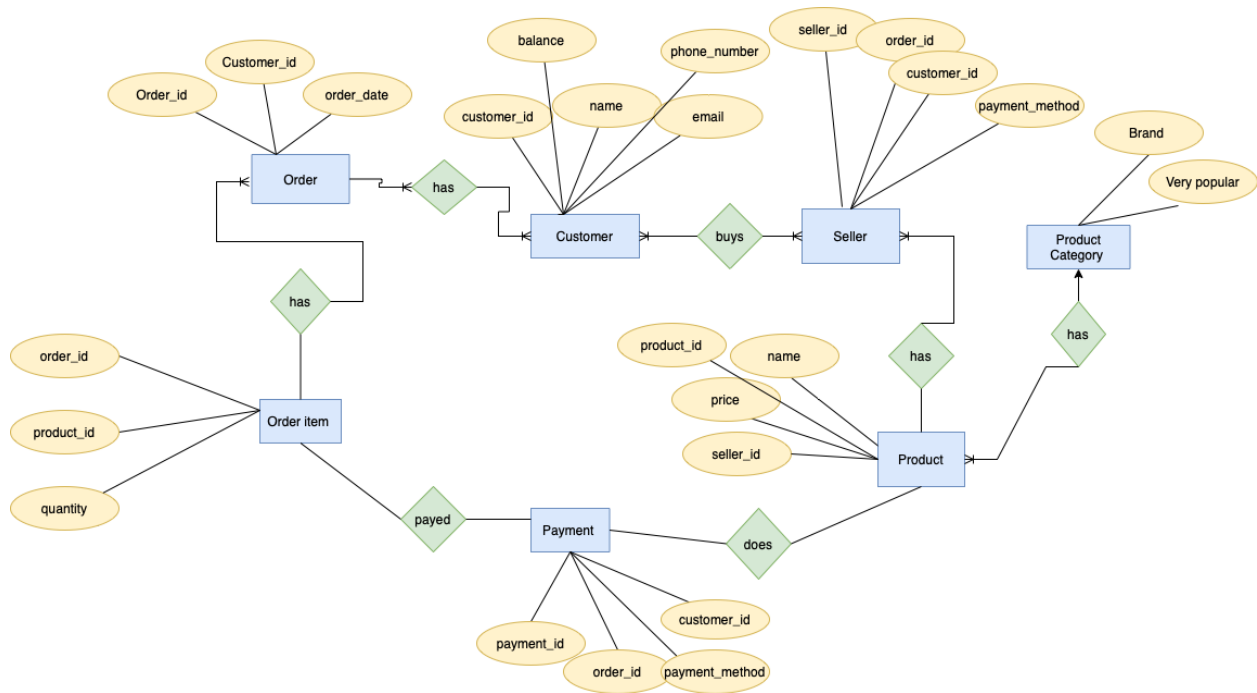


## Online Shop

Who is do it

210103096 Nurgali Daulet

In this project first



Sure, I can provide additional information about this ERD for an online shop.

One important concept to understand is the use of primary and foreign keys in the tables. A primary key is a unique identifier for each record in a table. In this ERD, the primary keys for the tables are as follows:

- **Buyer:** order\_id
- **Customer:** customer\_id
- **Payment:** payment\_id
- **Product:** product\_id
- **Seller:** seller\_id

A foreign key, on the other hand, is a column or set of columns in one table that refers to the primary key of another table. In this ERD, the foreign keys are as follows:

- Buyer: customer\_id (refers to Customer table)
- OrderItem: order\_id (refers to Buyer table), product\_id (refers to Product table)
- Payment: order\_id (refers to Buyer table), customer\_id (refers to Customer table)
- Product: seller\_id (refers to Seller table)

By using primary and foreign keys, it is possible to establish relationships between the different tables in the database, making it easier to query and update data.

Another important concept in this ERD is normalization. Normalization is the process of organizing data in a database to minimize redundancy and ensure data consistency. This ERD appears to be normalized up to at least third normal form (3NF), which means that each table is free of repeating groups and dependencies on non-key attributes.

Overall, this ERD provides a solid foundation for an online shop database, allowing for efficient management of customer orders, payments, and product information.

