Getting Started with Databases

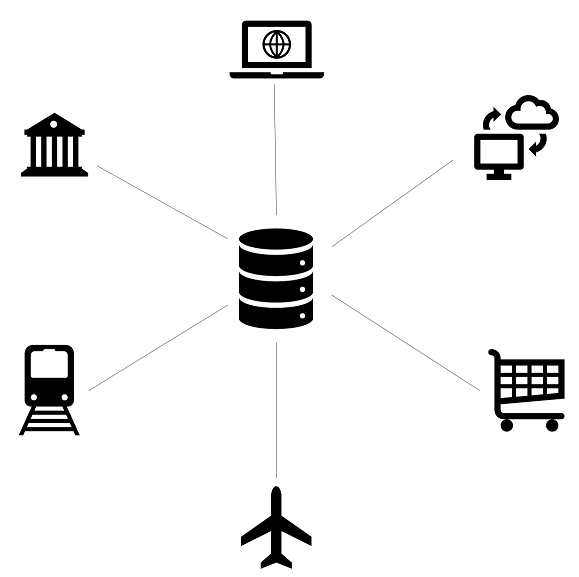
What is a database?

In simple terms, a database is a collection of data stored in a computer system. Here are other definitions of a database:  
   
According to **Wikipedia**:

A database is an organized collection of data, generally stored and accessed electronically from a computer system.

According to **Oracle**:

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

People use data and databases with or without awareness in their daily life activities.  
  
**How are databases used in the real-world?**  
****

Databases are almost everywhere. Your bank, your grocery store, an app on your cellphone, websites all use databases to keep track of your data. When you access a website, the website starts to collect your data (e.g. accessing date and time, your location, your browser info) and store it in its database.

Let's take another example. When you order a product on a commercial website, your order is stored in a database. You withdrew money from your bank account. Your bank stores this transaction in the database. Social media platforms such as Facebook, Instagram, Twitter use databases to store data like members, their friends, member activities, messages, advertisements, etc.

(Note: In diagrams, databases are represented as a cylinder shape.)

| **Applications of the databases** | |
| --- | --- |
| **Category** | **Usage** |
| Banking&Finance | Customer information, accounts, transactions |
| Education | Student information, course registrations, grades |
| Telecommunication | Internet&phone usage, subscriber information |
| Human resources | Employees, managers, salaries, hire&termination dates |
| Websites | Products, visitors, website traffic statistics |
| Transportation | Passenger, reservation and schedule information |

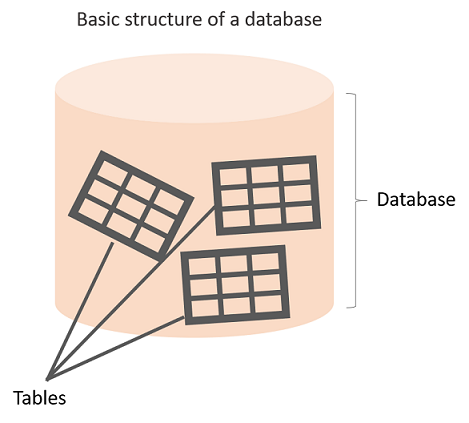
**☍ Discussion:** Think about other applications of databases. Try to give some examples.

A database is typically controlled by a **database management system (DBMS)**. Data and DBMS along with the applications that are associated with them are called a database system, often shortened to just database.  
  
Now that we've learned what the database is, we can move on to what is inside the database, how the data is stored in the database.

Q: What is a Database?  
A: A database is an organized collection of data, generally stored and accessed electronically from a computer system. In simple terms, a database is a collection of data stored in a computer system. When you order a product on a commercial website, your order is stored in a database. You withdrew money from your bank account. Your bank stores this transaction in the database. Social media platforms such as Facebook, Instagram, Twitter use databases to store data like members, their friends, member activities, messages, advertisements, etc.

## Getting Started with Databases

### What is in a database?

The information inside the database is grouped into tables. A table in a database is called a *database table*. Tables are the basic unit of data storage in databases. We talked about the definition of the database in the previous lesson. We used the term **structured data.**Structured data heremeans table. A table consists of columns and rows. You may think of it as an Excel or Google spreadsheet. Although there are similarities between the database table and Excel/Google spreadsheet, they are different things. We won't get into details here.

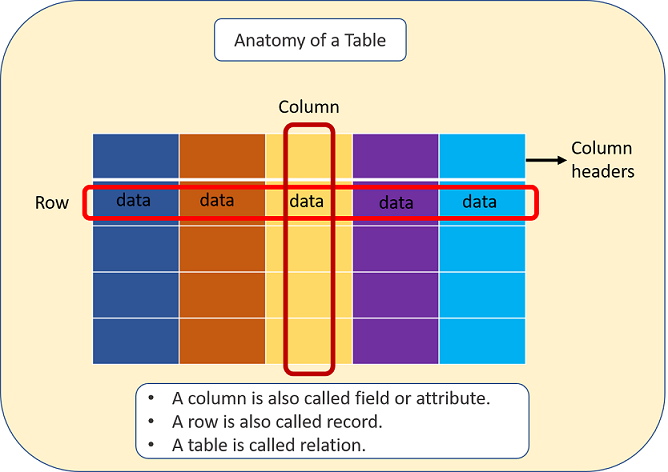
**Anatomy of a Table**

A table is made up of columns and rows. A column is a piece of data stored by the table. A row is a single set of columns that describe the attributes of a single thing. Columns should have a unique name. Columns and rows together make up a table.

**💡Tips:**

* In database world;
* A column is also called a field or attribute,
* A row is also called a record or a tuple,
* A table is also called a relation.

Column header names are written in lowercase, and there shouldn't be any space in a single name. Put an underscore between them. For instance, don't write "*last name"* as a header name, instead, write "*last\_name"*. Table names are also written in lowercase and without space (*employees*, *customers*, etc.)



A database can consist of one or more tables. In most cases, more than one table. Each table has a unique name, such as employees, departments, or customers, etc.

##### Example

Let's take the case of a company database. Suppose that it has two tables. One is employees, and other is departments. Take a closer look at the employees table below. Here is the breakdown of the table.

* Table's name is *employees.*
* The table has seven columns (aka, fields or attributes)
* The table has ten rows (aka, records or tuples)
* Table's column header names are:  emp\_id, first\_name, last\_name, salary, job\_title, gender, hire\_date
* Inside the employees' table, there is data about each employee in the company



Of course, a real-world company would have many more employees. We use a small table to illustrate concepts.

Q: What is a table, column and row?  
A: A table is an organized collection of data stored in the form of columns and rows. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

Getting Started with Databases

Type of Databases

Typically, there are two main database storage types:

* **Relational Database - SQL**
* **Non-Relational Database - NoSQL**

A *relational database* is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the *key*. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

Actually, the term "relational database" was invented by E. F. Codd at IBM in 1970. Codd introduced the term in his research paper "A Relational Model of Data for Large Shared Data Banks". In this paper and later papers, he defined what he meant by "relational". One well-known definition of what constitutes a relational database system is composed of Codd's 12 rules. However, no commercial implementations of the relational model conform to all of Codd's rules, so the term has gradually come to describe a broader class of database systems, which at a minimum:

* Present the data to the user as relations (a presentation in tabular form, i.e. as a *collection* of tables with each table consisting of a set of rows and columns);
* Provide relational operators to manipulate the data in tabular form.

A software system used to maintain relational databases is called a *Relational Database Management System (RDBMS).*  Here are some examples of RDBMS:

* Amazon Aurora
* Amazon RDS
* Microsoft SQL Server
* Oracle Database
* MySQL
* IBM DB2
* Maria DB
* PostgreSQL
* SQLite

SQL (stands for Structured Query Language) is accepted as the standard Relational Database Management System (RDBMS) language. So we usually prefer to call Relational Database as SQL and Non-Relational database as NoSQL.

Until now in the course, what we've shown you some relational database features (structured data, tables, rows, columns, etc). Of course, there are more than those features coming in the upcoming parts of the course. During the pre-class phase of the SQL course, we will use SQLite as an RDBMS tool.

**💡Tips:**

* SQLite is a relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program. (Wikipedia)

Now, it's time to communicate with our database with SQL.

Getting Started with Databases

Structured Query Language (SQL)

SQL stands for Structured Query Language [and](http://ec2-35-173-203-107.compute-1.amazonaws.com/lms/mod/lesson/view.php?id=295) used to communicate with relational databases. SQL is a declarative language, not a procedural language. You write a single SQL declaration and hand it to the DBMS. The DBMS then executes internal code, which is hidden from us.

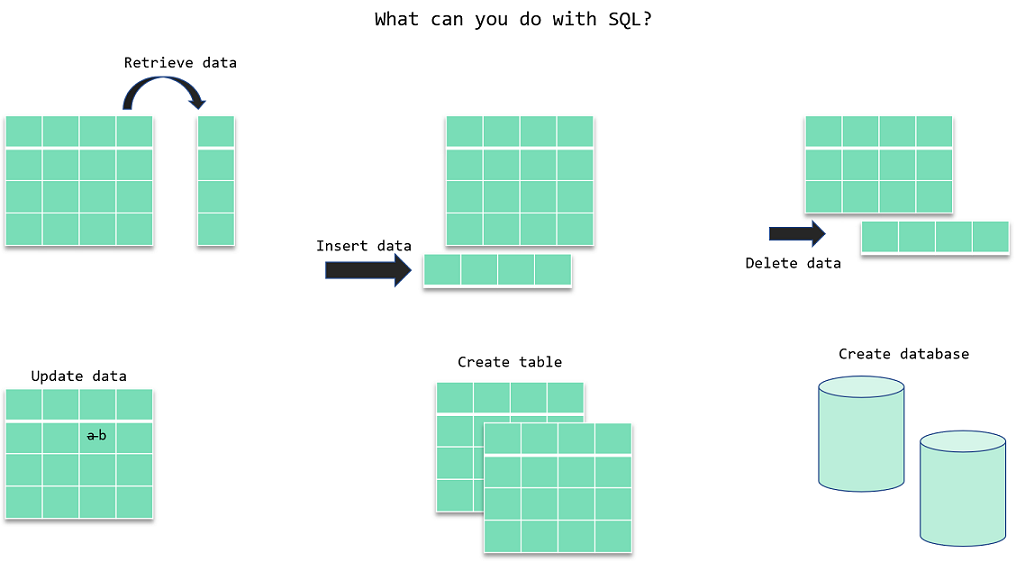
**💡Tips:**

* Declarative paradigm is where you say what you want without having to say how to do it. With procedural paradigm (used in JAVA, C), you have to specify exact steps to get the result. SQL is declarative than procedural since the queries don't specify steps to produce the result.
* SQL in it's purest form is not a programming language, but a query language. Because, it needs to be able to perform loops and control structures. However, with some extensions, SQL can have looping and control structures but they exist outside or rather as an appendage to the original SQL spec. In this manner, some argue that SQL is a programming language.

Most commercial database systems employ the SQL language.With SQL, you can access or manipulate data stored in the database. There are different types of access. These are:

* Retrieval of data from the database
* Insertion of new data into the database
* Updating the data in the database
* Deletion of data from the database

Besides, you can create new databases and tables using SQL.



In the next section, we will focus on the retrieval portion of the SQL. There is a particular word that is called *query* for retrieval information from the database. You will often encounter this word during your SQL course journey.

ⓘ A **query** is a statement asking for the retrieval of information from the database.

Since SQL is a language, it has grammar. Now, let's learn how to write in that language.

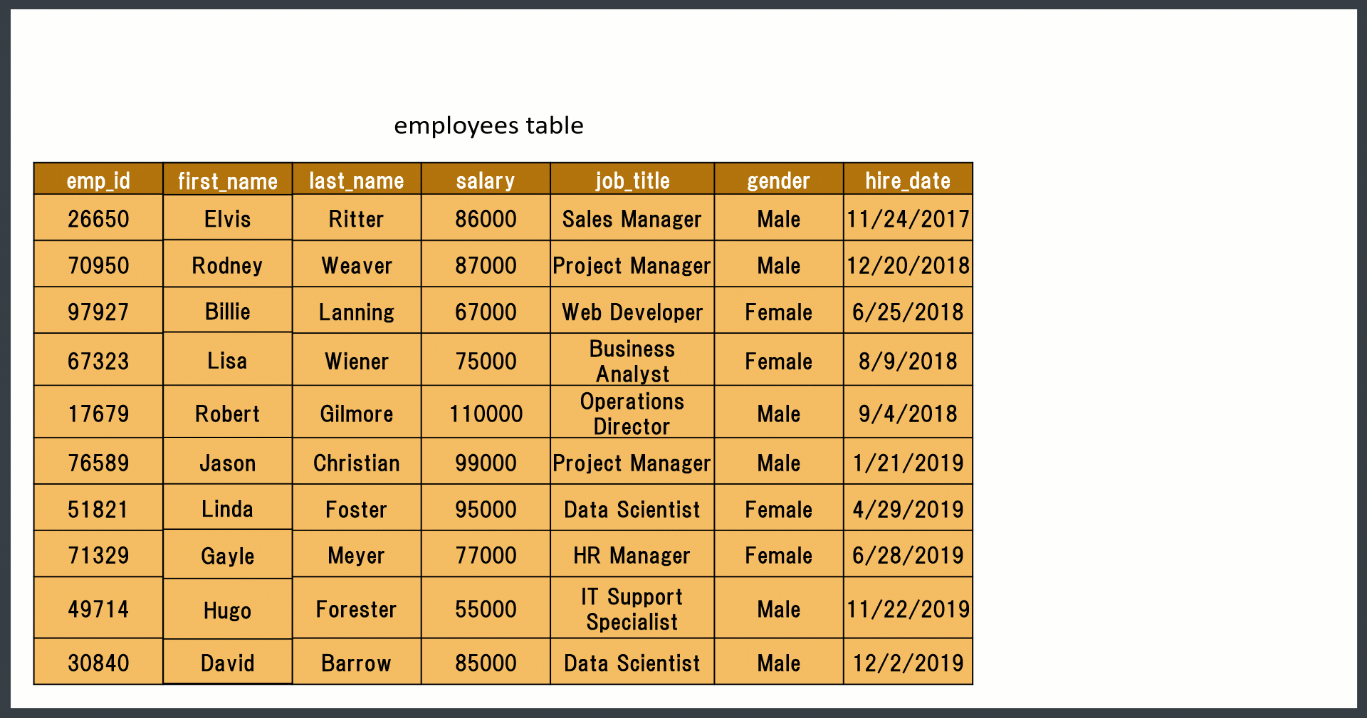
Q: What is SQL?  
A: SQL stands for Structured Query Language and used to communicate with a database. With SQL, you can access or manipulate data stored in the database.

