SQL password:833283

CREATE DATABASE imdb;

USE imdb;

SHOW TABLES;

DESCRIBE movies;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SELECT \* FROM movies;

# more data transfer

#result-set: a set of rows that form the result of a query along with column-names and meta-data.

SELECT name,year FROM movies;

SELECT rankscore,name FROM movies;

#row order same as the one in the table

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LIMIT:

SELECT name,rankscore FROM movies LIMIT 20;

SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ORDER BY:

# list recent movies first

SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 10;

# default:ASC

SELECT name,rankscore,year FROM movies ORDER BY year LIMIT 10;

# the output row order maynot be same as the one in the table due to query optimzier and internal data-structres/indices.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISTINCT:

# list all genres of

SELECT DISTINCT genre FROM movies\_genres;

# multiple-column DISTINCT

SELECT DISTINCT first\_name, last\_name FROM directors;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WHERE:

# list all movies with rankscore>9

SELECT name,year,rankscore FROM movies WHERE rankscore>9 ;

SELECT name,year,rankscore FROM movies WHERE rankscore>9 ORDER BY rankscore DESC LIMIT 20;

# Condition's outputs: TRUE, FALSE, NULL

# Comparison Operators: = , <> or != , < , <= , >, >=

SELECT \* FROM movies\_genres WHERE genre = 'Comedy';

SELECT \* FROM movies\_genres WHERE genre <> 'Horror';

NULL => doesnot-exist/unknown/missing

# "=" doesnot work with NULL, will give you an empty result-set.

SELECT name,year,rankscore FROM movies WHERE rankscore = NULL;

SELECT name,year,rankscore FROM movies WHERE rankscore IS NULL LIMIT 20;

SELECT name,year,rankscore FROM movies WHERE rankscore IS NOT NULL LIMIT 20;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# LOGICAL OPERATORS: AND, OR, NOT, ALL, ANY, BETWEEN, EXISTS, IN, LIKE, SOME

# website search filters

SELECT name,year,rankscore FROM movies WHERE rankscore>9 AND year>2000;

SELECT name,year,rankscore FROM movies WHERE NOT year<=2000 LIMIT 20;

SELECT name,year,rankscore FROM movies WHERE rankscore>9 OR year>2007;

# will discsuss about ANY and ALL when we discuss sub-queries

SELECT name,year,rankscore FROM movies WHERE year BETWEEN 1999 AND 2000;

#inclusive: year>=1999 and year<=2000

SELECT name,year,rankscore FROM movies WHERE year BETWEEN 2000 AND 1999;

#lowvalue <= highvalue else you will get an empty result set

SELECT director\_id, genre FROM directors\_genres WHERE genre IN ('Comedy','Horror');

# same as genre='Comedy' OR genre='Horror'

SELECT name,year,rankscore FROM movies WHERE name LIKE 'Tis%';

# % => wildcard character to imply zero or more characters

SELECT first\_name, last\_name FROM actors WHERE first\_name LIKE '%es';

# first name ending in 'es'

SELECT first\_name, last\_name FROM actors WHERE first\_name LIKE '%es%';

#first name contains 'es'

SELECT first\_name, last\_name FROM actors WHERE first\_name LIKE 'Agn\_s';

# '\_' implies exactly one character.

# If we want to macth % or \_, we should use the backslash as the escape character: \% and \\_

SELECT first\_name, last\_name FROM actors WHERE first\_name LIKE 'L%' AND first\_name NOT LIKE 'Li%';

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Aggregate functions: Computes a single value on a set of rows and returns the aggreagate

COUNT, MIN, MAX, SUM, AVG

SELECT MIN(year) FROM movies;

SELECT MAX(year) FROM movies;

SELECT COUNT(\*) FROM movies;

SELECT COUNT(\*) FROM movies where year>2000;

SELECT COUNT(year) FROM movies;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GROUP-BY

# find number of movies released per year

SELECT year, COUNT(year) FROM movies GROUP BY year;

SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;

SELECT year, COUNT(year) year\_count FROM movies GROUP BY year ORDER BY year\_count;

# year\_count is an alias.

# often used with COUNT, MIN, MAX or SUM.

# if grouping columns contain NULL values, all null values are grouped together.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HAVING:

# Print years which have >1000 movies in our DB [Data Scientist for Analysis]

SELECT year, COUNT(year) year\_count FROM movies GROUP BY year HAVING year\_count>1000;

# specify a condition on groups using HAVING.

Order of execution:

1. GROUP BY to create groups

2. apply the AGGREGATE FUNCTION

3. Apply HAVING condition.

# often used along with GROUP BY. Not Mandatory.

SELECT name, year FROM movies HAVING year>2000;

# HAVING without GROUP BY is same as WHERE

SELECT year, COUNT(year) year\_count FROM movies WHERE rankscore>9 GROUP BY year HAVING year\_count>20;