The ERD for the online shop consists of seven tables:

1. **Buyer:** This table stores information about the buyer's orders. It has attributes such as order\_id (primary key), customer\_id (foreign key referencing customer table), and order\_date.
2. **OrderItem:** This table stores information about the items that are part of an order. It has attributes such as order\_id (foreign key referencing order table), product\_id, and quantity.
3. **Customer:** This table stores information about the customers. It has attributes such as customer\_id (primary key), balance, name, email, and phone\_number.
4. **Payment:** This table stores information about the payments made by customers. It has attributes such as payment\_id, order\_id (foreign key referencing order table), customer\_id (foreign key referencing customer table), and payment\_method.
5. **Product:** This table stores information about the products available for sale. It has attributes such as product\_id, name, price, and seller\_id (foreign key referencing seller table).

6. **Seller:** This table stores information about the sellers. It has attributes such as **seller\_id** (primary key), **name**, **email**, and **phone\_number**.
7. **Product Category**

### **Functional Dependencies**

**Functional dependency**

**Buyer(Order)**

**Order\_id, customer\_id  $\rightarrow$  order\_date**

**Order\_id, order\_date  $\rightarrow$  customer\_id**

1. **Identify the functional dependencies (FDs) in the given relations.**
2. **Create new relations for each FD with a composite key that includes all the attributes on the right-hand side of the FD.**
3. **Remove the attributes involved in the FDs from the original relations, leaving only the attributes that form the keys for the new relations.**

**Given the following relations:**

**R1(Order\_id, customer\_id  $\rightarrow$  order\_date)**

**R2(Order\_id, order\_date  $\rightarrow$  customer\_id)**

**We can identify the following FDs:**

**FD1: (Order\_id, customer\_id)  $\rightarrow$  order\_date (from R1)**

**FD2: (Order\_id, order\_date)  $\rightarrow$  customer\_id (from R2)**

**To convert these relations to 3NF, we can follow these steps:**

**Step 1: Create a new relation for FD1 with a composite key of (Order\_id, customer\_id) and an attribute of order\_date.**

**R3(Order\_id, customer\_id, order\_date)**

**Step 2: Create a new relation for FD2 with a composite key of (Order\_id, order\_date) and an attribute of customer\_id.**

**R4(Order\_id, order\_date, customer\_id)**

**Step 3: Remove the attributes involved in the FDs from the original relations, leaving only the attributes that form the keys for the new relations.**

**R1(Order\_id, customer\_id)**

**R2(Order\_id, order\_date)**

**Now, let's check if these relations are in 3NF or not:**

- 1. All attributes in each relation are atomic - Yes, all attributes are atomic.**
- 2. There are no transitive dependencies - Yes, there are no transitive dependencies since we created new relations for each FD with a composite key that includes all the attributes on the right-hand side of the FD.**

**Therefore, the resulting relations are in 3NF.**

### **OrderItem**

- 1. Order\_id(FK) → product\_id(FK), quantity**
- 2. Identify the functional dependencies in the relation.**
- 3. Eliminate transitive dependencies.**
- 4. Create new relations for the eliminated dependencies.**

**Given Relation:**

**Order\_id(FK) → product\_id(FK), quantity**

**Step 1: Identify the functional dependencies in the relation.**

**The given relation has two functional dependencies:**

- $\text{Order\_id} \rightarrow \text{product\_id}$
- $\text{Order\_id} \rightarrow \text{quantity}$

**Step 2: Eliminate transitive dependencies.**

**There are no transitive dependencies in this relation.**

**Step 3: Create new relations for the eliminated dependencies.**

**The given relation is already in 2NF because it has only one composite primary key,  $\text{Order\_id}$  and  $\text{product\_id}$ . To convert it into 3NF, we need to separate the non-key attribute  $\text{quantity}$  into a separate relation.**

**New relations:**

**Relation 1: Order ( $\text{Order\_id}(\text{PK})$ )**

- $\text{Order\_id}$

**Relation 2: Product ( $\text{product\_id}(\text{PK})$ ,  $\text{quantity}$ )**

- $\text{product\_id}$
- $\text{quantity}$

**Therefore, the given relation is now in 3NF.**

**Customer**

1.  $\text{Customer\_id}(\text{PK}) \rightarrow \text{name, email, phone\_number, balance}$

1. Identify the functional dependencies in the relation.
2. Eliminate transitive dependencies.
3. Create new relations for the eliminated dependencies.

