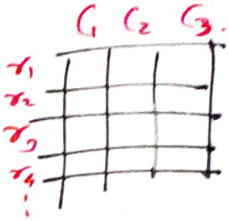
MySQL =>



* Databases = Oracle, MySql etc. Popular Database.
* Relational Databases. = SQL. DB = Everywhere eg booking flight, bus, train, movie.
* Non-Relational = MongoDB.
* Why Database ? Why not flat files like textfile, csv etc ?

⇒ ① Software => Tools = Makes life Simpler, faster, Reliable, Simple Access to Data

E.g. ① Find Row of Data where Column Value > 100 DB = Very Good.

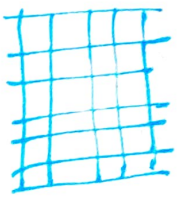
* Faster = Indexing. => Retrival Of Data. => So We Use Database.

* No Need to Worry About File handling, If the table is Large, Concepts In OS.
Just Write Single line In SQL It will Do All.

* SECURE = It Makes Backup Of Your Data At Multiple location (RAID) So In Case Of HDD Failure No Need to Worry.

* TABLES = In Relational Database => Stores Data Across Multiple Tables.

NORMALIZATION => ① Avoid Duplication.
② Split Data Across Multiple Tables.



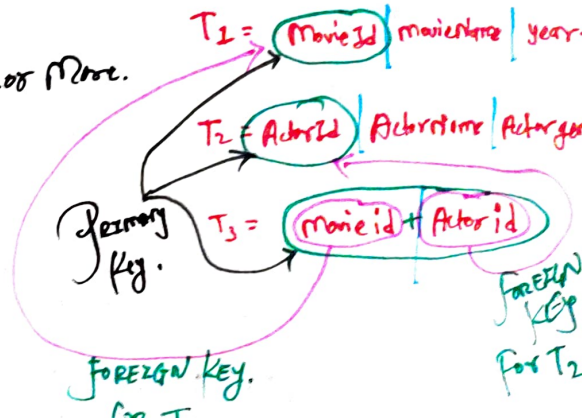
* PRIMARY Key => Unique Field = No Duplication.
③ Can Be Combination Of 2 Field or More.

Uniquely Identify Each Row of the Table.

* FOREIGN Key.

Tells Us What All Values Can be Present In the Column Of Data.

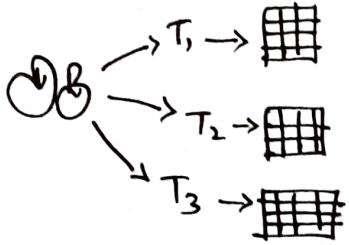
Started Its Research Project.



* Why SQL = ?

SQL = Structured Query Language.
1970 = IBM = Relational DB.
Boyce & Codd.

* Standard Way to Query | obtain | add | delete.



Query This Table
SQL = Standard Way to Query. (All Information)
Data Scientist, ML Engineer, Sfw Engineer.