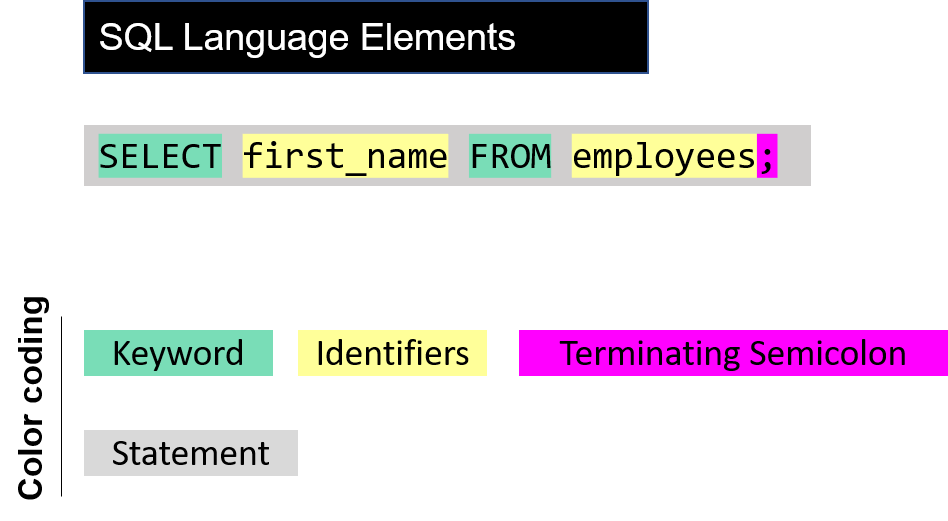
Getting Started with Databases

SQL Language Elements

SQL language structure is easy to understand. It looks like plain English. Here is an example SQL statement:

SELECT first\_name FROM employees;

If we apply this SQL statement to our *employees*table, we get a single column that is *first\_name. See the command in action below: (Note: if the below gif doesn't play fast, please refresh the page)*  
  


The whole command SELECT first\_name FROM employees; is called **statement.**Let's breakdown of this statement. Herein SELECT, FROM words are keywords. They are special commands for SQL. first\_name, employees are identifiers. SQL statements end with a semicolon (;).  SQL Language Elements is also called SQL Syntax.   
  
  
There are other SQL elements which we will cover later in the course. In the next part, we will introduce you SELECT statement in detail. We will learn the syntax and start to write our commands.

## SELECT Statement

### Introduction

In this lesson, you will focus on the SELECT statement in SQL. Generally;

* The SELECT statement is used to select data from a database.
* You can retrieve rows from the columns of the table by using this statement.
* SELECT statement is used with FROM keyword.
* The syntax of the SELECT statement can be seen below.

SELECT column\_name(s) FROM table\_name;

By using this query, you explain to SQL that you want to see the data from a column/columns in the given table. The result of the query is stored in a result table called *result-set*.

## SELECT Statement

### Basic Syntax

The term syntax in SQL refers to strict structural patterns used when creating a query. As in any other programming language, SQL also has some general syntax rules to follow. But these rules are very flexible compared with others. Let' look at the syntax of the last query again.

SELECT column\_name(s) FROM table\_name;

As you see, SELECT and FROM words were written in uppercase, and the query ends with a semicolon. Is this usage example a syntax rule? In SQL syntax;  
  
SQL statements start with a keyword like SELECT, INSERT, UPDATE, DELETE, etc. and all the statements end with a semicolon (;).

* The semicolon at the end indicates that the statement is completed and ready to be executed.
* SQL is also case insensitive, which means you can use both SELECT and select in your query. They mean the same thing for SQL.
* Writing SQL commands in the upper-case is the most common and preferred style. But, you can write the same query in both ways as below:

select column\_name(s) from table\_name;

SELECT COLUMN\_NAME(s) FROM TABLE\_NAME;

* White spaces and empty lines are ignored in SQL. So, the below query is correct by all means.

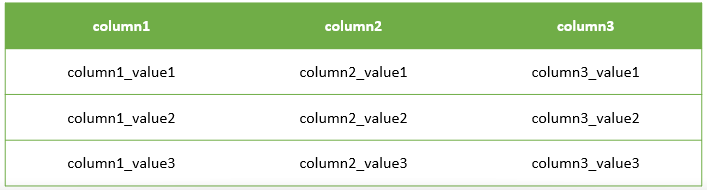
SELECT column\_name(s)

FROM table\_name;

**💡 Tip:** To maintain your query clean and more readable, it's not recommended to use unnecessary empty lines and white spaces .

## SELECT Statement

### Selecting Multiple Columns



 Assume that;

* You have a table named as **table1** like above.
* You want to see only column1 and column2 data in it.
* Then, you should write a query like this:

query :

SELECT column1, column2 FROM table1;

After you execute this query, SQL will return you only the data that you want from column1 and column2 of table1.  
  
output :

column1 column2

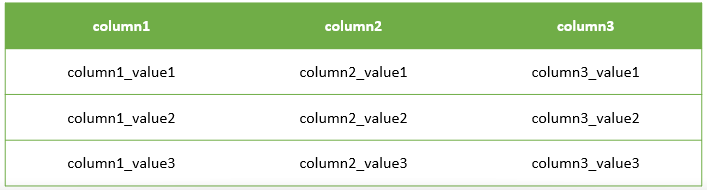
-------------- --------------

column1\_value1 column2\_value1

column1\_value2 column2\_value2

column1\_value3 column2\_value3

### Selecting All Columns



Assume that; this time you want to see all of the data in your table.

You have 3 columns in your table, so you should write a query like this.

query :

SELECT column1, column2, column3 FROM table1;

Because you wrote the names of all the columns in the SELECT statement, you will get the whole table via this query.

output :

column1 column2 column3

-------------- -------------- --------------

column1\_value1 column2\_value1 column3\_value1

column1\_value2 column2\_value2 column3\_value2

column1\_value3 column2\_value3 column3\_value3

* Ok, it's not difficult to write 3 columns in this case, but think about you have a big table with lots of columns.
* Would you have to write all the column names in your query?

**Brainstorming:** Think about a more easy way to use for getting all the data in your table. We'll talk about it in the next step.

### Selecting All Columns (Special Character)

The asterisk character “\*” has special meaning in SQL.

To retrieve all of the information from your table, an asterisk (\*) character can be used after the SELECT command.

So, you wouldn't have to type in each of the column names separately.

query :

SELECT \* FROM table1;

By using **the asterisk character “\*”**, you can get every column with all of the data in it as a result-set of the query.

output :

column1 column2 column3

-------------- -------------- --------------

column1\_value1 column2\_value1 column3\_value1

column1\_value2 column2\_value2 column3\_value2

column1\_value3 column2\_value3 column3\_value3

## DISTINCT Clause

### Introduction

In this lesson, you will focus on the SELECT DISTINCT statement in SQL. Columns in the tables may often contain some duplicate values, but you may only need the distinct values as a result. Here comes the SELECT statement with the DISTINCT clause.

**ⓘ Info:** We have learned what the statement and keyword are. Here is another term which we introduced to you in this lesson: Clause. We want to define each three terms to help you gain better understanding. We will use query SELECT first\_name, last\_name, gender FROM employees; as an example to explain the concepts.

* **Keyword:**These are the individual elements which are predefined. In the example these are SELECT and FROM seperately.
* **Clause:**It's a part of a SQL statement. In our example, these ere SELECT first name, last name, gender and FROM employees.
* **Statement:**Thecomplete query is a statement. A statement may consist of two or more clauses.

The SELECT DISTINCT is used to return only distinct (different/unique) values to eliminate duplicate rows in a result set. Here is the syntax of the DISTINCT clause:

SELECT DISTINCT column\_name(s) FROM table\_name;

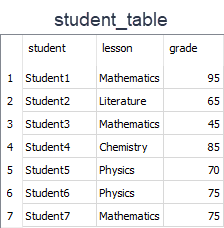
Q: What are some common clauses used with SELECT query in SQL?  
A: WHERE clause, ORDER BY clause, GROUP BY clause and HAVING clause

- Interview Q&A

By using this query, you explain to SQL that you only want to see the unique/distinct data from the column/columns in the given table.

## DISTINCT Clause

## No Duplicated Rows



Assume that;  
 You have a table named as *student\_table* like above.

* You want to see only UNIQUE Values in student\_column in the result set.
* Then, you should write a query like this.

query :

SELECT DISTINCT student FROM student\_table;

After the execution of the query, you will get a result set like below.  
output :

student

----------

Student1

Student2

Student3

Student4

Student5

Student6

Student7

As you see in the result set, the query returned all the rows in the student column. Because all the rows in that column have unique values. So, in this example, we can see that if there are no duplicated rows in a column, SELECT and SELECT DISTINCTgives the same result.

Let's write a different query to select unique lesson names from the lesson column. You should write a query like this.  
query :

SELECT DISTINCT lesson FROM student\_table;

After the execution of the query, you will get a result set like below.  
output:

lesson

-----------

Mathematics

Literature

Chemistry

Physics

There are 7 rows in the student\_table. But the result set has only 4 rows. Why?

* Lesson column consists of the following lessons; 3 Mathematics, 2 Physics, a Literature, and a Chemistry.
* So there are 3 duplicated rows of which 2 Mathematics and 1 Physics.
* DISTINCT clause eliminated 3 rows and so we get 4 unique rows.

## WHERE & LIMIT Clauses

### Introduction

In this lesson, you will focus on the WHERE clause in SQL. The WHERE clause is used to filter records.

* It allows you to define a specific search condition for the result set returned by a query.
* So, the result set only consists of the records that fulfill the predefined condition(s).

The WHERE clause is mostly used with the SELECT statement. In addition to the SELECT statement, it may also be used with some other statements like DELETE and UPDATE. It's used in a query after the FROM clause as in the below example.

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

By using this query, you explain to SQL that you only want to get the data that pass the defined condition(s) as a result set.

WHERE Clause - Operators

You can use the following operators in the WHERE clause. 

| **Operators in the WHERE Clause** | |
| --- | --- |
| **Operator** | **Description** |
| = | Equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| <> | Not equal. This operator may be written as != in some versions of SQL |
| BETWEEN | Test if a value is between a certain range of values |
| LIKE | Determine if a character string matches a predefined pattern |
| IN | Test whether or a value matches any value in a list |

**student\_table**



Assume that;

You have a table named student\_table as above.

If we want to select only the records of which grade is higher than 70 in the result set, then we should write a query like this.  
  
query :

SELECT \* FROM student\_table WHERE grade > 70

After the execution of the query, you will get a result set like below.  
output:

student lesson grade

---------- ----------- ----------

Student1 Mathematics 95

Student4 Chemistry 85

Student6 Physics 75

Student7 Mathematics 75

## WHERE & LIMIT Clauses

### Example-1



* You want to see only the records of which lesson is Mathematics in the result set.
* Then, you should write a query like this.

query :

SELECT \* FROM student\_table WHERE lesson = "Mathematics";

After the execution of the query, you will get a result set like below.

output :

student lesson grade

---------- ----------- ----------

Student1 Mathematics 95

Student3 Mathematics 45

Student7 Mathematics 75

WHERE & LIMIT Clauses

Example-2



Let's write a different query to select only the records of which grade is lower than 70 in the result set. Our  query will be as below:  
query :

SELECT \* FROM student\_table WHERE grade < 70

After the execution of the query, you will get a result set like below.  
output:

student lesson grade

---------- ---------- ----------

Student2 Literature 65

Student3 Mathematic 45

WHERE & LIMIT Clauses

LIMIT Clause

In this lesson, you will focus on the LIMIT clause in SQL. The LIMIT clause is used to filter records. It constrains the number of rows returned by a query. Assume that your query returns one thousand rows. But you only want to see the first 10 rows in the result set. In such cases, we use LIMIT clause to obtain the desired output.   
  
Here is the syntax of the LIMIT clause.

