S33: SQL SORTING AND GROUPING

SORINTG IN SQL?

- Sorting in **SQL** is accomplished using the **ORDER BY** clause, which allows you to arrange the rows returned by a query based on one or more columns.
- Here's a detailed explanation of how to use sorting in **SQL**, including the usage of **ASC** (ascending), **DESC** (descending), and the **LIMIT** clause.

ORDER BY CLAUSE

- The **ORDER BY** clause is used to sort the result set of a query by one or more columns.
- By default, sorting is done in ascending order.

KEYWORDS: ASC & DESC

- **ASC:** Stands for ascending order. It sorts the data from the smallest to the largest value.
- **DESC:** Stands for descending order. It sorts the data from the largest to the smallest value.
- Usage of **ASC** and **DESC**:
  You use **ASC** and **DESC** within the **ORDER BY** clause to control the sorting direction of each column

LIMIT CLAUSE

- The **LIMIT** clause restricts the number of rows returned by the query.
- It is particularly useful for pagination or retrieving the top N rows.

OFFSET CLAUSE

- In **MySQL**, the **OFFSET** clause is used in conjunction with the **LIMIT** clause to specify the starting point for the rows to be returned by the query.
- It essentially tells the database to skip a specified number of rows before beginning to return rows from the query result.
- There are two ways to use **LIMIT** with **OFFSET** in **MySQL**:
- **Syntax: Separate LIMIT and OFFSET Clauses:**
  - **offset:** The number of rows to skip.
  - **row_count:** The number of rows to return.

```
SELECT col1, col2,...
FROM table_name
ORDER BY col1 [ASC | DESC], col2 [ASC | DESC], ...
LIMIT offset, row_count;
```

- **Syntax for ORDER BY:**

```
SELECT col1, col2,...
FROM table_name
ORDER BY col1 [ASC | DESC], col2 [ASC | DESC], ...
LIMIT num_of_rows;
```

- **Examples**:

Ex1: Find the top 5 samsung phones with the biggest screen size

```
SELECT model, screen_size FROM flipkart.smartphones  WHERE brand_name = 'samsung'
ORDER BY screen_size DESC LIMIT 5;
```

model	screen_size
Samsung Galaxy Z Fold 3	7.6
Samsung Galaxy Note 10 Plus 5G	6.8
Samsung Galaxy Note 10 Plus	6.8
Samsung Galaxy A83 5G	6.71
Samsung Galaxy A82 5G	6.71

Ex2: Sort all the phones in descending order of number of total cameras

```sql
SELECT brand_name, model, num_rear_cameras + num_front_cameras AS 'total_cameras' FROM flipkart.smartphones
ORDER BY screen_size DESC;
```

brand_name	model	total_cameras
vivo	Vivo V21 Pro	6
realme	Realme X50 Pro 5G (12GB RAM + 256GB)	6
oppo	OPPO F17 Pro	6
motorola	Motorola Edge S	6
vivo	Vivo V19 (8GB RAM + 256GB)	6

Ex3: Find Sort data on the basis of ppi in decreasing order

```sql
SELECT brand_name, model
ROUND(SQRT(resolution_width*resolution_width + resolution_height*resolution_height)/screen_size, 2) AS 'ppi'
FROM flipkart.smartphones ORDER BY ppi DESC;
```

brand_name	model	ppi
sony	Sony Xperia 1 II	642.63
samsung	Samsung Galaxy S20	565.98
samsung	Samsung Galaxy S20 5G	565.98
xiaomi	Xiaomi 14	545.73
google	Google Pixel 2 XL	536.66

Ex4: Find the phone with 2nd largest battery capacity

```sql
SELECT brand_name, model, battery_capacity
FROM flipkart.smartphones
ORDER BY battery_capacity DESC;
```

brand_name	model	battery_capacity
oukitel	Oukitel WP19	21000

Ex5: Find the name and rating of the worst rated apple phone

```sql
SELECT model, rating FROM flipkart.smartphones
WHERE  brand_name =  'apple'
ORDER BY rating ASC LIMIT 1;
```

model	rating
Apple iPhone 9	61

Ex6: Sort phones alphabetically and then on the basis of rating in desc order

```sql
SELECT  brand_name, model, price, rating  FROM flipkart.smartphones
ORDER BY brand_name ASC, rating DESC;
```

brand_name	model	price	rating
apple	Apple iPhone 13 Pro Max (1TB)	179900	86
apple	Apple iPhone 13 Pro Max (256GB)	139900	84
apple	Apple iPhone 13 Pro Max	129900	84
apple	Apple iPhone 13 Pro (1TB)	147900	84
apple	Apple iPhone 13 Pro	119900	83

