ALL SQL QUARRIES

Command	Syntax	Use	Example
SELECT	SELECT column1, column2 FROM table_name WHERE condition;	Retrieve data from a database.	SELECT name, age FROM students WHERE age > 18;
INSERT	INSERT INTO table_name (column1, column2) VALUES (value1, value2);	Insert new data into a table.	INSERT INTO students (name, age) VALUES ('John', 20);
UPDATE	UPDATE table_name SET column1 = value1 WHERE condition;	Modify existing data in a table.	UPDATE students SET age = 21 WHERE name = 'John';
DELETE	DELETE FROM table_name WHERE condition;	Delete data from a table.	DELETE FROM students WHERE name = 'John';
CREATE TABLE	CREATE TABLE table_name (column1 datatype, column2 datatype);	Create a new table in the database.	CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(100), age INT);
ALTER TABLE	ALTER TABLE table_name ADD/DROP/MODIFY column_name datatype;	Modify the structure of an existing table.	ALTER TABLE students ADD email VARCHAR(100);
DROP TABLE	DROP TABLE table_name;	Delete a table from the database.	DROP TABLE students;
TRUNCATE	TRUNCATE TABLE table_name;	Remove all data from a table without deleting the table itself.	TRUNCATE TABLE students;
CREATE DATABASE	CREATE DATABASE database_name;	Create a new database.	CREATE DATABASE school;
DROP DATABASE	DROP DATABASE database_name;	Delete a database.	DROP DATABASE school;

FULL JOIN	SELECT columns FROM table1 FULL OUTER JOIN table2 ON table1.column = table2.column;	Returns all records when there is a match in either table.	SELECT students.name, courses.course_name FROM students FULL OUTER JOIN courses ON students.id = courses.student_id;
GROUP BY	SELECT column, aggregate_function(column) FROM table_name GROUP BY column;	Groups rows that have the same values in specified columns.	SELECT age, COUNT(*) FROM students GROUP BY age;
ORDER BY	`SELECT columns FROM table_name ORDER BY column1 [ASC	DESC];`	Sort the result set in ascending or descending order.
HAVING	SELECT columns FROM table_name GROUP BY column HAVING condition;	Filter groups based on a condition.	SELECT age, COUNT(*) FROM students GROUP BY age HAVING COUNT(*) > 1;
DISTINCT	SELECT DISTINCT column1, column2 FROM table_name;	Retrieve distinct (unique) values from a column.	SELECT DISTINCT age FROM students;
UNION	SELECT columns FROM table1 UNION SELECT columns FROM table2;	Combine the result sets of two SELECT statements (without duplicates).	SELECT name FROM students UNION SELECT name FROM alumni;
UNION ALL	SELECT columns FROM table1 UNION ALL SELECT columns FROM table2;	Combine the result sets of two SELECT statements (with duplicates allowed).	SELECT name FROM students UNION ALL SELECT name FROM alumni;
EXISTS	SELECT column1 FROM table_name WHERE EXISTS (subquery);	Check for the existence of any record in a subquery.	SELECT name FROM students WHERE EXISTS (SELECT * FROM courses WHERE students.id = courses.student_id);
IN	SELECT column1 FROM table_name WHERE column2 IN (value1, value2, ...);	Check if a value is present in a set of values.	SELECT name FROM students WHERE age IN (18, 20, 22);

BETWEEN	SELECT column1 FROM table_name WHERE column2 BETWEEN value1 AND value2;	Filter records within a certain range.	SELECT name FROM students WHERE age BETWEEN 18 AND 22;
LIKE	SELECT column1 FROM table_name WHERE column2 LIKE pattern;	Search for a specific pattern in a column.	SELECT name FROM students WHERE name LIKE 'J%';
COUNT	SELECT COUNT(column) FROM table_name;	Return the number of rows that match a condition.	SELECT COUNT(*) FROM students WHERE age > 18;
AVG	SELECT AVG(column) FROM table_name;	Calculate the average value of a numeric column.	SELECT AVG(age) FROM students;
SUM	SELECT SUM(column) FROM table_name;	Calculate the total sum of a numeric column.	SELECT SUM(age) FROM students;
MIN	SELECT MIN(column) FROM table_name;	Find the smallest value in a column.	SELECT MIN(age) FROM students;
MAX	SELECT MAX(column) FROM table_name;	Find the largest value in a column.	SELECT MAX(age) FROM students;