SELECT column\_name(s) FROM table\_name LIMIT number\_rows;

Let's select all the columns of the student\_table and return the first 3 rows.  
  
query:

SELECT \* FROM student\_table LIMIT 3;

output:

student lesson grade

---------- ----------- ----------

Student1 Mathematics 95

Student2 Literature 65

Student3 Mathematics 45

We can also combine LIMIT with WHERE. LIMIT clause is placed after the WHERE clause. Let's select the students whose grade is higher than 70 and let our query return the first 2 rows.  
  
query:

SELECT \* FROM student\_table WHERE grade > 70 LIMIT 2;

output:

student lesson grade

---------- ----------- ----------

Student1 Mathematics 95

Student4 Chemistry 85

ORDER BY Clause

Order By Clause

SELECT statement returns records in an unspecified order. In case you want to retrieve data in alphabetical or numeric order, we use ORDER BY keyword.

The ORDER BY keyword sorts the result-set in descending or ascending order.

By default ORDER BY keyword sorts the records in ascending order. Use the keyword DESC to sort the records in descending order. You can also use ASC to sort the data in ascending order. You have to use either of them.

Here is the syntax of ORDER BY:

SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s) ASC|DESC;

Herein **"|"** symbol means "use either ASC or DESC". If you don't use any of them, the default value is ASC (ascending order).  
Let's see ORDER BY in an action.  
  


Here is our employees table. I want to sort the *first\_name*column in alphabetical order (A-Z). This is the appropriate query:  
  
query :

SELECT \* FROM employees ORDER BY first\_name ASC;

After executing the query, we get the result table below. Our table is now sorted by the first names in ascending order. Not only the *first\_name* column is sorted, but also other columns are affected by the sort accordingly.

output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ------------- ----------

    ----------

97927 Billie Lanning 67000 Web Developer Female 6/25

    /2018

30840 David Barrow 85000 Data Scientis Male 12/2

    /2019

26650 Elvis Ritter 86000 Sales Manager Male 11/24

    /2017

71329 Gayle Meyer 77000 HR Manager Female 6/28

    /2019

49714 Hugo Forester 55000 IT Support Sp Male 11/22

    /2019

76589 Jason Christian 99000 Project Manag Male 1/21

    /2019

51821 Linda Foster 95000 Data Scientis Female 4/29

    /2019

67323 Lisa Wiener 75000 Business Anal Female 8/9/2018

17679 Robert Gilmore 110000 Operations Di Male 9/4/2018

70950 Rodney Weaver 87000 Project Manag Male 12/20

    /2018

You could write the query which returns the same result table as below. Since ASC is the default order value in case you don't specify any ascending or descending order, both queries will yield the same result.

SELECT \* FROM employees ORDER BY first\_name;

Alright. Now it's time to put your theory into practice. Try to write as many queries as you want in the coding playground.

ORDER BY Clause

Sorting in Descending Order

In the previous part, we sorted our table by the first names of the employees in ascending order. What if we asked you to sort it in descending order (Z-A)? You would probably say that "I use the **DESC** keyword." You're right. Let's write the query.query :

SELECT \* FROM employees ORDER BY first\_name DESC;

Below is the query result. Other columns are also affected.  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- --------------- ----------

    ----------

70950 Rodney Weaver 87000 Project Manager Male 12/20

    /2018

17679 Robert Gilmore 110000 Operations Dire Male 9/4

    /2018

67323 Lisa Wiener 75000 Business Analys Female 8/9

    /2018

51821 Linda Foster 95000 Data Scientist Female 4/29

    /2019

76589 Jason Christian 99000 Project Manager Male 1/21

    /2019

49714 Hugo Forester 55000 IT Support Spec Male 11/22

    /2019

71329 Gayle Meyer 77000 HR Manager Female 6/28

    /2019

26650 Elvis Ritter 86000 Sales Manager Male 11/24

    /2017

30840 David Barrow 85000 Data Scientist Male 12/2

    /2019

97927 Billie Lanning 67000 Web Developer Female 6/25

    /2018

**💡 Tip:**When you sort the data, the original table's order is not affected. Remember from the previous lessons that a query returns a result table. Thus, we sort the result table, not the original one.

Until now, we've sorted the column containing textual data. It's time to sort numerical data in our table. You may wonder whose salary is the highest. Let's write the query. This time we don't want to retrieve all columns instead we want first name, last name and salary.  
  
query :

SELECT first\_name, last\_name, salary FROM employees ORDER BY salary DESC;

output :

first\_name last\_name salary

---------- ---------- ----------

Robert Gilmore 110000

Jason Christian 99000

Linda Foster 95000

Rodney Weaver 87000

Elvis Ritter 86000

David Barrow 85000

Gayle Meyer 77000

Lisa Wiener 75000

Billie Lanning 67000

Hugo Forester 55000

Robert Gilmore, Operations Director, has the highest salary. We sorted the result table from largest to smallest.

ORDER BY Clause

Sorting By Multiple Columns

We are now able to sort by one column using the **ORDER BY**keyword. In some cases, we may need to sort our data by two columns or more. To do this, separate the columns by a comma. Here is the **syntax:**

SELECT column\_name(s) FROM table\_name ORDER BY column1 ASC|DESC, column2 ASC|DESC,

    columnN ASC|DESC;



Above is our original table. Let's sort it by gender in descending order.

query :

SELECT \* FROM employees ORDER BY gender DESC;

Here is the result-set:  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ------------------- ----------

    ----------

17679 Robert Gilmore 110000 Operations Director Male 9

    /4/2018

26650 Elvis Ritter 86000 Sales Manager Male 11

    /24/2017

30840 David Barrow 85000 Data Scientist Male 12

    /2/2019

49714 Hugo Forester 55000 IT Support Speciali Male 11

    /22/2019

70950 Rodney Weaver 87000 Project Manager Male 12

    /20/2018

76589 Jason Christian 99000 Project Manager Male 1

    /21/2019

51821 Linda Foster 95000 Data Scientist Female 4

    /29/2019

67323 Lisa Wiener 75000 Business Analyst Female 8

    /9/2018

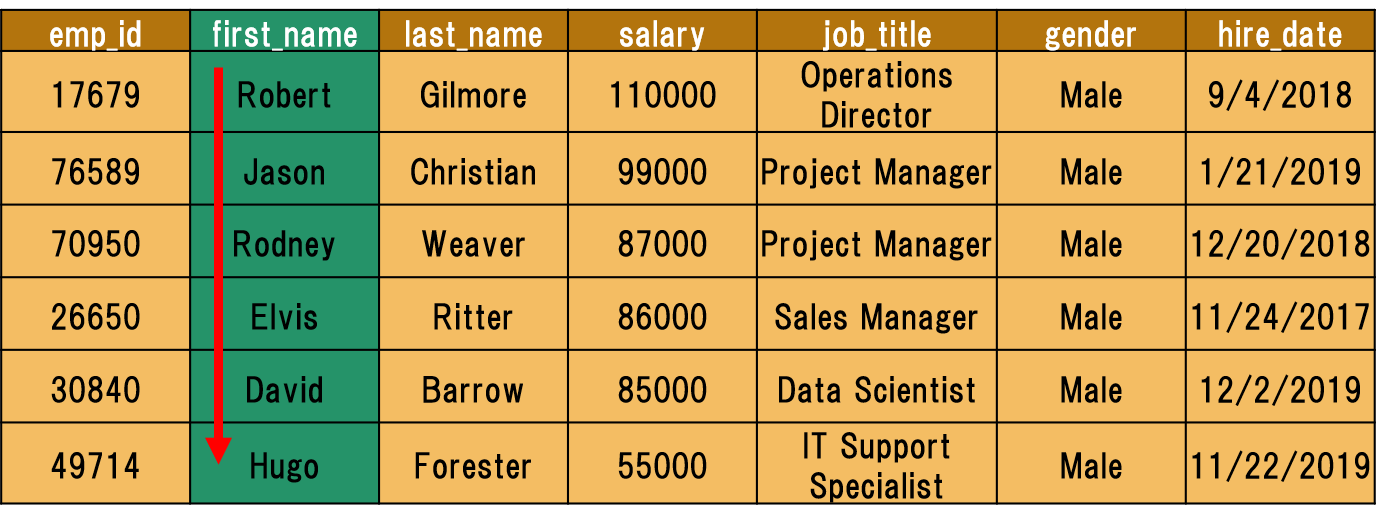
71329 Gayle Meyer 77000 HR Manager Female 6

    /28/2019

97927 Billie Lanning 67000 Web Developer Female 6

    /25/2018

In the above output, you can see that the result table is sorted in descending order according to the gender of the employees. Suppose that we also want to sort it by first names.



In this case, we have to append the first\_name column to our existing query. Use a comma to add the new column.  
  
query :

SELECT \* FROM employees ORDER BY gender DESC, first\_name ASC;

Here is the output:  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- -------------- ----------

    ----------

30840 David Barrow 85000 Data Scientist Male 12/2

    /2019

26650 Elvis Ritter 86000 Sales Manager Male 11/24

    /2017

49714 Hugo Forester 55000 IT Support Spe Male 11/22

    /2019

76589 Jason Christian 99000 Project Manage Male 1/21

    /2019

17679 Robert Gilmore 110000 Operations Dir Male 9/4

    /2018

70950 Rodney Weaver 87000 Project Manage Male 12/20

    /2018

97927 Billie Lanning 67000 Web Developer Female 6/25

    /2018

71329 Gayle Meyer 77000 HR Manager Female 6/28

    /2019

51821 Linda Foster 95000 Data Scientist Female 4/29

    /2019

67323 Lisa Wiener 75000 Business Analy Female 8/9

    /2018

In the above example, we first sorted the data by gender, then we sorted by first names.

## ORDER BY Clause

### ORDER BY Clause with WHERE Clause

In this part, we will use ORDER BY with the WHERE clause.

This is the syntax:

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

    ASC|DESC;

ORDER BY clause is placed after the WHERE clause.

Technically, any SQL statement can be written on a single line. However, it will become difficult to read when you start to write long queries. The solution in such cases is to organize the code, not just horizontally, but also vertically. This is called **beautifying.**Let's rewrite the syntax above.

SELECT column\_name(s)

FROM table\_name

WHERE condition

ORDER BY column\_name(s)s ASC|DESC;

As you see that we put each clause on a separate line. Now let's continue with an example.



Assume that we try to find the employees whose salary is higher than $80,000. Next, we will sort it by first\_name in descending order.

Here is the query:

query :

SELECT \*

FROM employees

WHERE salary > 80000

ORDER BY first\_name DESC;

This is our result table:

output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- --------------- ----------

    ----------

70950 Rodney Weaver 87000 Project Manager Male 2018-12

    -20

17679 Robert Gilmore 110000 Operations Dire Male 2018-09

    -04

51821 Linda Foster 95000 Data Scientist Female 2019-04

    -29

76589 Jason Christian 99000 Project Manager Male 2019-01

    -21

26650 Elvis Ritter 86000 Sales Manager Male 2017-11

    -24

30840 David Barrow 85000 Data Scientist Male 2019-12

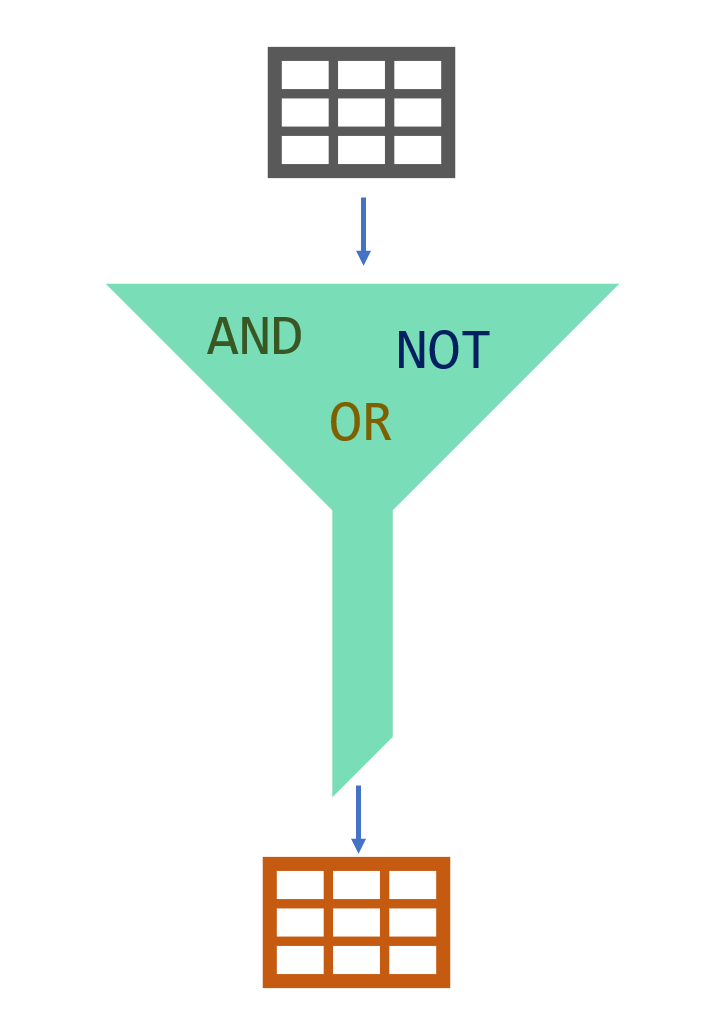
    -02

We first returned the employees whose salary is higher than $80,000. Next, we sorted this by the first names in descending order. Alright, we now know how to sort our table. It's time to put them into practice.