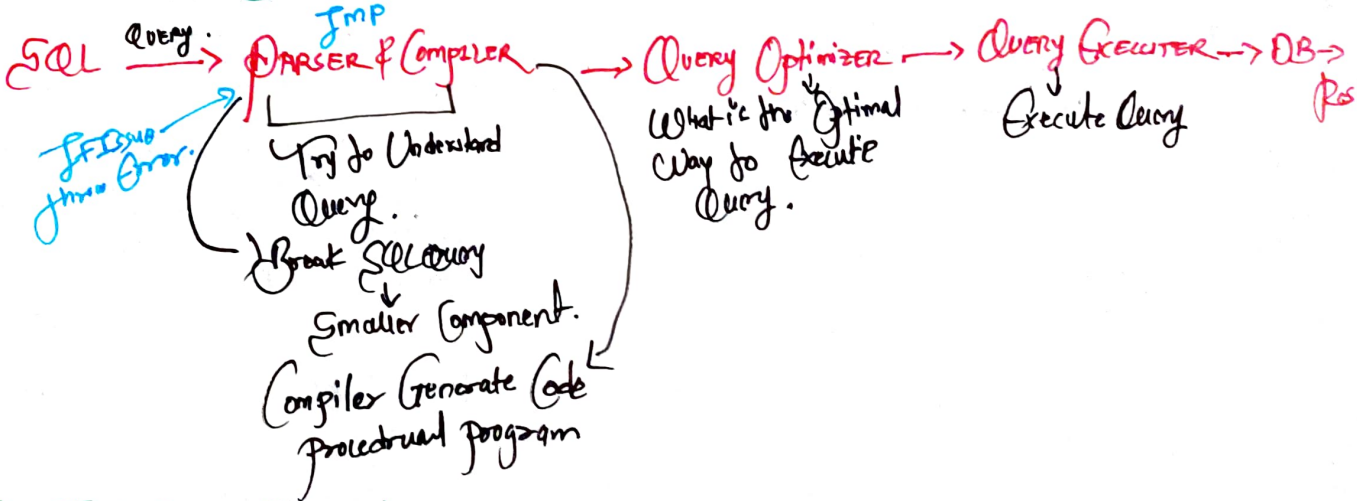
SQL = Not General Purpose Programming Language like C, C++, Python.
↳ Domain Specific Language: Only Used For Building Database (CRUD)
Task of SQL.

Procedural Programming Language - C, C++, Python, Java.
↓
Explain Step By Step
↓
Using Loops
Dict Of Movie where year > 2010.

SQL: DECLARATIVE PROGRAMMING => No Need to do Step-By-Step.
Select Movie name from Movies where year > 2010.
- Declaring What you want
- Not How to get it (No Step).
- No Loops in SQL.

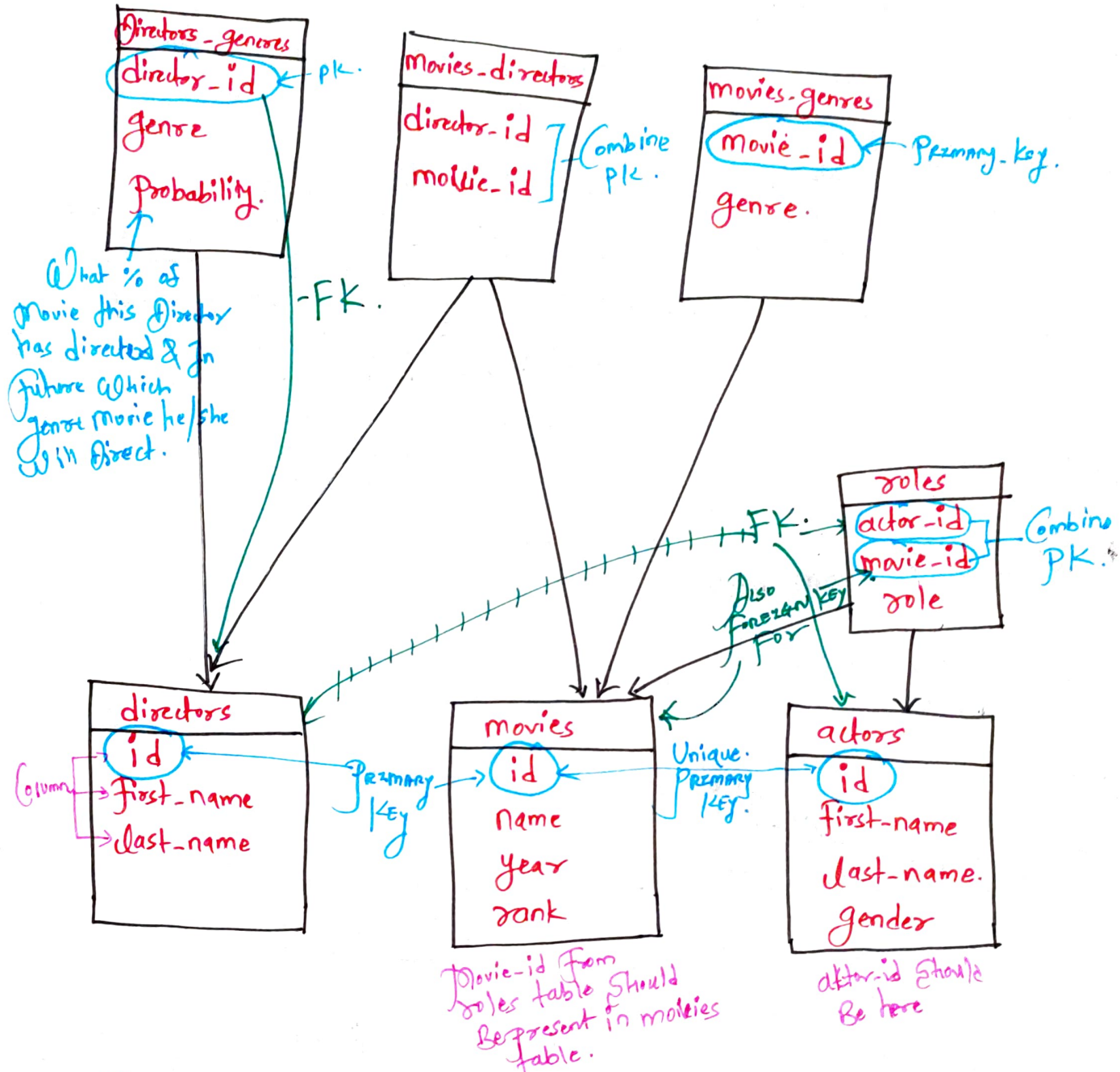
SQL - Simple, Easy to Use, Powerful.

