**SQL Fundamentals**

**Getting Started with Databases**

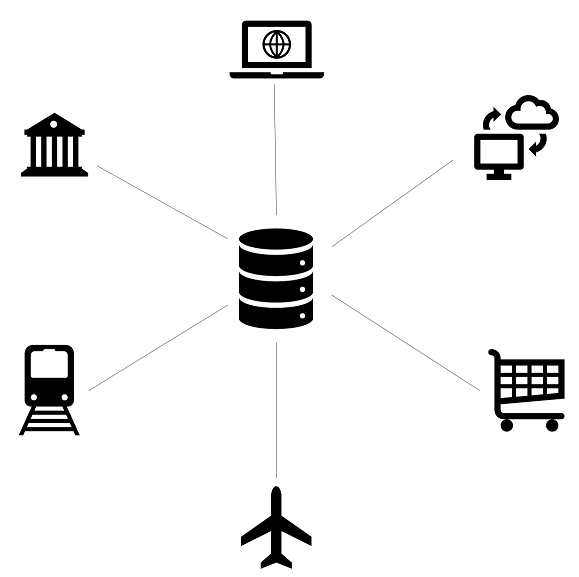
**What is a database?**

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 |

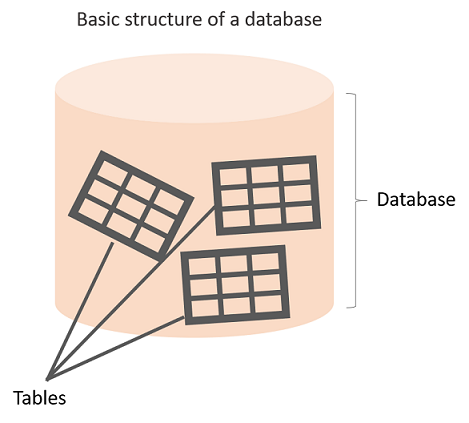
**☍ Discussion:** Think about other applications of databases. Try to give some examples.

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.

- Interview Q&A

**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.



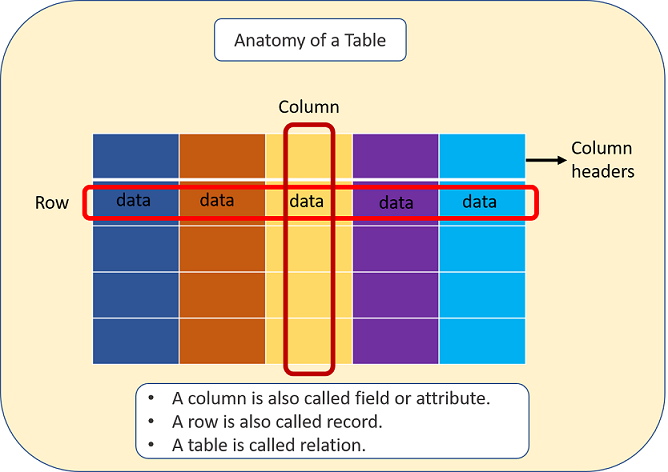
**Anatomy of a Table**

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.

- Interview Q&A

**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)(Yapılandırılmış Sorgu Dili) 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.