**Given Relation:**

**$\text{Customer\_id}(\text{PK}) \rightarrow \text{name, email, phone\_number, balance}$**

**Step 1: Identify the functional dependencies in the relation.**

**The given relation has one functional dependency:**

- **Customer\_id  $\rightarrow$  name, email, phone\_number, balance**

**Step 2: Eliminate transitive dependencies.**

**There are no transitive dependencies in this relation.**

**Step 3: Create new relations for the eliminated dependencies.**

**Since there are no transitive dependencies in this relation, it is already in 3NF.**

**Therefore, the given relation is already in 3NF and does not require any further normalization.**

## **Payment**

1. **Payment\_id  $\rightarrow$  order\_id, customer\_id, payment\_method**
2. **Order\_id  $\rightarrow$  customer\_id, payment\_method**

1. **Identify the functional dependencies in the relation.**
2. **Eliminate transitive dependencies.**
3. **Create new relations for the eliminated dependencies.**

**Given Relation:**

**Payment\_id  $\rightarrow$  order\_id, customer\_id, payment\_method**

**Order\_id  $\rightarrow$  customer\_id, payment\_method**

**Step 1: Identify the functional dependencies in the relation.**

**The given relation has two functional dependencies:**

- **Payment\_id  $\rightarrow$  order\_id, customer\_id, payment\_method**

- **Order\_id  $\rightarrow$  customer\_id, payment\_method**

**Step 2: Eliminate transitive dependencies.**

**There is a transitive dependency between Payment\_id and customer\_id, and another between Order\_id and payment\_method.**

**New Relation 1:**

**Payment\_id  $\rightarrow$  order\_id, payment\_method**

**Order\_id  $\rightarrow$  customer\_id**

**New Relation 2:**

**Payment\_id  $\rightarrow$  customer\_id**

**New Relation 3:**

**Customer\_id  $\rightarrow$  name, email, phone\_number**

**Step 3: Create new relations for the eliminated dependencies.**

- **Relation 1: Payment\_id  $\rightarrow$  order\_id, payment\_method**
- **Relation 2: Order\_id  $\rightarrow$  customer\_id**
- **Relation 3: Customer\_id  $\rightarrow$  name, email, phone\_number**

**All the new relations are in 3NF.**

**Therefore, the final 3NF relations are:**

- **Payment\_id  $\rightarrow$  order\_id, payment\_method**
- **Order\_id  $\rightarrow$  customer\_id**
- **Customer\_id  $\rightarrow$  name, email, phone\_number**

**Product**

1. **product\_id  $\rightarrow$  name, price, seller\_id**

1. Identify the functional dependencies in the relation.
2. Eliminate transitive dependencies.
3. Create new relations for the eliminated dependencies.

**Given Relation:**

**product\_id  $\rightarrow$  name, price, seller\_id**

**Step 1: Identify the functional dependencies in the relation.**

**The given relation has one functional dependency:**

- **product\_id  $\rightarrow$  name, price, seller\_id**

**Step 2: Eliminate transitive dependencies.**

**There are no transitive dependencies in this relation.**

**Step 3: Create new relations for the eliminated dependencies.**

**Since there are no transitive dependencies in this relation, it is already in 3NF.**

**Therefore, the given relation is already in 3NF and does not require any further normalization.**

**Seller**

1. **seller\_id(PK)  $\rightarrow$  name, email, phone\_number**

1. Identify the functional dependencies in the relation.
2. Eliminate transitive dependencies.
3. Create new relations for the eliminated dependencies.

