MSc Data Science

Coursework Assessment:

CI7320 Database & Data Management

Title:- Database Design

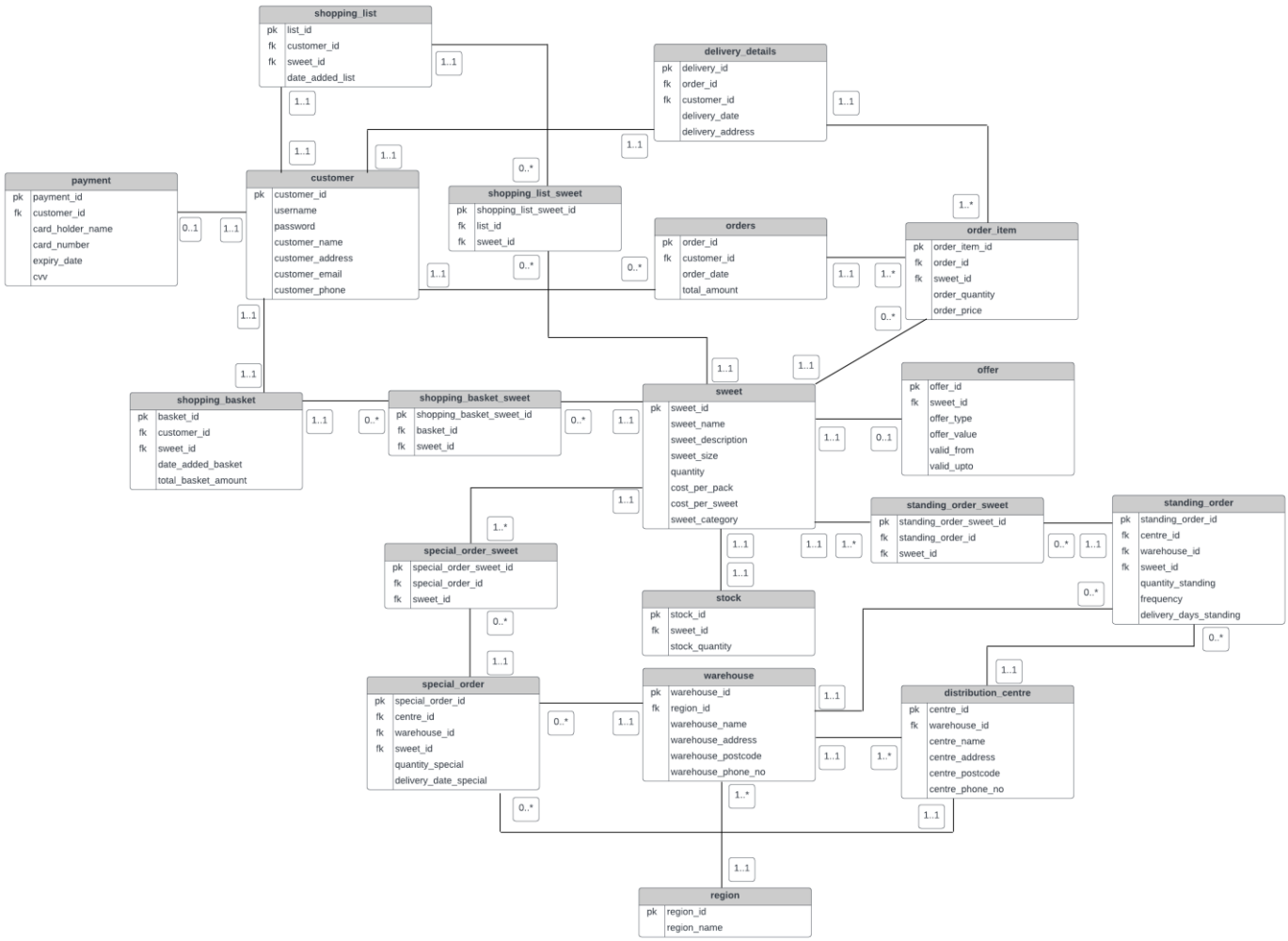
Submitted by:-

Name:- Jay Santosh Dugad

TABLE OF CONTENTS

CHAPTER 1: ENTITY RELATIONSHIP DIAGRAM

WITH CONSTRAINTS & ASSUMPTIONS



Link for ER Diagram

In the above ER diagram PK is referred as Primary Key and FK is referred as Foreign Key.

Constraints: -

- Sweet_id, customer_id, warehouse_id, region_id, centre_id, basket_id, list_id, payment_id, delivery_id, order_id, order_item_id, stock_idoffer_id, shopping_list_sweet_id, shopping_basket_sweet_id, standing_order_sweet_id, special_order_sweet_id should unique and not null.
- In the payment table, card_holder_name, card_number and cvv should be unique.
- Sweets can be stored in multiple sizes and categories.
- Only one offer can be applicable to individual sweet. And one sweet can have only one offer.
- A customer must log in with their username and password before adding items to their shopping basket or list.
- Payment and delivery details are required for order checkout.
- Delivery charges are £5.00 for orders under £50, £10.00 for orders between £50 and £100, and free for orders over £100.
- Items may be delivered separately based on stock availability.

- Delivery is scheduled based on the customer's preferred time during checkout.
- Customers can change their address for delivery before placing order.
- Website accepts only online mode of payment.

Assumptions: -

- Each customer has a unique username and password combination.
- Standing orders have predefined frequencies and delivery days.
- New sweets are promoted on the website before they become available for purchase.
- Regional warehouses keep track of stock levels for each sweet.
- Discounts and offers are applied only to individual sweets, not packs.
- The database system will handle the calculation of delivery dates based on stock availability.
- Customers can add sweets directly to their shopping basket or list from the website interface.
- Shopping basket contents are saved for checkout until the session ends or the order is completed.
- Each order is assigned a unique order ID.
- Payment details include cardholder name, card number, expiry date, and CVV.
- The system maintains historical records of orders for customer tracking purposes.

CHAPTER 2: IMPLEMENTATION

Customer Table

```
CREATE TABLE customer (
    customer_id INT PRIMARY KEY NOT NULL,
    username VARCHAR(50),
    password VARCHAR(50),
    customer_name VARCHAR(100),
    customer_address VARCHAR(255),
    customer_email VARCHAR(100),
    customer_phone VARCHAR(20)
);
```

Sweet Table

```
CREATE TABLE sweet (
    sweet_id INT PRIMARY KEY NOT NULL,
    sweet_name VARCHAR(50),
    sweet_description VARCHAR(50),
    sweet_size VARCHAR(50),
    quantity INT,
```

```
    cost_per_pack DECIMAL(10, 2),
    cost_per_sweet DECIMAL(10, 2),
    sweet_category VARCHAR(50)
    );
```

Region Table

```
CREATE TABLE region (
    region_id INT PRIMARY KEY NOT NULL,
    region_name VARCHAR(50)
);
```

Warehouse Table.

```
CREATE TABLE warehouse (
    warehouse_id INT PRIMARY KEY NOT NULL,
    region_Cid INT,
    warehouse_name VARCHAR(50),
    warehouse_address VARCHAR(100),
    warehouse_postcode VARCHAR(10),
    warehouse_phone_no VARCHAR(15),
    FOREIGN KEY (region_id) REFERENCES region(region_id)
);
```

Distribution Centre Table

```
CREATE TABLE distribution_centre (
    centre_id INT PRIMARY KEY NOT NULL,
    warehouse_id INT,
    centre_name VARCHAR(50),
    centre_address VARCHAR(50),
    centre_postcode VARCHAR(50),
    centre_phone_no VARCHAR(15),
    FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id)
);
```

Orders Table

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY NOT NULL,
    customer_id INT,
    order_date DATE,
```

```
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);
```

Order Item Table

```
CREATE TABLE order_item (
    order_item_id INT PRIMARY KEY NOT NULL,
    order_id INT,
    sweet_id INT,
    order_quantity INT,
    order_price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Shopping Basket Table

```
CREATE TABLE shopping_basket (
    basket_id INT PRIMARY KEY NOT NULL,
    customer_id INT,
    sweet_id INT,
    date_added_basket DATE,
    total_basket_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Shopping List Table

```
CREATE TABLE shopping_list (
    list_id INT PRIMARY KEY NOT NULL,
    customer_id INT,
    sweet_id INT,
    date_added_list DATE,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Stock Table

```
CREATE TABLE stock (
    stock_id INT PRIMARY KEY NOT NULL,
    sweet_id INT,
    stock_quantity INT,
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Special Order Table

```
CREATE TABLE special_order (
    special_order_id INT PRIMARY KEY NOT NULL,
    centre_id INT,
    warehouse_id INT,
    sweet_id INT,
    quantity_special INT,
    delivery_date_special DATE,
    FOREIGN KEY (centre_id) REFERENCES distribution_centre(centre_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Standing Order Table

```
CREATE TABLE standing_order (
    standing_order_id INT PRIMARY KEY NOT NULL,
    centre_id INT,
    warehouse_id INT,
    sweet_id INT,
    quantity_standing INT,
    frequency VARCHAR(20),
    delivery_days_standing VARCHAR(15) ,
    FOREIGN KEY (centre_id) REFERENCES distribution_centre(centre_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Delivery Details Table

```
CREATE TABLE delivery_details (
    delivery_id INT PRIMARY KEY NOT NULL,
    order_id INT,
    customer_id INT,
    delivery_date DATE,
    delivery_address VARCHAR(100),
    payment_type VARCHAR(50),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);
```

Offer Table

```
CREATE TABLE offer (
    offer_id INT PRIMARY KEY NOT NULL,
    sweet_id INT,
    offer_type VARCHAR(100),
    offer_value VARCHAR(50),
    valid_from DATE,
    valid_upto DATE,
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Payment Table

```
CREATE TABLE payment (
    payment_id NUMBER PRIMARY KEY NOT NULL,
    customer_id NUMBER,
    card_holder_name VARCHAR2(50) NOT NULL,
    card_number VARCHAR2(20) NOT NULL,
    expiry_date DATE NOT NULL,
    cvv NUMBER NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);
```

Special Order Sweet Table *(Junction Table)*

```
CREATE TABLE special_order_sweet (
    special_order_sweet_id INT PRIMARY KEY NOT NULL,
    special_order_id INT,
    sweet_id INT,
    FOREIGN KEY (special_order_id) REFERENCES special_order(special_order_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Standing Order Sweet Table *(Junction Table)*

```
CREATE TABLE standing_order_sweet (
    standing_order_sweet_id INT PRIMARY KEY NOT NULL,
    standing_order_id INT,
    sweet_id INT,
    FOREIGN KEY (standing_order_id) REFERENCES standing_order(standing_order_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Shopping List Sweet Table *(Junction Table)*

