1. create table table\_name(column1 data type,………..);

create table t\_name(

id integer primary key,

name text unique,

date\_of birth text not null;

data\_of\_death text default “not applicable”;);

1. insert into table\_name(column1, column2..)

values(c\_1, c\_2…);

1. insert into celebs(id, name, age)

values(2, “Beyonce Knowles”, 33);

1. update table\_name

set column\_i=..

where id=3;

1. alter table table\_name add column

new\_column\_name data\_type;

1. delete from celebs where col is null;(delete all rows whose col is null)
2. select col\_name as “….” From table\_name (rename column in the result,does not change the original table)
3. select distrinct col from table;
4. select \* from movies where name like “\_oo”;

like “A%” ”%man%”

is null

between 20 and 30(inclusive on both sides)

between “a” and “j” ( does not include the end point)

1. select \* from movies

where imdb\_rating>8

order by year desc/asc;

1. limit 10(end of clause, number of rows shown)
2. **SELECT** player\_name,

weight,

**CASE** **WHEN** weight > 250 **THEN** 'over 250'

**WHEN** weight > 200 **THEN** '201-250'

**WHEN** weight > 175 **THEN** '176-200'

**ELSE** '175 or under' **END** **AS** weight\_group

**FROM** benn.college\_football\_players

1. Aggregate: count, max/min, avg, round(col, decimal), sum
2. SELECT year, AVG(imdb\_rating) FROM movies GROUP BY year ORDER BY year;
3. select price,count(\*)

from fake\_apps

where downloads>20000

group by price;

* When we want to limit the results of a query based on an aggregate property, we use HAVING.

HAVING statement always comes after GROUP BY, but before ORDER BY and LIMIT

1. SELECT \* FROM orders JOIN customers ON orders.customer\_id = customers.customer\_id
2. SELECT shirts.shirt\_color, pants.pant\_color FROM shirts CROSS JOIN pants;

(all combinations of columns)

1. Cross join , left/right join, inner join, join.. on.., union(combine rows)

select \* from newspaper

union

select \* from online;

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each SELECT statement must also be in the same order

1. with previous\_query as (

select customer\_id, count(subscription\_id) as

subscriptions

from orders

group by customer\_id)

select customers.customer\_name, previous\_query.subscriptions

from customers join previous\_query

on customers.customer\_id=previous\_query.customer\_id;

1. SELECT CITY,LENGTH(CITY) AS LENGTH

FROM STATION

ORDER BY LENGTH(CITY),CITY

LIMIT 1;

SELECT CITY,LENGTH(CITY) AS LENGTH

FROM STATION

ORDER BY LENGTH(CITY) DESC,CITY

LIMIT 1;

1. **SELECT** **\*** **FROM** baby\_names

**WHERE** **state** **IN**('CA', 'NY', 'TX');

**SELECT** **\*** **FROM** baby\_names

**WHERE** **state** **NOT** **IN**('CA', 'NY', 'TX');

1. SELECT DISTINCT city FROM station WHERE city REGEXP "^[aeiou].\*";

WHERE city REGEXP '[aeiou]$';

WHERE CITY REGEXP "^[aeiou].\*[aeiou]$";

WHERE CITY REGEXP '^[^aeiou]'

WHERE CITY REGEXP '^[^aeuio]' OR CITY REGEXP '[^aeuio]$'

1. SELECT NAME FROM STUDENTS WHERE MARKS > 75

ORDER BY RIGHT(NAME, 3), ID ASC;

1. SELECT CONCAT(NAME,'(',Substring(Occupation,1,1),')') FROM OCCUPATIONS

order by name;

select concat("There are a total of ",count(name)," ",lcase(occupation),"s.") from occupations

group by occupation

order by count(name) asc, occupation asc;

