SVKM'S NMIMS Deemed to be UNIVERSITY

Data Management – Project Report

Topic- Marketing Campaign Analysis

Submitted By: Group IX Date: 12-08-2025

Submitted To: Prof. Sujatha Ayyangar

Group Members - Dhruv Kushwaha - C022
Piyush Chandak - C028
Pragya Rao - C037
Apurv Agrawal - C060
Sakshi Rathi - C072
Suraj N - C075
Srijan Chakraborty - C083
Ayush Singh - C084

Course: MBA - Business Analytics
Division: C

1. Introduction

This project is submitted as part of the Data Management course for the MBA-Business Analytics program. It aims to provide hands-on experience in designing and managing databases using both relational (MySQL) and NoSQL (MongoDB) systems within a real-world business context. The chosen domain for this project is **Marketing**, with a focus on analyzing the effectiveness of various marketing campaigns.

2. Project Description

Domain: Marketing

Marketing campaign analysis is the process of evaluating the performance of promotional activities to understand their effectiveness, measure return on investment (ROI), and identify areas for improvement. It involves studying customer responses, sales impact, and channel performance to make informed marketing decisions.

This project uses SQL and NoSQL to analyze customer, product, and campaign data, focusing on Indian market references. It examines campaign reach, customer demographics, sales trends, and feedback to extract key insights. The findings aim to help optimize marketing strategies, improve targeting, and increase overall campaign profitability.

3. Entities & Attributes

The database is designed with the following entities and their corresponding attributes:

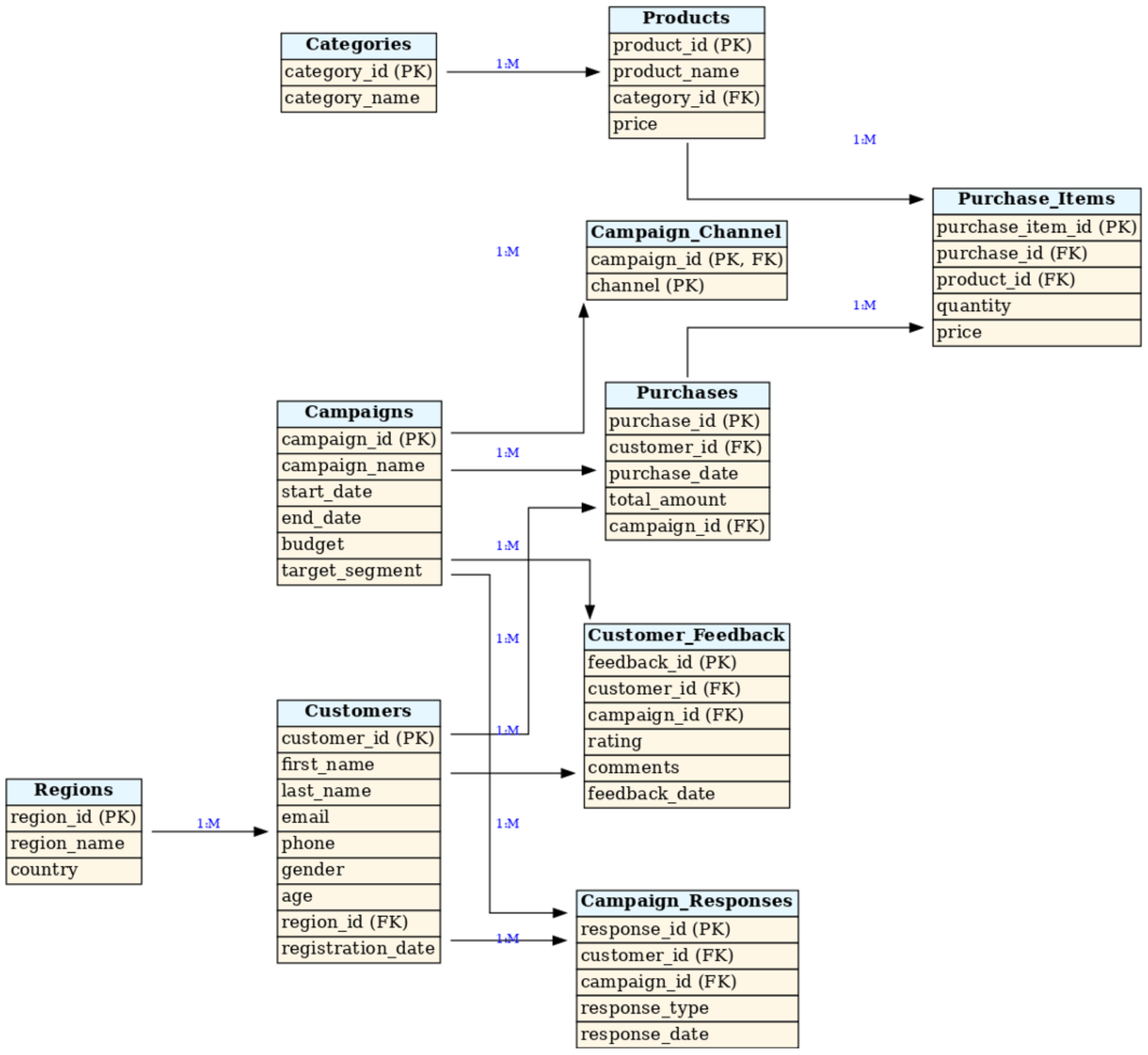
- **Regions**: Represents geographical areas.
- o **Attributes**: region_id (Primary Key), region_name, country
- **Customers**: Stores information about customers.
- o **Attributes**: customer_id (Primary Key), first_name, last_name, email, phone, gender, age, region_id (Foreign Key), registration_date
- **Categories**: Defines product categories.
- o **Attributes**: category_id (Primary Key), category_name
- **Products**: Contains details about the products offered.
- o **Attributes**: product_id (Primary Key), product_name, category_id (Foreign Key), price
- **Campaigns**: Holds information about marketing campaigns.
- o **Attributes**: campaign_id (Primary Key), campaign_name, start_date, end_date, budget, target_segment
- **Campaign_Channel**: A junction table representing the channels used for a campaign.
- o **Attributes**: campaign_id (Primary Key, Foreign Key), channel (Primary Key)
- **Purchases**: Records customer purchases.
- o **Attributes**: purchase_id (Primary Key), customer_id (Foreign Key), purchase_date, total_amount, campaign_id (Foreign Key)

- **Purchase_Items**: A junction table detailing individual items within a purchase.
  - **Attributes**: purchase_item_id (Primary Key), purchase_id (Foreign Key), product_id (Foreign Key), quantity, price
- **Customer_Feedback**: A junction table storing customer feedback for campaigns.
  - **Attributes**: feedback_id (Primary Key), customer_id (Foreign Key), campaign_id (Foreign Key), rating, comments, feedback_date
- **Campaign_Responses**: A junction table recording how customers responded to campaigns.
  - **Attributes**: response_id (Primary Key), customer_id (Foreign Key), campaign_id (Foreign Key), response_type, response_date

4. Relationships and Cardinality

- **Customers and Regions**: A Region can have many Customers (1:M). A Customer belongs to exactly one Region.
- **Products and Categories**: A Category can contain many Products (1:M). A Product belongs to exactly one Category.
- **Campaigns and Campaign_Channel**: A Campaign can use many Campaign_Channels (1:M).
- **Purchases and Customers**: A Customer can make many Purchases (1:M). A Purchase is made by exactly one Customer.
- **Purchases and Campaigns**: A Campaign can lead to many Purchases (1:M). A Purchase can be associated with one Campaign.
- **Purchases and Products (via Purchase_Items)**: This is a many-to-many relationship. A Purchase can include many Purchase_Items, and a Product can be part of many Purchase_Items.
- **Customers and Campaigns (via Customer_Feedback)**: This is a many-to-many relationship. A Customer can provide many Feedback entries, and a Campaign can receive many Feedback entries.
- **Customers and Campaigns (via Campaign_Responses)**: This is a many-to-many relationship. A Customer can have many Responses, and a Campaign can have many Responses.

5. ER Diagram


Categories
category_id (PK)
category_name
Products
product_id (PK)
product_name
category_id (FK)
price
Purchase_Items
purchase_item_id (PK)
purchase_id (FK)
product_id (FK)
quantity
price
Campaign_Channel
campaign_id (PK, FK)
channel (PK)
Purchases
purchase_id (PK)
customer_id (FK)
purchase_date
total_amount
campaign_id (FK)
Campaigns
campaign_id (PK)
campaign_name
start_date
end_date
budget
target_segment
Customer_Feedback
feedback_id (PK)
customer_id (FK)
campaign_id (FK)
rating
comments
feedback_date
Customers
customer_id (PK)
first_name
last_name
email
phone
gender
age
region_id (FK)
registration_date
Regions
region_id (PK)
region_name
country
Campaign_Responses
response_id (PK)
customer_id (FK)
campaign_id (FK)
response_type
response_date
1:M

6. MySQL Implementation

6.1. Database Design Flow & Schema Design

The relational database was designed using MySQL. The schema consists of 10 tables as described in the "Entities & Attributes" section. Primary Keys and Foreign Keys are established to maintain data integrity and define relationships between the tables.

6.2. SQL Script

Database and Table Creation

```
-- Database creation

CREATE DATABASE MarketingCampaignAnalysis;
USE MarketingCampaignAnalysis;


-- Table creation


-- Table: Regions

CREATE TABLE Regions (
   region_id VARCHAR(10) PRIMARY KEY,
   region_name VARCHAR(100),
   country VARCHAR(100)
);
DESC Regions;


-- Table: Customers

CREATE TABLE Customers (
   customer_id VARCHAR(10) PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   email VARCHAR(100) UNIQUE,
   phone VARCHAR(15),
   gender VARCHAR(10),
   age INT,
   region_id VARCHAR(10),
   registration_date DATE,
   FOREIGN KEY (region_id) REFERENCES Regions(region_id)
);
DESC Customers;


-- Table: Categories

CREATE TABLE Categories (
   category_id VARCHAR(10) PRIMARY KEY,
   category_name VARCHAR(100)
);
```

```
DESC Categories;


-- Table: Products

CREATE TABLE Products (
    product_id VARCHAR(10) PRIMARY KEY,
    product_name VARCHAR(100),
    category_id VARCHAR(10),
    price DECIMAL(10,2),
    FOREIGN KEY (category_id) REFERENCES Categories(category_id)
);
DESC Products;


-- Table: Campaigns

CREATE TABLE Campaigns (
    campaign_id VARCHAR(10) PRIMARY KEY,
    campaign_name VARCHAR(100),
    start_date DATE,
    end_date DATE,
    budget DECIMAL(12,2),
    target_segment VARCHAR(100)
);
ALTER TABLE Campaigns ADD COLUMN status VARCHAR(20) DEFAULT 'Planned';
DESC Campaigns;


-- Table: Campaign_Channel

CREATE TABLE Campaign_Channel (
    campaign_id VARCHAR(10),
    channel VARCHAR(50),
    PRIMARY KEY (campaign_id, channel),
    FOREIGN KEY (campaign_id) REFERENCES Campaigns(campaign_id)
);
DESC Campaign_Channel;


-- Table: Purchases

CREATE TABLE Purchases (
    purchase_id VARCHAR(10) PRIMARY KEY,
    customer_id VARCHAR(10),
    purchase_date DATE,
    total_amount DECIMAL(12,2),
    campaign_id VARCHAR(10),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (campaign_id) REFERENCES Campaigns(campaign_id)
);
DESC Purchases;


-- Table: Purchase_Items
```

```
CREATE TABLE Purchase_Items (
   purchase_item_id VARCHAR(10) PRIMARY KEY,
   purchase_id VARCHAR(10),
   product_id VARCHAR(10),
   quantity INT,
   price DECIMAL(10,2),
   FOREIGN KEY (purchase_id) REFERENCES Purchases(purchase_id),
   FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
DESC Purchase_Items;
```

-- Table: Customer_Feedback

```
CREATE TABLE Customer_Feedback (
   feedback_id VARCHAR(10) PRIMARY KEY,
   customer_id VARCHAR(10),
   campaign_id VARCHAR(10),
   rating INT CHECK (rating BETWEEN 1 AND 5),
   comments TEXT,
   feedback_date DATE,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
   FOREIGN KEY (campaign_id) REFERENCES Campaigns(campaign_id)
);
DESC Customer_Feedback;
```

-- Table: Campaign_Responses

```
CREATE TABLE Campaign_Responses (
   response_id VARCHAR(10) PRIMARY KEY,
   customer_id VARCHAR(10),
   campaign_id VARCHAR(10),
   response_type VARCHAR(50),
   response_date DATE,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
   FOREIGN KEY (campaign_id) REFERENCES Campaigns(campaign_id)
);
DESC Campaign_Responses;
```

Insert Sample Data

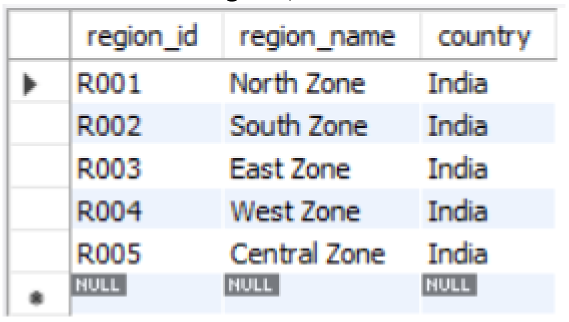
Data was loaded from CSV files into the respective tables using MySQL Workbench's Table Data Import Wizard. The import order was maintained to respect foreign key constraints:

1. Regions
2. Categories
3. Products
4. Campaigns
5. Campaign_Channel

6. Customers
7. Purchases
8. Purchase_Items
9. Customer_Feedback
10. Campaign_Responses

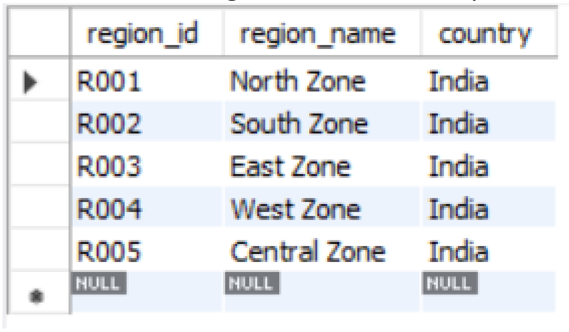
Execution of Queries & Screenshots

-- For table Regions:
-- Basic SELECT
SELECT * FROM Regions;



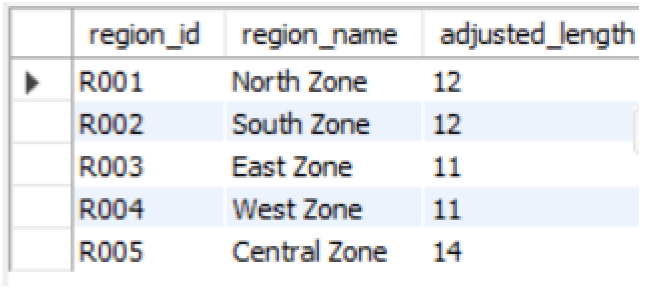
region_id	region_name	country
R001	North Zone	India
R002	South Zone	India
R003	East Zone	India
R004	West Zone	India
R005	Central Zone	India
NULL	NULL	NULL

-- WHERE condition
SELECT * FROM Regions WHERE country = 'India';



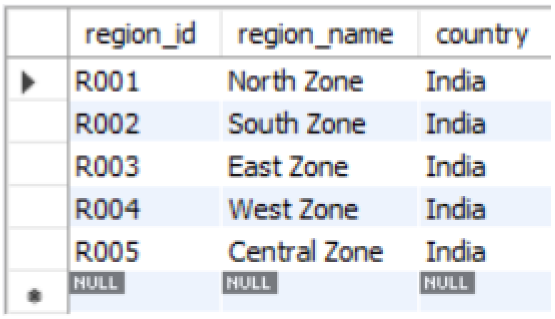
region_id	region_name	country
R001	North Zone	India
R002	South Zone	India
R003	East Zone	India
R004	West Zone	India
R005	Central Zone	India
NULL	NULL	NULL

-- Arithmetic (count regions by length of name, example)
SELECT region_id, region_name, LENGTH(region_name) + 2 AS adjusted_length FROM Regions;



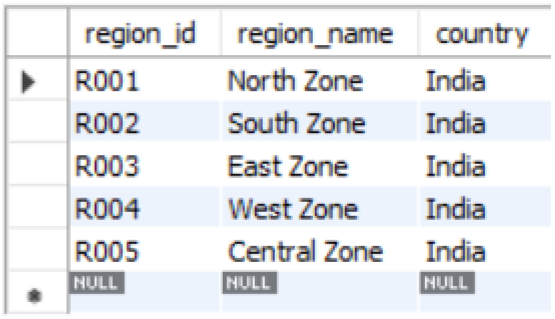
region_id	region_name	adjusted_length
R001	North Zone	12
R002	South Zone	12
R003	East Zone	11
R004	West Zone	11
R005	Central Zone	14

-- Logical operators
SELECT * FROM Regions
WHERE country = 'India' OR country = 'Nepal';



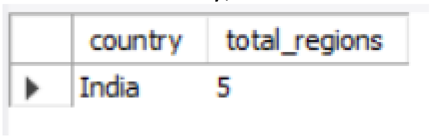
	region_id	region_name	country
▶	R001	North Zone	India
	R002	South Zone	India
	R003	East Zone	India
	R004	West Zone	India
	R005	Central Zone	India
*	NULL	NULL	NULL

```
-- LIKE, IN, LIMIT
SELECT * FROM Regions
WHERE country LIKE 'I%' OR country IN ('India', 'Nepal')
LIMIT 5;
```



