PROJECT TECHNICAL REPORT

Professor: Gasan Elkhodari

Class Code: BUAN 6320 SP23

Group: 1

Authors:

1) Anubha Jain
2) Divya Srishti Chittoor Muppala
3) Hrudya Raviprolu
4) Raghu Veera Gudapati
5) Salman Muhammad
6) Veda Swaroop Dasagrandhi

NAVEEN JINDAL SCHOOL OF MANAGEMENT

THE UNIVERSITY OF TEXAS AT DALLAS

800 W Campbell Rd, Richardson,
TX 75080

Introduction to the Database

This Database Design Document narrates the outline and implementation of a database created to store user information of the business working of the E-commerce wing of the Multi-national Technological Company, Amazon, Inc. Amazon which is one of the first and the largest E-commerce retailer has a User-friendly Interface application that provides a platform for various Sale transactions between the Buyers, who are mostly common public, and Small to Big businesses, that are looking to endorse their products and make sales. This Database aims to showcase the Sophisticated Cobweb of a network that the business has established to make a successful

Overview of Amazon's E-commerce

The Amazon Market places deal with everything from the Online store to the logistics that get the deliveries to the customers. The products on their website include a large plethora of products from everyday groceries and necessities up to Furniture, Clothing, and Electronics, contributing hugely to Amazon's total revenue. The eCommerce market was valued at $16.6 trillion in 2022.

There is a huge base of Customers, around 200 million of them having a Membership called 'The Amazon Prime Membership'. Amazon customers buy roughly 7,400 products per minute from U.S. sellers. They also have a huge customer support base dealing with any issues reported by the customers, refunds, and money-backs, etc.

The Sellers range from International to Domestic, Large to Small, dealing and endorsing all kinds of products. The products are categorized by Category, Customer reviews and ratings, Price range, Amazon sponsored, etc. The details about the sellers are verified and revealed to the customer. These include Seller rating, Order Cancellations, Guarantee terms, etc. Amazon is one of the largest seller platforms in the world, with 6.3 million total sellers and an incredible 1.5 million active sellers.

The interface also includes features and stores information such as Customer/User information including Personal Shopping Carts, Membership and benefits Details, Subscriptions, Personal Digital Content and Devices, Card Details, Addresses, History of orders, refunds, and payments, Payment type and details, Payment transaction history, etc.

Design Layout

Key Factors Influencing the Design of the Database:

The main and base design of the database revolves around the traditional Buyer, Mediator, and Customer relationship. The Buyer or Customer information, its relation to the Amazon (Mediator) who takes the Purchase Orders and then processes it to the Seller who possesses the Product is clearly depicted in the Database. Then there are the transactions that take place between these three entities, the payment transactions, and since Amazon is involved in logistics and deliveries, the database also processes the data related to Shipping of these products.

Accordingly, the independent entities taken into account are Customer, Product, Shipping, and Payment. All these entities are linked through the Purchase Order that is generated when a sale transaction has been made. The flow chart starts when the customer places an order for a particular product and simultaneously the full payment of the purchase order is made. After the purchase, the products as a whole package under one purchase order get shipped to the customer's given address.

Assumptions:

The Customer places the Order such that all the Products under a single Purchase Order are to be Shipped to a single destination.

Consequently, a single Purchase Order placed by one Customer will be shipped through a single Shipment. And different Shipment orders are allotted to each Purchase Order.

Statement of Purpose

Purpose and Objective:

The database language will allow the user to easily retrieve specific information about the business using queries based on various criterion. This database should be able to provide a structured format for storing and organizing data, by defining structure, relationships, and constraints of the data, all while ensuring data integrity and consistency. Through the queries, it should be able to imitate the working of the Retail E-commerce operations of the Company. The user should be able to retrieve data regarding Customer details, information regarding the payments made, products included in a single purchase order, and the shipping details, all through their relation to the purchase order through the Order ID and the Customer ID. Insertion and Alteration of the data should be easier due to the use of Trigger in the DDL.

Project Scope:

The scope of the project is restricted to the outline and implementation of the retail business of Amazon. This involves documenting the database model and its design, establishing entities and relationships by formulating an Entity-Relationship model, creating and modifying the structure of the several objects of the database using predefined commands and a specific syntax which is the use of Data Definition Language, and finally enabling the option to manage and modify data in the database in response to various possibilities and situations through Data Manipulation Language and Structured Query Language scripts.

In-Scope Work

- Project requirements documentation
- Entity-relationship model
- DDL Scripts
- Example DML Scripts
- Example SQL Scripts
- Comprehensive Report

Database Goals and Expectations:

At the completion of this project, the database is expected to contain fields and respective keys of all attributes to the business. Should successfully follow standards and conventions, display data integrity, with minimal or no data redundancies.

Diagram Tool

ER-Assistant Version 2.10, running on Windows 11

Office Productivity Tools

Microsoft Office 365 for Enterprise, Google Drive, Google Suite (Docs, Presentation) Microsoft Word Enterprise, running on Windows 11

General Format

- Use consistent and descriptive identifiers and names.
- Use white space and indentation to make code easier to read.

- Store time and date formation in ISO-8601 format (YYYY-MM-DD/HH:MM:SS.SSSSS).
- Avoid redundant SQL, such as unnecessary quoting or parentheses or WHERE clauses that can be derived.
- Use C-style comments with opening /* and closing /* digraphs whenever possible; otherwise, precede comments with -- and finish them with a new line.
- For the sake of quick readability, prefer snake_case over CamelCase.
- Avoid Hungarian notation and other descriptive prefixes.
- Favor collective nouns over plurals, such as using staff instead of employees.
- When using quoted identifiers, use SQL92 double quotes to preserve portability.
- Avoid applying object-oriented design principles to SQL or database structures.

Naming Conventions

- Names must begin with a letter and may not end with an underscore.
- Ensure that all names are unique and do not conflict with reserved keywords.
- Keep name length to 30 bytes; this usually means 30 characters, unless the name uses a multi-byte character set.
- Names may contain only letters, numbers, and underscores.
- Multiple consecutive underscores are not allowed.
- Use underscores to represent spaces in names, e.g. "first name" becomes first_name.
- Avoid abbreviations; if it is necessary to use them, ensure they are commonly known and understood.
- Prefer collective nouns for table names.
- Tables and columns should never share the same name.
- Avoid concatenating the names of two tables when naming their relationship table.
- When naming columns, always prefer singular nouns.
- Avoid the name id for primary keys
- Use lowercase in column names whenever reasonable.
- Use commonly known suffixes to indicate the purpose of a column: _id, _name, _size, _addr, etc.

Requirement Definition Document

Business Rules:

- A CUSTOMER may place zero to many PURCHASE_ORDER
- A PURCHASE_ORDER can be placed by only one CUSTOMER
- A PURCHASE_ORDER can be paid by only one PAYMENT
- A PAYMENT can only pay one PURCHASE_ORDER
- A single PURCHASE_ORDER contains one-to-many PRODUCTS
- A PRODUCT can be contained in zero or one PURCHASE_ORDER
- A PURCHASE_ORDER is shipped by one SHIPPING
- A SHIPPING order can be placed for a single PURCHASE_ORDER

Entity and Attribute Description

Entities:

Entity Name: **CUSTOMER**

Entity Description: The primary end-user of the software.

Main Attributes of CUSTOMER:

Customer_ID: (Primary Key) A unique identifier for the customer's ID

First_Name: A character record of the customer's first name

Last_Name: A character record of the customer's last name

Email: A character record of the customer's email address

Phone_Number: A numeric record of the customer's phone number

Address: A character record of the customer's address

Zipcode: A numeric record of the customer address' zip code

State: A character record of the customer address' state

City: A character record of the customer address' city

Entity Name: **PAYMENT**

Entity Description: Transaction made by the customer for making a Purchase Order

Main Attributes of PAYMENT:

Payment_ID: (Primary Key) A unique identifier for the Payment ID

Payment_Date: A date record of the day of payment

Payment_Amount: A numeric record of the Payment made

Order_id_fk: (Foreign Key) An identifier for the Purchase Order ID

Payment_Status: A character record of the status of the payment

Payment_Method: A character record of the mode of payment

Entity Name: **PRODUCT**

Entity Description: Content of the Purchase order for which the Customer pays

Main Attributes of PRODUCT:

Product_ID: (Primary Key) A unique identifier for the Product ID

Name: A character record of the product’s name

Price: A numeric record of the product’s price

Description: A character record of the product’s details

Category: A character record of the product’s type

Order_id_fk: (Foreign Key) An identifier for the Purchase Order ID

Entity Name: **SHIPPING**

Entity Description: Means of transporting the Purchase order to the Customer’s provided address

Main Attributes of SHIPPING:

Order_Id_fk: (Primary Key) An identifier for the Purchase Order ID

Shipping_address: A character record of the shipping address

Shipping_cost: A numeric record of the transit cost

Shipping_Date: A date record of the day of the shipment

Shipping_Id: (Foreign Key) A unique identifier for the shipment ID

Shipping_type: A character record of the shipment type

Tracking_Number: A numeric record for the tracking of shipment

Entity Name: **PURCHASE ORDER**

Entity Description: Order placed by the customer for a single or many products at once.

Main Attributes of SHIPPING: PURCHASE ORDER

Customer_id_fk: (Foreign Key) An identifier for the Customer ID

Delivery_date: A date record of the day of the final Delivery

Order_Date: A date record of the day of the purchase order was placed

Order_id: (Primary Key) An identifier for the Purchase Order ID

Order_status: A character record of the status of the order placement

Total_price: A numeric record of the total amount of all the products in the Order

Relationship and Cardinality Description:

Relationship "places" between **CUSTOMER and PURCHASE_ORDER**.

Cardinality: 1:M between CUSTOMER and PURCHASE_ORDER

Business Rule: A customer can place zero, or many orders; an order can be placed by only one customer.

Relationship "paid by" between **PURCHASE_ORDER and PAYMENT**

Cardinality: 1:1 between PURCHASE_ORDER and PAYMENT

Business Rule: An order uses one payment; a payment belongs to only one order.

Relationship "contains" between **PURCHASE_ORDER and PRODUCT**

Cardinality: 1:M between PURCHASE_ORDER and PRODUCT

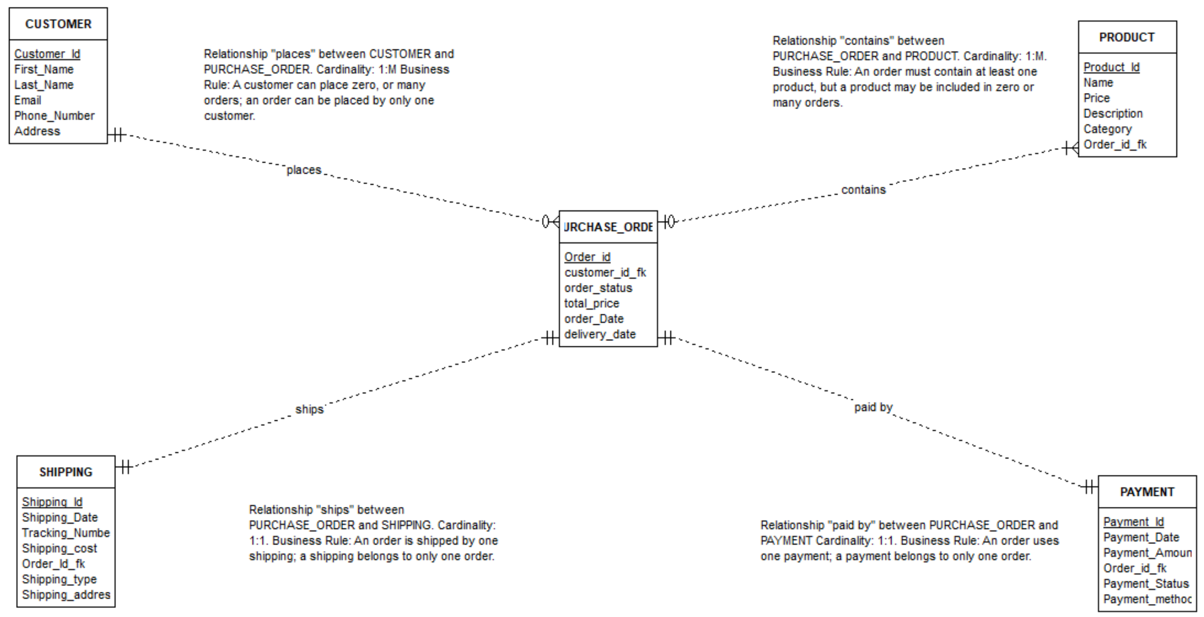
Business Rule: An order must contain at least one product, but a product may be included in zero or many orders.

Relationship "ships" between **PURCHASE_ORDER and SHIPPING**

Cardinality: 1:1 between PURCHASE_ORDER and SHIPPING

Business Rule: An order is shipped by one shipping; a shipping belongs to only one order.

Entity – Relationship Diagram



Relationship "places" between CUSTOMER and PURCHASE_ORDER. Cardinality: 1:M Business Rule: A customer can place zero, or many orders; an order can be placed by only one customer.

Relationship "contains" between PURCHASE_ORDER and PRODUCT. Cardinality: 1:M. Business Rule: An order must contain at least one product, but a product may be included in zero or many orders.

Relationship "ships" between PURCHASE_ORDER and SHIPPING. Cardinality: 1:1. Business Rule: An order is shipped by one shipping; a shipping belongs to only one order.

Relationship "paid by" between PURCHASE_ORDER and PAYMENT Cardinality: 1:1. Business Rule: An order uses one payment; a payment belongs to only one order.

