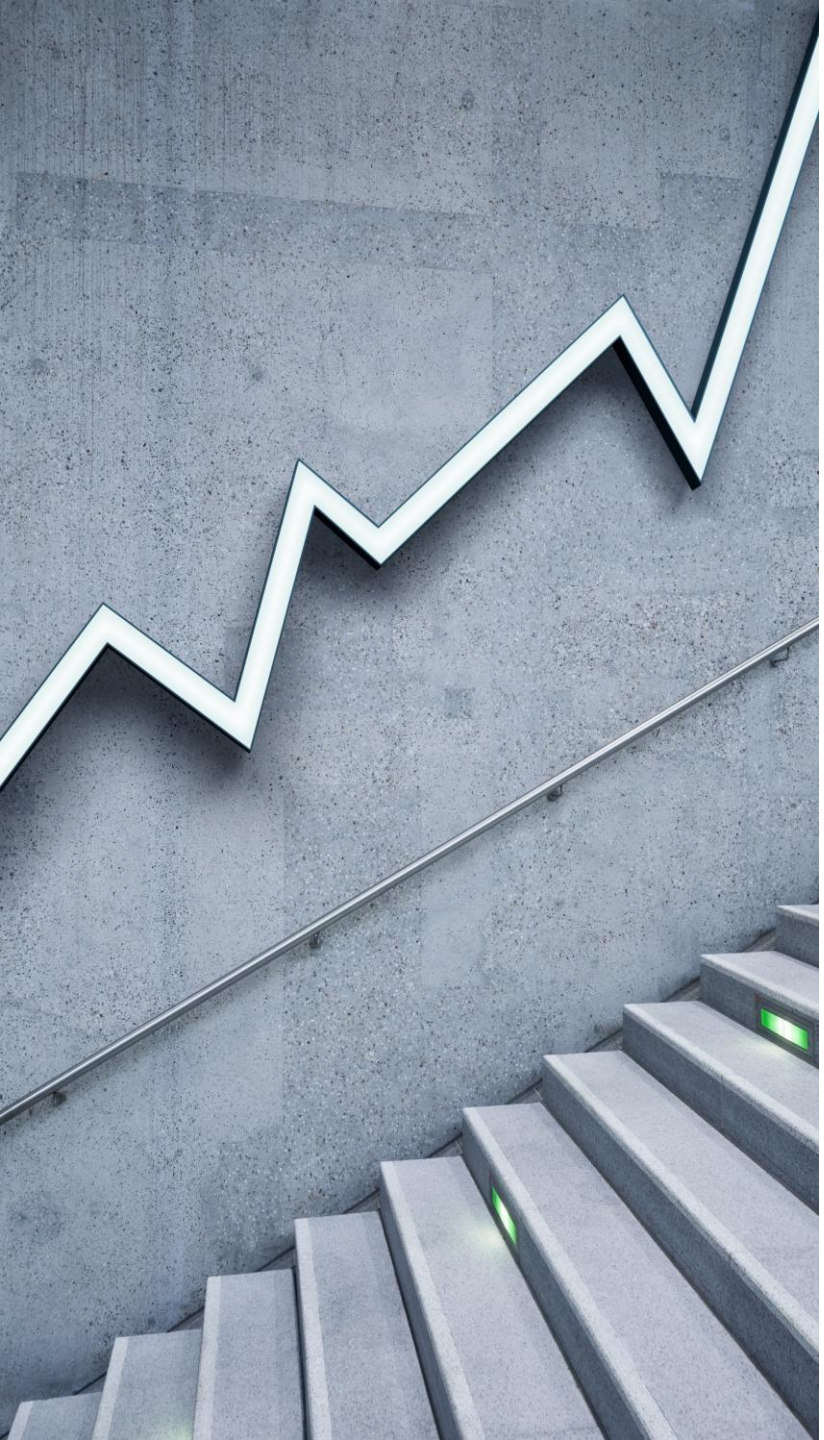

CIS 11051- PRACTICAL FOR DATABASE DESIGN

AGGREGATE FUNCTIONS

&

CLAUSES

Aggregations in MySQL

- Aggregations are **functions** that combine multiple rows into a single result based on some calculations or grouping.
- Aggregations operate on multiple rows and return a single value (e.g., the sum, average, count).
- Commonly used with GROUP BY clause to calculate results for each group of rows.

