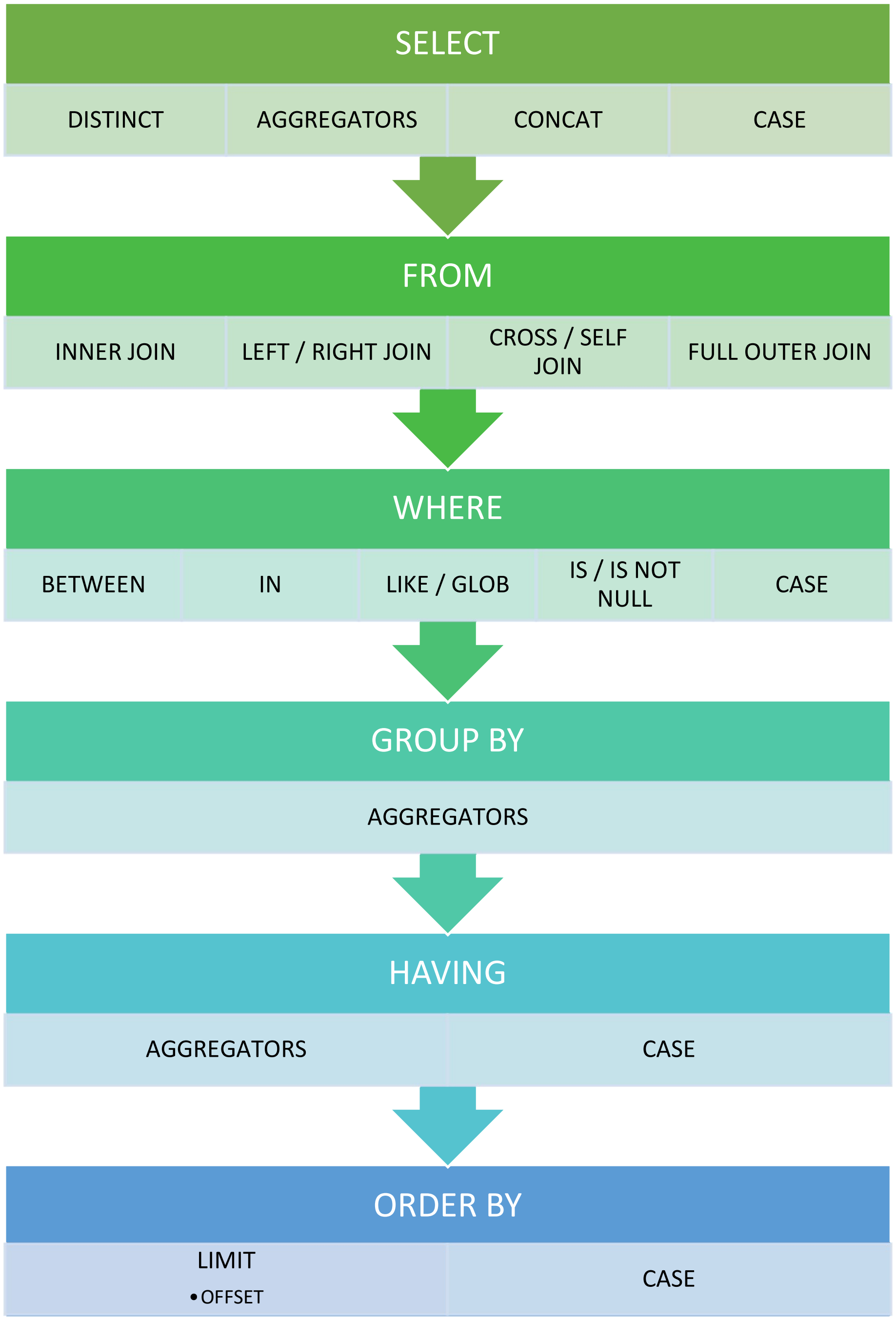
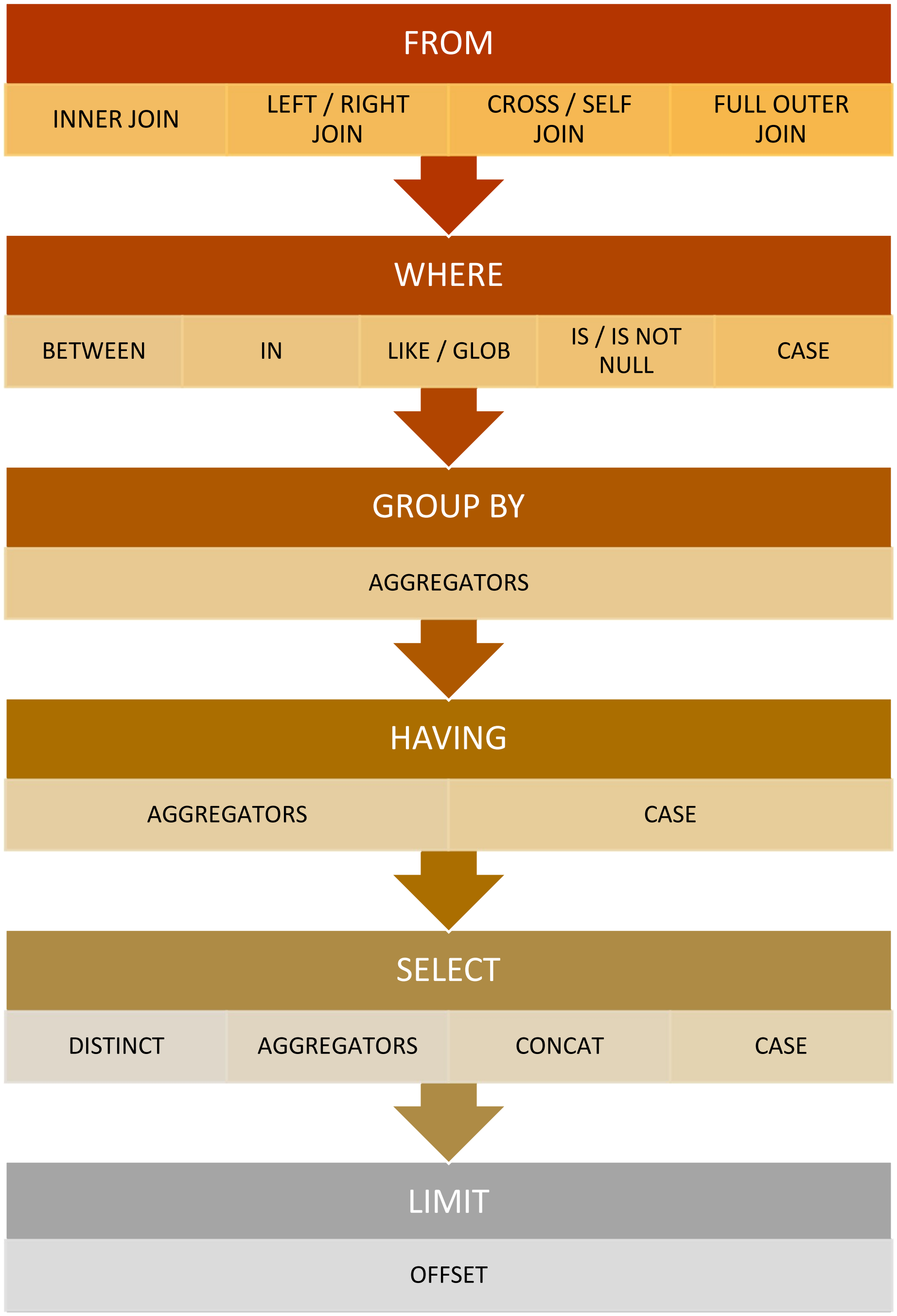
Lexical