DDL Source Code

```
/*
 Project BAUN 6320 - DDL - UTD Group Project S23 1
*/


/* DROP statements to clean up objects from previous run */
-- Triggers
DROP TRIGGER TRG_PAYMENT;
DROP TRIGGER TRG_SHIPPING;
DROP TRIGGER TRG_PURCHASE_ORDER;
DROP TRIGGER TRG_Product;
DROP TRIGGER TRG_Customer;

--Sequences
DROP SEQUENCE SEQ_Customer_customer_id;
DROP SEQUENCE SEQ_Product_Product_id;
DROP SEQUENCE SEQ_Shipping_shipping_id;

--VIEWS
DROP VIEW CustomerInfo;
DROP VIEW OrderInfo;
DROP VIEW OrderOver50;
DROP VIEW ShippingInfo;
```

```
--Indices
DROP INDEX IDX_Payment_Payment_Status;
DROP INDEX IDX_Payment_order_id_FK;
DROP INDEX IDX_Shipping_Tracking_No;
DROP INDEX IDX_Shipping_order_id_FK;
DROP INDEX IDX_Product_order_id_FK;
DROP INDEX IDX_Product_Name;
DROP INDEX IDX_Purchase_Order_customer_id_FK;
DROP INDEX IDX_Customer_First_Name;

/* Drop table */

DROP TABLE Payment;
DROP TABLE Shipping;
DROP TABLE Product;
DROP TABLE Purchase_order;
DROP TABLE Customer;

/* Create tables based on entities */

CREATE TABLE customer (
    customer_id  VARCHAR(20)    NOT NULL,
    first_name   VARCHAR(30)    NOT NULL,
    last_name    VARCHAR(30)    NOT NULL,
    email        VARCHAR(50)    NOT NULL,
    phone_number VARCHAR(15)    NOT NULL,
    address      VARCHAR(50)    NOT NULL,
    city         VARCHAR(20)    NOT NULL,
    state        VARCHAR(20)    NOT NULL,
    zipcode      INTEGER        NOT NULL,
    CONSTRAINT pk_customer PRIMARY KEY ( customer_id )
);

CREATE TABLE Purchase_Order (
    order_id        VARCHAR(20)  NOT NULL,
    customer_id     VARCHAR(20)  NOT NULL,
    order_status    VARCHAR(20)  NOT NULL,
    total_price     INTEGER      NOT NULL,
    order_date      DATE         NOT NULL,
    delivery_date   DATE           NOT NULL,

    CONSTRAINT PK_Purchase_Order         PRIMARY KEY (order_id),
    CONSTRAINT FK_Purchase_Order_customer_id FOREIGN KEY (customer_id)
REFERENCES Customer
);


CREATE TABLE Product (
    product_id  VARCHAR(20) NOT NULL,
    name        VARCHAR(30) NOT NULL,
    price       INTEGER NOT NULL,
    description VARCHAR(50) NOT NULL,
    category    VARCHAR(30) NOT NULL,
    order_id    VARCHAR(20) NOT NULL,

    CONSTRAINT pk_product PRIMARY KEY ( product_id ),
CONSTRAINT FK_PRODUCT_ORDER_ID FOREIGN KEY ( ORDER_ID ) REFERENCES
PURCHASE_ORDER

);


CREATE TABLE shipping (
    shipping_id      VARCHAR(20) NOT NULL,
    shipping_date    DATE        NOT NULL,
    tracking_number  VARCHAR(20) NOT NULL,
    shipping_cost    INTEGER     NOT NULL,
    order_id         VARCHAR(20) NOT NULL,
    shipping_type    VARCHAR(20) NOT NULL,
    shipping_address VARCHAR(50) NOT NULL,

    CONSTRAINT pk_shipping PRIMARY KEY (shipping_id),
    CONSTRAINT fk_shipping_order_id FOREIGN KEY (order_id)
        REFERENCES purchase_order
);


CREATE TABLE Payment (
    payment_id     VARCHAR(20) NOT NULL,
    payment_date   DATE        NOT NULL,
    payment_amount INTEGER     NOT NULL,
    order_id       VARCHAR(20) NOT NULL,
    payment_method VARCHAR(20) NOT NULL,
    payment_status VARCHAR(20) NOT NULL,

    CONSTRAINT PK_Payment       PRIMARY KEY (payment_id),
    CONSTRAINT FK_Payment_order_id FOREIGN KEY (order_id) REFERENCES
Purchase_Order
);


/* Create indices for natural keys, foreign keys, and frequently-queried
columns */

-- Customer
-- Natural Keys
CREATE INDEX IDX_Customer_First_Name ON Customer (First_Name);

-- Pruchase_Order
--  Foreign Keys
CREATE INDEX IDX_Purchase_Order_customer_id_FK ON Purchase_Order
(customer_id);

-- Product
-- Frequently-queried columns
CREATE INDEX IDX_Product_Name  ON Product (Name);
```

```
--  Foreign Keys
CREATE INDEX IDX_Product_order_id_FK  ON Product (order_id);


-- Shipping
--  Foreign Keys
CREATE INDEX IDX_Shipping_order_id_FK   ON Shipping (order_id);

--  Frequently-queried columns
CREATE INDEX IDX_Shipping_Tracking_No ON Shipping (Tracking_Number);


-- Payment
--  Foreign Keys
CREATE INDEX IDX_Payment_order_id_FK  ON Payment (order_id);

--  Frequently-queried columns
CREATE INDEX IDX_Payment_Payment_Status   ON Payment (Payment_Status);



/* Alter Tables by adding Audit Columns */
ALTER TABLE CUSTOMER ADD (
    created_by    VARCHAR2(30),
    date_created  DATE,
    modified_by   VARCHAR2(30),
    date_modified DATE
);

ALTER TABLE Purchase_Order ADD (
    created_by    VARCHAR2(30),
    date_created  DATE,
    modified_by   VARCHAR2(30),
    date_modified DATE
);

ALTER TABLE Product ADD (
    created_by    VARCHAR2(30),
    date_created  DATE,
    modified_by   VARCHAR2(30),
    date_modified DATE
);

ALTER TABLE Shipping ADD (
    created_by    VARCHAR2(30),
    date_created  DATE,
    modified_by   VARCHAR2(30),
    date_modified DATE
);

ALTER TABLE Payment ADD (
    created_by    VARCHAR2(30),
    date_created  DATE,
    modified_by   VARCHAR2(30),
    date_modified DATE
);

/* Create Views */
-- Business purpose: The CustomerInfo view will be used primarily for rapidly
fetching information about customers.
CREATE OR REPLACE VIEW CustomerInfo AS
SELECT
    customer_id,
    first_name,
    last_name,
    email,
    phone_number,
    address
FROM
    Customer;

-- Business purpose: The OrderInfo view will be used to fetch information
about an Order for a Customer.
CREATE OR REPLACE VIEW OrderInfo AS
SELECT
    p.order_id,
    p.customer_id,
    c.first_name,
    c.last_name,
    p.order_status,
    p.total_price,
    p.order_date
FROM
         Purchase_Order p
    JOIN Customer c ON p.customer_id = c.customer_id;

-- Business purpose: The OrderOver50 view will be used to fetch information
of all orders from customer that are over $50.
CREATE OR REPLACE VIEW OrderOver50 AS
SELECT
    p.product_id,
    p.name,
    p.price,
    p.category,
    o.order_id,
    o.total_price,
    c.first_name,
    c.last_name
FROM
         product p
    JOIN purchase_order o ON p.order_id = o.order_id
    JOIN customer       c ON o.customer_id = c.customer_id
WHERE
    o.total_price > 50;
```

```
-- Business purpose: The ShippingInfo view will be used to populate a list of
all Shipping information for an order.
CREATE OR REPLACE VIEW ShippingInfo AS
SELECT
    shipping_id,
    shipping_date,
    tracking_number,
    shipping_type,
    order_id
FROM
    Shipping;

/* Create Sequences */
CREATE SEQUENCE SEQ_Customer_customer_id
    INCREMENT BY 1
    START WITH 1
    NOMAXVALUE
    MINVALUE 1
    NOCACHE;

CREATE SEQUENCE SEQ_Product_Product_id
    INCREMENT BY 1
    START WITH 1
    NOMAXVALUE
    MINVALUE 1
    NOCACHE;

CREATE SEQUENCE SEQ_Shipping_shipping_id
    INCREMENT BY 1
    START WITH 1
    NOMAXVALUE
    MINVALUE 1
    NOCACHE;

/* Create Triggers */
-- Business purpose: The TRG_Customer trigger automatically assigns a
sequential Customer ID to a newly-inserted row in the Customer table,
-- as well as assigning appropriate values to the created_by and date_created
fields.
--If the record is being inserted or updated, appropriate values are assigned
to the modified_by and modified_date fields.
CREATE OR REPLACE TRIGGER TRG_Customer
    BEFORE INSERT OR UPDATE ON Customer
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.customer_id IS NULL THEN
                :NEW.customer_id := SEQ_Customer_customer_id.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
```

```
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Product trigger automatically assigns a
sequential level ID to a newly-inserted row in the Product table,
--as well as assigning appropriate values to the created_by and date_created
fields.
--If the record is being inserted or updated, appropriate values are assigned
to the modified_by and modified_date fields.
CREATE OR REPLACE TRIGGER TRG_Product
    BEFORE INSERT OR UPDATE ON Product
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.Product_id IS NULL THEN
                :NEW.Product_id := SEQ_Product_Product_id.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Pruchase_Order trigger sets the modified_by and
date_modified fields to appropriate values in a newly inserted or updated
record;
--if the record is being inserted, then the created_by and date_created
fields are set to appropriate values too.
CREATE OR REPLACE TRIGGER TRG_Purchase_Order
    BEFORE INSERT OR UPDATE ON Purchase_Order
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Shipping trigger automatically assigns a
sequential comment ID to a newly-inserted row
--in the Shipping table, as well as assigning appropriate values to the
created_by and date_created fields.  If the record is being inserted or
updated,
--appropriate values are assigned to the modified_by and modified_date
fields.
CREATE OR REPLACE TRIGGER TRG_Shipping
    BEFORE INSERT OR UPDATE ON Shipping
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.Shipping_id IS NULL THEN
                :NEW.Shipping_id := SEQ_Shipping_shipping_id.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Payment trigger sets the modified_by and
date_modified fields to appropriate values in a newly inserted or updated
record;
--if the record is being inserted, then the created_by and date_created
fields are set to appropriate values too.
CREATE OR REPLACE TRIGGER TRG_Payment
    BEFORE INSERT OR UPDATE ON Payment
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Check the DBMS data dictionary to make sure that all objects have been
created successfully
SELECT TABLE_NAME FROM USER_TABLES;

SELECT OBJECT_NAME, STATUS, CREATED, LAST_DDL_TIME FROM USER_OBJECTS;
```

DML Source Code

```
/*Customer Table Data*/
insert all
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('hithesh','nayak','hnayak@gmail.com',9874568908,'4567
Renneer rd','FL','Oakland',34760)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('john','doe','johndoe@gmail.com',9873456908,'4563 West
rd','FL','Sanford',32771)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('mayank','mishra','mayankm@gmail.com',9873123908,'4561 West
rd','TX','Richardson',75080)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('amit','kulkarni','amitk@gmail.com',9873345908,'4562 East
rd','TX','Dallas',75088)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('glen','smith','gsmith@gmail.com',9873678908,'4564 South
rd','TX','Plano',75325)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('michael','john','michaelj@gmail.com',9123345908,'4565 North
rd','TX','Frisco',75034)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('aarti','singh','asingh@gmail.com',9345345908,'4566 West
rd','TX','Dallas',75252)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('lesley','moore','lmoore@gmail.com',9678345908,'4567 South
rd','NY','Richardson',10025)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('mary','keller','maryk@gmail.com',9890345908,'4568 Coit
rd','NY','New York',10035)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('kruthi','jaishwal','kjaishwal@gmail.com',9873345123,'4569
East rd','NY','Albany',10045)
```

```
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('andrea','josh','ajosh@gmail.com',9873345345,'4531 West
rd','NY','New York',10025)
into customer(first_name,last_name,email,phone_number, address, state, city,
zipcode) values ('kevin','paul','kpaul@gmail.com',9873345678,'4532 North
rd','NY','Yonkers',11005)
SELECT 1 FROM dual;


/*Purchase Order Table Data*/
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(1,1000,25,'Submitted','05-MAY-2023','15-MAY-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(4,1001,56,'Returned','03-MAY-2023','08-MAY-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(2,1002,45,'Shipped','04-MAY-2023','09-MAY-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(5,1003,40,'In-Transit','01-MAY-2023','03-MAY-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(8,1004,800,'Delivered','24-APR-2023','27-APR-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(6,1005,500,'Cancelled','25-APR-2023','27-APR-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(9,1006,12,'On-Hold','25-APR-2023','29-APR-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(3,1007,50,'Out-for-Delivery','25-APR-2023','28-APR-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(7,1008,35,'Closed','25-APR-2023','28-APR-2023');
insert into
purchase_order(customer_id,order_id,total_price,order_status,order_date,deliv
ery_date) values(10,1009,115,'Rejected','25-APR-2023','28-APR-2023');



/*Product Table- Data*/
insert into product(name,price, description, order_id,category) values
('Sketcher Shoes',50,'Shoes with Memory Foam',1007,'Footwear');
insert into product(name,price, description, order_id,category) values
('Google Pixel 7 Pro',800,'16GB RAM with 512GB Storage',1004,'Mobiles');
insert into product(name,price, description, order_id,category) values
('Comforter',115,'Ultra Soft=Reversible',1009,'Household');
insert into product(name,price, description, order_id,category) values
('Universal Travel Adapter',12,'All-In-One',1006,'Accessories');
```

```
insert into product(name,price, description, order_id,category) values
('Vacuum Cleaner',35,'Multipurpose',1008,'Home and Kitchen');
insert into product(name,price, description, order_id,category) values ('ASUS
Laptop',500,'32GB RAM with 1.5TB Storage',1005,'Electronics');
insert into product(name,price, description, order_id,category) values
('Amazon Gift Card',25,'Multipurpose',1000,'Gift Cards');
insert into product(name,price, description, order_id,category) values
('Trimmer',40,'Multipurpose Trimmer',1003,'Personal');
insert into product(name,price, description, order_id,category) values
('Similac',45,'Infant Formula',1002,'Toys, Kids and Baby');
insert into product(name,price, description, order_id,category) values
('Brake Pads',56,'Ceramic Brake Pads for Automobiles',1001,'Automotive and
Industrial');



/*Shipping-Table*/
insert all
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1000,'ABCDE10001','07-MAY-2023',4,'Express','4567
Renneer rd FL Oakland 34760')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1001,'EFGHI10002','04-MAY-2023',2,'Standard','4562
East rd TX Dallas 75088')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1002,'AABBC10003','04-MAY-2023',0,'Free','4563 West
rd FL Sanford 32771')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1003,'XXYYZ10004','03-MAY-2023',4,'Express','4564
South rd TX Plano 75325')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1004,'MMNNO10005','25-APR-2023',0,'Free','4567 South
rd NY Richardson 10025')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1005,'JKLMN10006','26-APR-2023',2,'Standard','4565
North rd TX Frisco 75034')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1006,'OPQRS10007','25-APR-2023',4,'Express','4568
Coit rd NY New York 10035')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1007,'TUVWX10008','26-APR-2023',2,'Standard','4561
West rd TX Richardson 75080')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1008,'YZABC10009','26-APR-2023',0,'Free','4566 West
rd TX Dallas 75252')
 into
shipping(order_id,tracking_number,shipping_date,shipping_cost,shipping_type,s
hipping_address) values(1009,'RRSST10010','25-APR-2023',4,'Express','4569
East rd NY Albany 10045')
select 1 from dual;

/*Payment Table Data*/
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (100,1000,29,'Credit Card','Paid','05-MAY-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (101,1001,58,'Debit Card','Refunded','03-MAY-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (102,1002,45,'Gift Card','Paid','04-MAY-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (103,1003,44,'Electronic Check','Paid','01-MAY-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (104,1004,800,'Credit Card','Paid','24-APR-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (105,1005,502,'Debit Card','Refunded','26-APR-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (106,1006,16,'Electronic Check','Paid','25-APR-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (107,1007,52,'Gift Card','Paid','25-APR-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (108,1008,35,'Debit Card','Paid','25-APR-2023');
insert into
payment(payment_id,order_id,payment_amount,payment_method,payment_status,paym
ent_date) values (109,1009,119,'Credit Card','Refunded','26-APR-2023');
commit;
```