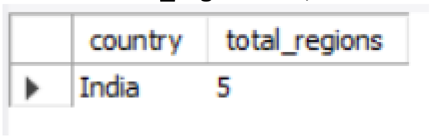
	region_id	region_name	country
▶	R001	North Zone	India
	R002	South Zone	India
	R003	East Zone	India
	R004	West Zone	India
	R005	Central Zone	India
*	NULL	NULL	NULL

```
-- GROUP BY
SELECT country, COUNT(region_id) AS total_regions
FROM Regions
GROUP BY country;
```



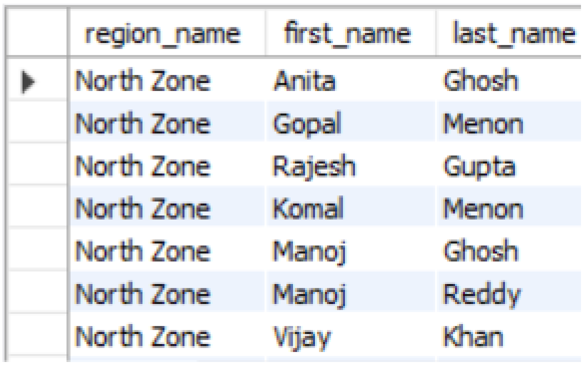
	country	total_regions
▶	India	5

```
-- GROUP BY with HAVING
SELECT country, COUNT(region_id) AS total_regions
FROM Regions
GROUP BY country
HAVING total_regions > 1;
```



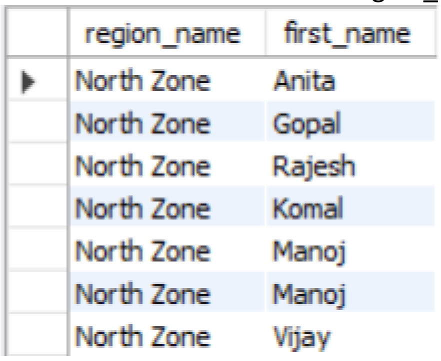
	country	total_regions
▶	India	5

```
-- INNER JOIN: Regions with customers
SELECT R.region_name, C.first_name, C.last_name
FROM Regions R
INNER JOIN Customers C ON R.region_id = C.region_id;
```



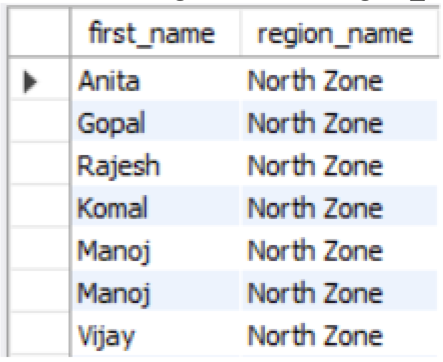
region_name	first_name	last_name
North Zone	Anita	Ghosh
North Zone	Gopal	Menon
North Zone	Rajesh	Gupta
North Zone	Komal	Menon
North Zone	Manoj	Ghosh
North Zone	Manoj	Reddy
North Zone	Vijay	Khan

```
-- LEFT JOIN: All regions, even without customers
SELECT R.region_name, C.first_name
FROM Regions R
LEFT JOIN Customers C ON R.region_id = C.region_id;
```



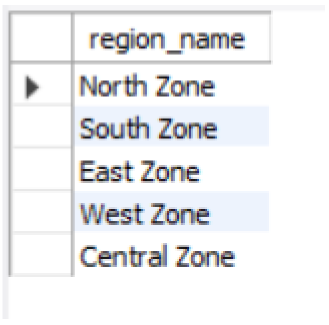
region_name	first_name
North Zone	Anita
North Zone	Gopal
North Zone	Rajesh
North Zone	Komal
North Zone	Manoj
North Zone	Manoj
North Zone	Vijay

```
-- RIGHT JOIN: All customers, with their regions
SELECT C.first_name, R.region_name
FROM Customers C
RIGHT JOIN Regions R ON R.region_id = C.region_id;
```



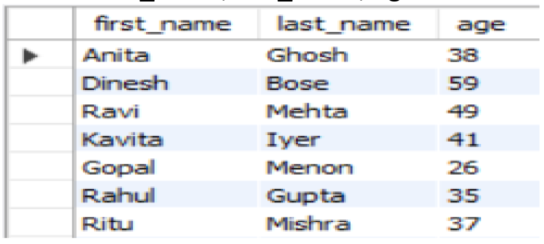
first_name	region_name
Anita	North Zone
Gopal	North Zone
Rajesh	North Zone
Komal	North Zone
Manoj	North Zone
Manoj	North Zone
Vijay	North Zone

```
-- Subquery: Regions with more than 5 customers
SELECT region_name
FROM Regions
WHERE region_id IN (
   SELECT region_id
   FROM Customers
   GROUP BY region_id
   HAVING COUNT(customer_id) > 5
);
```



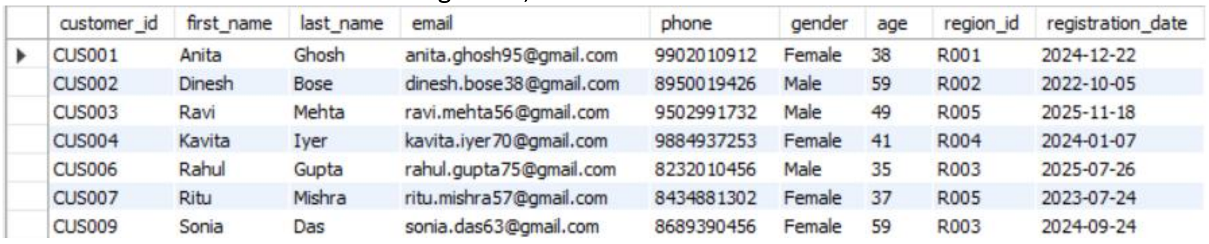
region_name
North Zone
South Zone
East Zone
West Zone
Central Zone

-- For table Customers
-- SELECT
SELECT first_name, last_name, age FROM Customers;



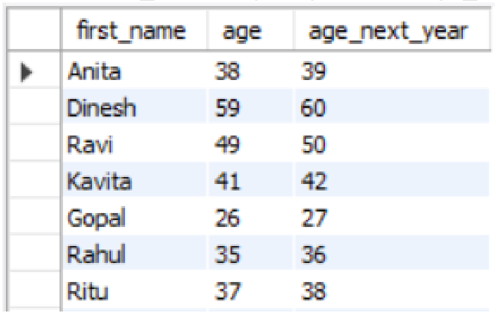
first_name	last_name	age
Anita	Ghosh	38
Dinesh	Bose	59
Ravi	Mehta	49
Kavita	Iyer	41
Gopal	Menon	26
Rahul	Gupta	35
Ritu	Mishra	37

-- WHERE condition
SELECT * FROM Customers WHERE age > 30;



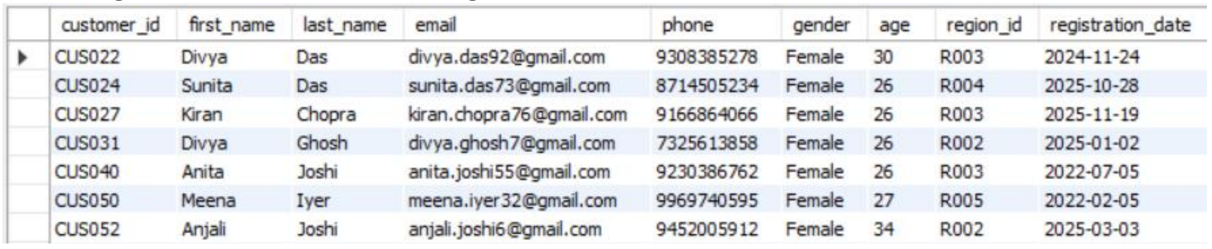
customer_id	first_name	last_name	email	phone	gender	age	region_id	registration_date
CUS001	Anita	Ghosh	anita.ghosh95@gmail.com	9902010912	Female	38	R001	2024-12-22
CUS002	Dinesh	Bose	dinesh.bose38@gmail.com	8950019426	Male	59	R002	2022-10-05
CUS003	Ravi	Mehta	ravi.mehta56@gmail.com	9502991732	Male	49	R005	2025-11-18
CUS004	Kavita	Iyer	kavita.iyer70@gmail.com	9884937253	Female	41	R004	2024-01-07
CUS006	Rahul	Gupta	rahul.gupta75@gmail.com	8232010456	Male	35	R003	2025-07-26
CUS007	Ritu	Mishra	ritu.mishra57@gmail.com	8434881302	Female	37	R005	2023-07-24
CUS009	Sonia	Das	sonia.das63@gmail.com	8689390456	Female	59	R003	2024-09-24

-- Arithmetic (age next year)
SELECT first_name, age, age + 1 AS age_next_year FROM Customers;



first_name	age	age_next_year
Anita	38	39
Dinesh	59	60
Ravi	49	50
Kavita	41	42
Gopal	26	27
Rahul	35	36
Ritu	37	38

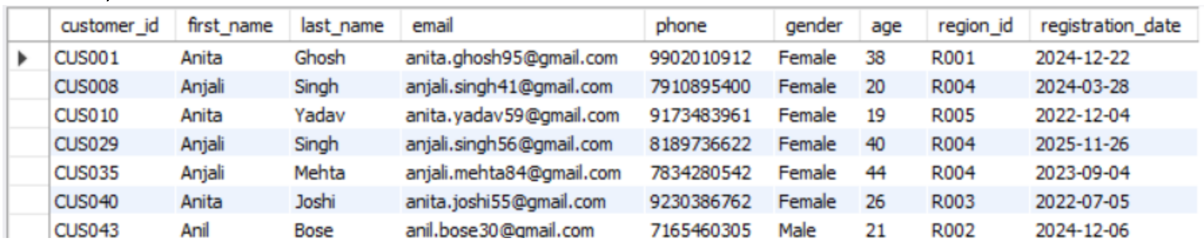
-- Logical operators
SELECT * FROM Customers
WHERE age BETWEEN 25 AND 35 AND gender = 'Female';



customer_id	first_name	last_name	email	phone	gender	age	region_id	registration_date
CUS022	Divya	Das	divya.das92@gmail.com	9308385278	Female	30	R003	2024-11-24
CUS024	Sunita	Das	sunita.das73@gmail.com	8714505234	Female	26	R004	2025-10-28
CUS027	Kiran	Chopra	kiran.chopra76@gmail.com	9166864066	Female	26	R003	2025-11-19
CUS031	Divya	Ghosh	divya.ghosh7@gmail.com	7325613858	Female	26	R002	2025-01-02
CUS040	Anita	Joshi	anita.joshi55@gmail.com	9230386762	Female	26	R003	2022-07-05
CUS050	Meena	Iyer	meena.iyer32@gmail.com	9969740595	Female	27	R005	2022-02-05
CUS052	Anjali	Joshi	anjali.joshi6@gmail.com	9452005912	Female	34	R002	2025-03-03

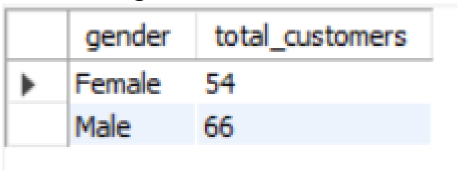
-- LIKE, IN, LIMIT
SELECT * FROM Customers

```sql
WHERE first_name LIKE 'A%' OR region_id IN (1, 3)
LIMIT 10;
```



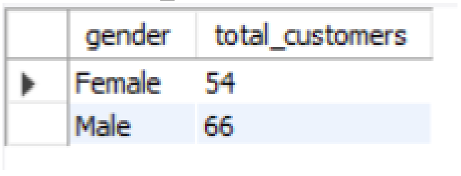
customer_id	first_name	last_name	email	phone	gender	age	region_id	registration_date
CUS001	Anita	Ghosh	anita.ghosh95@gmail.com	9902010912	Female	38	R001	2024-12-22
CUS008	Anjali	Singh	anjali.singh41@gmail.com	7910895400	Female	20	R004	2024-03-28
CUS010	Anita	Yadav	anita.yadav59@gmail.com	9173483961	Female	19	R005	2022-12-04
CUS029	Anjali	Singh	anjali.singh56@gmail.com	8189736622	Female	40	R004	2025-11-26
CUS035	Anjali	Mehta	anjali.mehta84@gmail.com	7834280542	Female	44	R004	2023-09-04
CUS040	Anita	Joshi	anita.joshi55@gmail.com	9230386762	Female	26	R003	2022-07-05
CUS043	Anil	Bose	anil.bose30@gmail.com	7165460305	Male	21	R002	2024-12-06

```sql
-- GROUP BY
SELECT gender, COUNT(customer_id) AS total_customers
FROM Customers
GROUP BY gender;
```



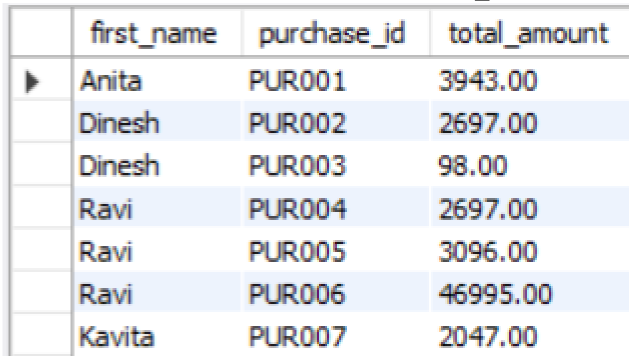
gender	total_customers
Female	54
Male	66

```sql
-- GROUP BY with HAVING
SELECT gender, COUNT(customer_id) AS total_customers
FROM Customers
GROUP BY gender
HAVING total_customers > 30;
```



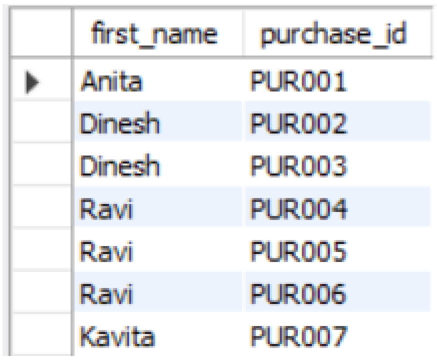
gender	total_customers
Female	54
Male	66

```sql
-- INNER JOIN: Customers with their purchases
SELECT C.first_name, P.purchase_id, P.total_amount
FROM Customers C
INNER JOIN Purchases P ON C.customer_id = P.customer_id;
```



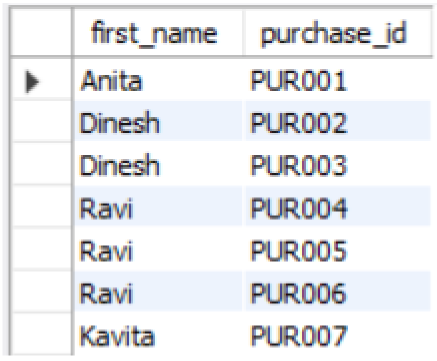
first_name	purchase_id	total_amount
Anita	PUR001	3943.00
Dinesh	PUR002	2697.00
Dinesh	PUR003	98.00
Ravi	PUR004	2697.00
Ravi	PUR005	3096.00
Ravi	PUR006	46995.00
Kavita	PUR007	2047.00

```sql
-- LEFT JOIN: All customers, even if no purchase
SELECT C.first_name, P.purchase_id
FROM Customers C
LEFT JOIN Purchases P ON C.customer_id = P.customer_id;
```



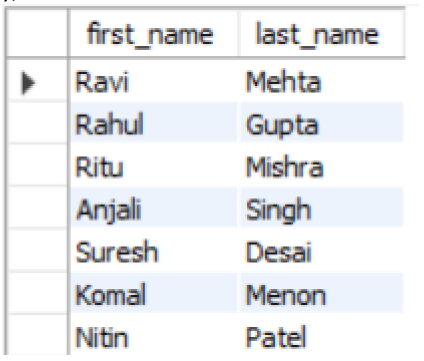
first_name	purchase_id
Anita	PUR001
Dinesh	PUR002
Dinesh	PUR003
Ravi	PUR004
Ravi	PUR005
Ravi	PUR006
Kavita	PUR007

```
-- RIGHT JOIN: Purchases ensuring all purchase records
SELECT P.purchase_id, C.first_name
FROM Purchases P
RIGHT JOIN Customers C ON C.customer_id = P.customer_id;
```



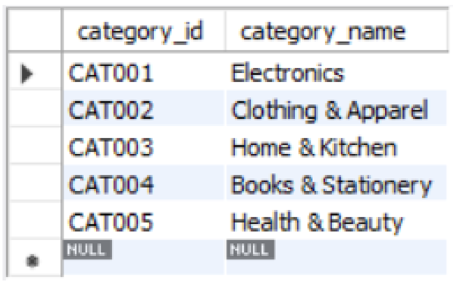
first_name	purchase_id
Anita	PUR001
Dinesh	PUR002
Dinesh	PUR003
Ravi	PUR004
Ravi	PUR005
Ravi	PUR006
Kavita	PUR007

```
-- Subquery: Customers who spent above average total
SELECT first_name, last_name
FROM Customers
WHERE customer_id IN (
   SELECT customer_id
   FROM Purchases
   GROUP BY customer_id
   HAVING SUM(total_amount) > (
      SELECT AVG(total_amount) FROM Purchases
   )
);
```



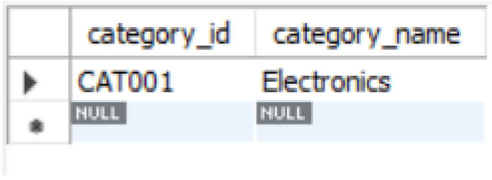
first_name	last_name
Ravi	Mehta
Rahul	Gupta
Ritu	Mishra
Anjali	Singh
Suresh	Desai
Komal	Menon
Nitin	Patel

```
-- For table Categories
-- SELECT
SELECT category_id, category_name FROM Categories;
```