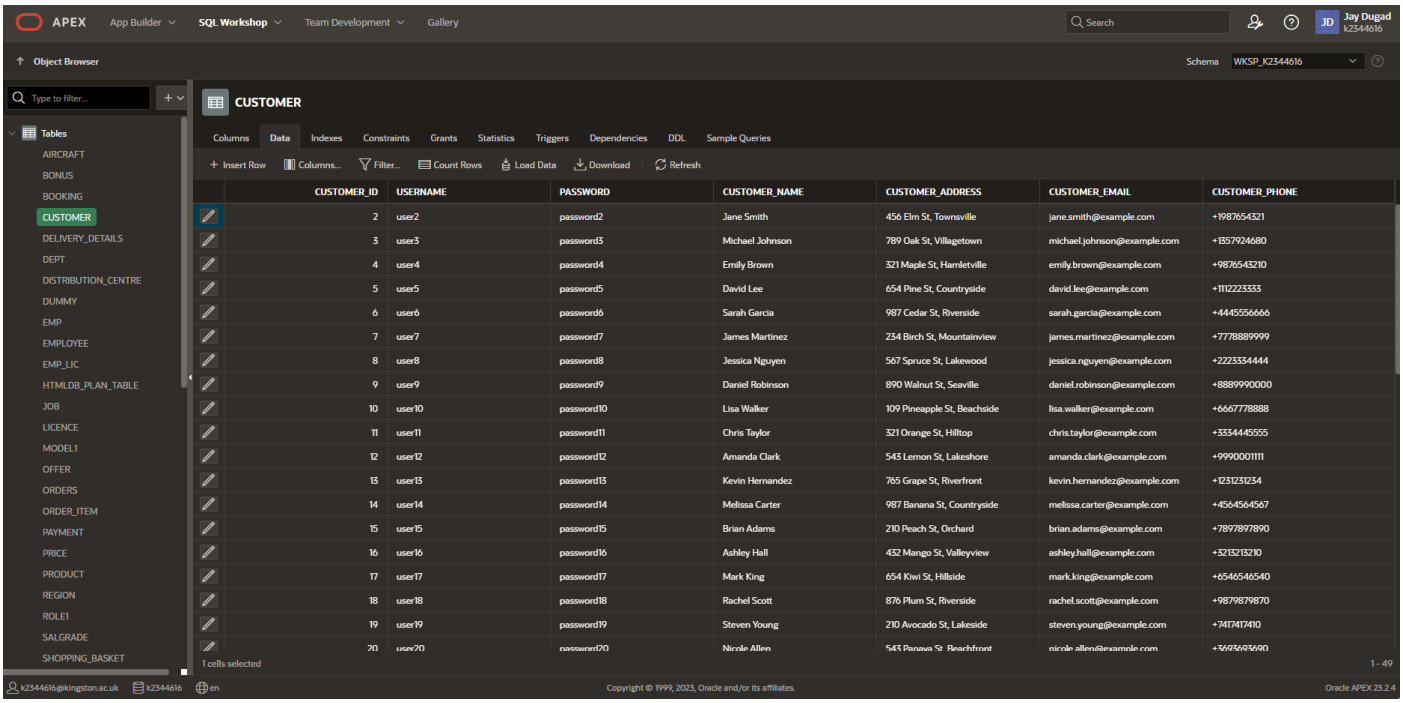
```
CREATE TABLE shopping_list_sweet (
    shopping_list_sweet_id INT PRIMARY KEY NOT NULL,
    list_id INT,
    sweet_id INT,
    FOREIGN KEY (list_id) REFERENCES shopping_list(list_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

Shopping Basket Sweet Table *(Junction Table)*

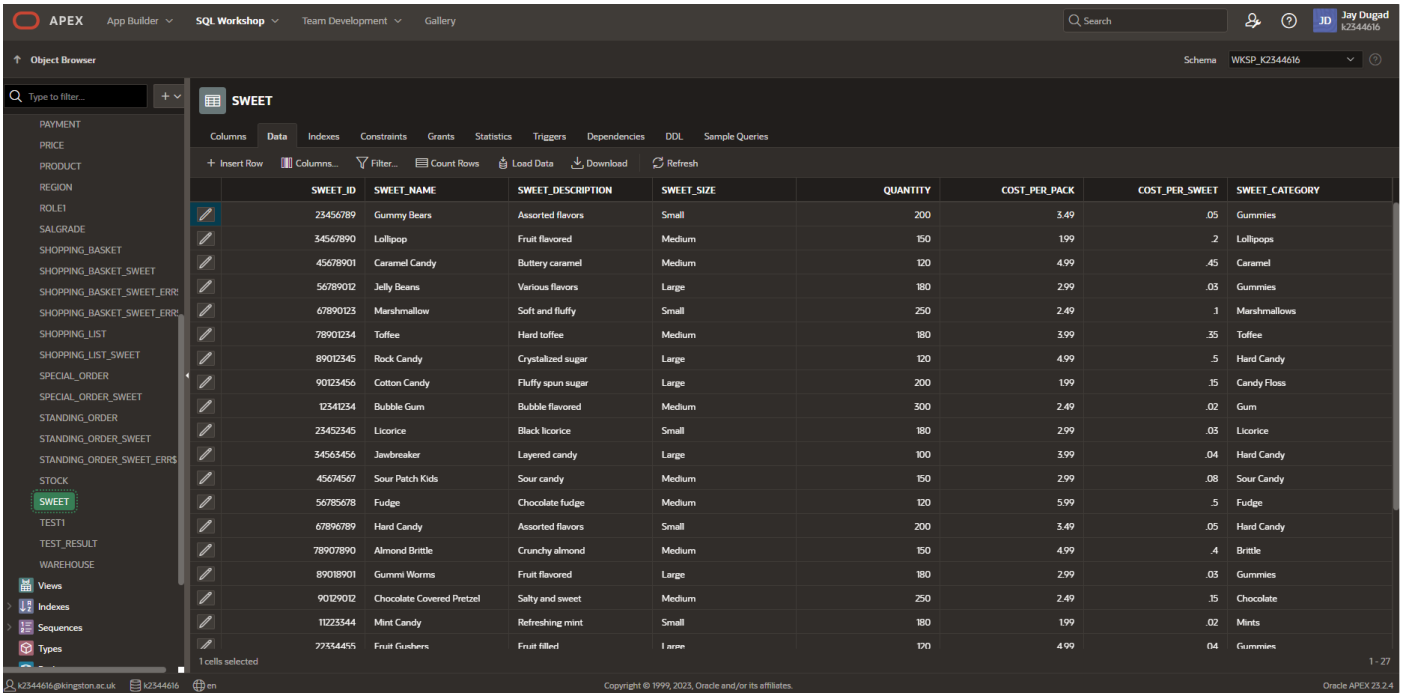
```
CREATE TABLE shopping_basket_sweet (
    shopping_basket_sweet_id INT PRIMARY KEY NOT NULL,
    basket_id INT,
    sweet_id INT,
    FOREIGN KEY (basket_id) REFERENCES shopping_basket(basket_id),
    FOREIGN KEY (sweet_id) REFERENCES sweet(sweet_id)
);
```

CHAPTER 3: MULTIPLICITY OF THE RELATIONSHIP IN THE TABLES

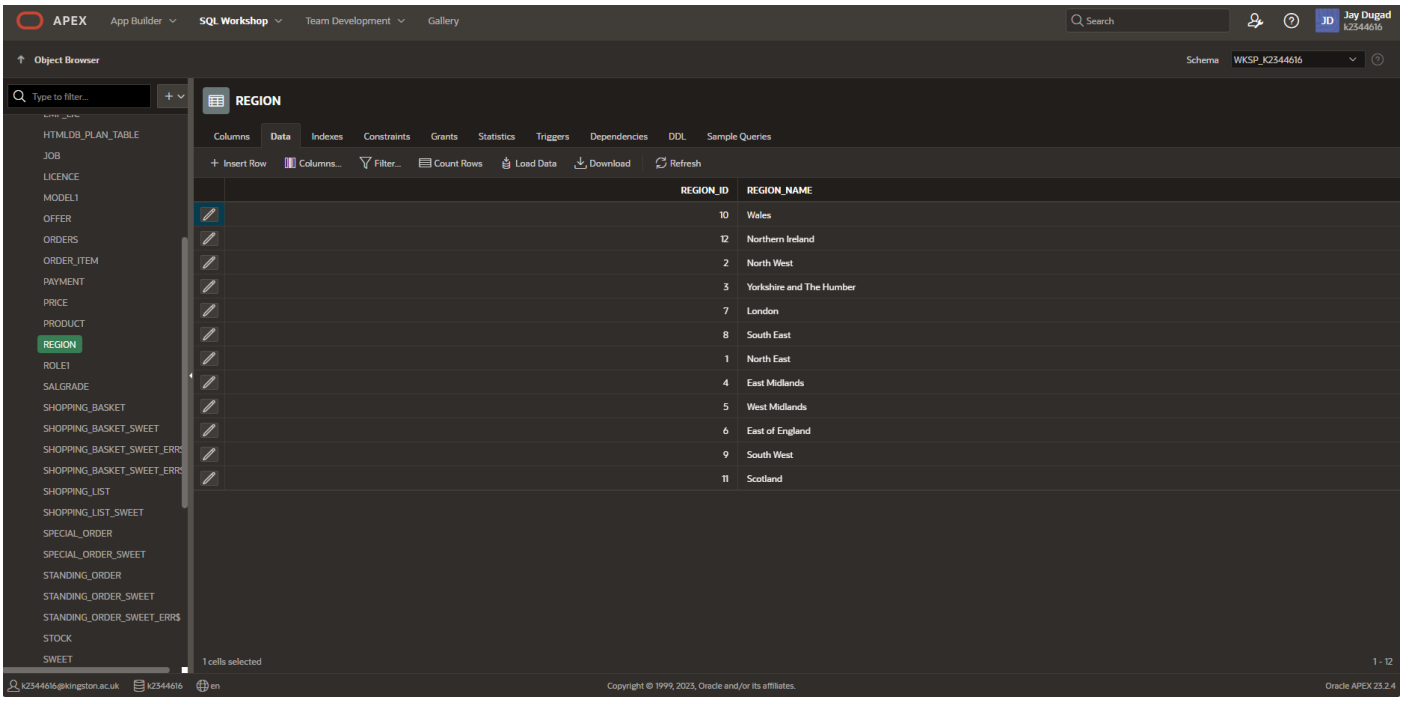
- **Customer Table: -** A customer table consists of customer_id as primary key, username, password, customer_name, customer_address, customer_email, customer_phone. This table consists of 48 rows of data. Here all the data entered is unique and not repeated.



- **Sweet Table: -** A sweet table consists of sweet_id as primary key, sweet_name, sweet_description, sweet_size, quantity, cost_per_pack, cost_per_sweet, sweet_category. These table contains total 44 rows of data.

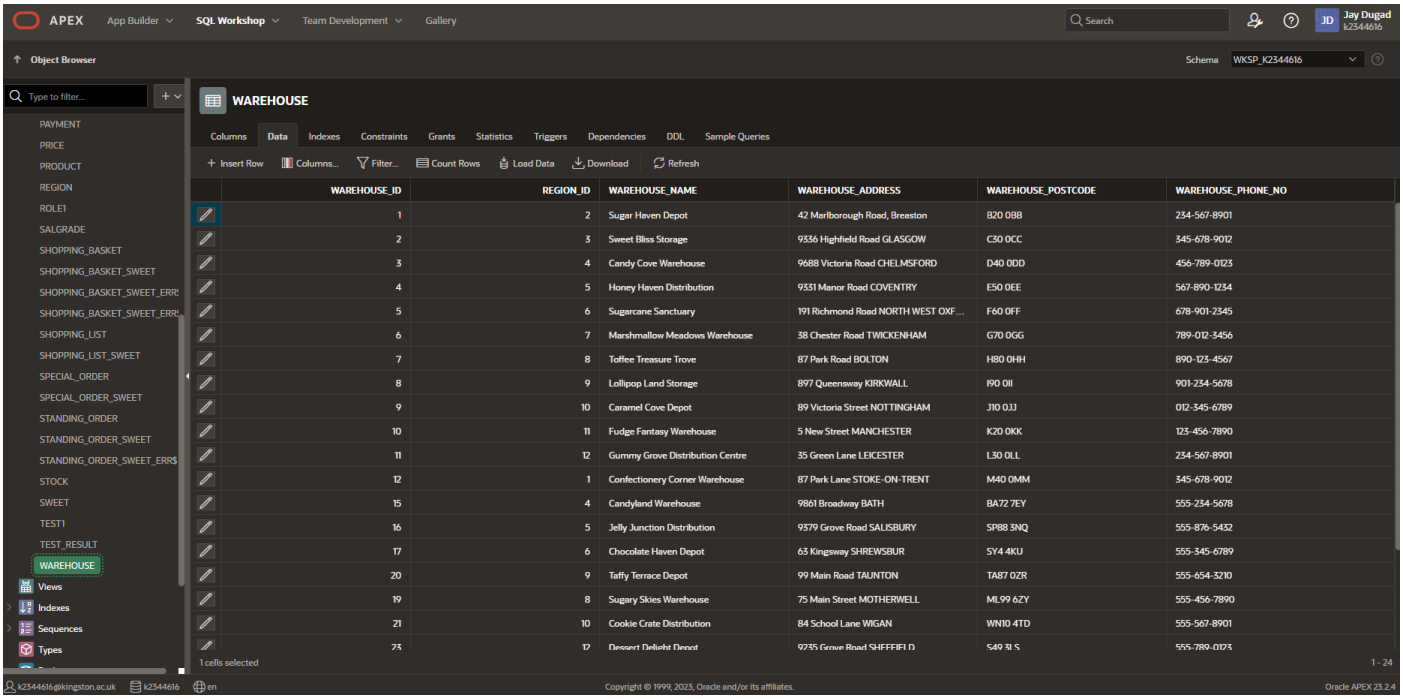


- **Region Table: -** A region table consists of region_id as primary key and region_name. There are total 12 regions.



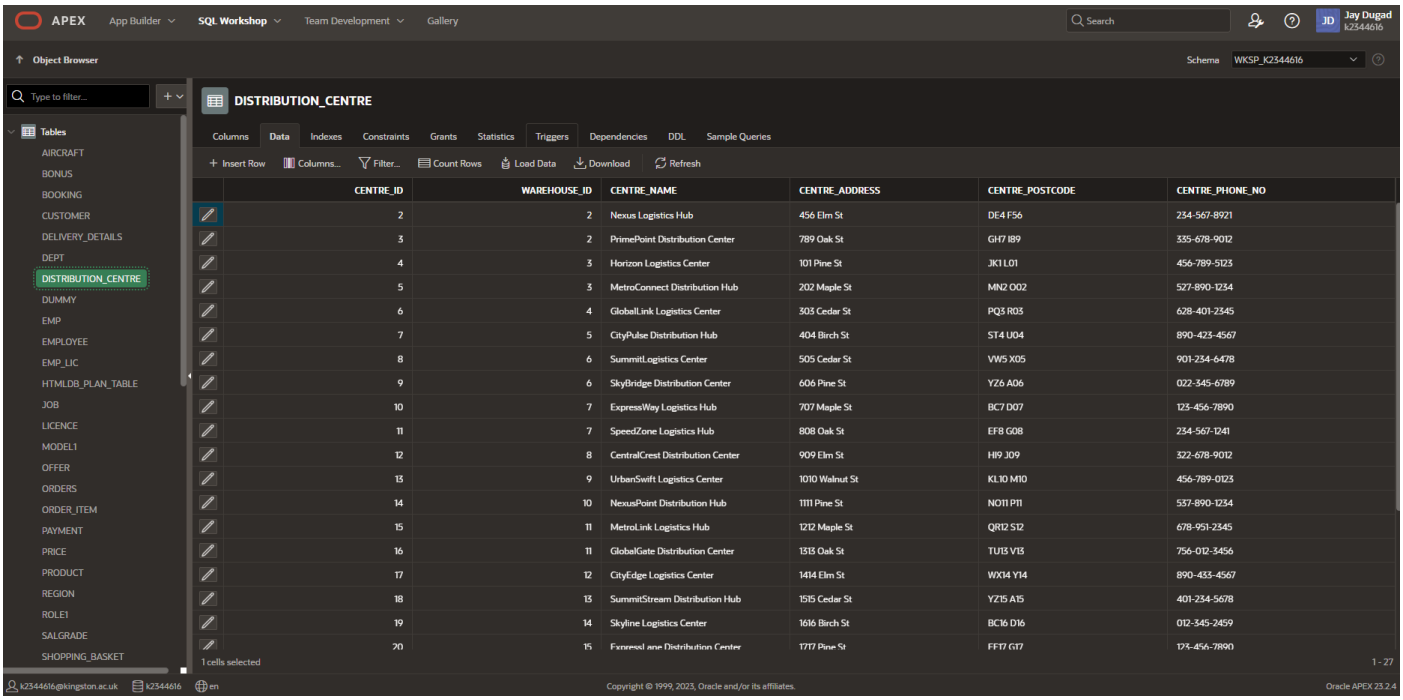
- **Warehouse Table: -** A warehouse consists of only one and one region whereas a region can have many warehouses [region (1..1)-----(1..*) warehouse]

The given figure demonstrates that how region_id 4 and 5 contains multiple warehouse_id which are 3,15 and 4,16.



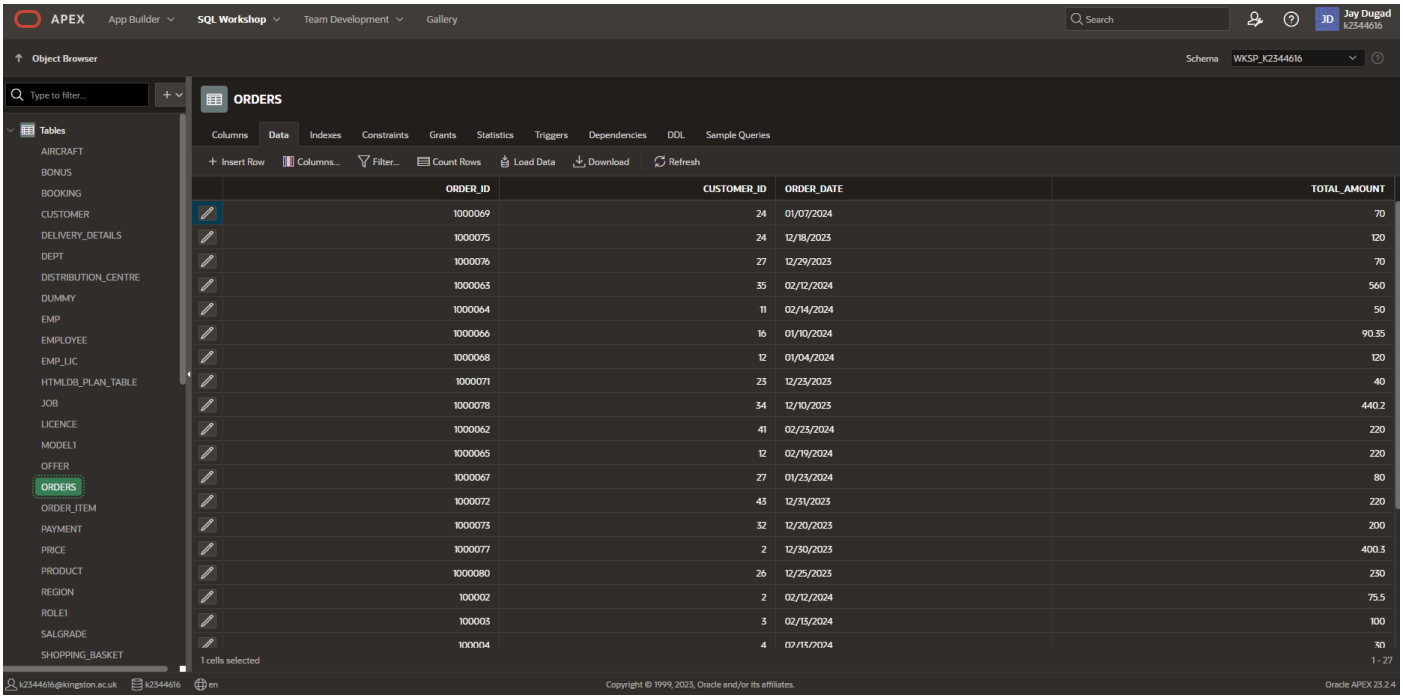
- **Distribution Centre Table: -** A distribution centre consists of only one and one warehouse whereas a warehouse can have many distributions centre [warehouse (1..1)-----(1..*) distribution_centre]

The given figure demonstrates that how warehouse_id 3 and 6 contains more than one centre_id which are 4, 5 and 8,9.



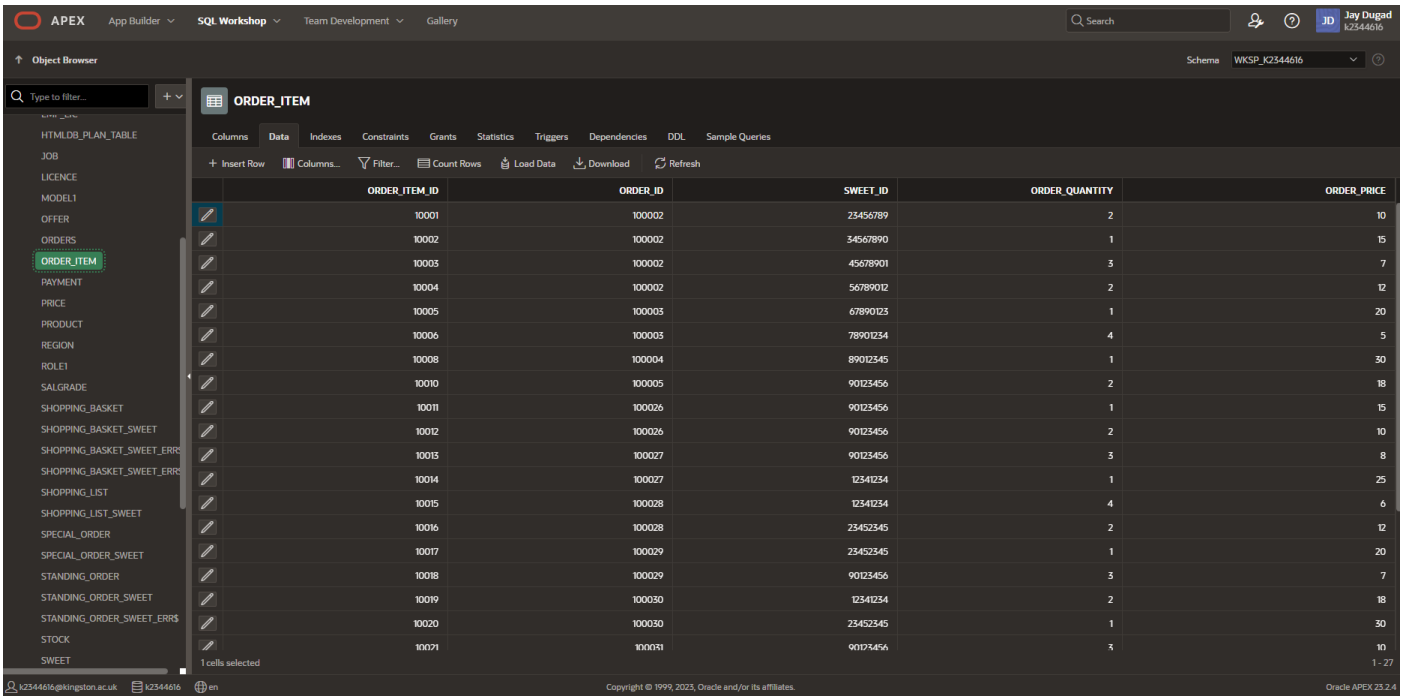
- **Orders Table: -** An order_id consists of only one and one customer whereas a customer can place zero or multiple orders [orders (0..*)-----(1..1) customer]

The given figure demonstrates that how customer_id 24 and 2 contains many orders which are order_id 1000069,1000075 and 1000077,1000002.