DDL Output

```
Trigger TRG_PAYMENT dropped.

Trigger TRG_SHIPPING dropped.

Trigger TRG_PURCHASE_ORDER dropped.
Trigger TRG_PRODUCT dropped.

Trigger TRG_CUSTOMER dropped.

Sequence SEQ_CUSTOMER_CUSTOMER_ID dropped.

Sequence SEQ_PRODUCT_PRODUCT_ID dropped.

Sequence SEQ_SHIPPING_SHIPPING_ID dropped.

View CUSTOMERINFO dropped.

View ORDERINFO dropped.

View ORDEROVER50 dropped.

View SHIPPINGINFO dropped.

Index IDX_PAYMENT_PAYMENT_STATUS dropped.

Index IDX_PAYMENT_ORDER_ID_FK dropped.

Index IDX_SHIPPING_TRACKING_NO dropped.

Index IDX_SHIPPING_ORDER_ID_FK dropped.

Index IDX_PRODUCT_ORDER_ID_FK dropped.

Index IDX_PRODUCT_NAME dropped.

Index IDX_PURCHASE_ORDER_CUSTOMER_ID_FK dropped.

Index IDX_CUSTOMER_FIRST_NAME dropped.

Table PAYMENT dropped.

Table SHIPPING dropped.

Table PRODUCT dropped.

Table PURCHASE_ORDER dropped.

Table CUSTOMER dropped.

Table CUSTOMER created.

Table PURCHASE_ORDER created.

Table PRODUCT created.

Table SHIPPING created.

Table PAYMENT created.
```

```
Index IDX_CUSTOMER_FIRST_NAME created.

Index IDX_PURCHASE_ORDER_CUSTOMER_ID_FK created.

Index IDX_PRODUCT_NAME created.

Index IDX_PRODUCT_ORDER_ID_FK created.

Index IDX_SHIPPING_ORDER_ID_FK created.

Index IDX_SHIPPING_TRACKING_NO created.

Index IDX_PAYMENT_ORDER_ID_FK created.

Index IDX_PAYMENT_PAYMENT_STATUS created.

Table CUSTOMER altered.

Table PURCHASE_ORDER altered.

Table PRODUCT altered.

Table SHIPPING altered.

Table PAYMENT altered.

View CUSTOMERINFO created.

View ORDERINFO created.

View ORDEROVER50 created.

View SHIPPINGINFO created.

Sequence SEQ_CUSTOMER_CUSTOMER_ID created.

Sequence SEQ_PRODUCT_PRODUCT_ID created.

Sequence SEQ_SHIPPING_SHIPPING_ID created.

Trigger TRG_CUSTOMER compiled

Trigger TRG_PRODUCT compiled

Trigger TRG_PURCHASE_ORDER compiled

Trigger TRG_SHIPPING compiled

Trigger TRG_PAYMENT compiled
```

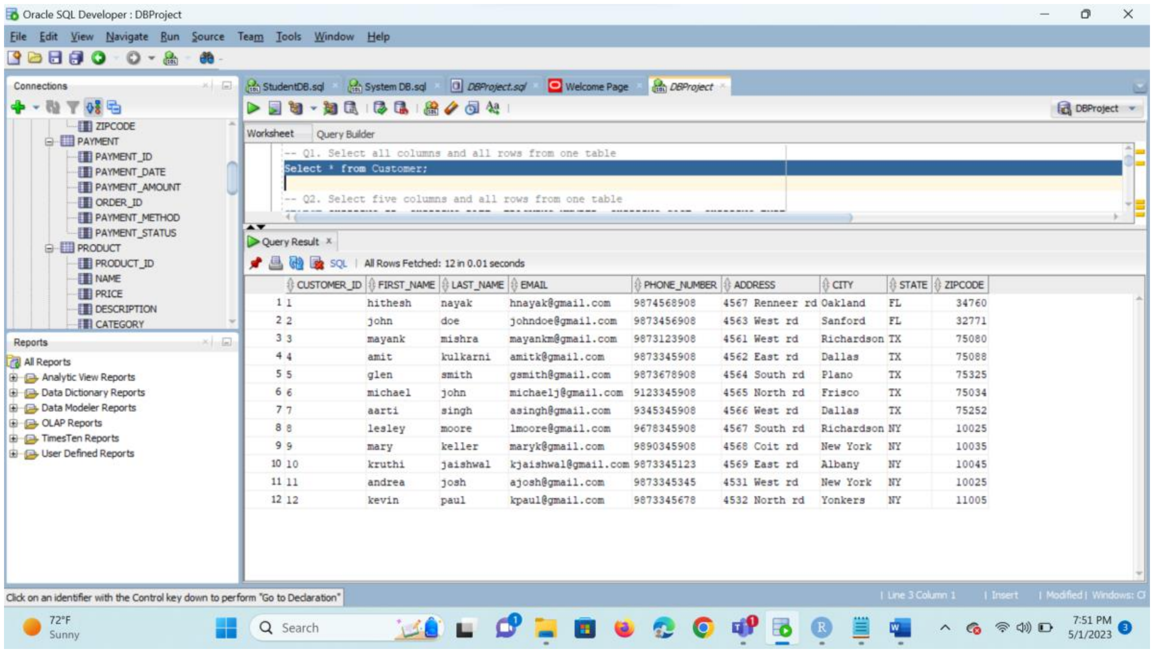
DML Output

```
12 rows inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

10 rows inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.
```

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Commit complete.
```

Query Source Code and Output

```
--query--
-- Q1. Select all columns and all rows from one table
Select * from Customer;
```

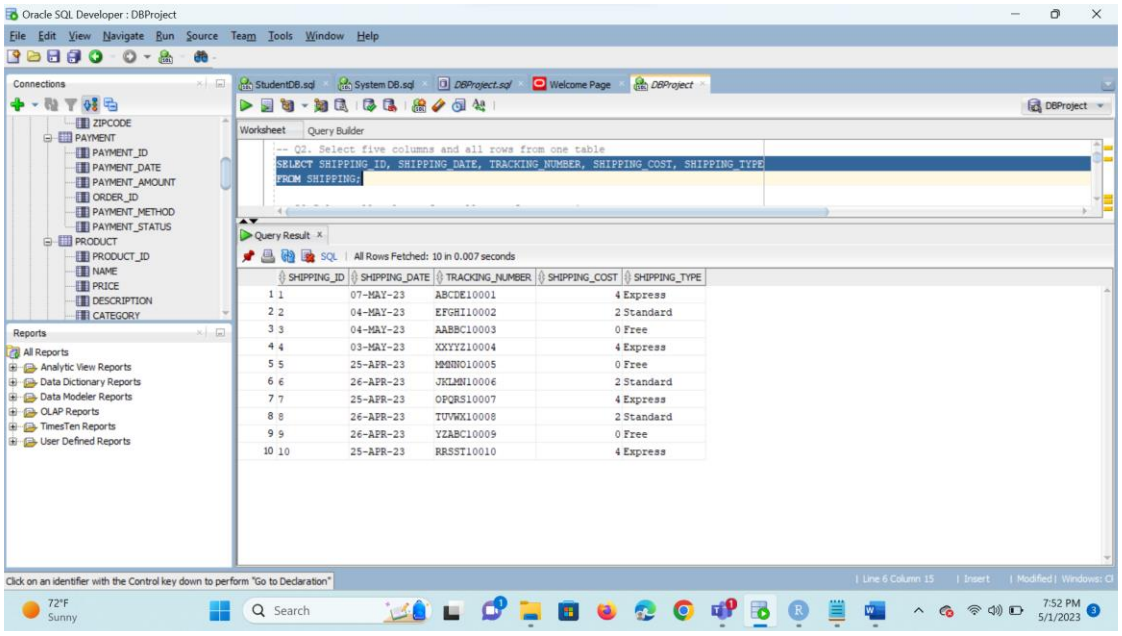


```
-- Q1. Select all columns and all rows from one table
Select * from Customer;

-- Q2. Select five columns and all rows from one table
```

All Rows Fetched: 12 in 0.01 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS	CITY	STATE	ZIPCODE
1	1	hithesh	nayak	hnayak@gmail.com	9874568908	4567 Renneer rd	Oakland	FL	34760
2	2	john	doe	johndoe@gmail.com	9873456908	4563 West rd	Sanford	FL	32771
3	3	mayank	mishra	mayankm@gmail.com	9873123908	4561 West rd	Richardson	TX	75080
4	4	amit	kulkarni	amitk@gmail.com	9873345908	4562 East rd	Dallas	TX	75088
5	5	glen	smith	gsmith@gmail.com	9873678908	4564 South rd	Plano	TX	75325
6	6	michael	john	michaelj@gmail.com	9123345908	4565 North rd	Frisco	TX	75034
7	7	aarti	singh	asingh@gmail.com	9345345908	4566 West rd	Dallas	TX	75252
8	8	lesley	moore	lmoore@gmail.com	9678345908	4567 South rd	Richardson	NY	10025
9	9	mary	keller	maryk@gmail.com	9890345908	4568 Coit rd	New York	NY	10035
10	10	kruthi	jaishwal	kjaishwal@gmail.com	9873345123	4569 East rd	Albany	NY	10045
11	11	andrea	josh	ajosh@gmail.com	9873345345	4531 West rd	New York	NY	10025
12	12	kevin	paul	kpaul@gmail.com	9873345678	4532 North rd	Yonkers	NY	11005

```
-- Q2. Select five columns and all rows from one table
SELECT SHIPPING_ID, SHIPPING_DATE, TRACKING_NUMBER, SHIPPING_COST,
SHIPPING_TYPE
FROM SHIPPING;
```

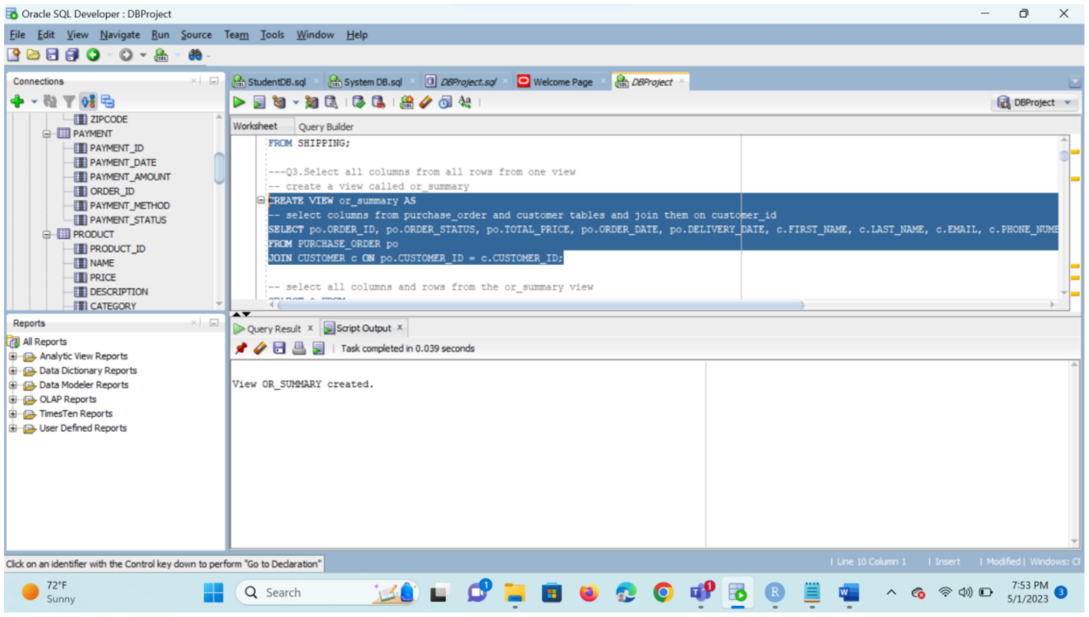


```
-- Q2. Select five columns and all rows from one table
SELECT SHIPPING_ID, SHIPPING_DATE, TRACKING_NUMBER, SHIPPING_COST, SHIPPING_TYPE
FROM SHIPPING;
```

All Rows Fetched: 10 in 0.007 seconds

	SHIPPING_ID	SHIPPING_DATE	TRACKING_NUMBER	SHIPPING_COST	SHIPPING_TYPE
1	1	07-MAY-23	ABCDE10001	4	Express
2	2	04-MAY-23	EFGHI10002	2	Standard
3	3	04-MAY-23	AABBC10003	0	Free
4	4	03-MAY-23	XXYYZ10004	4	Express
5	5	25-APR-23	MMNNO10005	0	Free
6	6	26-APR-23	JKLMN10006	2	Standard
7	7	25-APR-23	OPQRS10007	4	Express
8	8	26-APR-23	TUVWX10008	2	Standard
9	9	26-APR-23	YZABC10009	0	Free
10	10	25-APR-23	RRSST10010	4	Express

```
---Q3.Select all columns from all rows from one view
-- create a view called or_summary
CREATE VIEW or_summary AS
-- select columns from purchase_order and customer tables and join them on
customer_id
SELECT po.ORDER_ID, po.ORDER_STATUS, po.TOTAL_PRICE, po.ORDER_DATE,
po.DELIVERY_DATE, c.FIRST_NAME, c.LAST_NAME, c.EMAIL, c.PHONE_NUMBER,
c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE
FROM PURCHASE_ORDER po
JOIN CUSTOMER c ON po.CUSTOMER_ID = c.CUSTOMER_ID;
```

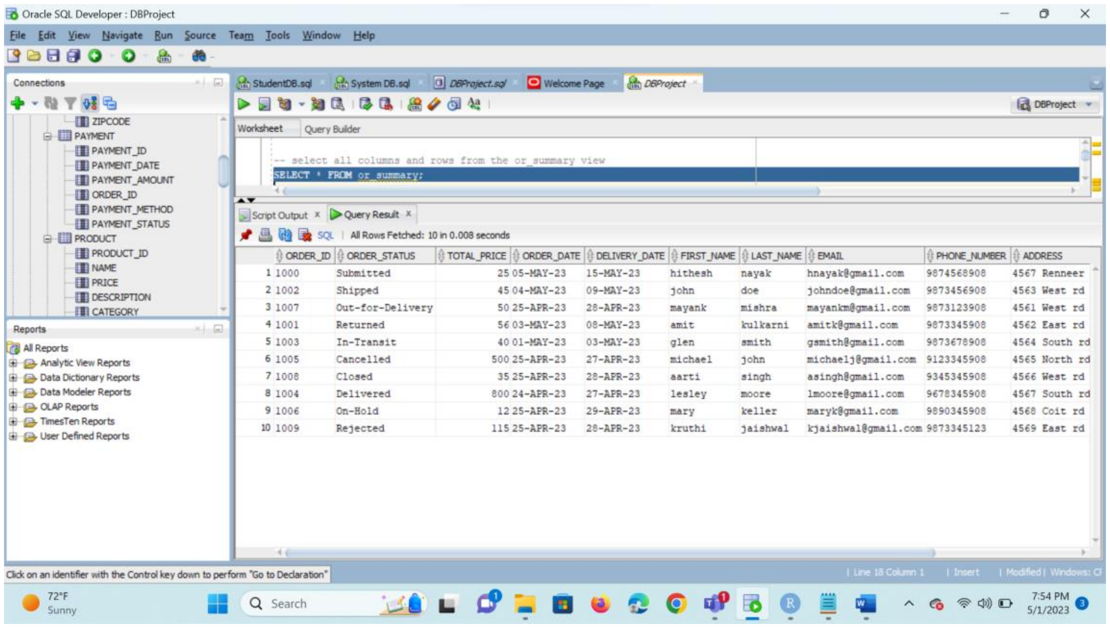


```
FROM SHIPPING;

---Q3.Select all columns from all rows from one view
-- create a view called or_summary
CREATE VIEW or_summary AS
-- select columns from purchase_order and customer tables and join them on customer_id
SELECT po.ORDER_ID, po.ORDER_STATUS, po.TOTAL_PRICE, po.ORDER_DATE, po.DELIVERY_DATE, c.FIRST_NAME, c.LAST_NAME, c.EMAIL, c.PHONE_NUME
FROM PURCHASE_ORDER po
JOIN CUSTOMER c ON po.CUSTOMER_ID = c.CUSTOMER_ID;