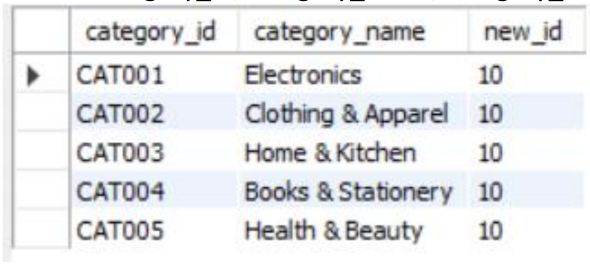
category_id	category_name
CAT001	Electronics
CAT002	Clothing & Apparel
CAT003	Home & Kitchen
CAT004	Books & Stationery
CAT005	Health & Beauty
NULL	NULL

```
-- WHERE condition
SELECT * FROM Categories WHERE category_name = 'Electronics';
```



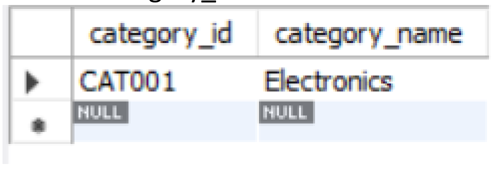
category_id	category_name
CAT001	Electronics
NULL	NULL

```
-- Arithmetic (example ID+10)
SELECT category_id, category_name, category_id + 10 AS new_id FROM Categories;
```



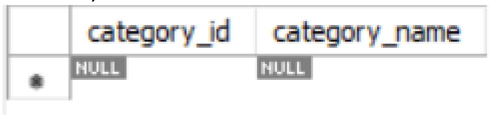
category_id	category_name	new_id
CAT001	Electronics	10
CAT002	Clothing & Apparel	10
CAT003	Home & Kitchen	10
CAT004	Books & Stationery	10
CAT005	Health & Beauty	10

```
-- Logical operators
SELECT * FROM Categories
WHERE category_name = 'Electronics' OR category_name = 'Clothing';
```



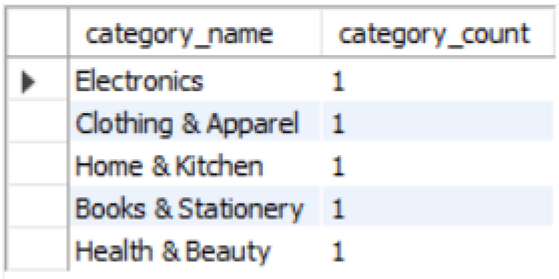
category_id	category_name
CAT001	Electronics
NULL	NULL

```
-- LIKE, IN, LIMIT
SELECT * FROM Categories
WHERE category_name LIKE '%wear%' OR category_id IN (2, 4)
LIMIT 5;
```



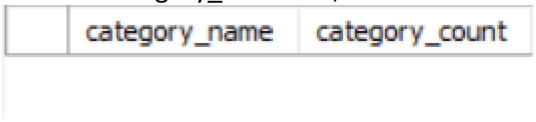
category_id	category_name
NULL	NULL

```
-- GROUP BY
SELECT category_name, COUNT(category_id) AS category_count
FROM Categories
GROUP BY category_name;
```



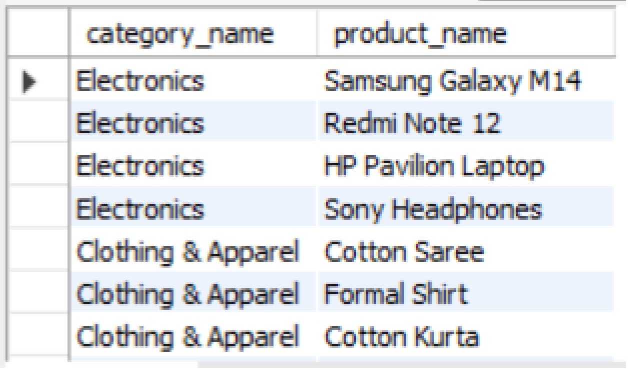
category_name	category_count
Electronics	1
Clothing & Apparel	1
Home & Kitchen	1
Books & Stationery	1
Health & Beauty	1

```
-- GROUP BY with HAVING
SELECT category_name, COUNT(category_id) AS category_count
FROM Categories
GROUP BY category_name
HAVING category_count > 1;
```



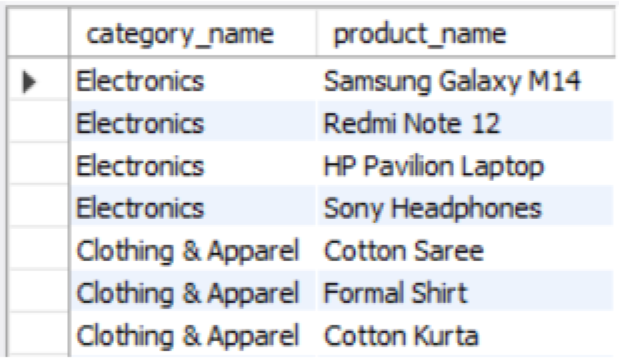
category_name	category_count

```
-- INNER JOIN: Categories with products
SELECT Cat.category_name, P.product_name
FROM Categories Cat
INNER JOIN Products P ON Cat.category_id = P.category_id;
```



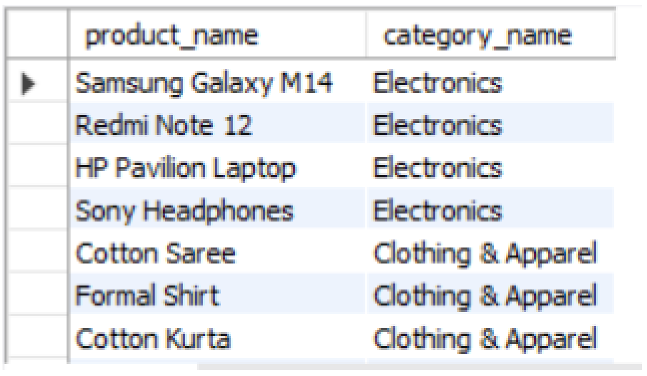
category_name	product_name
Electronics	Samsung Galaxy M14
Electronics	Redmi Note 12
Electronics	HP Pavilion Laptop
Electronics	Sony Headphones
Clothing & Apparel	Cotton Saree
Clothing & Apparel	Formal Shirt
Clothing & Apparel	Cotton Kurta

```
-- LEFT JOIN: All categories, even without products
SELECT Cat.category_name, P.product_name
FROM Categories Cat
LEFT JOIN Products P ON Cat.category_id = P.category_id;
```



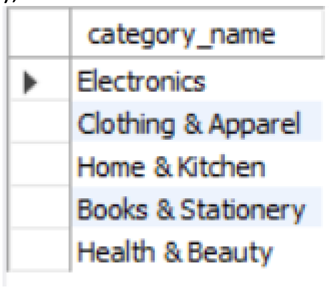
category_name	product_name
Electronics	Samsung Galaxy M14
Electronics	Redmi Note 12
Electronics	HP Pavilion Laptop
Electronics	Sony Headphones
Clothing & Apparel	Cotton Saree
Clothing & Apparel	Formal Shirt
Clothing & Apparel	Cotton Kurta

```
-- RIGHT JOIN: All products and their categories
SELECT P.product_name, Cat.category_name
FROM Products P
RIGHT JOIN Categories Cat ON Cat.category_id = P.category_id;
```



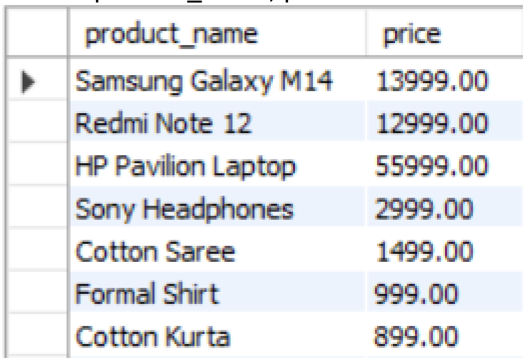
product_name	category_name
Samsung Galaxy M14	Electronics
Redmi Note 12	Electronics
HP Pavilion Laptop	Electronics
Sony Headphones	Electronics
Cotton Saree	Clothing & Apparel
Formal Shirt	Clothing & Apparel
Cotton Kurta	Clothing & Apparel

```
-- Subquery: Categories with more than 3 products
SELECT category_name
FROM Categories
WHERE category_id IN (
   SELECT category_id
   FROM Products
   GROUP BY category_id
   HAVING COUNT(product_id) > 3
);
```



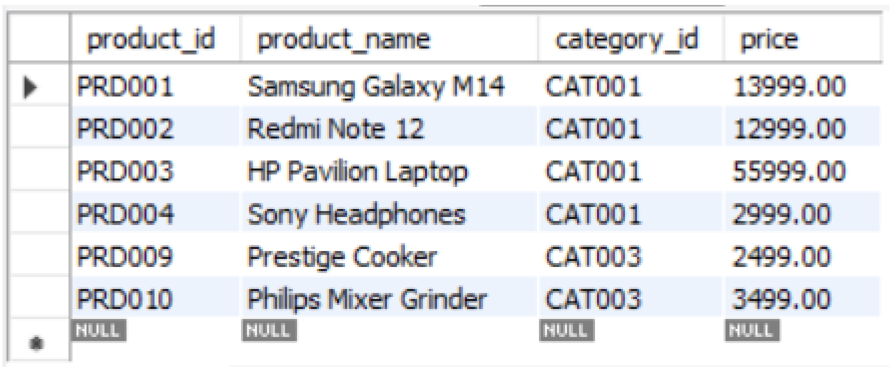
category_name
Electronics
Clothing & Apparel
Home & Kitchen
Books & Stationery
Health & Beauty

```
-- For table Products
-- SELECT
SELECT product_name, price FROM Products;
```



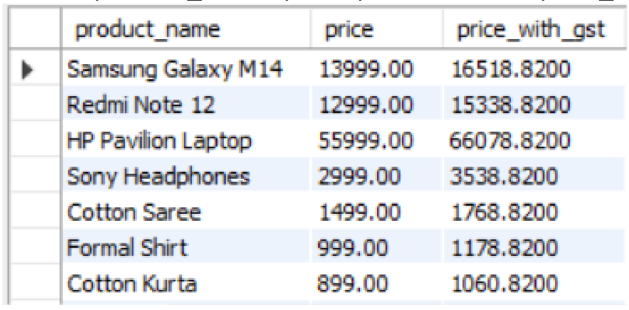
product_name	price
Samsung Galaxy M14	13999.00
Redmi Note 12	12999.00
HP Pavilion Laptop	55999.00
Sony Headphones	2999.00
Cotton Saree	1499.00
Formal Shirt	999.00
Cotton Kurta	899.00

```
-- WHERE condition
SELECT * FROM Products WHERE price > 2000;
```



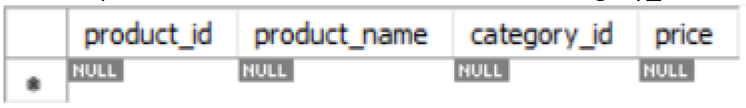
	product_id	product_name	category_id	price
▶	PRD001	Samsung Galaxy M14	CAT001	13999.00
	PRD002	Redmi Note 12	CAT001	12999.00
	PRD003	HP Pavilion Laptop	CAT001	55999.00
	PRD004	Sony Headphones	CAT001	2999.00
	PRD009	Prestige Cooker	CAT003	2499.00
	PRD010	Philips Mixer Grinder	CAT003	3499.00
*	NULL	NULL	NULL	NULL

```
-- Arithmetic
SELECT product_name, price, price * 1.18 AS price_with_gst FROM Products;
```



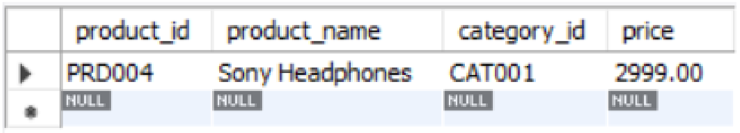
	product_name	price	price_with_gst
▶	Samsung Galaxy M14	13999.00	16518.8200
	Redmi Note 12	12999.00	15338.8200
	HP Pavilion Laptop	55999.00	66078.8200
	Sony Headphones	2999.00	3538.8200
	Cotton Saree	1499.00	1768.8200
	Formal Shirt	999.00	1178.8200
	Cotton Kurta	899.00	1060.8200

```
-- Logical operators
SELECT * FROM Products
WHERE price BETWEEN 1500 AND 3000 AND category_id = 2;
```



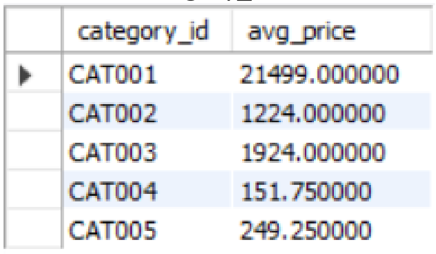
	product_id	product_name	category_id	price
*	NULL	NULL	NULL	NULL

```
-- LIKE, IN, LIMIT
SELECT * FROM Products
WHERE product_name LIKE '%Phone%' OR category_id IN (1, 3)
LIMIT 8;
```



	product_id	product_name	category_id	price
▶	PRD004	Sony Headphones	CAT001	2999.00
*	NULL	NULL	NULL	NULL

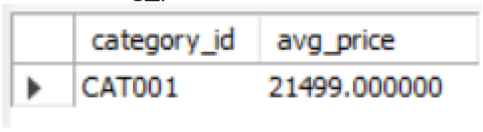
```
-- GROUP BY
SELECT category_id, AVG(price) AS avg_price
FROM Products
GROUP BY category_id;
```



	category_id	avg_price
▶	CAT001	21499.000000
	CAT002	1224.000000
	CAT003	1924.000000
	CAT004	151.750000
	CAT005	249.250000

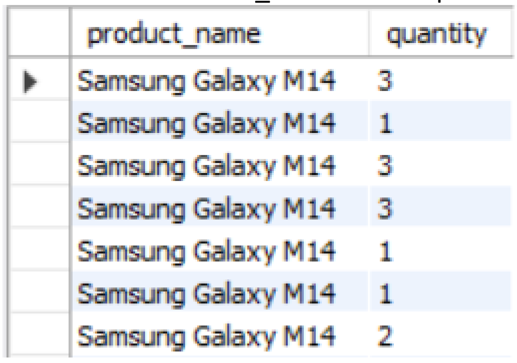
```
-- GROUP BY with HAVING
SELECT category_id, AVG(price) AS avg_price
FROM Products
GROUP BY category_id
```

```
HAVING avg_price > 2500;
```



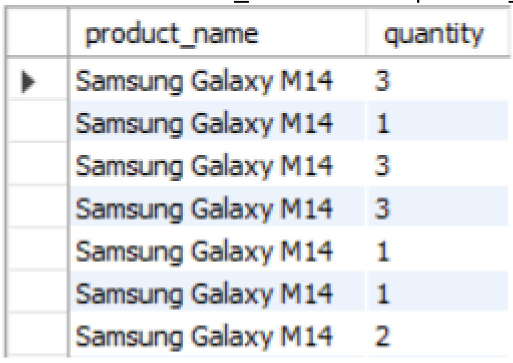
	category_id	avg_price
▶	CAT001	21499.000000

```
-- INNER JOIN: Products with purchase items
SELECT P.product_name, PI.quantity
FROM Products P
INNER JOIN Purchase_Items PI ON P.product_id = PI.product_id;
```



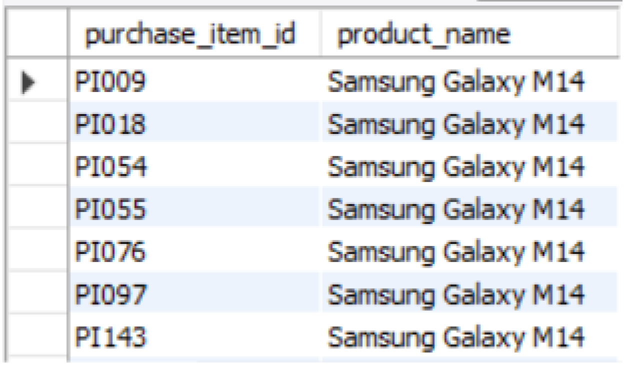
	product_name	quantity
▶	Samsung Galaxy M14	3
	Samsung Galaxy M14	1
	Samsung Galaxy M14	3
	Samsung Galaxy M14	3
	Samsung Galaxy M14	1
	Samsung Galaxy M14	1
	Samsung Galaxy M14	2

```
-- LEFT JOIN: All products, even if never purchased
SELECT P.product_name, PI.quantity
FROM Products P
LEFT JOIN Purchase_Items PI ON P.product_id = PI.product_id;
```



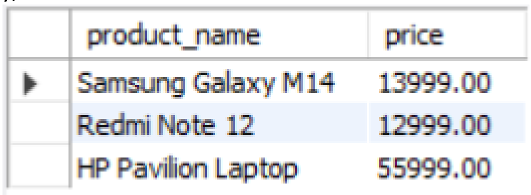
	product_name	quantity
▶	Samsung Galaxy M14	3
	Samsung Galaxy M14	1
	Samsung Galaxy M14	3
	Samsung Galaxy M14	3
	Samsung Galaxy M14	1
	Samsung Galaxy M14	1
	Samsung Galaxy M14	2

```
-- RIGHT JOIN: All purchase items with their product names
SELECT PI.purchase_item_id, P.product_name
FROM Purchase_Items PI
RIGHT JOIN Products P ON P.product_id = PI.product_id;
```