**Given Relation:**



**$\text{seller\_id(PK)} \rightarrow \text{name, email, phone\_number}$**

**Step 1: Identify the functional dependencies in the relation.**

**The given relation has one functional dependency:**

- **$\text{seller\_id} \rightarrow \text{name, email, phone\_number}$**

**Step 2: Eliminate transitive dependencies.**

**There are no transitive dependencies in this relation.**

**Step 3: Create new relations for the eliminated dependencies.**

**Since there are no transitive dependencies in this relation, it is already in 3NF.**

**Therefore, the given relation is already in 3NF and does not require any further normalization.**

The screenshot shows the APEX SQL Workshop interface. At the top, there's a navigation bar with 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'DN Daulet Nurgali shymkent' are on the right. Below this, the 'SQL Commands' section shows the 'Schema' as 'WKSP\_SHYMKENT'. The 'Language' is set to 'SQL' and 'Rows' to '10'. There are buttons for 'Clear Command', 'Find Tables', 'Save', and 'Run'. The main editor area contains a PL/SQL block with the following code:

```
1 BEGIN
2
3
4     INSERT INTO BUYER
5     VALUES (5555, 123, SYSDATE);
6
7     INSERT INTO PAYMENT
8     VALUES (9999, 5555, '123', 'PayPal');
9
10
11     COMMIT;
12 EXCEPTION
13
14     WHEN OTHERS THEN
15         ROLLBACK TO start_of_transaction;
16         RAISE;
17 END;
```

Below the editor, the 'Results' tab is active, showing the execution output:

```
rows before insert : 22
1 row(s) inserted.
0.11 seconds
```

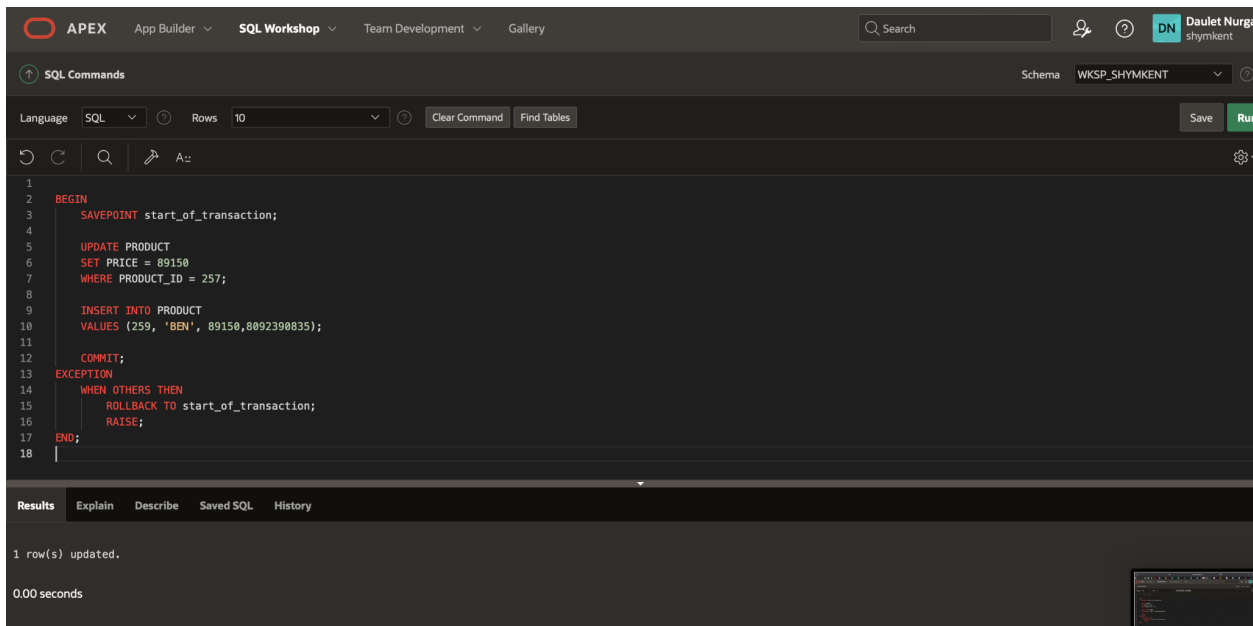
This code is a PL/SQL block that inserts a new record into the "BUYER" table and a corresponding record into the "PAYMENT" table, using the same "ORDER\_ID" value for both.

The block begins with a "BEGIN" statement and sets a savepoint called "start\_of\_transaction". It then inserts a new record into the "BUYER" table with an "ORDER\_ID" value of 5555, a "CUSTOMER\_ID" value of 123, and the current system date as the "ORDER\_DATE".

Next, it inserts a new record into the "PAYMENT" table with a "PAYMENT\_ID" value of 9999, an "ORDER\_ID" value of 5555 (matching the "ORDER\_ID" value of the record just inserted into the "BUYER" table), a "CUSTOMER\_ID" value of '123', and a "PAYMENT\_METHOD" value of 'PayPal'.

