Data Manipulation Language (DML)

Statements available for DML:

1. **SELECT:** It is the most common statement used and it allows us to retrieve information from a table.
   Syntax:
   SELECT Column1, Column2,etc **FROM** table
   Enter the name of the columns separated by a comma (,)
   If you want to select all the columns use (*)
   i.e. **SELECT** * **FROM** table

Ex.:

SELECT First_name FROM Actor
(Gives first name from Actor table)

SELECT Last_name FROM Actor
(Gives last name from Actor table)

SELECT * FROM Actor
(Gives all columns from Actor table)

SELECT DISTINCT (Release_year) FROM Film - here Brackets are optional
(Gives the unique value of column 'Release_year' from Film table)

2. **COUNT:** Give the count of entries for the specified range called for
   Syntax:
   SELECT COUNT (Column1**) FROM** table
   Can be combine with **DISTINCT**

   SELECT COUNT (DISTNCT Column1**) FROM** table
3. **DISTINCT:** Give Unique Values

4. **WHERE:** It allows us to specify the conditions on columns for the rows to be returned. (Used with SELECT statement)

Operator List: '=' , '>' , '<' , '<=' , '>=' , '<>' or '!=', 'AND' , 'OR' , 'NOT' (offset of specific condition)

Examples:

1. **SELECT** * **FROM** Customer
   WHERE Firstname = 'Jared'

2. **WHERE** Rental_Rate > 4 **AND** Replacement_Cost >= 19.99
3. **WHERE** Rating = 'R' **OR** Rating = 'PG-13'
4. **WHERE** Rating != 'R'

5. **ORDER BY:** To sort the data
   Syntax:

   SELECT Column1, Column2 **FROM** table
   ORDER BY Column1 ASC/DESC
   By default order is ascending not need to mention ASC.
   Need to mention DESC If you want descending order.

Examples:

1. **SELECT** Column1, Column2 **FROM** table
   ORDER BY Column1 ASC/DESC

2. **ORDER BY** Column1 ASC, Column2 DESC

3. **ORDER BY** Column1, Column2

6. **LIMIT:** limit the number of rows output

Examples:

1. **SELECT * FROM table**
   LIMIT 5

2. **SELECT * FROM Payment**
   WHERE amount != 0
   ORDER BY Payment_Date
   LIMIT 10

7. **BETWEEN – AND:** Gives data between a range (both ends inclusive)

Examples:

1. **SELECT** * **FROM** Payments
   WHERE amount **NOT BETWEEN** 7.00 **AND** 9.99

2. **WHERE** Payment_Date **BETWEEN** '2022-02-01' AND '2007-05-15'

8. **IN:** Gives data bases on certain values.
   Syntax:

   SELECT Column1 **FROM** table
   WHERE Column1 **IN** (value1, value2, value3)

Examples:

1. **SELECT** * **FROM** table
   WHERE First_Name **IN** (‘John’ , ‘Jake’ , ‘Julie’)

2. **WHERE** amount **IN** (0.99, 1.99, 11.99)

9. **LIKE**(Case Sensitive) **/ ILIKE**(Not - Case Sensitive)
   In order to match a string against a general pattern we use LIKE/ILIKE.
   Ex. All mails ending with ‘@gmail.com’: ‘%@gmail.com’

   LIKE & ILIKE allows us to perform pattern matching against string data with the use of wild card characters.

   Wild Card Characters:

   % : Matches any sequence of character [A% = Ab, Abc, Abcd]

   _ : Matches any single character [A_ = Ab, Ac, Ad]

Examples:

1. **SELECT** * **FROM** Customer
   WHERE First_Name **LIKE** ‘J%’
   [Gives all columns, rows ref. to First_Name Start with J]

2. **WHERE** First_Name **LIKE** ‘J%’ **AND** Last_Name **LIKE** ‘S%’
   [Gives all columns, rows ref. to First_Name Start with J and Last_Name starts with S]

3. **WHERE** First_Name **LIKE** ‘j%’ **AND** Last_Name **LIKE** ‘s%’
   [Since LIKE is case sensitive it may not return any thing]

4. **WHERE** First_Name **ILIKE** ‘J%’ **AND** Last_Name I**LIKE** ‘S%’
   [Gives all columns, rows ref. to First_Name Start with J or j and Last_Name starts with S or s]

5. **LIKE ‘%er%’ :** anything that contains ‘er’
6. **LIKE ‘_er_’ :** list of four character words containing ‘er’ in middle.

10. **Aggregate Functions:**
    - AVG()
    - ROUND()
    - COUNT()
    - MIN()
    - MAX()
    - SUM()

Examples:

1. **SELECT MIN(**Replacement_Cost) **FROM** Film

2. **SELECT ROUND(AVG(**Cost),2**) FROM** Film

3. **SELECT COUNT(***) **FROM** Film

4. **SELECT MIN(**Cost**) AS** Minimum_Cost **FROM** Film

11. GROUP BY: Use to Group Similar Categories, used with aggregate functions AGG().

Examples:

1. **SELECT** cust_id **FROM** payments
   GROUP BY cust_id
   [Gives unique cust_id]

2. **SELECT** cust_id, **SUM(**amount**) FROM** payments
   GROUP BY cust_id
   ORDER BY SUM(amount**)**
   [Give unique cust_id an sum the amount against that cust_id. Also the order will be according to amount in ascending order.]

3. **SELECT** cust_id, staff_id, **SUM(**amount**) FROM** payments.
   GROUP BY cust_id
   ORDER BY SUM(amount**)**

4. **SELECT DATE(**payment_date**), SUM(**amount**) FROM** payments
   GROUP BY cust_id
   ORDER BY SUM(amount**)**

12. HAVING: HAVING allows to filter after an aggregation has already taken place we can use it along with a GROUPBY.

Examples:

1. **SELECT** cust_id, **SUM(**amount**) FROM** payments
   GROUP BY cust_id
   HAVING SUM(amount**) > 200**

2. **SELECT** store_id, **COUNT(**cust_id**) AS** Total **FROM** customer
   GROUP BY store_id
   HAVING COUNT(cust_id**) > 300**

13. AS: Use to rename or Alias

SELECT COUNT(amount**) AS** transactions **FROM** payments