SQL Order of Operations

Logical





GitHub: Tomalot, sam-genoese

SQL Order of Operations & Wildcards

SELECT

- Statement used to identify which columns to pull from a table
- (*) selects all columns in a table
- Can use aggregators (COUNT, SUM, MIN, MAX, and AVG)
    - Aggregators are NOT REQUIRED
- REQUIRED STATEMENT

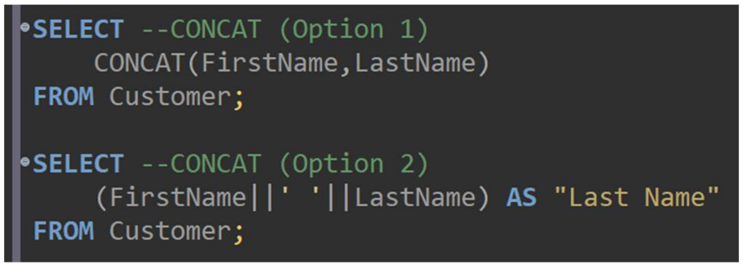
CONCAT (option 1)

- Adds two or more strings together
- Used with the SELECT statement
- NOT A REQUIRED STATEMENT

CONCAT (option 2)

- Identical to CONCAT (version 1)
- Can be indicated between columns with a double capital backslash (||)
    - Keystroke: SHIFT + \
- Add apostrophes (') and spaces between concatenated columns to insert white space
    - Parentheses are not required but clearly delineate concatenated columns
- Recommended to use aliases to make column names cleaner
- NOT A REQUIRED STATEMENT

EXAMPLES of CONCAT (option 1 and option 2):



```
SELECT --CONCAT (Option 1)
    CONCAT(FirstName,LastName)
FROM Customer;

SELECT --CONCAT (Option 2)
    (FirstName||' '||LastName) AS "Last Name"
FROM Customer;
```

