[SQL for Data Science] {CheatSheet}

1. Basic Data Exploration

- **View All Tables**: `SHOW TABLES;`
- **Preview Table Structure**: `DESCRIBE table_name;`
- **Select Entire Table**: `SELECT * FROM table_name;`
- **Select Specific Columns**: `SELECT column1, column2 FROM table_name;`
- **Count Total Records**: `SELECT COUNT(*) FROM table_name;`

2. Data Filtering and Sorting

- **Filter Rows with Conditions**: `SELECT * FROM table_name WHERE condition;`
- **Sort Data**: `SELECT * FROM table_name ORDER BY column ASC/DESC;`
- **Filter and Sort Combined**: `SELECT * FROM table_name WHERE condition ORDER BY column;`
- **Limiting Results**: `SELECT * FROM table_name LIMIT number;`
- **Filter with Multiple Conditions**: `SELECT * FROM table_name WHERE condition1 AND/OR condition2;`

3. Aggregation Functions

- **Calculate Average**: `SELECT AVG(column) FROM table_name;`
- **Sum a Column**: `SELECT SUM(column) FROM table_name;`
- **Find Maximum/Minimum Value**: `SELECT MAX(column), MIN(column) FROM table_name;`
- **Count Distinct Values**: `SELECT COUNT(DISTINCT column) FROM table_name;`
- **Group By Aggregations**: `SELECT column, COUNT(*), AVG(column) FROM table_name GROUP BY column;`

4. Advanced Aggregations

- **Using GROUP BY with Conditions**: `SELECT column, SUM(column) FROM table_name GROUP BY column HAVING condition;`

By: Waleed Mousa

- **Rollup for Subtotals**: `SELECT column1, column2, SUM(column3) FROM table_name GROUP BY column1, column2 WITH ROLLUP;`
- **CUBE for Cross-Tabulation**: `SELECT column1, column2, SUM(column3) FROM table_name GROUP BY CUBE (column1, column2);`
- **Window Functions (e.g., Running Total)**: `SELECT column, SUM(column) OVER (ORDER BY column) FROM table_name;`
- **Ranking within Group**: `SELECT column, RANK() OVER (PARTITION BY column1 ORDER BY column2) FROM table_name;`

5. Join Operations

- **Inner Join Between Tables**: `SELECT * FROM table1 INNER JOIN table2 ON table1.common_column = table2.common_column;`
- **Left Join (Including Unmatched Rows)**: `SELECT * FROM table1 LEFT JOIN table2 ON table1.common_column = table2.common_column;`
- **Right Join**: `SELECT * FROM table1 RIGHT JOIN table2 ON table1.common_column = table2.common_column;`
- **Full Outer Join**: `SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.common_column = table2.common_column;`
- **Self Join for Hierarchical Data**: `SELECT t1.column, t2.column FROM table t1 INNER JOIN table t2 ON t1.id = t2.parent_id;`

6. Subqueries and Nested Queries

- **Subquery in WHERE Clause**: `SELECT * FROM table WHERE column IN (SELECT column FROM another_table);`
- **Subquery in FROM Clause**: `SELECT * FROM (SELECT * FROM table) subquery;`
- **Subquery in SELECT Clause**: `SELECT column, (SELECT COUNT(*) FROM another_table) FROM table;`
- **Correlated Subqueries**: `SELECT * FROM table1 t1 WHERE EXISTS (SELECT * FROM table2 t2 WHERE t1.id = t2.foreign_id);`
- **Common Table Expressions (WITH Clause)**: `WITH cte AS (SELECT * FROM table) SELECT * FROM cte;`

7. String Manipulation

- **Concatenate Strings**: `SELECT CONCAT(string1, ' ', string2) FROM table_name;`
- **Substring Extraction**: `SELECT SUBSTRING(string_column, start, length) FROM table_name;`
- **Replace Text in a String**: `SELECT REPLACE(string_column, 'old', 'new') FROM table_name;`
- **Change String Case**: `SELECT UPPER(string_column), LOWER(string_column) FROM table_name;`
- **Trimming Whitespace**: `SELECT TRIM(string_column) FROM table_name;`