**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.

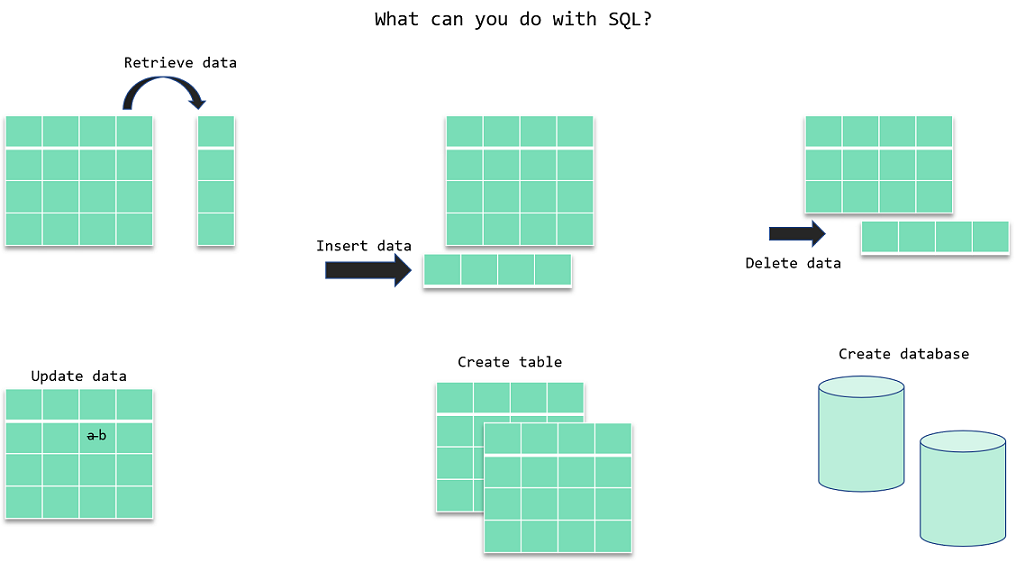
**💡Tips:**

* 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(ek, ilave) 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.

