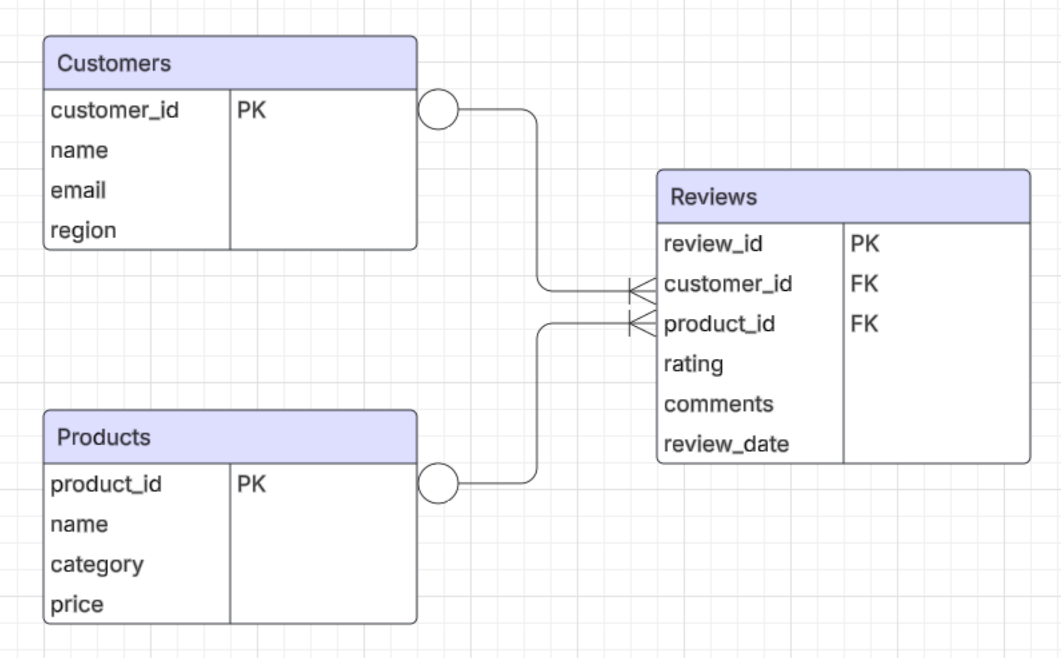
Bushra Hoteit

Part 1: Database Design

1. **Design an Entity-Relationship (ER) diagram that models:**
    - **Customers (customer_id, name, email, region).**
    - **Products (product_id, name, category, price).**
    - **Reviews (review_id, customer_id, product_id, rating, comments, review_date).**



Relationships:

A customer can write 0 or more reviews → 1 to many relationship

A product can have 0 or more reviews → 1 to many relationship

Each review is related to 1 product & 1 customer → Many to 1 relationship

2. **Create normalized relational database schemas.**

Customers (customer_id PK, name, email, region)

Products (product_id PK, name, category, price)

Reviews (review_id PK, customer_id FK, product_id FK, rating, comments, review_date)

Customers		
Columns	Keys	Data type
customer_id	PK	Integer
name		varchar
email		varchar
region		varchar

Products		
Columns	Keys	Data type
product_id	PK	Integer
name		varchar
category		varchar
price		Decimal

Reviews		
Columns	Keys	Data type
review_id	PK	Integer
customer_id	FK	Integer
product_id	FK	Integer
rating		Integer
comments		Text
review_date		Date

3. Define primary keys and foreign keys to enforce relationships.

Primary Key:

Customers table → customer_id

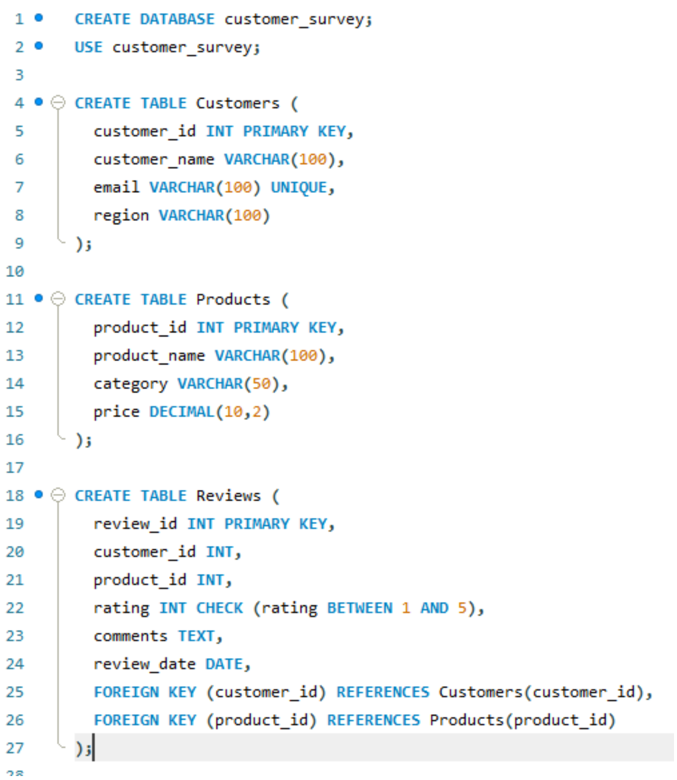
Products table → product_id

Reviews table → review_id

Foreign Key:

Reviews table → customer_id, product_id

4. Implement database tables using SQL commands.



```sql
CREATE DATABASE customer_survey;
USE customer_survey;

CREATE TABLE Customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
  region VARCHAR(100)
);

CREATE TABLE Products (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(100),
  category VARCHAR(50),
  price DECIMAL(10,2)
);

CREATE TABLE Reviews (
  review_id INT PRIMARY KEY,
  customer_id INT,
  product_id INT,
  rating INT CHECK (rating BETWEEN 1 AND 5),
  comments TEXT,
  review_date DATE,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

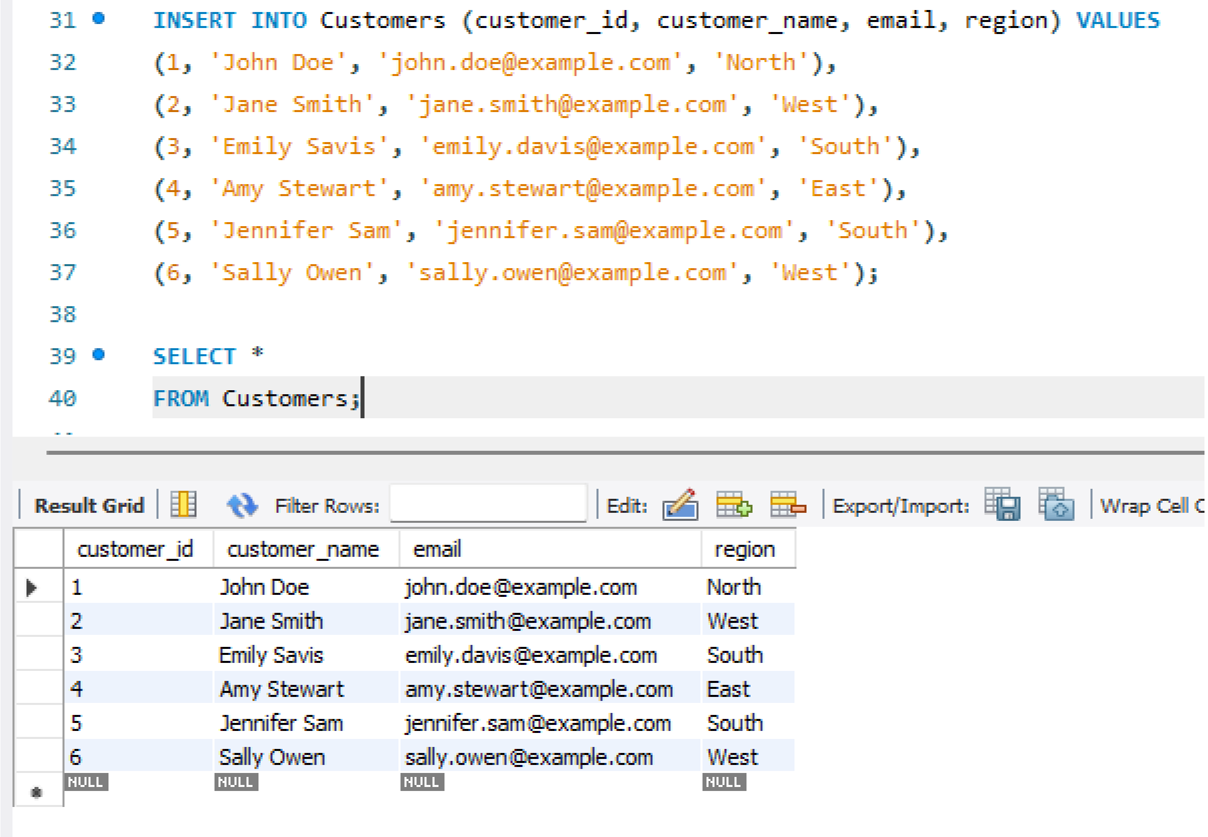
Part 2: SQL Queries and Optimization (Lessons 8–13)

1. **Write SQL queries to:**

   ○ **Insert data into tables from parsed files.**

I inserted the data into SQL instead of parsing files since 'Part 3' question is about importing data from parsed files.

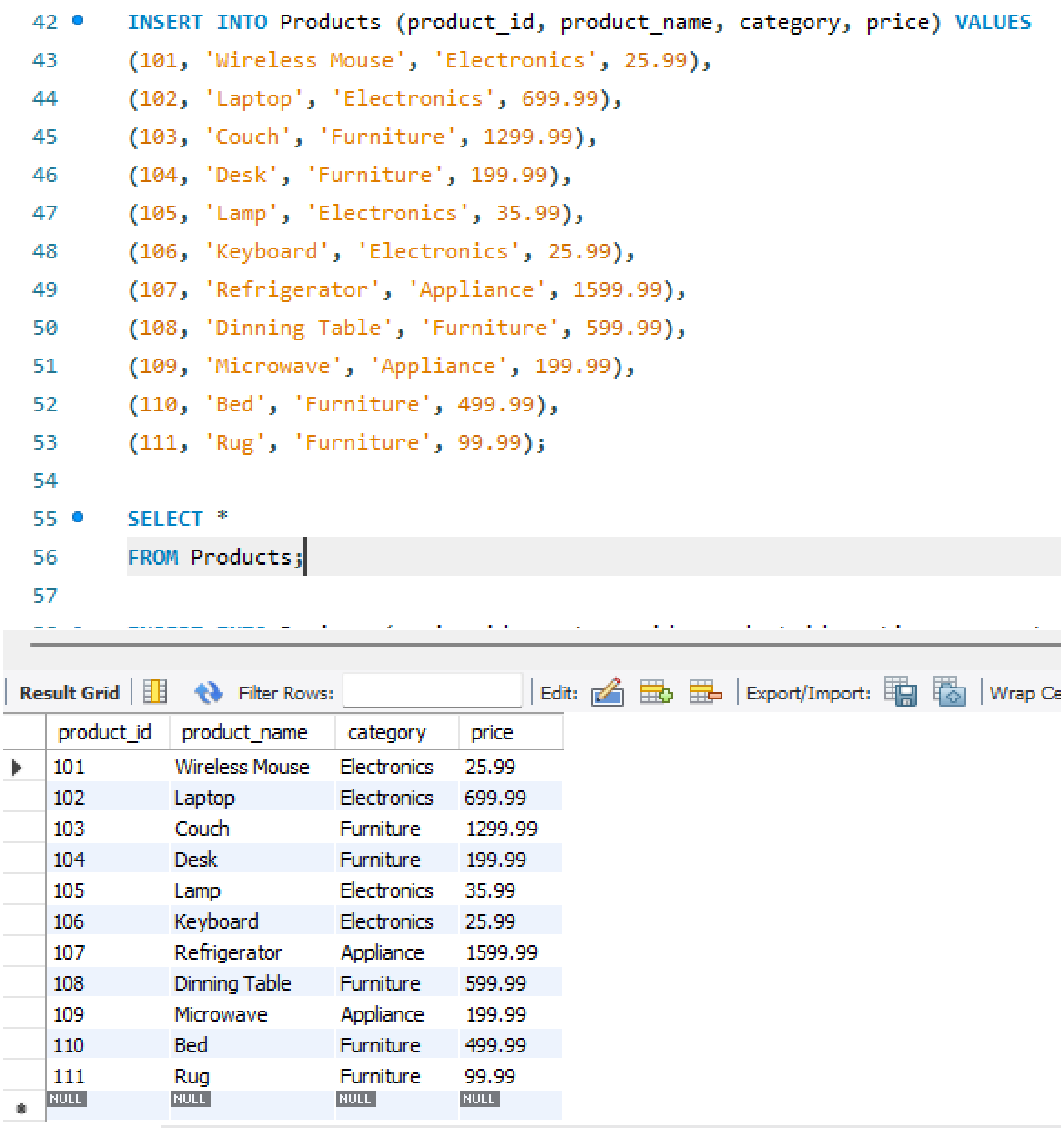
Customers Table:



```sql
INSERT INTO Customers (customer_id, customer_name, email, region) VALUES
(1, 'John Doe', 'john.doe@example.com', 'North'),
(2, 'Jane Smith', 'jane.smith@example.com', 'West'),
(3, 'Emily Savis', 'emily.davis@example.com', 'South'),
(4, 'Amy Stewart', 'amy.stewart@example.com', 'East'),
(5, 'Jennifer Sam', 'jennifer.sam@example.com', 'South'),
(6, 'Sally Owen', 'sally.owen@example.com', 'West');

SELECT *
FROM Customers;
```

customer_id	customer_name	email	region
1	John Doe	john.doe@example.com	North
2	Jane Smith	jane.smith@example.com	West
3	Emily Savis	emily.davis@example.com	South
4	Amy Stewart	amy.stewart@example.com	East
5	Jennifer Sam	jennifer.sam@example.com	South
6	Sally Owen	sally.owen@example.com	West
NULL	NULL	NULL	NULL

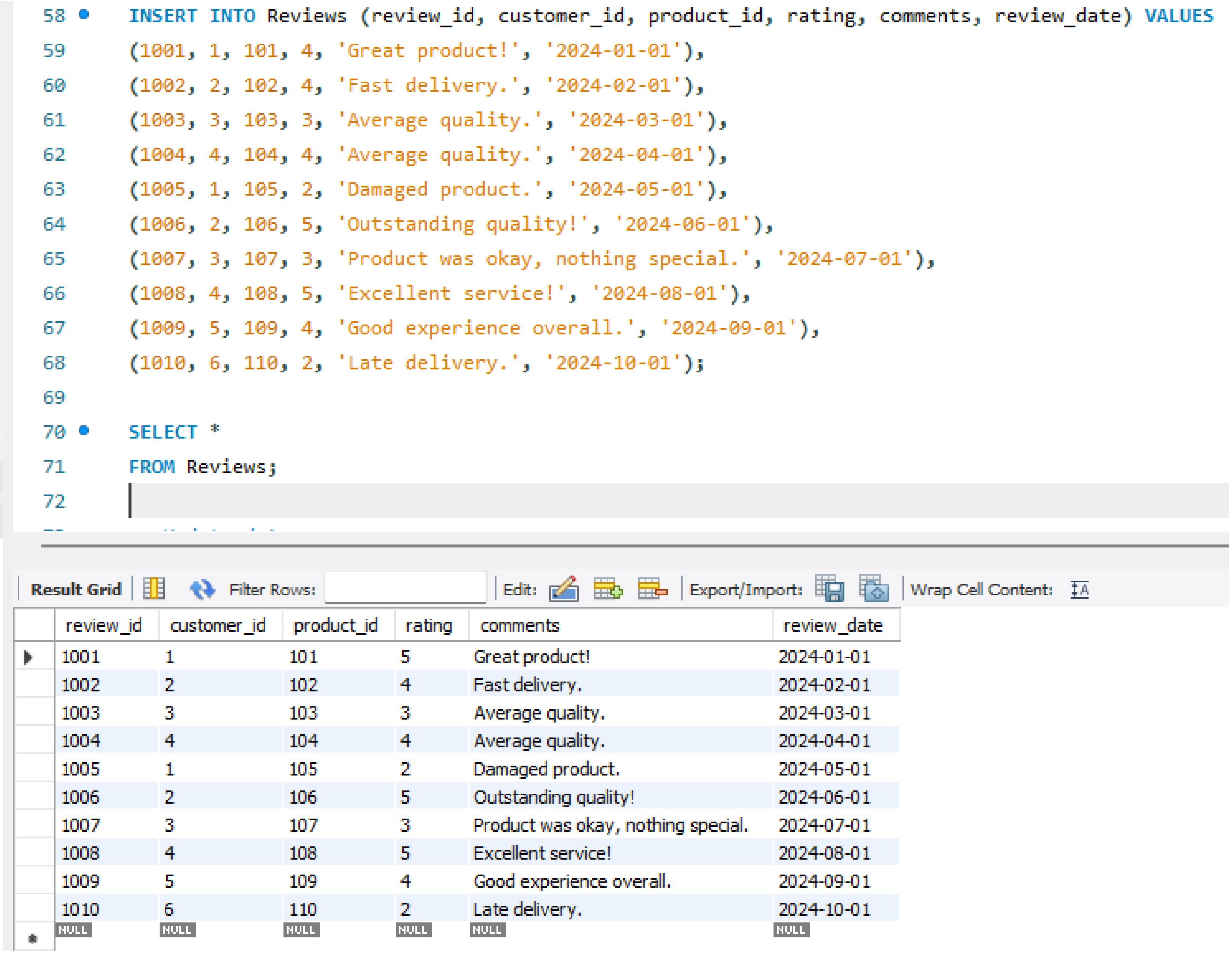
Products Table:



