## Drag and drop from the options below to build a query to get distinct results from the ''customers'' table.

SELECT DISTINCT column\_name1, column\_name2 FROM table\_name;

**Drag and drop from the options below to complete the following statement, which selects five names from ''students''.**

SELECT column list FROM table\_name LIMIT [number of records];

You can also pick up a set of records from a particular **offset**. In the following example, we pick up **four**records, starting from the **third**position:

SELECT column list FROM table\_name LIMIT 2,4;  
  
In SQL, you can provide the table name prior to the column name, by separating them with a **dot**.  
**The following statements are equivalent:**

SELECT City FROM customers;   
  
SELECT **customers.**City FROM customers;

ORDER BY is used with SELECT to sort the returned data.

select \* from customers  
order by FirstName;

ORDER BY can sort retrieved data by multiple columns. When using ORDER BY with more than one column, separate the list of columns to follow ORDER BY with **commas**.

Тoday a customer want a cake set that has minimal calories.  
Write a query to sort the cakes by calorie count and select the first 3 cakes from the list to offer the customer.  
**select \* from cakes order by calories limit 3 ;**

Write a query to output the **names** of employees whose salaries are **between 1500 and 1900**.

select firstname from staff where salary between 1500 and 1900;  
  
Write a query to output the **firstname** and **lastname** columns into one column named **fullname** separated by space, and the total annual salary for each employee keeping in mind bonuses named '**total'.** Sort by the **'total'** column.

select concat (firstname , ' ', lastname ) as fullname,

salary\*12 + experience\*500 as total

from staff

order by total

The **UPPER**function converts all letters in the specified string to uppercase.  
The **LOWER**function converts the string to lowercase.  
The **SQRT** function returns the square root of given value in the argument.  
The **SUM**function is used to calculate the sum for a column's values.

Write a query to output the **average** of Sam's exam scores for **the first semester**.  
select avg(score) from sam\_grades where semester=1

A **subquery**is a query within another query.  
The **DESC**keyword sorts results in **descending**order. Similarly, **ASC**sorts the results in **ascending**order.  
SELECT FirstName, Salary FROM employees   
WHERE  Salary > 3100  
ORDER BY Salary DESC;

**A single subquery will return the same result more easily.**

SELECT FirstName, Salary FROM employees

WHERE Salary > (SELECT AVG(Salary) FROM employees)

ORDER BY Salary DESC;

Enclose the subquery in **parentheses**.  
Also, note that there is no semicolon at the end of the subquery, as it is part of our single query.

Help Monica lose weight by writing a query to choose the foods, whose fat percentages are lower than the average from the "Foods" table. Then show the resulting table.  
select \*   
from Foods   
where fatpercentage<(select avg(fatpercentage)from Foods ) ;

The **LIKE**keyword is useful when specifying a **search condition** within your WHERE clause.

SQL **pattern**matching enables you to use "\_" to match any single character and "%" to match an arbitrary number of characters (including zero characters).  
For example, to select employees whose *FirstNames*begin with the letter **A**, you would use the following query:  
SELECT \* FROM employees   
WHERE FirstName LIKE 'A%';

all employees with a *LastName*ending with the letter "s  
SELECT \* FROM employees   
WHERE LastName LIKE '%s';

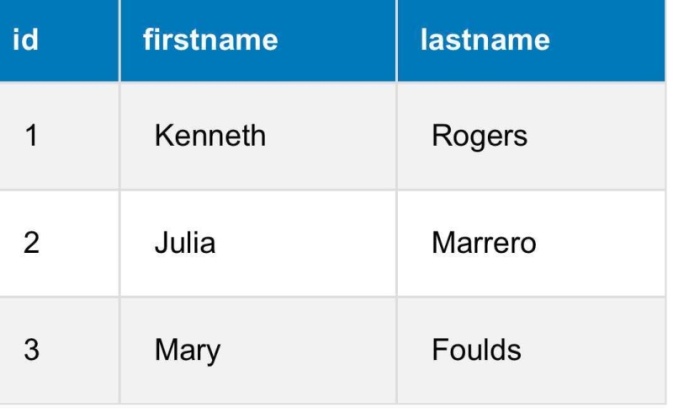
Write a query to output only **chocolate** desserts.  
select \*  
from desserts   
where name like '%Chocolate%';

Write a query to output the apartments whose prices **are greater than the average** and are also **not rented**, sorted by the 'Price' column.  
select \*  
from Apartments  
where price >= (select avg(price)from Apartments)   
and status in ('Not rented')   
order by price;

In SQL, "**joining tables**" means combining data from two or more tables. A table join creates a **temporary table** showing the data from the joined tables.