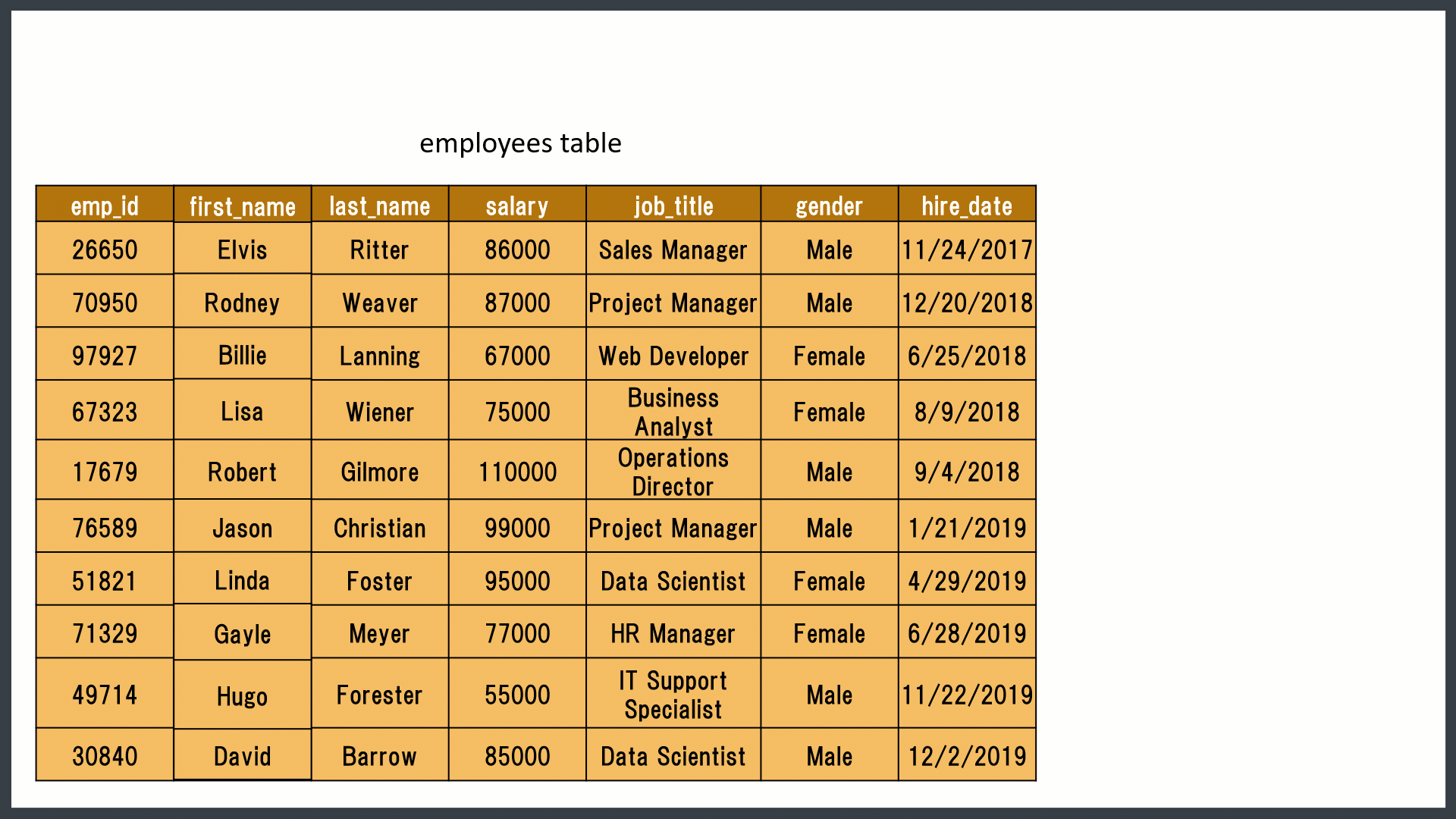
- Interview Q&A

SQL Language Elements

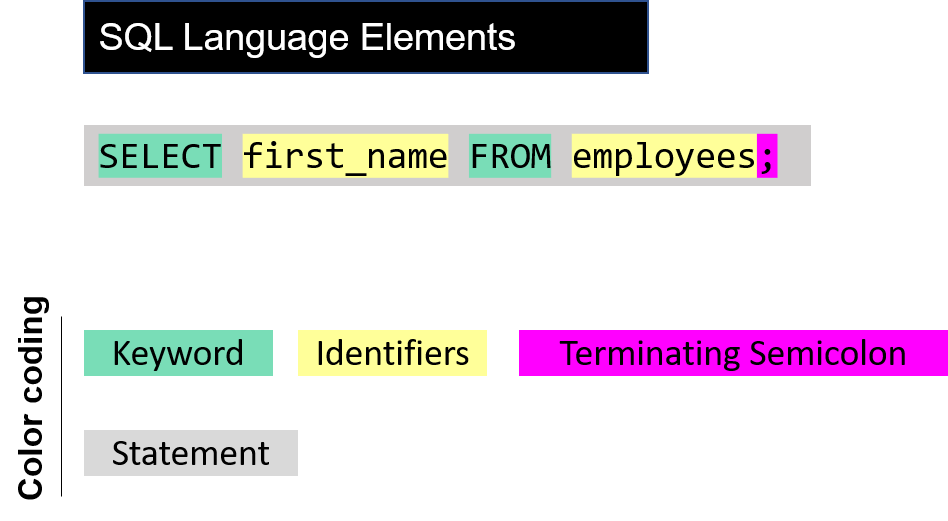
SQL language structure is easy to understand. It looks like plain English. Here is an example SQL statement:

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*.



***SQL Playground***

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named employees (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**employees**

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

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 .



***SQL Playground***

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named employees (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**employees**

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

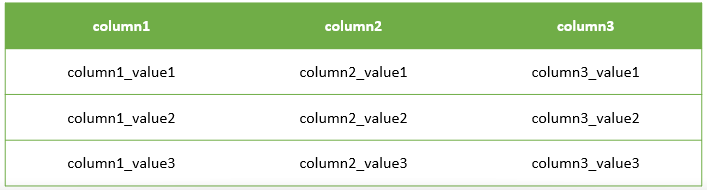
---------- ---------- ---------- ---------- ------------------- ---------- ----------

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

**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



***SQL Playground***

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named table1 (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**table1**

column1 column2 column3

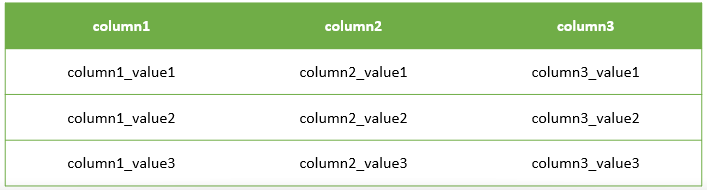
-------------- -------------- --------------

column1\_value1 column2\_value1 column3\_value1

column1\_value2 column2\_value2 column3\_value2

column1\_value3 column2\_value3 column3\_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.



***SQL Playground***

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named table1 (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**table1**

column1 column2 column3

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

column1\_value1 column2\_value1 column3\_value1

column1\_value2 column2\_value2 column3\_value2

column1\_value3 column2\_value3 column3\_value3

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(Madde):**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.