EXECUTION OF SQL STATEMENT.



JMDB Dataset.
Movie Dataset.
Website = Amazon.com. = Data Related to Every Movies - All major language.
388269 = 1888 → 2008 = Data we have
817718 = Actors
86880 ⇒ Directors.
Large Amount of Data. } Scale. } Organized Access Table.

IMDB SCHEMA. = How the Data Is Organized = All the Relationship Between Table.



CREATE => Not Case Sensitive But good manner to Write In Capital => Increase Readability.

CREATE DATABASE database_name ;

USE => USE database_name ;

Load => SOURCE path/file.sql. ;

Bunch Of sql Command => load All tables, Rows, Columns in Database.

USE, DESCRIBE, SHOW TABLES =>

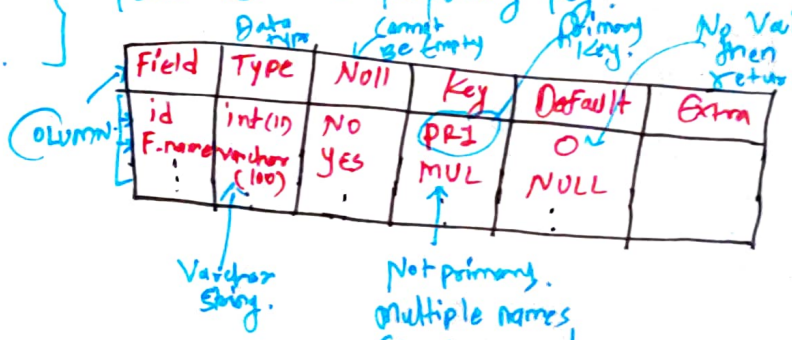
USE imdb; USE <DB-NAME>

↑ DB Name.

SHOW TABLES; ← Shows list of tables we have in our database.

DESCRIBE <Table Name>
actors
directors etc.

} Tells Which is Primary key.



List all the Data / Movies Available In Database

- SELECT * FROM movies;

Select Keyword — All Column (*=Everything.) — Keyword — table Name.

= Select All the Column from the table movies.

Slow Coz - ALL Column.

META DATA ↓ Data About Data.

List Name & Year

Order will be whatever you give.