-- select all columns and rows from the or_summary view
```

Task completed in 0.039 seconds

```
View OR_SUMMARY created.
```

```
-- select all columns and rows from the or_summary view
SELECT * FROM or_summary;
```

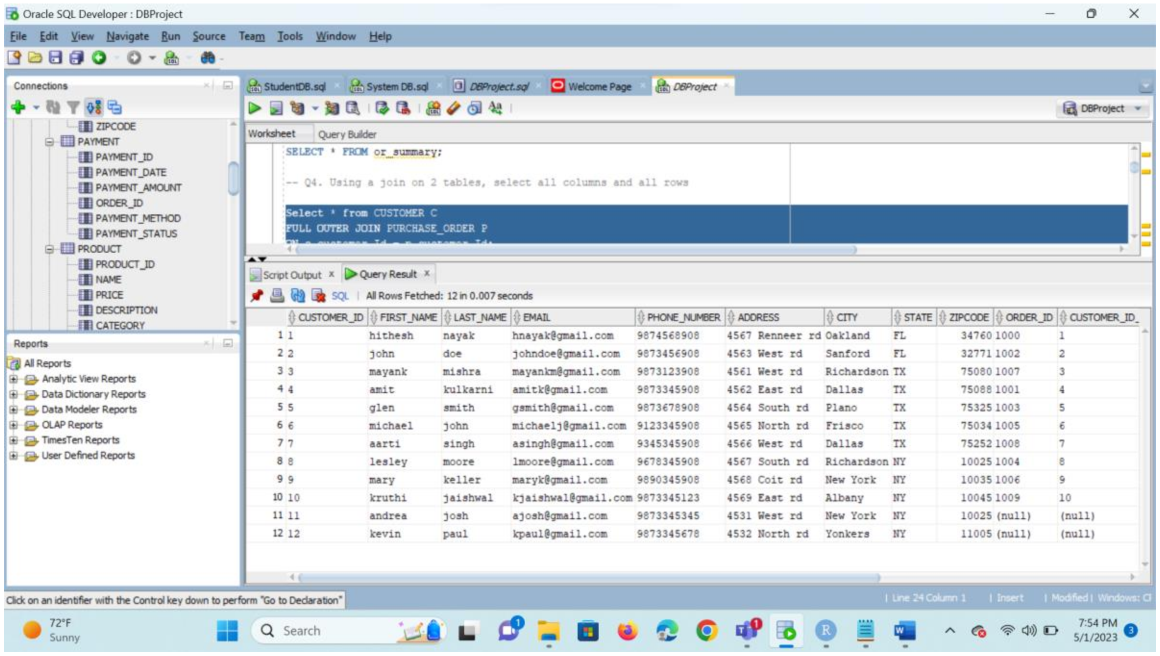


```
-- select all columns and rows from the or_summary view
SELECT * FROM or_summary;
```

All Rows Fetched: 10 in 0.008 seconds

	ORDER_ID	ORDER_STATUS	TOTAL_PRICE	ORDER_DATE	DELIVERY_DATE	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS
1	1000	Submitted	25	05-MAY-23	15-MAY-23	hithesh	nayak	hnayak@gmail.com	9874568908	4567 Renneer
2	1002	Shipped	45	04-MAY-23	09-MAY-23	john	doe	johndoe@gmail.com	9873456908	4563 West rd
3	1007	Out-for-Delivery	50	25-APR-23	28-APR-23	mayank	mishra	mayankm@gmail.com	9873123908	4561 West rd
4	1001	Returned	56	03-MAY-23	08-MAY-23	amit	kulkarni	amitk@gmail.com	9873345908	4562 East rd
5	1003	In-Transit	40	01-MAY-23	03-MAY-23	glen	smith	gsmith@gmail.com	9873678908	4564 South rd
6	1005	Cancelled	500	25-APR-23	27-APR-23	michael	john	michaelj@gmail.com	9123345908	4565 North rd
7	1008	Closed	35	25-APR-23	28-APR-23	aarti	singh	asingh@gmail.com	9345345908	4566 West rd
8	1004	Delivered	800	24-APR-23	27-APR-23	lesley	moore	lmoore@gmail.com	9678345908	4567 South rd
9	1006	On-Hold	12	25-APR-23	29-APR-23	mary	keller	maryk@gmail.com	9890345908	4568 Coit rd
10	1009	Rejected	115	25-APR-23	28-APR-23	kruthi	jaishwal	kjaishwal@gmail.com	9873345123	4569 East rd

```
-- Q4. Using a join on 2 tables, select all columns and all rows

Select * from CUSTOMER C
FULL OUTER JOIN PURCHASE_ORDER P
ON c.customer_Id = p.customer_Id;
```

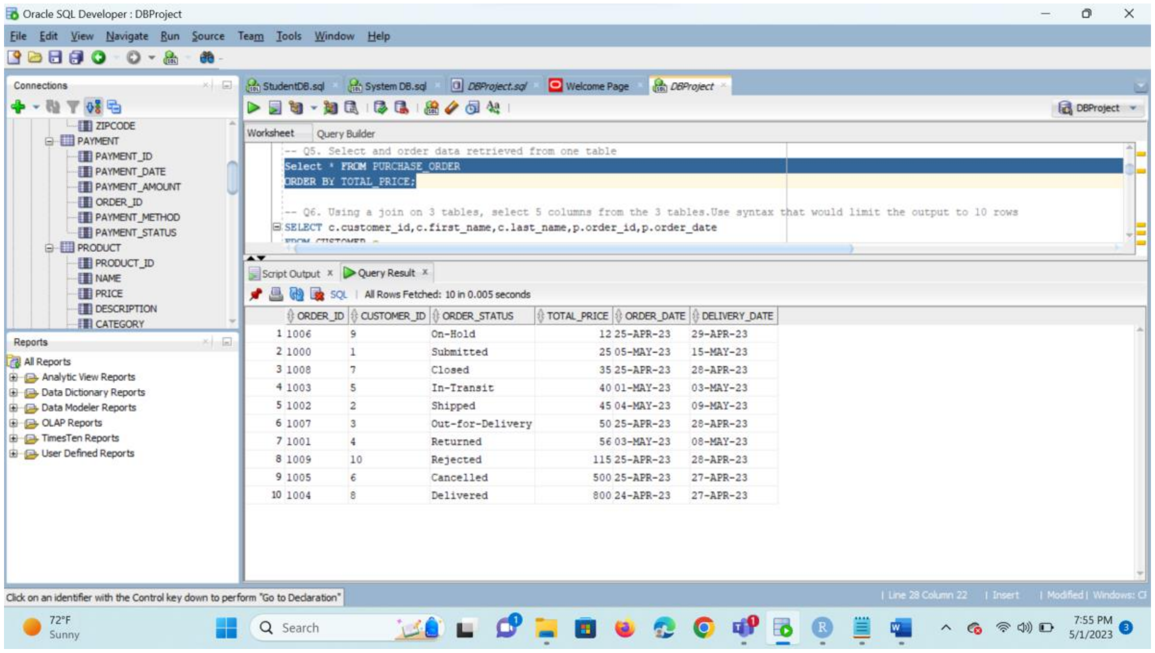


```
SELECT * FROM or_summary;

-- Q4. Using a join on 2 tables, select all columns and all rows

Select * from CUSTOMER C
FULL OUTER JOIN PURCHASE_ORDER P
```

All Rows Fetched: 12 in 0.007 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS	CITY	STATE	ZIPCODE	ORDER_ID	CUSTOMER_ID_
1	1	hithesh	nayak	hnayak@gmail.com	9874568908	4567 Renneer rd	Oakland	FL	34760	1000	1
2	2	john	doe	johndoe@gmail.com	9873456908	4563 West rd	Sanford	FL	32771	1002	2
3	3	mayank	mishra	mayankm@gmail.com	9873123908	4561 West rd	Richardson	TX	75080	1007	3
4	4	amit	kulkarni	amitk@gmail.com	9873345908	4562 East rd	Dallas	TX	75088	1001	4
5	5	glen	smith	gsmith@gmail.com	9873678908	4564 South rd	Plano	TX	75325	1003	5
6	6	michael	john	michaelj@gmail.com	9123345908	4565 North rd	Frisco	TX	75034	1005	6
7	7	aarti	singh	asingh@gmail.com	9345345908	4566 West rd	Dallas	TX	75252	1008	7
8	8	lesley	moore	lmoore@gmail.com	9678345908	4567 South rd	Richardson	NY	10025	1004	8
9	9	mary	keller	maryk@gmail.com	9890345908	4568 Coit rd	New York	NY	10035	1006	9
10	10	kruthi	jaishwal	kjaishwal@gmail.com	9873345123	4569 East rd	Albany	NY	10045	1009	10
11	11	andrea	josh	ajosh@gmail.com	9873345345	4531 West rd	New York	NY	10025	(null)	(null)
12	12	kevin	paul	kpaul@gmail.com	9873345678	4532 North rd	Yonkers	NY	11005	(null)	(null)

```
-- Q5. Select and order data retrieved from one table
Select * FROM PURCHASE_ORDER
ORDER BY TOTAL_PRICE;
```

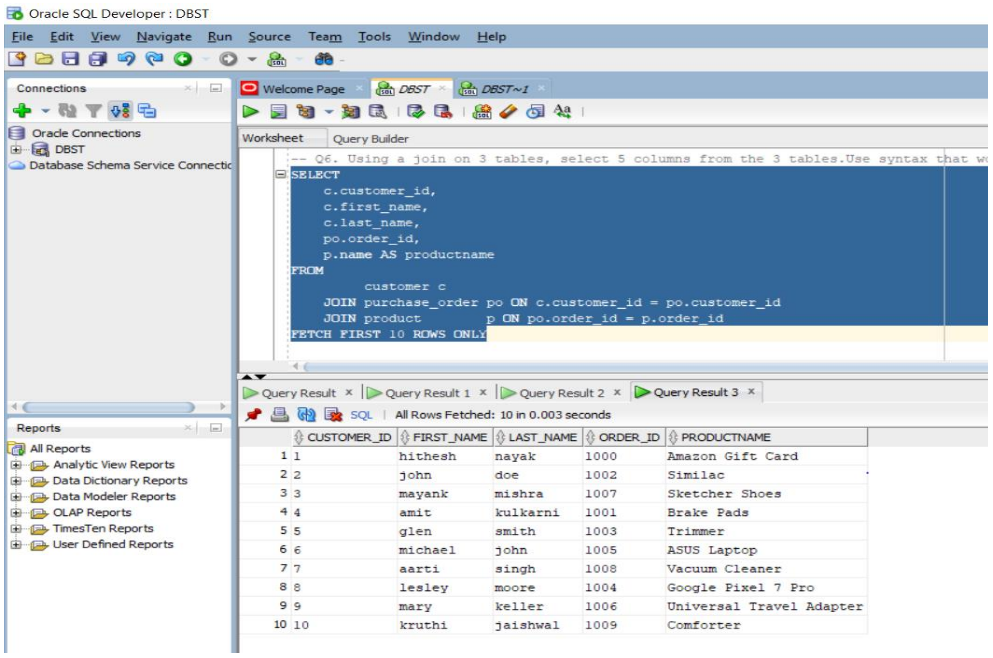


```
-- Q5. Select and order data retrieved from one table
Select * FROM PURCHASE_ORDER
ORDER BY TOTAL_PRICE;

-- Q6. Using a join on 3 tables, select 5 columns from the 3 tables.Use syntax that would limit the output to 10 rows
SELECT c.customer_id,c.first_name,c.last_name,p.order_id,p.order_date
```

All Rows Fetched: 10 in 0.005 seconds

	ORDER_ID	CUSTOMER_ID	ORDER_STATUS	TOTAL_PRICE	ORDER_DATE	DELIVERY_DATE
1	1006	9	On-Hold	12	25-APR-23	29-APR-23
2	1000	1	Submitted	25	05-MAY-23	15-MAY-23
3	1008	7	Closed	35	25-APR-23	28-APR-23
4	1003	5	In-Transit	40	01-MAY-23	03-MAY-23
5	1002	2	Shipped	45	04-MAY-23	09-MAY-23
6	1007	3	Out-for-Delivery	50	25-APR-23	28-APR-23
7	1001	4	Returned	56	03-MAY-23	08-MAY-23
8	1009	10	Rejected	115	25-APR-23	28-APR-23
9	1005	6	Cancelled	500	25-APR-23	27-APR-23
10	1004	8	Delivered	800	24-APR-23	27-APR-23

```
-- Q6. Using a join on 3 tables, select 5 columns from the 3 tables.Use
syntax that would limit the output to 10 rows
SELECT c.customer_id, c.first_name, c.last_name, po.order_id, p.name
  AS productname
FROM customer c
JOIN purchase_order po ON c.customer_id = po.customer_id
JOIN product p ON po.order_id = p.order_id
FETCH FIRST 10 ROWS ONLY
```



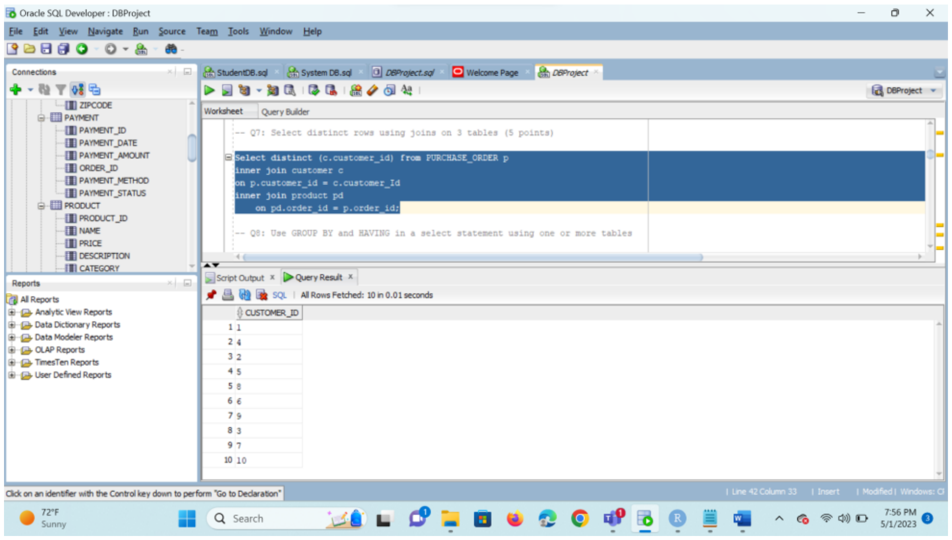
```
-- Q6. Using a join on 3 tables, select 5 columns from the 3 tables.Use syntax that w
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    po.order_id,
    p.name AS productname
FROM
         customer c
    JOIN purchase_order po ON c.customer_id = po.customer_id
    JOIN product        p ON po.order_id = p.order_id
FETCH FIRST 10 ROWS ONLY
```

All Rows Fetched: 10 in 0.003 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	ORDER_ID	PRODUCTNAME
1	1	hithesh	nayak	1000	Amazon Gift Card
2	2	john	doe	1002	Similac
3	3	mayank	mishra	1007	Sketcher Shoes
4	4	amit	kulkarni	1001	Brake Pads
5	5	glen	smith	1003	Trimmer
6	6	michael	john	1005	ASUS Laptop
7	7	aarti	singh	1008	Vacuum Cleaner
8	8	lesley	moore	1004	Google Pixel 7 Pro
9	9	mary	keller	1006	Universal Travel Adapter
10	10	kruthi	jaishwal	1009	Comforter