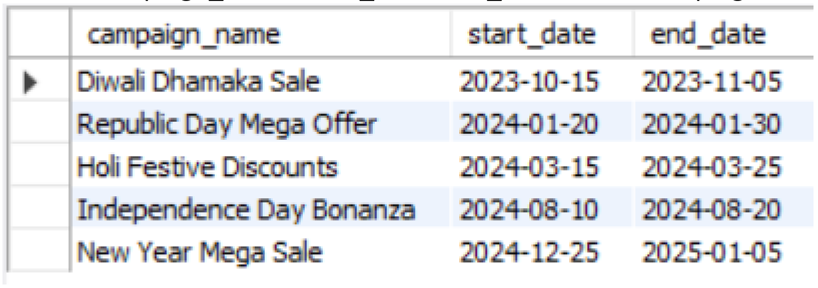
	purchase_item_id	product_name
▶	PI009	Samsung Galaxy M14
	PI018	Samsung Galaxy M14
	PI054	Samsung Galaxy M14
	PI055	Samsung Galaxy M14
	PI076	Samsung Galaxy M14
	PI097	Samsung Galaxy M14
	PI143	Samsung Galaxy M14

```
-- Subquery: Products with price above category average
SELECT product_name, price
FROM Products
WHERE price > (
    SELECT AVG(price)
    FROM Products
);
```



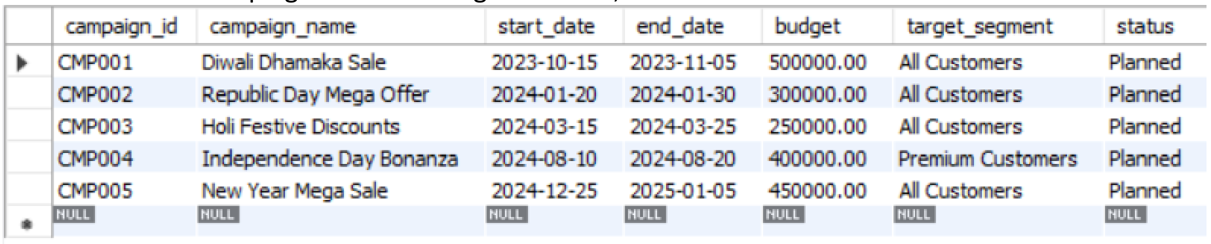
product_name	price
Samsung Galaxy M14	13999.00
Redmi Note 12	12999.00
HP Pavilion Laptop	55999.00

```
-- For table Campaigns
-- SELECT
SELECT campaign_name, start_date, end_date FROM Campaigns;
```



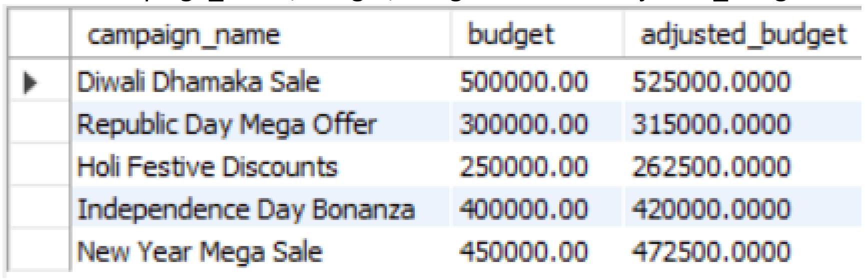
campaign_name	start_date	end_date
Diwali Dhamaka Sale	2023-10-15	2023-11-05
Republic Day Mega Offer	2024-01-20	2024-01-30
Holi Festive Discounts	2024-03-15	2024-03-25
Independence Day Bonanza	2024-08-10	2024-08-20
New Year Mega Sale	2024-12-25	2025-01-05

```
-- WHERE
SELECT * FROM Campaigns WHERE budget > 50000;
```



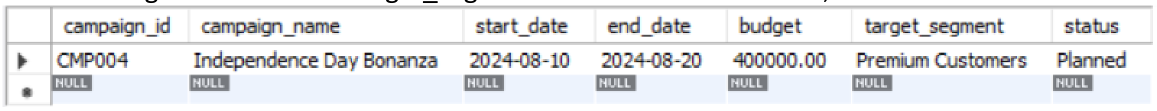
campaign_id	campaign_name	start_date	end_date	budget	target_segment	status
CMP001	Diwali Dhamaka Sale	2023-10-15	2023-11-05	500000.00	All Customers	Planned
CMP002	Republic Day Mega Offer	2024-01-20	2024-01-30	300000.00	All Customers	Planned
CMP003	Holi Festive Discounts	2024-03-15	2024-03-25	250000.00	All Customers	Planned
CMP004	Independence Day Bonanza	2024-08-10	2024-08-20	400000.00	Premium Customers	Planned
CMP005	New Year Mega Sale	2024-12-25	2025-01-05	450000.00	All Customers	Planned
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
-- Arithmetic
SELECT campaign_name, budget, budget * 1.05 AS adjusted_budget FROM Campaigns;
```



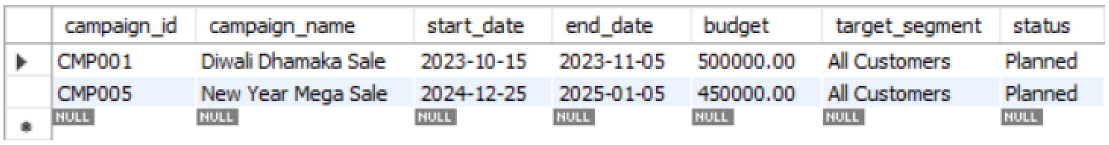
campaign_name	budget	adjusted_budget
Diwali Dhamaka Sale	500000.00	525000.0000
Republic Day Mega Offer	300000.00	315000.0000
Holi Festive Discounts	250000.00	262500.0000
Independence Day Bonanza	400000.00	420000.0000
New Year Mega Sale	450000.00	472500.0000

```
-- Logical operators
SELECT * FROM Campaigns
WHERE budget > 50000 AND target_segment = 'Premium Customers';
```



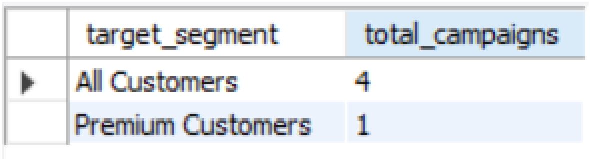
campaign_id	campaign_name	start_date	end_date	budget	target_segment	status
CMP004	Independence Day Bonanza	2024-08-10	2024-08-20	400000.00	Premium Customers	Planned
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
-- LIKE, IN, LIMIT
SELECT * FROM Campaigns
WHERE campaign_name LIKE '%Sale%' OR campaign_id IN (1, 4)
LIMIT 5;
```



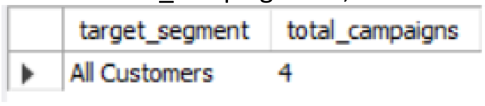
campaign_id	campaign_name	start_date	end_date	budget	target_segment	status
CMP001	Diwali Dhamaka Sale	2023-10-15	2023-11-05	500000.00	All Customers	Planned
CMP005	New Year Mega Sale	2024-12-25	2025-01-05	450000.00	All Customers	Planned
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
-- GROUP BY
SELECT target_segment, COUNT(campaign_id) AS total_campaigns
FROM Campaigns
GROUP BY target_segment;
```



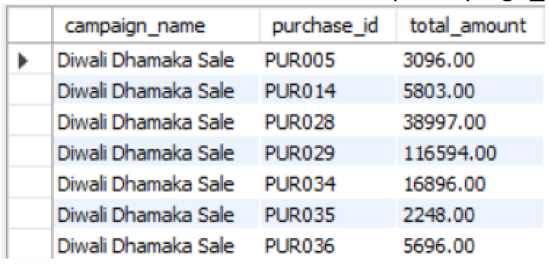
target_segment	total_campaigns
All Customers	4
Premium Customers	1

```
-- GROUP BY with HAVING
SELECT target_segment, COUNT(campaign_id) AS total_campaigns
FROM Campaigns
GROUP BY target_segment
HAVING total_campaigns > 1;
```



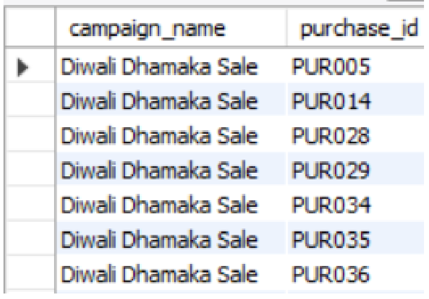
target_segment	total_campaigns
All Customers	4

```
-- INNER JOIN: Campaigns with purchases
SELECT Camp.campaign_name, P.purchase_id, P.total_amount
FROM Campaigns Camp
INNER JOIN Purchases P ON Camp.campaign_id = P.campaign_id;
```



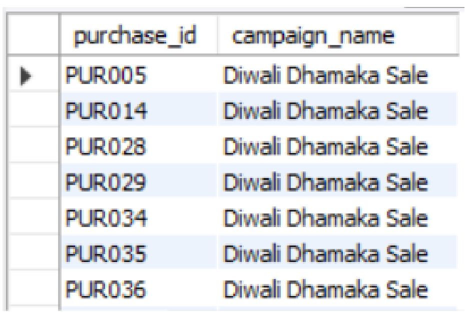
campaign_name	purchase_id	total_amount
Diwali Dhamaka Sale	PUR005	3096.00
Diwali Dhamaka Sale	PUR014	5803.00
Diwali Dhamaka Sale	PUR028	38997.00
Diwali Dhamaka Sale	PUR029	116594.00
Diwali Dhamaka Sale	PUR034	16896.00
Diwali Dhamaka Sale	PUR035	2248.00
Diwali Dhamaka Sale	PUR036	5696.00

```
-- LEFT JOIN: All campaigns, even without purchases
SELECT Camp.campaign_name, P.purchase_id
FROM Campaigns Camp
LEFT JOIN Purchases P ON Camp.campaign_id = P.campaign_id;
```



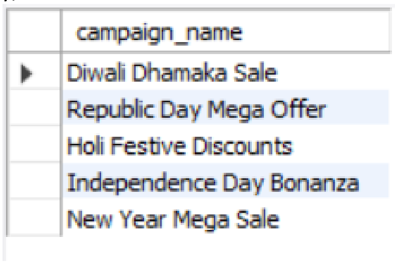
campaign_name	purchase_id
Diwali Dhamaka Sale	PUR005
Diwali Dhamaka Sale	PUR014
Diwali Dhamaka Sale	PUR028
Diwali Dhamaka Sale	PUR029
Diwali Dhamaka Sale	PUR034
Diwali Dhamaka Sale	PUR035
Diwali Dhamaka Sale	PUR036

```
-- RIGHT JOIN: All purchases, ensuring campaign match
SELECT P.purchase_id, Camp.campaign_name
FROM Purchases P
RIGHT JOIN Campaigns Camp ON Camp.campaign_id = P.campaign_id;
```



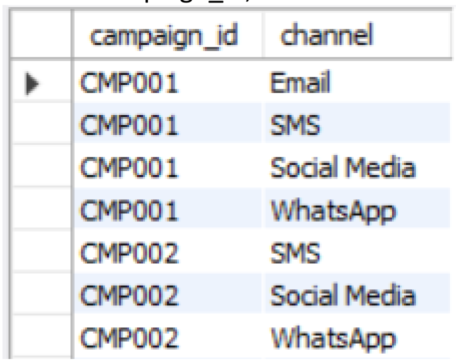
purchase_id	campaign_name
PUR005	Diwali Dhamaka Sale
PUR014	Diwali Dhamaka Sale
PUR028	Diwali Dhamaka Sale
PUR029	Diwali Dhamaka Sale
PUR034	Diwali Dhamaka Sale
PUR035	Diwali Dhamaka Sale
PUR036	Diwali Dhamaka Sale

```
-- Subquery: Campaigns that generated more than ₹50,000
SELECT campaign_name
FROM Campaigns
WHERE campaign_id IN (
   SELECT campaign_id
   FROM Purchases
   GROUP BY campaign_id
   HAVING SUM(total_amount) > 50000
);
```



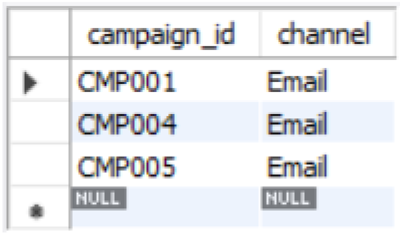
campaign_name
Diwali Dhamaka Sale
Republic Day Mega Offer
Holi Festive Discounts
Independence Day Bonanza
New Year Mega Sale

```
-- For table Campaign_Channel
-- SELECT
SELECT campaign_id, channel FROM Campaign_Channel;
```



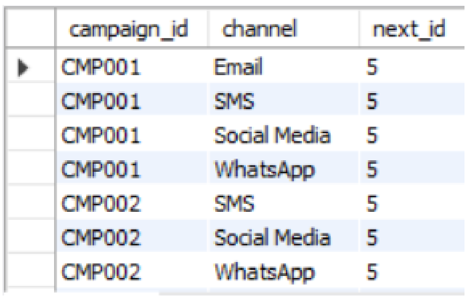
campaign_id	channel
CMP001	Email
CMP001	SMS
CMP001	Social Media
CMP001	WhatsApp
CMP002	SMS
CMP002	Social Media
CMP002	WhatsApp

```
-- WHERE
SELECT * FROM Campaign_Channel WHERE channel = 'Email';
```



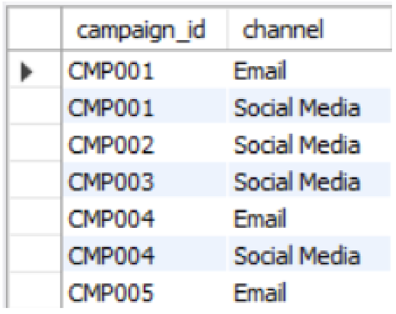
campaign_id	channel
CMP001	Email
CMP004	Email
CMP005	Email
NULL	NULL

```
-- Arithmetic (example id+5)
SELECT campaign_id, channel, campaign_id + 5 AS next_id FROM Campaign_Channel;
```



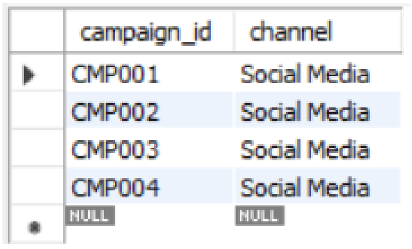
campaign_id	channel	next_id
CMP001	Email	5
CMP001	SMS	5
CMP001	Social Media	5
CMP001	WhatsApp	5
CMP002	SMS	5
CMP002	Social Media	5
CMP002	WhatsApp	5

```
-- Logical operators
SELECT * FROM Campaign_Channel
WHERE channel = 'Email' OR channel = 'Social Media';
```



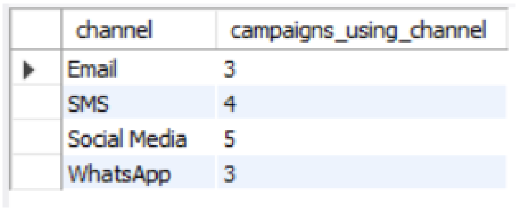
campaign_id	channel
CMP001	Email
CMP001	Social Media
CMP002	Social Media
CMP003	Social Media
CMP004	Email
CMP004	Social Media
CMP005	Email

```
-- LIKE, IN, LIMIT
SELECT * FROM Campaign_Channel
WHERE channel LIKE '%Media%' OR campaign_id IN (1, 3)
LIMIT 4;
```



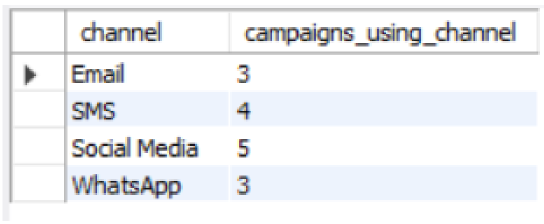
campaign_id	channel
CMP001	Social Media
CMP002	Social Media
CMP003	Social Media
CMP004	Social Media
NULL	NULL

```
-- GROUP BY
SELECT channel, COUNT(campaign_id) AS campaigns_using_channel
FROM Campaign_Channel
GROUP BY channel;
```



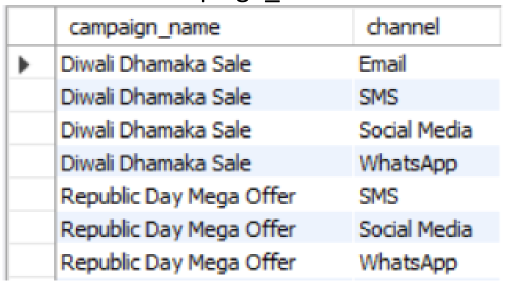
channel	campaigns_using_channel
Email	3
SMS	4
Social Media	5
WhatsApp	3

```
-- GROUP BY with HAVING
SELECT channel, COUNT(campaign_id) AS campaigns_using_channel
FROM Campaign_Channel
GROUP BY channel
HAVING campaigns_using_channel > 2;
```



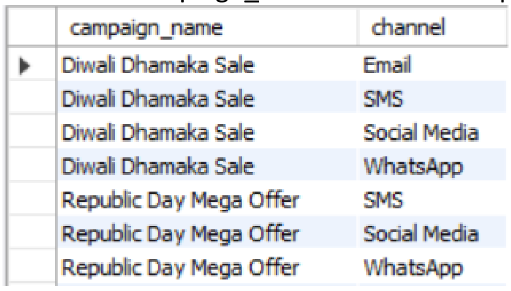
channel	campaigns_using_channel
Email	3
SMS	4
Social Media	5
WhatsApp	3