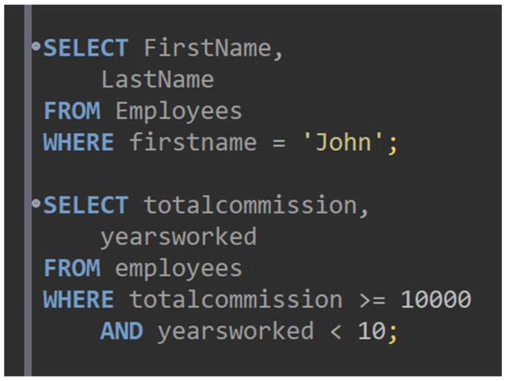
FROM

- Statement used to identify from which table columns are pulled
- Cannot use (*)
- REQUIRED STATEMENT

WHERE

- Clause that filters rows
- Can use operators (=, >, <, IN, BETWEEN, AND, OR, etc.)
- NOT A REQUIRED CLAUSE

EXAMPLES:

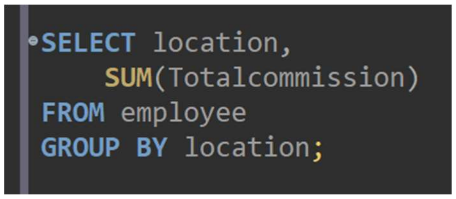


```
SELECT FirstName,
    LastName
FROM Employees
WHERE firstname = 'John';

SELECT totalcommission,
    yearsworked
FROM employees
WHERE totalcommission >= 10000
    AND yearsworked < 10;
```

GROUP BY

- Clause used to group rows that have the same values into summary rows
- Often used with aggregate functions (COUNT, SUM, AVG, etc.)
- Works well with HAVING clause
- Not necessary if you are not aggregating
- NOT A REQUIRED CLAUSE

EXAMPLE:

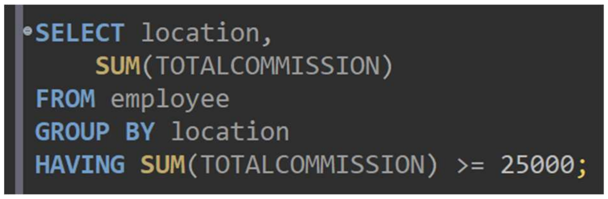


```
SELECT location,
    SUM(Totalcommission)
FROM employee
GROUP BY location;
```

HAVING

- Clause used to filter the results of a GROUP BY
- Unlike WHERE, HAVING cannot be used for regular filtering of records
- NOT A REQUIRED CLAUSE

EXAMPLE:



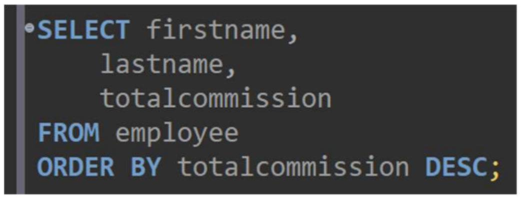
```
SELECT location,
    SUM(TOTALCOMMISSION)
FROM employee
GROUP BY location
HAVING SUM(TOTALCOMMISSION) >= 25000;
```

SQL Order of Operations & Wildcards

ORDER BY

- Clause is used to sort your records
    - Similar to sorting in Excel or on a webpage
- The default is ascending (LOW to HIGH)
    - Can be specified with ASC
- Use DESC to sort high to low
- Must specify a column to sort by
- NOT A REQUIRED CLAUSE

EXAMPLE:

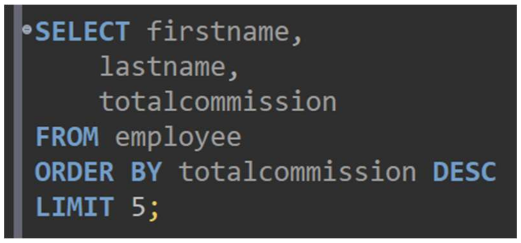


```sql
SELECT firstname,
    lastname,
    totalcommission
FROM employee
ORDER BY totalcommission DESC;
```

LIMIT

- Clause specifies a maximum number of rows for the output to display
- Best when used with ORDER BY (ASC or DESC) so that the output is organized
- NOT A REQUIRED CLAUSE

EXAMPLE:

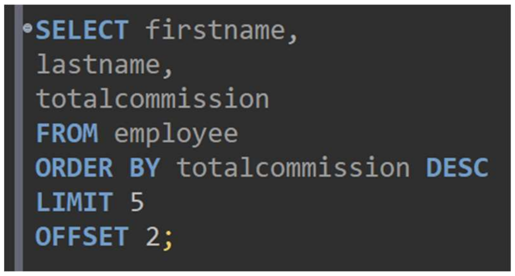


```sql
SELECT firstname,
    lastname,
    totalcommission
FROM employee
ORDER BY totalcommission DESC
LIMIT 5;
```