```
-- Q7: Select distinct rows using joins on 3 tables (5 points)
```

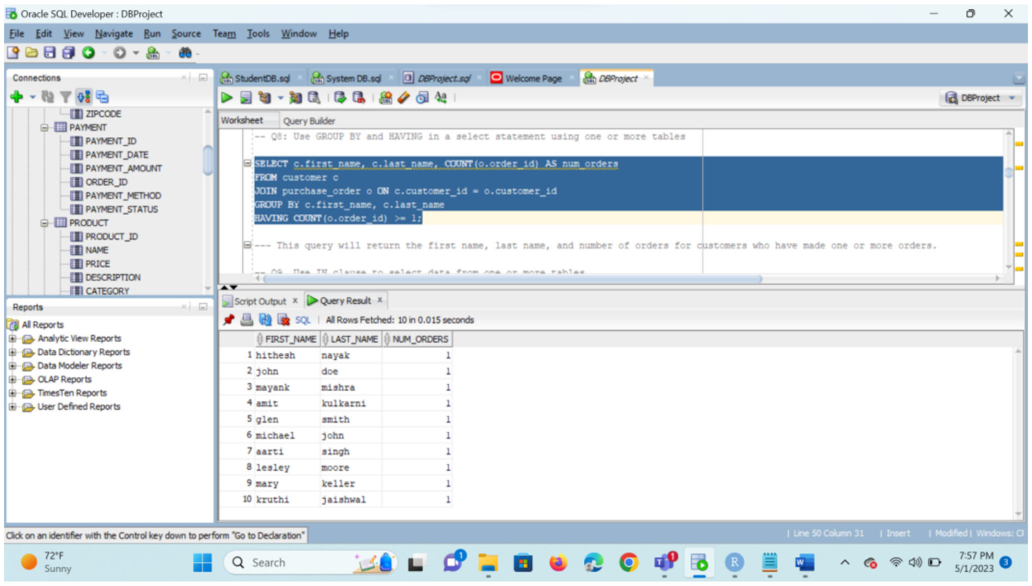
```
Select distinct (c.customer_id) from PURCHASE_ORDER p
inner join customer c
on p.customer_id = c.customer_Id
inner join product pd
    on pd.order_id = p.order_id;
```



```
-- Q7: Select distinct rows using joins on 3 tables (5 points)

Select distinct (c.customer_id) from PURCHASE_ORDER p
inner join customer c
on p.customer_id = c.customer_Id
inner join product pd
    on pd.order_id = p.order_id;

-- Q8: Use GROUP BY and HAVING in a select statement using one or more tables
```

```
All Rows Fetched: 10 in 0.01 seconds

   CUSTOMER_ID
 1 1
 2 4
 3 2
 4 5
 5 8
 6 6
 7 9
 8 3
 9 7
10 10
```

```
-- Q8: Use GROUP BY and HAVING in a select statement using one or more tables

SELECT c.first_name, c.last_name, COUNT(o.order_id) AS num_orders
FROM customer c
JOIN purchase_order o ON c.customer_id = o.customer_id
GROUP BY c.first_name, c.last_name
HAVING COUNT(o.order_id) >= 1;
```

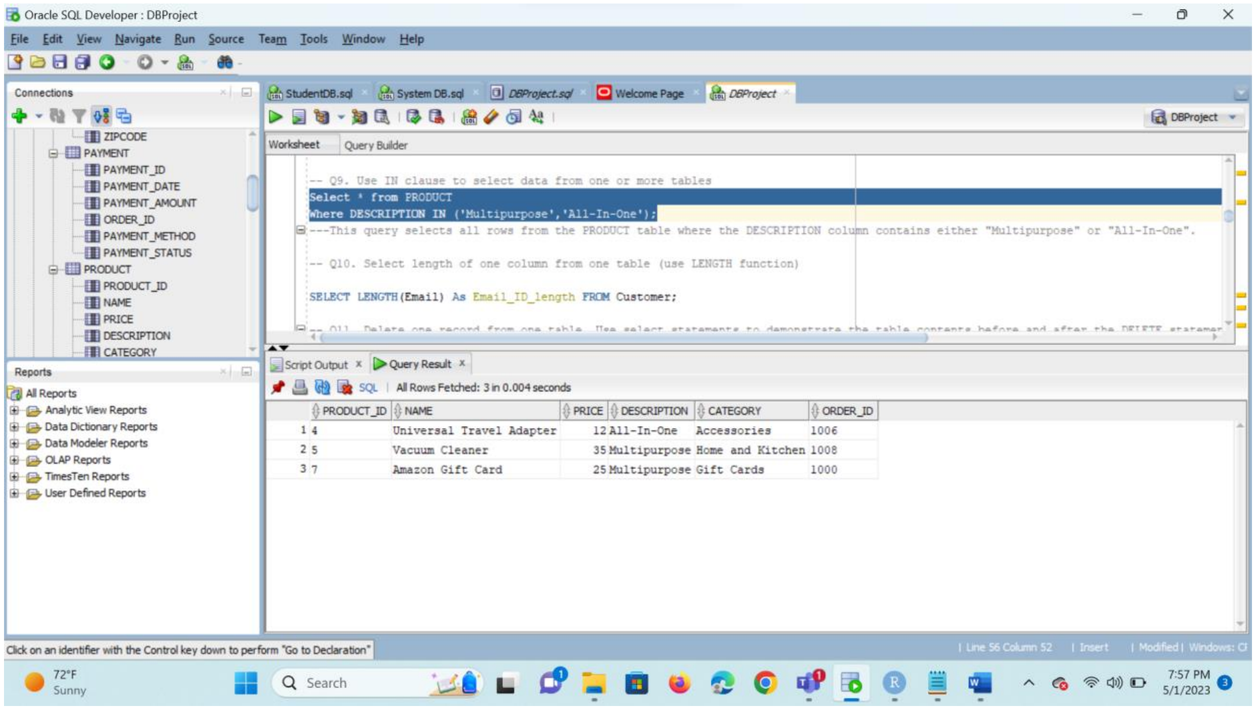


```
-- Q8: Use GROUP BY and HAVING in a select statement using one or more tables

SELECT c.first_name, c.last_name, COUNT(o.order_id) AS num_orders
FROM customer c
JOIN purchase_order o ON c.customer_id = o.customer_id
GROUP BY c.first_name, c.last_name
HAVING COUNT(o.order_id) >= 1;

--- This query will return the first name, last name, and number of orders for customers who have made one or more orders.
```

```
All Rows Fetched: 10 in 0.015 seconds

   FIRST_NAME LAST_NAME  NUM_ORDERS
 1 hithesh    nayak               1
 2 john       doe                 1
 3 mayank     mishra              1
 4 amit       kulkarni            1
 5 glen       smith               1
 6 michael    john                1
 7 aarti      singh               1
 8 lesley     moore               1
 9 mary       keller              1
10 kruthi     jaishwal            1
```

--- This query will return the first name, last name, and number of orders for customers who have made one or more orders.

```
-- Q9. Use IN clause to select data from one or more tables
Select * from PRODUCT
Where DESCRIPTION IN ('Multipurpose','All-In-One');
---This query selects all rows from the PRODUCT table where the DESCRIPTION
column contains either "Multipurpose" or "All-In-One".
```

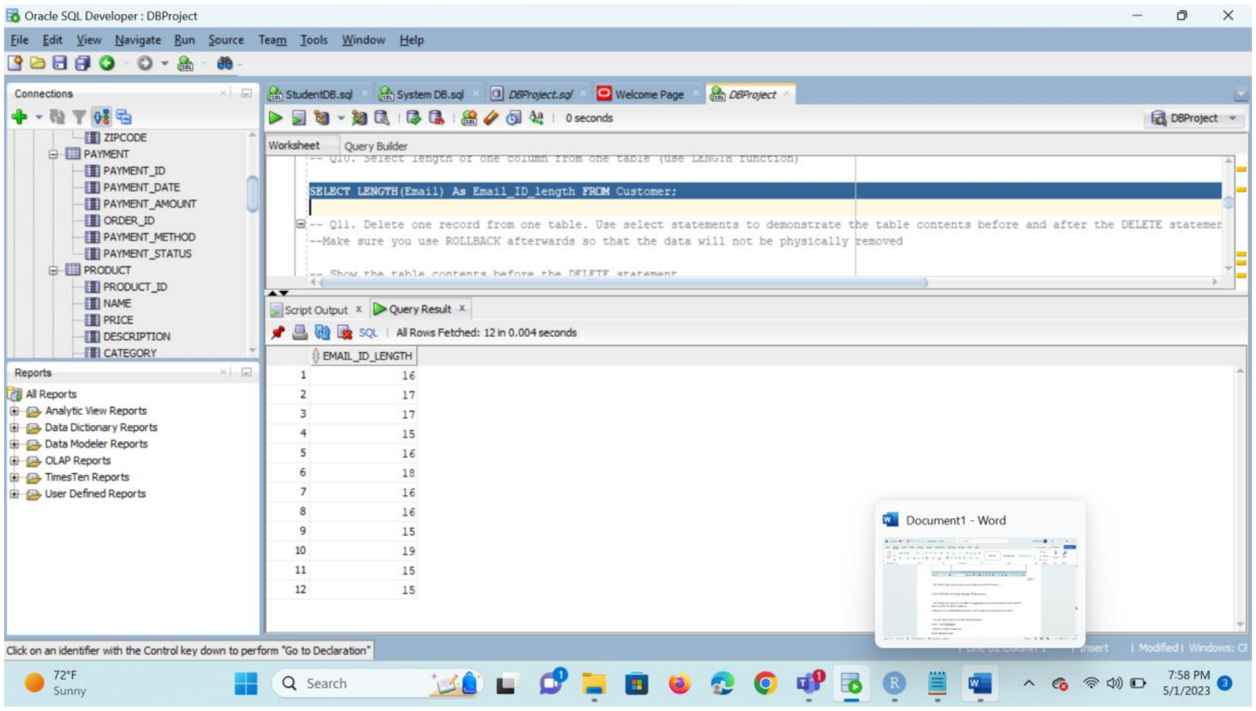


```
-- Q9. Use IN clause to select data from one or more tables
Select * from PRODUCT
Where DESCRIPTION IN ('Multipurpose','All-In-One');
---This query selects all rows from the PRODUCT table where the DESCRIPTION column contains either "Multipurpose" or "All-In-One".

-- Q10. Select length of one column from one table (use LENGTH function)

SELECT LENGTH(Email) As Email_ID_length FROM Customer;
```

All Rows Fetched: 3 in 0.004 seconds

	PRODUCT_ID	NAME	PRICE	DESCRIPTION	CATEGORY	ORDER_ID
1	4	Universal Travel Adapter	12	All-In-One	Accessories	1006
2	5	Vacuum Cleaner	35	Multipurpose	Home and Kitchen	1008
3	7	Amazon Gift Card	25	Multipurpose	Gift Cards	1000

```
-- Q10. Select length of one column from one table (use LENGTH function)

SELECT LENGTH(Email) As Email_ID_length FROM Customer;
```

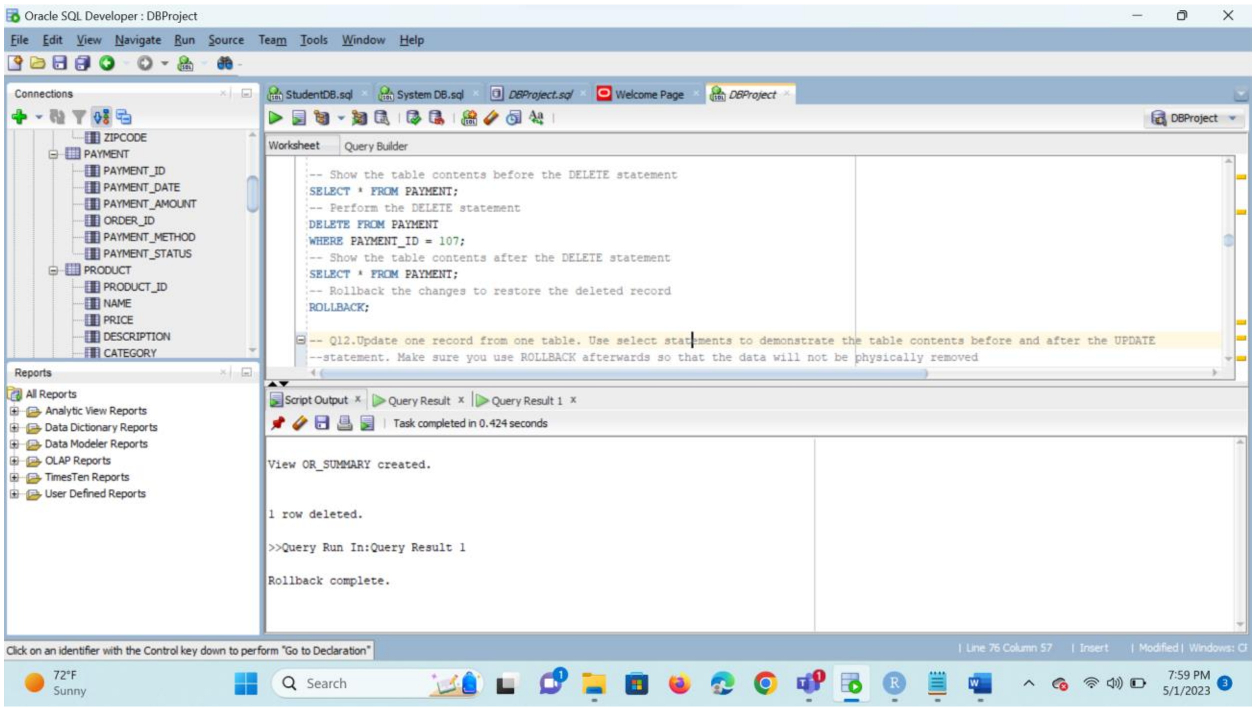


```
-- Q10. Select length of one column from one table (use LENGTH function)

SELECT LENGTH(Email) As Email_ID_length FROM Customer;

-- Q11. Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statemer
--Make sure you use ROLLBACK afterwards so that the data will not be physically removed

-- Show the table contents before the DELETE statement
```

All Rows Fetched: 12 in 0.004 seconds