## AND, OR & NOT Operators

In SQL, **AND, OR & NOT** keywords are called operators. In particular, they are called logical operators. Their purposes are filtering the data based on conditions.

The WHERE clause can be combined with AND, OR & NOT operators. Let's start with the AND operator.



AND, OR & NOT Operators

AND Operator

The AND operator is used with the WHERE clause and combines multiple expressions. It returns only those records where both conditions (in WHERE clause) evaluate to True. The syntax has the following form in the WHERE condition:

WHERE left\_conditon AND right\_condition

Now, display the employees whose title is a data scientist and gender is male. You would be asked for the same thing as "show me the male data scientists in the company. " They are both the same.   
  


There are two conditions here. One is the title of the employee is a data scientist, other is the gender of his/her should be male. The correct search condition in where clause is job\_title = 'data scientist' AND gender = 'Male'.

Let's write the query.

query :

SELECT \*

FROM employees

WHERE job\_title = 'Data Scientist' AND gender = 'Male';

There is only one record that meets both conditions. So only one record returns.   
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- -------------- ----------

    ----------

30840 David Barrow 85000 Data Scientist Male 2019-12

    -02

## AND, OR & NOT Operators

### OR Operator

The OR operator is used with the WHERE clause and combines multiple expressions. It displays the record where either one of conditions (in WHERE clause) evaluates to True. The syntax has the following form in the WHERE condition.

WHERE first\_condition OR second\_condition

Display the employees whose title is a data scientist or gender is male.   
  


query :

SELECT \*

FROM employees

WHERE job\_title = 'Data Scientist' OR gender = 'Male';

The query returns all the male employees and data scientists. Since there is a female data scientist, the result table also displays it.  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ------------------- ----------

    ----------

17679 Robert Gilmore 110000 Operations Director Male

    2018-09-04

26650 Elvis Ritter 86000 Sales Manager Male

    2017-11-24

30840 David Barrow 85000 Data Scientist Male

    2019-12-02

49714 Hugo Forester 55000 IT Support Speciali Male

    2019-11-22

51821 Linda Foster 95000 Data Scientist Female

    2019-04-29

70950 Rodney Weaver 87000 Project Manager Male

    2018-12-20

76589 Jason Christian 99000 Project Manager Male

    2019-01-21

As we mentioned above, the records which meet either of the conditions return as a result. 

**💡 Tip:** Don't get confused with ANDs and ORs!

* When you want **ALL** of your conditions to be true, use **AND**
* When you want **ANY** of your conditions to be true, use **OR**

AND, OR & NOT Operators

NOT Operator

The NOT operator is used to negate a condition in the WHERE clause. NOT is placed right after WHERE keyword. You can use it with AND & OR operators. Here is the syntax of NOT operator.

WHERE NOT first\_condition

Display the male employees.   
  


At first, you may write this query:

SELECT \*

FROM employees

WHERE gender = 'Male';

We can also write it in another way using NOT operator.  
  
query :

SELECT \*

FROM employees

WHERE NOT gender = 'Female';

Both queries will yield the same result below.  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ------------------- ----------

    ----------

17679 Robert Gilmore 110000 Operations Director Male

    2018-09-04

26650 Elvis Ritter 86000 Sales Manager Male

    2017-11-24

30840 David Barrow 85000 Data Scientist Male

    2019-12-02

49714 Hugo Forester 55000 IT Support Speciali Male

    2019-11-22

70950 Rodney Weaver 87000 Project Manager Male

    2018-12-20

76589 Jason Christian 99000 Project Manager Male

    2019-01-21

Now, it's your turn. Time to practice in the playground.

BETWEEN OPERATOR

Introduction

The BETWEEN operator is used for comparison in WHERE clauses. It's a comparison operator. You can use it to test if a value is in a range of values. If the value is in the specified range, the query returns all records fallen within that range.

The following displays the syntax of the BETWEEN operator:

WHERE test\_expression BETWEEN low\_expression AND high\_expression

Note that the BETWEEN operator is inclusive. The above syntax can be written as follows:

WHERE test\_expression >= low\_expression AND test\_expression <= high\_expression

**☝ Important:** The **BETWEEN** operator is inclusive. To specify an exclusive range, use the greater than (>) and less than operators (<).



If we need to find the names of the employees with salary amounts between $80,000 and $90,000, we can use the BETWEEN comparison operator to write:

query :

SELECT \*

FROM employees

WHERE salary BETWEEN 80000 AND 90000;

Here is the output:

output :

emp\_id first\_name last\_name salary job\_title gender hire\_date

---------- ---------- ---------- ---------- ------------- ----------

    ----------

26650 Elvis Ritter 86000 Sales Manager Male 2017-11

    -24

30840 David Barrow 85000 Data Scientis Male 2019-12

    -02

70950 Rodney Weaver 87000 Project Manag Male 2018-12

    -20

You could also write the above query as follows:

SELECT \*

FROM employees

WHERE salary >= 80000 AND salary <= 90000;

They both return the same result table.

BETWEEN OPERATOR

NOT BETWEEN

We can use NOT BETWEEN to negate the result of the BETWEEN operator. The following is the syntax:

WHERE test\_expression NOT BETWEEN low\_expression AND high\_expression

For instance, you need to find the employees whose salary is not between $80,000 and $90,000. Here is the query:  
  
query :

SELECT \*

FROM employees

WHERE salary NOT BETWEEN 80000 AND 90000;

output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ------------------- ----------

    ----------

17679 Robert Gilmore 110000 Operations Director Male

    2018-09-04

49714 Hugo Forester 55000 IT Support Speciali Male

    2019-11-22

51821 Linda Foster 95000 Data Scientist Female

    2019-04-29

67323 Lisa Wiener 75000 Business Analyst Female

    2018-08-09

71329 Gayle Meyer 77000 HR Manager Female

    2019-06-28

76589 Jason Christian 99000 Project Manager Male

    2019-01-21

97927 Billie Lanning 67000 Web Developer Female

    2018-06-25

There are seven employees whose salary is not between $80,000 and $90,000.

**🛈**We could write this query as follows:

SELECT \*

FROM employees

WHERE salary < 80000 OR salary > 90000;

Think about why both queries yield the same result.

BETWEEN OPERATOR

BETWEEN with Date Example

It's also possible to use the BETWEEN operator with dates.

Assume that we try to find employees who have joined the company from June 1, 2018 to March 31, 2019. We also want to sort by hire date in ascending order.



Let's write the query.

**☝ Important:** Please enclose your date values with single quote (') and use YYYY-MM-DD date format in your query.

query :

SELECT \* FROM employees

WHERE hire\_date BETWEEN '2018-06-01' AND '2019-03-31'

ORDER BY hire\_date;

output :

emp\_id first\_name last\_name salary job\_title gender hire\_date

---------- ---------- ---------- ---------- ------------- ----------

    ----------

97927 Billie Lanning 67000 Web Developer Female 2018-06

    -25

67323 Lisa Wiener 75000 Business Anal Female 2018-08

    -09

17679 Robert Gilmore 110000 Operations Di Male 2018-09

    -04

70950 Rodney Weaver 87000 Project Manag Male 2018-12

    -20

76589 Jason Christian 99000 Project Manag Male 2019-01

    -21

There are five employees who have joined the company from June 1, 2018 to March 31, 2019.

IN Operator

Introduction

The IN operator is used to determine whether a value matches any value in a list. We use IN operator with WHERE clause. Following is the syntax of the IN operator:

WHERE column\_name IN (value\_list)

Suppose that you are building a team in your company. The team is comprised of Data Scientist and Business Analyst. You need to search the employee table to find the right candidates for your team. You may come up with this query:  
  
query :

SELECT \*

FROM employees

WHERE job\_title = 'Data Scientist'

OR

job\_title = 'Business Analyst';

That's a correct query which returns this:  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ---------------- ----------

    ----------

30840 David Barrow 85000 Data Scientist Male 2019

    -12-02

51821 Linda Foster 95000 Data Scientist Female 2019

    -04-29

67323 Lisa Wiener 75000 Business Analyst Female 2018

    -08-09

Three people meet your criteria.

However, there is a better operator in case you try to match a value in a specified list. We may rewrite the query as follows:

SELECT \*

FROM employees

WHERE job\_title IN ('Data Scientist', 'Business Analyst');

The query retrieves the employees whose job title is Data Scientist or Business Analyst. Herein the value list is ('Data Scientist', 'Business Analyst'). When any value in the job title column matches one of the values in the list, the related row is returned.

IN Operator

An Extended Value List

Suppose that you have decided to add teammates to your existing team. In addition to data scientist and business analyst, you are in need of a project manager and web developer. You modified the old query in which you used the OR operator.

Here is the new query:

query :

SELECT \*

FROM employees

WHERE job\_title = 'Data Scientist'

OR

job\_title = 'Business Analyst'

OR

job\_title = 'Project Manager'

OR

job\_title = 'Web Developer';

Below is the output:

output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- ---------------- ----------

    ----------

30840 David Barrow 85000 Data Scientist Male 2019

    -12-02

51821 Linda Foster 95000 Data Scientist Female 2019

    -04-29

67323 Lisa Wiener 75000 Business Analyst Female 2018

    -08-09

70950 Rodney Weaver 87000 Project Manager Male 2018

    -12-20

76589 Jason Christian 99000 Project Manager Male 2019

    -01-21

97927 Billie Lanning 67000 Web Developer Female 2018

    -06-25

Then you thought that it's a better idea to use IN operator and revised the query using IN operator as below:

SELECT \*

FROM employees

WHERE job\_title IN ('Data Scientist', 'Business Analyst', 'Project Manager', 'Web

    Developer');

This query also returns the same result table. Six people meet your criteria.  
As you can see that using the IN operator makes your code shorter.

**💡 Tip:** If you have a query in which you use many OR operators, consider using the IN operator instead. This will make your query more readable.

IN Operator

NOT IN Operator

In this part, we are going to add the keyword NOT to our IN operator.  You're building a team again. This time you decided to select the right candidates in a different way. You don't want to include Operations Director, HR Manager, and Sales Manager in the team.  
  


If we know which values we don't want to include in a list, we can use NOTkeyword with IN. NOT gives you the opposite results, anything that doesn't match the list. Use NOT just before IN operator.  
Let's write the query:  
  
 query :

SELECT \*

FROM employees

WHERE job\_title

NOT IN ('Operations Director', 'HR Manager', 'Sales Manager');

The following is the output:  
  
output :

emp\_id first\_name last\_name salary job\_title gender

    hire\_date

---------- ---------- ---------- ---------- --------------------- ----------

    ----------

30840 David Barrow 85000 Data Scientist Male

    2019-12-02

49714 Hugo Forester 55000 IT Support Specialist Male

    2019-11-22

51821 Linda Foster 95000 Data Scientist Female

    2019-04-29

67323 Lisa Wiener 75000 Business Analyst Female

    2018-08-09

70950 Rodney Weaver 87000 Project Manager Male

    2018-12-20

76589 Jason Christian 99000 Project Manager Male

    2019-01-21

97927 Billie Lanning 67000 Web Developer Female

    2018-06-25

There are seven employees whose job title is not in the Operations Director, HR Manager, Sales Manager list. Alright, we completed the IN operator.

## LIKE Operator

### Introduction

There are some cases where you don't know exactly the complete value you query. For instance, consider that you're trying to recall a student's county info from our student\_info table. You know that the county's name starts with 'Wo'. In such cases, we use LIKE operator.  We combine LIKE operator with WHERE clause.

The general syntax is:

SELECT column\_name(s)

FROM table\_name

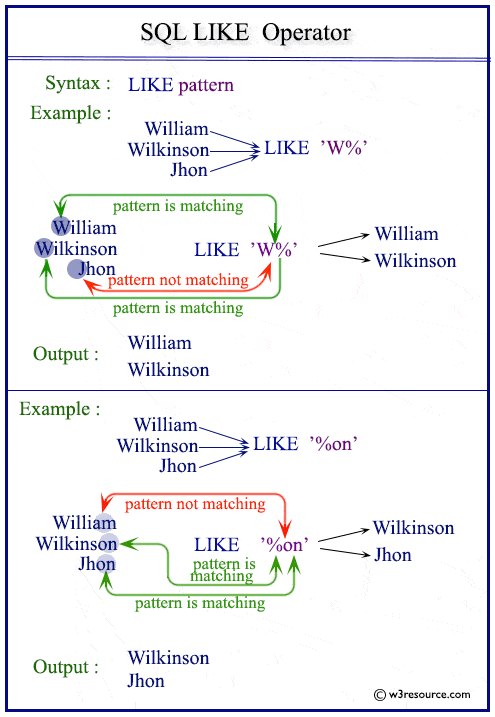
WHERE column\_1 LIKE pattern;

After LIKE keyword, we construct a pattern. SQL provides two special characters for constructing patterns. These are also called wildcards.