```sql
INSERT INTO Products (product_id, product_name, category, price) VALUES
(101, 'Wireless Mouse', 'Electronics', 25.99),
(102, 'Laptop', 'Electronics', 699.99),
(103, 'Couch', 'Furniture', 1299.99),
(104, 'Desk', 'Furniture', 199.99),
(105, 'Lamp', 'Electronics', 35.99),
(106, 'Keyboard', 'Electronics', 25.99),
(107, 'Refrigerator', 'Appliance', 1599.99),
(108, 'Dinning Table', 'Furniture', 599.99),
(109, 'Microwave', 'Appliance', 199.99),
(110, 'Bed', 'Furniture', 499.99),
(111, 'Rug', 'Furniture', 99.99);

SELECT *
FROM Products;
```

product_id	product_name	category	price
101	Wireless Mouse	Electronics	25.99
102	Laptop	Electronics	699.99
103	Couch	Furniture	1299.99
104	Desk	Furniture	199.99
105	Lamp	Electronics	35.99
106	Keyboard	Electronics	25.99
107	Refrigerator	Appliance	1599.99
108	Dinning Table	Furniture	599.99
109	Microwave	Appliance	199.99
110	Bed	Furniture	499.99
111	Rug	Furniture	99.99
NULL	NULL	NULL	NULL

Reviews Table:

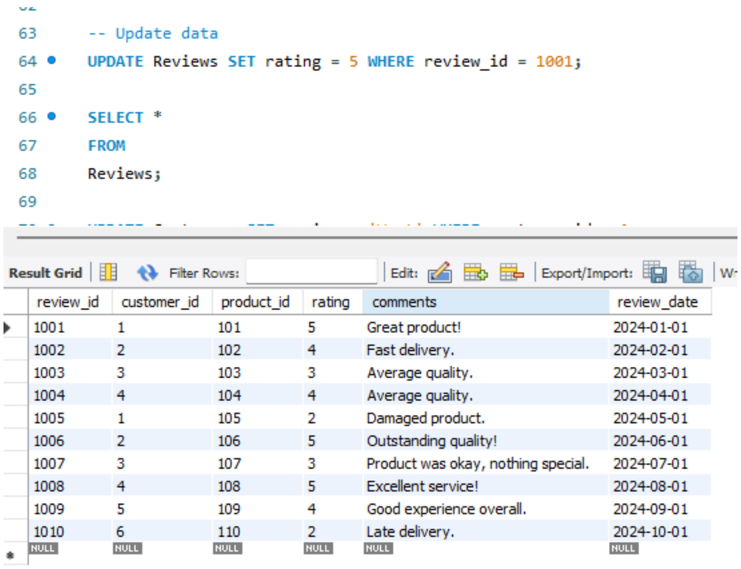


```sql
INSERT INTO Reviews (review_id, customer_id, product_id, rating, comments, review_date) VALUES
(1001, 1, 101, 4, 'Great product!', '2024-01-01'),
(1002, 2, 102, 4, 'Fast delivery.', '2024-02-01'),
(1003, 3, 103, 3, 'Average quality.', '2024-03-01'),
(1004, 4, 104, 4, 'Average quality.', '2024-04-01'),
(1005, 1, 105, 2, 'Damaged product.', '2024-05-01'),
(1006, 2, 106, 5, 'Outstanding quality!', '2024-06-01'),
(1007, 3, 107, 3, 'Product was okay, nothing special.', '2024-07-01'),
(1008, 4, 108, 5, 'Excellent service!', '2024-08-01'),
(1009, 5, 109, 4, 'Good experience overall.', '2024-09-01'),
(1010, 6, 110, 2, 'Late delivery.', '2024-10-01');

SELECT *
FROM Reviews;
```

review_id	customer_id	product_id	rating	comments	review_date
1001	1	101	5	Great product!	2024-01-01
1002	2	102	4	Fast delivery.	2024-02-01
1003	3	103	3	Average quality.	2024-03-01
1004	4	104	4	Average quality.	2024-04-01
1005	1	105	2	Damaged product.	2024-05-01
1006	2	106	5	Outstanding quality!	2024-06-01
1007	3	107	3	Product was okay, nothing special.	2024-07-01
1008	4	108	5	Excellent service!	2024-08-01
1009	5	109	4	Good experience overall.	2024-09-01
1010	6	110	2	Late delivery.	2024-10-01
NULL	NULL	NULL	NULL	NULL	NULL

- **Update records to reflect any data changes or corrections.**

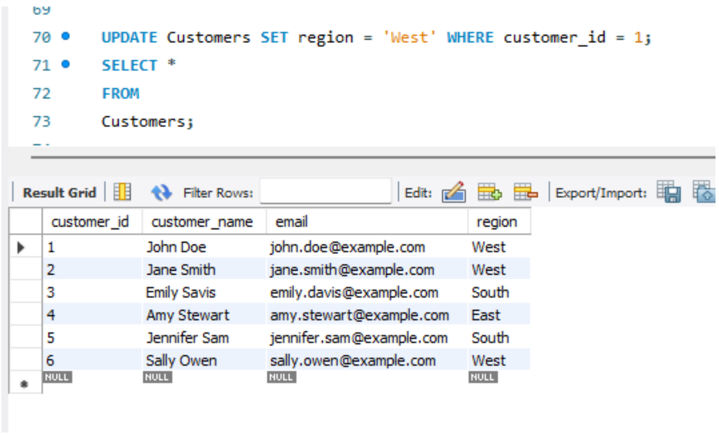
Updated a wrong rating for review_id 1001



```sql
-- Update data
UPDATE Reviews SET rating = 5 WHERE review_id = 1001;

SELECT *
FROM
Reviews;
```

review_id	customer_id	product_id	rating	comments	review_date
1001	1	101	5	Great product!	2024-01-01
1002	2	102	4	Fast delivery.	2024-02-01
1003	3	103	3	Average quality.	2024-03-01
1004	4	104	4	Average quality.	2024-04-01
1005	1	105	2	Damaged product.	2024-05-01
1006	2	106	5	Outstanding quality!	2024-06-01
1007	3	107	3	Product was okay, nothing special.	2024-07-01
1008	4	108	5	Excellent service!	2024-08-01
1009	5	109	4	Good experience overall.	2024-09-01
1010	6	110	2	Late delivery.	2024-10-01
NULL	NULL	NULL	NULL	NULL	NULL

Updated the region of customer_id 1 since the customer changed his address



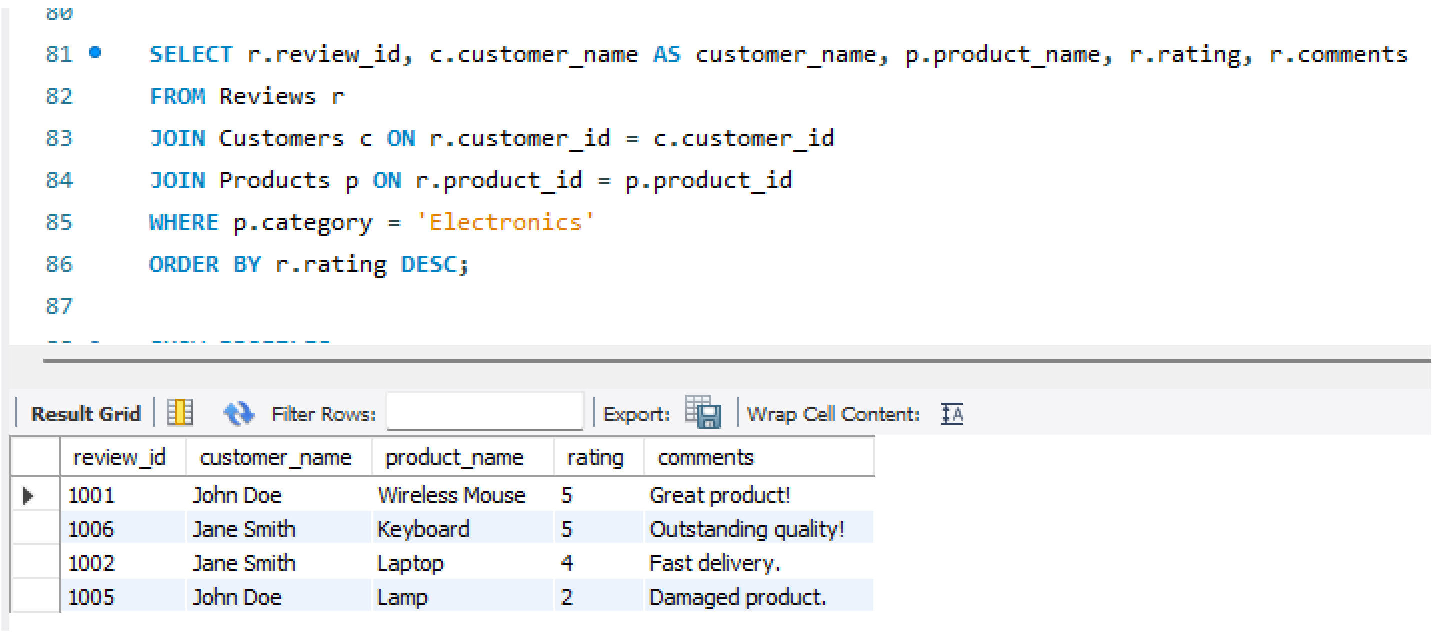
```
UPDATE Customers SET region = 'West' WHERE customer_id = 1;
SELECT *
FROM
Customers;
```

customer_id	customer_name	email	region
1	John Doe	john.doe@example.com	West
2	Jane Smith	jane.smith@example.com	West
3	Emily Savis	emily.davis@example.com	South
4	Amy Stewart	amy.stewart@example.com	East
5	Jennifer Sam	jennifer.sam@example.com	South
6	Sally Owen	sally.owen@example.com	West
NULL	NULL	NULL	NULL

- **Retrieve data using SELECT statements with filtering, grouping, and sorting.**

Query 1:

--Get the details for those products under 'Electronics' category

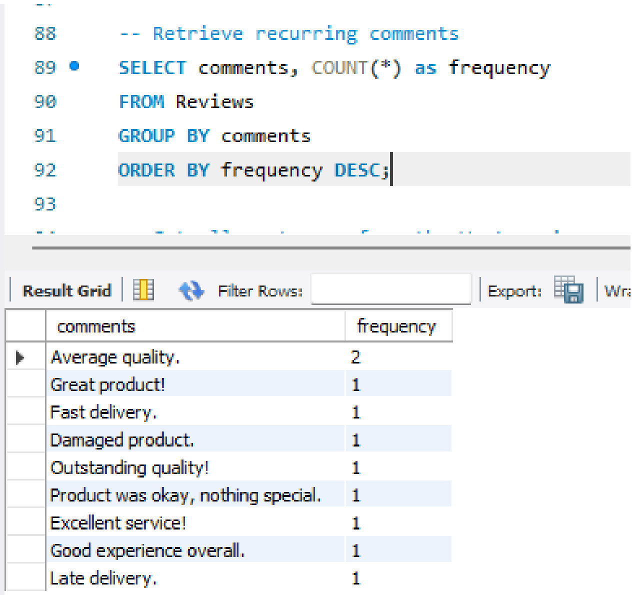


```
SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.comments
FROM Reviews r
JOIN Customers c ON r.customer_id = c.customer_id
JOIN Products p ON r.product_id = p.product_id
WHERE p.category = 'Electronics'
ORDER BY r.rating DESC;
```

review_id	customer_name	product_name	rating	comments
1001	John Doe	Wireless Mouse	5	Great product!
1006	Jane Smith	Keyboard	5	Outstanding quality!
1002	Jane Smith	Laptop	4	Fast delivery.
1005	John Doe	Lamp	2	Damaged product.

Query 2:

--Retrieve recurring comments

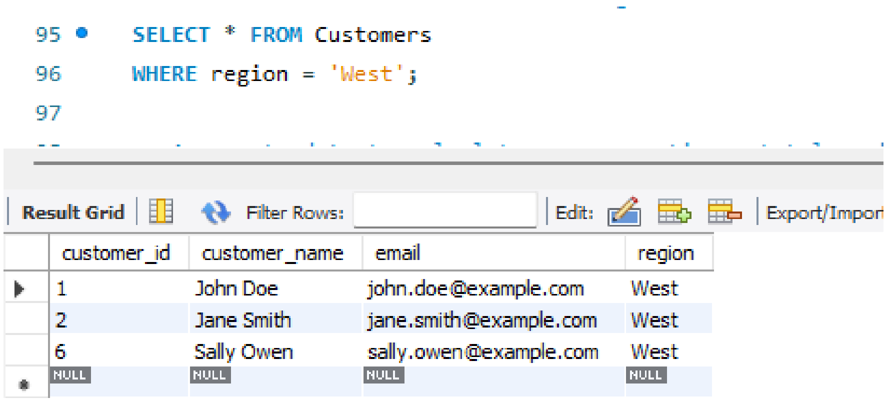


```sql
-- Retrieve recurring comments
SELECT comments, COUNT(*) as frequency
FROM Reviews
GROUP BY comments
ORDER BY frequency DESC;
```

comments	frequency
Average quality.	2
Great product!	1
Fast delivery.	1
Damaged product.	1
Outstanding quality!	1
Product was okay, nothing special.	1
Excellent service!	1
Good experience overall.	1
Late delivery.	1

Query 3:

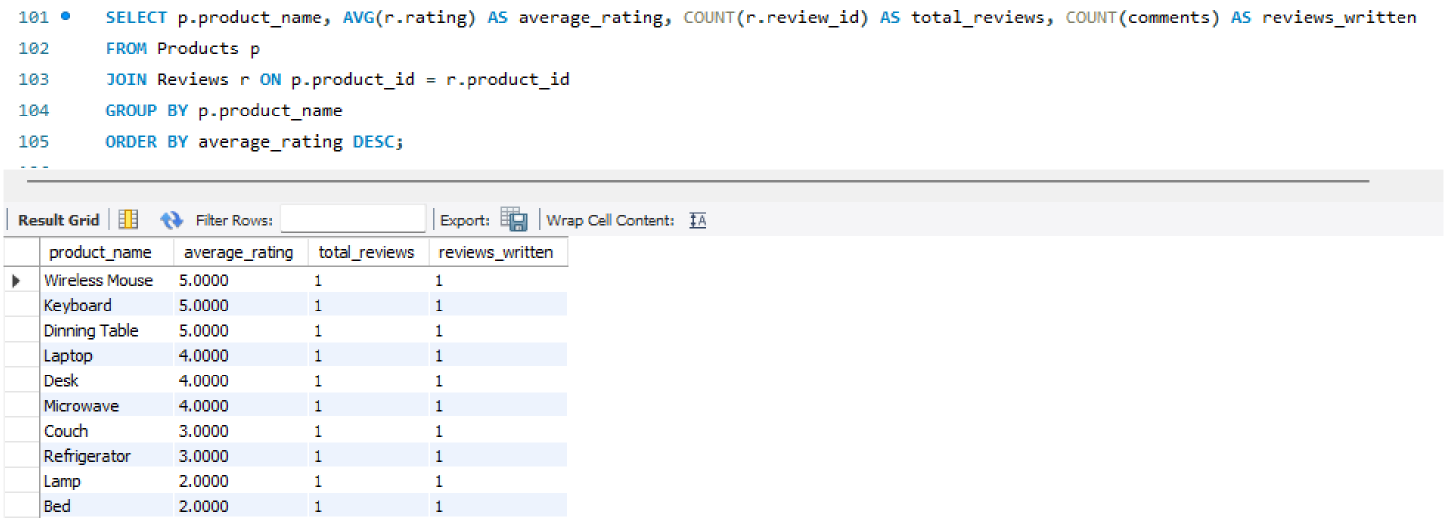
--Filter only the customers from the 'West' region



```sql
SELECT * FROM Customers
WHERE region = 'West';
```

customer_id	customer_name	email	region
1	John Doe	john.doe@example.com	West
2	Jane Smith	jane.smith@example.com	West
6	Sally Owen	sally.owen@example.com	West
NULL	NULL	NULL	NULL

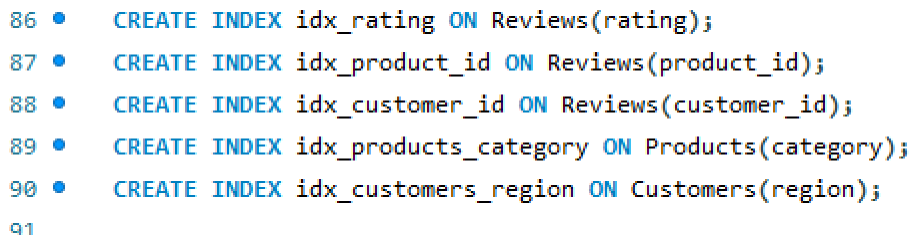
- **Aggregate data to calculate average ratings, total reviews, and customer engagement.**



