SQL Joins Cheatsheet

A quick reference guide to SQL joins, covering different types of joins, their usage, and examples. This cheatsheet provides a visual and practical understanding of how to combine data from multiple tables in SQL.

SQL Join Types

INNER JOIN

Returns rows when there is a match in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

The INNER JOIN keyword selects records that have matching values in both tables.

If columns have the same names in multiple tables, use table names to specify the columns

LEFT (OUTER) JOIN

Returns all rows from the left table, and the matched rows from the right table. If there is no match, the result is NULL on the right side.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

RIGHT (OUTER) JOIN

Returns all rows from the right table, and the matched rows from the left table. If there is no match, the result is NULL on the left side.

Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID
ORDER BY Customers.CustomerName;
```

The RIGHT JOIN keyword returns all records from the right table (Customers), even if there are no matches in the left table (Orders).

FULL (OUTER) JOIN

Returns all rows when there is a match in one of the tables. Combines the results of both LEFT and RIGHT outer joins.

Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The FULL OUTER JOIN keyword returns all records from both tables: Customers and Orders. It fills NULL where there is no match

SQL Join Conditions and Filtering

Using WHERE Clause with Joins

The `WHERE` clause can be used to filter records based on specific conditions after joining tables.

Example:

```
SELECT Orders.OrderID,
Customers.CustomerName
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID = Customers.CustomerID
WHERE Customers.Country = 'USA';
```

This query selects order IDs and customer names only for customers from the USA.

Use aliases for better readability when using the `WHERE` clause with joins.

JOIN with Multiple Conditions

Joins can include multiple conditions using `AND` or `OR` operators to refine the matching criteria.

Example:

```
SELECT Employees.FirstName,
Orders.OrderID
FROM Employees
INNER JOIN Orders ON
Employees.EmployeeID = Orders.EmployeeID
AND Orders.ShippedDate > '2023-01-01';
```

This query selects employees' first names and order IDs for orders shipped after January 1, 2023.

Using Aliases in Joins

Aliases can be used to give tables a temporary name, making queries shorter and more readable.

Syntax:

```
SELECT alias1.column_name,
alias2.column_name
FROM table1 AS alias1
JOIN table2 AS alias2
ON alias1.column_name =
alias2.column_name;
```

Example:

```
SELECT c.CustomerName, o.OrderID
FROM Customers AS c
INNER JOIN Orders AS o ON c.CustomerID =
o.CustomerID;
```

Use aliases, especially when joining tables with similar column names.

Advanced SQL Join Techniques

Self Join

A self join is used to join a table to itself, as if the table were two different tables.

Syntax:

```
SELECT column_name(s)
FROM table1 AS t1
JOIN table1 AS t2
ON t1.column_name = t2.column_name;
```

Example:

```
SELECT E1.Name AS EmployeeName, E2.Name
AS ManagerName
FROM Employees AS E1
INNER JOIN Employees AS E2 ON
E1.ManagerID = E2.EmployeeID;
```

This query finds the names of employees and their managers from the same `Employees` table.

Use self joins to compare values within the same table.

Cross Join

A cross join returns the Cartesian product of rows from the tables. Each row from the first table is combined with each row from the second table.

Syntax:

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT Customers.CustomerName,
Products.ProductName
FROM Customers
CROSS JOIN Products;
```

This query generates all possible combinations of customers and products.

Avoid using cross joins on large tables without a `WHERE` clause, as it can produce a very large result set.

Natural Join

A natural join is a join based on all columns having the same name and data type in both tables. The common columns must be specified with the same name in both tables. It implicitly joins rows based on matching values in all common columns.

Syntax:

```
SELECT column_name(s)
FROM table1
NATURAL JOIN table2;
```

Example:

```
SELECT *
FROM Customers
NATURAL JOIN Orders;
```

Practical SQL Join Examples

Joining Three Tables

SQL joins can be extended to combine data from more than two tables by chaining multiple `JOIN` clauses.

Example:

```
SELECT Customers.CustomerName,
Orders.OrderID, Shippers.ShipperName
FROM Customers
INNER JOIN Orders ON
Customers.CustomerID = Orders.CustomerID
INNER JOIN Shippers ON Orders.ShipperID
= Shippers.ShipperID;
```

This query combines customer names, order IDs, and shipper names from three different tables.

Complex Join with Subqueries

Subqueries can be used within `JOIN` clauses to filter or transform data before joining.

Example:

```
SELECT c.CustomerName, o.OrderID
FROM Customers AS c
INNER JOIN (
    SELECT OrderID, CustomerID
    FROM Orders
    WHERE OrderDate > '2023-01-01'
) AS o ON c.CustomerID = o.CustomerID;
```

This query selects customer names and order IDs for orders placed after January 1, 2023, using a subquery.

Using CASE Statements in Joins

The `CASE` statement can be used within joins to conditionally determine values based on join conditions or table data.

Example:

```
SELECT
    Customers.CustomerName,
    Orders.OrderID,
    CASE
        WHEN Orders.ShippedDate IS NULL
THEN 'Not Shipped'
        ELSE 'Shipped'
    END AS ShippingStatus
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID
= Orders.CustomerID;
```

This query selects customer names, order IDs, and a shipping status determined by whether the `ShippedDate` is null.