Faster Coz only 2 Columns

SELECT name, year From movies;

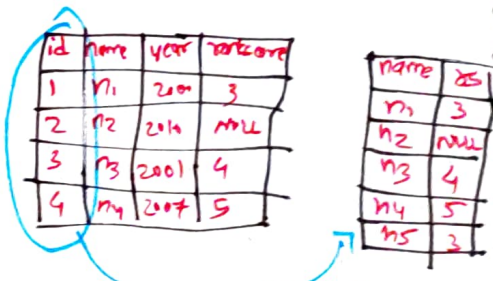
Keyword — Only two Columns — Keyword — table name

} We are not saying how to generate output => Simply we say what we want. How to get => None of my Business.

O/P -> Result Set -> Another Table With -> GENERATE NEW Table With Given Condition.

Result Set ↓ Set of Rows With column name.

It Create. 2 Column.



Same Order => preserved => Original table.

⊙ Can Be Some Cases not preserve.

• Result Set.

Row Order Preservation.

Order of Row preserved => Same as It was stored In the Database

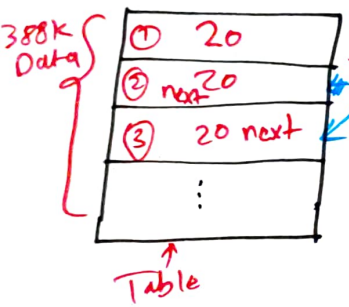
LIMIT & OFFSET

@ LIMIT => SELECT name, rankscores FROM movies LIMIT 20;

SELECT = Keyword; name, rankscores = Column to Select; FROM = Keyword; movies = Table Name; LIMIT 20 = Keyword which takes only first 20 Rows

@ OFFSET => SELECT name, rankscores FROM movies LIMIT 20 OFFSET 40;

SELECT = Keyword; name, rankscores = Columns; FROM = Keyword; movies = table name; LIMIT 20 = Skip 1st 40 then print next 20; OFFSET 40 = Ignore First 40 then Start the Count



How many Rows from the start to Ignore

ORDER BY. = Sorting By ASC or DESC.

ASC = oldest; DESC = Latest. DEFAULT.

Recent movies.

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

Columns. table. Keyword. Check Years. Descending order. TOP 10.

Eg. Amazon => Sort by -> Price low -> high, high -> Low.

THE OUTPUT ORDER MAY NOT BE SAME AS THE ONE IN THE TABLE DUE TO QUERY OPTIMIZER & INTERNAL DATA STRUCTURE / INDICES.

Row

DISTINCT

Keyword. set of Unique Values. Does not consist Duplicate. Only take one entry -> No Repetition.

Each entry only once. Comedy, fantasy, Horror

Eg. Amazon web T-shirt Size - Filter. Distinct color type.

SELECT DISTINCT genres FROM movies_genres;

SELECT DISTINCT (first_name, last_name) FROM directors;

So 1 column Value can be Repeate

Combine. multiple Columns.

Shubham S. Distinct
Shubham K.

WHERE, Comparison Operators, NULL.

List All Movies With Rankscore > 9.

e.g. Amazon Filters.

keyword.

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

Most Used

Columns.

Condition.

SELECT name, year, rankscore From movies WHERE rankscore > 9 ORDER By rankscore DESE LIMIT 20;

DESE order

Top 20 ← have highest Rank Score > 9.

Condition Outputs = TRUE, FALSE, NULL.

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

NOT EQUAL.

e.g.

① SELECT * From movies-genres WHERE genre = 'Comedy';
<> 'Horror';

② NULL ⇒ Does not Exist / Unknown / Missing.

* "=" Does not Work with NULL, Will give you An Empty Result Set.

① SELECT name, year, rankscore From movies WHERE rankscore = NULL;

Can't Compare NULL.

Empty Set.

Not work

② SELECT name, year, rankscore From movies WHERE rankscore IS NULL LIMIT 20;

KIND OF SEARCH.

work

keyword. IS NOT NULL

Used Amazon filter

LOGICAL OPERATORS = AND, OR, NOT, ALL, ANY, BETWEEN, EXIST, IN, LIKE, SOME.