```sql
SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUNT(comments) AS reviews_written
FROM Products p
JOIN Reviews r ON p.product_id = r.product_id
GROUP BY p.product_name
ORDER BY average_rating DESC;
```

product_name	average_rating	total_reviews	reviews_written
Wireless Mouse	5.0000	1	1
Keyboard	5.0000	1	1
Dinning Table	5.0000	1	1
Laptop	4.0000	1	1
Desk	4.0000	1	1
Microwave	4.0000	1	1
Couch	3.0000	1	1
Refrigerator	3.0000	1	1
Lamp	2.0000	1	1
Bed	2.0000	1	1

2. Optimize queries with indexing for faster retrieval.



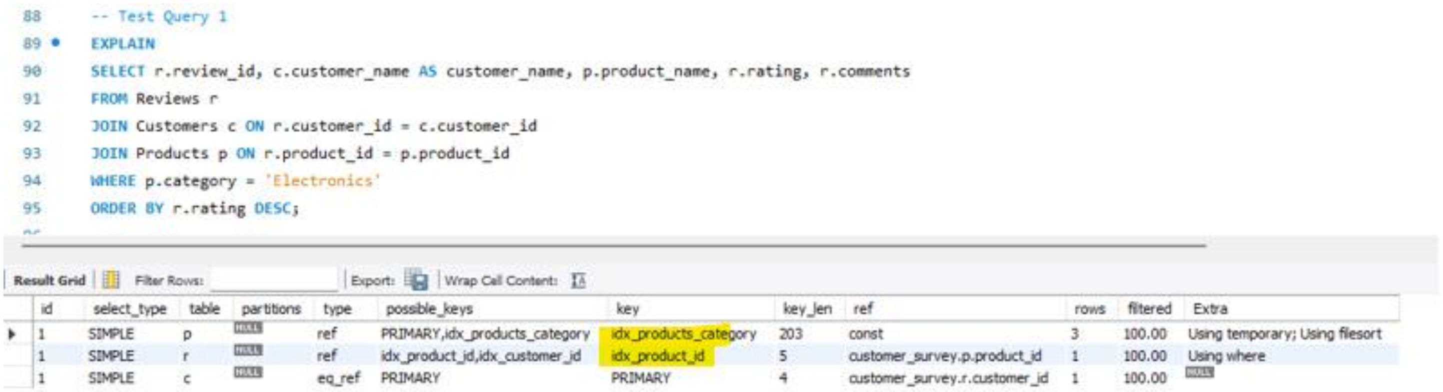
```sql
CREATE INDEX idx_rating ON Reviews(rating);
CREATE INDEX idx_product_id ON Reviews(product_id);
CREATE INDEX idx_customer_id ON Reviews(customer_id);
CREATE INDEX idx_products_category ON Products(category);
CREATE INDEX idx_customers_region ON Customers(region);
```

3. Test SQL queries

The queries are tested if working correctly & results shared in previous questions above.

Testing indexes:

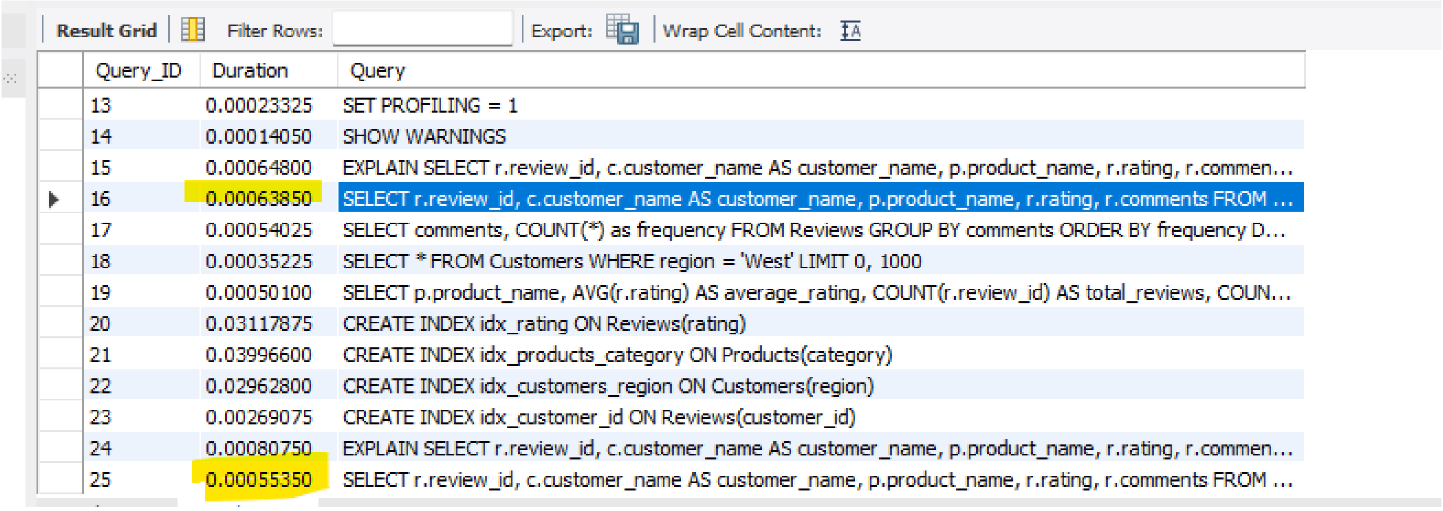
--Query 1 is now using indexes to retrieve data which should be more efficient



```sql
-- Test Query 1
EXPLAIN
SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.comments
FROM Reviews r
JOIN Customers c ON r.customer_id = c.customer_id
JOIN Products p ON r.product_id = p.product_id
WHERE p.category = 'Electronics'
ORDER BY r.rating DESC;
```

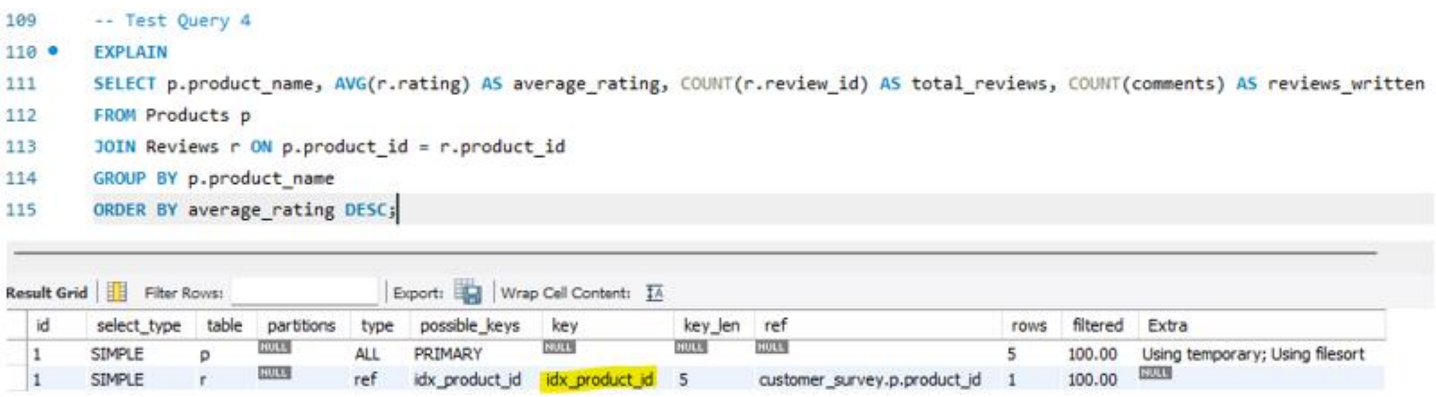
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ref	PRIMARY,idx_products_category	idx_products_category	203	const	3	100.00	Using temporary; Using filesort
1	SIMPLE	r	NULL	ref	idx_product_id,idx_customer_id	idx_product_id	5	customer_survey.p.product_id	1	100.00	Using where
1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	4	customer_survey.r.customer_id	1	100.00	NULL

--Duration of the query execution has decreased after creating the indexes:



Query_ID	Duration	Query
13	0.00023325	SET PROFILING = 1
14	0.00014050	SHOW WARNINGS
15	0.00064800	EXPLAIN SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.commen...
16	0.00063850	SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.comments FROM ...
17	0.00054025	SELECT comments, COUNT(*) as frequency FROM Reviews GROUP BY comments ORDER BY frequency D...
18	0.00035225	SELECT * FROM Customers WHERE region = 'West' LIMIT 0, 1000
19	0.00050100	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
20	0.03117875	CREATE INDEX idx_rating ON Reviews(rating)
21	0.03996600	CREATE INDEX idx_products_category ON Products(category)
22	0.02962800	CREATE INDEX idx_customers_region ON Customers(region)
23	0.00269075	CREATE INDEX idx_customer_id ON Reviews(customer_id)
24	0.00080750	EXPLAIN SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.commen...
25	0.00055350	SELECT r.review_id, c.customer_name AS customer_name, p.product_name, r.rating, r.comments FROM ...

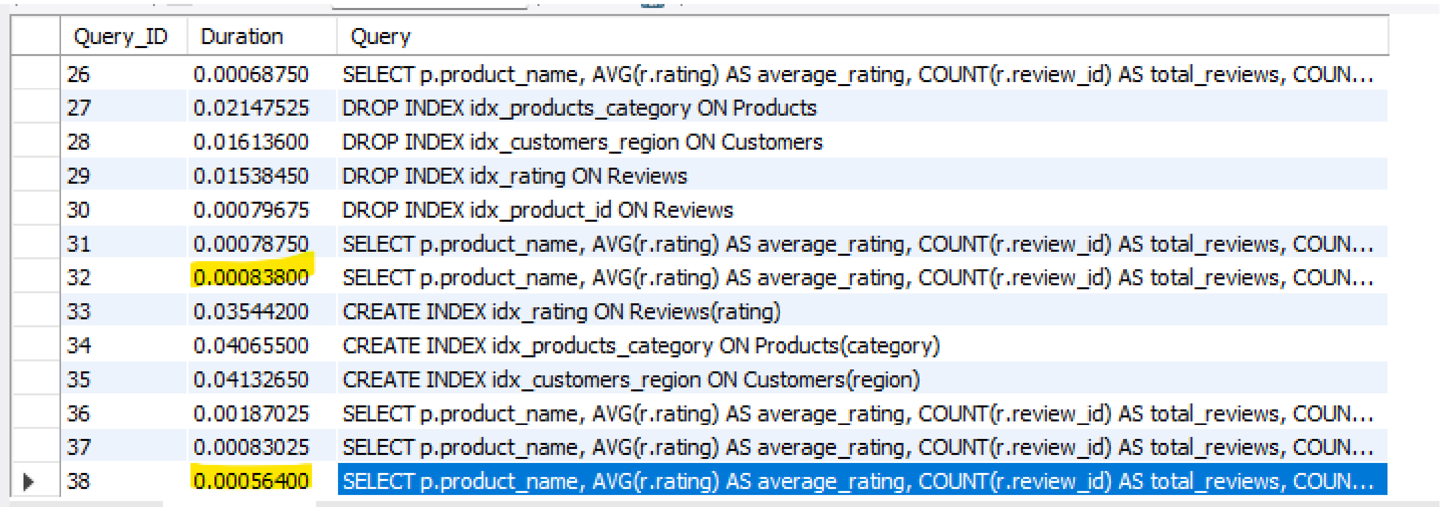
--Query 4 is now using index from the Reviews table to retrieve data which should be more efficient



```
-- Test Query 4
EXPLAIN
SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUNT(comments) AS reviews_written
FROM Products p
JOIN Reviews r ON p.product_id = r.product_id
GROUP BY p.product_name
ORDER BY average_rating DESC;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	100.00	Using temporary; Using filesort
1	SIMPLE	r	NULL	ref	idx_product_id	idx_product_id	5	customer_survey.p.product_id	1	100.00	NULL

--Duration of the query execution has decreased after creating the index:

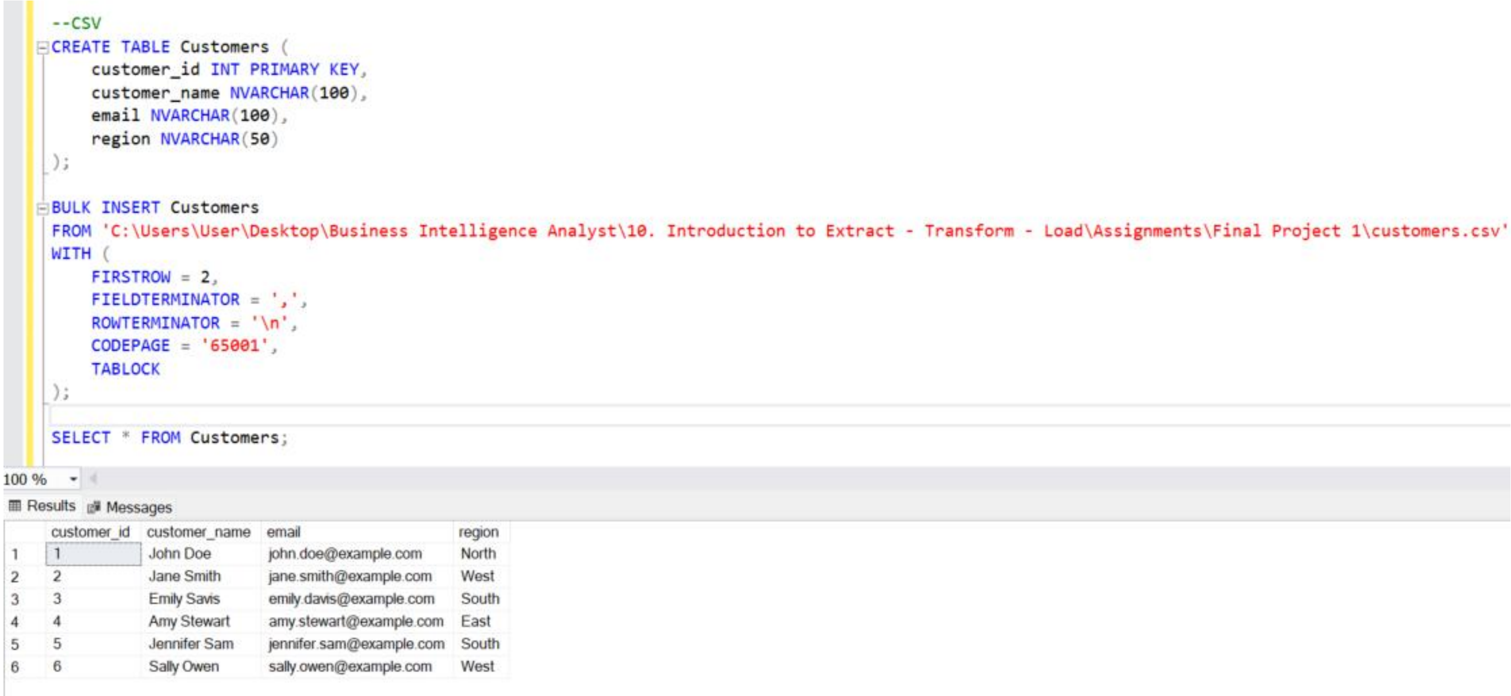


Query_ID	Duration	Query
26	0.00068750	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
27	0.02147525	DROP INDEX idx_products_category ON Products
28	0.01613600	DROP INDEX idx_customers_region ON Customers
29	0.01538450	DROP INDEX idx_rating ON Reviews
30	0.00079675	DROP INDEX idx_product_id ON Reviews
31	0.00078750	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
32	0.00083800	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
33	0.03544200	CREATE INDEX idx_rating ON Reviews(rating)
34	0.04065500	CREATE INDEX idx_products_category ON Products(category)
35	0.04132650	CREATE INDEX idx_customers_region ON Customers(region)
36	0.00187025	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
37	0.00083025	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...
38	0.00056400	SELECT p.product_name, AVG(r.rating) AS average_rating, COUNT(r.review_id) AS total_reviews, COUN...