1. select CEIL(avg(salary)-avg(replace(salary,"0",""))) from employees
2. select months\*salary as earnings, count(\*) from employee

group by earnings

order by earnings desc

limit 1

1. power(number,power) for square
2. Select round(S.LAT\_N,4) median from station S where (select count(Lat\_N) from station where Lat\_N < S.LAT\_N ) = (select count(Lat\_N) from station where Lat\_N > S.LAT\_N)
3. select country.continent,floor(avg(city.population)) from city join country on city.countrycode=country.code

group by country.continent

1. select if(grade > 7, name, null), grade, marks

from students, grades

where marks between min\_mark and max\_mark

order by grade desc, name

1. select h.hacker\_id, h.name

from submissions s

inner join challenges c

on s.challenge\_id = c.challenge\_id

inner join difficulty d

on c.difficulty\_level = d.difficulty\_level

inner join hackers h

on s.hacker\_id = h.hacker\_id

where s.score = d.score and c.difficulty\_level = d.difficulty\_level

group by h.hacker\_id, h.name

having count(s.hacker\_id) > 1

order by count(s.hacker\_id) desc, s.hacker\_id asc

select host.name from

(select f.id, students.name, p.salary as host\_salary from friends as f

join packages as p

on f.ID=p.ID

join students

on f.id=students.id) as host

join

(select f.id as host\_id, f.friend\_id,p.salary as friend\_salary from friends as f

join packages as p

on f.friend\_id=p.ID

join students

on f.friend\_id=students.id) as friend

on

host.id=friend.host\_id

where host\_salary<friend\_salary

order by friend\_salary

;

<> not equal

Is null/is not null do not use <>

ORDER BY Country ASC, CustomerName DESC;

Insert: if not specified, Null

Don’t have to specify the primary key, will auto add 1 to the end of table

Insert multiple?

SELECT TOP 50 PERCENT \* FROM Customers;

SELECT TOP 20 \* FROM CUSTOMERS

Except for COUNT , **aggregate functions** ignore **null** values.

There are two wildcards used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

|  |  |
| --- | --- |
| LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

LIKE “A\_%\_%” AT LEAST 3 CHAR LENGTH

* [*charlist*] - Defines sets and ranges of characters to match
* [^*charlist*] or [!*charlist*] - Defines sets and ranges of characters NOT to match
* LIKE '[bsp]%'
* LIKE '[a-c]%';

LIKE “[!ABC]” === NOT LIKE [ABC]

The BETWEEN operator is inclusive: begin and end values are included.

OrderDate BETWEEN '04-July-1996' and '09-July-1996':

Self join:

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City

FROM Customers A, Customers B

WHERE A.CustomerID <> B.CustomerID

AND A.City = B.City

ORDER BY A.City;

Add a new column filled with “customer”

SELECT 'Customer' FROM Customers

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records.

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);

returns TRUE and lists the suppliers with a product price less than 20:

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

SELECT ProductName  
FROM Products  
WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

The SELECT INTO statement copies data from one table into a new table.

SELECT \* INTO CustomersGermany  
FROM Customers  
WHERE Country = 'Germany';

SELECT Customers.CustomerName, Orders.OrderID  
INTO CustomersOrderBackup2017  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

* INSERT INTO SELECT requires that data types in source and target tables match
* The existing records in the target table are unaffected

INSERT INTO *table2*(*column1*, *column2*, *column3*, ...)  
SELECT *column1*, *column2*, *column3*, ...  
FROM *table1*  
WHERE *condition*;

Null Functions:

IFNULL(Col, 0)

COALESCE(Col, 0)

NULLIF(Col,value) value->NULL

Databases:

CREATE DATABASE testDB;

DROP DATABASE testDB;

DROP TABLE Shippers;

Create table:

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)   
);

ALTER TABLE *table\_name*  
ADD *column\_name datatype*;

DROP COLUMN *column\_name*;

ALTER COLUMN DateOfBirth year; change data type

**MySQL** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* YEAR - format YYYY or YY

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;