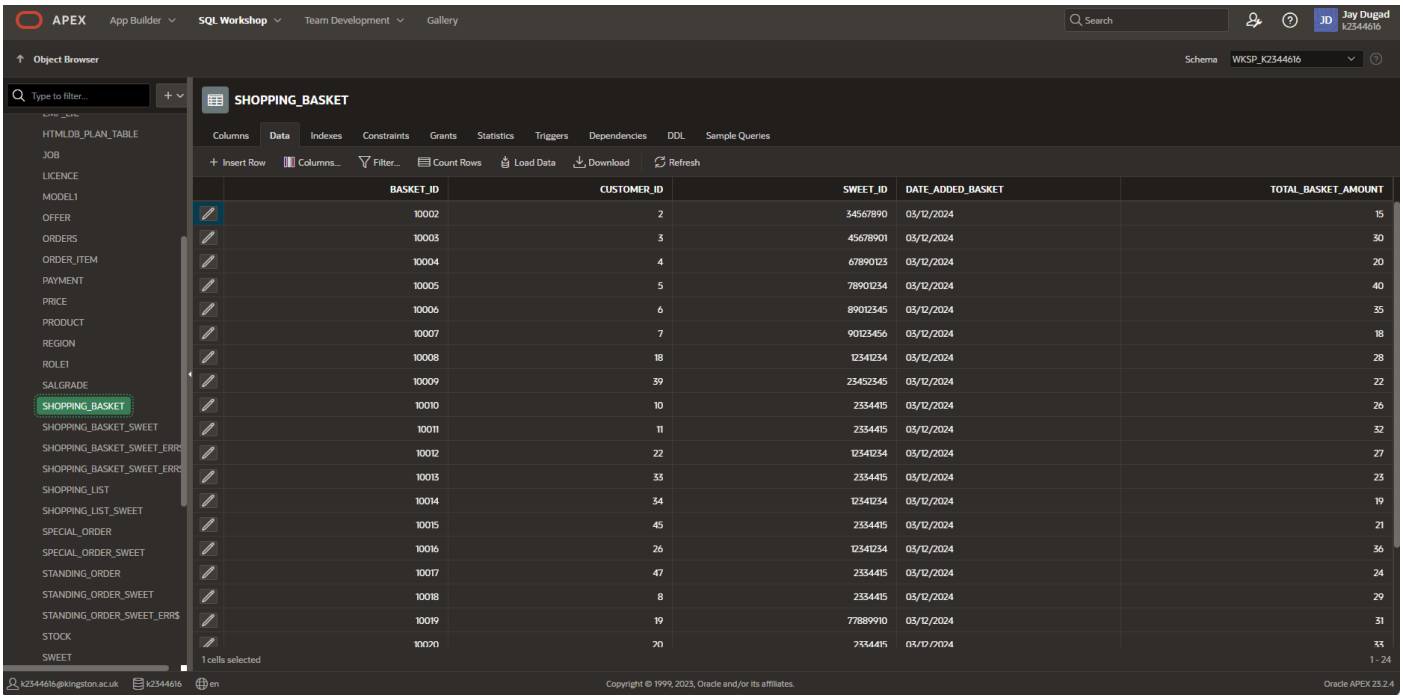
- **Order Item Table: -** An order_item_id consists of only one and one sweet_id whereas a sweet_id can be used in zero or multiple order items [order_item (0..1)-----(1..1) sweet]

The given figure demonstrates that how sweet_id 90123456 and 23452345 appears in many order_item_id which are 10010, 10011, 10012, 10013, 10018 and 10016, 10017, 10020.



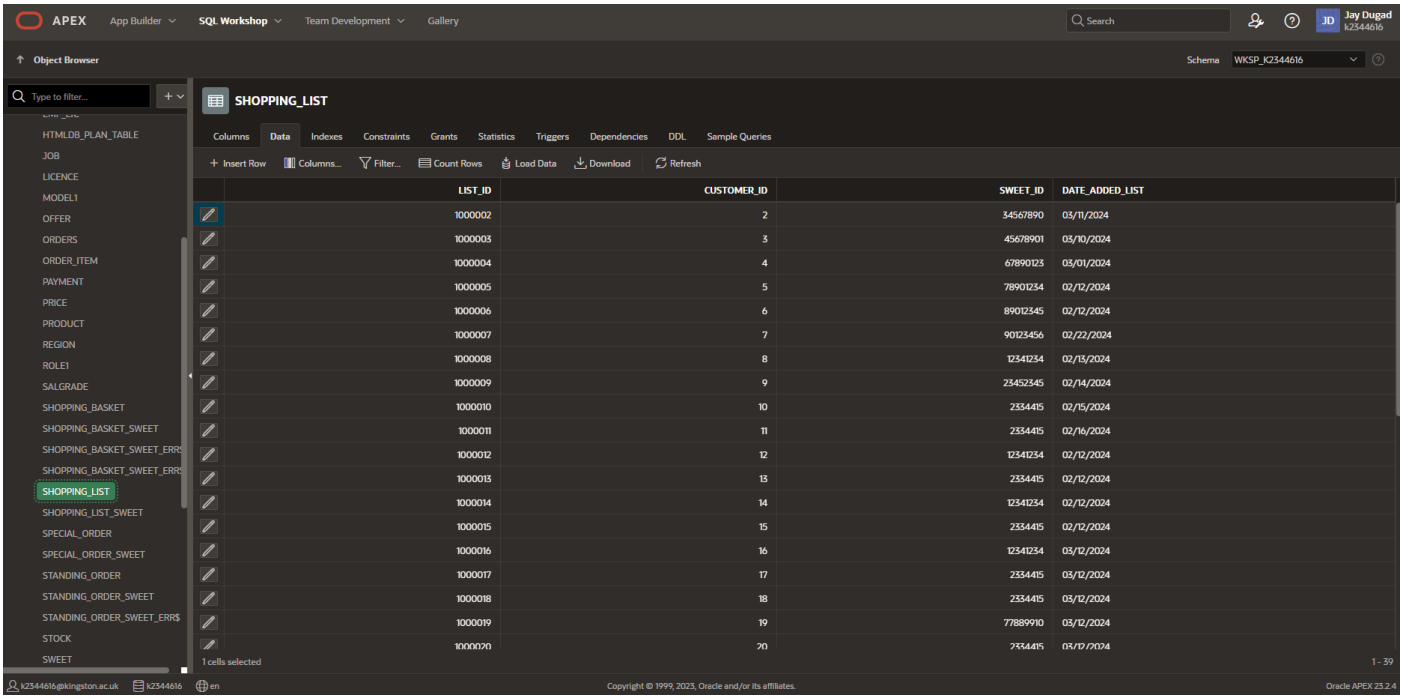
- **Shopping Basket Table: -** A shopping basket consists of only one and one customer_id and a customer can have only one shopping basket [shopping_basket (1..1)-----(1..1) customer]

The given figure demonstrates that basket_id for every customer_id is different.



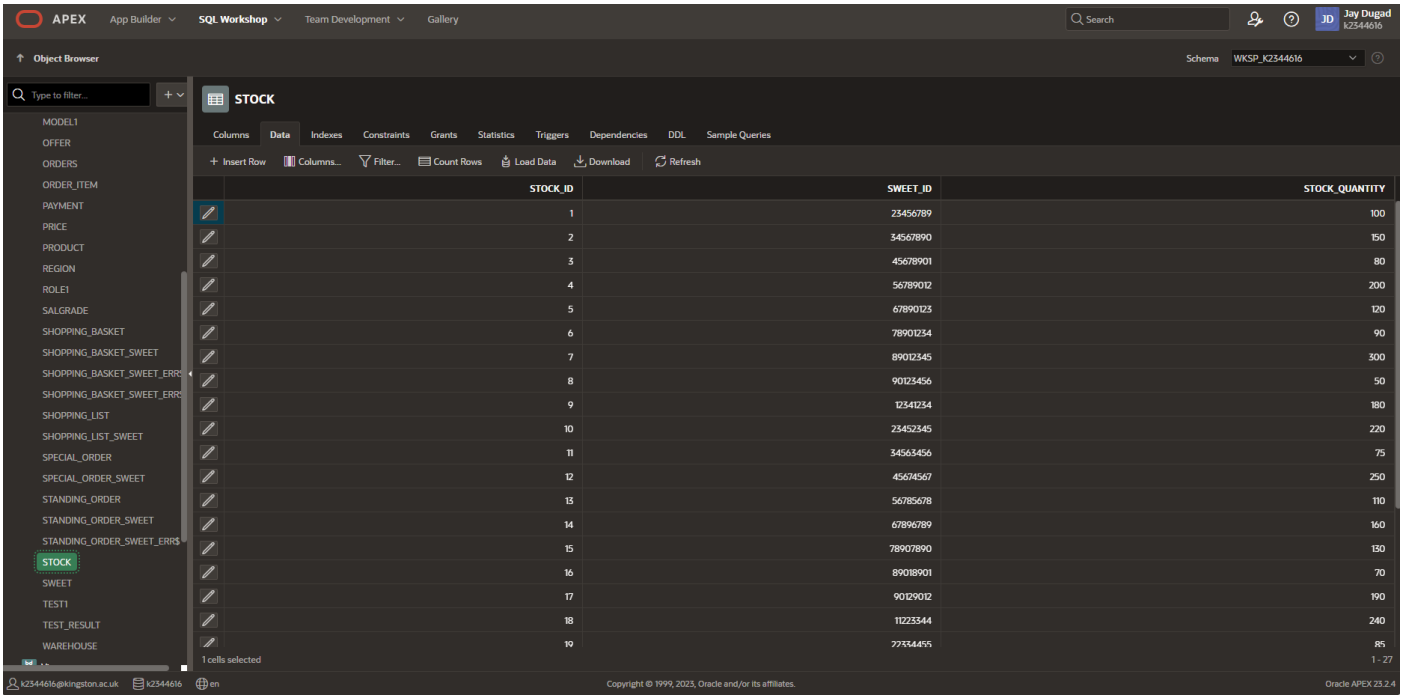
- **Shopping List Table: -** A shopping list consists of only one and one customer_id and a customer can have only one shopping list [shopping_list (1..1)-----(1..1) customer]

The given figure demonstrates that list_id for every customer_id is different, and both are not repeated.



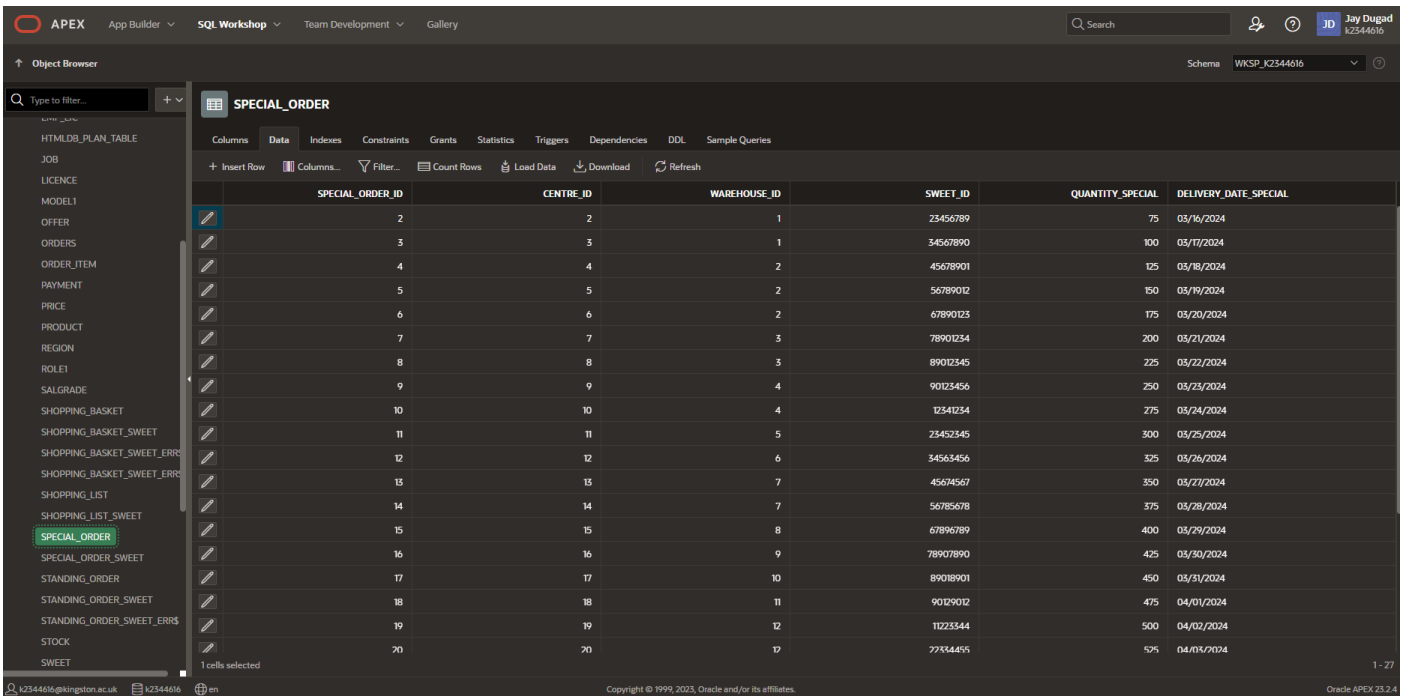
- **Stock Table: -** A stock_id belongs to only one sweet_id and a sweet_id addresses only one stock_id. [stock (1..1)-----(1..1) sweet]

The given figure demonstrates that stock_id for every sweet_id is different, and both are not repeated.



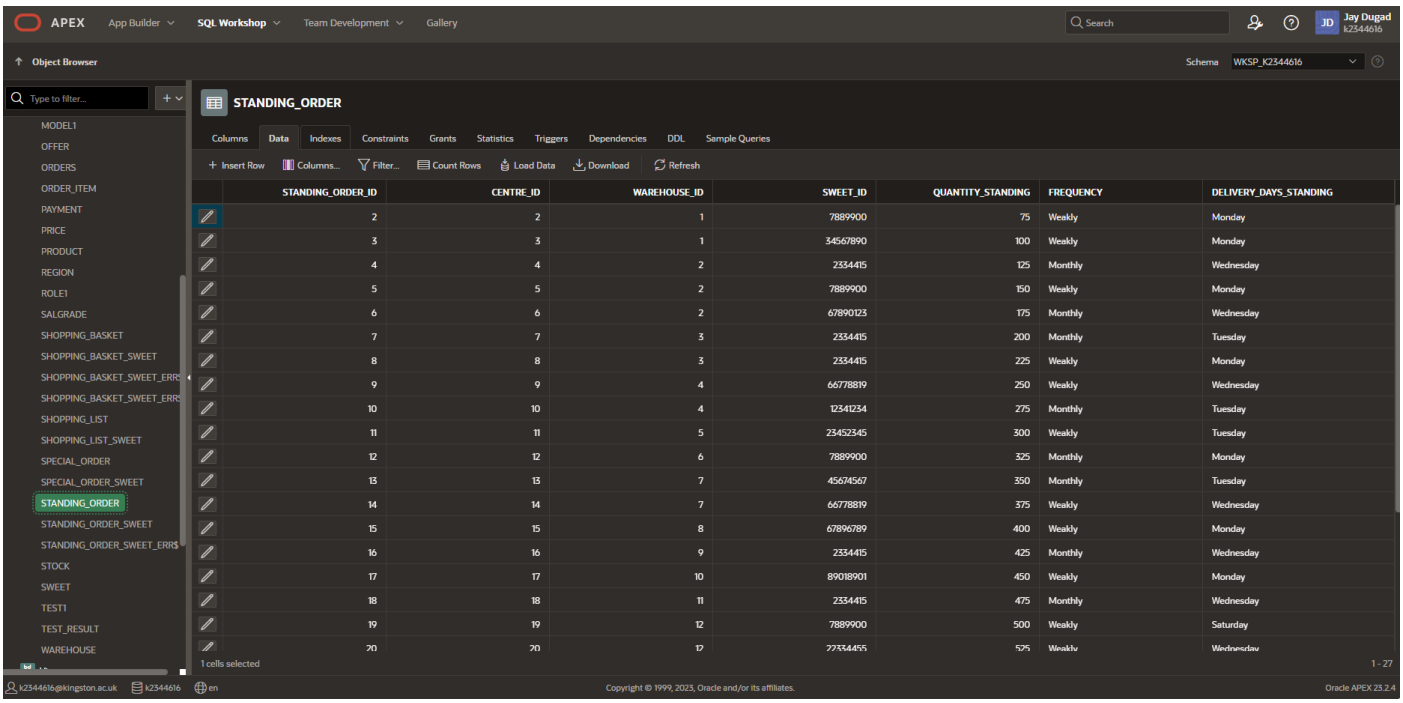
- **Special Order Table: -** A special order can be placed to only one warehouse whereas a warehouse can have multiple special orders. [special_order (0..*)-----(1..1) warehouse]

The given figure demonstrates that different special_order_id like 4, 5, 6 are placed to same warehouse_id that is 2.



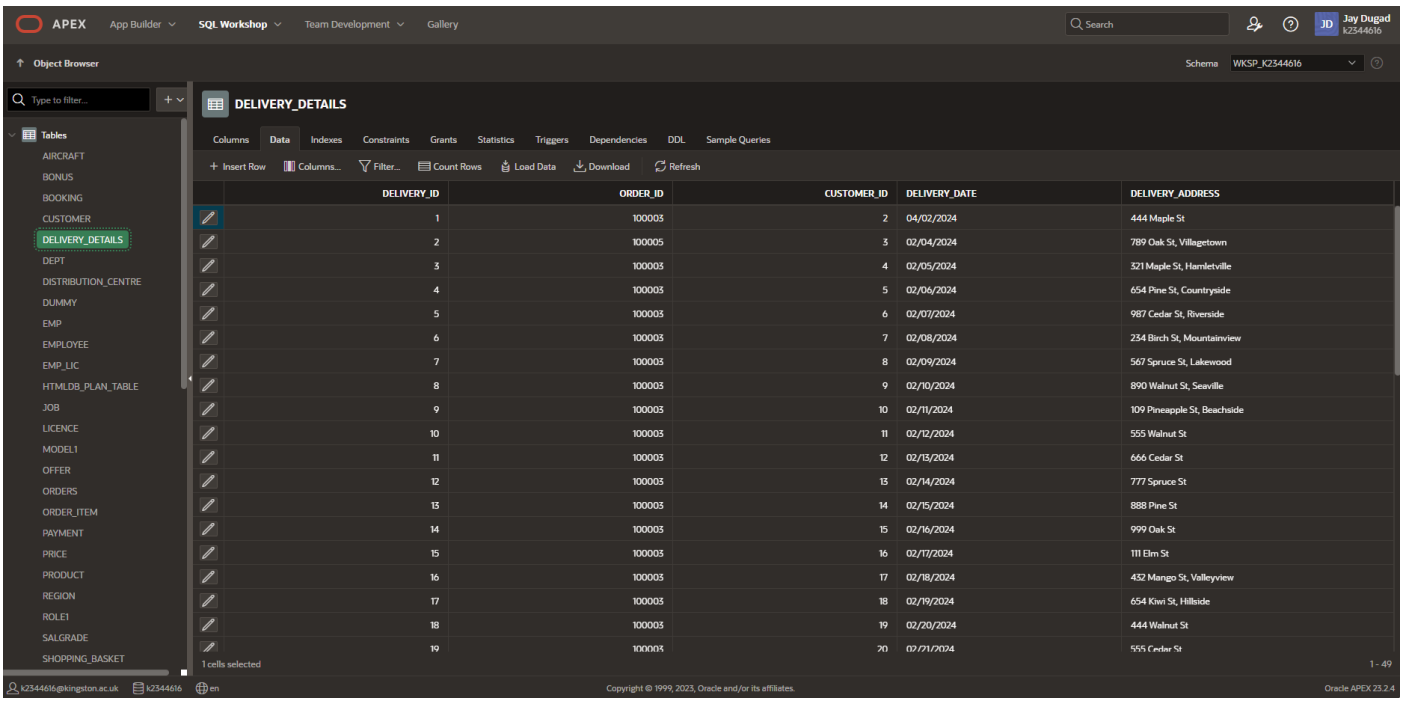
- **Standing Order Table: -** A standing order can be placed to only one warehouse whereas a warehouse can have multiple standing orders. [standing_order (0..*)-----(1..1) warehouse]

The given figure demonstrates that different standing_order_id like 19, 20 are placed to same warehouse_id that is 12.



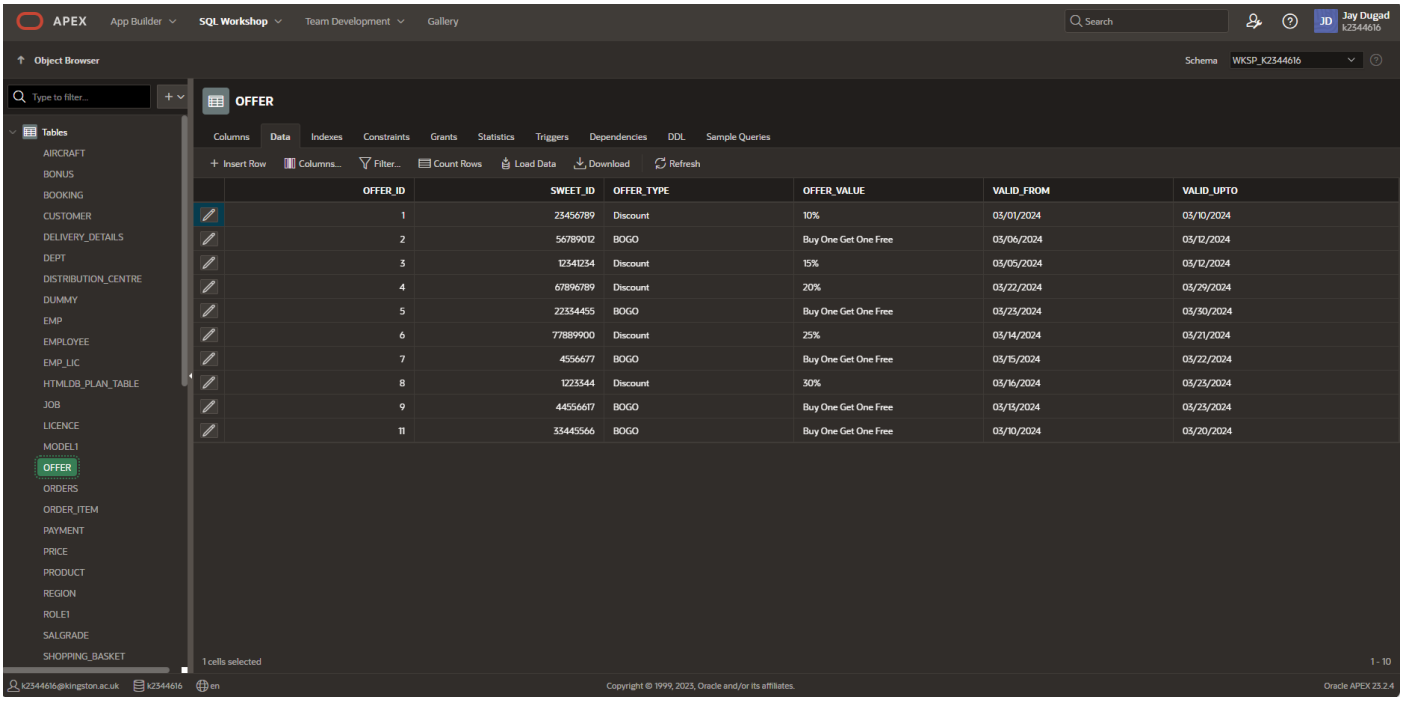
- **Delivery Details Table: -** Here a delivery_id can be assigned to only and only one customer and a customer can store only one delivery details at a time i.e., customer can save only one delivery address.[delivery_details(1..1)-----(1..1)customer].

In the diagram below every delivery_id is assigned to different customer_id. Both are unique for each data.

