Querying from a table

SQL Commands	SQL Operation	Example	Description
SELECT	SELECT c1, c2 FROM t;	SELECT e.first_name, e.last_name FROM employees;	Query data in columns c1, c2 from a table
	SELECT * FROM t;	SELECT * FROM employees;	Query all rows and columns from a table
WHERE	SELECT c1, c2 FROM t WHERE CONDITION;	SELECT e.first_name, e.last_name FROM employees WHERE d.department_id = 3;	Query data and filter rows with a condition
DISTINCT	SELECT DISTINCT c1 FROM t WHERE CONDITION;	SELECT DISTINCT e.department_id FROM employees e WHERE e.salary > 50000;	Query distinct rows from a table
GROUP BY	SELECT c1, AGGREGATE(c2) FROM t GROUP BY c1;	SELECT d.department_name, COUNT(e.employee_id) FROM employees e GROUP BY d.department_name;	Group rows using an aggregate function
HAVING	SELECT c1, AGGREGATE(c2) FROM t GROUP BY c1 HAVING CONDITION;	SELECT d.department_name, AVG(e.salary) FROM employees e GROUP BY d.department_name HAVING AVG(e.salary) > 60000;	Filter groups using HAVING clause
ORDER BY	SELECT c1, c2 FROM t ORDER BY c1 ASC [DESC];	SELECT e.first_name, e.last_name FROM employees ORDER BY e.last_name ASC;	Sort the result set in ascending or descendin order
LIMIT OFFSET	SELECT c1, c2 FROM t ORDER BY c1 LIMIT n OFFSET offset;	SELECT e.first_name, e.last_name FROM employees ORDER BY e.salary DESC LIMIT 10 OFFSET 5;	Skip offset of rows and return the next n ro

Sulatha Ullal

Querying from multiple tables

SQL Commands	SQL Operation	Example	Description
INNER JOIN	SELECT c1, c2 FROM t1 INNER JOIN t2 ON condition;	SELECT e.first_name, d.department_name FROM employees e INNER JOIN departments d ON e.department_id = d.id;	Inner join t1 and t2
LEFT JOIN	SELECT c1, c2 FROM t1 LEFT JOIN t2 ON condition;	SELECT e.first_name, d.department_name FROM employees e LEFT JOIN departments d ON e.department_id = d.id;	Left join t1 and t1
RIGHT JOIN	SELECT c1, c2 FROM t1 RIGHT JOIN t2 ON condition;	SELECT e.first_name, d.department_name FROM employees e RIGHT JOIN departments d ON e.department_id = d.id;	Right join t1 and t2
FULL OUTER JOIN	SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 ON condition;	SELECT e.first_name, d.department_name FROM employees e FULL OUTER JOIN departments d ON e.department_id = d.id;	Perform full outer join
CROSS JOIN	SELECT c1, c2 FROM t1 CROSS JOIN t2;	SELECT e.first_name, d.department_name FROM employees e CROSS JOIN departments d;	Produce a Cartesian product of rows in tables
	SELECT c1, c2 FROM t1, t2;	SELECT e.first_name, d.department_name FROM employees e, departments d;	Another way to perform cross join
SELF JOIN	SELECT c1, c2 FROM t1 A INNER JOIN t2 B ON condition;	SELECT A.first_name AS Employee, B.first_name AS Manager FROM employees A INNER JOIN employees B ON A.manager_id = B.employee_id;	Join t1 to itself using INNER JOIN clause

Sulatha Ullal

SQL Operators

SQL Commands	SQL Operation	Example	Description
UNION ALL	SELECT c1, c2 FROM t1 UNION [ALL] SELECT c1, c2 FROM t2;	SELECT e.first_name, e.last_name FROM employees UNION SELECT d.manager_first_name, d.manager_last_name FROM departments d;	Combine rows from two queries
INTERSECT	SELECT c1, c2 FROM t1 INTERSECT SELECT c1, c2 FROM t2;	SELECT e.first_name, e.last_name FROM employees INTERSECT SELECT e2.first_name, e2.last_name FROM employees e2 WHERE e2.salary > 70000;	Return the intersection of two queries
MINUS	SELECT c1, c2 FROM t1 MINUS SELECT c1, c2 FROM t2;	SELECT e.first_name, e.last_name FROM employees MINUS SELECT m.first_name, m.last_name FROM managers m;	Subtract a result set from another result set
LIKE	SELECT c1, c2 FROM t1 WHERE c1 [NOT] LIKE pattern;	SELECT e.first_name, e.last_name FROM employees WHERE e.first_name LIKE 'J%';	Query rows using pattern matching %, _
IN	SELECT c1, c2 FROM t1 WHERE c1 [NOT] IN value_list;	SELECT e.first_name, e.last_name FROM employees WHERE e.department_id NOT IN (1, 2, 3);	Query rows in a list
BETWEEN	SELECT c1, c2 FROM t1 WHERE c1 BETWEEN low AND high;	SELECT e.first_name, e.salary FROM employees WHERE e.salary BETWEEN 50000 AND 80000;	Query rows between two values
NOT NULL	SELECT c1, c2 FROM t WHERE c1 IS [NOT] NULL;	SELECT e.first_name, d.department_name FROM employees e JOIN departments d ON e.department_id = d.id WHERE e.first_name IS NOT NULL;	Check if values in a table is NULL or not

