SDU 27.04.2023

Midterm 2 (BOOK SHOP)

Database systems

Presented By

Shakman Amina
Bekbossinova Ayaulym
Madina Shanshar
Darkhan Seisekenov
Bexultan Kussainov

Presentation content

1 Introduction to system

2 ER Diagram

3 Normalization

4 Procedure which does group by information

5 Function

6 Procedure which uses SQL%ROWCOUNT

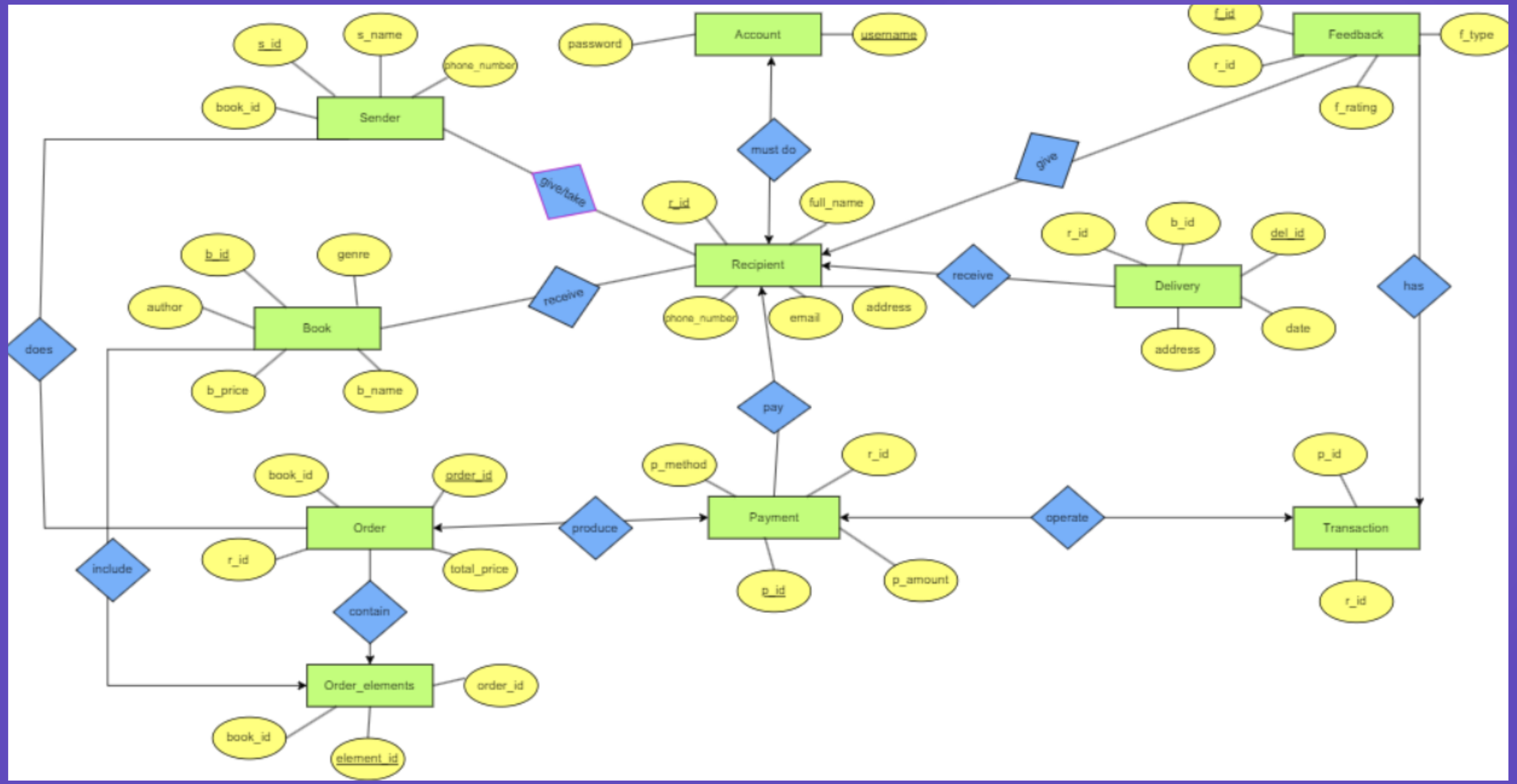
7 Exception

8 Triggers

Introduction

We are creating a database for a bookstore that allows you to manage orders, inventory and customer information. The system also makes reports on how much books are sold and how many are left. It's safe and secure. With our project, the bookstore can sell books better and give customers a better experience.