Part 3: Data Integration

1. **Import data from multiple formats:**

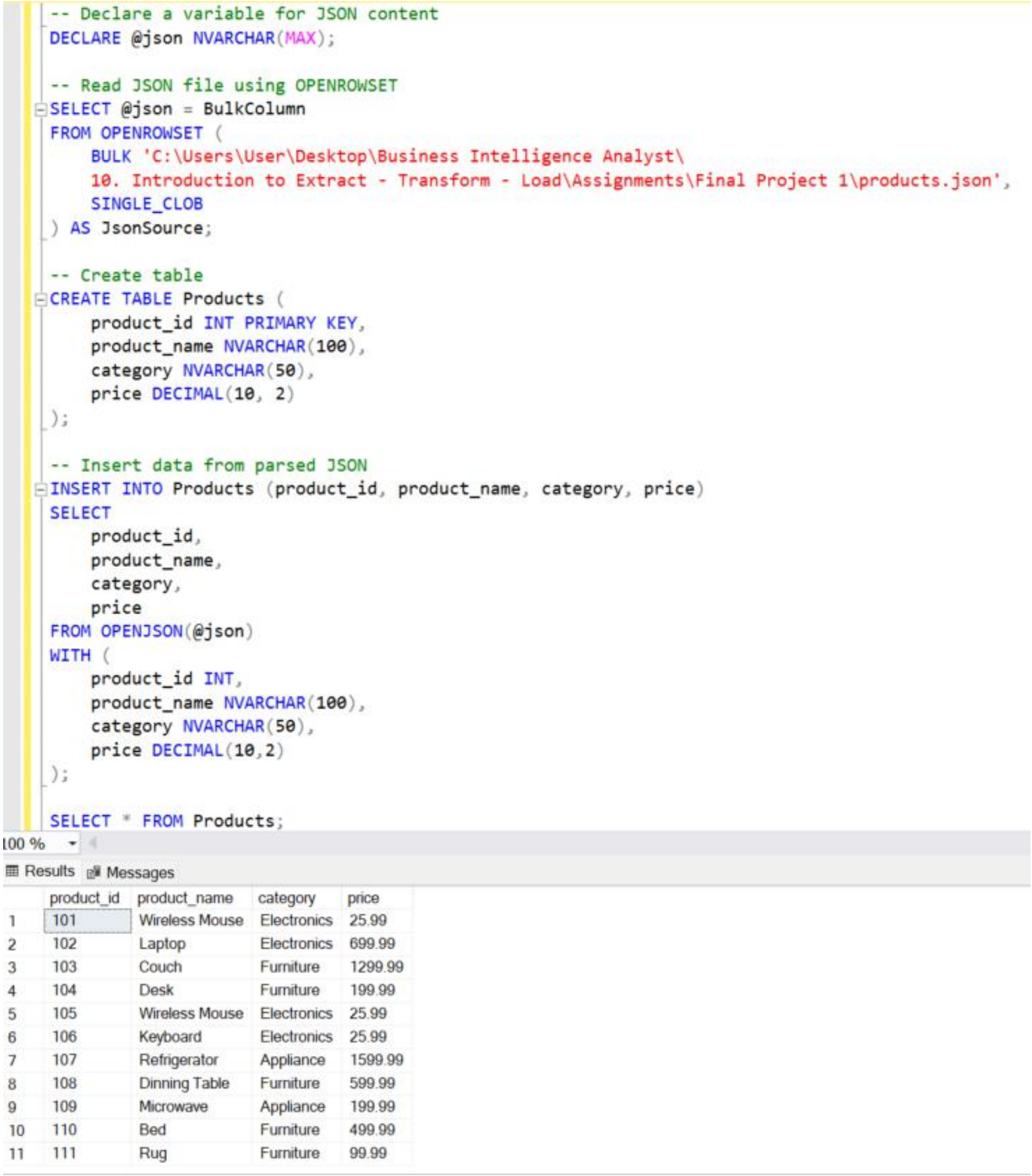
    - **Load CSV file ('customers.csv') into temporary tables using bulk insert commands.**



```sql
--CSV
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    customer_name NVARCHAR(100),
    email NVARCHAR(100),
    region NVARCHAR(50)
);

BULK INSERT Customers
FROM 'C:\Users\User\Desktop\Business Intelligence Analyst\10. Introduction to Extract - Transform - Load\Assignments\Final Project 1\customers.csv'
WITH (
    FIRSTROW = 2,
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    CODEPAGE = '65001',
    TABLOCK
);

SELECT * FROM Customers;
```

	customer_id	customer_name	email	region
1	1	John Doe	john.doe@example.com	North
2	2	Jane Smith	jane.smith@example.com	West
3	3	Emily Savis	emily.davis@example.com	South
4	4	Amy Stewart	amy.stewart@example.com	East
5	5	Jennifer Sam	jennifer.sam@example.com	South
6	6	Sally Owen	sally.owen@example.com	West

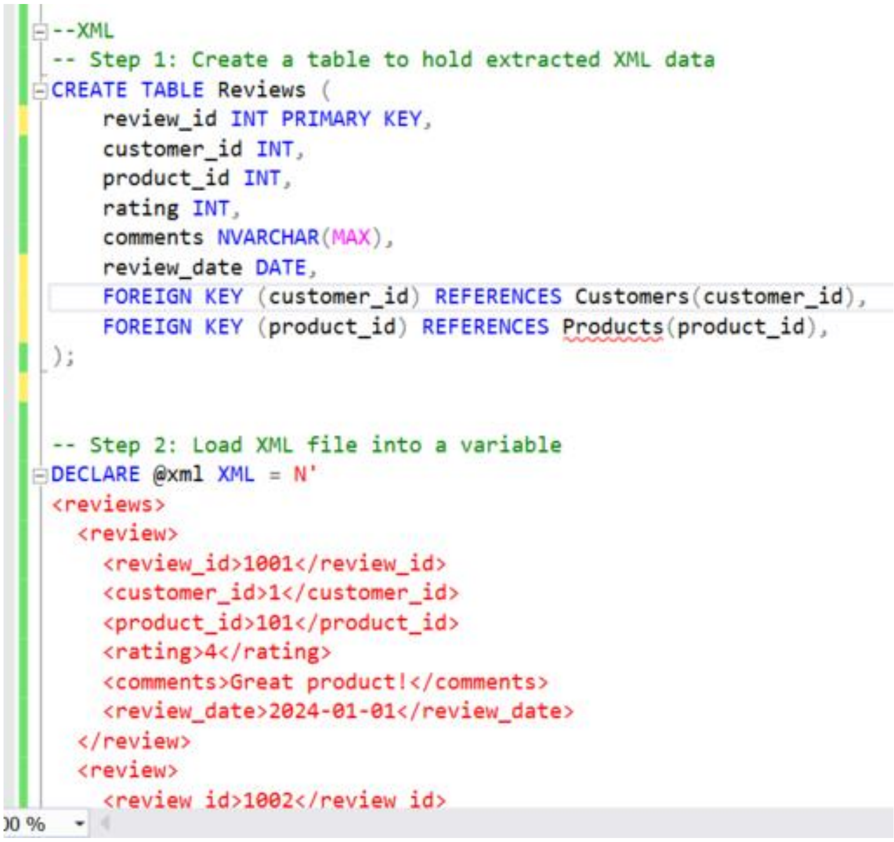
- **Parse JSON file ('products.json') into relational format using JSON functions or scripts.**



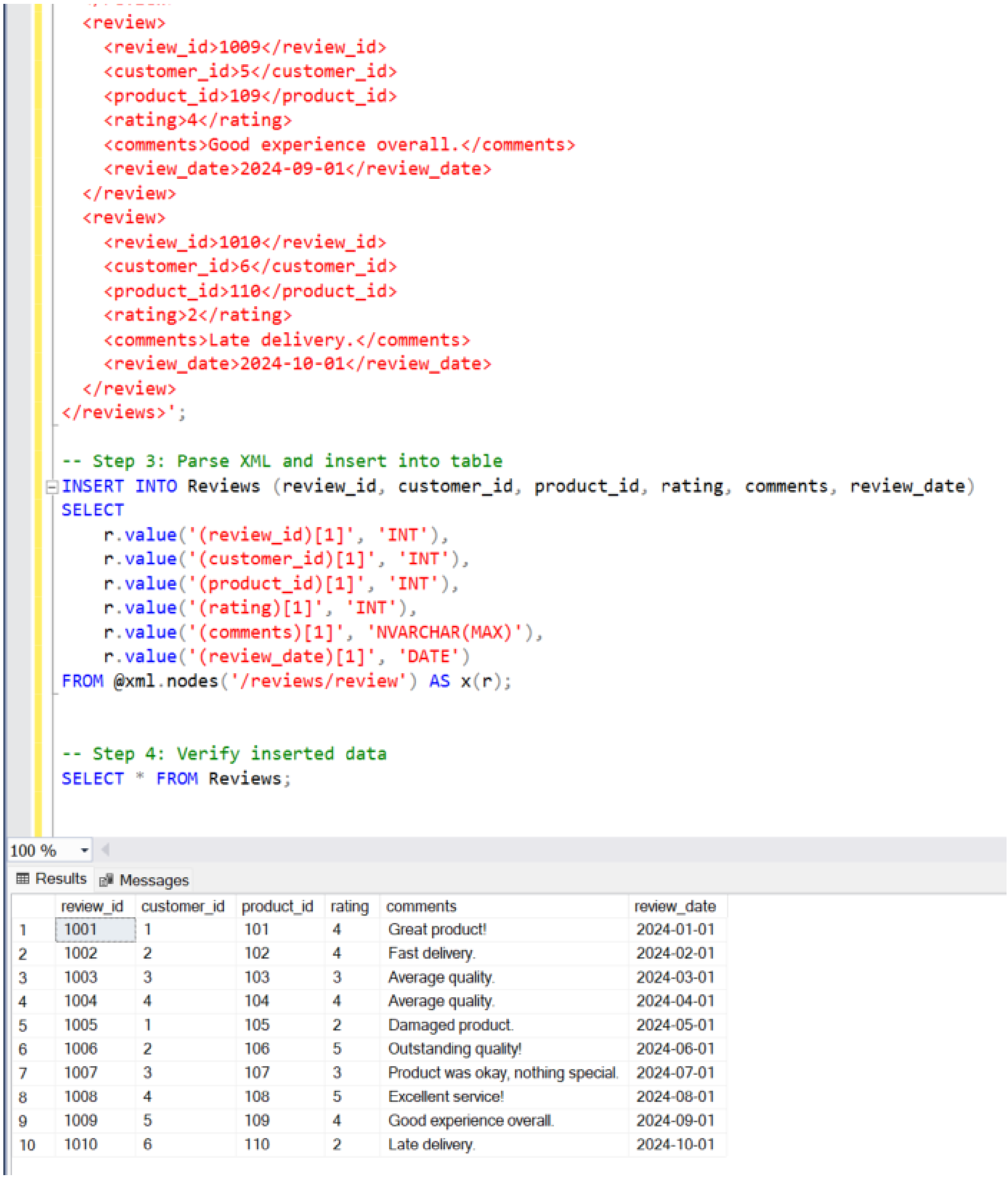
```sql
-- Declare a variable for JSON content
DECLARE @json NVARCHAR(MAX);

-- Read JSON file using OPENROWSET
SELECT @json = BulkColumn
FROM OPENROWSET (
    BULK 'C:\Users\User\Desktop\Business Intelligence Analyst\
    10. Introduction to Extract - Transform - Load\Assignments\Final Project 1\products.json',
    SINGLE_CLOB
) AS JsonSource;

-- Create table
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name NVARCHAR(100),
    category NVARCHAR(50),
    price DECIMAL(10, 2)
);

-- Insert data from parsed JSON
INSERT INTO Products (product_id, product_name, category, price)
SELECT
    product_id,
    product_name,
    category,
    price
FROM OPENJSON(@json)
WITH (
    product_id INT,
    product_name NVARCHAR(100),
    category NVARCHAR(50),
    price DECIMAL(10,2)
);

SELECT * FROM Products;
```

	product_id	product_name	category	price
1	101	Wireless Mouse	Electronics	25.99
2	102	Laptop	Electronics	699.99
3	103	Couch	Furniture	1299.99
4	104	Desk	Furniture	199.99
5	105	Wireless Mouse	Electronics	25.99
6	106	Keyboard	Electronics	25.99
7	107	Refrigerator	Appliance	1599.99
8	108	Dinning Table	Furniture	599.99
9	109	Microwave	Appliance	199.99
10	110	Bed	Furniture	499.99
11	111	Rug	Furniture	99.99

- **Extract XML data (e.g., 'external_reviews.xml') into structured tables using XML parsing tools.**



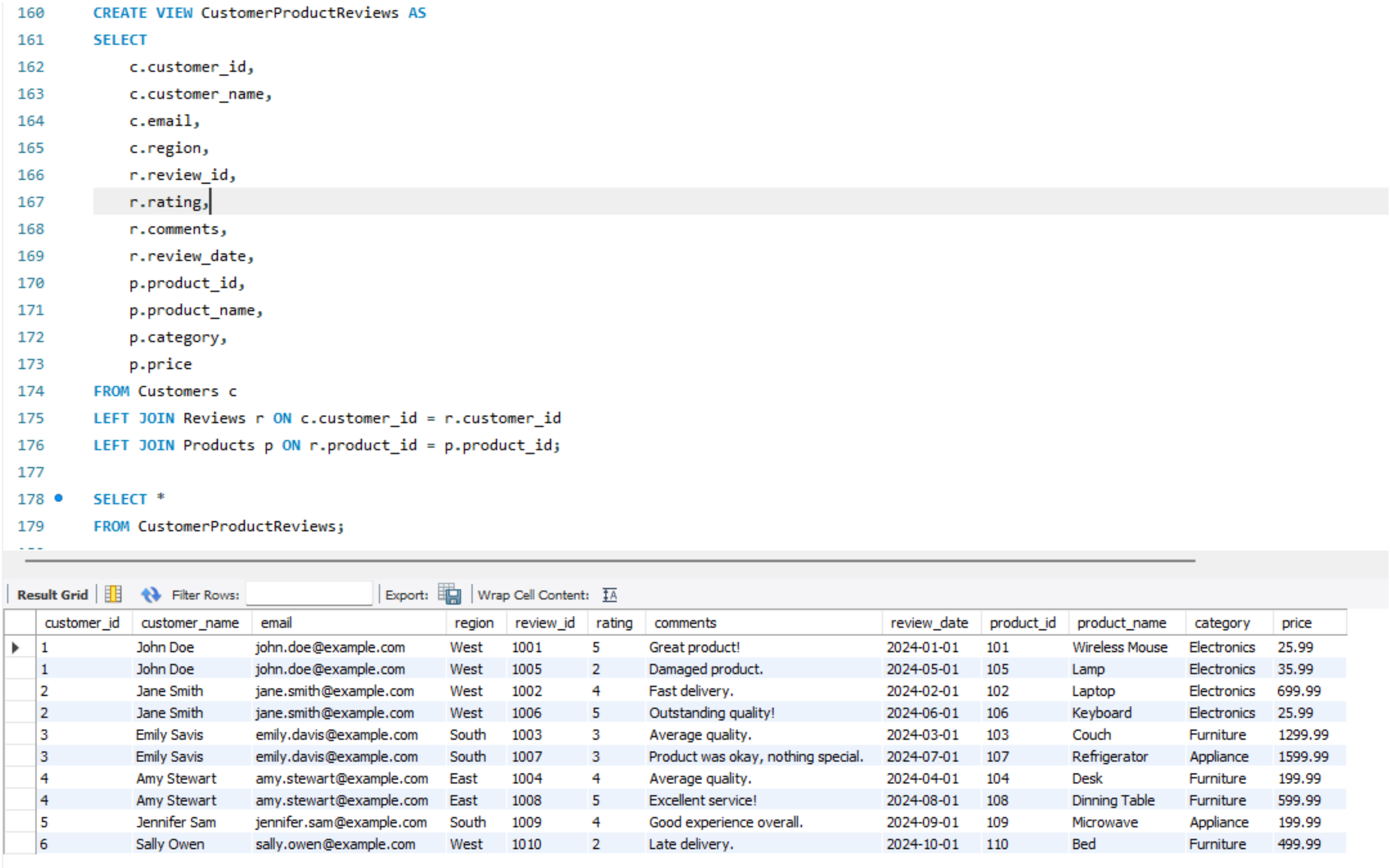
```
--XML
-- Step 1: Create a table to hold extracted XML data
CREATE TABLE Reviews (
    review_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    rating INT,
    comments NVARCHAR(MAX),
    review_date DATE,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id),
);


-- Step 2: Load XML file into a variable
DECLARE @xml XML = N'
<reviews>
  <review>
    <review_id>1001</review_id>
    <customer_id>1</customer_id>
    <product_id>101</product_id>
    <rating>4</rating>
    <comments>Great product!</comments>
    <review_date>2024-01-01</review_date>
  </review>
  <review>
    <review_id>1002</review_id>
```



```
  <review>
    <review_id>1009</review_id>
    <customer_id>5</customer_id>
    <product_id>109</product_id>
    <rating>4</rating>
    <comments>Good experience overall.</comments>
    <review_date>2024-09-01</review_date>
  </review>
  <review>
    <review_id>1010</review_id>
    <customer_id>6</customer_id>
    <product_id>110</product_id>
    <rating>2</rating>
    <comments>Late delivery.</comments>
    <review_date>2024-10-01</review_date>
  </review>
</reviews>';

-- Step 3: Parse XML and insert into table
INSERT INTO Reviews (review_id, customer_id, product_id, rating, comments, review_date)
SELECT
    r.value('(review_id)[1]', 'INT'),
    r.value('(customer_id)[1]', 'INT'),
    r.value('(product_id)[1]', 'INT'),
    r.value('(rating)[1]', 'INT'),
    r.value('(comments)[1]', 'NVARCHAR(MAX)'),
    r.value('(review_date)[1]', 'DATE')
FROM @xml.nodes('/reviews/review') AS x(r);


-- Step 4: Verify inserted data
SELECT * FROM Reviews;
```

	review_id	customer_id	product_id	rating	comments	review_date
1	1001	1	101	4	Great product!	2024-01-01
2	1002	2	102	4	Fast delivery.	2024-02-01
3	1003	3	103	3	Average quality.	2024-03-01
4	1004	4	104	4	Average quality.	2024-04-01
5	1005	1	105	2	Damaged product.	2024-05-01
6	1006	2	106	5	Outstanding quality!	2024-06-01
7	1007	3	107	3	Product was okay, nothing special.	2024-07-01
8	1008	4	108	5	Excellent service!	2024-08-01
9	1009	5	109	4	Good experience overall.	2024-09-01
10	1010	6	110	2	Late delivery.	2024-10-01