Sulatha Ullal

Managing Tables

SQL Commands	SQL Operation	Example	Description
CREATE TABLE	CREATE TABLE t (id INT PRIMARY KEY, name VARCHAR NOT NULL, price INT DEFAULT 0);	CREATE TABLE employees (id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL, last_name VARCHAR(50) NOT NULL, salary INT DEFAULT 0);	Create a new table with three columns
DROP TABLE	DROP TABLE t ;	DROP TABLE employees;	Delete the table from the database
ALTER TABLE	ALTER TABLE t ADD column;	ALTER TABLE employees ADD department_id INT;	Add a new column to the table
DROP COL	ALTER TABLE t DROP COLUMN c ;	ALTER TABLE employees DROP COLUMN department_id;	Drop column c from the table
ADD CONSTRAINT	ALTER TABLE t ADD constraint;	ALTER TABLE employees ADD CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES departments(id);	Add a constraint
DROP CONSTRAINT	ALTER TABLE t DROP constraint;	ALTER TABLE employees DROP CONSTRAINT fk_department;	Drop a constraint
RENAME	ALTER TABLE t1 RENAME to t2;	ALTER TABLE employees RENAME TO staff;	Rename a table from t1 to t2
	ALTER TABLE t1 RENAME c1 TO c2 ;	ALTER TABLE staff RENAME first_name TO first;	Rename column c1 to c2
TRUNCATE	TRUNCATE TABLE t;	TRUNCATE TABLE staff;	Remove all data in a table

Sulatha Ullal

SQL Constraints

SQL Commands	SQL Operation	Example	Description
PRIMARY KEY	CREATE TABLE t(c1 INT, c2 INT, c3 VARCHAR, PRIMARY KEY (c1,c2));	CREATE TABLE emp_dept (emp_id INT, dept_id INT, emp_name VARCHAR(50), PRIMARY KEY (emp_id, dept_id));	Set c1 and c2 as a primary key
FOREIGN KEY	CREATE TABLE t1(c1 INT PRIMARY KEY, c2 INT, FOREIGN KEY (c2) REFERENCES t2(c2));	CREATE TABLE dept (dept_id INT PRIMARY KEY, dept_name VARCHAR(100)); CREATE TABLE emp (emp_id INT PRIMARY KEY, dept_id INT, FOREIGN KEY (dept_id) REFERENCES dept(dept_id));	Set c2 column as a foreign key
UNIQUE	CREATE TABLE t(c1 INT, c1 INT, UNIQUE(c2,c3));	CREATE TABLE emp_details (emp_id INT, emp_email VARCHAR(100), emp_phone VARCHAR(15), UNIQUE (emp_email, emp_phone));	Make the values in c1 and c2 unique
CHECK	CREATE TABLE t(c1 INT, c2 INT, CHECK(c1> 0 AND c1 >= c2));	CREATE TABLE emp_salary (emp_id INT, salary INT, CHECK (salary > 0));	Ensure c1 > 0 and values in c1 >= c2
NOT NULL	CREATE TABLE t(c1 INT PRIMARY KEY, c2 VARCHAR NOT NULL);	CREATE TABLE dept (dept_id INT PRIMARY KEY, dept_name VARCHAR(100) NOT NULL);	Set values in c2 column not NULL