If both of the inserts succeed, the "COMMIT" statement is executed, which makes the changes permanent. However, if an exception is raised during the execution of either the

**"INSERT INTO BUYER" or "INSERT INTO PAYMENT" statement, the block catches the exception with a "WHEN OTHERS" clause, rolls back to the "start\_of\_transaction" savepoint, and re-raises the exception.**



The screenshot shows the APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'DN Daulet Nurg...' are on the right. The 'SQL Commands' tab is active, showing a PL/SQL block with the following code:

```
1 BEGIN
2   SAVEPOINT start_of_transaction;
3
4   UPDATE PRODUCT
5   SET PRICE = 89150
6   WHERE PRODUCT_ID = 257;
7
8   INSERT INTO PRODUCT
9   VALUES (259, 'BEN', 89150, 8092390835);
10
11 COMMIT;
12 EXCEPTION
13 WHEN OTHERS THEN
14   ROLLBACK TO start_of_transaction;
15   RAISE;
16 END;
```

The interface also shows 'Language: SQL', 'Rows: 10', and buttons for 'Clear Command', 'Find Tables', 'Save', and 'Run'. The 'Results' tab at the bottom indicates '1 row(s) updated.' and '0.00 seconds'.

**This code is a PL/SQL block that includes SQL statements to update and insert data into a table, with exception handling to handle any errors that might occur. Here's a breakdown of what's happening:**

- **The block starts with a SAVEPOINT statement to mark the start of the transaction.**
- **The UPDATE statement modifies the PRICE column of the PRODUCT table for the row with a PRODUCT\_ID of 257.**
- **The INSERT statement adds a new row to the PRODUCT table with a PRODUCT\_ID of 259, a NAME of 'BEN', a PRICE of 89150, and a SELLER\_ID of 8092390835.**

- If both statements are executed successfully, the COMMIT statement is executed to commit the transaction and make the changes permanent in the database.
- If an error occurs during the execution of either statement, the EXCEPTION block is executed.
- The ROLLBACK TO start\_of\_transaction statement rolls back the transaction to the SAVEPOINT, effectively undoing any changes made by the UPDATE and INSERT statements.

## Procedure

The screenshot displays the Oracle APEX SQL Workshop interface. On the left, the Object Browser shows a tree structure of database objects under the WKSP\_SHYMKENT schema. The 'Procedures' folder is expanded, and 'LIST\_PRODUCTS\_BY\_SELLER' is selected. The main editor area shows the SQL code for creating or replacing the procedure. The code is as follows:

```

1 create or replace PROCEDURE list_products_by_seller
2 IS
3 BEGIN
4 LOOP
5   FOR product IN (SELECT * FROM product WHERE product.seller_id = seller.seller_id ORDER BY price)
6   LOOP
7     DBMS_OUTPUT.PUT_LINE(' Product: ' || product.name || ', Price: ' || product.price);
8   END LOOP;
9 END LOOP;
10 END;
11 /

```

The interface includes a top navigation bar with 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'Daulet Nurgali shymkent' are also visible. The bottom status bar shows the user's email '210103096@stu.sdu.edu.kz', the username 'shymkent', and the Oracle APEX version '23.1.0-15'.

**APEX** App Builder SQL Workshop Team Development Gallery

Search

Schema WKSP\_SHYMKENT

Language SQL Rows 10 Clear Command Find Tables Save Run

```
1 BEGIN
2   group_by_info();
3 END;
```

Results Explain Describe Saved SQL History

Order ID: 3, Customer ID: 103, Order Date:  
Order ID: 13, Customer ID: 113, Order Date:  
Order ID: 16, Customer ID: 116, Order Date:  
Order ID: 17, Customer ID: 117, Order Date:  
Order ID: 5555, Customer ID: 123, Order Date:  
Order ID: 5, Customer ID: 105, Order Date:  
Order ID: 9, Customer ID: 109, Order Date:  
Order ID: 14, Customer ID: 114, Order Date:  
Order ID: 18, Customer ID: 118, Order Date:  
Order ID: 4, Customer ID: 104, Order Date:  
Order ID: 7, Customer ID: 107, Order Date:

210103096@stu.sdu.edu.kz shymkent en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.1.0-15

**APEX** App Builder SQL Workshop Team Development Gallery

Search

Schema WKSP\_SHYMKENT

Object Browser

Type to filter...

