SQL COMMANDS

```
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- In case of AVG if row contains NULL values then we will not consider that row even while dividing we will not count its number in denominator. You cannot use aggregate function inside WHERE that's why HAVING is created.

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 selete 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