Sulatha Ullal

Modifying Data

SQL Commands	SQL Operation	Example	Description
INSERT	INSERT INTO t(column_list) VALUES(value_list);	INSERT INTO emp (emp_id, emp_name, dept_id) VALUES (1, 'John Doe', 101);	Insert one row into a table
	INSERT INTO t(column_list) VALUES (value_list), (value_list), ….;	INSERT INTO emp (emp_id, emp_name, dept_id) VALUES (2, 'Jane Smith', 102), (3, 'Alice Johnson', 101), (4, 'Bob Brown', 103);	Insert multiple rows into a table
	INSERT INTO t1(column_list) SELECT column_list FROM t2;	INSERT INTO dept (dept_id, dept_name) SELECT dept_id, dept_name FROM temp_dept;	Insert rows from t2 into t1
UPDATE	UPDATE t SET c1 = new_value; WHERE condition;	UPDATE emp SET dept_id = 102 WHERE emp_id = 1;	Update new value in the column c1 for all rows
SET	UPDATE t SET c1 = new_value, c2 = new_value WHERE condition;	UPDATE emp SET emp_name = 'Johnathan Doe', dept_id = 103 WHERE emp_id = 1;	Update values in the column c1, c2 that match the condition
DELETE	DELETE FROM t;	DELETE FROM emp;	Delete all data in a table
	DELETE FROM t WHERE condition;	DELETE FROM emp WHERE emp_id = 2;	Delete subset of rows in a table

Sulatha Ullal

Managing views

SQL Commands	SQL Operation	Example	Description
CREATE VIEW	CREATE VIEW v(c1,c2) AS SELECT c1, c2 FROM t;	CREATE VIEW emp_view(emp_id, emp_name) AS SELECT employee_id, name FROM emp;	Create a new view that consists of c1 and c2
CHECK OPTION	CREATE VIEW v(c1,c2) AS SELECT c1, c2 FROM t; WITH [CASCADED | LOCAL] CHECK OPTION;	CREATE VIEW high_salary_emp_view(emp_id, emp_name, emp_salary) AS SELECT employee_id, name, salary FROM emp WHERE salary > 60000 WITH CHECK OPTION;	Create a new view with check option
RECURSIVE VIEW	CREATE RECURSIVE VIEW v AS select-statement -- anchor part UNION [ALL] select-statement; -- recursive part	CREATE RECURSIVE VIEW emp_hierarchy AS WITH RECURSIVE hierarchy AS (SELECT employee_id, manager_id, name FROM emp WHERE manager_id IS NULL -- Anchor part: Top-level employees UNION ALL SELECT e.employee_id, e.manager_id, e.name FROM emp e JOIN hierarchy h ON e.manager_id = h.employee_id -- Recursive part) SELECT * FROM hierarchy;	Create a recursive view
TEMPORARY VIEW	CREATE TEMPORARY VIEW v AS SELECT c1, c2 FROM t;	CREATE TEMPORARY VIEW temp_emp_view AS SELECT employee_id, name, department_id FROM emp;	Create a temporary view
DROP VIEW	DROP VIEW view_name	DROP VIEW emp_view;	Delete a view

Sulatha Ullal

Managing indexes

SQL Commands	SQL Operation	Example	Description
CREATE INDEX	CREATE INDEX idx_name ON t(c1,c2);	CREATE INDEX idx_salary_department ON emp(salary, department_id);	Create an index on c1 and c2 of the table t
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX idx_name ON t(c3,c4);	CREATE UNIQUE INDEX idx_unique_email_department ON emp(email, department_id);	Create a unique index on c3, c4 of the table t
DROP INDEX	DROP INDEX idx_name;	idx_unique_email_department ON	Drop an index

Sulatha Ullal

SQL Functions

SQL Commands	Example	Description
AVG	SELECT d.department_name, AVG(e.salary) AS average_salary FROM emp e JOIN dept d ON e.department_id = d.department_id GROUP BY d.department_name;	AVG returns the average of a list
COUNT	SELECT d.department_name, COUNT(e.employee_id) AS employee_count FROM emp e JOIN dept d ON e.department_id = d.department_id GROUP BY d.department_name;	COUNT returns the number of elements of a list
SUM	SELECT d.department_name, SUM(e.salary) AS total_salary FROM emp e JOIN dept d ON e.department_id = d.department_id GROUP BY d.department_name;	SUM returns the total of a list
MAX	SELECT d.department_name, MAX(e.salary) AS highest_salary FROM emp e JOIN dept d ON e.department_id = d.department_id GROUP BY d.department_name;	MAX returns the maximum value in a list
MIN	SELECT d.department_name, MIN(e.salary) AS lowest_salary FROM emp e JOIN dept d ON e.department_id = d.department_id GROUP BY d.department_name;	MIN returns the minimum value in a list
UPPER/LOWER	SELECT UPPER(first_name), LOWER(first_name) FROM emp;	Converts a string to uppercase/lowercase
LENGTH	SELECT LENGTH(first_name) FROM emp;	Returns the length of a string.
TRIM	SELECT TRIM(first_name) FROM emp;	Removes leading and trailing spaces from a string.
ROUND	SELECT ROUND(salary, 2) FROM emp;	Rounds a numeric value to a specified number of decimal places.
CURRENT_DATE	SELECT CURRENT_DATE;	Returns the current date.
DATEADD()	SELECT DATEADD('day', 7, CURRENT_DATE);	Adds a specified interval to a date.
DATEDIFF()	SELECT DATEDIFF(CURRENT_DATE, hire_date) FROM emp;	Returns the difference between two dates.
EXTRACT()	SELECT EXTRACT(YEAR FROM hire_date) FROM emp;	Extracts a specific part of a date (like year or month).