# HAVING vs WHERE

## WHERE is applied on individual rows while HAVING is applied on groups.

## HAVING is applied after grouping while WHERE is used before grouping.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JOINs:

#combine data in multiple tables

# For each movie, print name and the genres

SELECT m.name, g.genre from movies m JOIN movies\_genres g ON m.id=g.movie\_id LIMIT 20;

# table aliases: m and g

# natural join: a join where we have the same column-names across two tables.

#T1: C1, C2

#T2: C1, C3, C4

SELECT \* FROM T1 JOIN T2;

SELECT \* FROM T1 JOIN T2 USING (C1);

# returns C1,C2,C3,C4

# no need to use the keyword "ON"

# Inner join (default) vs left outer vs right outer vs full-outer join.

T1: C1, C2, C3

SELECT m.name, g.genre from movies m LEFT JOIN movies\_genres g ON m.id=g.movie\_id LIMIT 20;

#LEFT JOIN or LEFT OUTER JOIN

#RIGHT JOIN or RIGHT OUTER JOIN

#FULL JOIN or FULL OUTER JOIN

#JOIN or INNER JOIN

# NULL for missing counterpart rows.

# 3-way joins and k-way joins

SELECT a.first\_name, a.last\_name FROM actors a JOIN roles r ON a.id=r.actor\_id JOIN movies m on m.id=r.movie\_id AND m.name='Officer 444';

#Practical note about joins: Joins can be expensive computationally when we have large tables.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sub-Queries or Nested Queries or Inner Queries

# List all actors in the movie Schindler's List

#https://www.imdb.com/title/tt0108052/fullcredits/?ref\_=tt\_ov\_st\_sm

SELECT first\_name, last\_name from actors WHERE id IN

( SELECT actor\_id from roles WHERE movie\_id IN

(SELECT id FROM movies where name='Schindler's List)

);

# Syntax:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

# first the innner query is executed and then the outer query is executed using the output values in the inner query

# IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators

#EXISTS returns true if the subquery returns one or more records or NULL

# ANY operator returns TRUE if any of the subquery values meet the condition.

# ALL operator returns TRUE if all of the subquery values meet the condition.

SELECT \* FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);

# get all movies whose rankscore is same as the maximum rankscore.

# e.g: rankscore <> ALL(...)

# https://en.wikipedia.org/wiki/Correlated\_subquery

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data Manupulation Language: SELECT, INSERT, UPDATE, DELETE

INSERT INTO movies(id, name, year, rankscore) VALUES (412321, 'Thor', 2011, 7);

INSERT INTO movies(id, name, year, rankscore) VALUES (412321, 'Thor', 2011, 7), (412322, 'Iron Man', 2008, 7.9), (412323, 'Iron Man 2', 2010, 7);

# INSERT FROM one table to annother using nnested sub query: https://en.wikipedia.org/wiki/Insert\_(SQL)#Copying\_rows\_from\_other\_tables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# UPDATE Command

UPDATE <TableName> SET col1=val1, col2=val2 WHERE condition

UPDATE movies SET rankscore=9 where id=412321;

# Update multiple rows also.

# Can be used along with Sub-queries.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#DELETE

DELETE FROM movies WHERE id=412321;

# Remove all rows: TRUNCATE TABLE TableName;

# Same as delete without a WHERE Clause.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data Definition Language

CREATE TABLE language ( id INT PRIMARY, lang VARCHAR(50) NOT NULL);

# Datatypes: https://www.journaldev.com/16774/sql-data-types

# Constraints: https://www.w3schools.com/sql/sql\_constraints.asp

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Uniquely identifies a row/record in another table

CHECK - Ensures that all values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column when no value is specified

INDEX - Used to create and retrieve data from the database very quickly

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTER: ADD, MODIFY, DROP

ALTER TABLE language ADD country VARCHAR(50);

ALTER TABLE language MODIFY country VARCHAR(60);

ALTER TABLE langauge DROP country;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Removes both the table and all of the data permanently.

DROP TABLE Tablename;

DROP TABLE TableName IF EXISTS;

#https://dev.mysql.com/doc/refman/8.0/en/drop-table.html

TRUNCATE TABLE TableName;

# as discussed earlier same as DELETE FROM TableName;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data Control Language for DB Admins.

https://en.wikipedia.org/wiki/Data\_control\_language

https://dev.mysql.com/doc/refman/8.0/en/grant.html

https://dev.mysql.com/doc/refman/8.0/en/revoke.html

Execution order of SQL commands:

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE or WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

1. select case when (A+B) <= C or (A+B) <= C or (B+C) <= A then 'Not A Triangle' WHEN A=B AND A=C THEN 'Equilateral' when (A=B AND A+B>C) or (A=C and A+C>B) or (B=C and B+C>A) then 'Isosceles' when (A!=B) AND (B!=C) AND (C!=A) then 'Scalene' end from TRIANGLES ;

2.SELECT CONCAT(name,'(',LEFT(occupation,1),')') FROM occupations ORDER BY name

SELECT CONCAT('There are a total of ',COUNT(occupation),' ',LOWER(occupation),'s.') FROM occupations GROUP BY occupation ORDER BY COUNT(occupation),occupation

3. +-------------+---------+

| Column Name | Type |

+-------------+---------+

| product\_id | int |

| low\_fats | enum |

| recyclable | enum |

+-------------+---------+

product\_id is the primary key (column with unique values) for this table.

low\_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is low fat and 'N' means it is not.

recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product is recyclable and 'N' means it is not.