MAIN AGGREGATIONS IN MYSQL

Aggregation	What it does
SUM	Adds up the values in a column
AVG	Calculates the average value
COUNT	Counts the rows (all rows for COUNT(*), non-NULL values for COUNT(column))
MIN	Smallest value
MAX	Biggest value

1. SUM() AGGREGATION

- The SUM() function is used to **add up all the values in a specific numeric column**. It's commonly used to calculate totals, like the total sales, total revenue, etc.
- **Syntax:**

```
SELECT SUM(column_name)
FROM table_name;
```

- **Example:**

```
SELECT SUM(salary)
FROM employees;
```

2. AVG() AGGREGATION

- The AVG() function is used to **calculate the average value of a numeric column**. It's commonly used to find the mean value, such as the average salary, average age, etc.

- **Syntax:**

```
SELECT AVG(column_name)
FROM table_name;
```

- **Example:**

```
SELECT AVG(salary)
FROM employees;
```

3. COUNT() AGGREGATION

- The COUNT() function is used to **count the number of rows in a table or the number of non-NULL values in a specified column.**
- **Syntax:**

```
SELECT COUNT(column_name)
FROM table_name;
```

- **Example:**

```
SELECT COUNT(salary)
FROM employees;
```

Example Table : employees

id	name	salary
1	John	5000
2	Maria	6000
3	NULL	7000

✓ COUNT(*) – Counts all rows:

```
SELECT COUNT(*) FROM employees;
```

✓ Result : 3

✓ COUNT(name) – Counts non-NULL values in name:

```
SELECT COUNT(name) FROM employees;
```

✓ Result : 2

4. MIN() AGGREGATION

- The MIN() function is used to **find the smallest value in a column**. It can be used on any data type, like numbers, dates, or strings.
- **Syntax:**

```
SELECT MIN(column_name)
FROM table_name;
```

- **Example:**

```
SELECT MIN(salary)
FROM employees;
```

5. MAX() AGGREGATION

- The MAX() function is used to **find the largest value in a column**. It can be used on any data type, like numbers, dates, or strings.
- **Syntax:**

```
SELECT MAX(column_name)
FROM table_name;
```

- **Example:**

```
SELECT MAX(salary)
FROM employees;
```

SUMMARY

- COUNT(*) counts everything (including NULL).
- Other Aggregations (SUM , AVG, COUNT (column), MIN, MAX) ignore NULL values when performing calculations.

CLAUSES IN MYSQL

- Clauses are the components of a SQL query that define what data to retrieve or manipulate.
- They are essential in **filtering, sorting, and grouping** data.
- Clauses work together in a specific order to form a complete query.
  - Example: SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY → LIMIT

MAIN CLAUSES IN MYSQL

Clause	Purpose
SELECT	Choose what to show
FROM	Choose which table
WHERE	Filter rows before grouping
GROUP BY	Group rows together (important for aggregation!)
HAVING	Filter groups (after GROUP BY)
ORDER BY	Sort rows (ASC or DESC)
LIMIT	Limit how many rows to show

SELECT FROM WHERE CLAUSE

- The SELECT statement in MySQL is used to **query data from a database.**
- The FROM clause **specifies the table** from which to retrieve the data.
- The WHERE clause allows you to **filter the results based on specified conditions.**
- This combination is essential for fetching specific records from a table.

- Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- Example:

```
SELECT first_name, last_name
FROM employees
WHERE department = 'Sales';
```

- This query selects the first_name and last_name columns from the employees table where the department is 'Sales'.

GROUP BY CLAUSE

- GROUP BY is a clause in SQL used to **arrange identical data into groups.**
- It is often used with aggregate functions (COUNT, SUM, AVG, MAX, MIN) to perform calculations on each group of data.
- The GROUP BY clause allows you to group records that have the same values in specified columns, enabling you to get summary data from your query results.

- Syntax:

```
SELECT column1, column2, aggregate_function (column3) AS alias_name
FROM table_name
WHERE condition
GROUP BY column1, column2;
```

- Example:

```
SELECT product_name, SUM (amount_sold) AS total_sales
FROM sales
WHERE sales_date BETWEEN '2024-01-01' AND '2024-12-31’
GROUP BY product_name;
```

Explanation of the Example:

- SUM(amount_sold) AS total_sales: This computes the total sales for each product and Aliases name is total_sales for clarity.
- WHERE sales_date BETWEEN '2024-01-01' AND '2024-12-31': This filters the records to include only sales made during the year 2024.
- GROUP BY product_name: Groups the results by product_name to aggregate the sales for each product.