Sulatha Ullal

Managing Triggers

SQL Commands	SQL Operation	Example	Description

CREATE OR MODIFY TRIGGER	CREATE OR MODIFY TRIGGER trigger_name
	WHEN EVENT
	ON table_name TRIGGER_TYPE
	EXECUTE stored_procedure;
		CREATE TABLE dept (
		department_id INT PRIMARY KEY,
		department_name VARCHAR(100),
		total_salary DECIMAL(10, 2) DEFAULT 0
		);
		
		CREATE OR MODIFY TRIGGER update_department_salary
		AFTER UPDATE ON emp
		FOR EACH ROW
		BEGIN
		IF NEW.salary <> OLD.salary THEN
		UPDATE dept
		SET total_salary = total_salary + (NEW.salary - OLD.salary)
		WHERE department_id = NEW.department_id;
		END IF;
		END;
			Create or modify a trigger
			
			WHEN
			• BEFORE – invoke before the event occurs
			• AFTER – invoke after the event occurs
			
			EVENT
			• INSERT – invoke for INSERT
			• UPDATE – invoke for UPDATE
			• DELETE – invoke for DELETE
			
			TRIGGER_TYPE
			• FOR EACH ROW
			• FOR EACH STATEMENT

DROP	DROP TRIGGER trigger_name	DROP TRIGGER update_department_salary	Delete a specific trigger

Sulatha Ullal

SQL Tips & Tricks

SQL Commands	Example	Description
Where 1=1	SELECT e.name, e.salary, d.department_name FROM emp e JOIN dept d ON e.department_id = d.department_id WHERE 1=1 AND e.salary > 50000; -- Additional condition	Often used in dynamic SQL to simplify the addition of further conditions as where 1=1 is always true.
Row_number, Qualify	SELECT e.name, e.salary, d.department_name, ROW_NUMBER() OVER (PARTITION BY d.department_name ORDER BY e.salary DESC) AS rn FROM emp e JOIN dept d ON e.department_id = d.department_id QUALIFY rn = 1; -- Get the highest-paid employee in each department	ROW_NUMBER assigns a unique sequential integer to rows within a partition of a result set. QUALIFY is often used to filter the result of window functions.
Coalesce	SELECT e.name, COALESCE(e.salary, 0) AS salary FROM emp e;	Returns the first non-null value in a list.
Exists	SELECT d.department_name FROM dept d WHERE EXISTS (SELECT 1 FROM emp e WHERE e.department_id = d.department_id AND e.salary > 60000);	It is used to check if a subquery returns any results.
Temp tables	CREATE TEMPORARY TABLE temp_salaries AS SELECT employee_id, salary FROM emp WHERE salary > 50000; SELECT * FROM temp_salaries;	Creating and using a temporary table for analysis.

Sulatha Ullal

SQL Tips & Tricks

SQL Commands	Example	Description
Lag/Lead	SELECT e.name, e.salary, LAG(e.salary) OVER (ORDER BY e.salary) AS previous_salary, LEAD(e.salary) OVER (ORDER BY e.salary) AS next_salary FROM emp e;	LAG and LEAD functions allow you to access data from previous or subsequent rows in the result set.
Window functions	SELECT e.name, e.salary, d.department_name, AVG(e.salary) OVER (PARTITION BY d.department_name) AS avg_department_salary FROM emp e JOIN dept d ON e.department_id = d.department_id;	Window functions perform calculations across a set of table rows that are related to the current row.
CTE	WITH HighSalaryEmployees AS (SELECT e.name, e.salary, d.department_name FROM emp e JOIN dept d ON e.department_id = d.department_id WHERE e.salary > 70000) SELECT * FROM HighSalaryEmployees;	CTE (Common Table Expressions) allows you to create a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
CTAS	CREATE TABLE high_salary_emp AS SELECT e.employee_id, e.name, e.salary, d.department_name FROM emp e JOIN dept d ON e.department_id = d.department_id WHERE e.salary > 70000;	CTAS is a versatile command that can be used for data manipulation, transformation, and performance optimization. It's especially helpful when dealing with large datasets or complex queries where creating a new table can simplify future operations.

Sulatha Ullal