ER Diagram


s_id
s_name
phone_number
book_id
Sender
password
Account
username
must do
f_id
Feedback
f_type
r_id
f_rating
give
give/take
r_id
full_name
Recipient
b_id
genre
author
Book
b_price
b_name
receive
phone_number
email
address
receive
r_id
b_id
del_id
Delivery
address
date
has
does
pay
p_method
r_id
Payment
p_id
p_amount
operate
p_id
Transaction
r_id
book_id
order_id
Order
r_id
total_price
produce
include
contain
Order_elements
order_id
book_id
element_id

1.Account entity set means your user account in our online bookshop. Each account has a unique username and password. That is, the username and password are attributes of this entity set. Account and recipient are connected with a “must do”' relationship, which means that the recipient must have an account on the online bookshop in order to receive the order.

2.Sender entity set means the person who sends the book to the recipient. And it also has unique s_id, book_id, s_name, phone_number attributes. Sender and recipient connected with a “give/take” relationship, which means that the sender hands over the book to the recipient.

3.Recipient entity set means the person who receives the book from the sender. The recipient has attributes like that: a unique r_id, full_name, address, email and phone_number. Recipient and delivery are related by a “receive” relationship, which means that the recipient receives the delivery of the book.

4. Book entity set means a book that is available in our online bookstore. Each book has a unique b_id, author, genre, b_name and b_price. Book and recipient are connected with a “receive” relationship, which means that the recipient receives the book when it is purchased.

5. Order entity set means a recipient order for a book. The order includes a unique order_id, book_id, r_id, and total_price attributes. Order and sender are connected with a "does" relationship, which means that the sender does/makes the order.

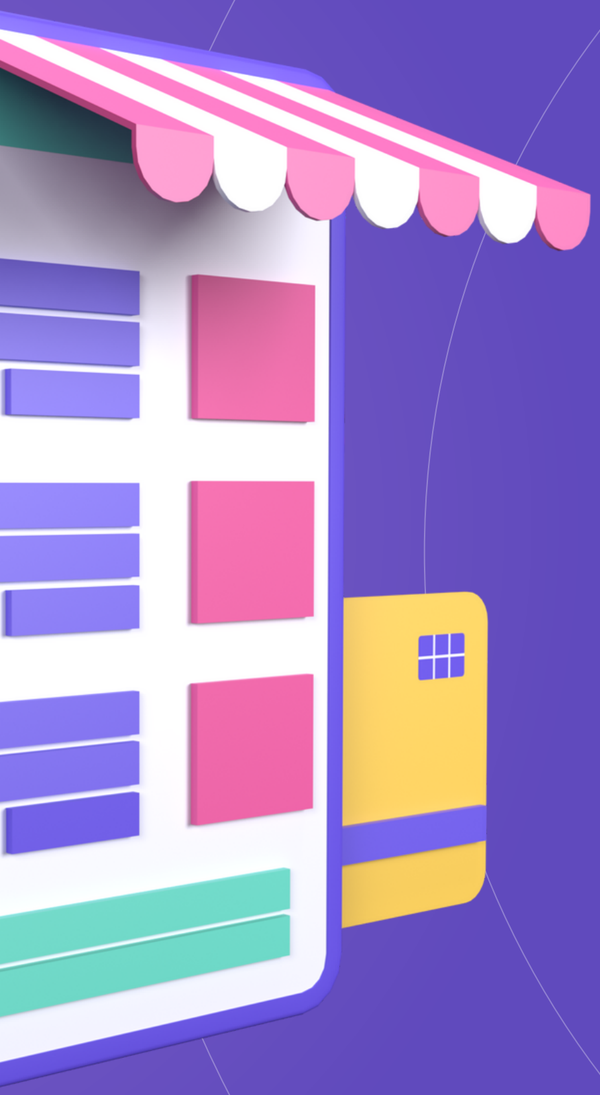
6. Order_elements entity set represents the elements included in the order. Order_elements and order connected with relationship “contain”. Order_elements include a unique element_id, order_id, and book_id. Order_elements and book have an "include" relationship, which means that the order_elements includes the book.