① SELECT name, year, rankscore From movies WHERE rankscore > 9
AND year > 2000;

Condition 1

Condition 2.

② SELECT name, year, rankscore FROM movies WHERE NOT year <= 2000 LIMIT 20;

Latest. year > 2000

Condition

③ SELECT name, year, rankscore FROM movies WHERE rankscore > 9 OR year > 2007;

Highly rated movie of from any year

Latest / RECENT.

④ SELECT name, year, rankscore FROM movies WHERE year BETWEEN 1999 AND 2000;

Columns

table.

low value

High value.

year >= 1999 AND year <= 2000.

Also give 1999 as well as 2000

inclusion = year >= 1999 & year <= 2000.

⑤ SELECT name, year, rankscore FROM movies WHERE year BETWEEN 2000 AND 1999;

High value becomes low value will not work.

year >= 2000 AND year <= 1999

low value <= highvalue else you will get empty set as result.

So lowvalue should be less than or equal to high value.

⑥ SELECT director_id, genre FROM directors_genres WHERE genre IN ('Comedy', 'Horror');

genre in this set. →

column.

table

Keyword

Same As genre = 'Comedy' OR genre = 'Horror'

⑦ SELECT name, year, rankscore FROM movies WHERE name LIKE 'Tis%';

Regular Expression. grep in os.

Text Matching.

% => Wildcard Character to imply zero or more characters.

can be empty.

Start with Tis followed by anything.

⑧ SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es';

Fixed

First name ending in 'es'.

match.

% => Zero or more characters.

End with es preceded by anything.

Useful to match string.

⑨ SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es%';

↑ Between.

*First name Contains 'es'. Anything preceded or succeded. but should Contains 'es' in between.

⑩ SELECT first_name, last_name FROM actors WHERE first_name LIKE 'Agn_s';

'_' ⇒ Implies Exactly one Character. ← Match only 1 Character.

Agn_s ↑ Only 1 Character can be anything but only 1 Char. ↑ Start with

If We Want to Match % or _, We Should use Backslash as the Escape Character. \% & _

At least 1 or 1. → nothing between → Agnes, Agns ← no match bet^n

To match or treat %, _ as Character use backslash.

name	percentage
n1	59%
n2	70%
n3	80%

SELECT * From T WHERE Percentage = '80%'

Problem.

Solution. → '80\%'

Symbol not as Wild card Character

⑪ SELECT First_name, last_name From actors WHERE First_name LIKE 'L%' AND First_name NOT LIKE 'Li%';

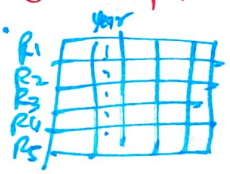
AGGREGATE FUNCTIONS COUNT, MIN, MAX, SUM, AVG.

Compute Single Value On a Set of Rows & Returns the Aggregate.

① SELECT MIN(year) From movies;

Used => Amazon, -> Price min max. Count -> total Results 1-50 page 1

Return Single Row.

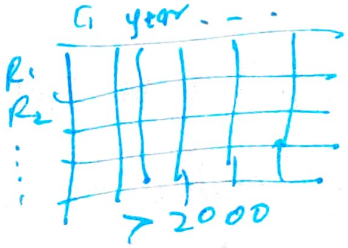


② SELECT MAX(year) FROM movies;

③ SELECT COUNT(*) From movies; ← ALL Columns. ← Count => How many Rows satisfying Condition.

④ SELECT COUNT(*) From movies WHERE year > 2000;

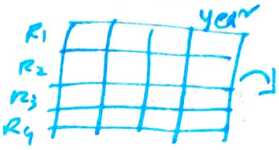
⑤ SELECT COUNT(year) From movies;



GROUP BY

- Find Number Of Movies Released Per Year.