To join the two tables, specify them as a comma-separated list in the FROM clause:  
SELECT customers.ID, customers.Name, orders.Name, orders.Amount  
FROM customers, orders  
WHERE customers.ID=orders.Customer\_ID  
ORDER BY customers.ID;

You are given the following **students** and **teachers** tables  
**students**(with their teachers ID's):



Write a query to output all of the students with their teachers' last names in one table, sorted by students ID.  
select students.id, students.firstname, students.lastname, teachers.lastname as teacher  
from students, teachers  
where students.teacherid=teachers.id  
order by students.id

Custom names can be used for tables as well. You can shorten the join statements by giving the tables "nicknames":

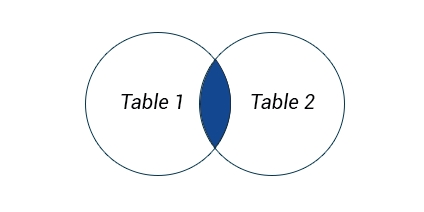
**- INNER** JOIN  
**- LEFT** JOIN  
**- RIGHT** JOIN

INNER JOIN is equivalent to JOIN. It returns rows when there is a match between the tables.

SELECT column\_name(s)

FROM table1 INNER JOIN table2

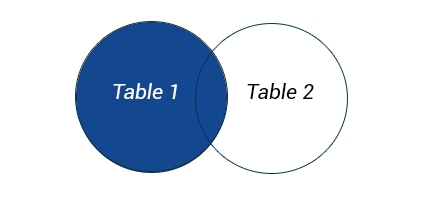
ON table1.column\_name=table2.column\_name;



Only the records matching the join condition are returned.

Write a query to output all products with their categories (productname-price-categoryname) in one table.  
select pd.productname, pd.price, ct.categoryname  
from products as pd inner join categories as ct  
on pd.categoryid=ct.id ;

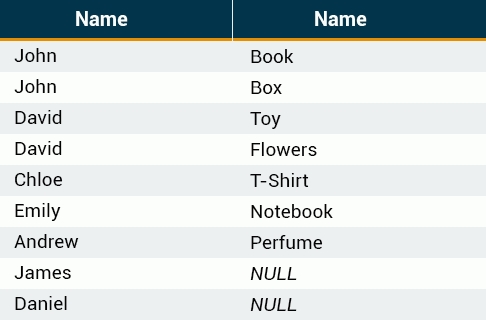
The **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table.  
This means that if there are no matches for the **ON**clause in the table on the right, the join will still return the rows from the first table in the result.

  
SELECT table1.column1, table2.column2...

FROM table1 LEFT OUTER JOIN table2

ON table1.column\_name = table2.column\_name; The **OUTER**keyword is optional, and can be omitted.  
The following SQL statement will return all **customers**, and the **items**they might have:

SELECT customers.Name, items.Name   
FROM customers LEFT OUTER JOIN items   
ON customers.ID=items.Seller\_id;



If no match is found for a particular row, **NULL**is returned.

The **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table.  
SELECT customers.Name, items.Name FROM customers  
RIGHT JOIN items ON customers.ID=items.Seller\_id;

There are other types of joins in the SQL language, but they are not supported by MySQL.  
Set Operation

Occasionally, you might need to combine data from multiple tables into one comprehensive dataset. This may be for tables with similar data within the same database or maybe there is a need to combine similar data across databases or even across servers.  
To accomplish this, use the **UNION**and **UNION ALL** operators.  
**UNION**combines multiple datasets into a single dataset, and removes any existing duplicates.  
**UNION ALL** combines multiple datasets into one dataset, but does not remove duplicate rows.

UNION ALL is faster than UNION, as it does not perform the duplicate removal operation over the data set.

The **UNION**operator is used to combine the result-sets of two or more SELECT statements.  
  
All SELECT statements within the UNION must have the **same number of columns**. The columns must also have the same**data types**. Also, the columns in each SELECT statement must be in the same order.  
**The syntax of UNION is as follows:**SELECT ID, FirstName, LastName, City FROM FirstUNIONSELECT ID, FirstName, LastName, City FROM Second;

**TIP:**  
If your columns don't match exactly across all queries, you can use a **NULL (or any other)** value such as:

**SELECT FirstName, LastName, Company FROM businessContacts**

**UNION**

**SELECT FirstName, LastName, NULL FROM otherContacts**

Write a query to merge 'NorwayChess' and 'TataSteel' tables. Then order the merged table by 'Rating' in descending order and show the final table of participants.  
select \* from NorwayChess  
UNION  
select \* from TataSteel order by rating desc;

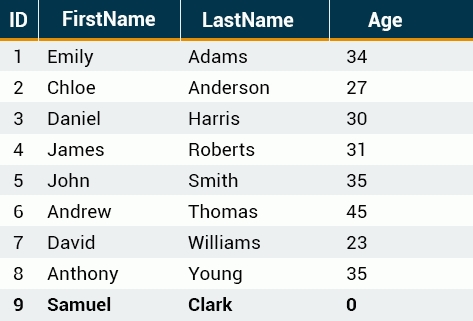
**UNION ALL** selects all rows from each table and combines them into a single table.