Ex7: Sort phones alphabetically and then on the basis of rating in asc order

```sql
SELECT  brand_name, model, price, rating  FROM flipkart.smartphones
ORDER BY brand_name ASC, rating ASC;
```

brand_name	model	price	rating
apple	Apple iPhone 9	29990	61
apple	Apple iPhone SE 2020	39900	63
apple	Apple iPhone 11	38999	73
apple	Apple iPhone 11 (128GB)	46999	75
apple	Apple iPhone 11 Pro Max	109900	77

GROUPING IN SQL

- Grouping in **SQL** is a fundamental concept used to aggregate data across multiple rows and generate summary results.
- This is typically done using the **GROUP BY** clause in **SQL** queries.
- **Purpose of Grouping:**
  - ✓ Aggregates data across multiple rows.
  - ✓ Generates summary results such as totals, averages, counts, etc.

GROUP BY CLAUSE

- The **GROUP BY clause** groups rows that have the same values into summary rows, like "find the number of customers in each country".
- This clause is used to arrange identical data into groups with the help of aggregate functions such as **COUNT(), SUM(), AVG(), MAX(),** and **MIN().**
- The **GROUP BY** clause is often used in conjunction with the **SELECT** statement to generate summary data.

- **Syntax:**

```
SELECT col1, aggregate_function(col2)
FROM table_name
WHERE condition,
GROUP BY col1, col2, ...
ORDER BY col1, col2, ...
```

- **Examples:**

Ex1: Group smartphones by brand and get the count, average price, mar rating, avg screen size and avg battery capacity

```
SELECT  brand_name, COUNT(*)  AS 'num_of_phones', ROUND(AVG(price), 2)  AS 'average_price',
MAX(rating)  AS 'max_rating', ROUND(AVG(screen_size), 1) AS 'average_screen_prize',
ROUND(AVG(battery_capacity))  AS 'average_battery_capacity'
FROM flipkart.smartphones
GROUP BY brand_name
ORDER BY num_of_phones DESC LIMIT 5;
```

brand_name	num_of_phones	average_price	max_rating	average_screen_size	average_battery_capacity
xiaomi	124	20280.56	89	6.6	4981
samsung	103	29290.48	89	6.5	4917
vivo	94	21114.28	88	6.5	4767
realme	92	18030.72	89	6.5	4920
oppo	82	25768.34	89	6.6	4665

Ex2: Group smartphones by whether they have on NFC and get average price and rating

```
SELECT  has_nfc,  AVG(price)  AS 'average_price',  AVG(rating)  AS 'rating'  FROM flipkart.smartphones
GROUP BY  has_nfc;
```

has_nfc	average_price	rating
TRUE	41449.9492	83.7525
FALSE	17204.3459	75.5846

Ex3: Group smartphones by the extended memory available and get the average price

```
SELECT  extended_memory_available,  AVG(price)  AS 'price',
FROM flipkart.smartphones
GROUP BY  extended_memory_available;
```

extended_memory_available	price
0	42961.2279
1	17468.4559

Ex4: Group smartphones by the brand and the processor brand and get the count of models and the average primary camera resolution (rear)

```
SELECT  brand_name, processor_brand, COUNT(*)  AS 'num_of_phones',
AVG(primary_camera_rear) AS 'average_camera_resolution'
FROM flipkart.smartphones
GROUP BY brand_name, processor_brand;
```

brand_name	processor_brand	num_of_phones	average_camera_resolution
oneplus	snapdragon	27	52.4815
samsung	exynos	42	44.2381
motorola	snapdragon	33	60.8485
realme	dimensity	27	53.5926
oneplus	snapdragon	27	52.4815