- OURPATIENT\_
- PAYMENT
- PAYMENT\_
- PAYMENT\_ERR\$
- PRODUCT**
- PRODUCT\_
- PRODUCT\_ERR\$
- ROOM
- SELLER
- SELLER\_
- SELLER\_ERR\$
- STAFF
- STUDENTS
- ZAKAZ
- Views
- Indexes
- Sequences
- Types
- Packages
- Procedures
- Functions
- Triggers
- Database Links

**UPDATE\_PRODUCT\_PRICE**

Code Dependencies Errors Grants

Download Save and Compile Drop Refresh

```
1 create or replace PROCEDURE update_product_price(
2   in_product_id IN product.product_id%TYPE,
3   in_seller_id IN seller.seller_id%TYPE,
4   in_new_price IN product.price%TYPE
5 )
6 IS
7 BEGIN
8   UPDATE product
9     SET price = in_new_price
10    WHERE product_id = in_product_id AND seller_id = in_seller_id;
11
12   IF SQL%ROWCOUNT = 0 THEN
13     RAISE_APPLICATION_ERROR(-20001, 'Product not found for the given seller');
14   ELSE
15     DBMS_OUTPUT.PUT_LINE('Product price updated successfully');
16   END IF;
17 END;
18 /
```

210103096@stu.sdu.edu.kz shymkent en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.1.0-15

The screenshot shows the Oracle APEX SQL Workshop interface. At the top, there's a navigation bar with 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'DN Daulet Nurgali shymkent' are on the right. Below this, the 'SQL Commands' tab is active, showing a PL/SQL procedure. The procedure is defined as follows:

```
1 declare
2   in_product_id number := 257;
3   in_seller_id number := 8092390835;
4   in_new_price number := 89150;
5 BEGIN
6   update_product_price(
7     in_product_id,
8     in_seller_id,
9     in_new_price
10  );
11 END;
```

The 'Results' tab at the bottom shows the execution output: 'Product price updated successfully', 'Statement processed.', and '0.12 seconds'. The footer includes the user email '210103096@stu.sdu.edu.kz', the username 'shymkent', and the Oracle APEX version '23.1.0-15'.

This PL/SQL code defines a procedure called `update_product_price` which updates the price of a product for a given seller. The procedure takes in three input parameters - `in_product_id`, `in_seller_id`, and `in_new_price`, which are of the same data types as their corresponding columns in the product and seller tables.

The `UPDATE` statement in the procedure updates the price column of the product table for the specified `product_id` and `seller_id` with the new price provided in the input parameter `in_new_price`.

The `IF` statement that follows checks if the `UPDATE` statement affected any rows. If no rows were updated, it means that the product with the given `product_id` and `seller_id` does not exist, and the procedure raises an error with the `RAISE_APPLICATION_ERROR` statement.

If the `UPDATE` statement affected one or more rows, the `ELSE` block executes, and the procedure prints a message to indicate that the product price has been updated successfully using the `DBMS_OUTPUT.PUT_LINE` statement.

In summary, this PL/SQL code defines a procedure to update the price of a product for a given seller and ensures that the operation is successful and that the product exists in the database before updating its price.

# Trigger

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'DN Daulet Nurgali shymkent' are on the right. The left sidebar shows the 'Object Browser' with a list of tables, including 'BUYER'. The main area is titled 'TRIGGBUYER' and shows the 'Code' tab with the following SQL script:

```
1 create or replace TRIGGER triggBuyer
2 BEFORE INSERT ON Buyer
3 DECLARE
4 rows_nums NUMBER;
5 BEGIN
6     SELECT COUNT(*) INTO rows_nums FROM BUYER;
7     dbms_output.put_line('rows before insert : ' || rows_nums);
8 END;
9 /
```

Buttons for 'Download', 'Save and Compile', 'Drop', and 'Refresh' are visible. The bottom status bar shows the user's email, session name, and copyright information.