### *SQL Playground*

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named employees (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**employees**

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

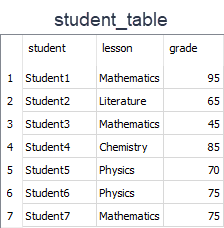
---------- ---------- ---------- ---------- ------------------- ---------- ----------

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

**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.

### *SQL Playground*

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named student\_table (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**student\_table**

student lesson grade

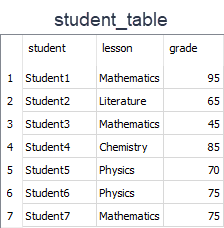
---------- ------------ ----------

Student1 Mathematics 95

Student2 Literature 65

Student3 Mathematics 45

**Duplicated Rows**

****

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.

| **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

### *SQL Playground*

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named student\_table (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**student\_table**

student lesson grade

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

Student1 Mathematics 95

Student2 Literature 65

Student3 Mathematics 45

**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

**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

**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**

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.

### *SQL Playground*

Hi **D1245 -**, you can write your SQL script using the editor provided.  
There is only one table named employees (only 3 rows are shown below) in your database. You can use SQL statements within your database. Once you write the script, click the run button to see the result.

**employees**

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

**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.

**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

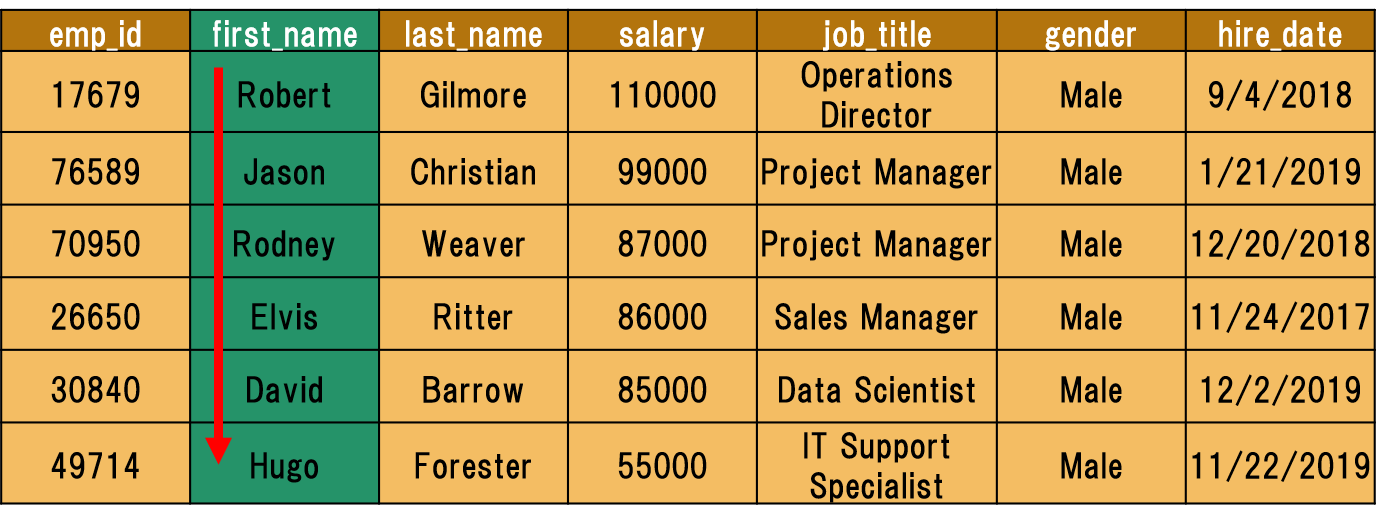
51821 Linda Foster 95000 Data Scientist Female 4/29/2019

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 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

