CASE: Perform Conditional Logic In A Query

```
SELECT
 first_name,
 last_name,
 CASE
   WHEN salary > 55000 THEN 'High'
   WHEN salary > 50000 THEN 'Medium'
   ELSE 'Low'
 END AS salary_category
FROM employees;
```

UPDATE: Modify Existing Records In A Table

```
UPDATE employees
SET salary = 55000.00
WHERE employee_id = 1;
```

DELETE: Remove Records From A Table

```
DELETE FROM employees
WHERE employee_id = 5;
```

Joins in SQL

Explore different join types to seamlessly merge data from multiple tables in your SQL queries.

45. INNER JOIN: Retrieves Records That Have Matching Values in Both Tables

```
SELECT * FROM employees
INNER JOIN departments ON employees.department_id =
departments.department_id;
```

This query will retrieve records from both the employees and departments tables where there is a match on the department_id column.

46. LEFT JOIN: Retrieves All Records from the Left Table and the Matched Records from the Right Table

```
SELECT * FROM employees
LEFT JOIN departments ON employees.department_id =
departments.department_id;
```

This query will retrieve all records from the employees table and only the matching records from the departments table.

47. RIGHT JOIN: Retrieves All Records from the Right Table and the Matched Records from the Left Table

```
SELECT * FROM employees
RIGHT JOIN departments ON employees.department_id =
departments.department_id;
```

This query will retrieve all records from the departments table and only the matching records from the employees table.

48. FULL OUTER JOIN: Retrieves All Records When There Is a Match in Either the Left or Right Table

```
SELECT * FROM employees
```

```
FULL OUTER JOIN departments ON employees.department_id =
departments.department_id;
```

This query will retrieve all records from both the employees and departments tables, including unmatched records.

49. CROSS JOIN: Retrieves the Cartesian Product of the Two Tables

```
SELECT * FROM employees

CROSS JOIN departments;
```

This query will retrieve all possible combinations of records from the employees and departments tables.

50. SELF JOIN: Joins a Table to Itself

```
SELECT e1.first_name, e2.first_name

FROM employees e1, employees e2

WHERE e1.employee_id = e2.manager_id;
```

In this example, the employees table is joined to itself to find employees and their respective managers based on the manager_id column.

CREATE VIEW: Create a Virtual Table Based on the Result of a SELECT Query

```
CREATE VIEW high_paid_employees AS

SELECT *

FROM employees

WHERE salary > 60000;
```

Advanced Mixed Data in SQL

In the last we have complied all the imprtant queries under the one advanced SQL cheat sheet.

67. Stored Procedures: Precompiled SQL Statements That Can Be Executed with a Single Command

```
CREATE PROCEDURE get_employee_count()
BEGIN
 SELECT COUNT(*) FROM employees;
END;
```

This query creates a stored procedure named get_employee_count that returns the count of employees.

68. Triggers: Automatically Execute a Set of SQL Statements When a Specified Event Occurs

```
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
 SET NEW.creation_date = NOW();
END;
```

This query creates a trigger named before_employee_insert that sets the creation_date column to the current date and time before inserting a new employee record.