Server

Cheat Sheet

for Data Engineers

Abhishek Agarwal
Data Engineer

Basic SQL Server Commands

✅ Create a Database

```
CREATE DATABASE my_database;
```

✅ Use a Database

```
USE my_database;
```

✅ Create a Table

```
CREATE TABLE customers (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    created_at DATETIME DEFAULT GETDATE()
);
```

✅ Insert Data

```
INSERT INTO customers (id, name, email)
VALUES (1, 'John Doe', 'john@example.com');
```

✅ Select Data

```
SELECT * FROM customers;
```

✅ Update Data

```
UPDATE customers
SET email = 'newemail@example.com'
WHERE id = 1;
```

✅ Delete Data

```
DELETE FROM customers WHERE id = 1;
```

Joins in SQL Server

- **INNER JOIN** – Returns matching rows from both tables

```
SELECT orders.id, customers.name
FROM orders
INNER JOIN customers ON orders.customer_id =
customers.id;
```

- **LEFT JOIN** – Returns all rows from the left table, with matching rows from the right

```
SELECT customers.name, orders.id
FROM customers
LEFT JOIN orders ON customers.id = orders.customer_id;
```

- **RIGHT JOIN** – Returns all rows from the right table, with matching rows from the left

```
SELECT customers.name, orders.id
FROM customers
RIGHT JOIN orders ON customers.id = orders.customer_id;
```

- **FULL JOIN** – Returns all rows when there is a match in either table

```
SELECT customers.name, orders.id
FROM customers
FULL JOIN orders ON customers.id = orders.customer_id;
```

Aggregate Functions

✅ Common Aggregations

```
SELECT COUNT(*) FROM orders; -- Total records
SELECT AVG(price) FROM products; -- Average price
SELECT MAX(price) FROM products; -- Maximum price
SELECT MIN(price) FROM products; -- Minimum price
SELECT SUM(price) FROM products; -- Total sum
```

✅ GROUP BY Example

```
SELECT category, COUNT(*) AS total_products
FROM products
GROUP BY category;
```

✅ HAVING Clause (Filter Aggregations)

```
SELECT category, COUNT(*) AS total_products
FROM products
GROUP BY category
HAVING COUNT(*) > 10;
```

Window Functions (Advanced Analytics)

✅ **ROW_NUMBER()** – Assigns row numbers

```
SELECT name, salary,
      ROW_NUMBER() OVER (PARTITION BY department
ORDER BY salary DESC) AS rank
FROM employees;
```

✅ **RANK()** – Ranks data with gaps

```
SELECT name, salary,
      RANK() OVER (PARTITION BY department ORDER BY
salary DESC) AS rank
FROM employees;
```

✅ **DENSE_RANK()** – Ranks without gaps

```
SELECT name, salary,
    DENSE_RANK() OVER (PARTITION BY department
ORDER BY salary DESC) AS rank
FROM employees;
```

✅ **LEAD() & LAG()** – Fetch next/previous row values

```
SELECT name, salary,
    LAG(salary) OVER (PARTITION BY department ORDER BY
salary) AS prev_salary,
    LEAD(salary) OVER (PARTITION BY department ORDER
BY salary) AS next_salary
FROM employees;
```

Performance Tuning Tips

- Indexing for Faster Queries

```
CREATE INDEX idx_customer_name ON
customers(name);
```

- **Avoid SELECT *** – Use specific columns

```
SELECT name, email FROM customers;
```

- Use EXISTS Instead of COUNT for Checking Existence

```
IF EXISTS (SELECT 1 FROM customers WHERE email =
'test@example.com')
   PRINT 'Email Exists';
```

- Optimize Joins with Indexed Columns

```
SELECT * FROM orders
INNER JOIN customers ON orders.customer_id =
customers.id
WHERE customers.name = 'John Doe';
```

- Use Temp Tables for Complex Queries

```
CREATE TABLE #TempResults (id INT, name
VARCHAR(100));
INSERT INTO #TempResults
SELECT id, name FROM customers WHERE name LIKE 'J%';
```

Follow for more content like this

Abhishek Agarwal

Azure Data Engineer