When inserting records into a table using the SQL INSERT statement, you must provide a value for every column that does not have a default value, or does not support NULL.  
insert into Garage

values(6,    'Mercedes-Benz',   'G 63',   2020), (7, 'Porsche', 'Panamera', 2020);  
select \* from Garage ;

Alternatively, you can specify the table's column names in the INSERT INTO statement  
INSERT INTO Employees (ID, FirstName, LastName, Age)  
VALUES (8, 'Anthony', 'Young', 35);  
SELECT \* FROM Employees;

It is also possible to insert data into **specific** columns only.  
INSERT INTO Employees (ID, FirstName, LastName)   
VALUES (9, 'Samuel', 'Clark');  
SELECT \* from Employees;



The *Age*column for that row automatically became **0**, as that is its default value

The **UPDATE**statement allows us to alter data in the table.

UPDATE table\_name

SET column1=value1, column2=value2, ...

WHERE condition;

If you omit the WHERE clause, **all** records in the table will be updated!

UPDATE Employees

SET Salary=5000, FirstName='Robert'

WHERE ID=1;

SELECT \* from Employees;

You can specify the column order any way you like in the SET clause

The **DELETE**statement is used to remove data from your table. DELETE queries work much like UPDATE queries.

DELETE FROM Employees

WHERE ID=1;

SELECT \* from Employees;

If you omit the WHERE clause, **all**records in the table will be deleted!  
The DELETE statement removes the data from the table permanently.

The **CREATE TABLE** statement is used to create a new table.

Creating a basic table involves naming the table and defining its columns and each column's data type.

Assume that you want to create a table called "Users" that consists of four columns: UserID, LastName, FirstName, and City.  
Use the following **CREATE TABLE** statement:

**CREATE TABLE Users**

**(**

**UserID int,**

**FirstName varchar(100),**

**LastName varchar(100),**

**City varchar(100)**

**);**

**varchar**is the datatype that stores characters. You specify the number of characters in the parentheses after the type. So in the example above, our fields can hold max **100**characters long text.

**The most common data types:**  
**Numeric**  
**INT**-A normal-sized integer that can be signed or unsigned.  
**FLOAT**(M,D) - A floating-point number that cannot be unsigned. You can optionally define the display length (M) and the number of decimals (D).  
**DOUBLE**(M,D) - A double precision floating-point number that cannot be unsigned. You can optionally define the display length (M) and the number of decimals (D).  
  
**Date and Time**  
**DATE**- A date in *YYYY-MM-DD* format.  
**DATETIME**- A date and time combination in YYYY-MM-DD HH:MM:SS format.  
**TIMESTAMP**- A timestamp, calculated from midnight, January 1, 1970  
**TIME**- Stores the time in HH:MM:SS format.  
  
**String Type**  
**CHAR**(M) - Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.  
**VARCHAR**(M) - Variable-length character string. Max size is specified in parenthesis.  
**BLOB -**"Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files.  
**TEXT**- Large amount of text data.

# Primary Key

CREATE TABLE Users

(

UserID int,

FirstName varchar(100),

LastName varchar(100),

City varchar(100),

PRIMARY KEY(UserID)

);

Specify the column name in the parentheses of the PRIMARY KEY keyword.

create table leaderboard

(

    place int not null,

    nickname VARCHAR(50) not null,

    rating VARCHAR(50) not null,

    primary key (nickname)

);

insert into leaderboard

values (1, 'Predator', 9500),(2, 'JohnWar', 9300),(3, 'NightWarrior', 8900);

select \* from leaderboard ;

SQL **constraints**are used to specify rules for table data.  
  
**The following are commonly used SQL constraints:**  
**NOT NULL** - Indicates that a column cannot contain any NULL value.  
**UNIQUE**- Does not allow to insert a duplicate value in a column. The UNIQUE constraint maintains the uniqueness of a column in a table. More than one UNIQUE column can be used in a table.  
**PRIMARY KEY** - Enforces the table to accept unique data for a specific column and this constraint create a unique index for accessing the table faster.  
**CHECK** - Determines whether the value is valid or not from a logical expression.  
**DEFAULT** - While inserting data into a table, if no value is supplied to a column, then the column gets the value set as DEFAULT.

name varchar(100) **NOT NULL**

Auto-increment allows a unique number to be generated when a new record is inserted into a table.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.  
Let's set the UserID field to be a primary key that automatically generates a new value:

UserID int NOT NULL **AUTO\_INCREMENT**,   
PRIMARY KEY (UserID)

Auto-increment allows a unique number to be generated when a new record is inserted into a table.