2. **Write SQL joins to combine datasets:**
   - ○ **Create a view that joins customer, product, and review data by keys.**
   - ○ **Perform left joins to ensure no data is excluded due to missing relationships.**

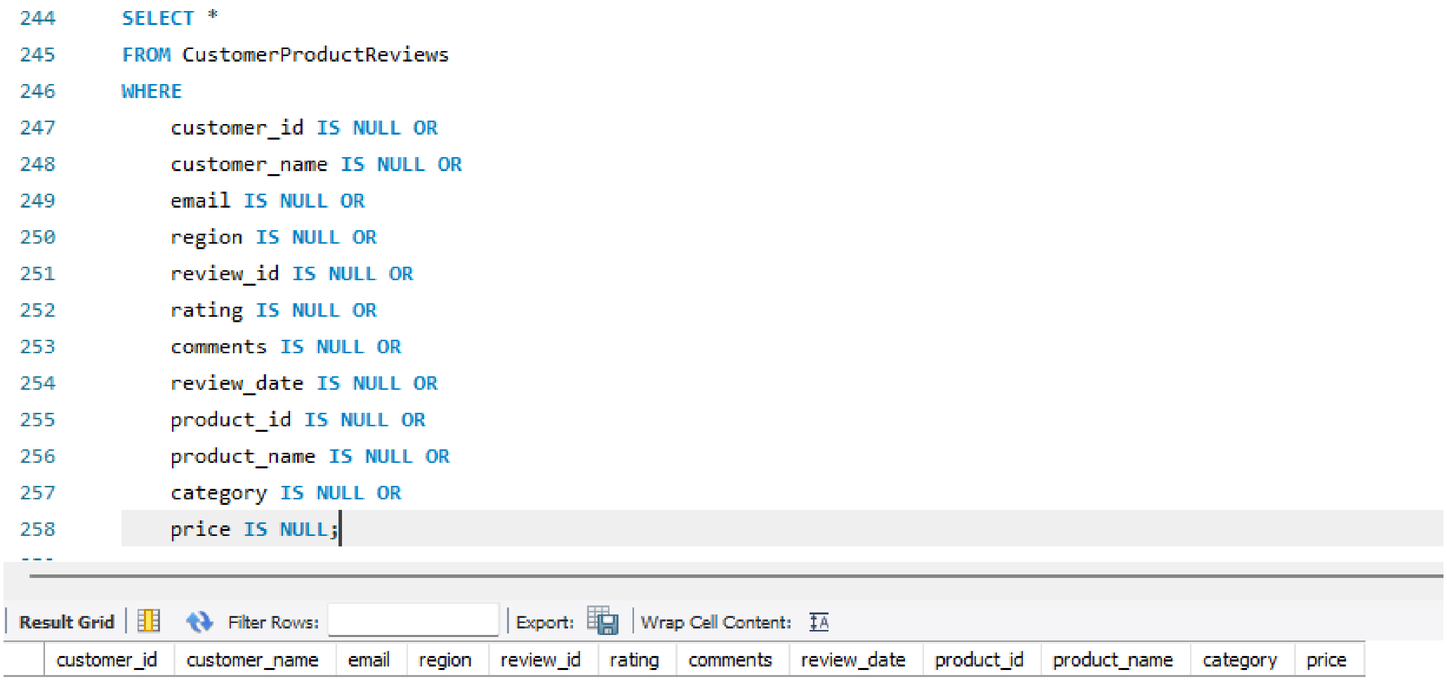


```sql
CREATE VIEW CustomerProductReviews AS
SELECT
    c.customer_id,
    c.customer_name,
    c.email,
    c.region,
    r.review_id,
    r.rating,
    r.comments,
    r.review_date,
    p.product_id,
    p.product_name,
    p.category,
    p.price
FROM Customers c
LEFT JOIN Reviews r ON c.customer_id = r.customer_id
LEFT JOIN Products p ON r.product_id = p.product_id;

SELECT *
FROM CustomerProductReviews;
```

customer_id	customer_name	email	region	review_id	rating	comments	review_date	product_id	product_name	category	price
1	John Doe	john.doe@example.com	West	1001	5	Great product!	2024-01-01	101	Wireless Mouse	Electronics	25.99
1	John Doe	john.doe@example.com	West	1005	2	Damaged product.	2024-05-01	105	Lamp	Electronics	35.99
2	Jane Smith	jane.smith@example.com	West	1002	4	Fast delivery.	2024-02-01	102	Laptop	Electronics	699.99
2	Jane Smith	jane.smith@example.com	West	1006	5	Outstanding quality!	2024-06-01	106	Keyboard	Electronics	25.99
3	Emily Savis	emily.davis@example.com	South	1003	3	Average quality.	2024-03-01	103	Couch	Furniture	1299.99
3	Emily Savis	emily.davis@example.com	South	1007	3	Product was okay, nothing special.	2024-07-01	107	Refrigerator	Appliance	1599.99
4	Amy Stewart	amy.stewart@example.com	East	1004	4	Average quality.	2024-04-01	104	Desk	Furniture	199.99
4	Amy Stewart	amy.stewart@example.com	East	1008	5	Excellent service!	2024-08-01	108	Dinning Table	Furniture	599.99
5	Jennifer Sam	jennifer.sam@example.com	South	1009	4	Good experience overall.	2024-09-01	109	Microwave	Appliance	199.99
6	Sally Owen	sally.owen@example.com	West	1010	2	Late delivery.	2024-10-01	110	Bed	Furniture	499.99

Check if we have any missing values in all the columns:

We don’t have any missing values



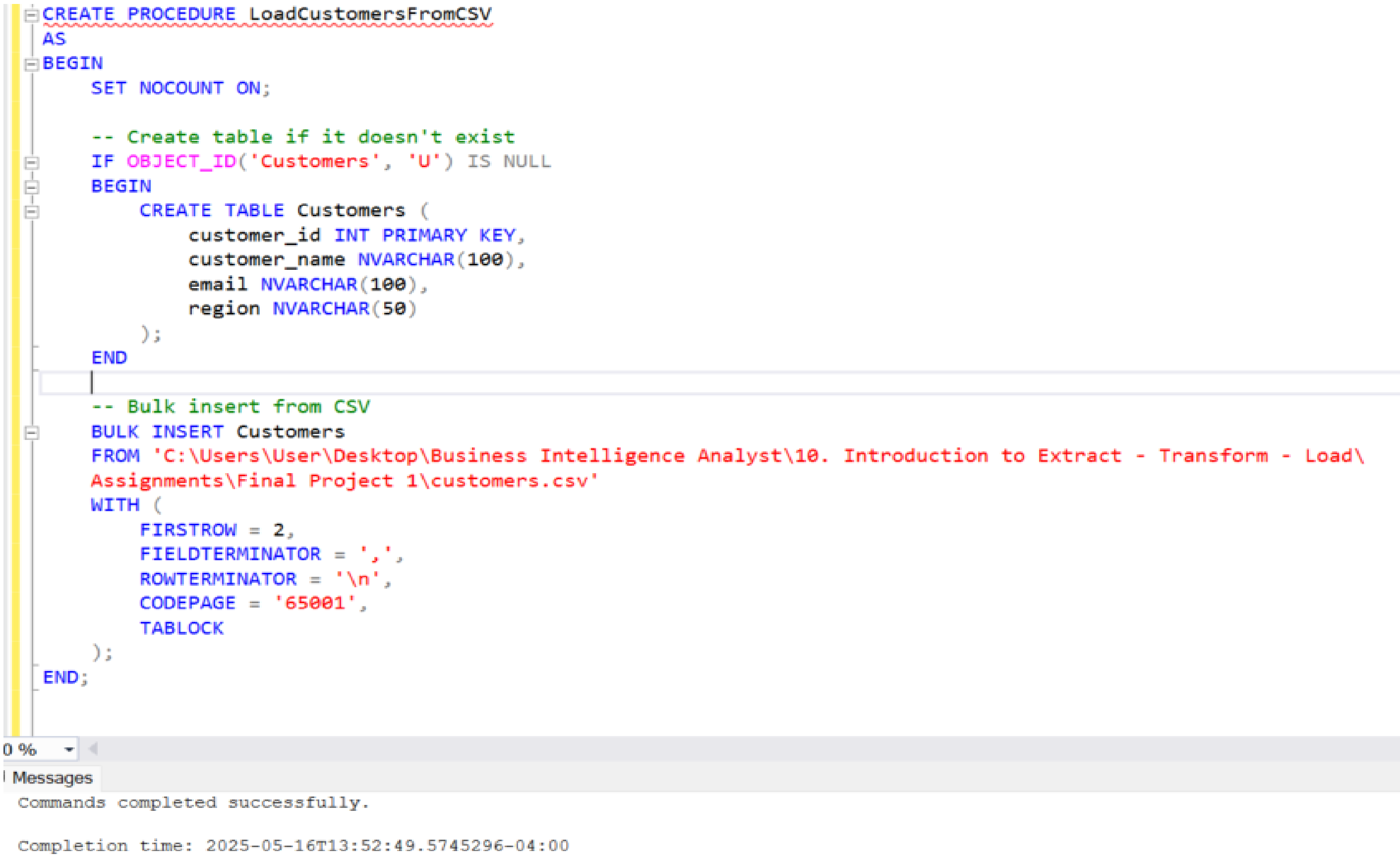
```sql
SELECT *
FROM CustomerProductReviews
WHERE
    customer_id IS NULL OR
    customer_name IS NULL OR
    email IS NULL OR
    region IS NULL OR
    review_id IS NULL OR
    rating IS NULL OR
    comments IS NULL OR
    review_date IS NULL OR
    product_id IS NULL OR
    product_name IS NULL OR
    category IS NULL OR
    price IS NULL;
```

customer_id	customer_name	email	region	review_id	rating	comments	review_date	product_id	product_name	category	price

3. **Create stored procedures and views:**

   - **Write stored procedures to automate data integration from raw files to final tables.**

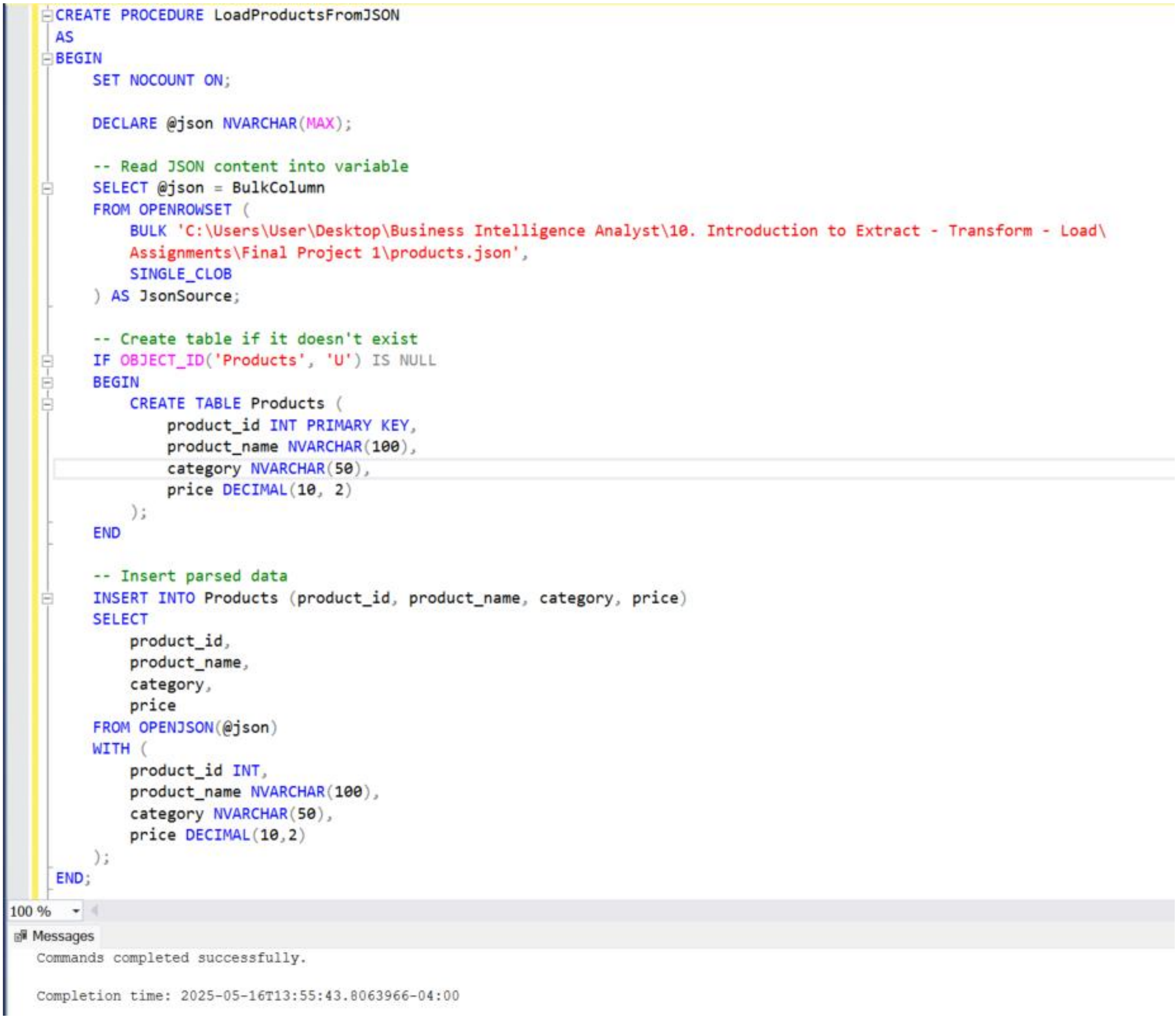
--Procedure for loading and inserting data from CSV file into a table:



```
CREATE PROCEDURE LoadCustomersFromCSV
AS
BEGIN
    SET NOCOUNT ON;

    -- Create table if it doesn't exist
    IF OBJECT_ID('Customers', 'U') IS NULL
    BEGIN
        CREATE TABLE Customers (
            customer_id INT PRIMARY KEY,
            customer_name NVARCHAR(100),
            email NVARCHAR(100),
            region NVARCHAR(50)
        );
    END

    -- Bulk insert from CSV
    BULK INSERT Customers
    FROM 'C:\Users\User\Desktop\Business Intelligence Analyst\10. Introduction to Extract - Transform - Load\
    Assignments\Final Project 1\customers.csv'
    WITH (
        FIRSTROW = 2,
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        CODEPAGE = '65001',
        TABLOCK
    );
END;
```

```
Commands completed successfully.

Completion time: 2025-05-16T13:52:49.5745296-04:00
```