7. Payment entity set means the recipient’s payment for the order. A payment includes a unique p_id, r_id, p_method and p_amount. A payment and a transaction are connected with an "operate" relationship.

8. Transaction entity set is a financial transaction between our online book shop and a recipient. The transaction includes attributes like that: p_id and r_id. Transaction and payment are connected with an "operate" relationship, which means that the transaction is controlled by the payment.

9.Delivery entity set represents the delivery of a book to a recipient. Delivery has attributes like a unique del_id, b_id, r_id, date and address. Delivery and recipient are connected with a "receive" relationship, which means that the recipient receives the delivery of the book.

10. Feedback entity set refers to recipient reviews about the online book shop. Feedback has attributes: a unique f_id, r_id, f_rating, f_type. Feedback and recipient are connected with a "give" relationship.



NORMALIZATION

1NF, 2NF, 3NF

1.Book

Attributes: B_ID (Primary key), Author, B_Name, B_Price, Genre
Functional Dependencies: B_ID -> Author, B_Name, B_Price, Genre
Superkey: {B_ID}, {B_ID, Author, B_Name, B_Price, Genre}
Candidate Key: B_ID
Primary key: B_ID
State: 3NF
• There are no multi-valued attributes. Each column and row intersection have only one value.
• There are no partial dependencies. It has one unique attribute, which is
Book_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
• There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

2.Sender

Attributes: S_ID (Primary key), S_Name, B_ID (FK), Phone
Functional Dependencies: S_ID-> S_Name, B_ID, Phone
Superkey: {S_ID}, {S_ID, B_ID}
Candidate Key: S_ID
Primary key: S_ID
State: 3NF
• There are no multi-valued attributes. Each column and row intersection have
only one value.
• There are no partial dependencies. It has one unique attribute, which is
Sender_ID, that identifies each tuple, and no non-key attributes depend only on a
part of the primary key.
• There are no transitive dependencies. No non-primary-key attribute is
transitively dependent on the primary key.

3. Recepient

Attributes: R_ID (Primary key), Full_Name, Phone, Email, Address
Functional Dependencies: R_ID -> Full_Name,Phone, EMAIL, Address
Superkey: {R_ID}, {Email}
Candidate Key: R_ID
Primary key: R_ID
State: 3NF
• There are no multi-valued attributes. Each column and row intersection have only one value.
• There are no partial dependencies. It has one unique attribute, which is RECEPIENT_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
• There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

4. Account

Attributes: Username (Primary key), Password
Functional Dependencies: Username -> Password
Superkey: {Username}
Candidate Key: Username
Primary key: Username
State: 3NF
• There are no multi-valued attributes. Each column and row intersection have only one value.
• There are no partial dependencies. It has one unique attribute, which is Username, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
• There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

1 Introdu
2 ER Diag

5. Order

Attributes: ORDER_ID (Primary key), R_ID(FK), B_ID(FK), Total_Price
Functional Dependencies: ORDER_ID -> R_ID, B_ID, Total_Price
Superkey: {ORDER_ID}, {ORDER_ID, R_ID, B_ID}
Candidate Key: ORDER_ID
Primary key: ORDER_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.
- There are no partial dependencies. It has one unique attribute, which is Order_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

6. Order_elements

Attributes: ELEMENT_ID (Primary key), ORDER_ID(FK), B_ID(FK)
Functional Dependencies: ELEMENT_ID -> ORDER_ID, B_ID
Superkey: {ELEMENT_ID}, {ELEMENT_ID, ORDER_ID, B_ID}
Candidate Key: ELEMENT_ID
Primary key: ELEMENT_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.
- There are no partial dependencies. It has one unique attribute, which is Element_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