4.

Prices

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product\_id | int |

| start\_date | date |

| end\_date | date |

| price | int |

+---------------+---------+

(product\_id, start\_date, end\_date) is the primary key (combination of columns with unique values) for this table.

Each row of this table indicates the price of the product\_id in the period from start\_date to end\_date.

For each product\_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product\_id.

Table: UnitsSold

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product\_id | int |

| purchase\_date | date |

| units | int |

+---------------+---------+

This table may contain duplicate rows.

Each row of this table indicates the date, units, and product\_id of each product sold.

Write a solution to find the average selling price for each product. average\_price should be **rounded to 2 decimal places**.

SELECT

p.product\_id,

IFNULL(ROUND(SUM(p.price \* u.units) / SUM(u.units), 2), 0) AS average\_price

FROM

Prices p

LEFT JOIN

UnitsSold u

ON

p.product\_id = u.product\_id

AND

u.purchase\_date BETWEEN p.start\_date AND p.end\_date

GROUP BY

p.product\_id;

5. Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.

SELECT user\_id, concat(UPPER(LEFT(name,1)), lower(substr(name,2,length(name)))) as name

from Users order by user\_id;

6. Write a solution to find the patient\_id, patient\_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix.

SELECT patient\_id, patient\_name, conditions

FROM Patients

WHERE conditions regexp '(^| )DIAB1[a-zA-Z0-9]\*( |$)';

7.find employees who are having same salaries

SELECT e1.emp\_id, e1.emp\_name, e1.salary

FROM employee e1, employee e2

WHERE e1.salary = e2.salary and e1.emp\_id != e2.emp\_id;

8. Calculates the difference between the highest salaries in the marketing and engineering departments. Output just the absolute difference in salaries.

https://platform.stratascratch.com/coding/10308-salaries-differences?code\_type=3

SELECT

ABS(MAX(CASE WHEN db\_dept.department LIKE 'engineering' THEN salary END) -

MAX(CASE WHEN db\_dept.department LIKE 'marketing' THEN salary END)) AS salary\_difference

FROM

db\_employee

JOIN

db\_dept

ON

db\_employee.department\_id = db\_dept.id;

9. Compare each employee's salary with the average salary of the corresponding department. Output the department, first name, and salary of employees along with the average salary of that department.

select department, first\_name, salary, avg(salary) over (partition by department)

from employee;

10. Find the email activity rank for each user. Email activity rank is defined by the total number of emails sent. The user with the highest number of emails sent will have a rank of 1, and so on. Output the user, total emails, and their activity rank.

• Order records first by the total emails in descending order.

• Then, sort users with the same number of emails in alphabetical order by their username.

• In your rankings, return a unique value (i.e., a unique rank) even if multiple users have the same number of emails.

select from\_user, count(\*) as total\_emails, row\_number() over(order by count(\*) desc,from\_user asc)

from google\_gmail\_emails

group by from\_user;

**GROUP BY**: Use when you want to collapse rows into aggregated results, like total sales per region.

**PARTITION BY**: Use when you need aggregated or ranked results without collapsing rows, such as calculating a running total or ranking within groups.

**COALESCE ():** It takes a list of expressions as input and evaluates them from left to right, returning the first expression that is not null.

SELECT name,

COALESCE (salary + bonus, salary, bonus, 0) AS total\_income

FROM employee;

Top Interview Questions:

1. Find Nth highest salary

Select emp.salary from (SELECT TOP N salary from employee order by salary desc) as emp order by emp.salary asc limit 1;

1. Sql query to delete duplicate rows from table
2. Joins
3. Constraints in sql
4. Aggregate questions and window functions
5. Keys in dbms
6. What is normalization and why is it used?
7. Different types of SQL commands

DDL, DML, DCL, DQL, TCL.

1. Difference between delete and truncate, drop?
2. Difference between where and having?
3. Difference between in and exists?
4. Difference between char and varchar?
5. Difference between union and union all?
6. How to create empty table with the same structure as another table?

Select \* into stu\_copy from students where 1=2;

1. What is pattern matching in sql?(%,\_ operators)
2. What is view?
3. Character manipulation functions?

Upper,lower,length,concat,initcap