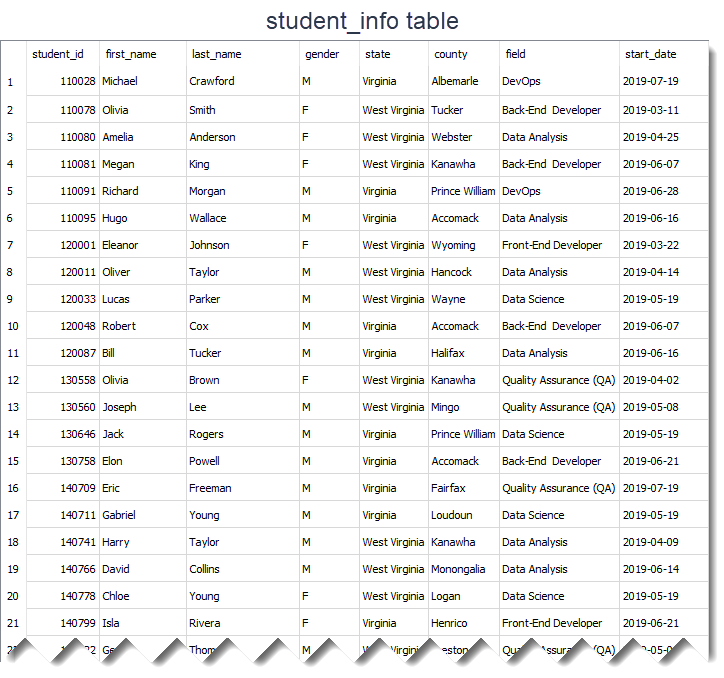
* **Percent (%):**The % character matches any sequence of zero or more characters.
* **Underscore ( \_ ):** The \_ character matches any single character.

Patterns are case insensitive. Uppercase characters do match lowercase characters or vice versa. Let's show pattern matching in the following examples. You will find some pattern examples which can be used after the LIKE operator. Patterns are written in single quotes.

* 'Out%' pattern matches any string beginning with "Out" such as "Outro".
* 's%' pattern matches any string that starts with "s" such as "silk", "soup", etc.
* '%per%' pattern matches any string containing "per" such as "percentile" and "peeper".
* 's\_n' pattern matches "son", "sun", etc.
* '\_\_te' pattern matches "mate", "Kate", "kate", etc.



Now. Let's try to find a solution to our question at the beginning. We were trying to recall the county name which starts with "Wo". Since we don't know how many letters coming after "Wo", it's a good idea to use % wildcard here. Here is the pattern 'Wo%'.



This is the query:

query :

SELECT \*

FROM student\_info

WHERE county LIKE 'Wo%';

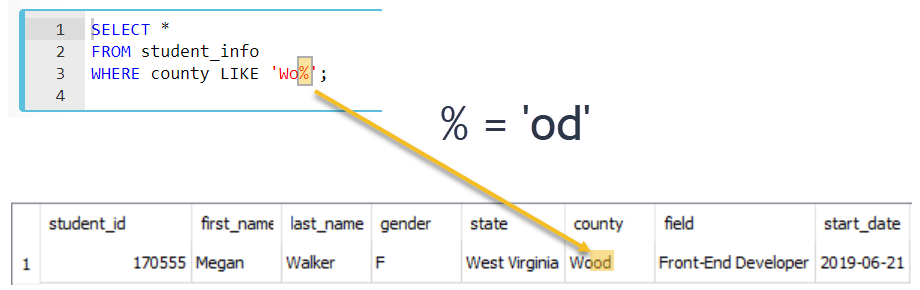
Result table:  
  
output :

student\_id first\_name last\_name gender state county field start\_date

---------- ---------- ---------- ---------- ------------- ---------- ------------------- ----------

170555 Megan Walker F West Virginia Wood Front-End Developer 2019-06-21

% wildcard matches with the letters 'od' in the Wood word. Remember that %wildcard matches any sequence of zero or more characters.



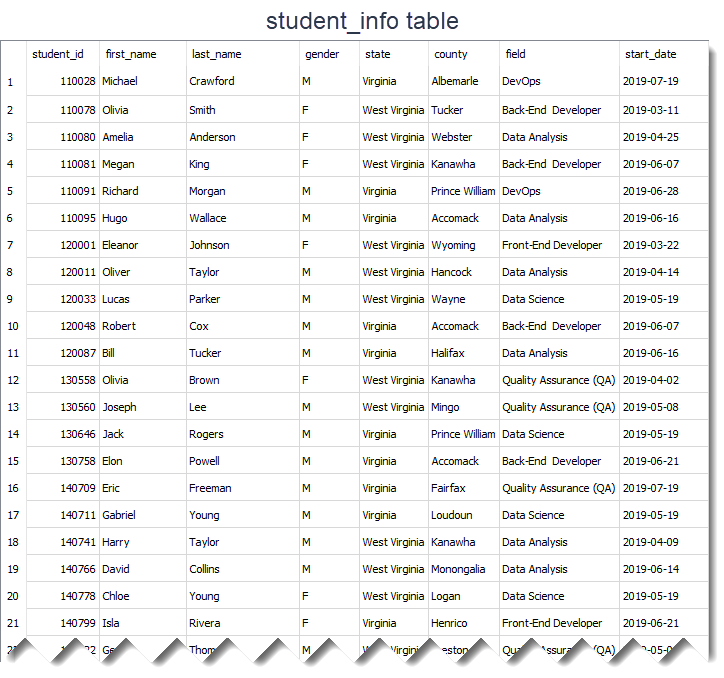
Since LIKE operator is case insensitive, you could write the pattern LIKE 'wo%' . Both patterns will produce the same results.

## LIKE Operator

### Percent Character Example

In this part, we continue with another example using % wildcard.

Suppose that we try to find front-end and back-end developers in our student\_info table. If we look at the **field** **column,**we see that the word "Developer" is found at the end of the each field value. So, what do you think the pattern will be?



The pattern is '%Developer'. Herein % wildcard will include all the characters before the word "Developer". Let's write the query:

query :

SELECT \*

FROM student\_info

WHERE field LIKE '%Developer';

The output of the query:  
  
output :

student\_id first\_name last\_name gender state county field start\_date

---------- ---------- ---------- ------- ------------- ---------- ------------------- ----------

110078 Olivia Smith F West Virginia Tucker Back-End Developer 2019-03-11

110081 Megan King F West Virginia Kanawha Back-End Developer 2019-06-07

120001 Eleanor Johnson F West Virginia Wyoming Front-End Developer 2019-03-22

120048 Robert Cox M Virginia Accomack Back-End Developer 2019-06-07

130758 Elon Powell M Virginia Accomack Back-End Developer 2019-06-21

140799 Isla Rivera F Virginia Henrico Front-End Developer 2019-06-21

150227 Chloe Fisher F Virginia Fairfax Back-End Developer 2019-07-18

150234 George Martinez M West Virginia Pocahontas Front-End Developer 2019-05-07

150246 Arthur Wright M West Virginia Monongalia Back-End Developer 2019-06-07

160021 Olivia Cooper F Virginia Bedford Front-End Developer 2019-06-21

170555 Megan Walker F West Virginia Wood Front-End Developer 2019-06-21

170566 Jack Morris M West Virginia Wetzel Front-End Developer 2019-06-28

There are 12 students whose field is Back-End Developer or Front-End Developer. We could write the pattern as '%Developer%'. This also returns the same result table. There is no character coming after the word "Developer". But remember % wildcard also matches zero characters.

In the previous part, we said that LIKE operator is case-insensitive. That means instead of '%Developer', we can write '%developer', '%DEVELOPER' even '%DEveloper'. All those patterns will match "Back-End Developer" or "Front-End Developer".

However, if you want to make LIKE operator case-sensitive, we need to use PRAGMA statement as follows:

PRAGMA case\_sensitive\_like = true;

Write the query again using PRAGMA statement and pattern '%developer'.  
  
query :

PRAGMA case\_sensitive\_like = true;

SELECT \*

FROM student\_info

WHERE field LIKE '%developer';

The output of the new query is a blank table.  
  
output :

In case you change the setting of LIKE operator from case-insensitive to case-sensitive, you need to construct the pattern as '%Developer' to match.

### Underscore Character Example

The underscore \_ wildcard matches a single character. Think of it as representing a single character. For instance, the following query finds the employee whose first\_name is "Elvis".



query :

SELECT first\_name

FROM employees

WHERE first\_name LIKE 'El\_is';

Here is the output of the query above.

output :

first\_name

----------

Elvis

The \_ wildcard in the pattern matches "v" letter in "El**v**is" in the first name column.

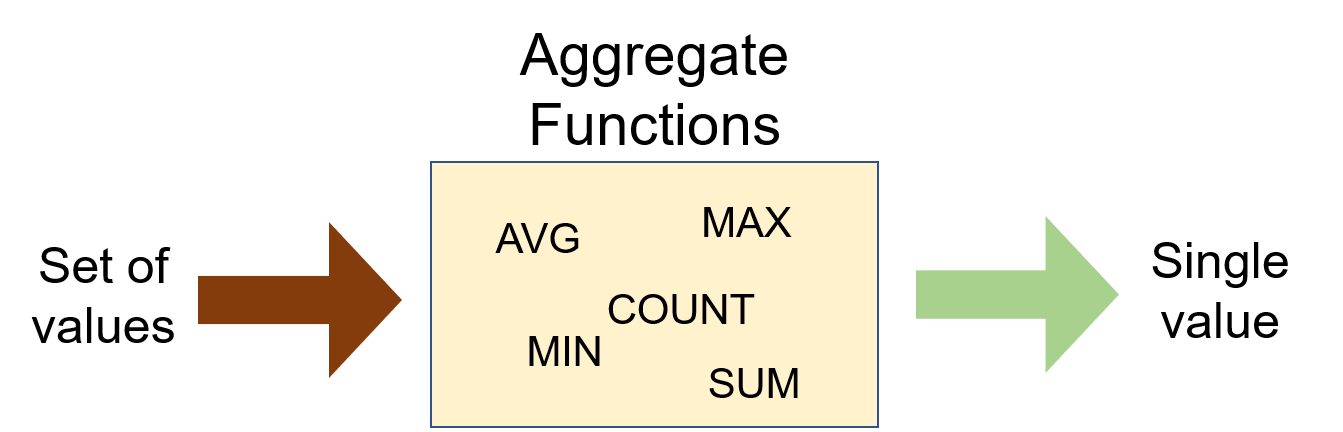
Which first name the 'Li\_\_a' pattern will match in the first name column? The answer is "Linda". Note that there are two underscore characters in the 'Li\_\_a' pattern. Thus, those will match "nd" letters in "Li**nd**a" in the first name column.

## COUNT

### What is an aggregate function?

In this section, we will introduce you to *aggregate functions*. Aggregate functions are functions that take a collection of values as input and return a single value. SQL has five built-in aggregate functions:

* COUNT (Count)
* MIN (Minimum)
* MAX (Maximum)
* SUM (Total)
* AVG (Average)



SUM and AVG functions must take only numeric values as input. However, the other functions (MIN, MAX, COUNT) may take non-numeric values (strings, date, etc.) as input other than numeric values. Before we dive into the aggregate functions, we should learn an important concept which is NULL.

NULL means no data and is a special value. It shows us that a piece of information is unknown or missing or not applicable. For instance, let's assume that we have a song database and we don't know the writers of some of them. We don't have that information. So, to store these unknown songwriters in a database table, we must use NULL. If a table has NULL value, then it will be displayed as NULL.**Tips:**

* NULL value represents the unknown value or missing value or not applicable.
* NULL is not equal to zero or empty string.
* NULL is not equal to itself.

It's important to understand what nulls mean and to know how to work with them. They are often ignored in our aggregation functions.

Alright, let's start with COUNT function.

Q: What is an aggregate function?  
A: An aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement.

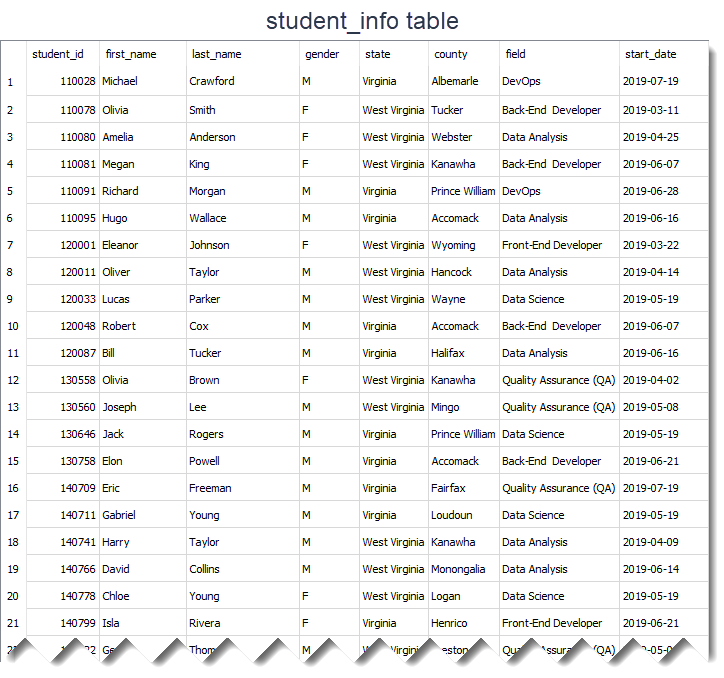
## COUNT

### Introduction

We use COUNT function to count the numbers of records (a.k.a row) in a table. Here is the syntax of the COUNT function:

SELECT COUNT(column\_name)

FROM table\_name;

Use COUNT keyword immediately after SELECT keyword. Then specify the column on which you want to operate.   
Suppose that we want to count how many students enrolled in the courses.   