7. Payment

Attributes: P_ID (Primary key), R_ID(FK), P_Amount, P_Method
Functional Dependencies: P_ID -> R_ID, P_Amount, P_Method
Superkey: {P_ID}, {P _ID, R_ID}
Candidate Key: P_ID
Primary key: P_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.
- There are no partial dependencies. It has one unique attribute, which is Payment_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

8. Delivery

Attributes: D_ID (Primary key), B_ID(FK), R_ID(FK), D_Address, D_Date
Functional Dependencies: D_ID -> B_ID, R_ID, D_Address, D_Date
Superkey: {D_ID}, {D_ID, B_ID, R_ID}
Candidate Key: D_ID
Primary key: D_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.
- There are no partial dependencies. It has one unique attribute, which is Delivery_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

9. Feedback

Attributes: F_ID (Primary key), R_ID(FK), F_Type, F_Rating
Functional Dependencies: F_ID -> R_ID, F_Type, F_Rating
Superkey: {F_ID}, {F_ID, R_ID}
Candidate Key: F_ID
Primary key: F_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.
- There are no partial dependencies. It has one unique attribute, which is Feedback_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

10. Transaction

Attributes: P_ID(FK), R_ID(FK), T_ID
Functional Dependencies: T_ID -> P_ID, R_ID
Superkey: {T_ID}
Candidate Key: T_ID
Primary key: T_ID
State: 3NF

- There are no multi-valued attributes. Each column and row intersection have only one value.

As we see here, we have two foreign keys, but for the 2NF form our attributes should be depended on primary key, so we should add one more attribute, which we named T_ID and our attributes will have functional dependencies with this PK.
And our tables look like this:

TRANSACTION	
P_ID	R_ID

R_ID	T_ID

P_ID	T_ID

- There are no partial dependencies. It has one unique attribute, which is TRANSACTION_ID, that identifies each tuple, and no non-key attributes depend only on a part of the primary key.
- There are no transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

Procedures

```sql
CREATE OR REPLACE PROCEDURE check_account_exists(
    p_username IN Account.username%TYPE,
    p_password IN Account.password%TYPE
)  AS  v_count NUMBER;

BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM Account
    WHERE username = p_username
    AND password = p_password;

    IF v_count = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Account exists.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Account does not exist.');
    END IF;
END;
```

This procedure takes in a username and password as input parameters and checks if the account exists in the Account table.

The procedure would call the check_account_exists procedure with the username "jfranzen0" and password "eoldacre0".

```sql
BEGIN
    check_account_exists('jfranzen0','eoldacre0');
END;
```

Procedures

```
1   CREATE OR REPLACE PROCEDURE get_book_name(
2       p_b_id IN Book.b_id%TYPE
3   ) AS book_name Book.b_name%TYPE;
4   BEGIN
5       SELECT b_name into book_name
6       FROM Book
7       WHERE b_id = p_b_id;
8       DBMS_OUTPUT.PUT_LINE('Book name: ' || book_name);
9
10  EXCEPTION
11      WHEN no_data_found THEN
12          DBMS_OUTPUT.PUT_LINE('Such a book does not exist');
13  END;
```

This procedure takes in a book_id as input and returns the book_name as an output parameter.

The procedure would call the get_book_name procedure with a book_id of 4941700007 and retrieve the corresponding book name, which would be stored in the v_book_name variable and prints out the book name.

```
1   BEGIN
2       get_book_name(4941700007);
3   END;
```

Procedures

```
1   CREATE OR REPLACE PROCEDURE update_order_total_price(
2       p_order_id IN OUT Orders.order_id%TYPE,
3       p_total_price OUT Orders.total_price%TYPE
4   ) AS
5   BEGIN
6       UPDATE Orders
7       SET total_price = total_price * 1.1
8       WHERE order_id = p_order_id
9       RETURNING total_price INTO p_total_price;
10  END;
```

This procedure takes in an order_id as input and updates the total_price in the Order table. It also returns the updated total_price as an output parameter.

The procedure would call the update_order_total_price procedure with an order_id of 456 and update the corresponding order's total price by 10%. It would then store the new total price in the v_total_price variable and use DBMS_OUTPUT.PUT_LINE to print out both the order ID and the new total price.