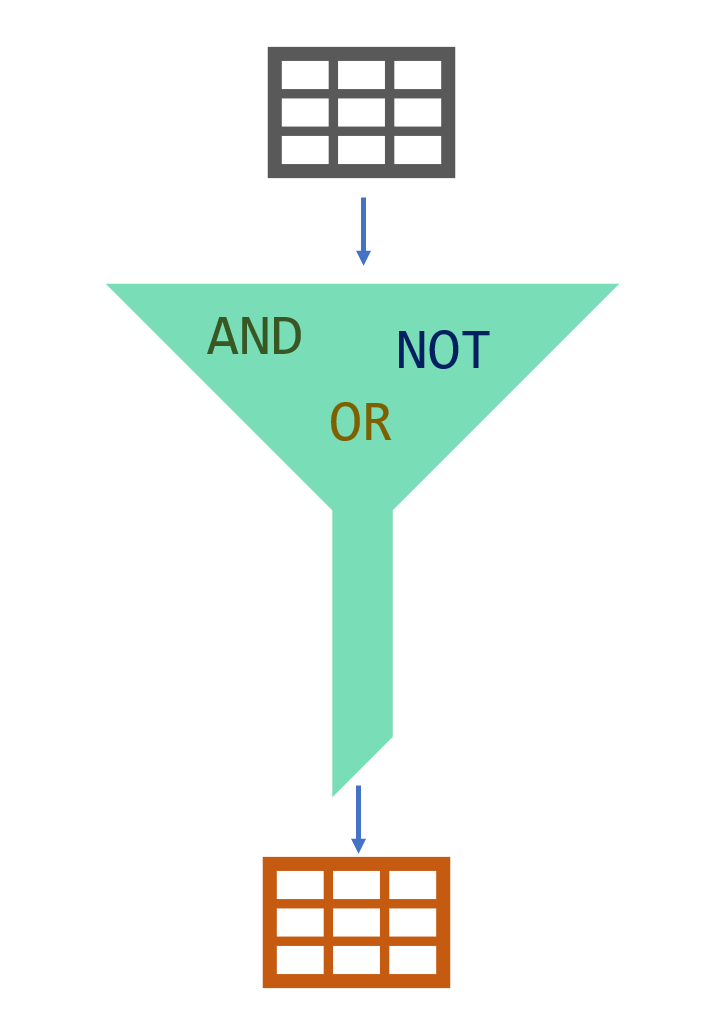
We first returned the employees whose salary is higher than $80,000. Next, we sorted this by the first names in descending order. Alright, we now know how to sort our table. It's time to put them into practice.

**AND, OR & NOT Operators**

Introduction

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 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

**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**

**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(kapsayan). 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.

**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 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.  
Now, it's time to put your theory into practice. Enjoy your playground. Feel free to write as many queries as you wish.

**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.

**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.

Please write all the queries in the playground.

**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

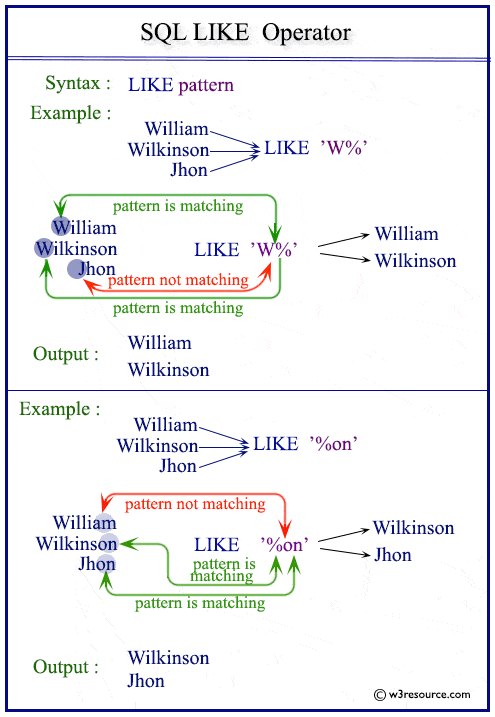
WHERE column\_1 LIKE pattern;