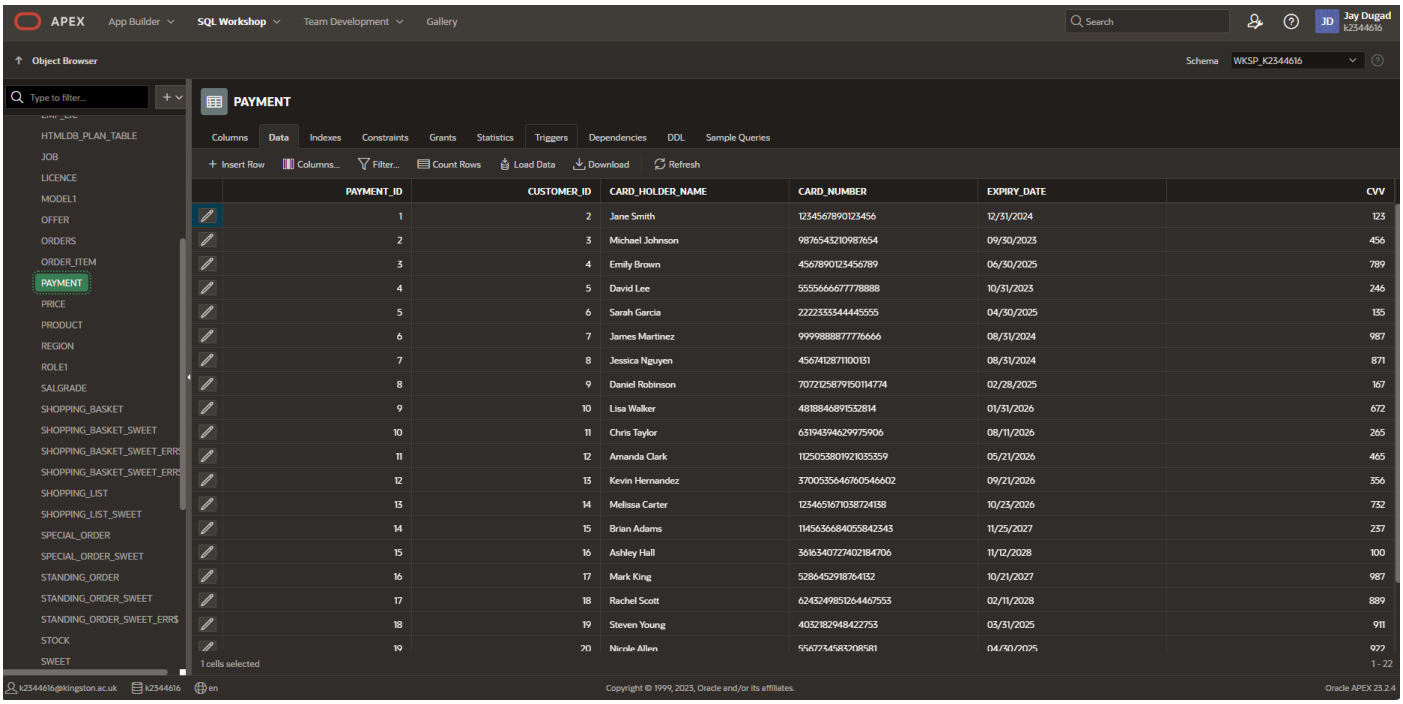


- **Offer Table: -** An offer can be placed to only one sweet and a sweet can have zero or one offers. [offer (0..1)-----(1..1) sweet]

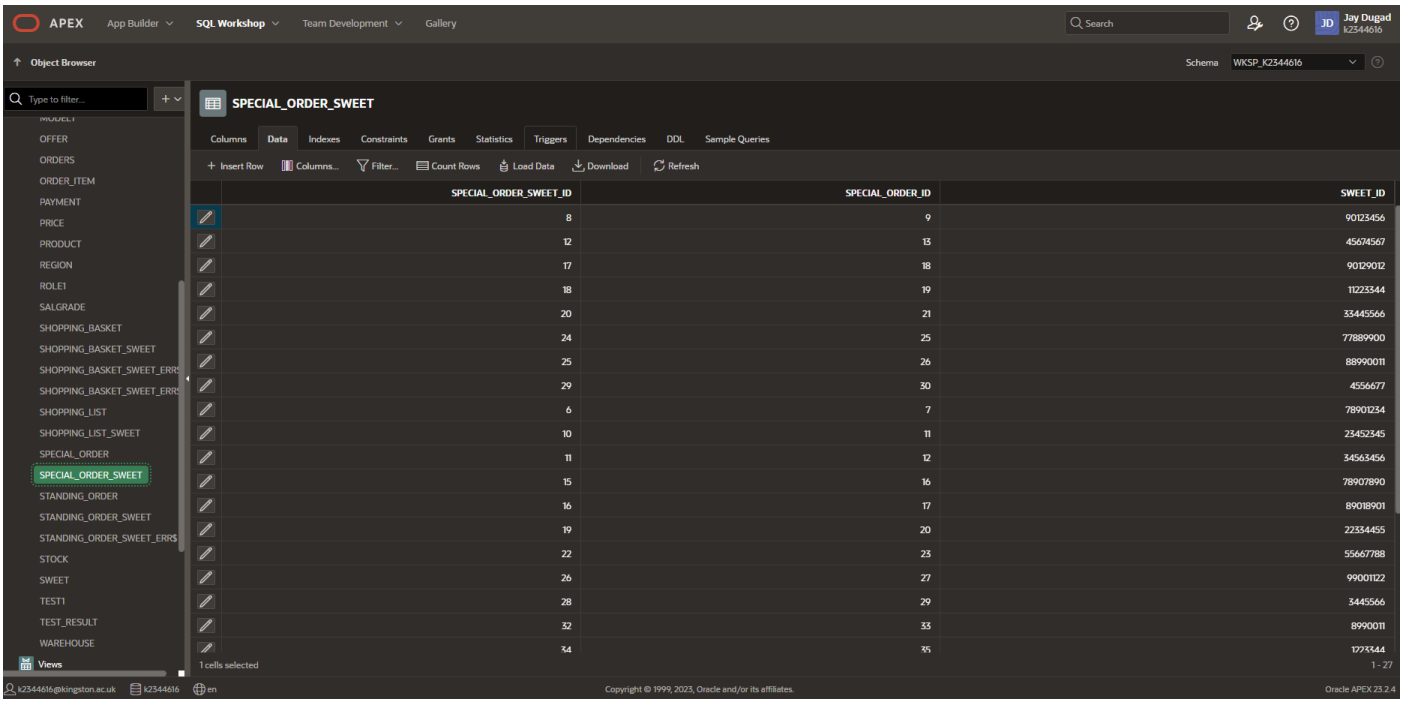
The given figure demonstrates that different offer_id are applied to different sweet_id. Here both sweet_id and offer_id are unique.



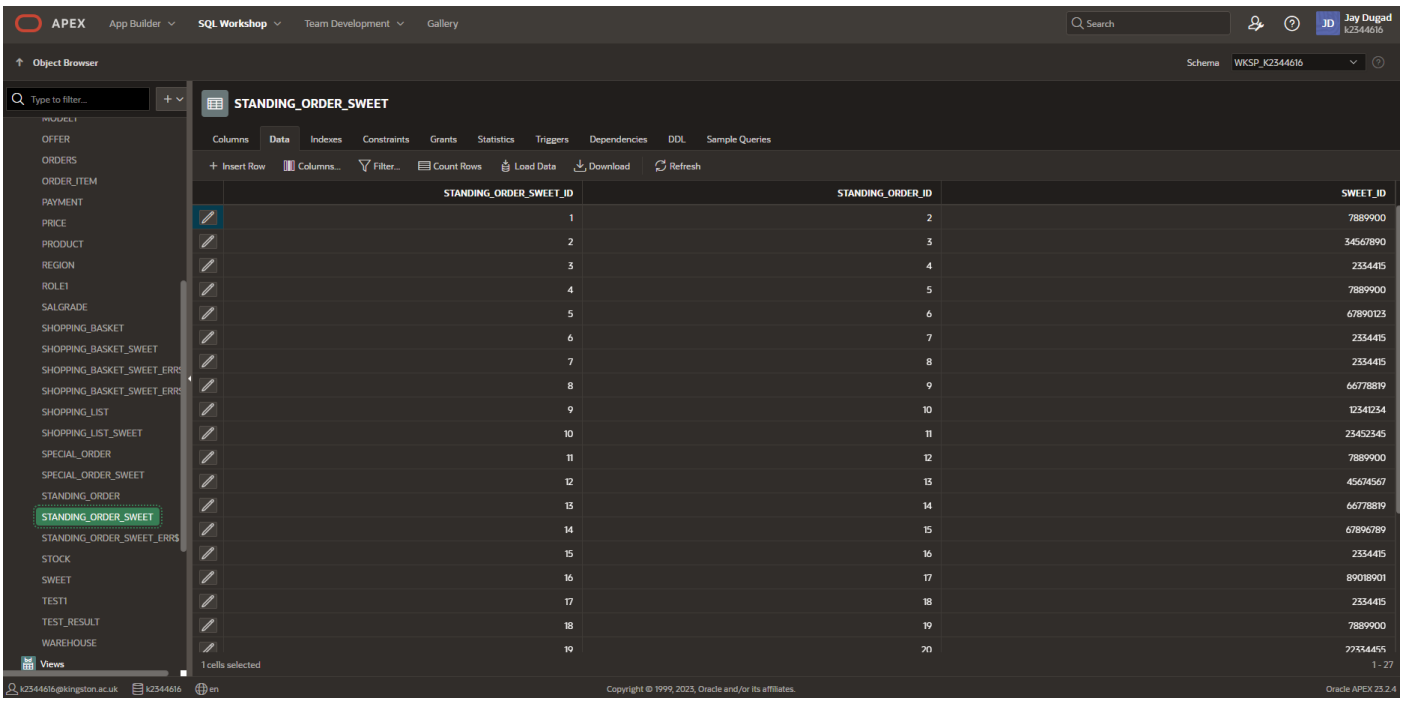
- **Payment Table: -** Here a customer can store only zero or one payment detail and one payment detail can belong to only one customer [payment(0..1)----(1..1)customer].



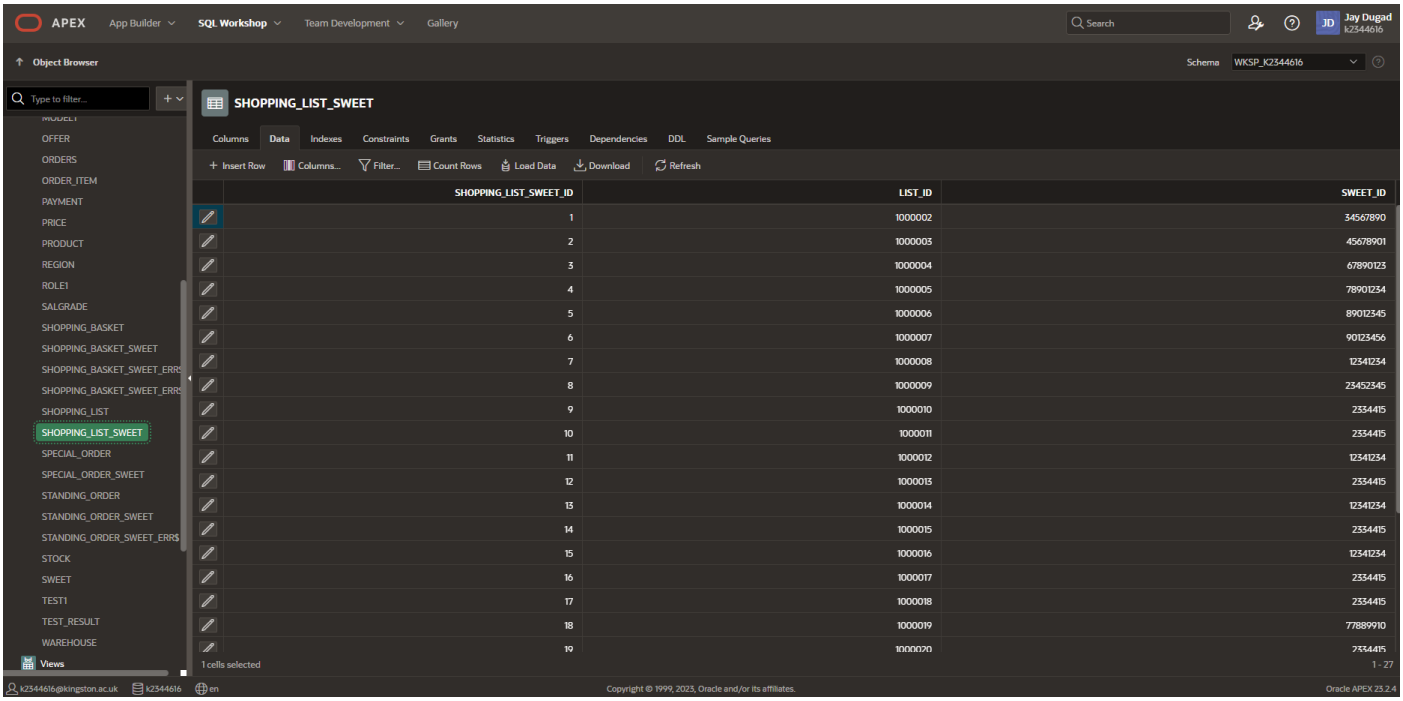
- **Special Order Sweet *(Junction Table)*: -** This is junction table 2 many-to-many relation tables [special_order(0.*)----(1.*)sweet]. This table consists primary key of both the tables special_order as well as sweet.



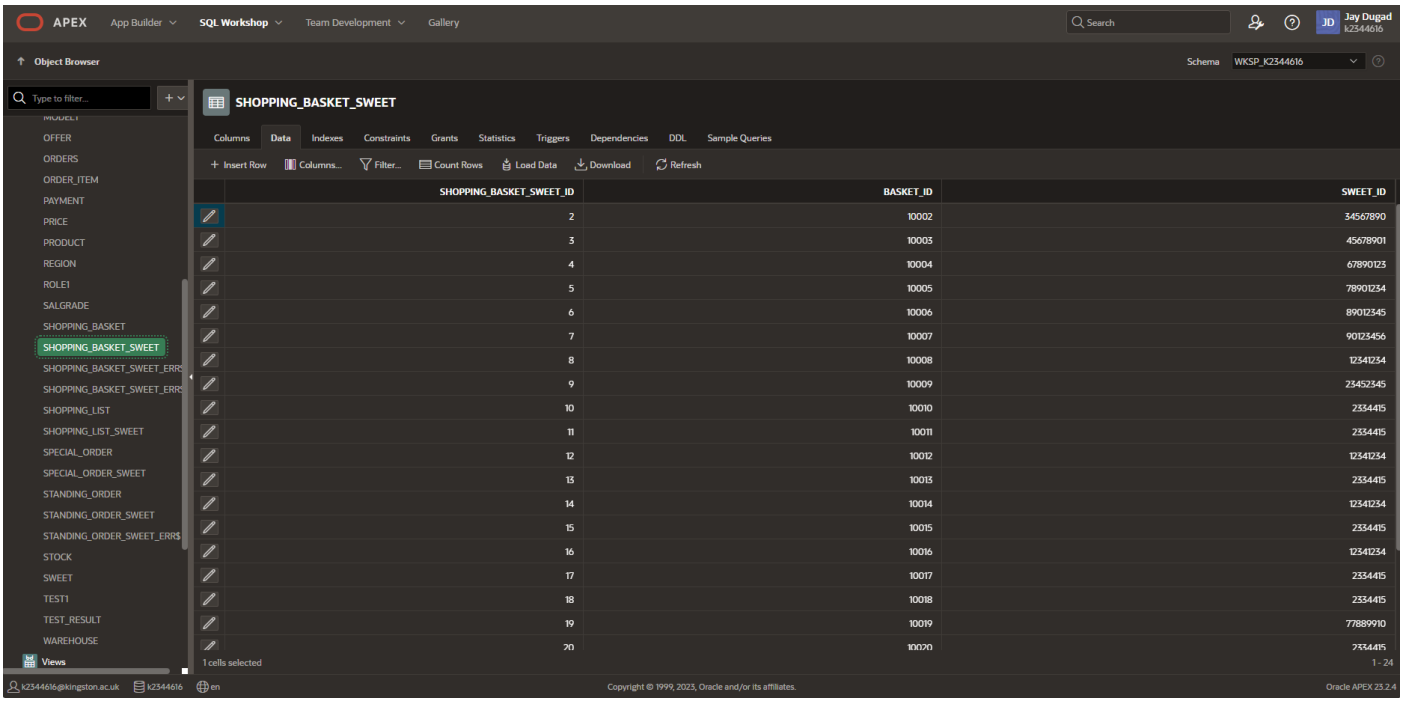
- **<u>Standing Order *Sweet (Junction Table)*: -</u>** This is junction table 2 many-to-many relation tables [standing_order(0.*)----(1.*)sweet]. This table consists primary key of both the tables standing_order as well as sweet.



- **<u>Shopping List Sweet *(Junction Table)*: -</u>** This is junction table 2 many-to-many relation tables [shopping_list(0.*)----(0.*)sweet]. This table consists primary key of both the tables shopping_list as well as sweet.



- **Shopping Basket Sweet *(Junction Table)*: -** This is junction table 2 many-to-many relation tables [shopping_basket(0.*)----(0.*)sweet]. This table consists primary key of both the tables shopping_basket as well as sweet.



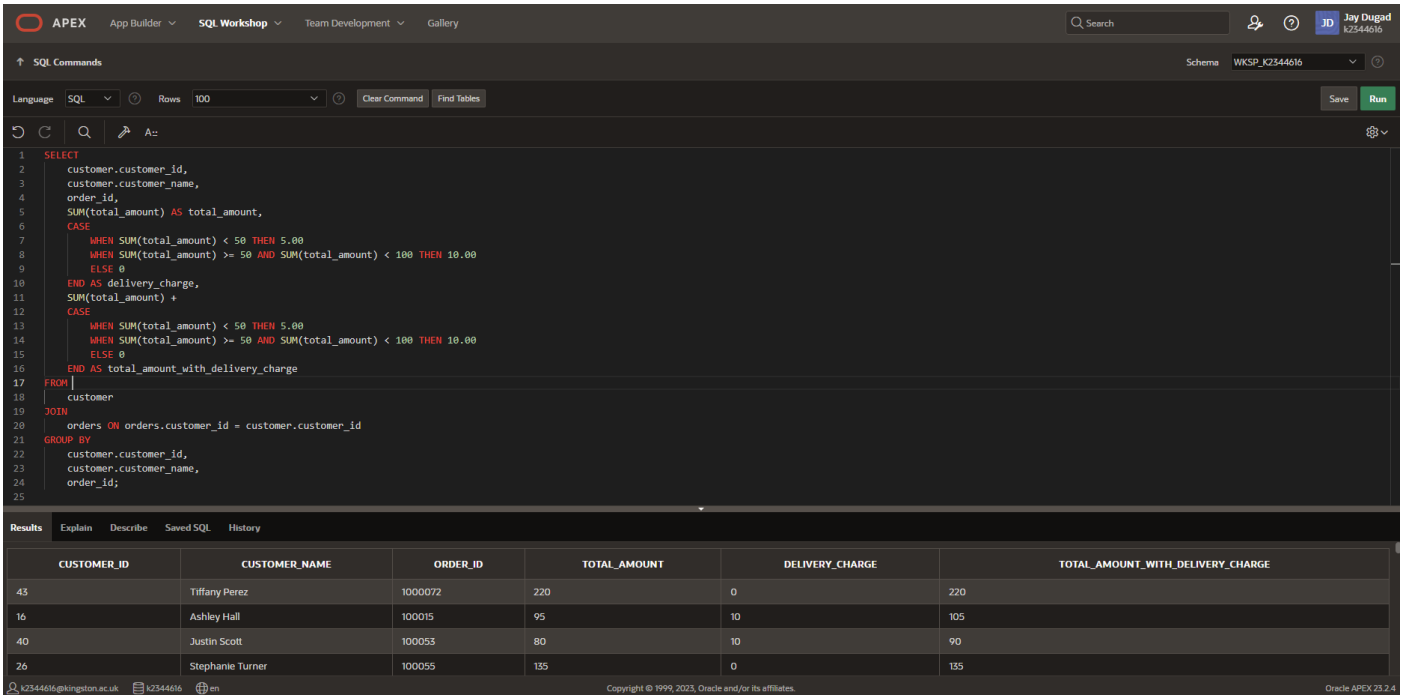
CHAPTER 4: SQL QUERIES

Query 1) This query calculates delivery charge based on the order's amount and displays delivery fee and total amount after adding delivery fee. It retrieves information about each customer's id, customer's name, order's id, order's amount who has ordered.

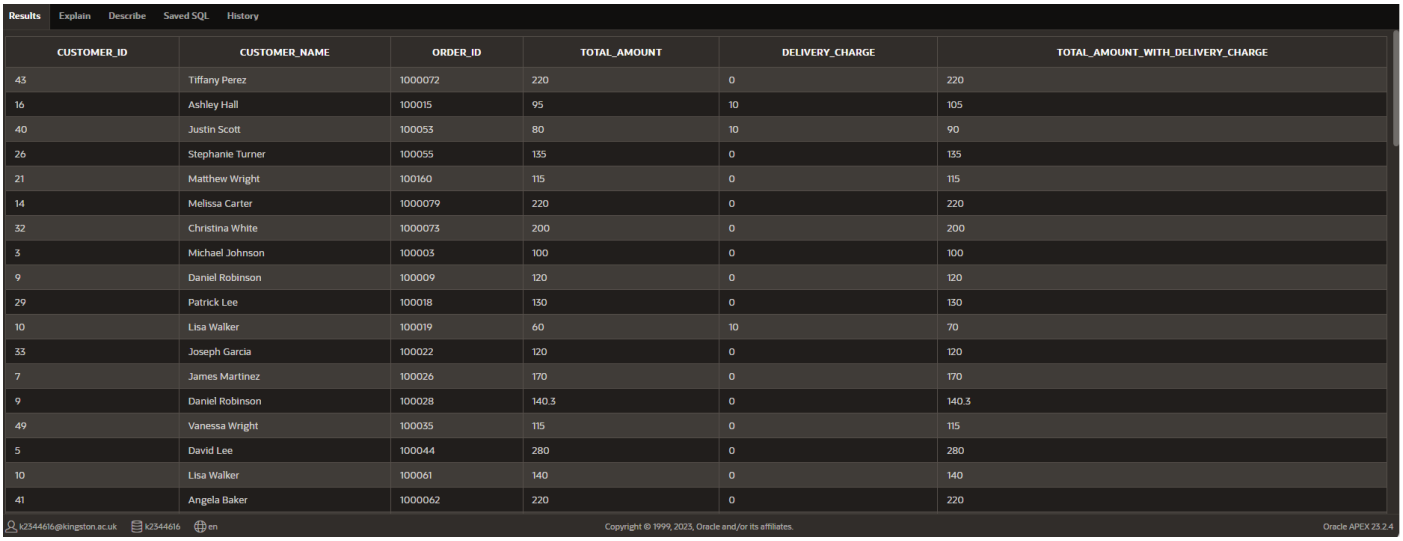
```
SELECT
   customer.customer_id, customer.customer_name,
   order_id,
   SUM(total_amount) AS total_amount,
   CASE
      WHEN SUM(total_amount) < 50 THEN 5.00
      WHEN SUM(total_amount) >= 50 AND SUM(total_amount) < 100 THEN 10.00
      ELSE 0
   END AS delivery_charge,
   SUM(total_amount) +
   CASE
      WHEN SUM(total_amount) < 50 THEN 5.00
      WHEN SUM(total_amount) >= 50 AND SUM(total_amount) < 100 THEN 10.00
```