```
-- INNER JOIN: Campaigns with channels
SELECT Camp.campaign_name, CC.channel
FROM Campaigns Camp
INNER JOIN Campaign_Channel CC ON Camp.campaign_id = CC.campaign_id;
```



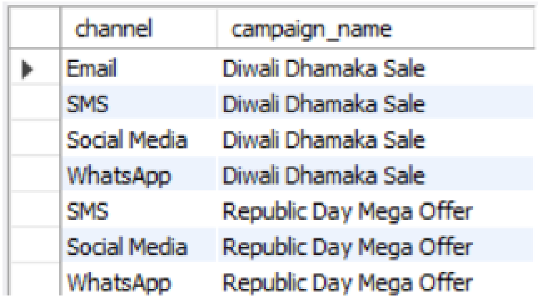
campaign_name	channel
Diwali Dhamaka Sale	Email
Diwali Dhamaka Sale	SMS
Diwali Dhamaka Sale	Social Media
Diwali Dhamaka Sale	WhatsApp
Republic Day Mega Offer	SMS
Republic Day Mega Offer	Social Media
Republic Day Mega Offer	WhatsApp

```
-- LEFT JOIN: All campaigns, even without channels
SELECT Camp.campaign_name, CC.channel
FROM Campaigns Camp
LEFT JOIN Campaign_Channel CC ON Camp.campaign_id = CC.campaign_id;
```



campaign_name	channel
Diwali Dhamaka Sale	Email
Diwali Dhamaka Sale	SMS
Diwali Dhamaka Sale	Social Media
Diwali Dhamaka Sale	WhatsApp
Republic Day Mega Offer	SMS
Republic Day Mega Offer	Social Media
Republic Day Mega Offer	WhatsApp

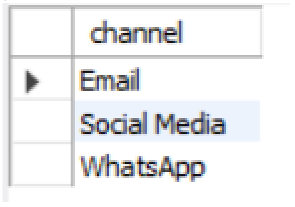
```
-- RIGHT JOIN: All channels, ensuring campaign exists
SELECT CC.channel, Camp.campaign_name
FROM Campaign_Channel CC
RIGHT JOIN Campaigns Camp ON Camp.campaign_id = CC.campaign_id;
```



channel	campaign_name
Email	Diwali Dhamaka Sale
SMS	Diwali Dhamaka Sale
Social Media	Diwali Dhamaka Sale
WhatsApp	Diwali Dhamaka Sale
SMS	Republic Day Mega Offer
Social Media	Republic Day Mega Offer
WhatsApp	Republic Day Mega Offer

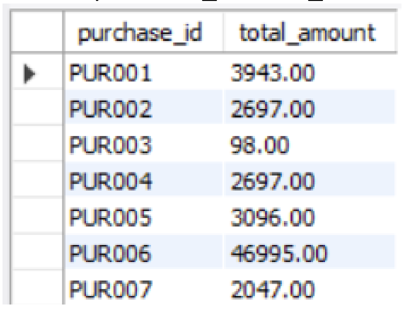
```
-- Subquery: Channels used by more than 2 campaigns
SELECT channel
FROM Campaign_Channel
WHERE campaign_id = (
   SELECT campaign_id
   FROM Purchases
   GROUP BY campaign_id
   ORDER BY SUM(total_amount) DESC
```

```
    LIMIT 1
);
```



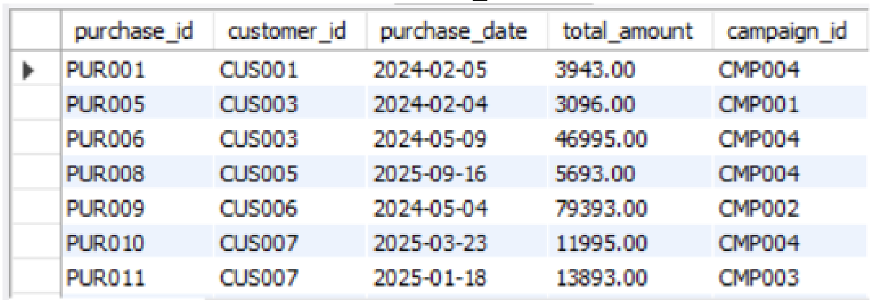
channel
Email
Social Media
WhatsApp

```
-- For table Purchases
-- SELECT
SELECT purchase_id, total_amount FROM Purchases;
```



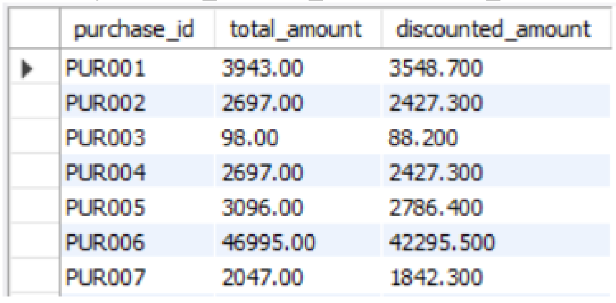
purchase_id	total_amount
PUR001	3943.00
PUR002	2697.00
PUR003	98.00
PUR004	2697.00
PUR005	3096.00
PUR006	46995.00
PUR007	2047.00

```
-- WHERE
SELECT * FROM Purchases WHERE total_amount > 3000;
```



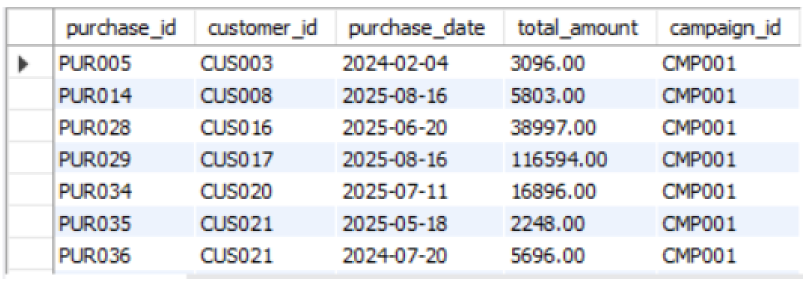
purchase_id	customer_id	purchase_date	total_amount	campaign_id
PUR001	CUS001	2024-02-05	3943.00	CMP004
PUR005	CUS003	2024-02-04	3096.00	CMP001
PUR006	CUS003	2024-05-09	46995.00	CMP004
PUR008	CUS005	2025-09-16	5693.00	CMP004
PUR009	CUS006	2024-05-04	79393.00	CMP002
PUR010	CUS007	2025-03-23	11995.00	CMP004
PUR011	CUS007	2025-01-18	13893.00	CMP003

```
-- Arithmetic
SELECT purchase_id, total_amount, total_amount * 0.9 AS discounted_amount FROM Purchases;
```



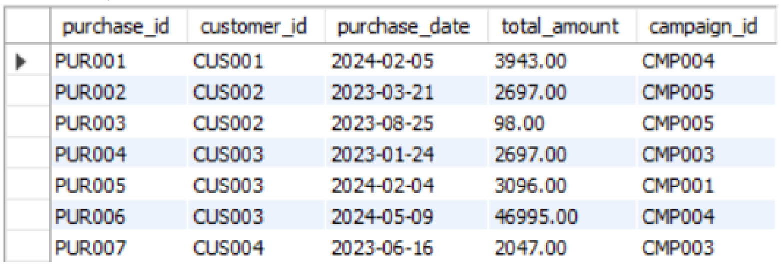
purchase_id	total_amount	discounted_amount
PUR001	3943.00	3548.700
PUR002	2697.00	2427.300
PUR003	98.00	88.200
PUR004	2697.00	2427.300
PUR005	3096.00	2786.400
PUR006	46995.00	42295.500
PUR007	2047.00	1842.300

```
-- Logical operators
SELECT * FROM Purchases
WHERE total_amount > 2000 AND campaign_id = ‘CMP001’;
```



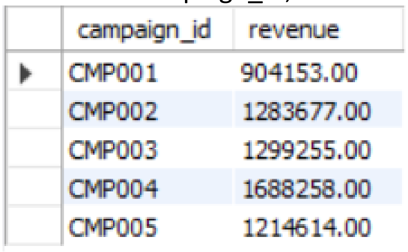
purchase_id	customer_id	purchase_date	total_amount	campaign_id
PUR005	CUS003	2024-02-04	3096.00	CMP001
PUR014	CUS008	2025-08-16	5803.00	CMP001
PUR028	CUS016	2025-06-20	38997.00	CMP001
PUR029	CUS017	2025-08-16	116594.00	CMP001
PUR034	CUS020	2025-07-11	16896.00	CMP001
PUR035	CUS021	2025-05-18	2248.00	CMP001
PUR036	CUS021	2024-07-20	5696.00	CMP001

```
-- LIKE, IN, LIMIT
SELECT * FROM Purchases
WHERE purchase_id LIKE 'PUR%' OR campaign_id IN (1, 2)
LIMIT 10;
```



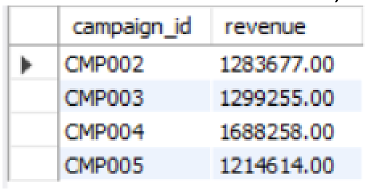
purchase_id	customer_id	purchase_date	total_amount	campaign_id
PUR001	CUS001	2024-02-05	3943.00	CMP004
PUR002	CUS002	2023-03-21	2697.00	CMP005
PUR003	CUS002	2023-08-25	98.00	CMP005
PUR004	CUS003	2023-01-24	2697.00	CMP003
PUR005	CUS003	2024-02-04	3096.00	CMP001
PUR006	CUS003	2024-05-09	46995.00	CMP004
PUR007	CUS004	2023-06-16	2047.00	CMP003

```
-- GROUP BY
SELECT campaign_id, SUM(total_amount) AS revenue
FROM Purchases
GROUP BY campaign_id;
```



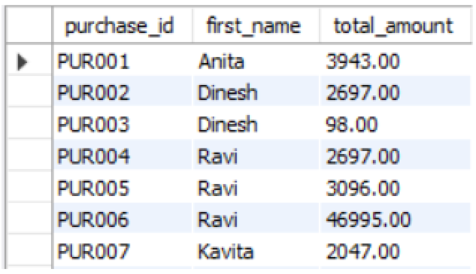
campaign_id	revenue
CMP001	904153.00
CMP002	1283677.00
CMP003	1299255.00
CMP004	1688258.00
CMP005	1214614.00

```
-- GROUP BY with HAVING
SELECT campaign_id, SUM(total_amount) AS revenue
FROM Purchases
GROUP BY campaign_id
HAVING revenue > 1000000;
```



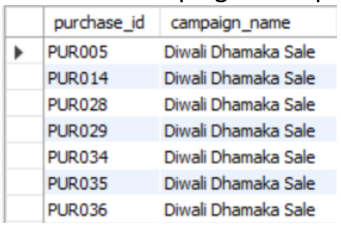
campaign_id	revenue
CMP002	1283677.00
CMP003	1299255.00
CMP004	1688258.00
CMP005	1214614.00

```
-- INNER JOIN: Purchases with customers
SELECT P.purchase_id, C.first_name, P.total_amount
FROM Purchases P
INNER JOIN Customers C ON P.customer_id = C.customer_id;
```



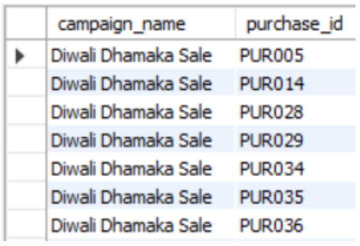
purchase_id	first_name	total_amount
PUR001	Anita	3943.00
PUR002	Dinesh	2697.00
PUR003	Dinesh	98.00
PUR004	Ravi	2697.00
PUR005	Ravi	3096.00
PUR006	Ravi	46995.00
PUR007	Kavita	2047.00

```
-- LEFT JOIN: All purchases, even without campaigns
SELECT P.purchase_id, Camp.campaign_name
FROM Purchases P
LEFT JOIN Campaigns Camp ON P.campaign_id = Camp.campaign_id;
```



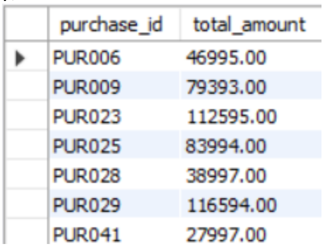
purchase_id	campaign_name
PUR005	Diwali Dhamaka Sale
PUR014	Diwali Dhamaka Sale
PUR028	Diwali Dhamaka Sale
PUR029	Diwali Dhamaka Sale
PUR034	Diwali Dhamaka Sale
PUR035	Diwali Dhamaka Sale
PUR036	Diwali Dhamaka Sale

```
-- RIGHT JOIN: All campaigns, ensuring purchase record
SELECT Camp.campaign_name, P.purchase_id
FROM Campaigns Camp
RIGHT JOIN Purchases P ON P.campaign_id = Camp.campaign_id;
```



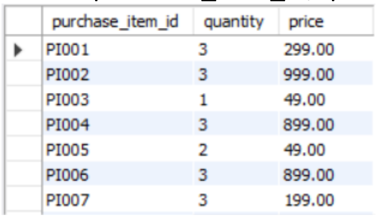
campaign_name	purchase_id
Diwali Dhamaka Sale	PUR005
Diwali Dhamaka Sale	PUR014
Diwali Dhamaka Sale	PUR028
Diwali Dhamaka Sale	PUR029
Diwali Dhamaka Sale	PUR034
Diwali Dhamaka Sale	PUR035
Diwali Dhamaka Sale	PUR036

```
-- Subquery: Purchases above campaign average
SELECT purchase_id, total_amount
FROM Purchases
WHERE total_amount > (
   SELECT AVG(total_amount) FROM Purchases
);
```



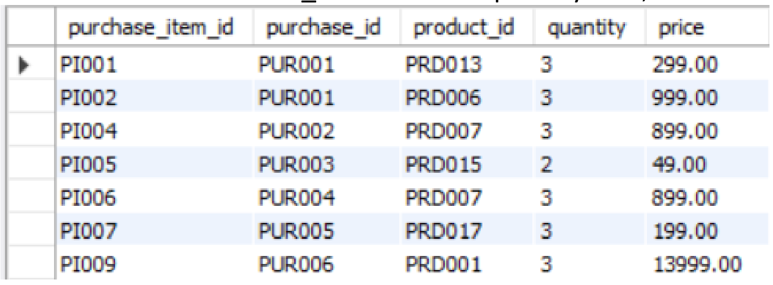
purchase_id	total_amount
PUR006	46995.00
PUR009	79393.00
PUR023	112595.00
PUR025	83994.00
PUR028	38997.00
PUR029	116594.00
PUR041	27997.00

```
-- For table Purchase_Items
-- SELECT
SELECT purchase_item_id, quantity, price FROM Purchase_Items;
```



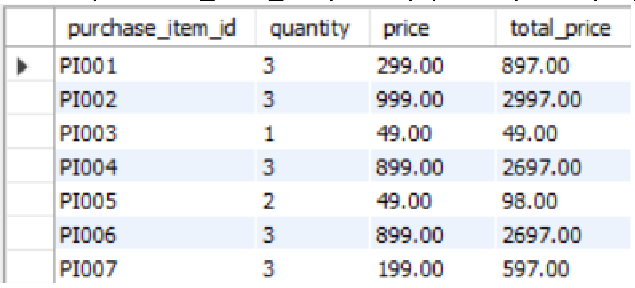
purchase_item_id	quantity	price
PI001	3	299.00
PI002	3	999.00
PI003	1	49.00
PI004	3	899.00
PI005	2	49.00
PI006	3	899.00
PI007	3	199.00

```
-- WHERE
SELECT * FROM Purchase_Items WHERE quantity >= 2;
```



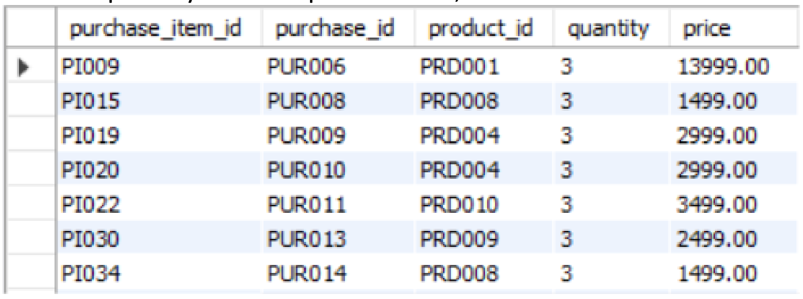
	purchase_item_id	purchase_id	product_id	quantity	price
▶	PI001	PUR001	PRD013	3	299.00
	PI002	PUR001	PRD006	3	999.00
	PI004	PUR002	PRD007	3	899.00
	PI005	PUR003	PRD015	2	49.00
	PI006	PUR004	PRD007	3	899.00
	PI007	PUR005	PRD017	3	199.00
	PI009	PUR006	PRD001	3	13999.00

```
-- Arithmetic
SELECT purchase_item_id, quantity, price, quantity * price AS total_price FROM Purchase_Items;
```



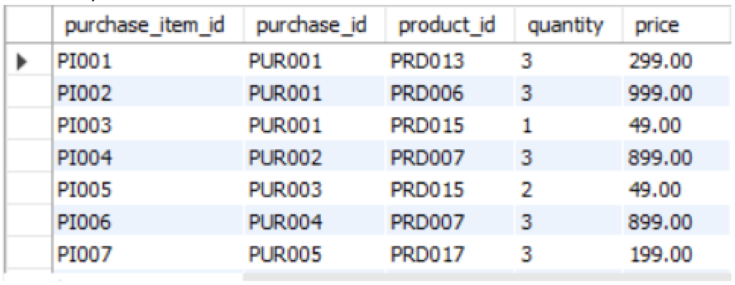
	purchase_item_id	quantity	price	total_price
▶	PI001	3	299.00	897.00
	PI002	3	999.00	2997.00
	PI003	1	49.00	49.00
	PI004	3	899.00	2697.00
	PI005	2	49.00	98.00
	PI006	3	899.00	2697.00
	PI007	3	199.00	597.00