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

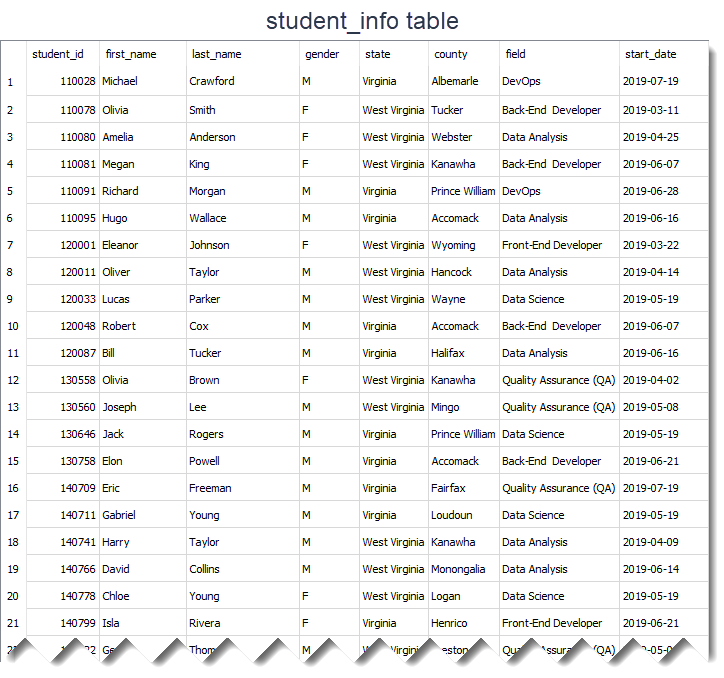
* **Percent (%):**The % character matches any sequence of zero or more characters.
* **Underscore ( \_ ):** The \_ character matches any single character.

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%';

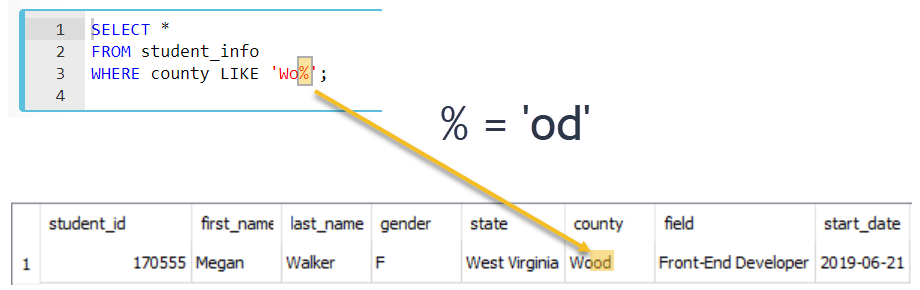
Result table:  
  
output :

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.

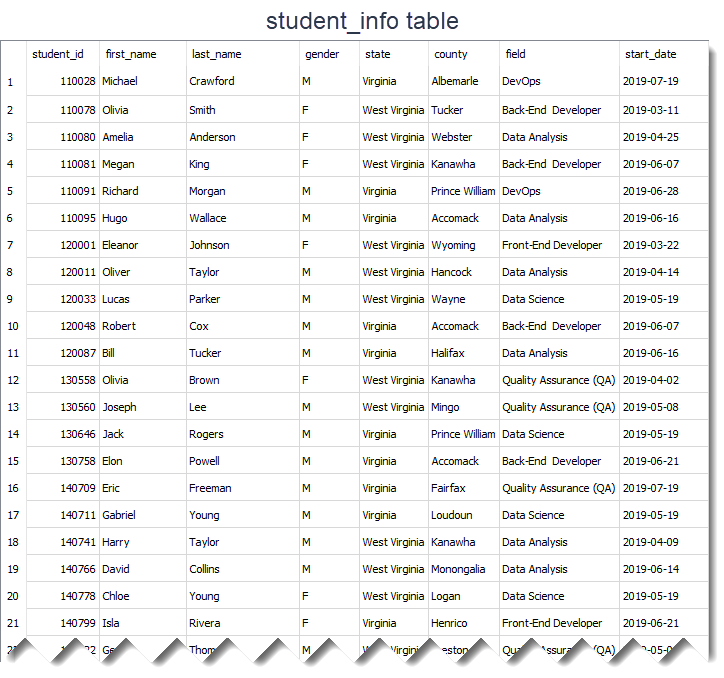


Since LIKE operator is case insensitive, you could write the pattern LIKE 'wo%' . Both patterns will produce the same results.

**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.