```
        ELSE 0
    END AS total_amount_with_delivery_charge
FROM
    customer
JOIN
    orders ON orders.customer_id = customer.customer_id
GROUP BY
    customer.customer_id, customer.customer_name, order_id
```

Query: -



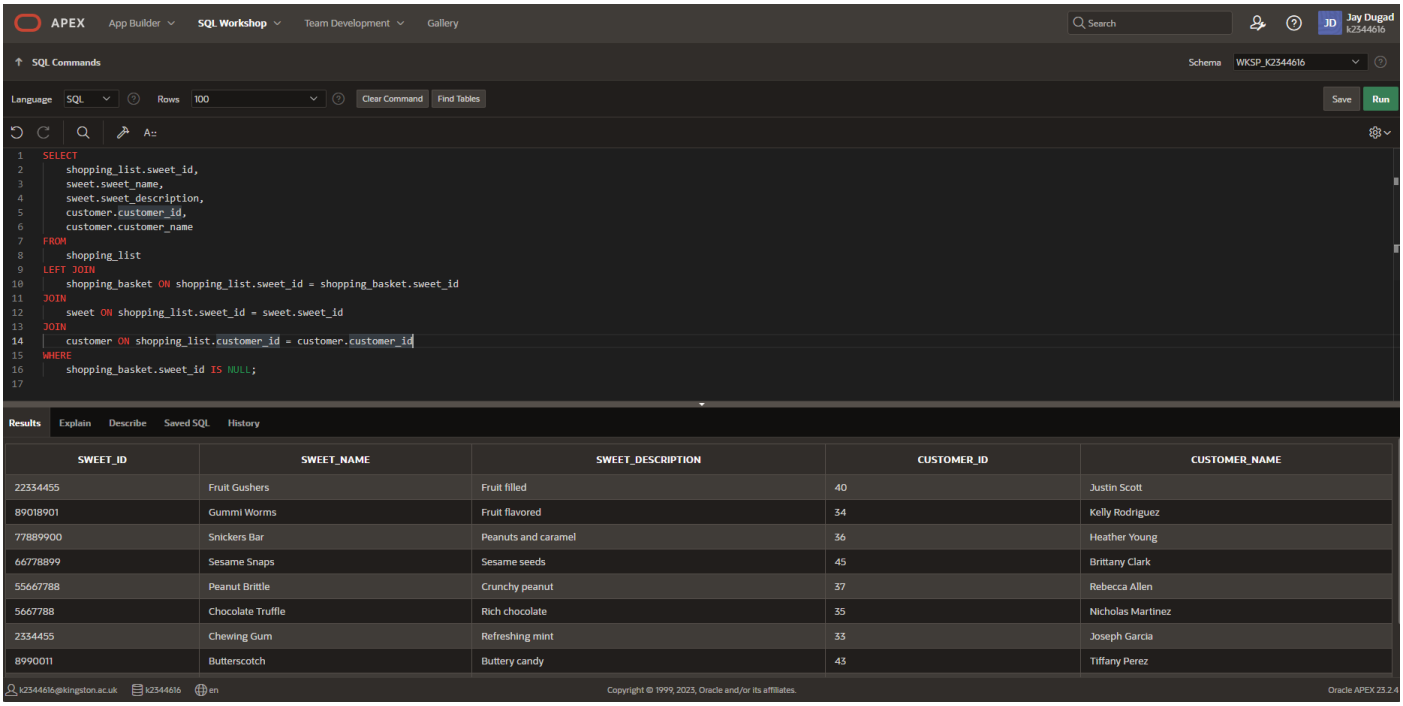
Output: -



Query 2) This query displays the sweets which were in shopping list but were not in the shopping basket while making an order.