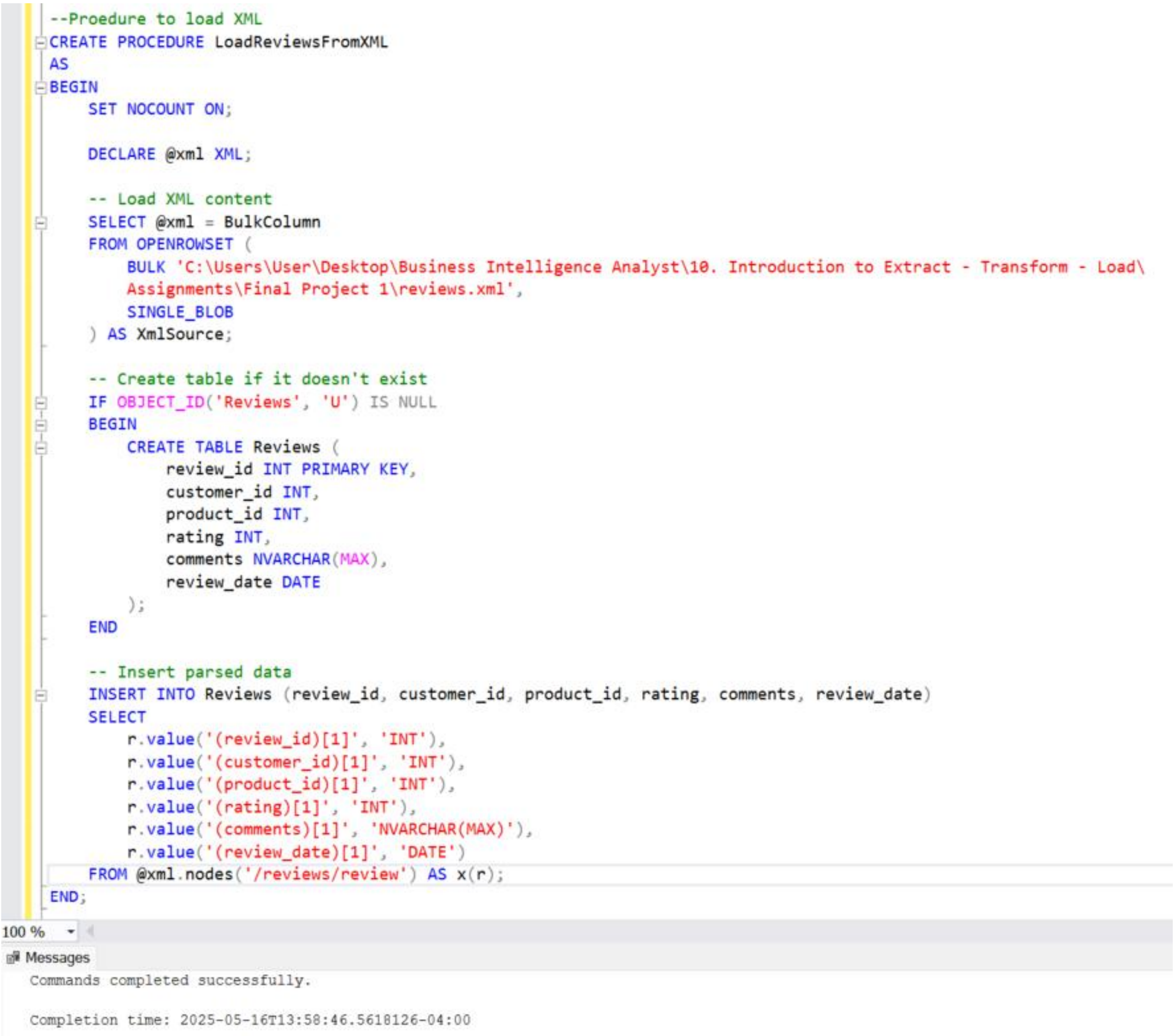
--Procedure for loading and inserting data from JSON file into a table:



```
CREATE PROCEDURE LoadProductsFromJSON
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @json NVARCHAR(MAX);

    -- Read JSON content into variable
    SELECT @json = BulkColumn
    FROM OPENROWSET (
        BULK 'C:\Users\User\Desktop\Business Intelligence Analyst\10. Introduction to Extract - Transform - Load\
        Assignments\Final Project 1\products.json',
        SINGLE_CLOB
    ) AS JsonSource;

    -- Create table if it doesn't exist
    IF OBJECT_ID('Products', 'U') IS NULL
    BEGIN
        CREATE TABLE Products (
            product_id INT PRIMARY KEY,
            product_name NVARCHAR(100),
            category NVARCHAR(50),
            price DECIMAL(10, 2)
        );
    END

    -- Insert parsed data
    INSERT INTO Products (product_id, product_name, category, price)
    SELECT
        product_id,
        product_name,
        category,
        price
    FROM OPENJSON(@json)
    WITH (
        product_id INT,
        product_name NVARCHAR(100),
        category NVARCHAR(50),
        price DECIMAL(10,2)
    );
END;
```

```
Commands completed successfully.

Completion time: 2025-05-16T13:55:43.8063966-04:00
```

--Procedure for loading and inserting data from XML file into a table:

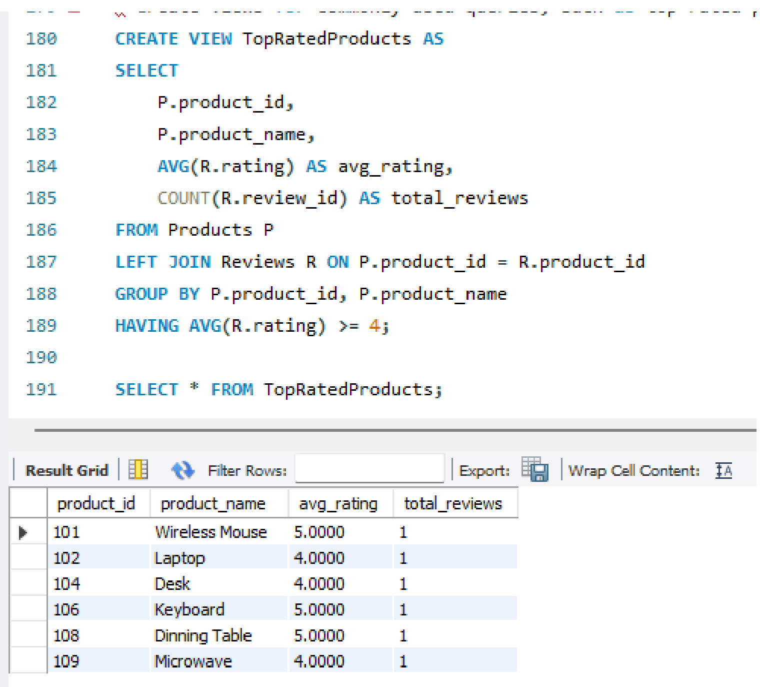


```
--Proedure to load XML
CREATE PROCEDURE LoadReviewsFromXML
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @xml XML;

    -- Load XML content
    SELECT @xml = BulkColumn
    FROM OPENROWSET (
        BULK 'C:\Users\User\Desktop\Business Intelligence Analyst\10. Introduction to Extract - Transform - Load\
        Assignments\Final Project 1\reviews.xml',
        SINGLE_BLOB
    ) AS XmlSource;

    -- Create table if it doesn't exist
    IF OBJECT_ID('Reviews', 'U') IS NULL
    BEGIN
        CREATE TABLE Reviews (
            review_id INT PRIMARY KEY,
            customer_id INT,
            product_id INT,
            rating INT,
            comments NVARCHAR(MAX),
            review_date DATE
        );
    END

    -- Insert parsed data
    INSERT INTO Reviews (review_id, customer_id, product_id, rating, comments, review_date)
    SELECT
        r.value('(review_id)[1]', 'INT'),
        r.value('(customer_id)[1]', 'INT'),
        r.value('(product_id)[1]', 'INT'),
        r.value('(rating)[1]', 'INT'),
        r.value('(comments)[1]', 'NVARCHAR(MAX)'),
        r.value('(review_date)[1]', 'DATE')
    FROM @xml.nodes('/reviews/review') AS x(r);
END;
```

```
Commands completed successfully.

Completion time: 2025-05-16T13:58:46.5618126-04:00
```

- **Create views for commonly used queries, such as top-rated products and flagged reviews.**

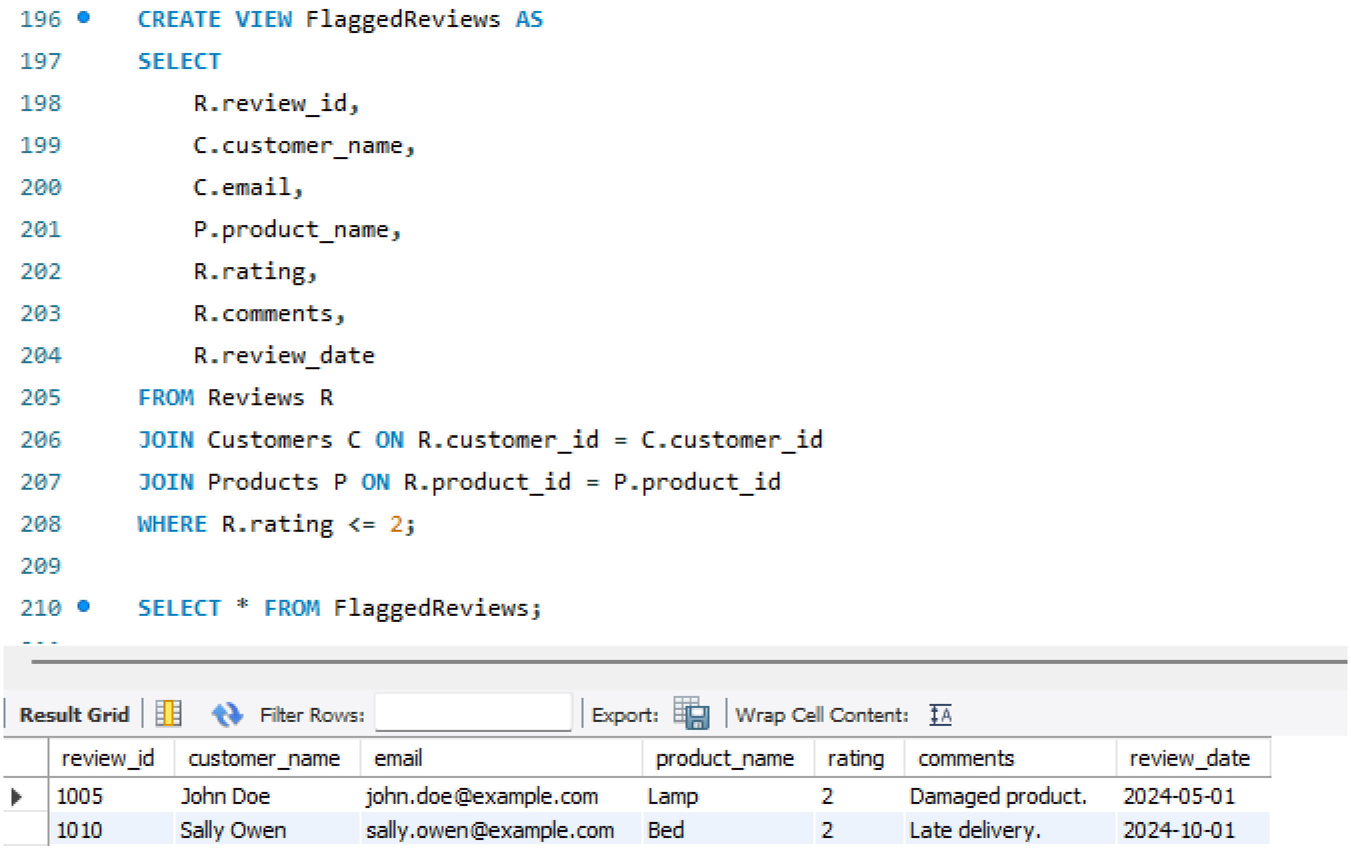
--Top rated products view:



```
CREATE VIEW TopRatedProducts AS
SELECT
    P.product_id,
    P.product_name,
    AVG(R.rating) AS avg_rating,
    COUNT(R.review_id) AS total_reviews
FROM Products P
LEFT JOIN Reviews R ON P.product_id = R.product_id
GROUP BY P.product_id, P.product_name
HAVING AVG(R.rating) >= 4;

SELECT * FROM TopRatedProducts;
```

product_id	product_name	avg_rating	total_reviews
101	Wireless Mouse	5.0000	1
102	Laptop	4.0000	1
104	Desk	4.0000	1
106	Keyboard	5.0000	1
108	Dinning Table	5.0000	1
109	Microwave	4.0000	1

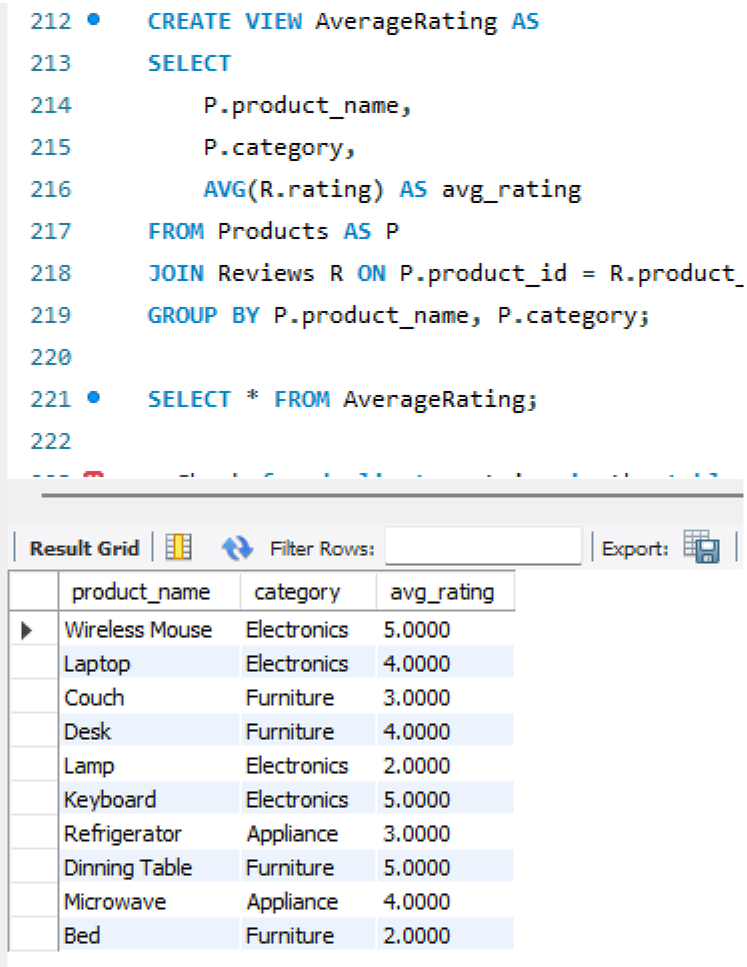
--Flagged reviews VIEW to show details of ratings less than or equal to 2:



```
CREATE VIEW FlaggedReviews AS
SELECT
    R.review_id,
    C.customer_name,
    C.email,
    P.product_name,
    R.rating,
    R.comments,
    R.review_date
FROM Reviews R
JOIN Customers C ON R.customer_id = C.customer_id
JOIN Products P ON R.product_id = P.product_id
WHERE R.rating <= 2;

SELECT * FROM FlaggedReviews;
```

review_id	customer_name	email	product_name	rating	comments	review_date
1005	John Doe	john.doe@example.com	Lamp	2	Damaged product.	2024-05-01
1010	Sally Owen	sally.owen@example.com	Bed	2	Late delivery.	2024-10-01

Average rating per product VIEW:



```
CREATE VIEW AverageRating AS
SELECT
    P.product_name,
    P.category,
    AVG(R.rating) AS avg_rating
FROM Products AS P
JOIN Reviews R ON P.product_id = R.product_
GROUP BY P.product_name, P.category;

SELECT * FROM AverageRating;

```

product_name	category	avg_rating
Wireless Mouse	Electronics	5.0000
Laptop	Electronics	4.0000
Couch	Furniture	3.0000
Desk	Furniture	4.0000
Lamp	Electronics	2.0000
Keyboard	Electronics	5.0000
Refrigerator	Appliance	3.0000
Dinning Table	Furniture	5.0000
Microwave	Appliance	4.0000
Bed	Furniture	2.0000

Part 4: Data Parsing

1. **Develop scripts to parse data files:**
   - ○ **CSV for survey data:**
     - ▪ **Use Python's csv module to read and validate data.**
     - ▪ **Check for missing or malformed entries (e.g., empty fields, invalid dates).**
     - ▪ **Save clean data into a staging table in the database using SQL Bulk Insert**.
   - ○ **JSON for web feedback:**
     - ▪ **Use Python's json module to parse nested structures.**
     - ▪ **Flatten data and extract fields (e.g., comments, ratings, timestamps).**
     - ▪ **Map JSON keys to database columns and load into the database using SQL scripts.**
   - ○ **XML for external reviews:**

- **Use Python's xml.etree.ElementTree library to parse XML structures.**
- **Validate schema conformity and extract relevant fields.**
- **Convert XML data to rows and load them into relational tables.**

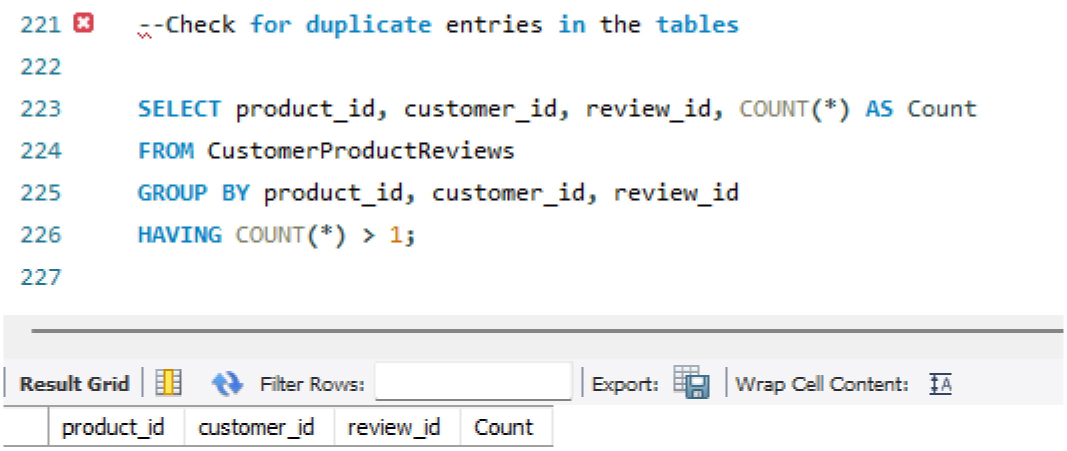
DONE in Part 3 using SQL Server (My answers to Part 3 & Part 4 are kind of combined together- validating & cleaning data is show in Part 4 below)

2. **Validate parsed data:**

    - **Check for duplicate entries using SQL SELECT DISTINCT queries.**

Using the joined data of the 3 tables in ‘CustomerProductReviews’ table.