The screenshot shows the Oracle APEX SQL Workshop interface with the 'SQL Commands' tab selected. The 'Language' is set to 'SQL' and 'Rows' is set to '10'. The SQL command entered is:

```
1 begin
2 insert_into BUYER values(2311,2122,'04/24/2023');
3 end;
4
```

The 'Run' button is highlighted. Below the command editor, the 'Results' tab shows the output:

```
rows before insert : 23
1 row(s) inserted.
0.02 seconds
```

The bottom status bar is identical to the first screenshot.

The trigger code first declares a variable named `rows_nums` of type `NUMBER`. Inside the trigger, a `SELECT COUNT(*)` statement is executed to count the number of rows in the `BUYER` table and the result is stored in the `rows_nums` variable.

Finally, the trigger code uses the `dbms_output.put_line` function to display a message to the console showing the number of rows in the `BUYER` table before the insert operation is executed.

Therefore, this trigger will be fired every time a row is inserted into the `BUYER` table, and it will display the number of rows in the table before the insert operation is performed.

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. The user is logged in as 'Daulet Nurgali shymkent'. The 'SQL Commands' panel is active, showing a query to select products that do not have corresponding order items. The query is as follows:

```
1 SELECT
2 FROM product
3 WHERE NOT EXISTS (
4   SELECT 1
5   FROM orderitem
6   WHERE orderitem.product_id = product.product_id
7 );
```

The 'Results' panel shows the output of the query, which is a table with four columns: `PRODUCT_ID`, `NAME`, `PRICE`, and `SELLER_ID`. The table contains three rows, all with `PRODUCT_ID` 259 and `NAME` BEN. The `PRICE` is 89150 and the `SELLER_ID` is 8092390835 for all rows. Below the table, it indicates '3 rows returned in 0.11 seconds' and provides a 'Download' link.

PRODUCT_ID	NAME	PRICE	SELLER_ID
259	BEN	89150	8092390835
259	BEN	89150	8092390835
259	BEN	89150	8092390835

3 rows returned in 0.11 seconds [Download](#)

The footer of the interface shows the user's email '210103096@stu.sdu.edu.kz', the username 'shymkent', the language 'en', and the copyright notice 'Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.1.0-15'.



**This SQL query is selecting all rows from the seller table where there does not exist any product sold by the seller within the last 60 days.**

**The query uses a subquery to check if there exists any product sold by the seller within the last 60 days. The subquery joins the product, orderitem, and buyer tables on their foreign key relationships to get the necessary data. It filters the results based on the seller\_id of the seller in the main query and the order\_date of the buyer table.**

**The NOT EXISTS operator in the main query checks if the subquery returns any rows. If the subquery returns no rows, it means that there does not exist any product sold by the seller within the last 60 days, and the seller is selected by the main query.**

**In summary, this query selects all sellers who have not sold any products within the last 60 days.**

APEX

App Builder

SQL Workshop

Team Development

Gallery

Search

DN

Daulet Nurgali

shymkent

SQL Commands

SchemaWKSP\_SHYMKENT

LanguageSQL

Rows10

Clear Command

Find Tables

Save

Run

A:

1

2

3

4

5

6

7

8

9

10

11

12

SELECT \*

FROM seller

WHERE NOT EXISTS (

SELECT 1

FROM product

INNER JOIN orderitem ON product.product\_id = orderitem.product\_id

INNER JOIN buyer ON orderitem.order\_id = buyer.order\_id

WHERE product.seller\_id = seller.seller\_id

AND buyer.order\_date >= SYSDATE - 60

)

Results

Explain

Describe

Saved SQL

History

SELLER_ID	NAME	EMAIL	PHONE_NUMBER
8092390835	Petey	pdinsell0@phpbb.com	234-507-6993
569901715	Inglis	iangerson1@macromedia.com	884-403-3074
7263825570	Gerek	gmurfill2@thetimes.co.uk	784-156-6708
3172226212	Lanie	lsiddle3@drupal.org	300-153-8804
217750311	Georas	gpurvey4@youtube.com	751-936-8872
6292197441	Allegra	ahelwig5@un.org	225-259-0509
353670243	Gennifer	gmaccarroll7@army.mil	515-256-6243
7093667015	Thorny	touldcott8@hatena.ne.jp	803-529-5781

210103096@stu.sdu.edu.kz

shymkent

en

Copyright © 1999, 2023, Oracle and/or its affiliates.

Oracle APEX 23.1.0-15