```
SELECT
    shopping_list.sweet_id,
    sweet.sweet_name,
    sweet.sweet_description,
    customer.customer_id,
    customer.customer_name
FROM
    shopping_list
LEFT JOIN
    shopping_basket ON shopping_list.sweet_id = shopping_basket.sweet_id
JOIN
    sweet ON shopping_list.sweet_id = sweet.sweet_id
JOIN
    customer ON shopping_list.customer_id = customer.customer_id
WHERE
    shopping_basket.sweet_id IS NULL;
```

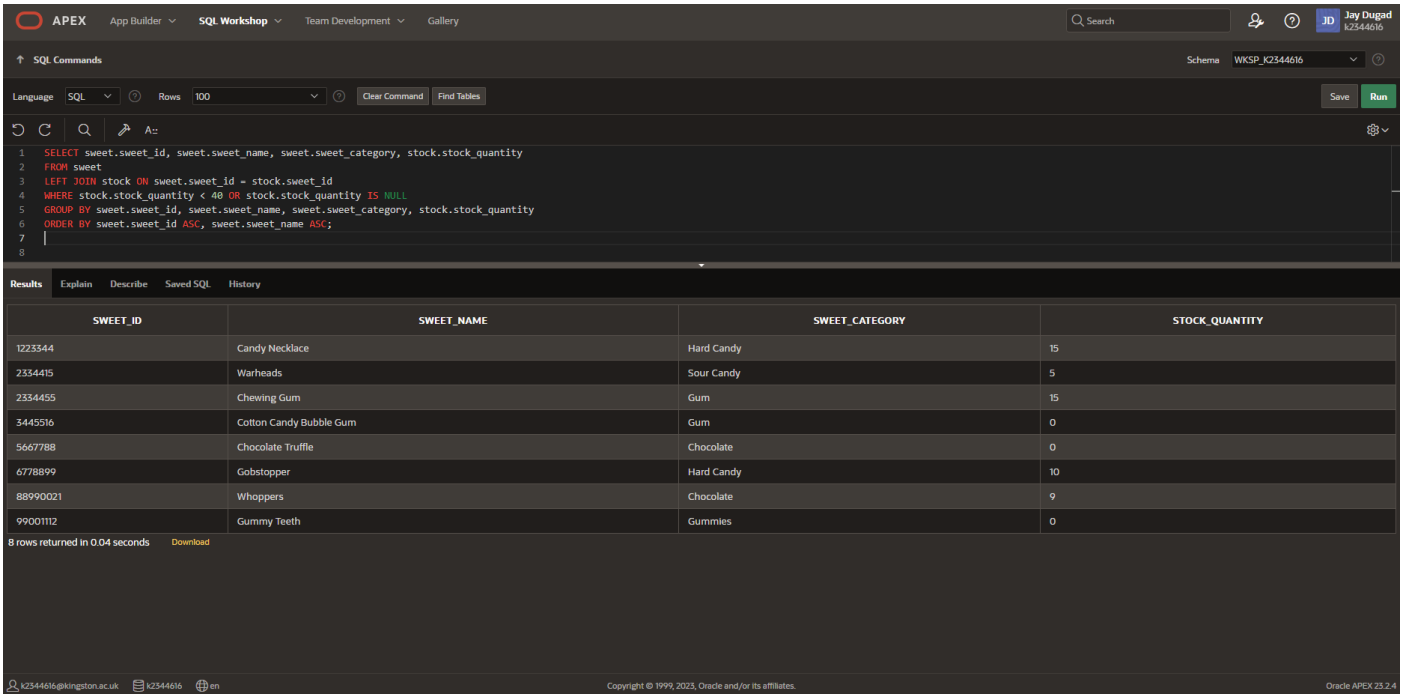
Output: -



Query 3) Retrieves sweet id, sweet name, sweet category and stock quantity who are soon going to be out of stock.