Let's write the query as follows:

query :

SELECT COUNT(first\_name)

FROM student\_info;

The output of the query:  
  
output :

COUNT(first\_name)

-----------------

32

32 students enrolled in the courses.   
This query counts all the first names in the student\_info table. You would reach the same result if you used COUNT(last\_name). There is another special character returning the number of rows in a table. That is \* character.  Use it inside the COUNT function as COUNT(\*).   
  
If you notice that the header of the output query is displayed as COUNT(first\_name). However, we can customize the header using AS keyword. AS is used to rename a column or table with an alias. This is the syntax:  
 SELECT column\_name AS alias\_name  
  
Let's rewrite the query above:  
  
query :

SELECT COUNT(first\_name) AS count\_of\_students

FROM student\_info;

output :

count\_of\_students

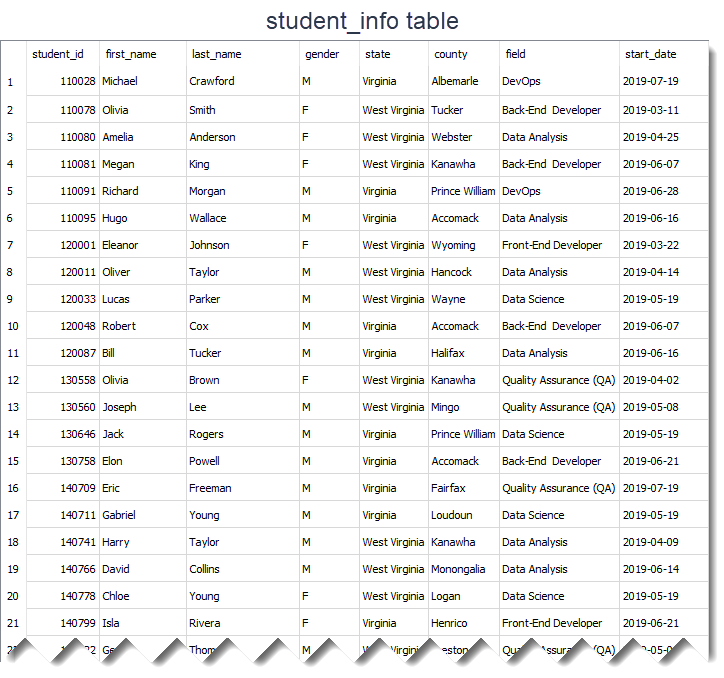
-----------------

32

As you see that what you wrote after AS keyword is displayed as a column header.   
  
An important point for COUNT(\*) function is that the result table includes NULL and duplicate values. If you want the number of non-null values, use the syntax COUNT(column\_name).

### COUNT DISTINCT

In some cases, we may want unique values. For instance, let's find how many different fields there are in the student\_info table.



First, write this query:

query :

SELECT COUNT(field) AS count\_of\_field

FROM student\_info;

The output of the query:

output :

count\_of\_field

--------------

32

As you see that there are 32 fields. But, is that the correct number? Absolutely no. There are duplicate values. For instance, there is more than one student who enrolled in the course in the field of *Data Science*. COUNT(field) counts the duplicate rows as separate rows. So, we need to use COUNT(DISTINCT column\_name) function here.

query:

SELECT COUNT(DISTINCT field) AS count\_of\_field

FROM student\_info;

output:

count\_of\_field

--------------

6

Actually there are 6 unique or distinct fields in which students may enroll. COUNT(DISTINCT field) returns the number of unique and non-null values in column *field*.

We can also combine COUNT(DISTINCT) or COUNT() functions with WHERE clause.

For instance, display the number of students from Virginia.

query :

SELECT COUNT(\*) AS count\_of\_students

FROM student\_info

WHERE state = 'Virginia';

output :

count\_of\_students

-----------------

14

There are 14 students from Virginia state.

### MIN Function

MIN function returns the minimum value in the selected column.

The syntax of the MIN function:

SELECT MIN(column\_name)

FROM table\_name;

The MIN function ignores the NULL values. Thus, it retrieves the non-NULL minimum value in the selected column.



Alright, let's see it in an example. Who gets paid the lowest wage in the company?  
  
query :

SELECT MIN(salary) AS lowest\_salary

FROM employees;

output :

lowest\_salary

-------------

55000

We can also combine MIN function with WHERE clause. Let's move the query above to the next level. Display the female employee who gets paid the lowest.   
  
query :

SELECT MIN(salary) AS lowest\_salary

FROM employees

WHERE gender = 'Female';

output :

lowest\_salary

-------------

67000

The lowest salary is $67,000 among female employees.

**✍ Exercise:** Try to write the query above without using MIN function.

Click for the solution  
  
You can use ORDER BY and LIMIT keywords to get the minimum value in a selected column. This will yield the same result as in the MIN function.  
  
The MIN function can also be used with dates. For instance, display the earliest hired employee's date.  
  
query :

SELECT MIN(hire\_date) AS earliest\_date

FROM employees;

output :

earliest\_date

-------------

2017-11-24

### MAX Function

MAX function returns the maximum value in the selected column. With the MAX function, you can find the highest salary among the employees, the most expensive products, the longest track, etc.

The syntax of the MAX function:

SELECT MAX(column\_name)

FROM table\_name;

The MAX function ignores the NULL values. 



Alright, let's take the same example in the previous part. But, this time find the highest wage in the company.  
  
query :

SELECT MAX(salary) AS highest\_salary

FROM employees;

output :

highest\_salary

-------------

110000

We can also combine MAX function with WHERE clause as we did in the MIN function. Display the male employee who gets paid the highest.   
  
query :

SELECT MAX(salary) AS highest\_salary

FROM employees

WHERE gender = 'Male';

output :

highest\_salary

-------------

110000

The highest salary is $110,000 among male employees. This amount is also the highest one among all employees.

**✍ Exercise:** Try to write the query above without using MAX function.

Click for the solution  
  
You can use ORDER BY and LIMIT keywords to get the maximum value in a selected column. This will yield the same result as in the MAX function.  
  
The MAX function can be used with dates. For instance, we can display the newest hired employee's date as below:  
  
 query :

SELECT MAX(hire\_date) AS newest\_date

FROM employees;

output:

newest\_date

-----------

2019-12-02

### SUM Function

SUM function returns the sum of a numeric column.

The syntax of the SUM function:

SELECT SUM(column\_name)

FROM table\_name;

Now, let's calculate the total amount of the salary of the employees in the company.  
  


query:

SELECT SUM(salary) AS total\_salary

FROM employees;

output:

total\_salary

------------

836000

The amount of the total salary is $836,000. We can breakdown this into categories. For instance, we can find the total amount of salary of female employees.  
query:

SELECT SUM(salary) AS total\_salary

FROM employees

WHERE gender = 'Female'

output:

total\_salary

------------

314000

**✍ Exercise:** Find the total amount of the salary for the male employees.

### AVG Function

AVG function calculates the average of a numeric column.

The syntax of the AVG function:

SELECT AVG(column\_name)

FROM table\_name;

What is the average salary of the employees?   
  
query:

SELECT AVG(salary) AS average\_salary

FROM employees;

output:

average\_salary

------------

83600.0

The average salary of the employees is $83,600.  The total amount of the salary is $836,000. Since there are 10 employees, the average salary is $836,000/10 = $83,600. We just proved it.  
  
Now, let's find the average salary amongst male employees.   
  
input:

SELECT AVG(salary) AS average\_salary

FROM employees

WHERE gender = 'Male';

output:

average\_salary

------------

87000.0

**✍ Exercise:** Find the average salary for the female employees.

Click for the solution

query:

SELECT AVG(salary) AS average\_salary

FROM employees

WHERE gender = 'Female';

output:

avg\_salary

----------

78500.0

GROUP BY Clause

There are some cases where we may need to apply the aggregate function not only to a selected group of rows. In the previous parts, we found the total amount of salaries of male and female employees separately.  What if want to show them in the same result table grouped by their gender. In such cases, we use GROUP BY statement.

The GROUP BY clause groups the rows into summary rows. It returns one value for each group and is typically used with aggregate functions (COUNT, MAX, MIN, SUM, AVG).

The syntax is:

SELECT column\_1, aggregate\_function(column\_2)

FROM table\_name

GROUP BY column\_1;

In case a WHERE clause is used, the GROUP BY clause should come after the WHERE clause. We specify the column or a list of comma-separated columns right after the GROUP BY.  Any nonaggregate expression in the SELECT must be included in the GROUP BY clause.

We've mentioned that the GROUP BY clause is often used with aggregate functions. Now, let's see how it is.

**💡Tips:**

* GROUP BY returns only one result per group of data.
* GROUP BY Clause always follows the WHERE Clause.
* GROUP BY Clause always precedes the ORDER BY.

### GROUP BY with COUNT Function



The following statement returns the number of employees per gender.

query:

SELECT gender, COUNT(gender)

FROM employees

GROUP BY gender;

output:

gender COUNT(gender)

---------- -------------

Female 4

Male 6

**☝ Important:** The GROUP BY clause groups results before calling the aggregate function. This allows you to apply aggregate function to groups than the entire query.

Now, let's write another query. What is the number of employees working as a data scientist? Break down it by gender.  
  
query:

SELECT gender, COUNT(job\_title)

FROM employees

WHERE job\_title = 'Data Scientist'

GROUP BY gender;

output:

gender COUNT(job\_title)

---------- ----------------

Female 1

Male 1

### GROUP BY with MIN&MAX Functions

Let's find the minimum salaries of each gender group using the MIN function.  
  
query:

SELECT gender, MIN(salary) AS min\_salary

FROM employees

GROUP BY gender;

output:

gender min\_salary

---------- ----------

Female 67000

Male 55000

Similarly, we can find the maximum salaries of each group using the MAX function.  
  
query:

SELECT gender, MAX(salary) AS max\_salary

FROM employees

GROUP BY gender;

output:

gender max\_salary

---------- ----------

Female 95000

Male 110000

You may also use the ORDER BY clause to sort the salaries in descending or ascending order.  The ORDER BY follows GROUP BY. For instance, sort the maximum salaries for each group in descending order.  
  
query:

SELECT gender,

MAX(salary) AS max\_salary

FROM employees

GROUP BY gender

ORDER BY max\_salary DESC;

output:

gender max\_salary

---------- ----------

Male 110000

Female 95000

**💡 Tip:** When sorting, you can use the alias with ORDER BY clause as we did in the query above (ORDER BY **max\_salary**). You may also use the original aggregate function as ORDER BY **MAX(salary**)

### GROUP BY with SUM&AVG Functions

In the previous part, we have learned how to use MIN and MAX functions with the GROUP BY clause. In this part, we will continue with SUM and AVG functions. Let's go with the same scenario. In this case, we will calculate the total salaries of each group (gender).

query:

SELECT gender, SUM(salary) AS total\_salary

FROM employees

GROUP BY gender;

output:

gender total\_salary

---------- ------------

Female 314000

Male 522000

Similarly, we can find the average salaries of each group using the AVG function.  
  
query:

SELECT gender, AVG(salary) AS average\_salary

FROM employees

GROUP BY gender;

output:

gender average\_salary

---------- --------------

Female 78500.0

Male 87000.0

You may also use the ORDER BY clause to sort the salaries in descending or ascending order. For instance, sort the total salaries for each group in descending order.  
  
query:

SELECT gender,

SUM(salary) AS total\_salary

FROM employees

GROUP BY gender

ORDER BY total\_salary DESC;

output:

gender total\_salary

---------- ------------

Male 522000

Female 314000

## What is JOIN?

### Introduction

In this section, we will introduce you to *JOIN clauses*. A JOIN clause is used to combine two or more tables into a single table. Joins are usually applied based on the keys that define the relationship between those tables or on common fields.

**💡Tips:**

* In most cases this joins are created using the primary key of one table and the foreign key of the table we want to join it with.

We've added a new table "departments" to our company database. We have now two tables "employees" and "departments" in the database. Now let's examine those two tables. In the "departments" table, the names of the employees are not written, only the ID numbers of the employees are written.

Let's look at the "employees" table:

emp\_id first\_name last\_name salary job\_title gender hire\_date

------ ---------- --------- ------ ------------------- -------- ----------

17679 Robert Gilmore 110000 Operations Director Male 2018-09-04

26650 Elvis Ritter 86000 Sales Manager Male 2017-11-24

30840 David Barrow 85000 Data Scientist Male 2019-12-02

49714 Hugo Forester 55000 IT Support Speciali Male 2019-11-22

51821 Linda Foster 95000 Data Scientist Female 2019-04-29

67323 Lisa Wiener 75000 Business Analyst Female 2018-08-09

70950 Rodney Weaver 87000 Project Manager Male 2018-12-20

71329 Gayle Meyer 77000 HR Manager Female 2019-06-28

76589 Jason Christian 99000 Project Manager Male 2019-01-21

97927 Billie Lanning 67000 Web Developer Female 2018-06-25

Then, let's look at the "departments" table:

emp\_id dept\_name dept\_id

------ -------------- -------

17679 Operations 13

26650 Marketing 14

30840 Operations 13

49823 Technology 12

51821 Operations 13

67323 Marketing 14

71119 Administrative 11

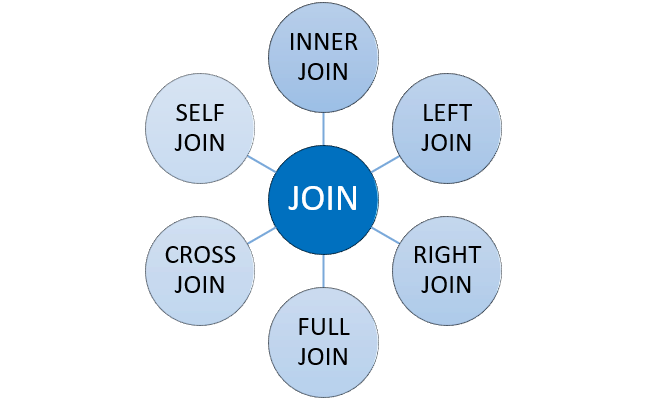
76589 Operations 13

97927 Technology 12

When looking at the "departments" table, it is not easy to understand which employee is working in which department, since this table does not contain the names of the employees. For this purpose, two tables should be combined and the name and surname of that employee should be found by looking at the "employees" table for each employee ID in the "departments" table. The "emp\_id" is the column that will form the relationship between the two tables above.