--Checking for duplicates:

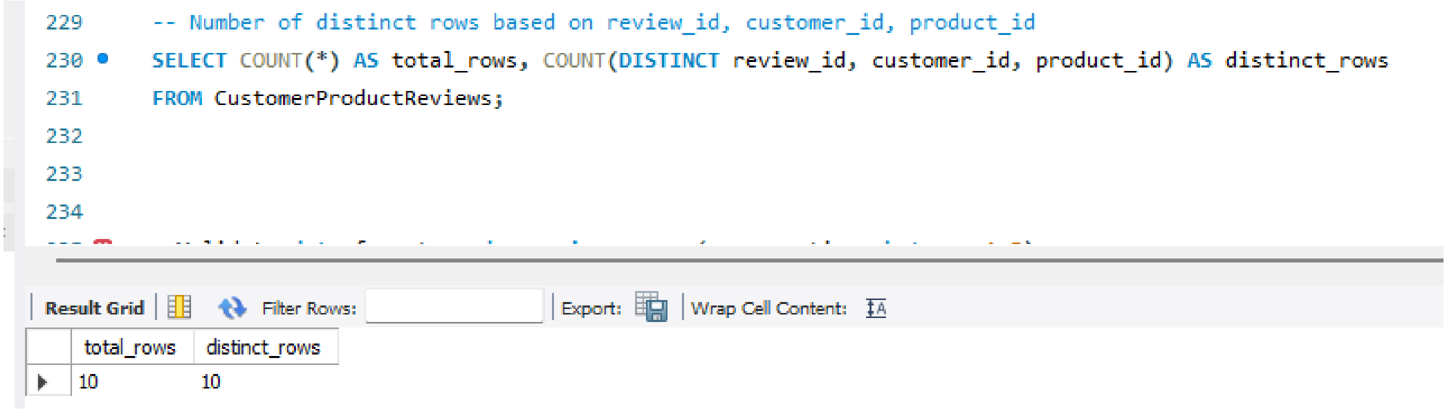


```
--Check for duplicate entries in the tables

SELECT product_id, customer_id, review_id, COUNT(*) AS Count
FROM CustomerProductReviews
GROUP BY product_id, customer_id, review_id
HAVING COUNT(*) > 1;

```

product_id	customer_id	review_id	Count

--Checking for duplicates using DISTINCT:

Since total rows are = to the distinct count of rows, then we don’t have duplicates.



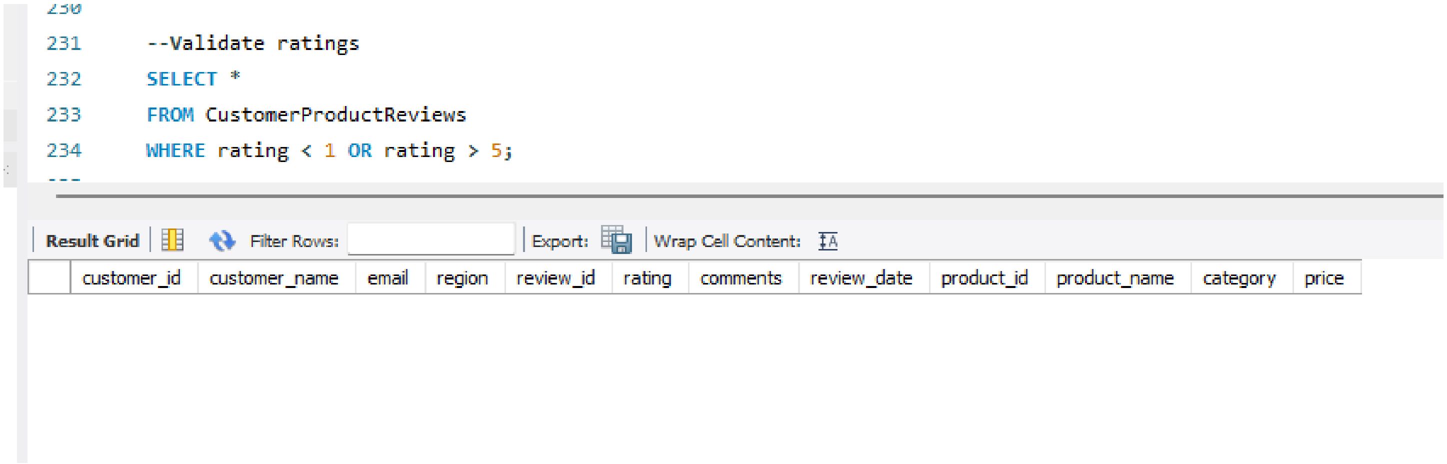
```
-- Number of distinct rows based on review_id, customer_id, product_id
SELECT COUNT(*) AS total_rows, COUNT(DISTINCT review_id, customer_id, product_id) AS distinct_rows
FROM CustomerProductReviews;
```

total_rows	distinct_rows
10	10

- **Validate date formats and numeric ranges (e.g., ratings between 1–5).**

--Validating ratings are between 1 & 5:

All ratings are within the specified range

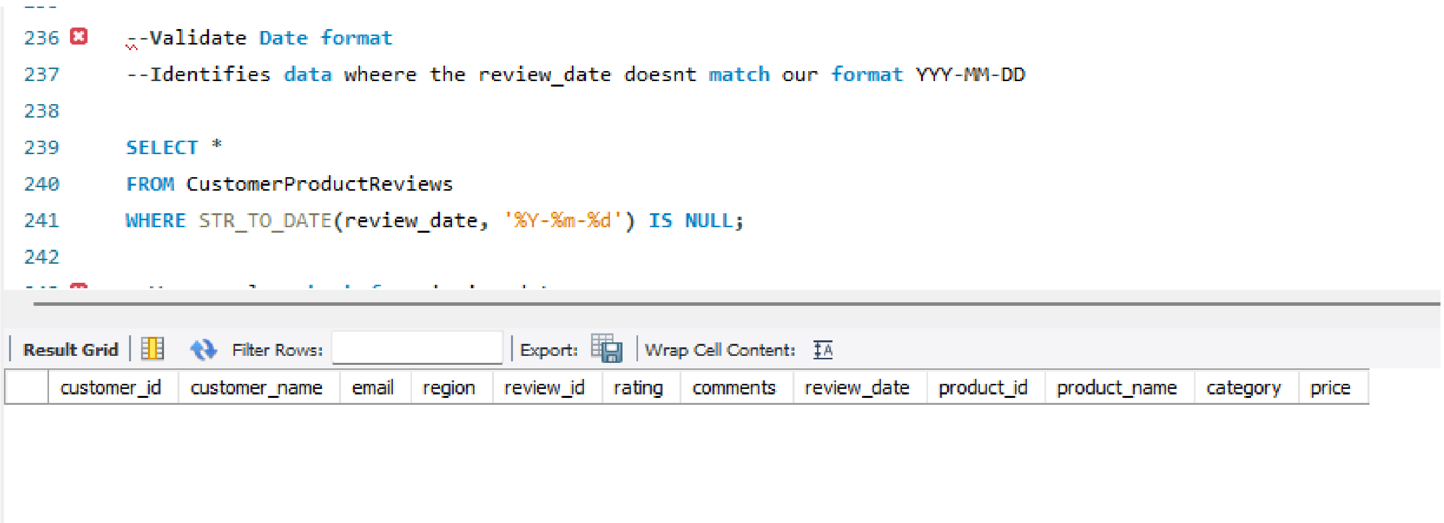


```
--Validate ratings
SELECT *
FROM CustomerProductReviews
WHERE rating < 1 OR rating > 5;
```

	customer_id	customer_name	email	region	review_id	rating	comments	review_date	product_id	product_name	category	price

--Validating data format:

All date entries conform to the same date format YYYY-MM-DD



```
--Validate Date format
--Identifies data wheere the review_date doesnt match our format YYY-MM-DD

SELECT *
FROM CustomerProductReviews
WHERE STR_TO_DATE(review_date, '%Y-%m-%d') IS NULL;
```

	customer_id	customer_name	email	region	review_id	rating	comments	review_date	product_id	product_name	category	price

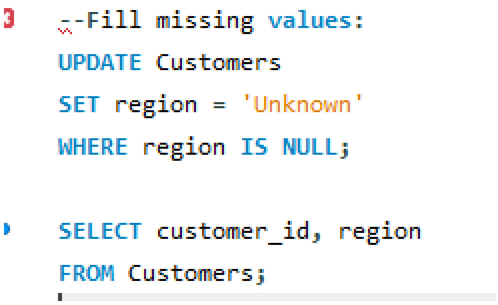
- **Log parsing errors for manual review, providing row numbers and error details**

3. **Load parsed data into database tables:**

   ○ **Use SQL scripts to insert validated data into target tables.**

My data doesn't have duplicates, missing values or incorrect formats. However, if we had to update the data with the validated data we could use UPDATE command.

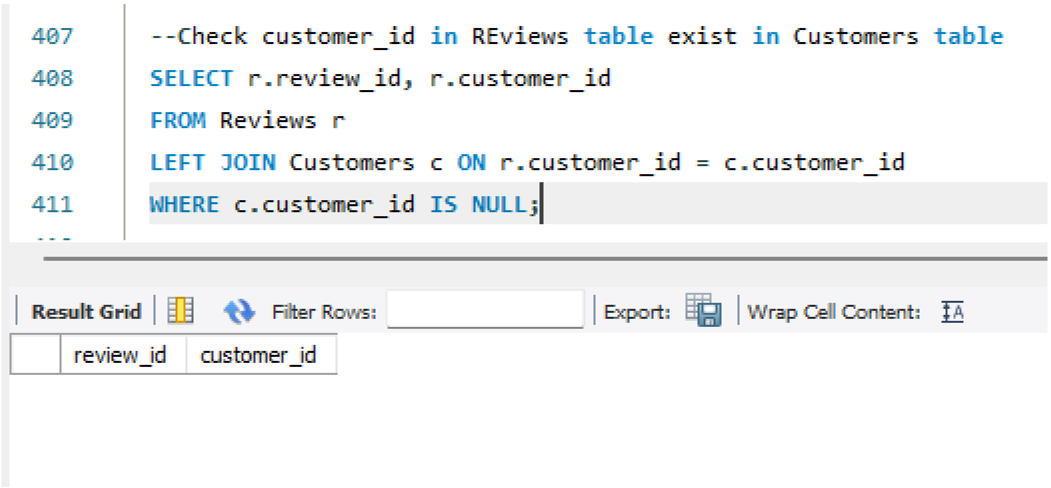
For example, if we were to fill out the missing data from region column:



```
--Fill missing values:
UPDATE Customers
SET region = 'Unknown'
WHERE region IS NULL;

SELECT customer_id, region
FROM Customers;
```

   ○ **Ensure referential integrity by checking foreign key constraints.**

All customer_id's in the Review's table should be in the Customer's table

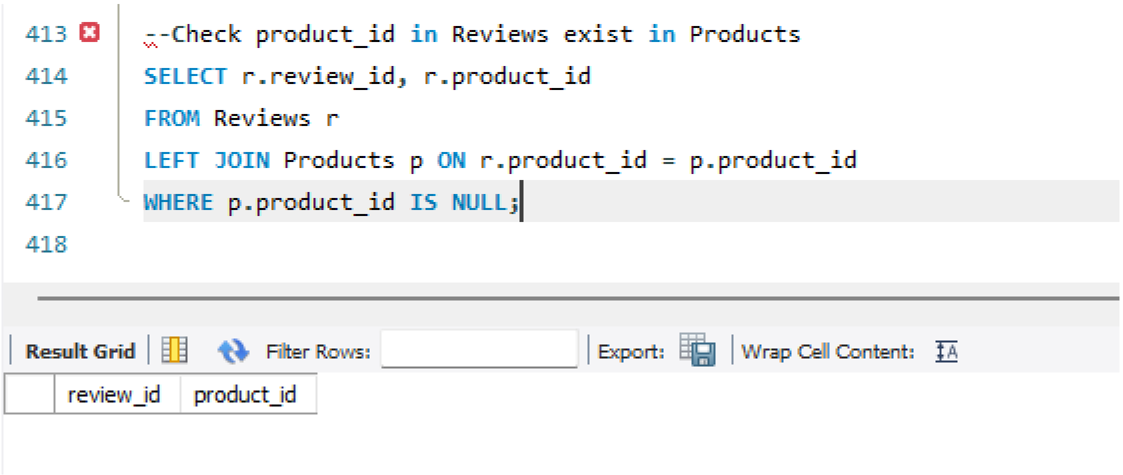


```
--Check customer_id in REviews table exist in Customers table
SELECT r.review_id, r.customer_id
FROM Reviews r
LEFT JOIN Customers c ON r.customer_id = c.customer_id
WHERE c.customer_id IS NULL;
```

	review_id	customer_id

All product_id's in the Review's table should be in the Product's table

Since the results are both empty, then referential integrity is confirmed.



```
--Check product_id in Reviews exist in Products
SELECT r.review_id, r.product_id
FROM Reviews r
LEFT JOIN Products p ON r.product_id = p.product_id
WHERE p.product_id IS NULL;
```

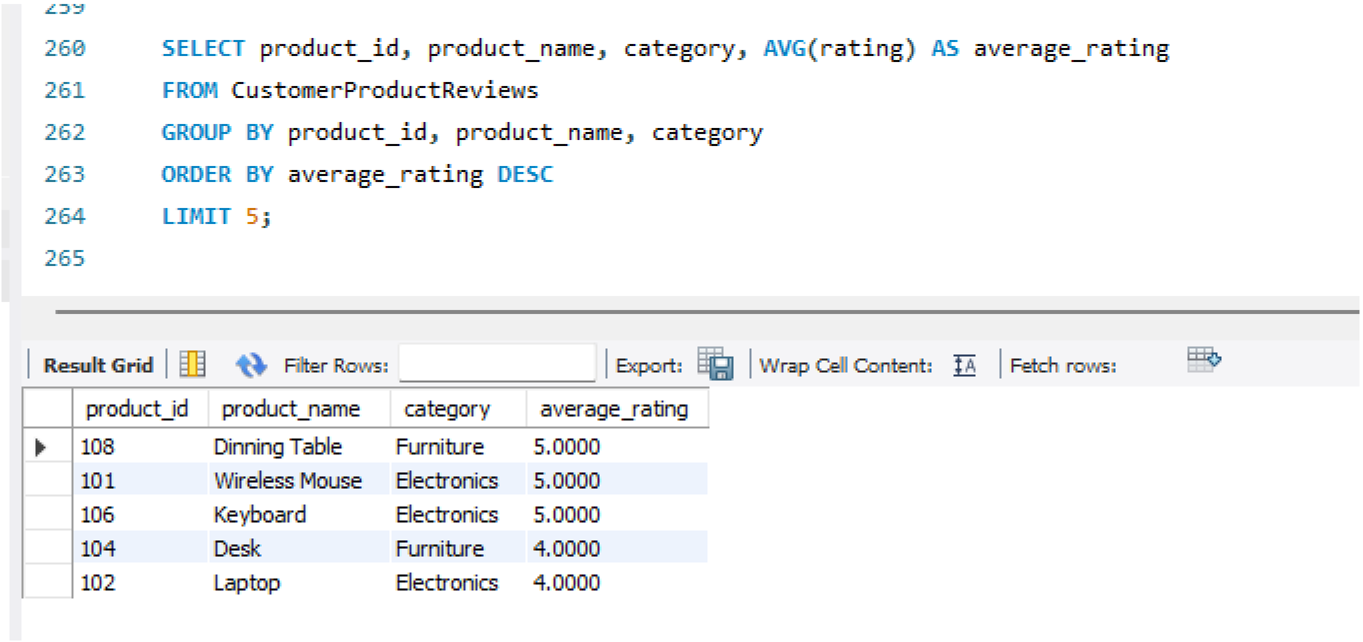
	review_id	product_id

- Generate logs indicating successful imports and rejected records for troubleshooting.

Part 5: Analysis and Presentation

1. **Write SQL queries to extract insights:**
2. **Top-rated products:**
   - **Calculate the average rating for each product using GROUP BY.**
   - **Identify the top 5 products with the highest average ratings and display their names, categories, and ratings.**

The top-rated products were dining table, mouse, keyboard, desk & laptop.

