# SQL Portfolio (A-Z) - Interview Showcase

Prepared by: Tushar Kalra

Target: Business Analyst / Product Analyst / APM

This portfolio demonstrates end-to-end SQL capability on a realistic retail/fintech-style schema.

It includes: schema design (DDL), sample data inserts, and a curated set of queries across all major SQL topics.

Each example includes a short intent/explanation and business relevance note so an interviewer can see practical value.

All SQL shown is generic ANSI-style where possible (a few examples use MySQL-like functions for illustration).

# Contents

# 1) Sample Database Schema (DDL)

Domain: Hyperlocal/FinTech retail.

Key entities: Customers, Orders, OrderItems, Products, Categories, Suppliers, Payments, Employees, Departments.

```
-- Database: godmode_retail

CREATE TABLE Customers (
  CustomerID INT PRIMARY KEY,
  CustomerName VARCHAR(100),
  Email VARCHAR(120),
  City VARCHAR(60),
  State VARCHAR(60),
  RegistrationDate DATE,
  CreditLimit DECIMAL(10,2)
);

CREATE TABLE Categories (
  CategoryID INT PRIMARY KEY,
  CategoryName VARCHAR(60)
);

CREATE TABLE Suppliers (
  SupplierID INT PRIMARY KEY,
  SupplierName VARCHAR(100),
  City VARCHAR(60)
);

CREATE TABLE Products (
  ProductID INT PRIMARY KEY,
  ProductName VARCHAR(120),
  CategoryID INT,
  SupplierID INT,
  Price DECIMAL(10,2),
  FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID),
  FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)
);

CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  CustomerID INT,
  OrderDate DATE,
  OrderAmount DECIMAL(10,2),
  PaymentMethod VARCHAR(30),
  City VARCHAR(60),
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE OrderItems (
  OrderItemID INT PRIMARY KEY,
  OrderID INT,
  ProductID INT,
  Quantity INT,
  UnitPrice DECIMAL(10,2),
  FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
  FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Payments (
  PaymentID INT PRIMARY KEY,
  OrderID INT,
  PaidAmount DECIMAL(10,2),
  PaymentDate DATE,
  Status VARCHAR(20),
  FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

CREATE TABLE Departments (
  DeptID INT PRIMARY KEY,
  DeptName VARCHAR(60)
);

CREATE TABLE Employees (
  EmpID INT PRIMARY KEY,
  EmpName VARCHAR(100),
  DeptID INT,
  Salary DECIMAL(10,2),
  ManagerID INT NULL,
  FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
```

## 2) Sample Data Inserts

Small but realistic seed data so examples are concrete. (Extend as needed.)

```
-- Categories
INSERT INTO Categories VALUES
(1,'Grocery'),(2,'Electronics'),(3,'Beverages'),(4,'Bakery');

-- Suppliers
INSERT INTO Suppliers VALUES
(10,'FreshFarm','Bengaluru'),
(11,'ElectroHub','Bengaluru'),
(12,'DailyDelight','Mumbai');

-- Products
INSERT INTO Products VALUES
(100,'Basmati Rice 5kg',1,10,599.00),
(101,'LED Bulb 12W',2,11,149.00),
(102,'Cold Drink 2L',3,12,99.00),
(103,'Whole Wheat Bread',4,12,55.00);

-- Customers
INSERT INTO Customers VALUES
(1,'Aarav','aarav@example.com','Bengaluru','KA','2024-03-01',5000),
(2,'Riya','riya@example.com','Bengaluru','KA','2024-04-12',7000),
(3,'Kabir','kabir@example.com','Mumbai','MH','2024-05-10',4000),
(4,'Isha','isha@example.com','Delhi','DL','2024-06-20',6000);

-- Orders
INSERT INTO Orders VALUES
(1000,1,'2025-01-10',748.00,'UPI','Bengaluru'),
(1001,1,'2025-01-25',299.00,'CARD','Bengaluru'),
(1002,2,'2025-02-01',149.00,'COD','Bengaluru'),
(1003,3,'2025-02-12',99.00,'UPI','Mumbai'),
(1004,4,'2025-02-12',654.00,'UPI','Delhi');

-- OrderItems
INSERT INTO OrderItems VALUES
(1,1000,100,1,599.00),
(2,1000,101,1,149.00),
(3,1001,102,3,99.00),
(4,1002,101,1,149.00),
(5,1003,102,1,99.00),
(6,1004,100,1,599.00),
(7,1004,103,1,55.00);

-- Payments
INSERT INTO Payments VALUES
(9000,1000,748.00,'2025-01-10','PAID'),
(9001,1001,299.00,'2025-01-26','PAID'),
(9002,1002,0.00,'2025-02-01','PENDING'),
(9003,1003,99.00,'2025-02-12','PAID'),
(9004,1004,654.00,'2025-02-12','PAID');

-- Departments
INSERT INTO Departments VALUES (1,'Ops'),(2,'Data'),(3,'Sales');

-- Employees
INSERT INTO Employees VALUES
(101,'Dev',2,1200000,NULL),
(102,'Meera',2,900000,101),
(103,'Raj',1,700000,101),
(104,'Tina',3,800000,101);
```