JOIN Types

There are basically five types of JOINs: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN and CROSS JOIN. As a special case, a table can join to itself and this type is called SELF JOIN.



*Types of JOINs*

Let's make a brief description of JOIN types without going in-depth:

* **INNER JOIN:** Returns the common records in both tables.
* **LEFT JOIN:** Returns all records from the left table and matching records from the right table.
* **RIGHT JOIN:** Returns all records from the right table and matching records from the left table.
* **FULL OUTER JOIN:** Returns all records of both left and right tables.
* **CROSS JOIN:** Returns the Cartesian product of records in joined tables.
* **SELF JOIN:** A join of a table to itself.

In our course, we'll focus on INNER join and LEFT join which are commonly used.

Q: What is a Join?  
A: The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.

## Inner JOIN

### Introduction

INNER JOIN is the most common type of JOINs. It creates a new result table based on the values in common columns from two or more tables. INNER JOIN returns a table that contains only matched rows that meet the specified join conditions.

Here is the syntax of the INNER JOIN clause:

SELECT columns

FROM table\_A

INNER JOIN table\_B ON join\_conditions

In this syntax,

* *columns*: Column names from table\_A or table\_B.
* *table\_A, table\_B*: The names of the joined tables.
* *join\_conditions:* It specifies the conditions to evaluate for each pair of joined rows.

A join condition generally takes the following form: table\_A.column = table\_B.column. The operator in this statement is usually an equal sign (=), but any comparison operator can also be used.

**💡Tips:**

* Note the ON keyword for specifying the INNER JOIN condition.
* Multiple join conditions can be written using AND or OR statements.

In addition, three or more tables can be combined using the INNER JOIN clause. The syntax used to join three or more tables is as follows:

SELECT columns

FROM table\_A

INNER JOIN table\_B

ON join\_conditions1 AND join\_conditions2

INNER JOIN table\_C

ON join\_conditions3 OR join\_conditions4

...

Q: What is inner join?  
A: Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.

## Inner JOIN

### Example

Suppose you want to join the "employees" table and the "departments" table. In this example, an INNER JOIN has been created that is based on the *emp\_id*columns in the two tables:

**Tables**

"employees" table:

emp\_id first\_name last\_name salary job\_title gender hire\_date

------ ---------- --------- ------ ------------------- -------- ----------

17679 Robert Gilmore 110000 Operations Director Male 2018-09-04

26650 Elvis Ritter 86000 Sales Manager Male 2017-11-24

30840 David Barrow 85000 Data Scientist Male 2019-12-02

49714 Hugo Forester 55000 IT Support Speciali Male 2019-11-22

51821 Linda Foster 95000 Data Scientist Female 2019-04-29

67323 Lisa Wiener 75000 Business Analyst Female 2018-08-09

70950 Rodney Weaver 87000 Project Manager Male 2018-12-20

71329 Gayle Meyer 77000 HR Manager Female 2019-06-28

76589 Jason Christian 99000 Project Manager Male 2019-01-21

97927 Billie Lanning 67000 Web Developer Female 2018-06-25

"departments" table:

emp\_id dept\_name dept\_id

------ -------------- -------

17679 Operations 13

26650 Marketing 14

30840 Operations 13

49823 Technology 12

51821 Operations 13

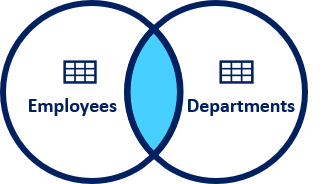
67323 Marketing 14

71119 Administrative 11

76589 Operations 13

97927 Technology 12

To understand easily how INNER JOIN works, we can use the following visual explanation. The intersection of the two tables represents the matching rows.



*Visual Representation of Inner JOIN*

**Syntax**

The INNER JOIN keyword selects all rows from both "employees" and "departments" tables as long as there is a match between the columns. If there are records in the "employees" table that do not have matches in "departments", these records are not shown in the output.

query:

SELECT

employees.emp\_id,

employees.first\_name,

employees.last\_name,

departments.dept\_name,

departments.dept\_id

FROM employees

INNER JOIN departments

ON employees.emp\_id = departments.emp\_id;

output:

emp\_id first\_name last\_name dept\_name dept\_id

------ ---------- --------- ---------- -------

17679 Robert Gilmore Operations 13

26650 Elvis Ritter Marketing 14

30840 David Barrow Operations 13

51821 Linda Foster Operations 13

67323 Lisa Wiener Marketing 14

76589 Jason Christian Operations 13

97927 Billie Lanning Technology 12

As you can see from the output, the seven employees in the "employees" table have matches in the "department" table. The remaining three employees are not listed in the output. As you can see, employee ID 17679 is in both tables and matches, so it is included in our output table. On the other hand, employee ID 49714 is included in the employee table but not in the department table and there is no match. Therefore, employee ID 49714 is not included in the output table.

## Left JOIN

### Introduction

In this JOIN statement, all the records of the left table and the common records of the right table are returned in the query. If no matching rows are found in the right table during the JOIN operation, these values are assigned as NULL.

Here is the syntax of the LEFT JOIN clause:

SELECT columns

FROM table\_A

LEFT JOIN table\_B ON join\_conditions

In this syntax,

* *columns*: Column names from table\_A or table\_B.
* *table\_A, table\_B*: The names of the joined tables.
* *join\_conditions:* It specifies the conditions to evaluate for each pair of joined rows.

LEFT JOIN and LEFT OUTER JOIN keywords are exactly the same. OUTER keyword is *optional*.

SELECT columns

FROM table\_A

LEFT OUTER JOIN table\_B ON join\_conditions

## Left JOIN

### Example

Suppose you want to join the "employees" table and the "departments" table. In this example, a LEFT JOIN has been created that is based on the *emp\_id*columns in the two tables:

**Tables**

"employees" table:

emp\_id first\_name last\_name salary job\_title gender hire\_date

------ ---------- --------- ------ ------------------- -------- ----------

17679 Robert Gilmore 110000 Operations Director Male 2018-09-04

26650 Elvis Ritter 86000 Sales Manager Male 2017-11-24

30840 David Barrow 85000 Data Scientist Male 2019-12-02

49714 Hugo Forester 55000 IT Support Speciali Male 2019-11-22

51821 Linda Foster 95000 Data Scientist Female 2019-04-29

67323 Lisa Wiener 75000 Business Analyst Female 2018-08-09

70950 Rodney Weaver 87000 Project Manager Male 2018-12-20

71329 Gayle Meyer 77000 HR Manager Female 2019-06-28

76589 Jason Christian 99000 Project Manager Male 2019-01-21

97927 Billie Lanning 67000 Web Developer Female 2018-06-25

"departments" table:

emp\_id dept\_name dept\_id

------ -------------- -------

17679 Operations 13

26650 Marketing 14

30840 Operations 13

49823 Technology 12

51821 Operations 13

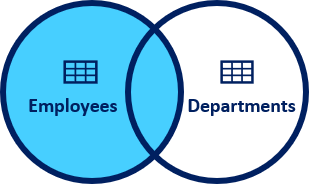
67323 Marketing 14

71119 Administrative 11

76589 Operations 13

97927 Technology 12

To understand easily how LEFT JOIN works, we can use the following visual explanation. The shaded area represents all rows in the "employees" table and all matching rows in the "departments" table.



*Visual Representation of Left JOIN*

**Syntax**

query :

SELECT

employees.emp\_id,

employees.first\_name,

employees.last\_name,

departments.dept\_name,

departments.dept\_id

FROM employees

LEFT JOIN departments

ON employees.emp\_id = departments.emp\_id;

output:

emp\_id first\_name last\_name dept\_name dept\_id

---------- ---------- ---------- ---------- ----------

17679 Robert Gilmore Operations 13

26650 Elvis Ritter Marketing 14

30840 David Barrow Operations 13

49714 Hugo Forester null null

51821 Linda Foster Operations 13

67323 Lisa Wiener Marketing 14

70950 Rodney Weaver null null

71329 Gayle Meyer null null

76589 Jason Christian Operations 13

97927 Billie Lanning Technology 12

As a result of the LEFT JOIN operation, all ten rows are returned from the "employees" table and all matching rows in the "departments" table are returned. The rows that do not match with the right table are returned as null.

## Subquery

### Introduction

A subquery is a SELECT statement that is nested within another statement. The subquery is also called the **inner query** or **nested query**. A subquery may be used in:

* SELECT clause
* FROM clause
* WHERE clause

A subquery is usually embedded inside the WHERE clause. You may use comparison operators such as >, <, =, IN, NOT IN with WHERE clause.

Here is the syntax:

SELECT column\_name

FROM table\_1, table\_2

WHERE column\_name OPERATOR (

SELECT column\_name

FROM table\_1, table\_2);

Herein the query

SELECT column\_name

FROM table\_1, table\_2

WHERE column\_name OPERATOR

is called the outer query.  
  
The query

(SELECT column\_name

FROM table\_1, table\_2)

is called inner query (nested or subquery query). The inner query is executed first before the outer query. The results of the inner query are passed to the outer query.  
  
There are some rules when using subquery:

* A subquery must be enclosed in parentheses.
* An ORDER BY clause is not allowed to use in a subquery.
* The BETWEEN operator can't be used with a subquery. But you can use BETWEEN within the subquery.



Let's do an example. We'll find the employees who get paid more than Rodney Weaver. Our query should return first name, last name, and salary info of the employees.  
✍ Give it a try before seeing the solution.  
  
Click for the solution

* The inner query returns 87000 which is the salary of Rodney and passes this value to the outer query, in particular to the WHERE clause. (WHERE salary > 87000)
* In the outer query, WHERE clause filters the employees whose salary is more than 87000 and returns the first name, last name, and salary info of those.

There are three main types of subqueries:

* Single-row subqueries
* Multiple-row subqueries
* Correlated subqueries\*

Now, let's find out them.  
\* : This type of subquery will not be covered in the course.

Q: What is a Subquery?  
A: A subquery is a query within another query, also known as nested query or inner query . It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively.

### Single-Row Subqueries

Single-row subqueries return one row with only one column and are used with single-row operators such as **=**, **>**, **>=**, **<=**, **<>**, **!=** .

Scalar subquery which returns a single row with one column is an example of single-row subqueries. In the [previous lesson](https://lms.clarusway.com/mod/lesson/view.php?id=3979&pageid=5069), we have shown you a single-row subquery example by finding Rodney's salary that returns a single row with one column (87000).

##### 

##### 

##### 

We can also use aggregated functions since it returns a single row with a single column. Now let' s find out the employees who get paid more than the average salary. Our query should return first name, last name, and salary info of the employees.

##### 

##### **query:**

SELECT first\_name, last\_name, salary

FROM employees

WHERE salary >

(SELECT AVG(salary)

FROM employees);

##### **output:**

first\_name last\_name salary

---------- ---------- ----------

Robert Gilmore 110000

Elvis Ritter 86000

David Barrow 85000

Linda Foster 95000

Rodney Weaver 87000

Jason Christian 99000

##### **Let's analyze the query above:**

##### **We've used > operator with WHERE clause.**

##### **The inner query returns the average salary which is 83600 and passes this single value to the outer query.**

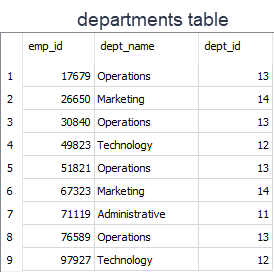
##### **Outer query filters those employees and returns those whose salary is more than 83600.**

As you see that combining aggregate functions with subqueries, we have more power to answer our analytical questions.

### Multiple-Row Subqueries

Multiple-row subqueries return sets of rows and are used with multiple-row operators such as IN, NOT IN, ANY, ALL.  Now, let's see it in an example.

As you know that we have added a new table to the company database consisting of one table. Our new table is the departments table. It has three columns and nine rows. Every row represents an employee's department info. Here are our database tables.****



Now with that said, find the employees (first name, last name from employees table) who work under the Operations department (departments table). Our query should return first name and last name info.

query:

SELECT first\_name, last\_name

FROM employees

WHERE emp\_id IN

(SELECT emp\_id

FROM departments

WHERE dept\_name = 'Operations');

output:

first\_name last\_name

---------- ----------

Robert Gilmore

David Barrow

Linda Foster

Jason Christian

Let's analyze the query above:

* We've used IN operator with WHERE clause.
* The inner query returns the employees ids who work under the Operations department and passes those to the outer query.
* Outer query filters those employees ids and returns their first\_name and last\_name as a result set.