```
     EMAIL_ID_LENGTH
 1                16
 2                17
 3                17
 4                15
 5                16
 6                18
 7                16
 8                16
 9                15
10                19
11                15
12                15
```

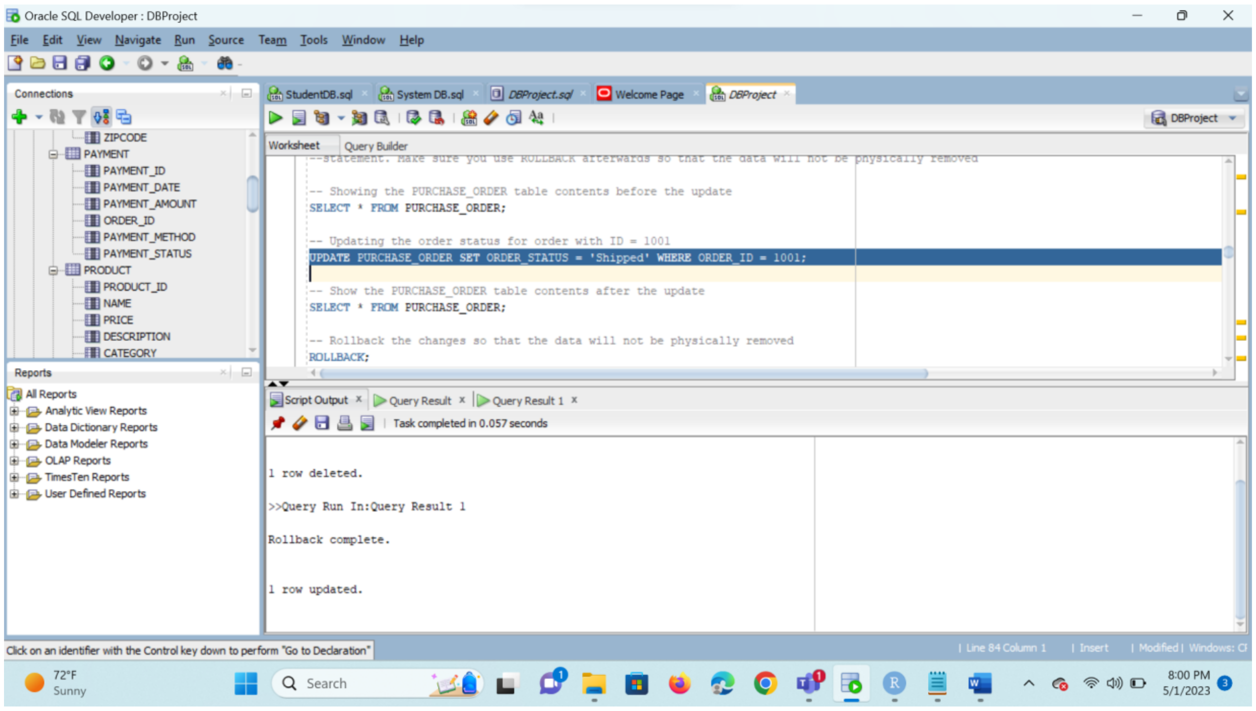
```
-- Q11. Delete one record from one table. Use select statements to
demonstrate the table contents before and after the DELETE statement.
--Make sure you use ROLLBACK afterwards so that the data will not be
physically removed

-- Show the table contents before the DELETE statement
SELECT * FROM PAYMENT;
-- Perform the DELETE statement
DELETE FROM PAYMENT
WHERE PAYMENT_ID = 107;
-- Show the table contents after the DELETE statement
SELECT * FROM PAYMENT;
-- Rollback the changes to restore the deleted record
ROLLBACK;
```



```
-- Show the table contents before the DELETE statement
SELECT * FROM PAYMENT;
-- Perform the DELETE statement
DELETE FROM PAYMENT
WHERE PAYMENT_ID = 107;
-- Show the table contents after the DELETE statement
SELECT * FROM PAYMENT;
-- Rollback the changes to restore the deleted record
ROLLBACK;

-- Q12.Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE
--statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed
```

```
View OR_SUMMARY created.


1 row deleted.

>>Query Run In:Query Result 1

Rollback complete.
```

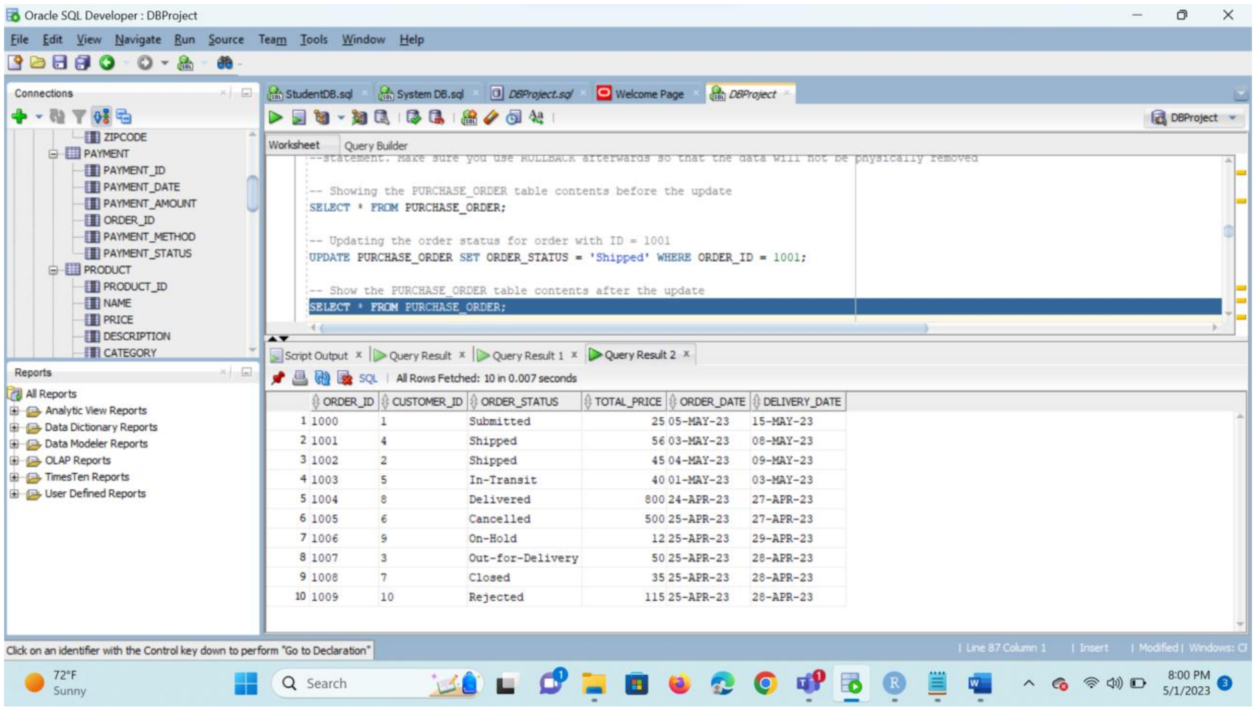
```
-- Q12.Update one record from one table. Use select statements to demonstrate
the table contents before and after the UPDATE
--statement. Make sure you use ROLLBACK afterwards so that the data will not
be physically removed

-- Showing the PURCHASE_ORDER table contents before the update
SELECT * FROM PURCHASE_ORDER;

-- Updating the order status for order with ID = 1001
UPDATE PURCHASE_ORDER SET ORDER_STATUS = 'Shipped' WHERE ORDER_ID = 1001;

-- Show the PURCHASE_ORDER table contents after the update
SELECT * FROM PURCHASE_ORDER;

-- Rollback the changes so that the data will not be physically removed
ROLLBACK;
```



```
--statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

-- Showing the PURCHASE_ORDER table contents before the update
SELECT * FROM PURCHASE_ORDER;

-- Updating the order status for order with ID = 1001
UPDATE PURCHASE_ORDER SET ORDER_STATUS = 'Shipped' WHERE ORDER_ID = 1001;

-- Show the PURCHASE_ORDER table contents after the update
SELECT * FROM PURCHASE_ORDER;

-- Rollback the changes so that the data will not be physically removed
ROLLBACK;
```

Task completed in 0.057 seconds

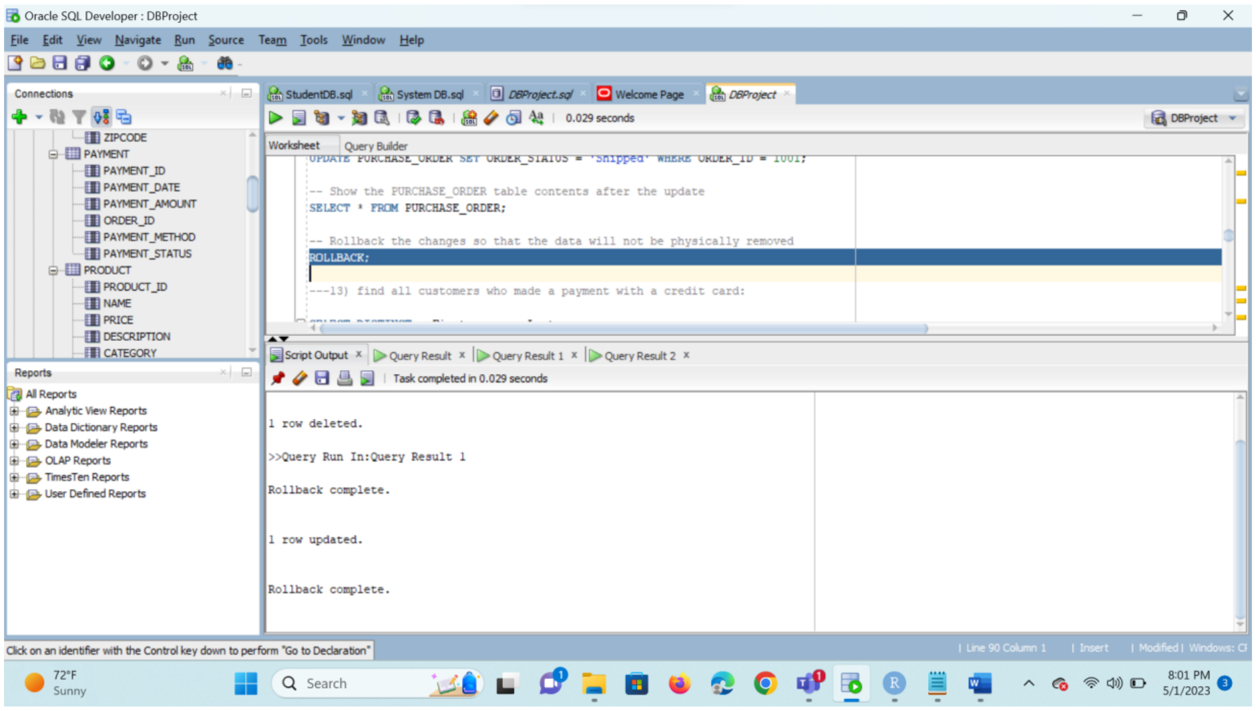
```
1 row deleted.

>>Query Run In:Query Result 1

Rollback complete.


1 row updated.
```



```
--statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

-- Showing the PURCHASE_ORDER table contents before the update
SELECT * FROM PURCHASE_ORDER;

-- Updating the order status for order with ID = 1001
UPDATE PURCHASE_ORDER SET ORDER_STATUS = 'Shipped' WHERE ORDER_ID = 1001;

-- Show the PURCHASE_ORDER table contents after the update
SELECT * FROM PURCHASE_ORDER;
```

All Rows Fetched: 10 in 0.007 seconds

	ORDER_ID	CUSTOMER_ID	ORDER_STATUS	TOTAL_PRICE	ORDER_DATE	DELIVERY_DATE
1	1000	1	Submitted	25	05-MAY-23	15-MAY-23
2	1001	4	Shipped	56	03-MAY-23	08-MAY-23
3	1002	2	Shipped	45	04-MAY-23	09-MAY-23
4	1003	5	In-Transit	40	01-MAY-23	03-MAY-23
5	1004	8	Delivered	800	24-APR-23	27-APR-23
6	1005	6	Cancelled	500	25-APR-23	27-APR-23
7	1006	9	On-Hold	12	25-APR-23	29-APR-23
8	1007	3	Out-for-Delivery	50	25-APR-23	28-APR-23
9	1008	7	Closed	35	25-APR-23	28-APR-23
10	1009	10	Rejected	115	25-APR-23	28-APR-23

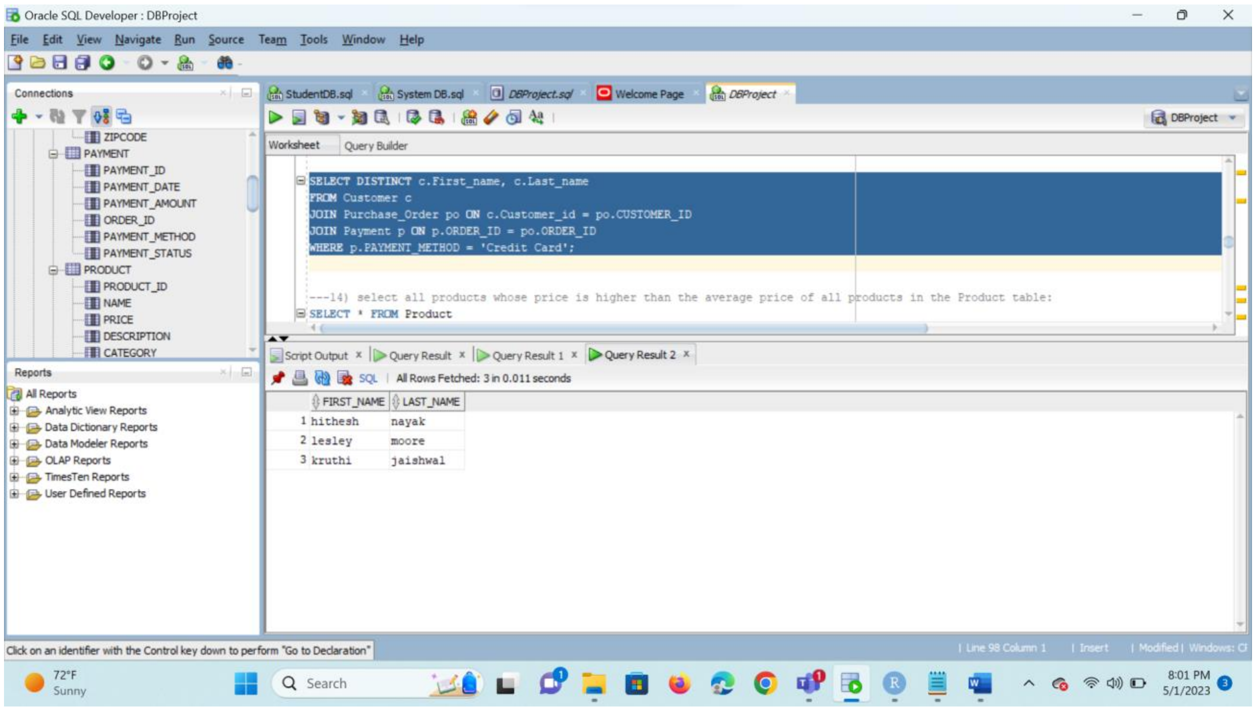


```
UPDATE PURCHASE_ORDER SET ORDER_STATUS = 'Shipped' WHERE ORDER_ID = 1001;

-- Show the PURCHASE_ORDER table contents after the update
SELECT * FROM PURCHASE_ORDER;

-- Rollback the changes so that the data will not be physically removed
ROLLBACK;

---13) find all customers who made a payment with a credit card:
```

Task completed in 0.029 seconds

```
1 row deleted.

>>Query Run In:Query Result 1

Rollback complete.


1 row updated.