# 3) SELECT and WHERE Essentials

## List customers from Bengaluru

```
SELECT CustomerID, CustomerName FROM Customers WHERE City = 'Bengaluru';
```

Business: Filter a dimension by exact match.

## Orders in Feb 2025

```
SELECT OrderID, OrderDate, OrderAmount FROM Orders WHERE OrderDate BETWEEN '2025-02-01' AND
'2025-02-28';
```

Business: Time-slicing for reporting windows.

## High value orders (>500)

```
SELECT OrderID, CustomerID, OrderAmount FROM Orders WHERE OrderAmount > 500;
```

Business: Identify premium orders for review.

# 4) JOINs (INNER, LEFT, RIGHT, FULL, CROSS, SELF)

### INNER JOIN: active customers with orders

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

Business: Find only revenue-generating customers.

### LEFT JOIN: all customers, flag orders

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

Business: Identify inactive customers (NULL orders).

### RIGHT JOIN: all orders, bring any customer info

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

Business: Audit orders even if customer master incomplete.

### FULL OUTER JOIN: union of both sides (if DB supports)

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

Business: Data reconciliation across systems.

### CROSS JOIN: combos of categories and months

```
SELECT CategoryName, m.month_str FROM Categories CROSS JOIN (SELECT '2025-01' AS month_str UNION SELECT '2025-02') m;
```

Business: Planning table for budgeting.

### SELF JOIN: employees and their managers

```
SELECT e.EmpName AS Employee, m.EmpName AS Manager FROM Employees e LEFT JOIN Employees m ON e.ManagerID = m.EmpID;
```

Business: Org chart / reporting lines.

# 5) GROUP BY and HAVING

### Total orders per customer

```
SELECT CustomerID, COUNT(*) AS TotalOrders FROM Orders GROUP BY CustomerID;
```

Business: Measure activity and segmentation.

### Revenue by city (only cities > 500 total)

```
SELECT City, SUM(OrderAmount) AS Revenue FROM Orders GROUP BY City HAVING SUM(OrderAmount) > 500;
```

Business: City-level performance filtering.

### Top product by quantity

```
SELECT oi.ProductID, SUM(oi.Quantity) AS Units FROM OrderItems oi GROUP BY oi.ProductID ORDER BY
Units DESC;
```

Business: Best-sellers listing.

# 6) Subqueries (Scalar, IN/NOT IN, EXISTS/NOT EXISTS, Correlated, Nested

### Scalar: orders above overall average

```
SELECT OrderID, OrderAmount FROM Orders WHERE OrderAmount > (SELECT AVG(OrderAmount) FROM Orders);
```

Business: Premium order identification.

### IN: customers who placed orders

```
SELECT CustomerID, CustomerName FROM Customers WHERE CustomerID IN (SELECT DISTINCT CustomerID FROM Orders);
```

Business: Active user segmentation.

### NOT IN: products never ordered

```
SELECT ProductID, ProductName FROM Products WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM OrderItems);
```

Business: Dead-stock detection.

### EXISTS: customers with any order

```
SELECT c.CustomerID, c.CustomerName FROM Customers c WHERE EXISTS (SELECT 1 FROM Orders o WHERE o.CustomerID = c.CustomerID);
```

Business: Presence/validation check.

### NOT EXISTS: orders with missing payment (audit)

```
SELECT o.OrderID, o.OrderAmount FROM Orders o WHERE NOT EXISTS (SELECT 1 FROM Payments p WHERE p.OrderID = o.OrderID);
```

Business: Compliance gap (unpaid or missing record).

### Correlated: orders above that customer's own average

```
SELECT o.OrderID, o.CustomerID, o.OrderAmount FROM Orders o WHERE o.OrderAmount > (SELECT AVG(o2.OrderAmount) FROM Orders o2 WHERE o2.CustomerID = o.CustomerID);
```

Business: Per-customer premium detection.

