: **SELECT select_list FROM table1 LEFT JOIN table2 ON join_predicate;**



LIMIT:

- The limit clause is used to specify the number of records to return.
- Syntax: **SELECT column_name(s) FROM table_name WHERE condition LIMIT number;**

MIN/MAX:

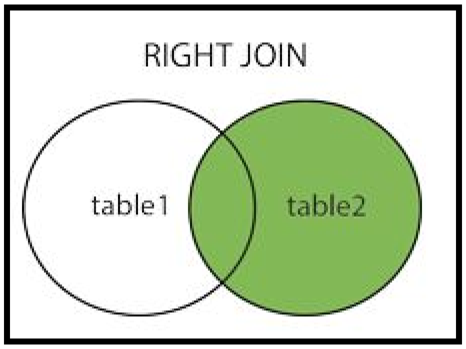
- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.
- Syntax: **SELECT MIN|MAX(column_name)FROM table_name WHERE condition;**

ORDER BY:

- 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. To sort the records in ascending order, use the ASC keyword.
- Syntax: **SELECT column1, column2, …, column(n) FROM table_name ORDER BY column1, column2, ... ASC|DESC;**
- We can also order by several columns.
- E.g. 1 The command **SELECT * FROM Customers ORDER BY Country, CustomerName;** will select all customers from the "Customers" table, sorted by the "Country" and "CustomerName" columns. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName.
- E.g. 2 The command **SELECT * FROM Customers ORDER BY Country ASC, CustomerName DESC;** will select all customers from the "Customers" table, sort the column "Country" ascending and sort the column "CustomerName" descending.

RIGHT JOIN:

- The RIGHT JOIN keyword returns all records from table2, and the matched records from table1. The result is NULL from table1, when there is no match.



- Syntax: **SELECT select_list FROM table1 RIGHT JOIN table2 ON join_predicate;**

SELECT:

- The SELECT operator is used to select data from a database. The data returned is stored in a result table, called the **result-set**.
- Syntax: **SELECT select_list FROM table_name;**
- **Note:** If you want to select all columns, you can use the following syntax: **select * from table_name;**

SELECT DISTINCT:

- The SELECT DISTINCT operator is used to return only distinct (different) values.
- Syntax: **SELECT DISTINCT select_list FROM table_name;**

SELF JOIN:

- A self join allows you to join a table to itself.
- Syntax: **SELECT select_list FROM table1 T1, table1 T2 WHERE condition;**

SUM:

- The SUM() function returns the total sum of a numeric column.
- Syntax: **SELECT SUM(column_name) FROM table_name WHERE condition;**

UNION:

- The UNION operator is used to combine the result-set of two or more SELECT statements.
    - Each SELECT statement within UNION must have the same number of columns.
    - The columns must also have similar data types.
    - The columns in each SELECT statement must also be in the same order.
- Syntax: **query1 UNION query2;**
- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.
- Syntax: **query1 UNION ALL query2;**

UPDATE:

- The UPDATE operator is used to modify the existing records in a table.
- There are 3 possible syntaxes for update:
    1. **UPDATE table_name SET column1 = value1, column2 = value2, …, column(n) = value(n)**
    2. **UPDATE table_name SET column1 = value1, column2 = value2, …, column(n) = value(n) WHERE condition;**
    3. **UPDATE table_name SET column = CASE**

```
                                    WHEN predicate1 THEN result1
                                    WHEN predicate2 THEN result2 ...
                                    WHEN predicate(n) THEN result(n)
                                    ELSE result0
                                END;
```

VIEW:

- In SQL, a view is a virtual table based on the result-set of an SQL statement. A view is a result set of a stored query. Think of it as a subset of a table or a snapshot into a table.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.
- Views have several advantages in the real world:
    - A view can hide certain columns from a table. This is useful if you want users to see certain columns but not others.
    - Can provide time savings in writing queries by having a group of frequently accessed tables joined together in a view.
    - Provide more ways to manipulate data and easily get the information you are looking for.
- Syntax: **CREATE VIEW view_name AS SELECT column1, column2, …, column(n) FROM table_name WHERE condition;**

<u>WHERE:</u>

- The WHERE clause is used to extract only those records that fulfill a specified condition.
- Syntax: **select column1, column2, …, column(n) from table_name where condition;**
- List of Operators for the WHERE clause:

Operator	**Description**
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal.
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in a SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list as per the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value. The syntax of the between statement is **SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;**
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified. The IN operator is a shorthand for multiple OR conditions.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators. There are 2 wildcards used with LIKE. % represents 0, 1 or multiple characters. _ represents a single character. The syntax of like is **SELECT column FROM table_name WHERE column LIKE pattern;**
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. This is a negate operator.
OR	The OR operator is used to combine multiple conditions in a SQL statement's WHERE clause.

IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness.

- Syntax for AND: **select column1, column2, …, column(n) from table_name where condition1 and condition2 and condition3 ...;**
- Syntax for OR: **select column1, column2, …, column(n) from table_name where condition1 or condition2 or condition3 ...;**
- Syntax for NOT: **select column1, column2, …, column(n) from table_name where not condition;**

Null Value:

- Every type can have the special value null.
- The NULL term is used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.
- A field with a NULL value is a field with no value.
  Note: A NULL value is different than a zero value or a field that contains spaces.
- You can use the IS NULL or IS NOT NULL operators to check for a NULL value.
- If we don't want null values, we can add a constraint.