```
-- Logical operators
SELECT * FROM Purchase_Items
WHERE quantity > 2 AND price > 1000;
```



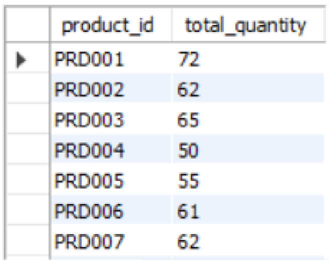
	purchase_item_id	purchase_id	product_id	quantity	price
▶	PI009	PUR006	PRD001	3	13999.00
	PI015	PUR008	PRD008	3	1499.00
	PI019	PUR009	PRD004	3	2999.00
	PI020	PUR010	PRD004	3	2999.00
	PI022	PUR011	PRD010	3	3499.00
	PI030	PUR013	PRD009	3	2499.00
	PI034	PUR014	PRD008	3	1499.00

```
-- LIKE, IN, LIMIT
SELECT * FROM Purchase_Items
WHERE purchase_id LIKE 'PUR%' OR product_id IN (1, 5)
LIMIT 8;
```



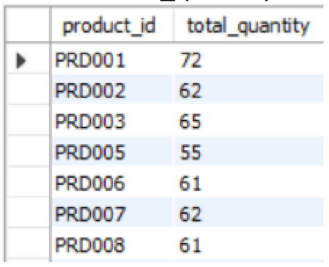
	purchase_item_id	purchase_id	product_id	quantity	price
▶	PI001	PUR001	PRD013	3	299.00
	PI002	PUR001	PRD006	3	999.00
	PI003	PUR001	PRD015	1	49.00
	PI004	PUR002	PRD007	3	899.00
	PI005	PUR003	PRD015	2	49.00
	PI006	PUR004	PRD007	3	899.00
	PI007	PUR005	PRD017	3	199.00

```
-- GROUP BY
SELECT product_id, SUM(quantity) AS total_quantity
FROM Purchase_Items
GROUP BY product_id;
```



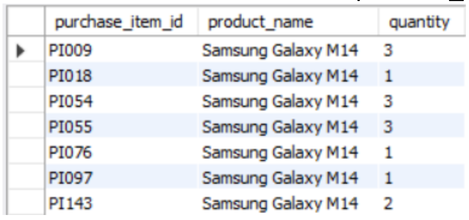
product_id	total_quantity
PRD001	72
PRD002	62
PRD003	65
PRD004	50
PRD005	55
PRD006	61
PRD007	62

```
-- GROUP BY with HAVING
SELECT product_id, SUM(quantity) AS total_quantity
FROM Purchase_Items
GROUP BY product_id
HAVING total_quantity > 50;
```



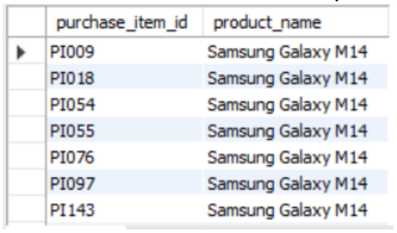
product_id	total_quantity
PRD001	72
PRD002	62
PRD003	65
PRD005	55
PRD006	61
PRD007	62
PRD008	61

```
-- INNER JOIN: Purchase items with products
SELECT PI.purchase_item_id, P.product_name, PI.quantity
FROM Purchase_Items PI
INNER JOIN Products P ON PI.product_id = P.product_id;
```



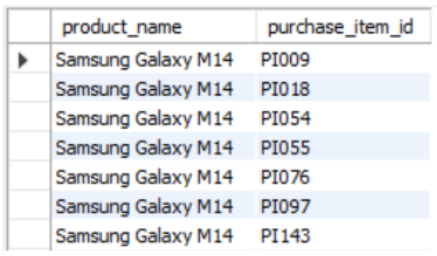
purchase_item_id	product_name	quantity
PI009	Samsung Galaxy M14	3
PI018	Samsung Galaxy M14	1
PI054	Samsung Galaxy M14	3
PI055	Samsung Galaxy M14	3
PI076	Samsung Galaxy M14	1
PI097	Samsung Galaxy M14	1
PI143	Samsung Galaxy M14	2

```
-- LEFT JOIN: All purchase items, even without product match
SELECT PI.purchase_item_id, P.product_name
FROM Purchase_Items PI
LEFT JOIN Products P ON PI.product_id = P.product_id;
```



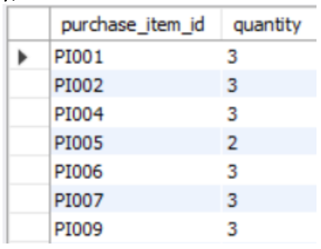
purchase_item_id	product_name
PI009	Samsung Galaxy M14
PI018	Samsung Galaxy M14
PI054	Samsung Galaxy M14
PI055	Samsung Galaxy M14
PI076	Samsung Galaxy M14
PI097	Samsung Galaxy M14
PI143	Samsung Galaxy M14

```
-- RIGHT JOIN: All products, ensuring purchase item record
SELECT P.product_name, PI.purchase_item_id
FROM Products P
RIGHT JOIN Purchase_Items PI ON PI.product_id = P.product_id;
```



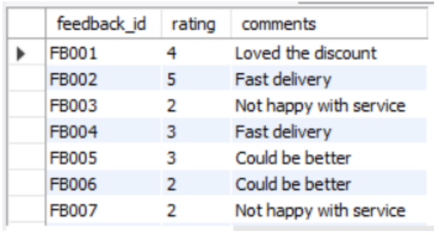
product_name	purchase_item_id
Samsung Galaxy M14	PI009
Samsung Galaxy M14	PI018
Samsung Galaxy M14	PI054
Samsung Galaxy M14	PI055
Samsung Galaxy M14	PI076
Samsung Galaxy M14	PI097
Samsung Galaxy M14	PI143

```
-- Subquery: Items with quantity above average
SELECT purchase_item_id, quantity
FROM Purchase_Items
WHERE quantity > (
   SELECT AVG(quantity) FROM Purchase_Items
);
```



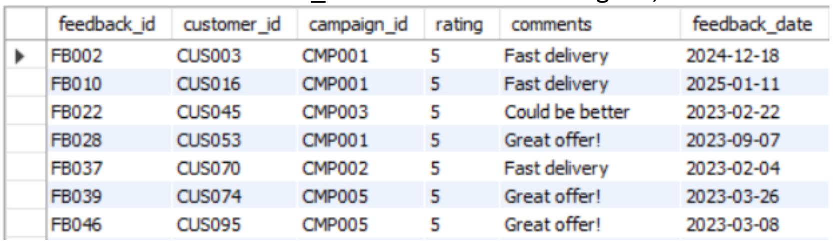
purchase_item_id	quantity
PI001	3
PI002	3
PI004	3
PI005	2
PI006	3
PI007	3
PI009	3

```
-- For table Customer_Feedback
-- SELECT
SELECT feedback_id, rating, comments FROM Customer_Feedback;
```



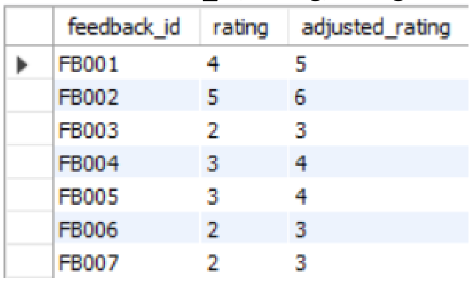
feedback_id	rating	comments
FB001	4	Loved the discount
FB002	5	Fast delivery
FB003	2	Not happy with service
FB004	3	Fast delivery
FB005	3	Could be better
FB006	2	Could be better
FB007	2	Not happy with service

```
-- WHERE
SELECT * FROM Customer_Feedback WHERE rating = 5;
```



feedback_id	customer_id	campaign_id	rating	comments	feedback_date
FB002	CUS003	CMP001	5	Fast delivery	2024-12-18
FB010	CUS016	CMP001	5	Fast delivery	2025-01-11
FB022	CUS045	CMP003	5	Could be better	2023-02-22
FB028	CUS053	CMP001	5	Great offer!	2023-09-07
FB037	CUS070	CMP002	5	Fast delivery	2023-02-04
FB039	CUS074	CMP005	5	Great offer!	2023-03-26
FB046	CUS095	CMP005	5	Great offer!	2023-03-08

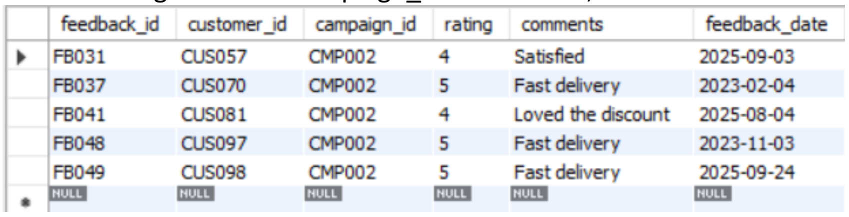
```
-- Arithmetic
SELECT feedback_id, rating, rating + 1 AS adjusted_rating FROM Customer_Feedback;
```



feedback_id	rating	adjusted_rating
FB001	4	5
FB002	5	6
FB003	2	3
FB004	3	4
FB005	3	4
FB006	2	3
FB007	2	3

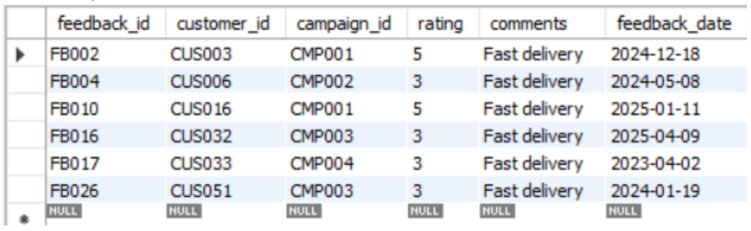
```
-- Logical operators
SELECT * FROM Customer_Feedback
```

```
WHERE rating >= 4 AND campaign_id = ‘CMP002’;
```



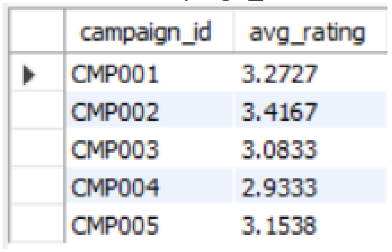
feedback_id	customer_id	campaign_id	rating	comments	feedback_date
FB031	CUS057	CMP002	4	Satisfied	2025-09-03
FB037	CUS070	CMP002	5	Fast delivery	2023-02-04
FB041	CUS081	CMP002	4	Loved the discount	2025-08-04
FB048	CUS097	CMP002	5	Fast delivery	2023-11-03
FB049	CUS098	CMP002	5	Fast delivery	2025-09-24
NULL	NULL	NULL	NULL	NULL	NULL

```
-- LIKE, IN, LIMIT
SELECT * FROM Customer_Feedback
WHERE comments LIKE '%fast%' OR customer_id IN (1, 4)
LIMIT 6;
```



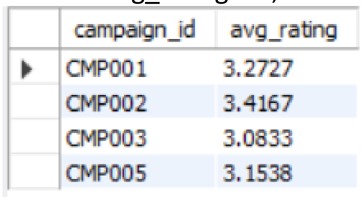
feedback_id	customer_id	campaign_id	rating	comments	feedback_date
FB002	CUS003	CMP001	5	Fast delivery	2024-12-18
FB004	CUS006	CMP002	3	Fast delivery	2024-05-08
FB010	CUS016	CMP001	5	Fast delivery	2025-01-11
FB016	CUS032	CMP003	3	Fast delivery	2025-04-09
FB017	CUS033	CMP004	3	Fast delivery	2023-04-02
FB026	CUS051	CMP003	3	Fast delivery	2024-01-19
NULL	NULL	NULL	NULL	NULL	NULL

```
-- GROUP BY
SELECT campaign_id, AVG(rating) AS avg_rating
FROM Customer_Feedback
GROUP BY campaign_id;
```



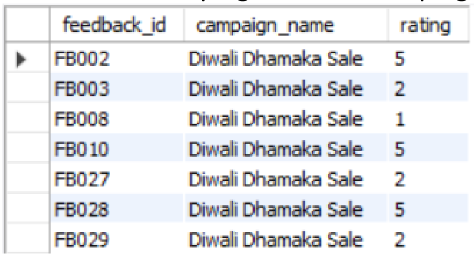
campaign_id	avg_rating
CMP001	3.2727
CMP002	3.4167
CMP003	3.0833
CMP004	2.9333
CMP005	3.1538

```
-- GROUP BY with HAVING
SELECT campaign_id, AVG(rating) AS avg_rating
FROM Customer_Feedback
GROUP BY campaign_id
HAVING avg_rating > 3;
```



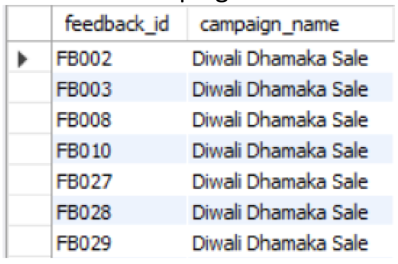
campaign_id	avg_rating
CMP001	3.2727
CMP002	3.4167
CMP003	3.0833
CMP005	3.1538

```
-- INNER JOIN: Feedback with campaigns
SELECT F.feedback_id, C.campaign_name, F.rating
FROM Customer_Feedback F
INNER JOIN Campaigns C ON F.campaign_id = C.campaign_id;
```



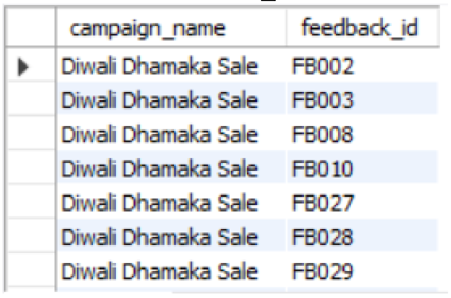
feedback_id	campaign_name	rating
FB002	Diwali Dhamaka Sale	5
FB003	Diwali Dhamaka Sale	2
FB008	Diwali Dhamaka Sale	1
FB010	Diwali Dhamaka Sale	5
FB027	Diwali Dhamaka Sale	2
FB028	Diwali Dhamaka Sale	5
FB029	Diwali Dhamaka Sale	2

```
-- LEFT JOIN: All feedback, even without campaign match
SELECT F.feedback_id, C.campaign_name
FROM Customer_Feedback F
LEFT JOIN Campaigns C ON F.campaign_id = C.campaign_id;
```



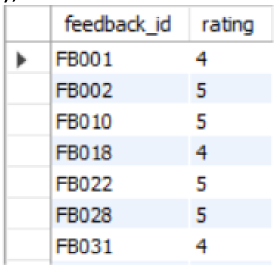
feedback_id	campaign_name
FB002	Diwali Dhamaka Sale
FB003	Diwali Dhamaka Sale
FB008	Diwali Dhamaka Sale
FB010	Diwali Dhamaka Sale
FB027	Diwali Dhamaka Sale
FB028	Diwali Dhamaka Sale
FB029	Diwali Dhamaka Sale

```
-- RIGHT JOIN: All campaigns, ensuring feedback record
SELECT C.campaign_name, F.feedback_id
FROM Campaigns C
RIGHT JOIN Customer_Feedback F ON F.campaign_id = C.campaign_id;
```



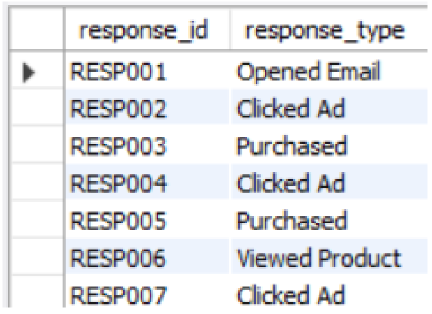
campaign_name	feedback_id
Diwali Dhamaka Sale	FB002
Diwali Dhamaka Sale	FB003
Diwali Dhamaka Sale	FB008
Diwali Dhamaka Sale	FB010
Diwali Dhamaka Sale	FB027
Diwali Dhamaka Sale	FB028
Diwali Dhamaka Sale	FB029

```
-- Subquery: Feedback ratings above overall average
SELECT feedback_id, rating
FROM Customer_Feedback
WHERE rating > (
  SELECT AVG(rating) FROM Customer_Feedback
);
```



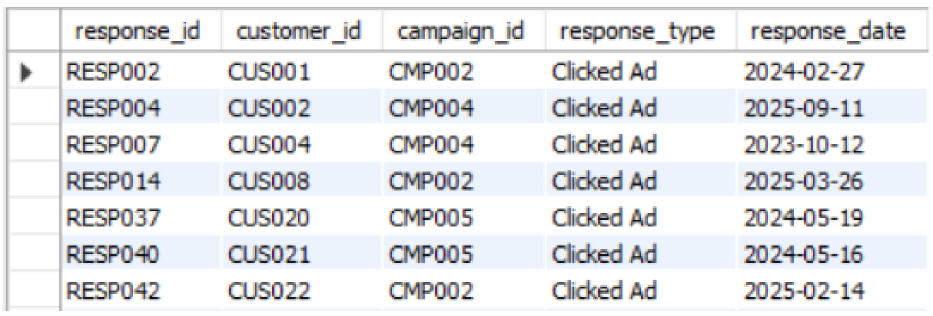
feedback_id	rating
FB001	4
FB002	5
FB010	5
FB018	4
FB022	5
FB028	5
FB031	4