## Introduction to DDL Commands

### Introduction

At the beginning of the course, we have mentioned that SQL performs some operations on the database such as retrieval, inserting, updating, creating. Of those, we have seen just SELECT statement for retrieval data operations on the database. As a side note, statements are also called "commands" in some sources.

Actually these statements are grouped into four main categories:

* DDL - Data Definition Language
* DML - Data Manipulation Language
* DCL - Data Control Language
* TCL - Transaction Control Language

[✱](https://www.toptal.com/designers/htmlarrows/symbols/heavy-asterisk/) Data Definition Language (DDL) specifies the database schema. The statements used in DDL are CREATE, ALTER, DROP. DDL statements are typically used to set up and configure a new database before we insert data.

[✱](https://www.toptal.com/designers/htmlarrows/symbols/heavy-asterisk/) Data Manipulation Language (DML) enables users to access or manipulate data. INSERT, UPDATE, DELETE, SELECT are the statements used in DML.

[✱](https://www.toptal.com/designers/htmlarrows/symbols/heavy-asterisk/) Data Control Language (DCL) is used to grant or revoke access control. Its statements are REVOKE and GRANT.

[✱](https://www.toptal.com/designers/htmlarrows/symbols/heavy-asterisk/) Transaction Control Language (TCL) controls the transactions of DML and DDL commands. Its statements are BEGIN and COMMIT.

DDL, DML, DCL, TCL are not separate languages. They compose parts of a single database language which is SQL. These are for categorical purposes. In this lesson, we'll learn DDL commands. As a side note, we are not going to cover other languages' commands in our course.

Before we dive into the DDL commands, we'll start with data types.

Data Types

In SQL, data types define what type of data a column can contain or store. Let's find out the data types of the employees' columns. There is a special function typeof in SQLite to display the data types of the values stored in the columns.



query:

SELECT typeof(emp\_id),

typeof(first\_name),

typeof(hire\_date)

FROM employees

LIMIT 1;

output:

typeof(emp\_id) typeof(first\_name) typeof(hire\_date)

-------------- ------------------ -----------------

integer text text

As you see that emp\_id stores integer values, first\_name and hire\_date store textual values. Different database systems may use different data types though there are common ones and SQLite is no exception.  
  
In SQLite, there are five primitive data types. They are called **storage classes**. Here are the details of them:

* **NULL:**The value is a NULL value. NULL values mean missing information or unknown.
* **INTEGER:**The value is a whole number (positive or negative), stored in 1, 2, 3, 4, 6 or 8 bytes depending on the magnitude of the value.
* **REAL:** The value is a floating-point value, stored as an 8-byte float. They are real numbers with decimal values.
* **TEXT:** The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
* **BLOB:** BLOB stands for a binary large object that can store any kind of data. The value is a blob of data, stored exactly as it was input.

In SQLite, there is no separate Boolean storage class. Boolean values are stored as integers 0 (false) and 1 (true).  
SQLite doesn't have a storage class allocated for dates and/or times. Instead, you use the TEXT, INTEGER, or REAL to store the date and/or time values. 

**💡Tips:**

* A storage class is slightly more general than a data type. For example, the INTEGER storage class covers six different integer data types of different lengths. Most of the part, storage class is indistinguishable from data type and the two terms can be used interchangeably.

SQLite determines the data type of a value based on its data type in accordance with the following rules:

* If a value is enclosed by single or double quotes, it is assigned the TEXT storage class.
* If a value has no enclosing quotes and decimal point or exponent, it is assigned INTEGER storage class.
* If a value has no enclosing quotes, but decimal point or exponent, it is assigned REAL storage class.
* If a value is NULL without quotes, it is assigned NULL storage class.
* If a value has X'ABCD' or x'abcd', it is assigned BLOB storage class.

### Example

As we've learned from the previous part, we can use the typeof() function to check the storage class of value based on its format. Let's look at the examples below. As a side note, you don't always need to use FROM table\_name clause if not needed.

query:

SELECT

typeof('100'),

typeof(100),

typeof(10.0),

typeof(x'1000'),

typeof(NULL);

output:

typeof('100') typeof(100) typeof(10.0) typeof(x'1000') typeof(NULL)

------------- ----------- ------------ --------------- ------------

text integer real blob null

As you clearly see that the format of the values determines the storage class of the values. You may think that the result of typeof('100') is an integer. But there are enclosed single quotes in 100. Remember the rule that if a value is enclosed with single quote or double quotes, it is stored as text.  
  
Alright, it's time to start DDL commands. We'll start with CREATE.

## CREATE TABLE

**Done: View**

✔ [Introduction](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1335)  
✔ [Example](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1450)  
✔ [Constraints](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1609)  
✔ [Primary Key](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1452)  
✔ [Foreign Key](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1608)  
✔ [Not Null](https://lms.clarusway.com/mod/lesson/view.php?id=874&pageid=1712)

### Introduction

In this lesson, we will learn how to create a new table in the database. When creating a table, we use CREATE TABLE statement. Here is the syntax:

CREATE TABLE database\_name.table\_name (

column1 data\_type,

column2 data\_type,

...

);

A CREATE TABLE statement specifies the following attributes of the new table:

* The name of the new table
* The database in which the new table is created. This is optional. You don't have to specify. For instance, we could have typed CREATE TABLE table\_name (...);
* The name of the columns
* The declared data type of each column

Now, let's create a table in the *students*database. The *students* database has only one table named *student\_table*. If you type the .table command in the playground, the table name(s) of the database will return. .table command is special to SQLite.

query:

.table

output:

student\_table

Alright, we'll now add a table to the *students* database in the next part.

Example

We will add a table named *teachers*to our students database.Each teacher has the following information:

* Name
* Lesson
* Salary

Let's create the table.

query:

CREATE TABLE teachers (

name TEXT,

lesson TEXT,

salary INT

);

Now, display the tables in the students database. We must have two tables: student\_table, teachers. query:

.table

output:

student\_table teachers

**⚠️Avoid:**

* Don't try to run DDL commands on the playground. Due to some technical limitations, DDL commands may not work properly. However, you can run DDL commands using DB Browser for SQLite program installed on your local machines. You may also use other RDBMS.

### Constraints

Constraints are the rules specified for data in a table. We can limit the type of data that will go into a table with the constraints. We can define the constraints with the CREATE TABLE statement or ALTER TABLE statement.

Syntax

CREATE TABLE table\_name (

column1 DATATYPE CONSTRAINT,

column1 DATATYPE CONSTRAINT,

...

);

The following are the most commonly used constraints in SQL:

**Constraints**

| **Constraint Name** | **Definition** |
| --- | --- |
| NOT NULL | Ensures that a column cannot have a NULL value |
| DEFAULT | Sets a default value for a column when no value is specified |
| UNIQUE | Ensures that all values in a column are different |
| PRIMARY KEY | Uniquely identifies each row in a table |
| FOREIGN KEY | Uniquely identifies a row/record in another table |

We will start with the PRIMARY KEY constraint.

Q: What are Constraints in SQL?  
A: **NOT NULL** - Restricts NULL value from being inserted into a column. **DEFAULT** - Automatically assigns a default value if no value has been specified for the field. **UNIQUE** - Ensures unique values to be inserted into the field. **PRIMARY KEY** - Uniquely identifies each record in a table. **FOREIGN KEY** - Ensures referential integrity for a record in another table.

### Primary Key

The primary key is a column in our table that makes each row (aka, record) unique.

**💡 Tips:** The primary key is used to uniquely identify each record

A table can have only one primary key. The primary key can consist of one or multiple columns. We define the PRIMARY KEY in the CREATE TABLE statement.  
  
Here is the syntax:

CREATE TABLE table\_name(

column\_1 INT PRIMARY KEY,

column\_2 TEXT,

...

);

It's also possible to define the primary key at the end of the statement. It's very simple syntax: we just say PRIMARY KEY and put the column name inside the parentheses.

CREATE TABLE table\_name(

column\_1 INT,

column\_2 TEXT,

...

PRIMARY KEY (column\_1)

);

We define the primary key to uniquely identify each record. That means the value in the primary key column cannot be repeated. Thus we should choose an appropriate column that meets the criteria of the primary key.  
Let's take the example of employees table. Which column do you think as the best fit for the primary key?   
The answer is emp\_id. Because it has no repetition of data and uniquely identifies each record. In this table, each record represents an employee.  
  


The primary key column cannot contain NULL values. You don't get an error when you don't define a primary key in your table. However, good database design starts with a primary key. In fact, our tables in the database need a primary key. That will ensure row-level accessibility. With the primary key, we can query each table row individually.

Suppose that we don't have any tables yet in our company database. Let's create the employees table from scratch and define a primary key.

CREATE TABLE employees(

emp\_id INT NOT NULL PRIMARY KEY,

first\_name TEXT,

last\_name TEXT,

salary INT,

job\_title TEXT,

gender TEXT,

hire\_date TEXT);

You may notice that we use NOT NULL constraint with the primary key. But it's not mandatory to use it with the primary key since the primary key is implicitly NOT NULL. 

Q: What is a Primary Key?  
A: The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.

Foreign Key

Foreign key is a column in a table that uniquely identifies each row of another table. That column refers to a primary key of another table. This creates a kind of link between the tables.

Assume that we have a database called *store*. Let's create two tables as customers and orders inside the *store*database.

query:

CREATE TABLE customers (customer\_id INT PRIMARY KEY,

first\_name TEXT,

second\_name TEXT);

Now, we will create the second table called *orders.* In the orders table, we defined the FOREIGN KEY for the customer\_id column and pointed it using REFERENCES to the customer\_id column in the *customers* table.  
  
query:

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

order\_number INT,

customer\_id INT,

FOREIGN KEY (customer\_id)

REFERENCES customers (customer\_id)

);

* The customer\_id column in the customers table is the PRIMARY KEY in the customers table.
* The customer\_id column in the orders table is the FOREIGN KEY in the orders table.
* The customers table is called a parent table.
* The orders table is called a child table.

Let's insert some values into our newly created tables. We use INSERT INTO statement to add data. The syntax is very straightforward.

INSERT INTO table\_name (column1, column2)

VALUES (value1, value2);

We start to enter values to the customers table first.

INSERT INTO customers (customer\_id, first\_name, last\_name)

VALUES (1, 'Robert', 'Cursor'),

(2, 'Julia', 'Loyds'),

(3, 'Jack', 'Morgten');

Now, let's do it for the orders table.

INSERT INTO orders (order\_id, order\_number, customer\_id)

VALUES (1, 101, 3),

(2, 102, 3),

(3, 103, 2),

(4, 104, 1);

But, what did we achieved by defining FOREIGN KEY? Using foreign keys ensures that not to destroy the links between the tables. Besides, it prevents invalid data entry into the foreign key column. For instance, you cannot add a new customer\_id value to the orders table.

query:

INSERT INTO orders (order\_id, order\_number, customer\_id)

VALUES (5, 105, 4);

If you run this query, you get a "FOREIGN KEY constraint failed" error. This is also called referential integrity.  Since there is no customer\_id '4' in the customers table, we cannot create a new customer\_id in the orders table.  
 

Q:What is a Foreign Key?  
A: A FOREIGN KEY comprises of single or collection of fields in a table that essentially refer to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables. The table with the foreign key constraint is labelled as the child table, and the table containing the candidate key is labelled as the referenced or parent table.

### Not Null

A column can include NULL values. A NULL value is a special value that means the value is unknown or does not exist. We can specify if a column can hold NULL values or not when we create a table. By default, all columns (except primary key's column) in a table can hold NULL values unless we explicitly specify NOT NULL constraints.

We define NOT NULL constraint for a column using the following syntax:

CREATE TABLE table\_name (

column\_name type\_name NOT NULL,

...);

The NOT NULL constraint follows the type of the column as you see in the syntax.  
Let's give an example. Suppose that order\_number column cannot hold NULL values. That means every order should have an order number.

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

order\_number INT NOT NULL,

customer\_id INT,

FOREIGN KEY (customer\_id)

REFERENCES customers (customer\_id)

);

Let's try to insert a null value to the customer\_id.

INSERT INTO orders (order\_number)

VALUES (NULL);

If you run the query, you will get a "NOT NULL constraint failed" error.

### ALTER TABLE Statement

Unlike other database systems, SQLite supports a limited functionality of the ALTER TABLE statement.

With SQLite ALTER TABLE statement, you can perform two operations:

* Rename a table
* Add a new column to a table

##### Renaming a Table:

To rename a table, we use the following ALTER TABLE RENAME TO statement syntax.

ALTER TABLE existing\_table

RENAME TO new\_table;

##### Adding a New Column to a Table:

Another operation you can perform with the SQLite ALTER TABLE statement is to add a new column to an existing table. In this case, SQLite appends the new column at the end of the existing columns list.

Here is the syntax of ALTER TABLE ADD COLUMN statement:

ALTER TABLE table

ADD COLUMN column\_definition;

### DROP TABLE Statement

We use DROP TABLE statement to remove a table in a database. The syntax is very straightforward.

DROP TABLE table\_name;

Let's remove the *orders* table from the *store* database and show the tables left in the store database.  
  
query:

DROP TABLE orders

.table;

output:

customers

As you see that we have only one table left in our database. we have successfully removed the *orders* table.