CREATE INDEX	CREATE INDEX index_name ON table_name (column1, column2);	Create an index on a table for faster retrieval.	CREATE INDEX idx_name ON students (name);
DROP INDEX	DROP INDEX index_name ON table_name;	Remove an index from a table.	DROP INDEX idx_name ON students;
CREATE VIEW	CREATE VIEW view_name AS SELECT columns FROM table_name WHERE condition;	Create a virtual table based on the result of a SELECT statement.	CREATE VIEW student_view AS SELECT name, age FROM students WHERE age > 18;
DROP VIEW	DROP VIEW view_name;	Delete a view.	DROP VIEW student_view;
CREATE TRIGGER	CREATE TRIGGER trigger_name BEFORE/AFTER INSERT/UPDATE/DELETE ON table_name FOR EACH ROW BEGIN ... END;	Execute a set of SQL statements automatically in response to certain events.	CREATE TRIGGER after_student_insert AFTER INSERT ON students FOR EACH ROW BEGIN INSERT INTO audit_log VALUES (NEW.id, NEW.name); END;
CREATE PROCEDURE	CREATE PROCEDURE procedure_name (parameters) BEGIN ... END;	Define a stored procedure with a set of SQL statements.	CREATE PROCEDURE GetStudent(IN studentID INT) BEGIN SELECT * FROM students WHERE id = studentID; END;
CALL	CALL procedure_name(parameters);	Execute a stored procedure.	CALL GetStudent(5);

CREATE FUNCTION	CREATE FUNCTION function_name (parameters) RETURNS datatype BEGIN ... END;	Define a function that returns a value.	CREATE FUNCTION CalculateAge (birthdate DATE) RETURNS INT BEGIN RETURN YEAR(CURDATE()) - YEAR(birthdate); END;
SET	SET variable_name = value;	Assign a value to a variable.	SET @total_students = (SELECT COUNT(*) FROM students);
SAVEPOINT	SAVEPOINT savepoint_name;	Set a savepoint within a transaction.	SAVEPOINT savepoint1;
ROLLBACK	ROLLBACK TO savepoint_name;	Roll back a transaction to a specific savepoint.	ROLLBACK TO savepoint1;
COMMIT	COMMIT;	Save all changes made in a transaction.	COMMIT;
REVOKE	REVOKE privilege ON object FROM user;	Remove privileges from a user.	REVOKE SELECT ON students FROM 'user';
GRANT	GRANT privilege ON object TO user;	Give privileges to a user.	GRANT SELECT, INSERT ON students TO 'user';

LOCK TABLE	LOCK TABLE table_name IN READ/WRITE mode;	Lock a table to prevent access by other users.	LOCK TABLE students IN WRITE;
UNLOCK TABLE	UNLOCK TABLES;	Unlock previously locked tables.	UNLOCK TABLES;
EXPLAIN	EXPLAIN SELECT column1 FROM table_name WHERE condition;	Get information about how SQL statements will be executed.	EXPLAIN SELECT name, age FROM students WHERE age > 18;
MERGE	MERGE INTO table_name USING source_table ON (condition) WHEN MATCHED THEN ... WHEN NOT MATCHED THEN ...;	Perform INSERT or UPDATE operations based on conditions.	MERGE INTO students USING new_students ON (students.id = new_students.id) WHEN MATCHED THEN UPDATE SET students.name = new_students.name;
COMMENT	COMMENT ON TABLE table_name IS 'comment';	Add comments to database objects such as tables or columns.	COMMENT ON TABLE students IS 'Stores student information';
CHECK	CREATE TABLE table_name (column1 datatype CHECK (condition));	Enforce conditions on columns (constraints).	CREATE TABLE students (age INT CHECK (age >= 18));