8. Working with Dates

- **Selecting Data in a Date Range**: `SELECT * FROM table_name WHERE date_column BETWEEN 'start_date' AND 'end_date';`
- **Extract Year, Month, Day**: `SELECT YEAR(date_column), MONTH(date_column), DAY(date_column) FROM table_name;`
- **Date Formatting**: `SELECT DATE_FORMAT(date_column, '%Y-%m-%d') FROM table_name;`
- **Calculating Age from Birthdate**: `SELECT DATEDIFF(CURDATE(), birthdate_column) FROM table_name;`
- **Time Difference between Dates**: `SELECT TIMEDIFF(date1, date2) FROM table_name;`

9. Data Analysis Techniques

- **Finding Percentiles**: `SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY column) FROM table_name;`
- **Linear Regression via SQL**: `SELECT REGR_SLOPE(y, x), REGR_INTERCEPT(y, x) FROM table_name;`
- **Correlation Coefficient**: `SELECT CORR(column1, column2) FROM table_name;`
- **Covariance**: `SELECT COVAR_POP(column1, column2) FROM table_name;`
- **Histogram Bin Analysis**: `SELECT WIDTH_BUCKET(numeric_column, min, max, num_buckets) FROM table_name;`

10. Conditional Logic and Case Statements

- **CASE Statement**: `SELECT CASE WHEN condition THEN result1 ELSE result2 END FROM table_name;`
- **IF Statement**: `SELECT IF(condition, value_if_true, value_if_false) FROM table_name;`
- **NULL Handling (IFNULL, COALESCE)**: `SELECT COALESCE(column, 'default_value') FROM table_name;`
- **Conditional Aggregation**: `SELECT column, SUM(CASE WHEN condition THEN value ELSE 0 END) FROM table_name GROUP BY column;`

11. Data Cleaning and Preparation

- **Removing Duplicates**: `SELECT DISTINCT * FROM table_name;`
- **Replacing NULLs with Default Value**: `SELECT IFNULL(column, 'default_value') FROM table_name;`
- **Standardize String Format**: `SELECT UPPER(TRIM(column)) FROM table_name;`
- **Handling Missing Data**: `SELECT * FROM table_name WHERE column IS NOT NULL;`
- **Flagging Data Anomalies**: `SELECT *, CASE WHEN column NOT IN (expected_values) THEN 'anomaly' ELSE 'ok' END FROM table_name;`

12. Advanced Data Filtering

- **Using LIKE for Pattern Matching**: `SELECT * FROM table_name WHERE column LIKE '%pattern%';`
- **Filtering with Regular Expressions**: `SELECT * FROM table_name WHERE column REGEXP 'regexp_pattern';`
- **Filtering with IN for Multiple Values**: `SELECT * FROM table_name WHERE column IN ('value1', 'value2');`
- **Complex Conditions Using AND, OR, NOT**: `SELECT * FROM table_name WHERE condition1 AND (condition2 OR condition3) NOT condition4;`
- **Range Queries with BETWEEN**: `SELECT * FROM table_name WHERE column BETWEEN lower AND upper;`

13. Data Transformation

- **Arithmetic Operations**: `SELECT column1 + column2, column1 * column2 FROM table_name;`

- **Data Type Conversion (CAST, CONVERT)**: `SELECT CAST(column AS datatype) FROM table_name;`
- **Normalization (e.g., Min-Max Scaling)**: `SELECT (column - MIN(column)) / (MAX(column) - MIN(column)) FROM table_name;`
- **Pivoting Data (CASE or PIVOT in some SQL dialects)**: `SELECT SUM(CASE WHEN condition THEN value ELSE 0 END) FROM table_name GROUP BY column;`
- **Unpivoting Data (UNPIVOT in some SQL dialects)**: `SELECT * FROM table_name UNPIVOT(value FOR column IN (column1, column2));`

14. Joining and Merging Data

- **Join Multiple Tables**: `SELECT * FROM table1 JOIN table2 ON table1.id = table2.id JOIN table3 ON table1.id = table3.id;`
- **Left Join with Filtering**: `SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id WHERE table2.id IS NULL;`
- **Join Using Using Clause**: `SELECT * FROM table1 JOIN table2 USING (common_column);`
- **Join with Aggregate Functions**: `SELECT table1.column, AVG(table2.column) FROM table1 JOIN table2 ON table1.id = table2.id GROUP BY table1.column;`
- **Join with Subqueries**: `SELECT * FROM table1 JOIN (SELECT id, COUNT(*) FROM table2 GROUP BY id) sub ON table1.id = sub.id;`

