## RDBMS Softwares

| | |
|---|---|
| Oracle | MYSQL |
| MS SQL Server | MariaDB |

## Follow @Sourabh Sahu

## QUERYING DATA

Fetch all columns from the products table: SELECT * FROM products;

Fetch product IDs and names from the inventory table: SELECT product_id, product_name FROM inventory;

Fetch employee names sorted by their salary in descending order: SELECT employee_name FROM employees ORDER BY salary DESC;

Fetch customer names sorted by their registration date in ascending order (default): SELECT customer_name FROM customers ORDER BY registration_date;

## ALIASING (AS)

SELECT name AS product_name FROM products;

## INDEXING

Indexing improves data retrieval speed by creating a reference system. It enhances query performance but may slightly impact insert/update operations.

Clustered Index: Dictates physical data order, often used for primary keys. Non-Clustered Index: Creates separate index structures, allowing multiple indexes for specific columns, without affecting data storage order.

## BETWEEN AND IN

IN checks if a column value matches any value in a list, while BETWEEN checks if a column value falls within a specified range.

SELECT * FROM product WHERE category IN ('Electronics', 'Clothing', 'Furniture');

SELECT * FROM product WHERE price BETWEEN 20 AND 50;

## CONSTRAINTS

Constraints are rules applied to database tables to ensure data accuracy, integrity, and consistency.

PRIMARY KEY enforces unique, non-null values in the "product_id" column.

FOREIGN KEY links the "manufacturer_id" column to a related table.

NOT NULL ensures the "product_name" must have a value.

CHECK enforces a condition, like ensuring "price" is positive.

DEFAULT assigns a default value to a column when no value is provided during insertion.

CREATE TABLE product ( product_id INT PRIMARY KEY, product_name VARCHAR(255) NOT NULL, price DECIMAL(10, 2) CHECK (price > 0), manufacturer_id INT, FOREIGN KEY (manufacturer_id) REFERENCES manufacturer(manufacturer_id), discount DECIMAL(5, 2) DEFAULT 0.00 -- Default discount set to 0.00 );

## SUBQUERIES

Scalar Subquery: Returns a single value and can be used within SELECT, WHERE, or FROM clauses.

SELECT product_name, (SELECT MAX(price) FROM products) AS max_price FROM products;

Single-Row Subquery: Subquery that returns a single row of results, typically used in comparison operations.

SELECT product_name FROM products WHERE price = (SELECT MAX(price) FROM products);

Multi-Row Subquery: Subquery that returns multiple rows, often used with IN, ANY, or ALL operators.

SELECT product_name FROM products WHERE manufacturer_id IN (SELECT manufacturer_id FROM manufacturers WHERE country = 'USA');

Correlated Subquery: References values from the outer query within the subquery.

SELECT product_name FROM products p WHERE price > (SELECT AVG(price) FROM products WHERE manufacturer_id = p.manufacturer_id);

Inline View Subquery: Creates a temporary table-like structure for complex subqueries within the FROM clause.

SELECT AVG(subquery.avg_price) FROM (SELECT manufacturer_id, AVG(price) AS avg_price FROM products GROUP BY manufacturer_id) AS subquery;

## SQL and NO SQL

| | |
|---|---|
| Uses a structured, tabular data model with predefined schemas. | Offers various flexible data models, suitable for semi-structured or unstructured data. |
| Primarily designed for vertical scaling, can be complex to scale horizontally | Designed for horizontal scalability, making it easier to handle high traffic and large datasets. |

## WINDOW AND RANKING FUNCTIONS

Window functions allow you to perform calculations across a set of rows related to the current row within the result set. Rank functions assign a rank or row number to each row based on specified criteria.

SELECT product_name, price, ROW_NUMBER() OVER (ORDER BY price) AS row_num, DENSE_RANK() OVER (ORDER BY price) AS dense_rank, SUM(price) OVER (PARTITION BY manufacturer_id) AS manufacturer_total FROM products;

we use window functions like ROW_NUMBER, DENSE_RANK, and SUM with the "products" table to assign row numbers, dense ranks, and calculate the total price for each manufacturer's products.

RANK assigns the same rank to rows with equal values, leaving gaps, while DENSE_RANK assigns consecutive ranks without gaps, and ROW_NUMBER assigns a unique row number to each row.

## SET OPERATION

INTERSECT: Returns the common rows between two SELECT statements. Retrieves products that exist in both sets of results.

SELECT product_id, product_name FROM products INTERSECT SELECT product_id, product_name FROM some_other_table;

UNION: Combines the results of two SELECT statements, removing duplicates. Retrieves all unique products from both sets of results.

SELECT product_id, product_name FROM products UNION SELECT product_id, product_name FROM some_other_table;

UNION ALL: Similar to UNION, but includes all rows, including duplicates. Retrieves all products from both sets of results, allowing duplicates.

SELECT product_id, product_name FROM products UNION ALL SELECT product_id, product_name FROM some_other_table;

EXCEPT: Returns rows that exist in the first SELECT statement but not in the second. Retrieves products that are in the "products" table but not in "some_other_table."