The example below demonstrates how to create a table using constraints.

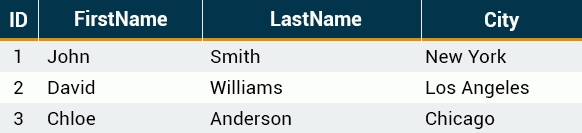
CREATE TABLE Users (   
id int **NOT NULL AUTO\_INCREMENT**,   
username varchar(40) **NOT NULL**,   
password varchar(10) **NOT NULL**,   
**PRIMARY KEY(id)**  
);

The following SQL enforces that the "id", "username", and "password" columns do not accept NULL values. We also define the "id" column to be an auto-increment primary key field.



When inserting a new record into the Users table, it's not necessary to specify a value for the id column; a unique new value will be added automatically.

The **ALTER TABLE** command is used to add, delete, or modify columns in an existing table.  
You would also use the ALTER TABLE command to add and drop various constraints on an existing table.



The following SQL code adds a new column named **DateOfBirth**

ALTER TABLE People ADD DateOfBirth date;

SELECT \* from People;

The following SQL code demonstrates how to delete the column named *DateOfBirth*in the People table.

ALTER TABLE People

DROP COLUMN DateOfBirth;

SELECT \* from People;

The column, along with all of its data, will be completely removed from the table.

To delete the entire table, use the **DROP TABLE** command:

**DROP TABLE** People;

Be careful when dropping a table. Deleting a table will result in the complete loss of the information stored in the table!

# **Renaming**

The ALTER TABLE command is also used to rename columns:

ALTER TABLE People

RENAME FirstName TO name;

SELECT \* from People;

This query will rename the column called FirstName to name.

**Add** a new column 'AttractivePlace' and **update** the values:  
- 'Belem Tower' for Lisbon  
- 'Plaza Mayor' for Madrid  
- 'Eiffel Tower' for Paris  
Then show the resulting table.

ALTER TABLE cities ADD AttractivePlace varchar(20);

update cities

set  AttractivePlace='Belem Tower' where name ='Lisbon';

update cities

set AttractivePlace='Plaza Mayor'where name ='Madrid';

update cities

set AttractivePlace='Eiffel Tower'where name ='Paris';

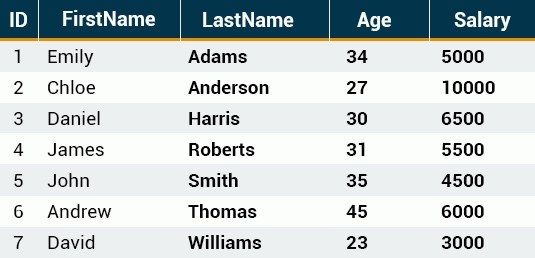
SELECT \* from cities;

In SQL, a VIEW is a **virtual table** that is based on the result-set of an SQL statement.  
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.  
Views allow us to:  
- Structure data in a way that users or classes of users find natural or intuitive.  
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.  
- Summarize data from various tables and use it to generate reports.

**CREATE VIEW** view\_name **AS**  
SELECT column\_name(s)   
FROM table\_name   
WHERE condition;

The SELECT query can be as complex as you need it to be. It can contain multiple JOINS and other commands.

Consider the **Employees**table, which contains the following records:



Let's create a view that displays each employee's FirstName and Salary.

**CREATE VIEW** List **AS**  
SELECT FirstName, Salary   
FROM Employees;

Now, you can query the **List**view as you would query an actual table.

CREATE VIEW List AS

SELECT FirstName, Salary

FROM  Employees;

SELECT \* FROM List;

A view always shows up-to-date data! The database engine uses the view's SQL statement to recreate the data each time a user queries a view.

Write a query to create a view to show only 'acc\_id' and 'status' columns and then show that view.

create view list as

select acc\_id, status

from users ;

select \* from list

 A new animal has come in, with the following details:  
name - "Slim", type - "Giraffe", country\_id - 1  
Add him to the Animals table.  
2) You want to make a complete list of the animals for the zoo’s visitors. Write a query to output a new table with each animal's name, type and country fields, sorted by countries.

insert into animals

values ('Slim','Giraffe',1);

SELECT animals.name, animals.type, countries.country FROM animals, countries

WHERE animals.country\_id=countries.id order by animals.country\_id desc