### Correlated: employees above their dept average salary

```
SELECT e.EmpID, e.EmpName, e.Salary, e.DeptID FROM Employees e WHERE e.Salary > (SELECT AVG(Salary) FROM Employees e2 WHERE e2.DeptID = e.DeptID);
```

Business: Top performers by department.

### Nested: second highest order amount

```
SELECT MAX(OrderAmount) AS SecondHighest FROM Orders WHERE OrderAmount < (SELECT MAX(OrderAmount) FROM Orders);
```

Business: Runner-up sizing.

# 7) CTEs (Common Table Expressions)

## Monthly revenue then prev-month delta (ANSI-ish)

```
WITH monthly AS (SELECT CAST(STRFTIME('%Y-%m', OrderDate) AS TEXT) AS ym, SUM(OrderAmount) AS revenue FROM Orders GROUP BY CAST(STRFTIME('%Y-%m', OrderDate) AS TEXT)) SELECT * FROM monthly;
```

Business: Readable pipelines; compute once, reuse.

## Top order per customer with ROW_NUMBER (shown in Windows section)

```
WITH ranked AS (SELECT CustomerID, OrderID, OrderAmount, ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY OrderAmount DESC) AS rn FROM Orders) SELECT * FROM ranked WHERE rn = 1;
```

Business: Classic top-N-per-group pattern.

# 8) Window Functions

## ROW_NUMBER and RANK: top orders per city

```
SELECT City, OrderID, OrderAmount, ROW_NUMBER() OVER (PARTITION BY City ORDER BY OrderAmount DESC)
AS rn, RANK() OVER (PARTITION BY City ORDER BY OrderAmount DESC) AS rnk FROM Orders;
```

Business: Leaderboards and top-N lists.

## LAG: difference vs previous order per customer

```
SELECT CustomerID, OrderDate, OrderAmount, OrderAmount - LAG(OrderAmount) OVER (PARTITION BY
CustomerID ORDER BY OrderDate) AS delta_vs_prev FROM Orders;
```

Business: Momentum and volatility signals.

## Running total of spend per customer

```
SELECT CustomerID, OrderDate, OrderAmount, SUM(OrderAmount) OVER (PARTITION BY CustomerID ORDER BY
OrderDate) AS running_spend FROM Orders;
```

Business: Wallet growth / CLV.

## NTILE quartiles by total spend

```
SELECT CustomerID, NTILE(4) OVER (ORDER BY SUM(OrderAmount) DESC) AS spend_quartile FROM Orders
GROUP BY CustomerID;
```

Business: Segmentation for campaigns.

# 9) CASE Expressions (Segmentation)

## Tag orders by value band

```
SELECT OrderID, OrderAmount, CASE WHEN OrderAmount >= 1000 THEN 'High' WHEN OrderAmount >= 500 THEN 'Medium' ELSE 'Low' END AS value_band FROM Orders;
```

Business: One-pass segmentation for dashboards.

## Customer tenure band

```
SELECT CustomerID, RegistrationDate, CASE WHEN RegistrationDate <= DATE('now','-365 day') THEN 'Veteran' WHEN RegistrationDate <= DATE('now','-180 day') THEN 'Settled' ELSE 'New' END AS tenure_band FROM Customers;
```

Business: Lifecycle messaging.

# 10) Date and String Functions

### Extract year/month/day (SQLite-style)

```
SELECT OrderID, STRFTIME('%Y', OrderDate) AS y, STRFTIME('%m', OrderDate) AS m, STRFTIME('%d', OrderDate) AS d FROM Orders;
```

Business: Calendar reporting.

### Build ym key and concat city label

```
SELECT STRFTIME('%Y-%m', OrderDate) AS ym, City || ', India' AS city_label FROM Orders;
```

Business: Cleaner labels for BI.

### Substring search in product name

```
SELECT ProductID, ProductName FROM Products WHERE ProductName LIKE '%Bread%';
```

Business: Merchandising and search.

## 11) Index and Performance Notes (brief)

- Indexes speed up WHERE/JOIN/GROUP BY on indexed columns but cost write speed/storage.
- Typical: index foreign keys (Orders.CustomerID, OrderItems.OrderID, ProductID), frequently filtered columns (Orders.OrderDate, City).
- Use EXPLAIN plans; avoid SELECT * in heavy queries; pre-aggregate with summary tables for BI dashboards.