```
-- For table Campaign_Responses
-- SELECT
SELECT response_id, response_type FROM Campaign_Responses;
```



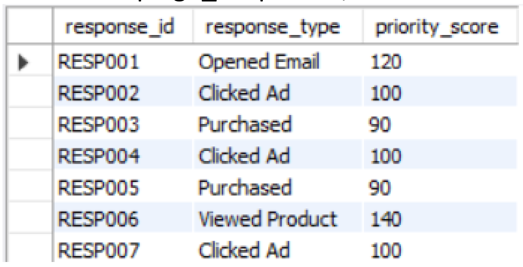
response_id	response_type
RESP001	Opened Email
RESP002	Clicked Ad
RESP003	Purchased
RESP004	Clicked Ad
RESP005	Purchased
RESP006	Viewed Product
RESP007	Clicked Ad

```
-- WHERE
SELECT * FROM Campaign_Responses WHERE response_type = 'Clicked Ad';
```



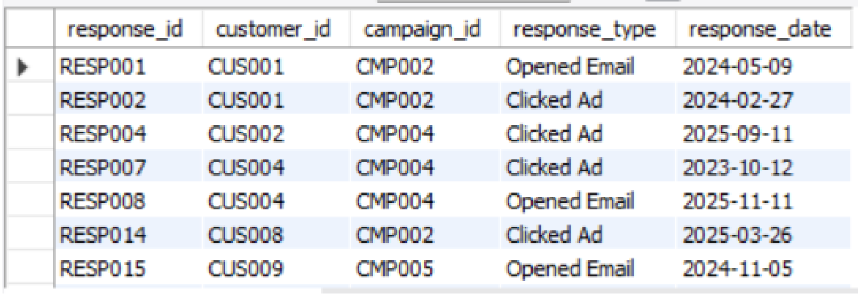
response_id	customer_id	campaign_id	response_type	response_date
RESP002	CUS001	CMP002	Clicked Ad	2024-02-27
RESP004	CUS002	CMP004	Clicked Ad	2025-09-11
RESP007	CUS004	CMP004	Clicked Ad	2023-10-12
RESP014	CUS008	CMP002	Clicked Ad	2025-03-26
RESP037	CUS020	CMP005	Clicked Ad	2024-05-19
RESP040	CUS021	CMP005	Clicked Ad	2024-05-16
RESP042	CUS022	CMP002	Clicked Ad	2025-02-14

```
-- Arithmetic (id+100)
SELECT
    response_id,
    response_type,
    (LENGTH(response_type) * 10) + campaign_id AS priority_score
FROM Campaign_Responses;
```



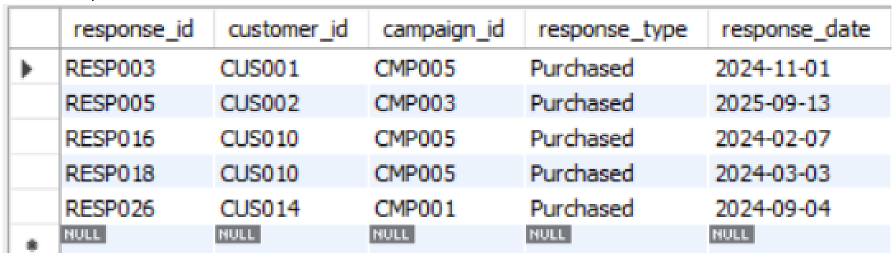
response_id	response_type	priority_score
RESP001	Opened Email	120
RESP002	Clicked Ad	100
RESP003	Purchased	90
RESP004	Clicked Ad	100
RESP005	Purchased	90
RESP006	Viewed Product	140
RESP007	Clicked Ad	100

```
-- Logical operators
SELECT * FROM Campaign_Responses
WHERE response_type = 'Clicked Ad' OR response_type = 'Opened Email';
```



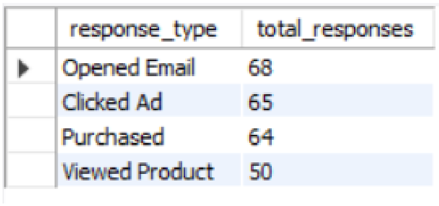
response_id	customer_id	campaign_id	response_type	response_date
RESP001	CUS001	CMP002	Opened Email	2024-05-09
RESP002	CUS001	CMP002	Clicked Ad	2024-02-27
RESP004	CUS002	CMP004	Clicked Ad	2025-09-11
RESP007	CUS004	CMP004	Clicked Ad	2023-10-12
RESP008	CUS004	CMP004	Opened Email	2025-11-11
RESP014	CUS008	CMP002	Clicked Ad	2025-03-26
RESP015	CUS009	CMP005	Opened Email	2024-11-05

```
-- LIKE, IN, LIMIT
SELECT * FROM Campaign_Responses
WHERE response_type LIKE '%ed' OR campaign_id IN (1, 3)
LIMIT 5;
```



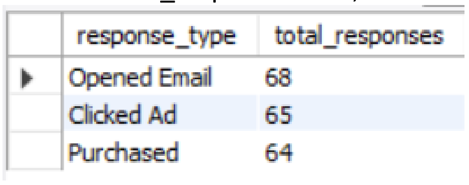
response_id	customer_id	campaign_id	response_type	response_date
RESP003	CUS001	CMP005	Purchased	2024-11-01
RESP005	CUS002	CMP003	Purchased	2025-09-13
RESP016	CUS010	CMP005	Purchased	2024-02-07
RESP018	CUS010	CMP005	Purchased	2024-03-03
RESP026	CUS014	CMP001	Purchased	2024-09-04
NULL	NULL	NULL	NULL	NULL

```
-- GROUP BY
SELECT response_type, COUNT(response_id) AS total_responses
FROM Campaign_Responses
GROUP BY response_type;
```



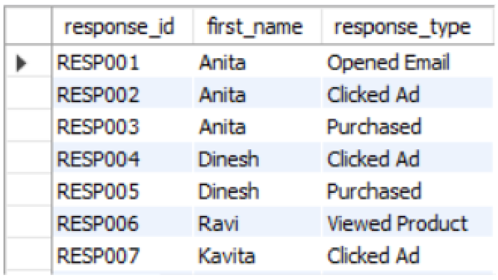
response_type	total_responses
Opened Email	68
Clicked Ad	65
Purchased	64
Viewed Product	50

```
-- GROUP BY with HAVING
SELECT response_type, COUNT(response_id) AS total_responses
FROM Campaign_Responses
GROUP BY response_type
HAVING total_responses > 50;
```



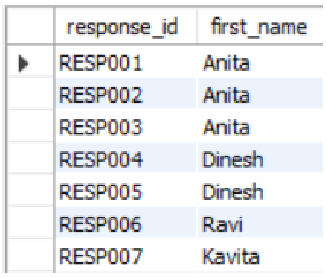
response_type	total_responses
Opened Email	68
Clicked Ad	65
Purchased	64

```
-- INNER JOIN: Campaign responses with customers
SELECT CR.response_id, C.first_name, CR.response_type
FROM Campaign_Responses CR
INNER JOIN Customers C ON CR.customer_id = C.customer_id;
```



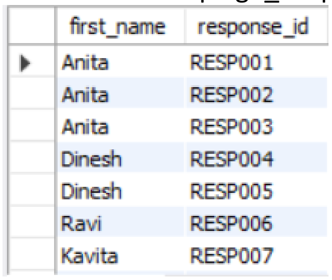
response_id	first_name	response_type
RESP001	Anita	Opened Email
RESP002	Anita	Clicked Ad
RESP003	Anita	Purchased
RESP004	Dinesh	Clicked Ad
RESP005	Dinesh	Purchased
RESP006	Ravi	Viewed Product
RESP007	Kavita	Clicked Ad

```
-- LEFT JOIN: All responses, even without customer match
SELECT CR.response_id, C.first_name
FROM Campaign_Responses CR
LEFT JOIN Customers C ON CR.customer_id = C.customer_id;
```



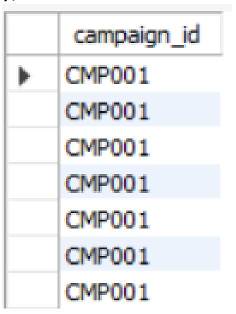
response_id	first_name
RESP001	Anita
RESP002	Anita
RESP003	Anita
RESP004	Dinesh
RESP005	Dinesh
RESP006	Ravi
RESP007	Kavita

```
-- RIGHT JOIN: All customers, ensuring response record
SELECT C.first_name, CR.response_id
FROM Customers C
RIGHT JOIN Campaign_Responses CR ON CR.customer_id = C.customer_id;
```



first_name	response_id
Anita	RESP001
Anita	RESP002
Anita	RESP003
Dinesh	RESP004
Dinesh	RESP005
Ravi	RESP006
Kavita	RESP007

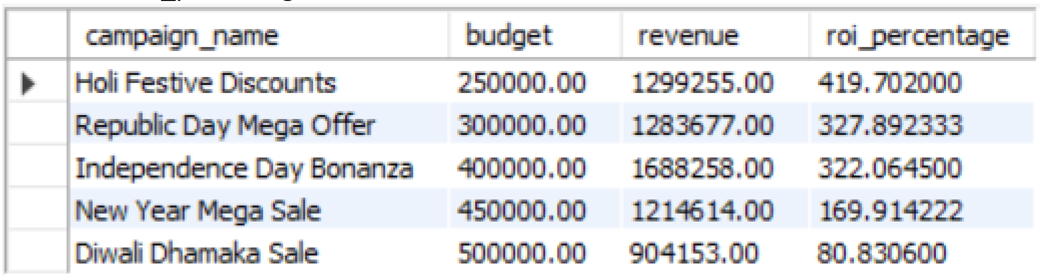
```
-- Subquery: Campaigns with 'Clicked Ad' responses
SELECT campaign_id
FROM Campaign_Responses
WHERE campaign_id IN (
   SELECT campaign_id
   FROM Campaign_Responses
   WHERE response_type = 'Clicked Ad'
);
```



campaign_id
CMP001
CMP001
CMP001
CMP001
CMP001
CMP001
CMP001

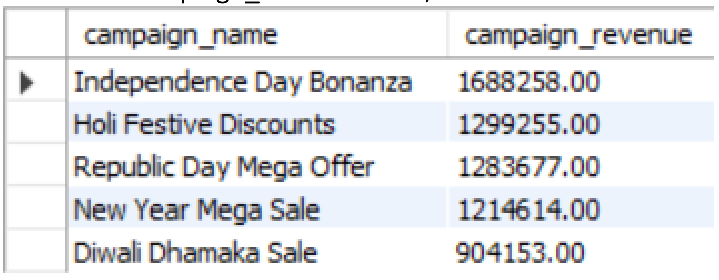
```
-- Advance Queries for business insights

-- ROI calculation per campaign
SELECT
   C.campaign_name,
   C.budget,
   SUM(P.total_amount) AS revenue,
   (SUM(P.total_amount) - C.budget) / C.budget * 100 AS roi_percentage
FROM Campaigns C
LEFT JOIN Purchases P ON C.campaign_id = P.campaign_id
GROUP BY C.campaign_id, C.campaign_name, C.budget
ORDER BY roi_percentage DESC;
```



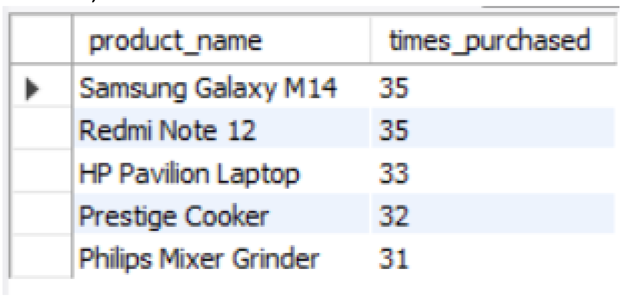
campaign_name	budget	revenue	roi_percentage
Holi Festive Discounts	250000.00	1299255.00	419.702000
Republic Day Mega Offer	300000.00	1283677.00	327.892333
Independence Day Bonanza	400000.00	1688258.00	322.064500
New Year Mega Sale	450000.00	1214614.00	169.914222
Diwali Dhamaka Sale	500000.00	904153.00	80.830600

```
-- Calculate total revenue by campaign
SELECT C.campaign_name, SUM(P.total_amount) AS campaign_revenue
FROM Purchases P
JOIN Campaigns C ON P.campaign_id = C.campaign_id
GROUP BY C.campaign_name
ORDER BY campaign_revenue DESC;
```



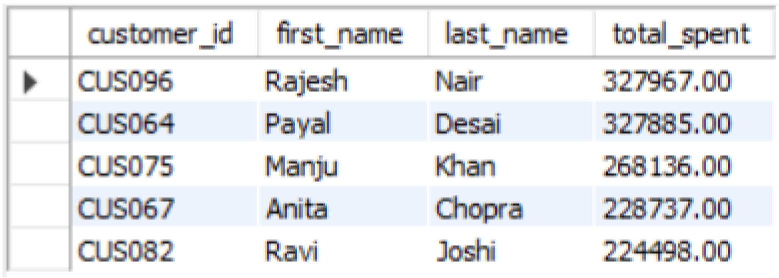
campaign_name	campaign_revenue
Independence Day Bonanza	1688258.00
Holi Festive Discounts	1299255.00
Republic Day Mega Offer	1283677.00
New Year Mega Sale	1214614.00
Diwali Dhamaka Sale	904153.00

```
-- Top 5 most purchased products
SELECT P.product_name, COUNT(PI.product_id) AS times_purchased
FROM Purchase_Items PI
JOIN Products P ON PI.product_id = P.product_id
GROUP BY P.product_name
ORDER BY times_purchased DESC
LIMIT 5;
```



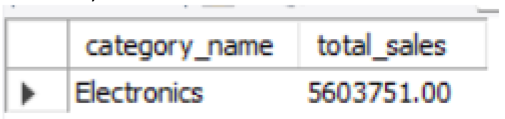
product_name	times_purchased
Samsung Galaxy M14	35
Redmi Note 12	35
HP Pavilion Laptop	33
Prestige Cooker	32
Philips Mixer Grinder	31

```
-- Top 5 most valuable customers
SELECT C.customer_id, C.first_name, C.last_name, SUM(P.total_amount) AS total_spent
FROM Customers C
JOIN Purchases P ON C.customer_id = P.customer_id
GROUP BY C.customer_id, C.first_name, C.last_name
ORDER BY total_spent DESC
LIMIT 5;
```



customer_id	first_name	last_name	total_spent
CUS096	Rajesh	Nair	327967.00
CUS064	Payal	Desai	327885.00
CUS075	Manju	Khan	268136.00
CUS067	Anita	Chopra	228737.00
CUS082	Ravi	Joshi	224498.00

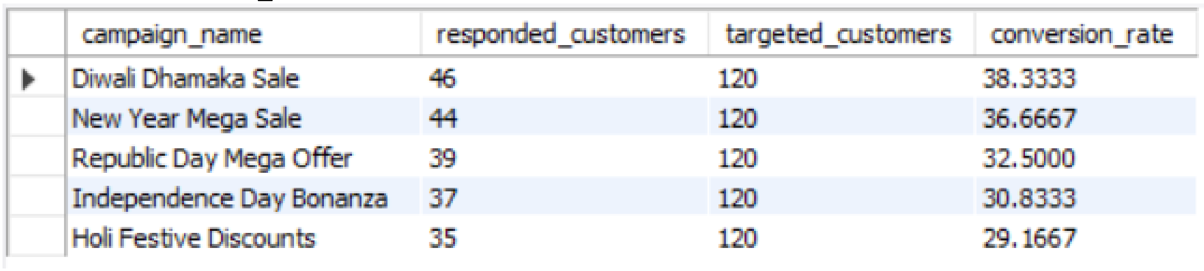
```
-- Most profitable product category
SELECT Cat.category_name, SUM(PI.quantity * PI.price) AS total_sales
FROM Purchase_Items PI
JOIN Products P ON PI.product_id = P.product_id
JOIN Categories Cat ON P.category_id = Cat.category_id
GROUP BY Cat.category_name
ORDER BY total_sales DESC
LIMIT 1;
```



category_name	total_sales
Electronics	5603751.00

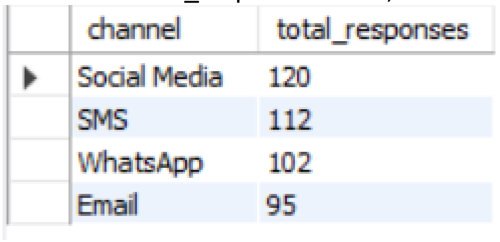
```
-- Conversion rate per campaign
SELECT
   C.campaign_name,
   COUNT(DISTINCT CR.customer_id) AS responded_customers,
   (SELECT COUNT(*) FROM Customers) AS targeted_customers,
   (COUNT(DISTINCT CR.customer_id) / (SELECT COUNT(*) FROM Customers)) * 100 AS conversion_rate
FROM Campaigns C
LEFT JOIN Campaign_Responses CR
     ON C.campaign_id = CR.campaign_id
GROUP BY C.campaign_name
```

```
ORDER BY conversion_rate DESC;
```



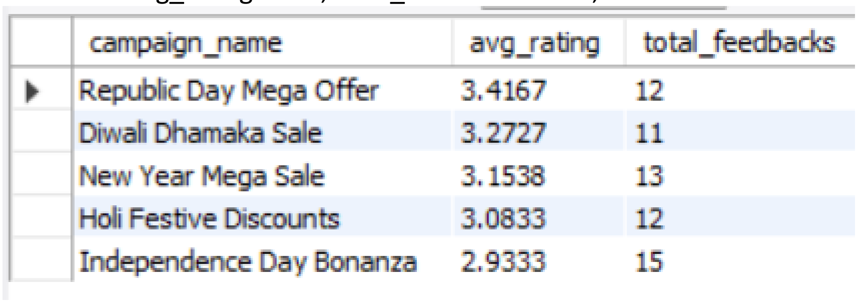
	campaign_name	responded_customers	targeted_customers	conversion_rate
▶	Diwali Dhamaka Sale	46	120	38.3333
	New Year Mega Sale	44	120	36.6667
	Republic Day Mega Offer	39	120	32.5000
	Independence Day Bonanza	37	120	30.8333
	Holi Festive Discounts	35	120	29.1667