OFFSET

- Clause specifies how many rows to skip before displaying output data
- Must be used with LIMIT clause
- NOT A REQUIRED CLAUSE

EXAMPLE:



```sql
SELECT firstname,
lastname,
totalcommission
FROM employee
ORDER BY totalcommission DESC
LIMIT 5
OFFSET 2;
```

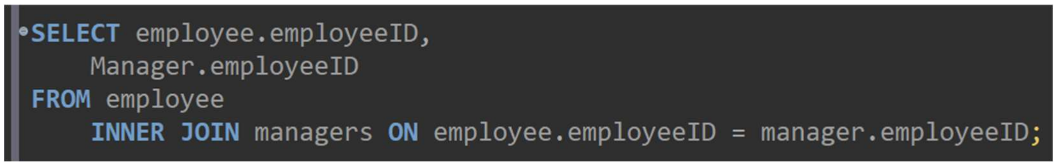
SQL Order of Operations & Wildcards

JOINS

INNER JOIN

- Only keeps rows that appear in both tables in the query

EXAMPLE:

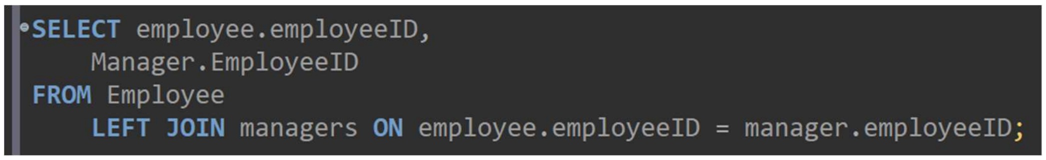


```
SELECT employee.employeeID,
    Manager.employeeID
FROM employee
    INNER JOIN managers ON employee.employeeID = manager.employeeID;
```

LEFT JOIN

- Returns everything that appears in the left table, regardless of a match or not
- Will return NULL values if no match found

EXAMPLE:



```
SELECT employee.employeeID,
    Manager.EmployeeID
FROM Employee
    LEFT JOIN managers ON employee.employeeID = manager.employeeID;
```

RIGHT JOIN

- Identical to a reversed LEFT JOIN

FULL JOIN

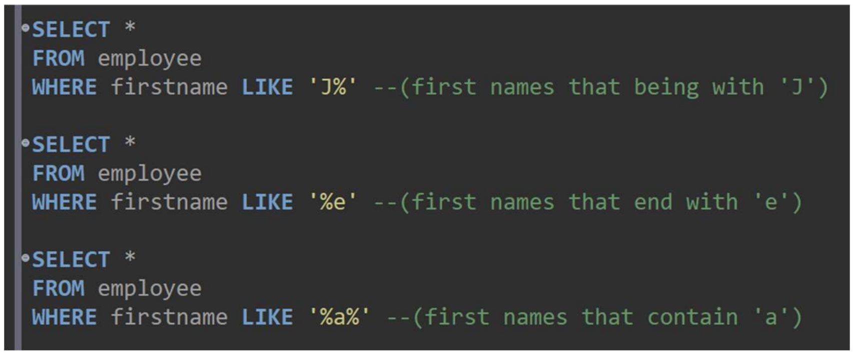
- Keeps all records from both tables, regardless of matches found

WILD CARDS

- Used in the WHERE clause along with the LIKE command
- Makes it easier to search for rows where you do not know the exact value
- Two main wildcards are (%) and (_)

(%): Looks for any number of preceding/following characters, including 0

EXAMPLES:

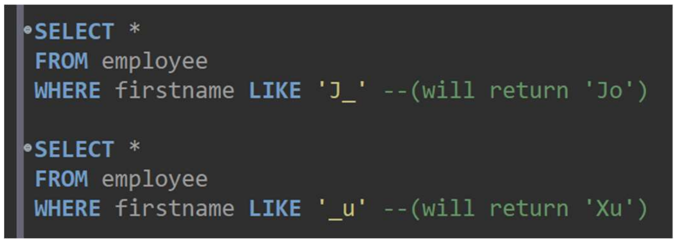


```
SELECT *
FROM employee
WHERE firstname LIKE 'J%' --(first names that being with 'J')

SELECT *
FROM employee
WHERE firstname LIKE '%e' --(first names that end with 'e')

SELECT *
FROM employee
WHERE firstname LIKE '%a%' --(first names that contain 'a')
```

SQL Order of Operations & Wildcards

(_): Looks for ONE preceding/following character instead of multiple

EXAMPLES:



```sql
SELECT *
FROM employee
WHERE firstname LIKE 'J_' --(will return 'Jo')

SELECT *
FROM employee
WHERE firstname LIKE '_u' --(will return 'Xu')
```