Rollback complete.
```

```
---13) find all customers who made a payment with a credit card:

SELECT DISTINCT c.First_name, c.Last_name
FROM Customer c
JOIN Purchase_Order po ON c.Customer_id = po.CUSTOMER_ID
JOIN Payment p ON p.ORDER_ID = po.ORDER_ID
WHERE p.PAYMENT_METHOD = 'Credit Card';
```

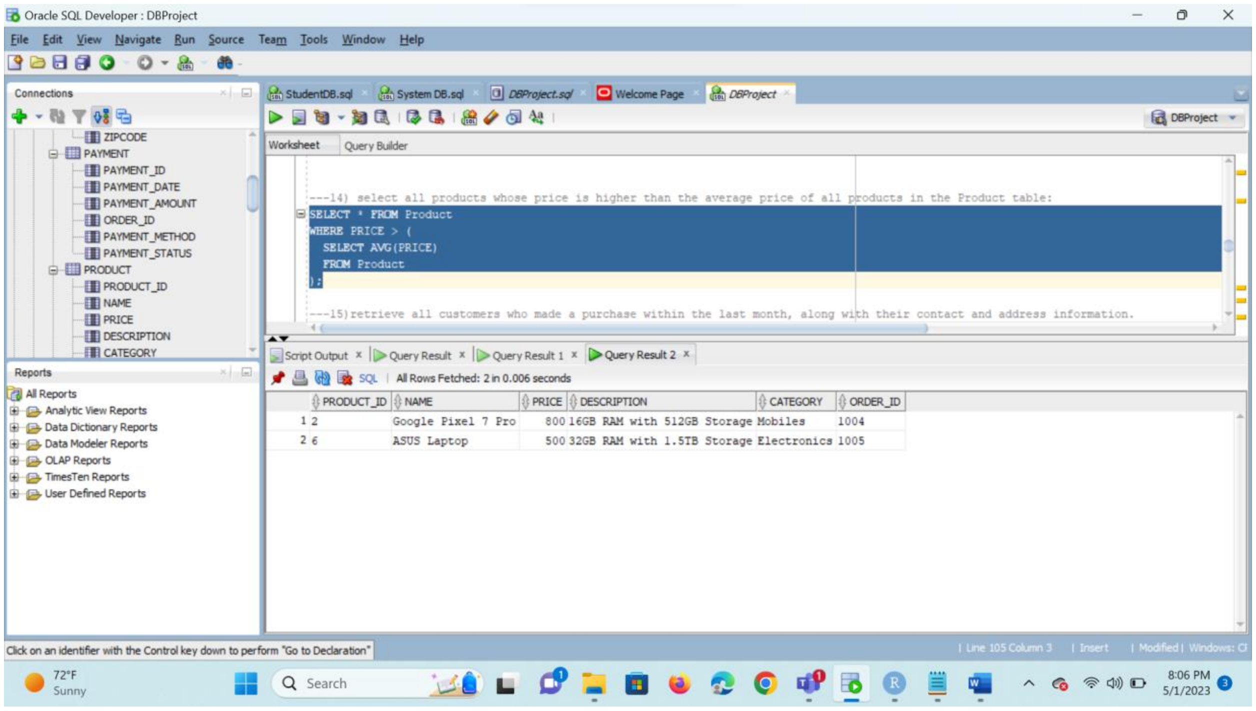


```
SELECT DISTINCT c.First_name, c.Last_name
FROM Customer c
JOIN Purchase_Order po ON c.Customer_id = po.CUSTOMER_ID
JOIN Payment p ON p.ORDER_ID = po.ORDER_ID
WHERE p.PAYMENT_METHOD = 'Credit Card';


---14) select all products whose price is higher than the average price of all products in the Product table:
SELECT * FROM Product
```

All Rows Fetched: 3 in 0.011 seconds

```
   FIRST_NAME  LAST_NAME
 1 hithesh     nayak
 2 lesley      moore
 3 kruthi      jaishwal
```

```
---14) select all products whose price is higher than the average price of
all products in the Product table:
SELECT * FROM Product
WHERE PRICE > (
  SELECT AVG(PRICE)
  FROM Product
);
```

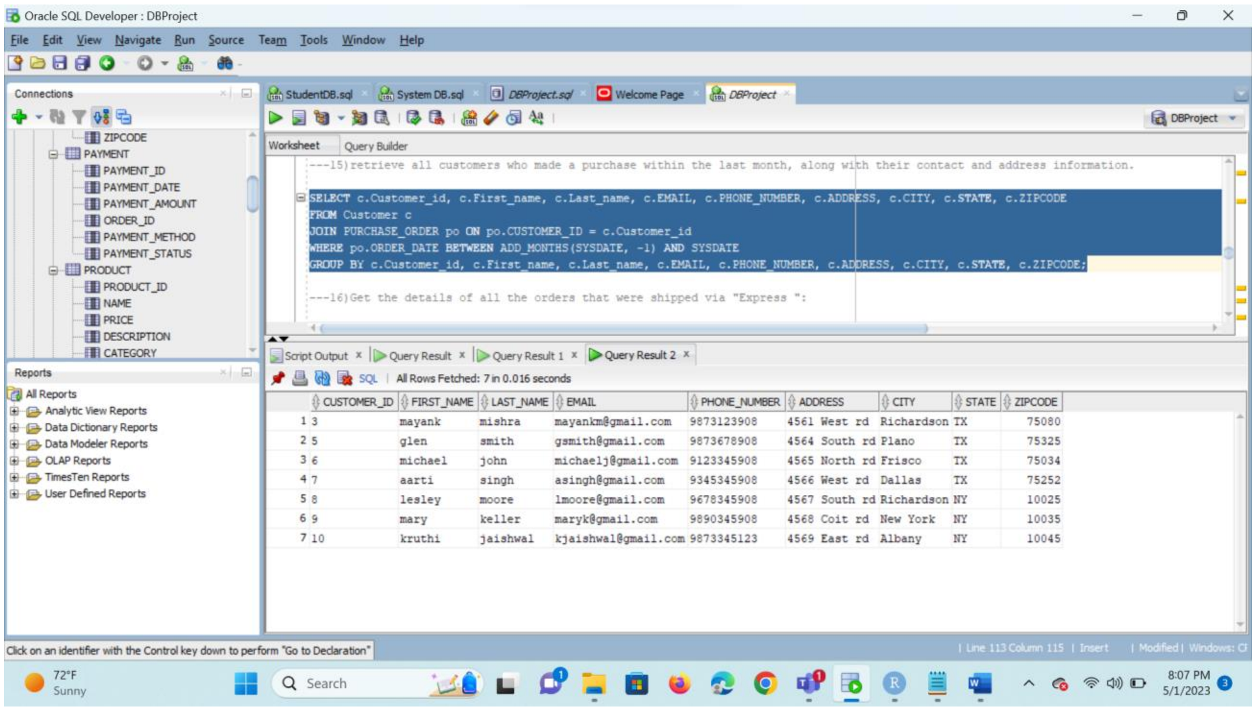


```
---14) select all products whose price is higher than the average price of all products in the Product table:
SELECT * FROM Product
WHERE PRICE > (
  SELECT AVG(PRICE)
  FROM Product
);

---15)retrieve all customers who made a purchase within the last month, along with their contact and address information.
```

All Rows Fetched: 2 in 0.006 seconds

```
   PRODUCT_ID NAME                PRICE DESCRIPTION                   CATEGORY    ORDER_ID
1  2          Google Pixel 7 Pro    800 16GB RAM with 512GB Storage   Mobiles     1004
2  6          ASUS Laptop           500 32GB RAM with 1.5TB Storage   Electronics 1005
```

```
---15)retrieve all customers who made a purchase within the last month, along
with their contact and address information.

SELECT c.Customer_id, c.First_name, c.Last_name, c.EMAIL, c.PHONE_NUMBER,
c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE
FROM Customer c
JOIN PURCHASE_ORDER po ON po.CUSTOMER_ID = c.Customer_id
WHERE po.ORDER_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
GROUP BY c.Customer_id, c.First_name, c.Last_name, c.EMAIL, c.PHONE_NUMBER,
c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE;
```

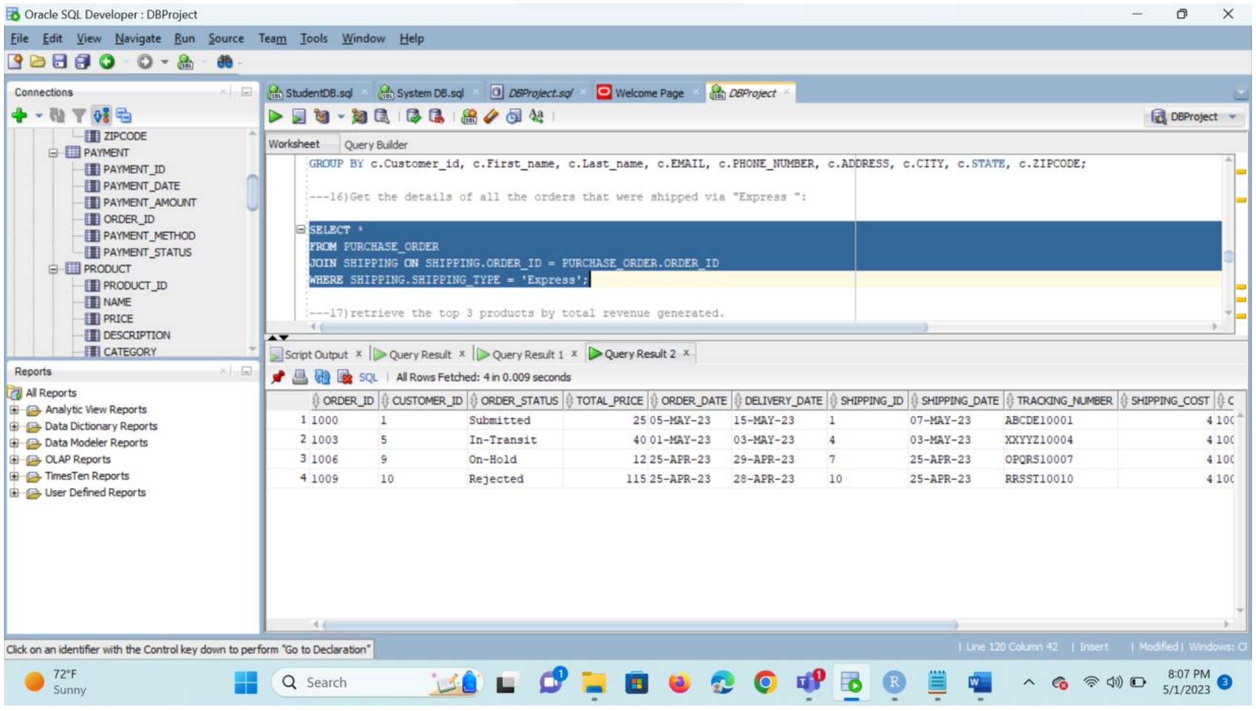


```
---15)retrieve all customers who made a purchase within the last month, along with their contact and address information.

SELECT c.Customer_id, c.First_name, c.Last_name, c.EMAIL, c.PHONE_NUMBER, c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE
FROM Customer c
JOIN PURCHASE_ORDER po ON po.CUSTOMER_ID = c.Customer_id
WHERE po.ORDER_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
GROUP BY c.Customer_id, c.First_name, c.Last_name, c.EMAIL, c.PHONE_NUMBER, c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE;

---16)Get the details of all the orders that were shipped via "Express ":
```

All Rows Fetched: 7 in 0.016 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS	CITY	STATE	ZIPCODE
1	3	mayank	mishra	mayankm@gmail.com	9873123908	4561 West rd	Richardson	TX	75080
2	5	glen	smith	gsmith@gmail.com	9873678908	4564 South rd	Plano	TX	75325
3	6	michael	john	michaelj@gmail.com	9123345908	4565 North rd	Frisco	TX	75034
4	7	aarti	singh	asingh@gmail.com	9345345908	4566 West rd	Dallas	TX	75252
5	8	lesley	moore	lmoore@gmail.com	9678345908	4567 South rd	Richardson	NY	10025
6	9	mary	keller	maryk@gmail.com	9890345908	4568 Coit rd	New York	NY	10035
7	10	kruthi	jaishwal	kjaishwal@gmail.com	9873345123	4569 East rd	Albany	NY	10045

```
---16)Get the details of all the orders that were shipped via "Express ":

SELECT *
FROM PURCHASE_ORDER
JOIN SHIPPING ON SHIPPING.ORDER_ID = PURCHASE_ORDER.ORDER_ID
WHERE SHIPPING.SHIPPING_TYPE = 'Express';
```

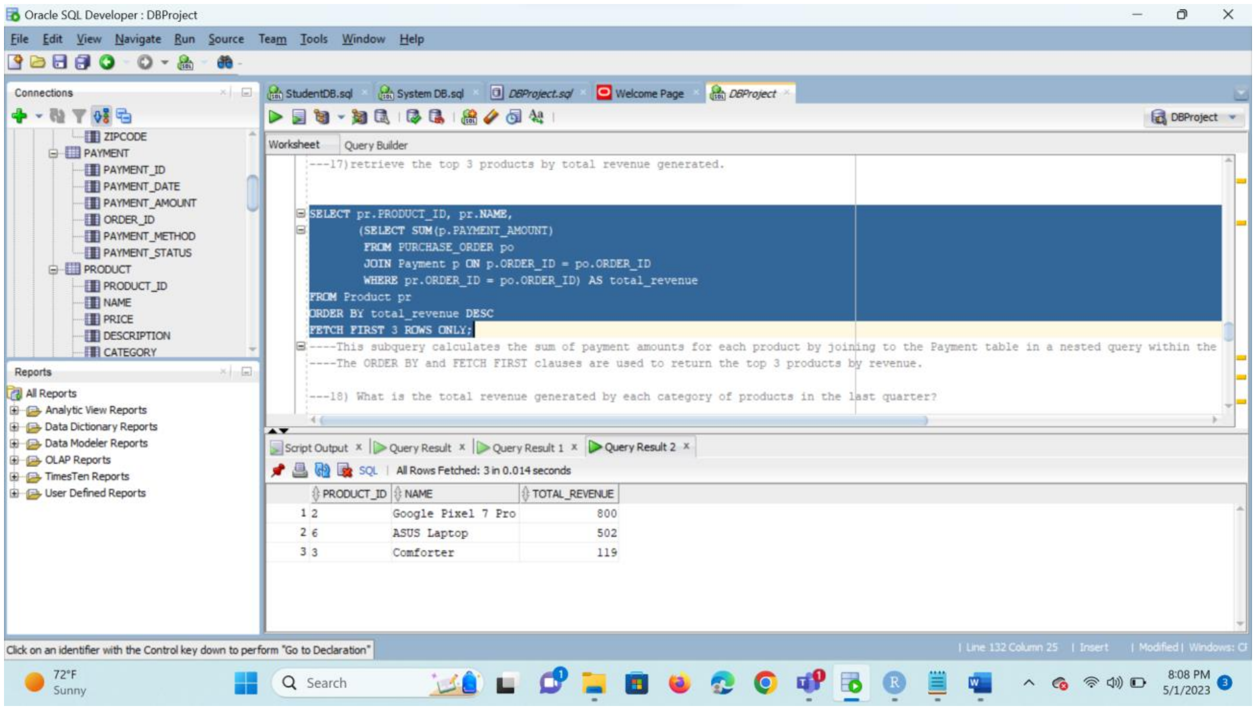


```
GROUP BY c.Customer_id, c.First_name, c.Last_name, c.EMAIL, c.PHONE_NUMBER, c.ADDRESS, c.CITY, c.STATE, c.ZIPCODE;

---16)Get the details of all the orders that were shipped via "Express ":

SELECT *
FROM PURCHASE_ORDER
JOIN SHIPPING ON SHIPPING.ORDER_ID = PURCHASE_ORDER.ORDER_ID
WHERE SHIPPING.SHIPPING_TYPE = 'Express';

---17)retrieve the top 3 products by total revenue generated.
```

All Rows Fetched: 4 in 0.009 seconds

	ORDER_ID	CUSTOMER_ID	ORDER_STATUS	TOTAL_PRICE	ORDER_DATE	DELIVERY_DATE	SHIPPING_ID	SHIPPING_DATE	TRACKING_NUMBER	SHIPPING_COST	C
1	1000	1	Submitted	25	05-MAY-23	15-MAY-23	1	07-MAY-23	ABCDE10001	4	10(
2	1003	5	In-Transit	40	01-MAY-23	03-MAY-23	4	03-MAY-23	XXYYZ10004	4	10(
3	1006	9	On-Hold	12	25-APR-23	29-APR-23	7	25-APR-23	OPQRS10007	4	10(
4	1009	10	Rejected	115	25-APR-23	28-APR-23	10	25-APR-23	RRSST10010	4	10(

```
---17)retrieve the top 3 products by total revenue generated.