```
-- Most responsive marketing channel
SELECT CC.channel, COUNT(DISTINCT CR.customer_id) AS total_responses
FROM Campaign_Channel CC
JOIN Campaign_Responses CR ON CC.campaign_id = CR.campaign_id
GROUP BY CC.channel
ORDER BY total_responses DESC;
```



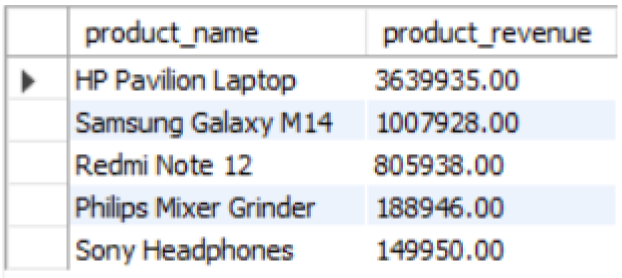
	channel	total_responses
▶	Social Media	120
	SMS	112
	WhatsApp	102
	Email	95

```
-- Feedback-based campaign ranking
SELECT C.campaign_name,
    AVG(F.rating) AS avg_rating,
    COUNT(F.feedback_id) AS total_feedbacks
FROM Campaigns C
LEFT JOIN Customer_Feedback F ON C.campaign_id = F.campaign_id
GROUP BY C.campaign_name
ORDER BY avg_rating DESC, total_feedbacks DESC;
```



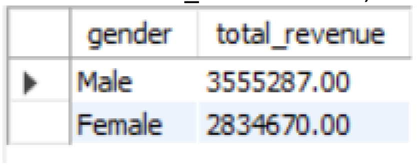
	campaign_name	avg_rating	total_feedbacks
▶	Republic Day Mega Offer	3.4167	12
	Diwali Dhamaka Sale	3.2727	11
	New Year Mega Sale	3.1538	13
	Holi Festive Discounts	3.0833	12
	Independence Day Bonanza	2.9333	15

```
-- Products that drive the most revenue
SELECT P.product_name, SUM(PI.quantity * PI.price) AS product_revenue
FROM Purchase_Items PI
JOIN Products P ON PI.product_id = P.product_id
GROUP BY P.product_name
ORDER BY product_revenue DESC
LIMIT 5;
```



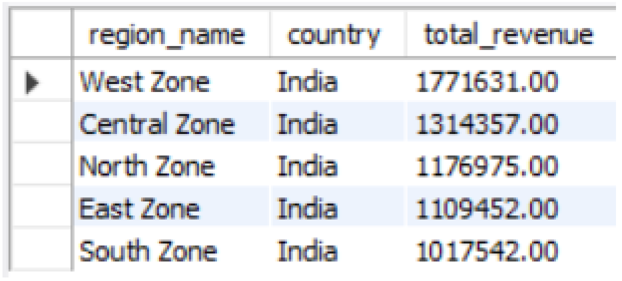
product_name	product_revenue
HP Pavilion Laptop	3639935.00
Samsung Galaxy M14	1007928.00
Redmi Note 12	805938.00
Philips Mixer Grinder	188946.00
Sony Headphones	149950.00

```
-- Gender-wise revenue contribution
SELECT C.gender, SUM(P.total_amount) AS total_revenue
FROM Customers C
JOIN Purchases P ON C.customer_id = P.customer_id
GROUP BY C.gender
ORDER BY total_revenue DESC;
```



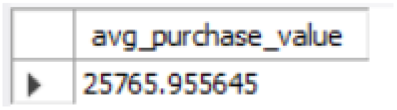
gender	total_revenue
Male	3555287.00
Female	2834670.00

```
-- Regions generating the highest revenue
SELECT R.region_name, R.country, SUM(P.total_amount) AS total_revenue
FROM Regions R
JOIN Customers C ON R.region_id = C.region_id
JOIN Purchases P ON C.customer_id = P.customer_id
GROUP BY R.region_name, R.country
ORDER BY total_revenue DESC;
```



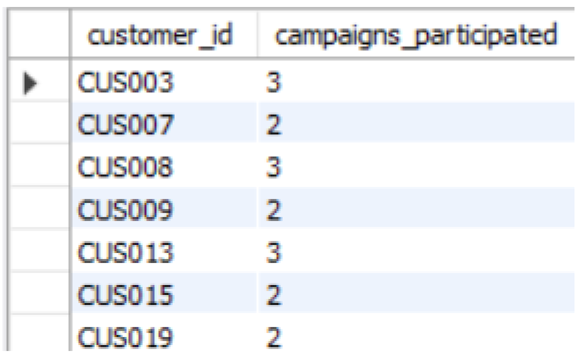
region_name	country	total_revenue
West Zone	India	1771631.00
Central Zone	India	1314357.00
North Zone	India	1176975.00
East Zone	India	1109452.00
South Zone	India	1017542.00

```
-- Average purchase size (per transaction)
SELECT AVG(total_amount) AS avg_purchase_value
FROM Purchases;
```



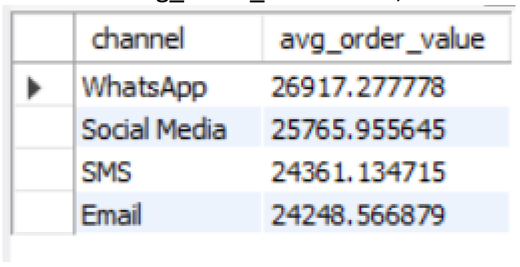
avg_purchase_value
25765.955645

```
-- Customers who purchased from multiple campaigns
SELECT customer_id, COUNT(DISTINCT campaign_id) AS campaigns_participated
FROM Purchases
GROUP BY customer_id
HAVING campaigns_participated > 1;
```



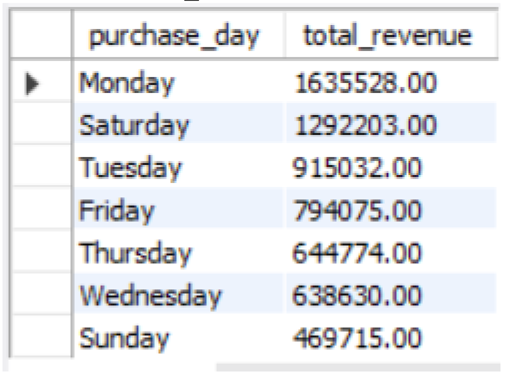
customer_id	campaigns_participated
CUS003	3
CUS007	2
CUS008	3
CUS009	2
CUS013	3
CUS015	2
CUS019	2

```
-- Average order value by marketing channel
SELECT CC.channel, AVG(P.total_amount) AS avg_order_value
FROM Campaign_Channel CC
JOIN Campaigns C ON CC.campaign_id = C.campaign_id
JOIN Purchases P ON C.campaign_id = P.campaign_id
GROUP BY CC.channel
ORDER BY avg_order_value DESC;
```



channel	avg_order_value
WhatsApp	26917.277778
Social Media	25765.955645
SMS	24361.134715
Email	24248.566879

```
-- Best day of the week for purchases
SELECT DAYNAME(purchase_date) AS purchase_day, SUM(total_amount) AS total_revenue
FROM Purchases
GROUP BY purchase_day
ORDER BY total_revenue DESC;
```



purchase_day	total_revenue
Monday	1635528.00
Saturday	1292203.00
Tuesday	915032.00
Friday	794075.00
Thursday	644774.00
Wednesday	638630.00
Sunday	469715.00

7. MongoDB Implementation

7.1. MongoDB Schema Design

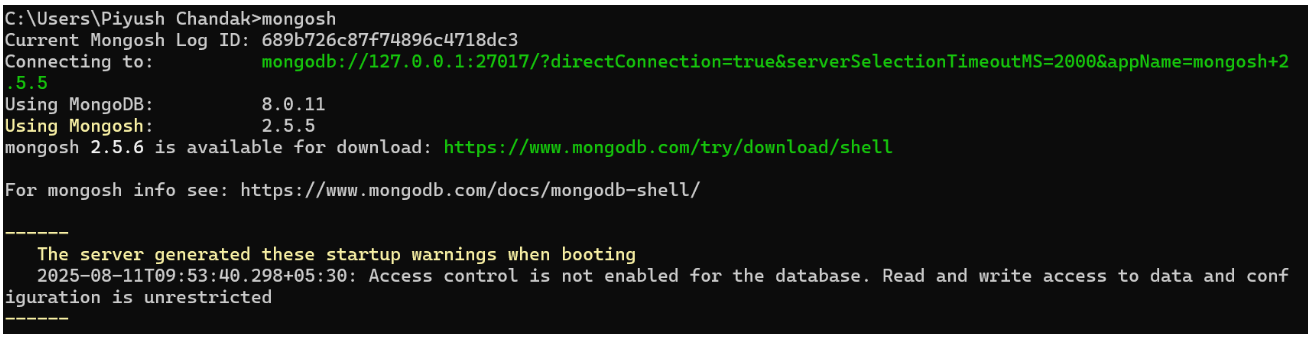
To leverage the strengths of a NoSQL database, the relational data was denormalized and consolidated into a single collection named customers. This design embeds related information (like purchases, responses, and feedback) directly within each customer document, eliminating the need for joins and improving query performance.

Structure of a Customer Document

```
{
 "customer_id": "CUS001",
 "first_name": "Anita",
 "last_name": "Ghosh",
 "email": "anita.ghosh95@gmail.com",
 "phone": "9902010912",
 "gender": "Female",
 "age": 38,
 "registration_date": "ISODate",
 "region": {
  "region_name": "North Zone",
  "country": "India"
 },
 "purchases": [
  {
   "purchase_id": "PUR001",
   "purchase_date": "ISODate",
   "total_amount": 3943,
   "campaign_id": "CMP004",
   "items": [
    {
     "product_id": "PRD013",
     "product_name": "Chetan Bhagat Novel",
     "quantity": 3,
     "price": 299
    }
   ]
  }
 ],
 "campaign_responses": [
  {
   "response_id": "RESP001",
   "campaign_id": "CMP002",
   "response_type": "Opened Email",
   "response_date": "ISODate"
  }
 ],
  "feedback": [
    {
      "feedback_id": "FB001",
      "campaign_id": "CMP005",
      "rating": 4,
      "comments": "Loved the discount",
      "feedback_date": "ISODate"
    }
  ]
}
```

Execute-

- mongosh

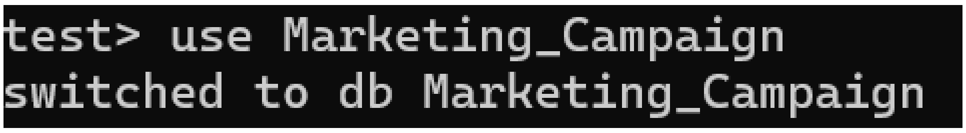


```
C:\Users\Piyush Chandak>mongosh
Current Mongosh Log ID: 689b726c87f74896c4718dc3
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.5
Using MongoDB:          8.0.11
Using Mongosh:          2.5.5
mongosh 2.5.6 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-08-11T09:53:40.298+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------
```

-use Marketing_Campaign



```
test> use Marketing_Campaign
switched to db Marketing_Campaign
```

7.2. MongoDB Operations

The following CRUD (Create, Read, Update, Delete) operations were performed in mongosh.

A. Insertion (insertOne)

Objective: To add a new customer named 'Rohan Das'.

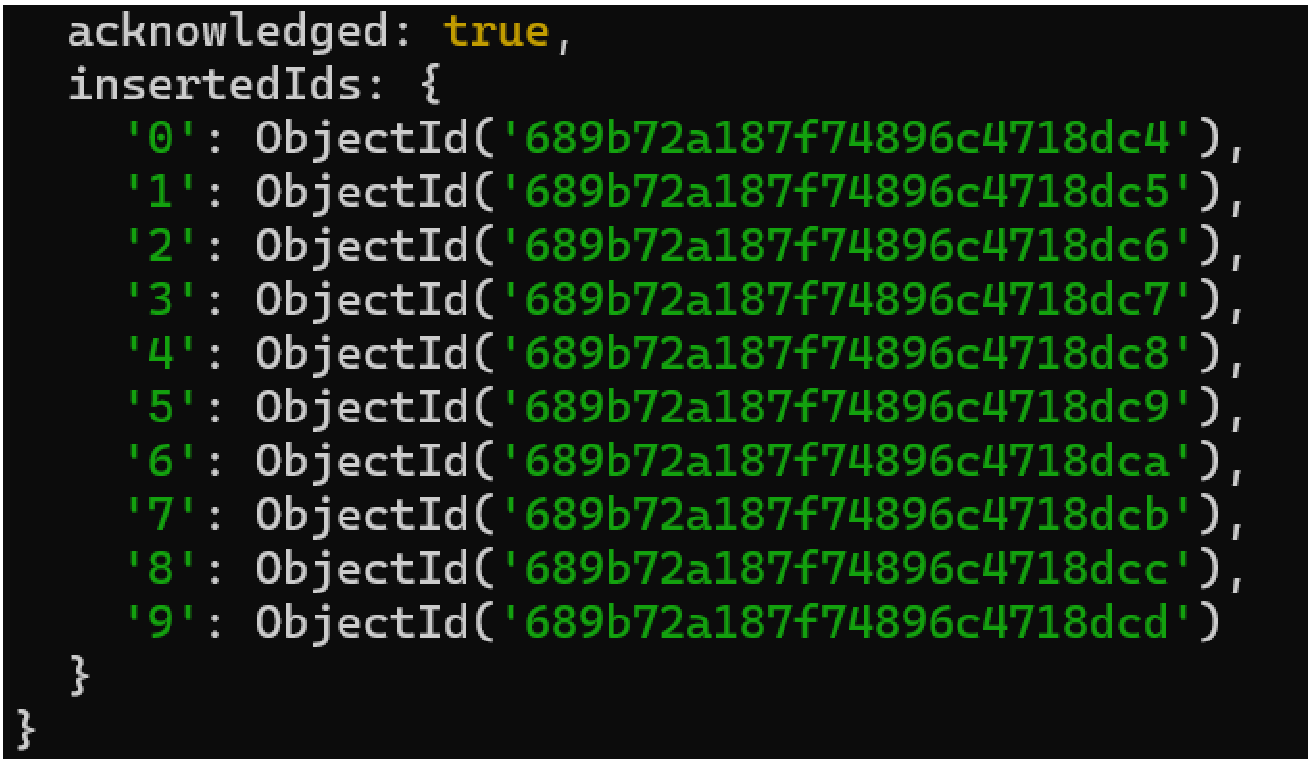
Command:

```
db.customers.insertOne({
  "customer_id": "CUS116", "first_name": "Rohan", "last_name": "Das",
  "email": "rohan.das@example.com", "gender": "Male", "age": 25,
  "region": { "region_name": "East Zone", "country": "India" },
  "purchases": [], "campaign_responses": [], "feedback": []
})
```

Output:

{ "acknowledged": true, "insertedId": ObjectId("...") }

For Insert_Many-
Output:



```
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('689b72a187f74896c4718dc4'),
    '1': ObjectId('689b72a187f74896c4718dc5'),
    '2': ObjectId('689b72a187f74896c4718dc6'),
    '3': ObjectId('689b72a187f74896c4718dc7'),
    '4': ObjectId('689b72a187f74896c4718dc8'),
    '5': ObjectId('689b72a187f74896c4718dc9'),
    '6': ObjectId('689b72a187f74896c4718dca'),
    '7': ObjectId('689b72a187f74896c4718dcb'),
    '8': ObjectId('689b72a187f74896c4718dcc'),
    '9': ObjectId('689b72a187f74896c4718dcd')
  }
}
```

B. Find Operations (find)

1. Find by Top-Level Field (All female customers):
   `db.customers.find({ gender: "Female" })`
2. Find by Embedded Document Field (Customers in "South Zone"):
   `db.customers.find({ "region.region_name": "South Zone" })`
3. Find with Comparison Operator (Customers older than 50):
   `db.customers.find({ age: { $gt: 50 } })`
4. Find with Multiple Conditions (Male customers from "Central Zone"):
   `db.customers.find({ gender: "Male", "region.region_name": "Central Zone" })`
5. Find by Matching an Element in an Array (Customers who gave a 1-star rating):
   `db.customers.find({ feedback: { $elemMatch: { rating: 1 } } })`

C. Update Operation (updateOne)

Objective: To update the email for customer 'Anita Ghosh' (CUS001).

Command:

```
db.customers.updateOne(
  { customer_id: "CUS001" },
  { $set: { email: "anita.g.new@example.com" } }
)
```

Output:

```
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
```

D. Delete Operation (deleteOne)

Objective: To remove the record for 'Rohan Das' (CUS116).

Command:

```
db.customers.deleteOne({ customer_id: "CUS116" })
```

Output:

```
{ "acknowledged": true, "deletedCount": 1 }
```

8. Conclusion

This project successfully demonstrated the design and implementation of a database for marketing campaign analysis using both MySQL and MongoDB.

In the **MySQL** section, a normalized relational schema was created, ensuring data integrity through primary and foreign keys. A comprehensive set of SQL queries was executed to perform tasks ranging from basic data retrieval to complex business analysis, such as calculating campaign ROI and identifying top customers.

In the **MongoDB** section, the data was transformed into a denormalized, document-based model. This schema design simplifies queries and enhances performance by embedding related data within a single customer document, thereby avoiding costly joins. Basic and advanced find operations, along with insert, update, and delete commands, showcased the flexibility and power of NoSQL for handling complex, nested data structures.

By working with both database paradigms, this project provided valuable insights into their respective strengths and use cases, highlighting how relational and NoSQL databases can be effectively utilized to solve real-world data management challenges.