```
SELECT sweet.sweet_id, sweet.sweet_name, sweet.sweet_category, stock.stock_quantity
FROM sweet
LEFT JOIN stock ON sweet.sweet_id = stock.sweet_id
WHERE stock.stock_quantity < 40 OR stock.stock_quantity IS NULL
GROUP BY sweet.sweet_id, sweet.sweet_name, sweet.sweet_category, stock.stock_quantity
ORDER BY sweet.sweet_id ASC;
```

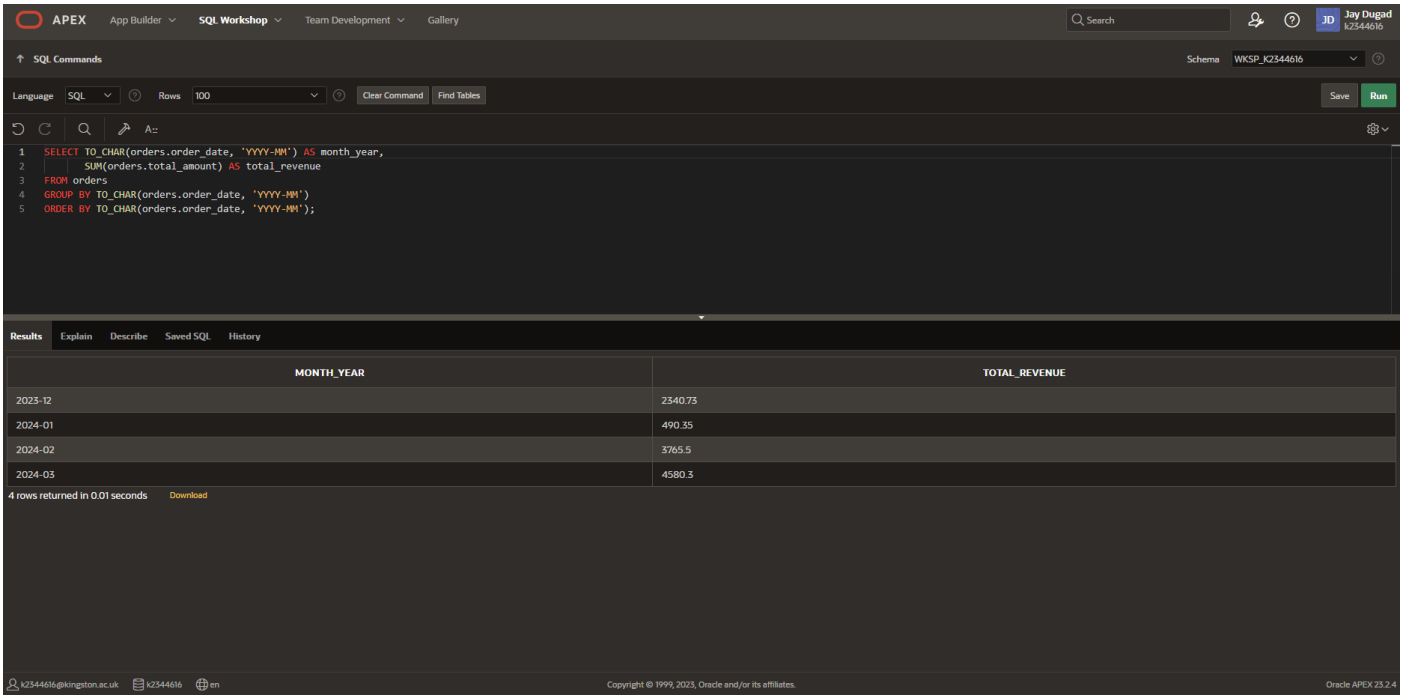
Output: -



Query 4) This query generates and displays the total revenue of every month and year by using total amount of all the orders.

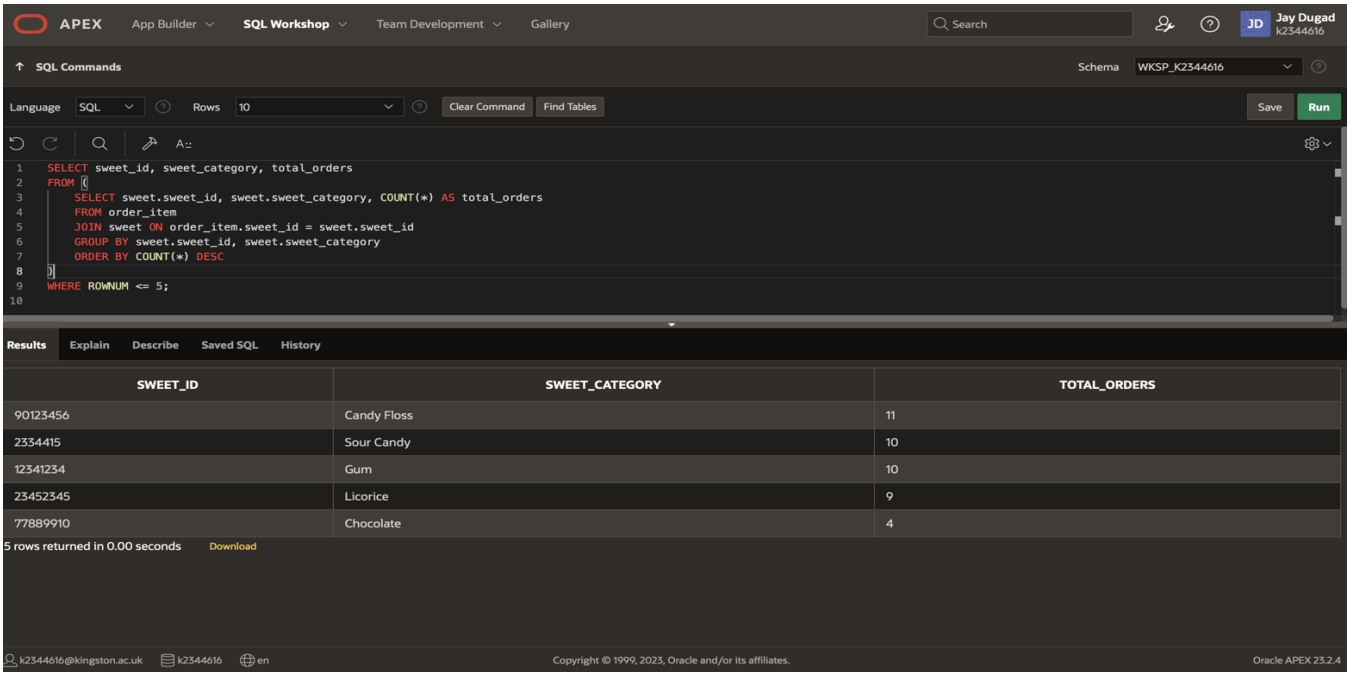
```
SELECT TO_CHAR(orders.order_date, 'YYYY-MM') AS month_year,
    SUM(orders.total_amount) AS total_revenue
FROM orders
GROUP BY TO_CHAR(orders.order_date, 'YYYY-MM')
ORDER BY TO_CHAR(orders.order_date, 'YYYY-MM');
```

Output: -



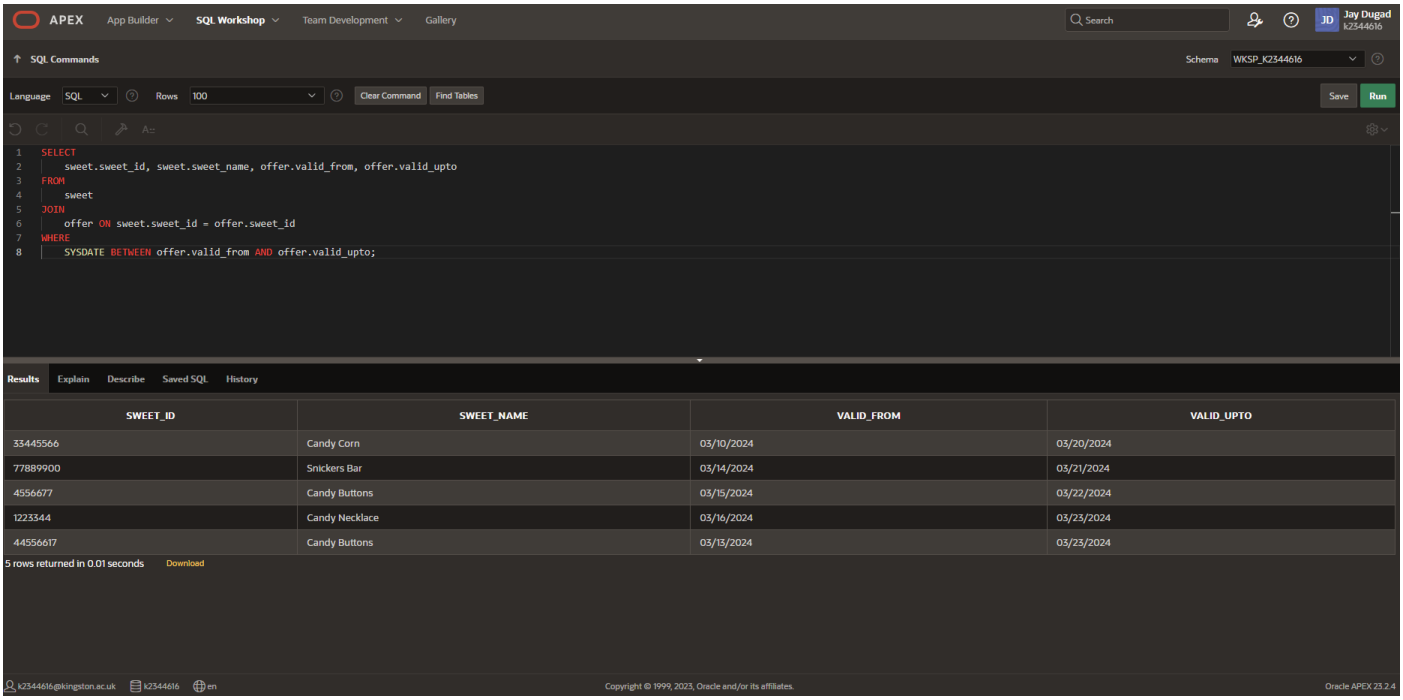
Query 5) This query identifies the top 5 best-selling sweet category.