SELECT product_id, product_name FROM products EXCEPT SELECT product_id, product_name FROM some_other_table;

## IS NULL / IS NOT NULL

IS NULL: Filters rows where the "manufacturer_id" is not assigned. SELECT * FROM product WHERE manufacturer_id IS NULL;

IS NOT NULL: Filters rows where the "manufacturer_id" is assigned. SELECT * FROM product WHERE manufacturer_id IS NOT NULL;

## FILTERING

Retrieve all products with a price greater than $50: SELECT * FROM product WHERE price > 50;

Get products with a quantity in stock less than or equal to 10: SELECT * FROM product WHERE stock_quantity <= 10;

Retrieve products with a specific category (e.g., "Electronics"): SELECT * FROM product WHERE category = 'Electronics';

Find products with names containing the word "phone": SELECT * FROM product WHERE product_name LIKE '%phone%';

Get products added after a certain date (e.g., '2023-01-01'): SELECT * FROM product WHERE date_added > '2023-01-01';

Retrieve products with a price between $20 and $30: SELECT * FROM product WHERE price BETWEEN 20 AND 30;

Find products with low stock and a price greater than $50: SELECT * FROM product WHERE stock_quantity < 5 AND price > 50;

## FILTERING (cont)

Retrieve products from a specific manufacturer (e.g., "Samsung"): SELECT * FROM product WHERE manufacturer = 'Samsung';

## JOINS

INNER JOIN: Returns only the rows with matching values in both tables.

SELECT * FROM sales INNER JOIN products ON sales.product_id = products.product_id;

LEFT JOIN (or LEFT OUTER JOIN): Returns all sales records and their corresponding products. If a product has no sales, it still appears with NULL values in sales-related columns.

SELECT * FROM products LEFT JOIN sales ON products.product_id = sales.product_id;

RIGHT JOIN (or RIGHT OUTER JOIN): Opposite of the LEFT JOIN. Returns all sales records and includes products that have no sales

SELECT * FROM sales RIGHT JOIN products ON sales.product_id = products.product_id;

FULL OUTER JOIN: Returns all sales records and all products, including those without sales. NULL values appear where there is no match.

SELECT * FROM products FULL OUTER JOIN sales ON products.product_id = sales.product_id;

SELF JOIN: A self join could be used if the "sales" table contains information about salespeople who sell products. You might join the table with itself to identify salespeople who have sold the same products.

## JOINS (cont)

SELECT s1.salesperson_name, s2.salesperson_name FROM sales AS s1 JOIN sales AS s2 ON s1.product_id = s2.product_id WHERE s1.salesperson_name <> s2.salesperson_name;

## AGGREGATION AND GROUPING

Aggregation functions like COUNT, SUM, AVG, MAX, and MIN are used to perform calculations on data. GROUP BY clause is used to group rows based on one or more columns. Aggregation functions are applied to each group, providing summary information.

-- Count the total number of products SELECT COUNT(*) AS total_products FROM products;

-- Calculate the total price of all products SELECT SUM(price) AS total_price FROM products;

-- Calculate the average price of products SELECT AVG(price) AS average_price FROM products;

-- Find the highest product price SELECT MAX(price) AS highest_price FROM products;

-- Find the lowest product price SELECT MIN(price) AS lowest_price FROM products;

## DIMENSIONAL MODELLING

A design approach for data warehousing.

Organizes data into fact tables (quantitative data) and dimension tables (descriptive data).

Simplifies complex queries, improves performance, and supports business analytics by creating a logical structure for data analysis.

## PERFORMANCE OPTIMIZATION

Indexing: Use appropriate indexes.

Optimize Queries: Write efficient SQL.

Limit Data Retrieval: Fetch only needed data.

Normalization: Properly structure tables.

Stored Procedures: Use precompiled procedures.

Table Partitioning: Divide large tables.

Regular Maintenance: Rebuild indexes, update stats.

Optimize Hardware: Ensure server resources.

Caching: Implement caching mechanisms.

Use Proper Data Types: Choose suitable types.

Concurrency Control: Manage transactions carefully.

## Data Types

| | | |
|---|---|---|
| CHAR(N): Fixed-length character string with a specified maximum length of N. | VARCHAR(N): Variable-length character string with a maximum length of N. | VARCHAR and VARCHAR2 are used interchangeably in most databases, but VARCHAR2 is specific to Oracle databases. |
| INTEGER,FLOAT,BOOLEAN | BLOB: Stores binary large objects, such as images or files. | DATETIME: Combines date and time in 'YYYY-MM-DD HH:MM:SS' format. |

## Data Types (cont)

DECIMAL(P, S): Fixed-point decimal number with P total digits and S decimal places.

DATE: Stores a date value in the format 'YYYY--MM-DD'.

TIME: Represents a time of day in 'HH:MM:SS' format.

By **sourabhsahu12345**

cheatography.com/sourabhsahu12345/

Not published yet.
Last updated 3rd September, 2023.