Ex5: Find top 5 costly phones brands

```
SEL SELECT  brand_name, ROUND(AVG(price), 2)  AS 'average_price',
FROM flipkart.smartphones
GROUP BY brand_name
ORDER BY average_screen_price ASC LIMIT 1;
```

brand_name	average_price
royole	129999.00
leitz	124990.00
apple	90908.52
asus	69162.67
sharp	59990.00

Ex6: Which brand makes the smallest screen smartphones

```
SELECT  brand_name, ROUND(AVG(screen_size), 2)  AS 'average_screen_size'
FROM flipkart.smartphones
GROUP BY brand_name ORDER BY average_screen_size ASC LIMIT 1;
```

brand_name	average_screen_size
lyf	5.5

Ex7: Group smartphones by the brand and find with the highest number of models that have both NFC and IR blaster

```
SELECT  brand_name, COUNT(*)  AS 'count'  FROM flipkart.smartphones
WHERE  has_nfc =  'TRUE' AND has_ir_blaster =  'TRUE'
GROUP BY brand_name ORDER BY count DESC LIMIT 1;
```

brand_name	model
xiaomi	27

Ex8: Find all samsung 5g enabled smartphones and find out the avg price for NFC and non NFC phones

```
SELECT has_nfc, AVG(price)  AS 'average_price'  FROM flipkart.smartphones
WHERE brand_name =  'samsung'
GROUP BY has_nfc;
```

has_nfc	average_price
FALSE	16307.4386
TRUE	45378.1522

HAVING CLAUSE

- The **HAVING clause** in **MySQL** is used to filter records/rows after the **GROUP BY** clause has been applied.
- It is similar to the **WHERE** clause but is used with aggregate functions.
- Here are some key points about the **HAVING** clause:
  - **Usage with Aggregate Functions:**
    - The **HAVING** clause is used to filter records that work on summarized **GROUP BY** results.
    - It is often used with aggregate functions such as **SUM(), COUNT(), AVG(), MIN(), MAX().**
  - **Placement in Query:**
    - The **HAVING** clause comes after the **GROUP BY** clause and before the **ORDER BY** clause.

- **Filter Groups, Not Rows:**
  - While **WHERE** filters individual rows before aggregation, **HAVING** filters groups after aggregation.

- **Syntax:**

```
SELECT col1, aggregate_function(col2)
FROM table_name
WHERE condition,
GROUP BY col1, ...
HAVING condition
ORDER BY col1, col2, ...;
```

- **Examples:**

Ex1: Find the avg rating of smartphones brands which have more than 20 phones

```
SELECT brand_name, COUNT(*) AS 'count',
AVG(price) AS 'average_price'
FROM flipkart.smartphones
GROUP BY brand_name HAVING count > 40;
ORDER BY average_price DESC;
```

brand_name	count	average_price
samsung	103	29290.4757
oppo	82	25768.3415
motorola	48	21587.8750
vivo	94	21114.2766
xiaomi	124	20280.5645

Ex2: Find the top 3 brands with the highest avg ram that have a refresh rate of a at least 90 Hz and fast charging available and don't consider brands which have less than 10 phones

```
SELECT brand_name,
AVG(ram_capacity) AS 'average_ram',
FROM flipkart.smartphones
WHERE refresh_rate > 90 AND fast_charing_available = 1
GROUP BY brand_name HAVING count(*) > 10;
ORDER BY average_ram DESC LIMIT 3;
```

Brand_name	average_ram
oppo	9.7000
oneplus	8.7826
vivo	8.3810

Ex3: Find avg price of all phone brands with avg rating > 70 and num_phones more 10 among all 5g enabled phones

```
SELECT brand_name, AVG(price) AS 'average_price',
FROM flipkart.smartphones WHERE has_5g = 'TRUE'
GROUP BY brand_name HAVING AVG(rating) > 70 AND COUNT(*) > 10 ;
```

brand_name	average_price
oneplus	34373.5484
samsung	40019.5532
motorola	28263.3214
realme	23268.7255
apple	102705.6875
xiaomi	26714.3676
oppo	31540.4681
vivo	28124.9762
poco	22768.8400
iqoo	31695.9310