```
SELECT sweet_id, sweet_category, total_orders
FROM (
    SELECT sweet.sweet_id, sweet.sweet_category, COUNT(*) AS total_orders
    FROM order_item
    JOIN sweet ON order_item.sweet_id = sweet.sweet_id
    GROUP BY sweet.sweet_id, sweet.sweet_category
    ORDER BY COUNT(*) DESC
)
WHERE ROWNUM <= 5;
```



Query 6) This query shows the sweets that have active offers right now.

```
SELECT
    sweet.sweet_id, sweet.sweet_name, offer.valid_from, offer.valid_upto
FROM
    sweet
JOIN
    offer ON sweet.sweet_id = offer.sweet_id
WHERE
    SYSDATE BETWEEN offer.valid_from AND offer.valid_upto;
```



- BSS's database structure is carefully designed to meet the operational requirements of the company. Tables pertaining to client confectionery orders, delivery, shopping lists, customer information, and other information are efficiently arranged. This methodical approach, which covers everything from stocking candy to handling client demands, is a reflection of the company's workflow and procedures. The data model ensures practical use and integrity inside the database by incorporating many data types and restrictions, including NOT NULL, PRIMARY KEY, and FOREIGN KEY
- Additionally, the database makes it easier to manage candy-related data, such as prices, sizes, descriptions, and stock levels, allowing BSS to effectively handle and preserve crucial data. Additionally, it improves browsing and search experiences for users by tracking offers and promotions and streamlining the segmentation of candies into categories. This feature draws customers by offering an enjoyable online buying experience, which incentivizes purchases.
- The database not only enhances browsing and buying experiences but also encourages users involvement with tools like shopping lists and baskets that help users easily bookmark products and finish transactions. Furthermore, the integration of user authentication guarantees secure entry to customized services like order histories and shopping lists, augmenting the general customer experience and contentment. BSS can meet website and database needs while optimizing performance and increasing user engagement thanks to our all-inclusive approach to database administration.

Review of the entire exercise: -

- I enjoyed working on this coursework since it allowed me to gain real-world experience creating and managing databases. Doing this assignment made me better suited for employment. I also learned how to handle large amounts of data and work with the relationships between various tables from this project.
- My ability to solve problems has improved as a result of working with complex data structures and developing effective database designs. I am now able to identify problems and come up with the best solutions. Asking doubts to classmates and professors throughout the assignment improved my communication abilities, allowing me to explain complex database ideas in simple terms and lead fruitful conversations on database architecture and use.
- Participating in coursework also offered me a chance for ongoing education and skill enhancement, encouraging a lifelong learning mindset and a dedication to remaining current with database technology and processes. My study has given me the thorough experience, knowledge, and confidence I need to explore professional prospects in database management roles, making me a competitive candidate in the job market.