HAVING CLAUSE

- The HAVING clause in SQL is used to **filter records** after the GROUP BY operation has been applied.
- It is often used to filter results based on aggregated data.
- The HAVING clause is **applied to** the **results of the GROUP BY clause,** allowing you to filter the grouped data based on the aggregated values.

- Syntax:

```
SELECT column1, column2, aggregate_function(column3) AS alias_name
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING aggregate_function(column3) condition;
```

- Example:

```
SELECT product_name, SUM(amount_sold) AS total_sales
FROM sales
GROUP BY product_name
HAVING SUM(amount_sold) > 1000;
```

Explanation of the Example:

- GROUP BY product_name: Groups the data by product_name to aggregate the sales for each product.
- SUM(amount_sold) AS total_sales: Calculates the total sales for each product.
- HAVING SUM(amount_sold) > 1000: Filters the results to only include products whose total sales exceed 1000.

ODER BY CLAUSE

- The ORDER BY clause in SQL is used to **sort the result** set of a query based on one or more columns. By default, it sorts the data in **ascending order** (ASC), but you can also specify **descending order (DESC).**
- The ORDER BY clause is useful when you want to display data in a **specific order** (e.g., alphabetically, numerically, or by date).
- You can use ORDER BY to sort the results of any SQL query, whether or not the query involves grouping or aggregation. It can sort in ascending (ASC) or descending (DESC) order.

- Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC];
```

- Example

```
SELECT product_name, amount_sold, sales_date
FROM sales
ORDER BY sales_date ASC;
```

Explanation of the Syntax:

- ORDER BY column1 [ASC|DESC]: Specifies the column by which to sort the result set. You can choose ascending (ASC) or descending (DESC) order. If no order is specified, the default is ascending (ASC).

SINGLE COLUMN SORTING

If you only want to sort the results by total_sales in ascending order, you would do:

- Example:

```
SELECT product_name, SUM(amount_sold) AS total_sales
FROM sales
GROUP BY product_name
ORDER BY total_sales ASC;
```

Explanation of the Syntax:

- SUM(amount_sold) AS total_sales: Aggregates sales for each product, and the result is aliased as total_sales.
- ORDER BY total_sales ASC: Orders the result set by total_sales in ascending order (lowest to highest sales).

MULTIPLE COLUMN SORTING

Let's assume we have a sales table with product_name, amount_sold, and sales_date. You want to sort the products by the total amount sold in descending order and then by the product_name in ascending order.

- Example:

```
SELECT product_name, SUM(amount_sold) AS total_sales

FROM sales

GROUP BY product_name

ORDER BY total_sales DESC, product_name ASC;
```

Explanation of the Syntax:

- SUM(amount_sold) AS total_sales: Calculates the total sales for each product, and the result is aliased as total_sales.
- ORDER BY total_sales DESC: Orders the result set by total_sales in descending order (highest sales first).
- product_name ASC: After sorting by total sales, products are then sorted alphabetically in ascending order.

LIMIT CLAUSE

- The LIMIT clause in SQL is used to **specify the number of records to return in the result set.**
- It is often used when you want to retrieve a subset of records (e.g., the top 10 rows).
- This is especially useful when working with large datasets and you want to limit the results to a specific number or range of rows.

- Syntax:

```
SELECT column1, column2
FROM table_name
LIMIT number_of_rows;
```

- Example

```
SELECT product_name, amount_sold
FROM sales
LIMIT 5;
```

Explanation of the Syntax:

- number_of_rows: Specifies the maximum number of rows to return.

OFFSET with LIMIT

The OFFSET keyword specifies how many rows to skip before starting to return the rows.

- Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column_name
LIMIT number_of_rows OFFSET number_of_skippable_rows;
```

Example:

Let's say you have a sales table with columns product_name, amount_sold, and sales_date. You want to skip the first 5 sales records and retrieve the next 5 records, sorted by sales_date:

Example:

```
SELECT *

FROM sales

ORDER BY sales_date

LIMIT 5 OFFSET 5;
```

Explanation:

- ORDER BY sales_date: Sorts the results by the sales_date column.
- LIMIT 5: Limits the result to 5 rows.
- OFFSET 5: Skips the first 5 rows and starts returning from the 6th row onward.

Thank You !!