1888 - How many Movies Released.
1889 -



All the rows same year present ↓ grouped into set

1888 {...}
1889 {...}
2007 {...}

① SELECT year, COUNT(year) FROM movies GROUP BY year;

RUN on GROUP

Count How many rows are there For each year.

② SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;

count.

output which we get.

SORT BY ASCENDING.

③ SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;

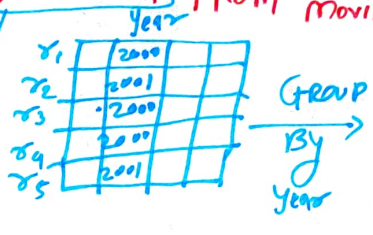
Alias. = Rename

Which year have lowest or highest

sort by number of movies.

ASC order.

Used for order by.



Year_Count is an Alias.

- Often Used With MIN, MAX, COUNT, SUM
- If grouping Column Contains NULL Values, All NULL Values are grouped together.

HAVING = Print years which have >1000 movies In Our DB. What pattern are there = Having

- SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count > 1000;

Aggregate ②

Alias.

table.

1 degree.

Applied on group.

③

① ② ③

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.

HAVING = Use when there is group by. But not compulsory.

Often Used Along with GROUP BY. NOT MANDATORY.

SELECT name, year FROM movies HAVING year > 2000;

WHERE

HAVING Without GROUP BY is Same as WHERE ← Use when no group by

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

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

(alias. · movies table. · Filter. · ① · ② · ④)

Print the year & no. of movies in the year such that year have more than 20 movies having rankscore greater than 9.

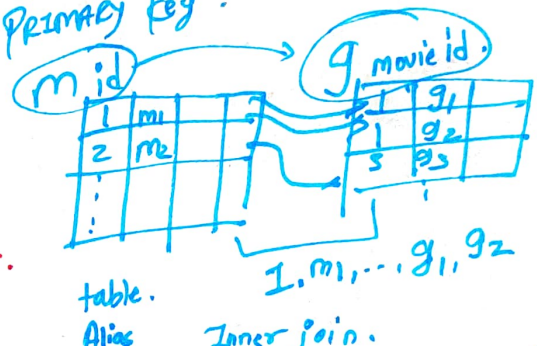
HAVING VS WHERE.

① WHERE is Applied On Individual Rows while HAVING is Applied on Group

② HAVING is Applied After GROUPING While WHERE is Used Before grouping

JOINS:

① Combine Data In Multiple tables.



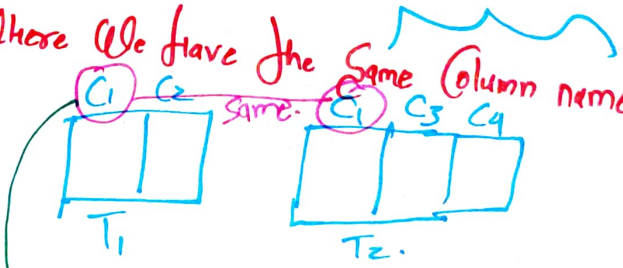
For Each Movie Print Name & Genres.

```
SELECT m.name, g.genre FROM movies m JOIN movies_genres g ON
m.id = g.movie_id LIMIT 20;
```

(Join · table Alias · Inner join. · table alias. · condition. · top 20. · Combine two tables. · T_1 · T_2)

Table Alias : m & g.

Natural Join = A Join where We have the Same Column names Across two tables



T_1 : C_1, C_2

T_2 : C_1, C_3, C_4

① `SELECT * FROM T1 JOIN T2;` ← If Column Names Are Exactly Same then Simply Join - Natural Join.

(All. · Exactly same. · $T_1.C_1$ · $T_2.C_1$)

② `SELECT * FROM T1 JOIN T2 USING (C1);`

(Ignore ON Condition when Column names are Same. · USING.)

Returns C_1, C_2, C_3, C_4

No Need to Use the keyword "ON". ← Natural Join.

Inner Join (Default) vs Left Outer vs Right Outer vs Full-Outer Joins.

T_1 : C_1, C_2, C_3

```
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.