SELECT pr.PRODUCT_ID, pr.NAME,
       (SELECT SUM(p.PAYMENT_AMOUNT)
        FROM PURCHASE_ORDER po
        JOIN Payment p ON p.ORDER_ID = po.ORDER_ID
        WHERE pr.ORDER_ID = po.ORDER_ID) AS total_revenue
FROM Product pr
ORDER BY total_revenue DESC
FETCH FIRST 3 ROWS ONLY;
```



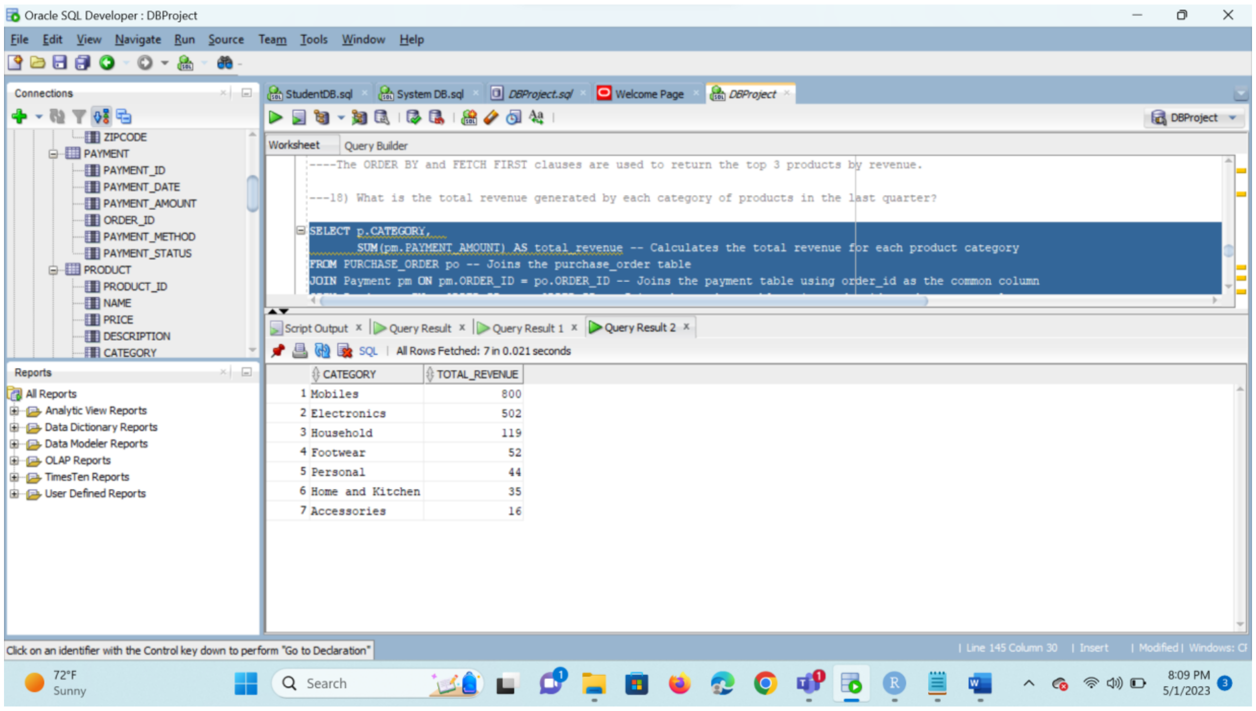
```
---17)retrieve the top 3 products by total revenue generated.


SELECT pr.PRODUCT_ID, pr.NAME,
       (SELECT SUM(p.PAYMENT_AMOUNT)
        FROM PURCHASE_ORDER po
        JOIN Payment p ON p.ORDER_ID = po.ORDER_ID
        WHERE pr.ORDER_ID = po.ORDER_ID) AS total_revenue
FROM Product pr
ORDER BY total_revenue DESC
FETCH FIRST 3 ROWS ONLY;
----This subquery calculates the sum of payment amounts for each product by joining to the Payment table in a nested query within the
----The ORDER BY and FETCH FIRST clauses are used to return the top 3 products by revenue.

---18) What is the total revenue generated by each category of products in the last quarter?
```

All Rows Fetched: 3 in 0.014 seconds

	PRODUCT_ID	NAME	TOTAL_REVENUE
1	2	Google Pixel 7 Pro	800
2	6	ASUS Laptop	502
3	3	Comforter	119

```
----This subquery calculates the sum of payment amounts for each product by
joining to the Payment table in a nested query within the main query.
----The ORDER BY and FETCH FIRST clauses are used to return the top 3
products by revenue.

---18) What is the total revenue generated by each category of products in
the last quarter?

SELECT p.CATEGORY,
       SUM(pm.PAYMENT_AMOUNT) AS total_revenue -- Calculates the total
revenue for each product category
FROM PURCHASE_ORDER po -- Joins the purchase_order table
JOIN Payment pm ON pm.ORDER_ID = po.ORDER_ID -- Joins the payment table using
order_id as the common column
JOIN Product p ON p.ORDER_ID = po.ORDER_ID -- Joins the product table using
order_id as the common column
WHERE po.ORDER_DATE BETWEEN ADD_MONTHS(SYSDATE, -3) AND SYSDATE -- Filters
orders in the last quarter using SYSDATE (current date)
```

```
GROUP BY p.CATEGORY -- Groups the results by product category
ORDER BY total_revenue DESC; -- Orders the results by total revenue in
descending order
```

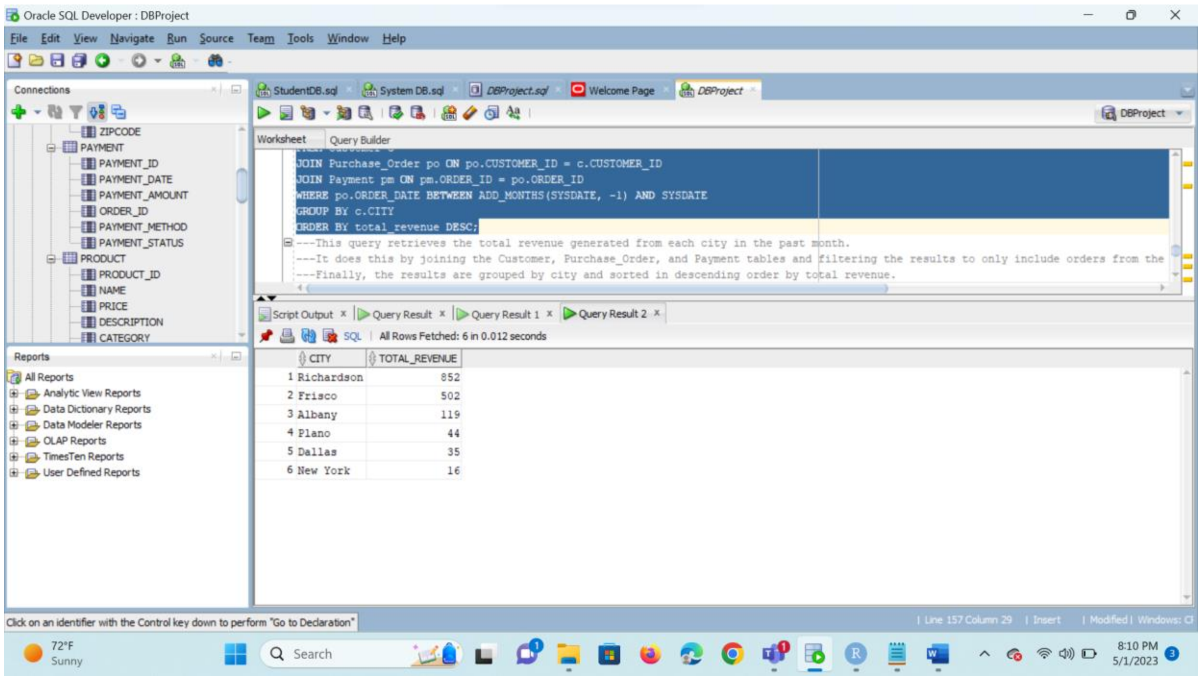


```
----The ORDER BY and FETCH FIRST clauses are used to return the top 3 products by revenue.

---18) What is the total revenue generated by each category of products in the last quarter?

SELECT p.CATEGORY,
       SUM(pm.PAYMENT_AMOUNT) AS total_revenue -- Calculates the total revenue for each product category
FROM PURCHASE_ORDER po -- Joins the purchase_order table
JOIN Payment pm ON pm.ORDER_ID = po.ORDER_ID -- Joins the payment table using order_id as the common column
```

All Rows Fetched: 7 in 0.021 seconds

	CATEGORY	TOTAL_REVENUE
1	Mobiles	800
2	Electronics	502
3	Household	119
4	Footwear	52
5	Personal	44
6	Home and Kitchen	35
7	Accessories	16

```
---19)Retrieve the total revenue generated by each city in the last month

SELECT c.CITY, SUM(pm.PAYMENT_AMOUNT) AS total_revenue
FROM Customer c
JOIN Purchase_Order po ON po.CUSTOMER_ID = c.CUSTOMER_ID
JOIN Payment pm ON pm.ORDER_ID = po.ORDER_ID
WHERE po.ORDER_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
GROUP BY c.CITY
ORDER BY total_revenue DESC;
```



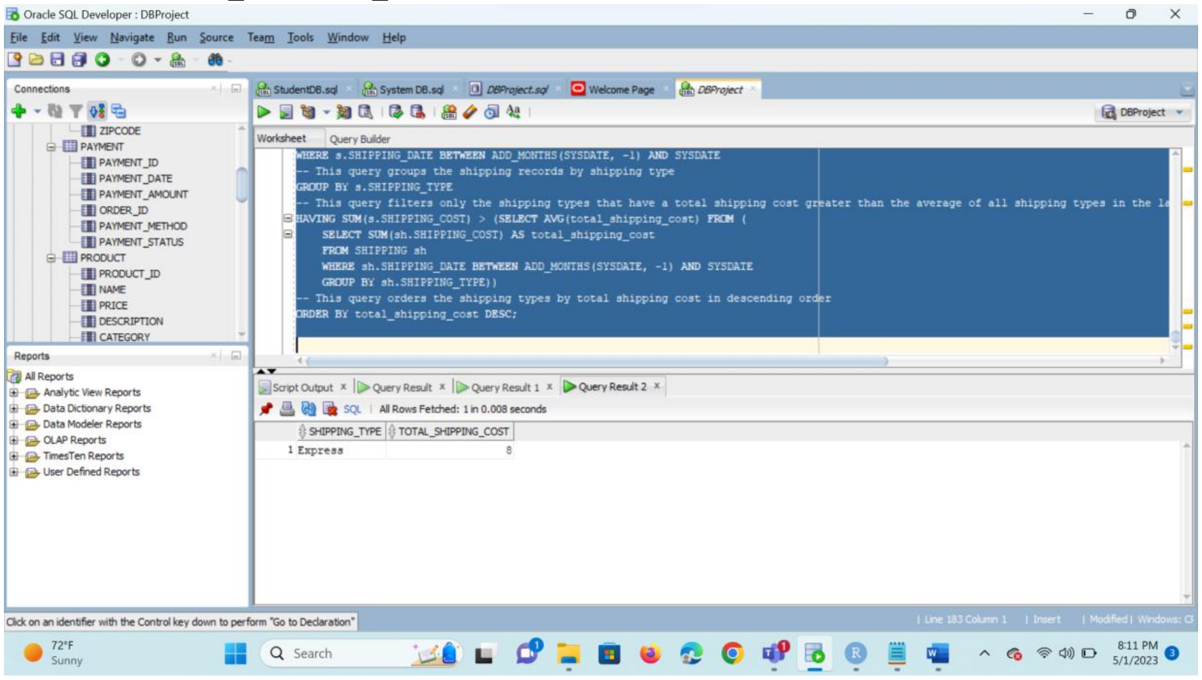
```
JOIN Purchase_Order po ON po.CUSTOMER_ID = c.CUSTOMER_ID
JOIN Payment pm ON pm.ORDER_ID = po.ORDER_ID
WHERE po.ORDER_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
GROUP BY c.CITY
ORDER BY total_revenue DESC;
---This query retrieves the total revenue generated from each city in the past month.
---It does this by joining the Customer, Purchase_Order, and Payment tables and filtering the results to only include orders from the
---Finally, the results are grouped by city and sorted in descending order by total revenue.
```

All Rows Fetched: 6 in 0.012 seconds

	CITY	TOTAL_REVENUE
1	Richardson	852
2	Frisco	502
3	Albany	119
4	Plano	44
5	Dallas	35
6	New York	16

```
---This query retrieves the total revenue generated from each city in the
past month.
---It does this by joining the Customer, Purchase_Order, and Payment tables
and filtering the results to only include orders from the past month using
the BETWEEN clause and ADD_MONTHS function.
---Finally, the results are grouped by city and sorted in descending order by
total revenue.


---20) provide a list of shipping types and their total shipping costs for
the last month,
----and only show those shipping types that have a total shipping cost
greater than the average total shipping cost for all shipping types during
the same period,
----sorted in descending order by total shipping cost?"

-- This query selects the shipping type and the total shipping cost for each
type
SELECT s.SHIPPING_TYPE, SUM(s.SHIPPING_COST) AS total_shipping_cost
-- This query filters only the shipping records that occurred in the last
month
FROM SHIPPING s
WHERE s.SHIPPING_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
-- This query groups the shipping records by shipping type
GROUP BY s.SHIPPING_TYPE
-- This query filters only the shipping types that have a total shipping cost
greater than the average of all shipping types in the last month
HAVING SUM(s.SHIPPING_COST) > (SELECT AVG(total_shipping_cost) FROM (
    SELECT SUM(sh.SHIPPING_COST) AS total_shipping_cost
    FROM SHIPPING sh
    WHERE sh.SHIPPING_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
    GROUP BY sh.SHIPPING_TYPE))
```

```
-- This query orders the shipping types by total shipping cost in descending
order
ORDER BY total_shipping_cost DESC;
```



```
WHERE s.SHIPPING_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
-- This query groups the shipping records by shipping type
GROUP BY s.SHIPPING_TYPE
-- This query filters only the shipping types that have a total shipping cost greater than the average of all shipping types in the la
HAVING SUM(s.SHIPPING_COST) > (SELECT AVG(total_shipping_cost) FROM (
    SELECT SUM(sh.SHIPPING_COST) AS total_shipping_cost
    FROM SHIPPING sh
    WHERE sh.SHIPPING_DATE BETWEEN ADD_MONTHS(SYSDATE, -1) AND SYSDATE
    GROUP BY sh.SHIPPING_TYPE))
-- This query orders the shipping types by total shipping cost in descending order
ORDER BY total_shipping_cost DESC;
```

All Rows Fetched: 1 in 0.008 seconds

	SHIPPING_TYPE	TOTAL_SHIPPING_COST
1	Express	8