CONSTRAINT	ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column);	Add or modify a constraint to enforce rules on data.	ALTER TABLE students ADD CONSTRAINT unique_email UNIQUE (email);
RENAME	ALTER TABLE old_table_name RENAME TO new_table_name;	Rename a table or column.	ALTER TABLE students RENAME TO student_info;
WITH	WITH subquery_name AS (subquery) SELECT columns FROM subquery_name WHERE condition;	Define common table expressions (CTE) for reuse within queries.	WITH top_students AS (SELECT name, grade FROM students WHERE grade > 90) SELECT * FROM top_students;
FETCH	FETCH FIRST N ROWS ONLY;	Limit the number of rows returned by a query (similar to LIMIT).	SELECT * FROM students FETCH FIRST 10 ROWS ONLY;
CASE	SELECT column1, CASE WHEN condition THEN result ELSE alternative END FROM table_name;	Apply conditional logic within queries.	SELECT name, CASE WHEN age < 18 THEN 'Minor' ELSE 'Adult' END AS age_group FROM students;
WINDOW FUNCTIONS	SELECT column1, aggregate_function() OVER (PARTITION BY column2 ORDER BY column3) FROM table_name;	Perform aggregate calculations across sets of rows (e.g., rankings, running totals).	SELECT name, SUM(salary) OVER (PARTITION BY department ORDER BY hire_date) AS running_total FROM employees;

SEQUENCE	CREATE SEQUENCE sequence_name START WITH value INCREMENT BY value;	Create a sequence generator for auto-incrementing values.	CREATE SEQUENCE student_seq START WITH 1 INCREMENT BY 1;
NEXTVAL	SELECT sequence_name.NEXTVAL;	Retrieve the next value from a sequence.	SELECT student_seq.NEXTVAL;
SET TRANSACTION	SET TRANSACTION ISOLATION LEVEL level_name;	Set the isolation level for the current transaction.	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ROLLBACK	ROLLBACK;	Roll back all changes made in the current transaction.	ROLLBACK;
DEALLOCATE	DEALLOCATE PREPARE stmt_name;	Deallocate a previously prepared statement.	DEALLOCATE PREPARE stmt_name;
PREPARE	PREPARE stmt_name AS statement;	Prepare a SQL statement for execution with a dynamic query.	PREPARE stmt_name AS SELECT * FROM students WHERE id = ?;
EXECUTE	EXECUTE stmt_name USING variable;	Execute a prepared statement with parameters.	EXECUTE stmt_name USING @student_id;
LISTEN/NOTIFY	LISTEN channel_name;	Listen for a notification from another session (PostgreSQL only).	LISTEN student_updates;
NOTIFY	NOTIFY channel_name, 'message';	Send a notification to all listeners on a channel (PostgreSQL).	NOTIFY student_updates, 'New student added';

ARRAY	SELECT ARRAY[1, 2, 3];	Create and work with arrays in queries.	SELECT ARRAY['John', 'Jane', 'Doe'];
JSON	SELECT column->>'json_key' FROM table_name;	Query JSON data stored in columns.	SELECT info->>'age' FROM students;
XML	SELECT XMLELEMENT(NAME "element", column) FROM table_name;	Query and manipulate XML data.	SELECT XMLELEMENT(NAME "StudentName", name) FROM students;
RECURSIVE	WITH RECURSIVE subquery_name AS (initial_query UNION ALL recursive_query) SELECT * FROM subquery_name;	Perform recursive queries (e.g., hierarchical data).	WITH RECURSIVE EmployeeHierarchy AS (SELECT id, manager_id FROM employees WHERE manager_id IS NULL UNION ALL SELECT e.id, e.manager_id FROM employees e INNER JOIN EmployeeHierarchy eh ON e.manager_id = eh.id) SELECT * FROM EmployeeHierarchy;