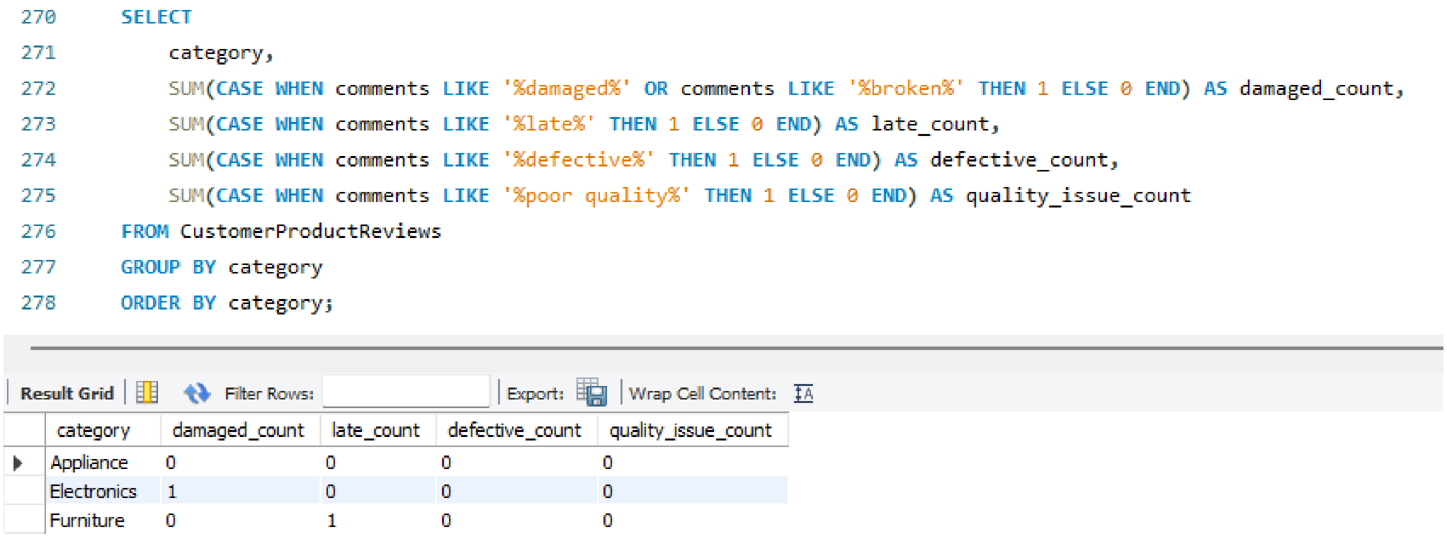


```sql
SELECT product_id, product_name, category, AVG(rating) AS average_rating
FROM CustomerProductReviews
GROUP BY product_id, product_name, category
ORDER BY average_rating DESC
LIMIT 5;
```

product_id	product_name	category	average_rating
108	Dinning Table	Furniture	5.0000
101	Wireless Mouse	Electronics	5.0000
106	Keyboard	Electronics	5.0000
104	Desk	Furniture	4.0000
102	Laptop	Electronics	4.0000

3. **Common complaints (using keyword searches in comments):**
   - **Search for common keywords like 'damaged', 'late', 'defective', etc., in comments.**
   - **Count occurrences of each keyword and group by product categories.**

Most common complaints were seen in the Electronics & Furniture categories.



```sql
SELECT
    category,
    SUM(CASE WHEN comments LIKE '%damaged%' OR comments LIKE '%broken%' THEN 1 ELSE 0 END) AS damaged_count,
    SUM(CASE WHEN comments LIKE '%late%' THEN 1 ELSE 0 END) AS late_count,
    SUM(CASE WHEN comments LIKE '%defective%' THEN 1 ELSE 0 END) AS defective_count,
    SUM(CASE WHEN comments LIKE '%poor quality%' THEN 1 ELSE 0 END) AS quality_issue_count
FROM CustomerProductReviews
GROUP BY category
ORDER BY category;
```

category	damaged_count	late_count	defective_count	quality_issue_count
Appliance	0	0	0	0
Electronics	1	0	0	0
Furniture	0	1	0	0

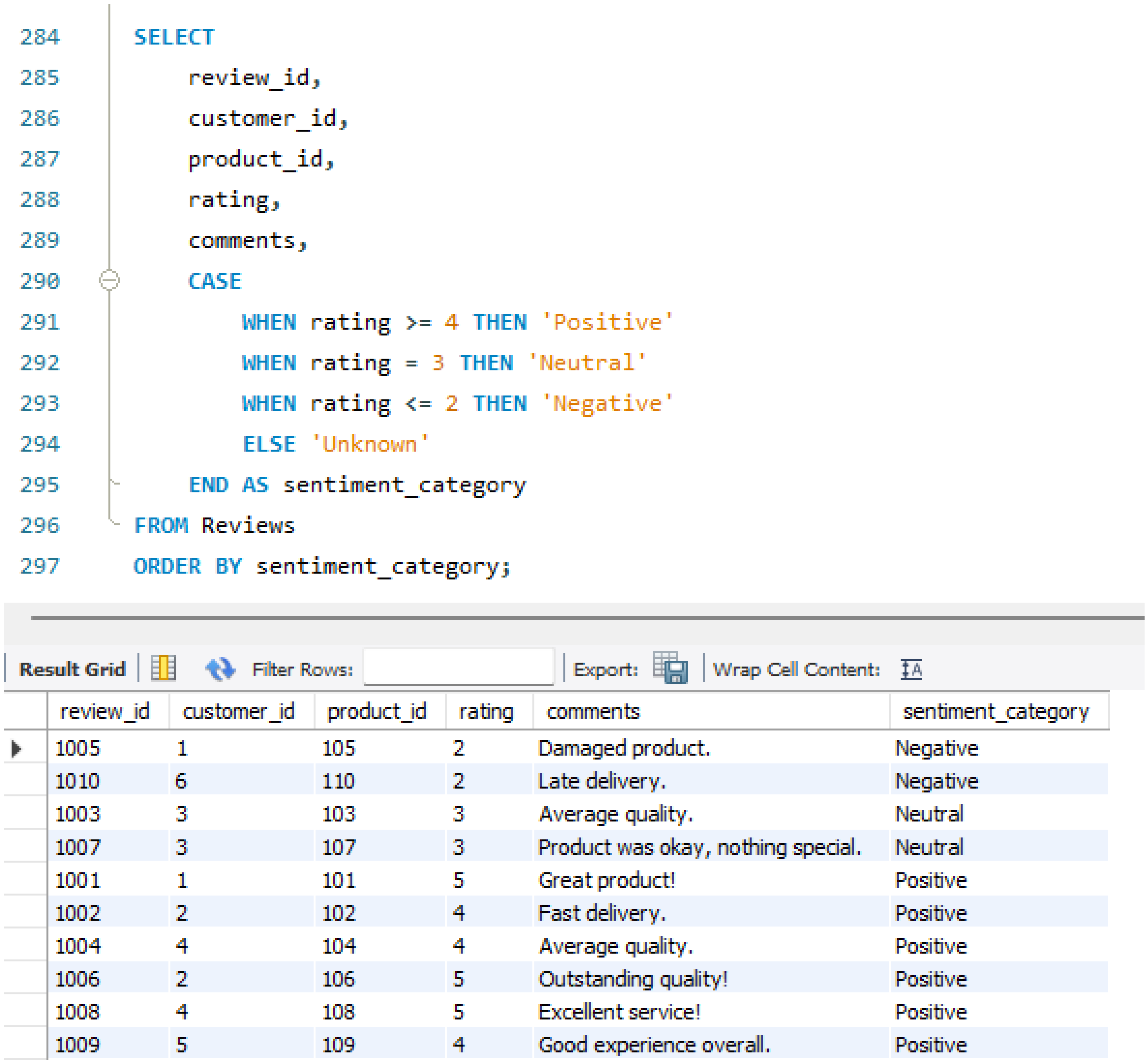
4. **Customer sentiment analysis based on ratings and comments:**

- **Classify reviews into categories such as Positive, Neutral, and Negative based on ratings.**
- **Use CASE statements to group ratings into sentiment categories.**

Ratings > than or = 4 are categorized as Positive

Ratings = 3 are categorized as Neutral

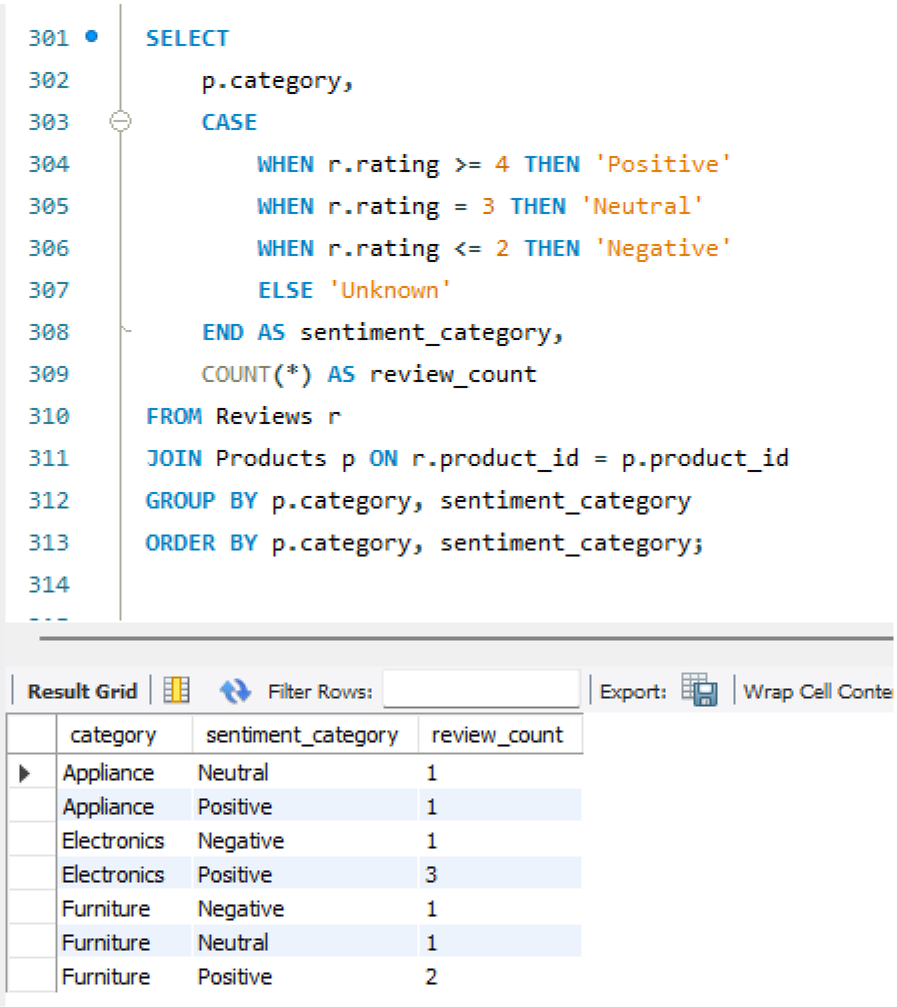
Ratings < than or = 2 are categorized as Negative



```
SELECT
    review_id,
    customer_id,
    product_id,
    rating,
    comments,
    CASE
        WHEN rating >= 4 THEN 'Positive'
        WHEN rating = 3 THEN 'Neutral'
        WHEN rating <= 2 THEN 'Negative'
        ELSE 'Unknown'
    END AS sentiment_category
FROM Reviews
ORDER BY sentiment_category;
```

review_id	customer_id	product_id	rating	comments	sentiment_category
1005	1	105	2	Damaged product.	Negative
1010	6	110	2	Late delivery.	Negative
1003	3	103	3	Average quality.	Neutral
1007	3	107	3	Product was okay, nothing special.	Neutral
1001	1	101	5	Great product!	Positive
1002	2	102	4	Fast delivery.	Positive
1004	4	104	4	Average quality.	Positive
1006	2	106	5	Outstanding quality!	Positive
1008	4	108	5	Excellent service!	Positive
1009	5	109	4	Good experience overall.	Positive

Sentiment view by product category:



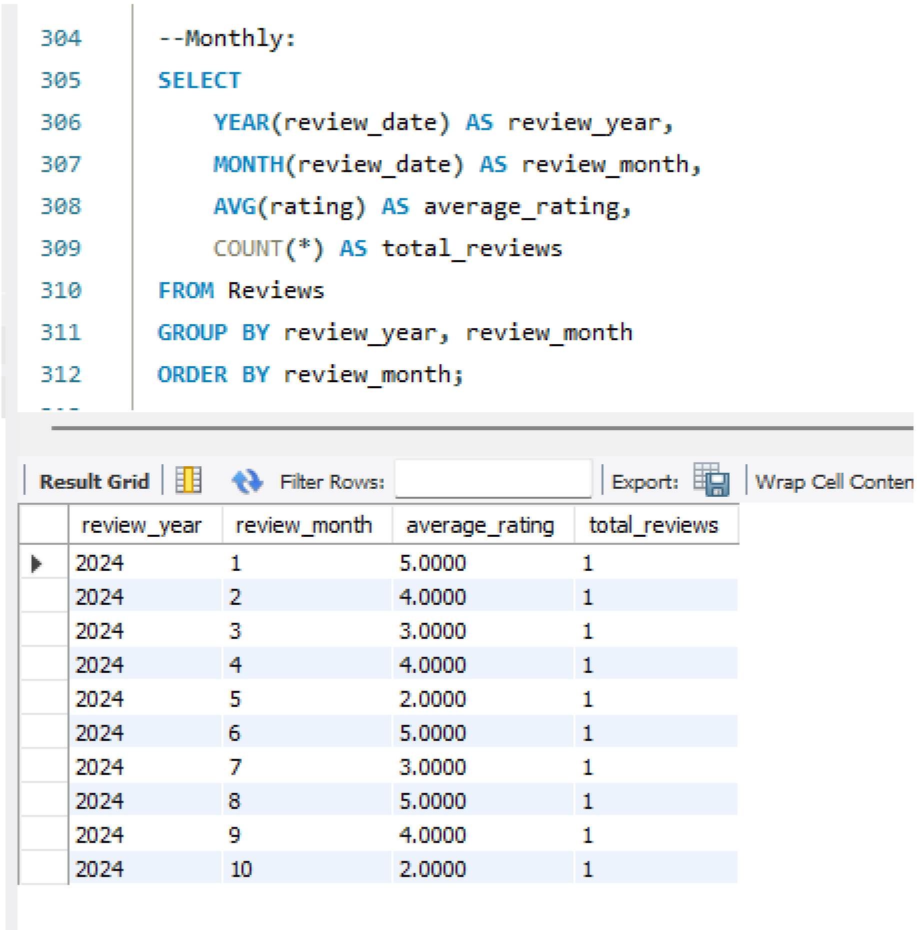
```sql
SELECT
    p.category,
    CASE
        WHEN r.rating >= 4 THEN 'Positive'
        WHEN r.rating = 3 THEN 'Neutral'
        WHEN r.rating <= 2 THEN 'Negative'
        ELSE 'Unknown'
    END AS sentiment_category,
    COUNT(*) AS review_count
FROM Reviews r
JOIN Products p ON r.product_id = p.product_id
GROUP BY p.category, sentiment_category
ORDER BY p.category, sentiment_category;
```

category	sentiment_category	review_count
Appliance	Neutral	1
Appliance	Positive	1
Electronics	Negative	1
Electronics	Positive	3
Furniture	Negative	1
Furniture	Neutral	1
Furniture	Positive	2

5. **Generate summary reports showing:**

- **Trends over time in ratings and reviews:**
    - **Use date fields to group data by weeks or months.**
    - **Track changes in average ratings and review counts.**

There is no visible trend across the months, the average rating fluctuates up & down every month.

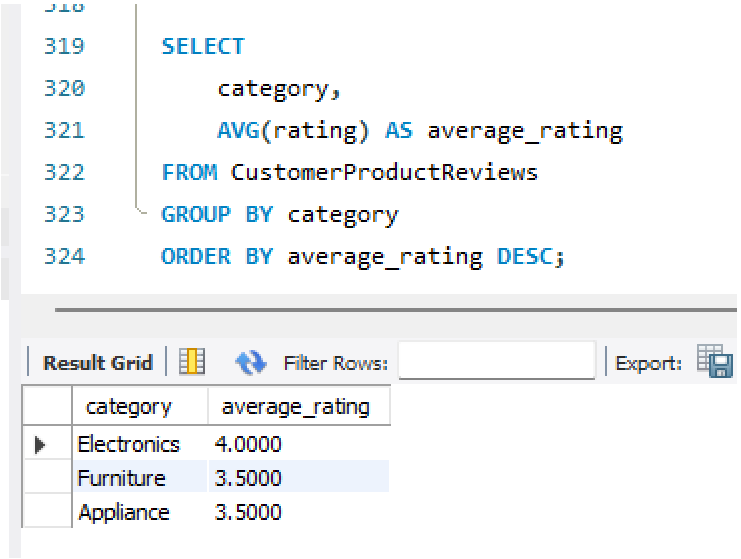


```
--Monthly:
SELECT
    YEAR(review_date) AS review_year,
    MONTH(review_date) AS review_month,
    AVG(rating) AS average_rating,
    COUNT(*) AS total_reviews
FROM Reviews
GROUP BY review_year, review_month
ORDER BY review_month;
```

review_year	review_month	average_rating	total_reviews
2024	1	5.0000	1
2024	2	4.0000	1
2024	3	3.0000	1
2024	4	4.0000	1
2024	5	2.0000	1
2024	6	5.0000	1
2024	7	3.0000	1
2024	8	5.0000	1
2024	9	4.0000	1
2024	10	2.0000	1

- **Product categories with the highest satisfaction scores:**
  - **Aggregate ratings by category and find averages.**
  - **Highlight categories with highest scores.**

The electronics category has the highest rating among all the categories.



```sql
SELECT
    category,
    AVG(rating) AS average_rating
FROM CustomerProductReviews
GROUP BY category
ORDER BY average_rating DESC;
```

category	average_rating
Electronics	4.0000
Furniture	3.5000
Appliance	3.5000