15. Advanced Subqueries

- **Nested Subqueries**: `SELECT * FROM (SELECT * FROM (SELECT * FROM table) sub1) sub2;`
- **Subquery as a Column**: `SELECT id, (SELECT COUNT(*) FROM table2 WHERE table2.id = table1.id) AS count FROM table1;`
- **Using EXISTS in Subquery**: `SELECT * FROM table1 WHERE EXISTS (SELECT * FROM table2 WHERE table1.id = table2.id);`

16. Working with Arrays and JSON

- **Query JSON Data**: `SELECT json_column->>'$.key' FROM table_name;`
- **Expand JSON Array to Rows**: `SELECT json_array_elements_text(json_column) FROM table_name;`

17. Handling Large Datasets

- **Efficient Pagination with Keyset**: `SELECT * FROM table_name WHERE id > last_seen_id ORDER BY id LIMIT page_size;`
- **Query Partitioning for Parallel Processing**: `SELECT * FROM table_name WHERE MOD(id, partition_count) = partition_index;`

18. Data Import/Export

- **Import Data from CSV**: `COPY table_name FROM '/path/to/csv_file.csv' DELIMITER ',' CSV;`
- **Export Data to CSV**: `COPY (SELECT * FROM table_name) TO '/path/to/csv_file.csv' DELIMITER ',' CSV HEADER;`

19. Database and Schema Management

- **Create New Schema**: `CREATE SCHEMA schema_name;`
- **Set Default Schema**: `SET search_path TO schema_name;`
- **List All Schemas**: `SELECT schema_name FROM information_schema.schemata;`

20. Advanced String Functions

- **Regular Expression Substring**: `SELECT REGEXP_SUBSTR(string_column, 'pattern') FROM table_name;`
- **String Aggregation**: `SELECT STRING_AGG(column, ', ') FROM table_name GROUP BY group_column;`
- **Split String into Array**: `SELECT STRING_TO_ARRAY(string_column, delimiter) FROM table_name;`

21. Geospatial Data Queries

- **Calculate Distance Between Two Points**: `SELECT ST_Distance(geom1, geom2) FROM table_name;`
- **Find Points Within a Radius**: `SELECT * FROM table_name WHERE ST_DWithin(geom, reference_geom, radius);`

22. Time Series Data

- **Time Series Aggregation**: `SELECT date_trunc('hour', time_column), SUM(value) FROM table_name GROUP BY 1;`
- **Lag and Lead Functions**: `SELECT time_column, value, LAG(value) OVER (ORDER BY time_column), LEAD(value) OVER (ORDER BY time_column) FROM table_name;`

23. Advanced Analytics Functions

- **Cumulative Distribution**: `SELECT value, CUME_DIST() OVER (ORDER BY value) FROM table_name;`
- **Percentile Calculation**: `SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY value) OVER () FROM table_name;`

24. Performance Optimization

- **Using Materialized Views**: `CREATE MATERIALIZED VIEW view_name AS SELECT * FROM table_name;`
- **Index Creation for Faster Queries**: `CREATE INDEX index_name ON table_name (column);`

25. Dynamic SQL and Stored Procedures

- **Execute Dynamic SQL**: `EXECUTE IMMEDIATE dynamic_sql;`
- **Create and Call Stored Procedure**: `CREATE PROCEDURE procedure_name AS BEGIN SQL_statements END; CALL procedure_name();`

26. Data Anonymization

- **Randomizing Sensitive Data**: `UPDATE table_name SET column = RANDOM() * range + offset WHERE condition;`
- **Masking Personal Data**: `UPDATE table_name SET email = REGEXP_REPLACE(email, '@.*', '@example.com');`

27. User and Access Management

- **Create Database User**: `CREATE USER username WITH PASSWORD 'password';`

- **Grant Privileges to User**: `GRANT SELECT, INSERT ON table_name TO username;`

28. Query Logging and Audit

- **Logging Queries for Audit**: `SET log_statement = 'all';`
- **Review Query Logs**: `SELECT * FROM pg_stat_activity WHERE query != '<IDLE>';`