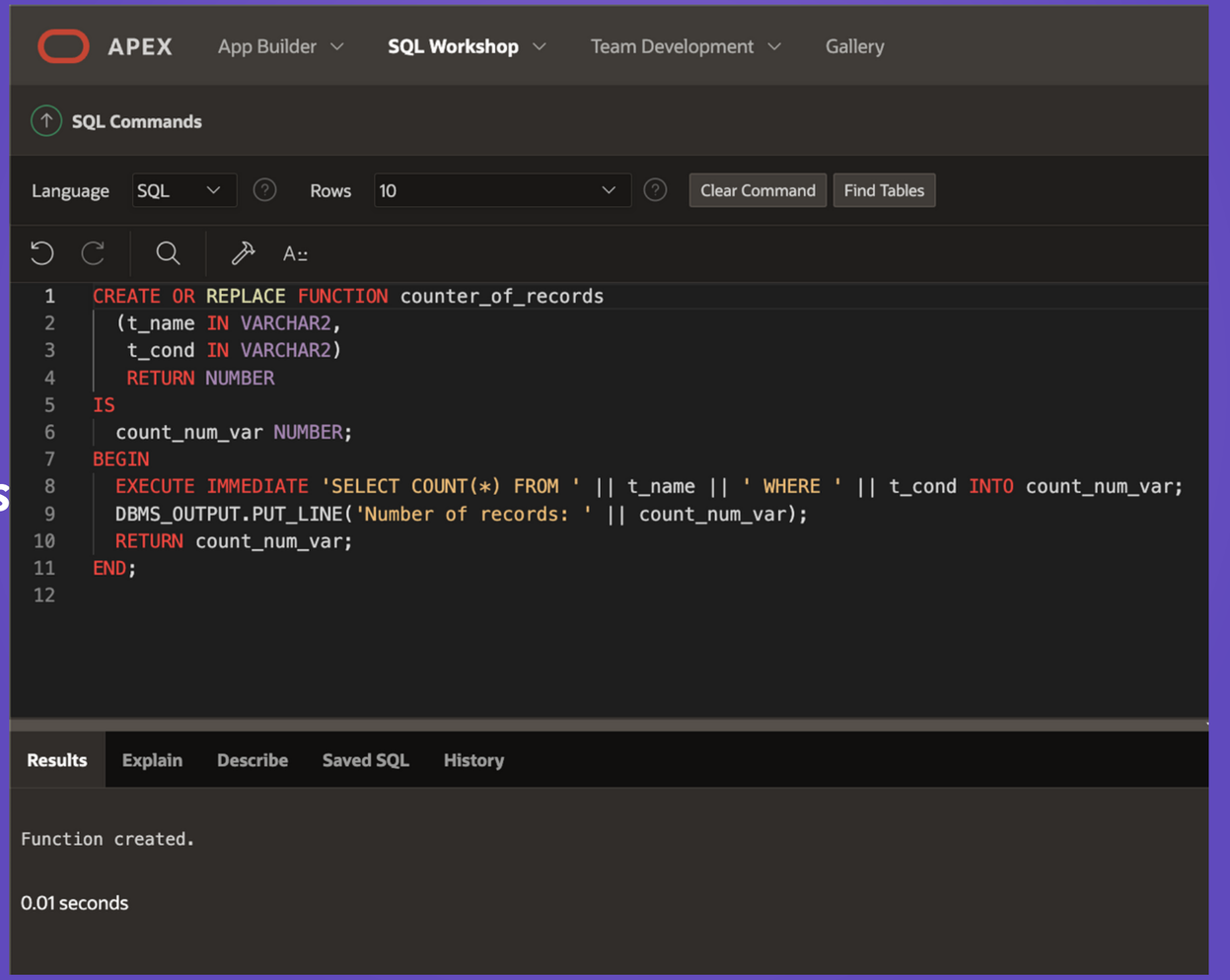
```
1   DECLARE
2       v_order_id Orders.order_id%TYPE := 456;
3       v_total_price Orders.total_price%TYPE;
4   BEGIN
5       update_order_total_price(v_order_id, v_total_price);
6       DBMS_OUTPUT.PUT_LINE('Order ID: '  v_order_id  ', new total price: ' || v_total_price);
7   END;
```

Function

REPORT

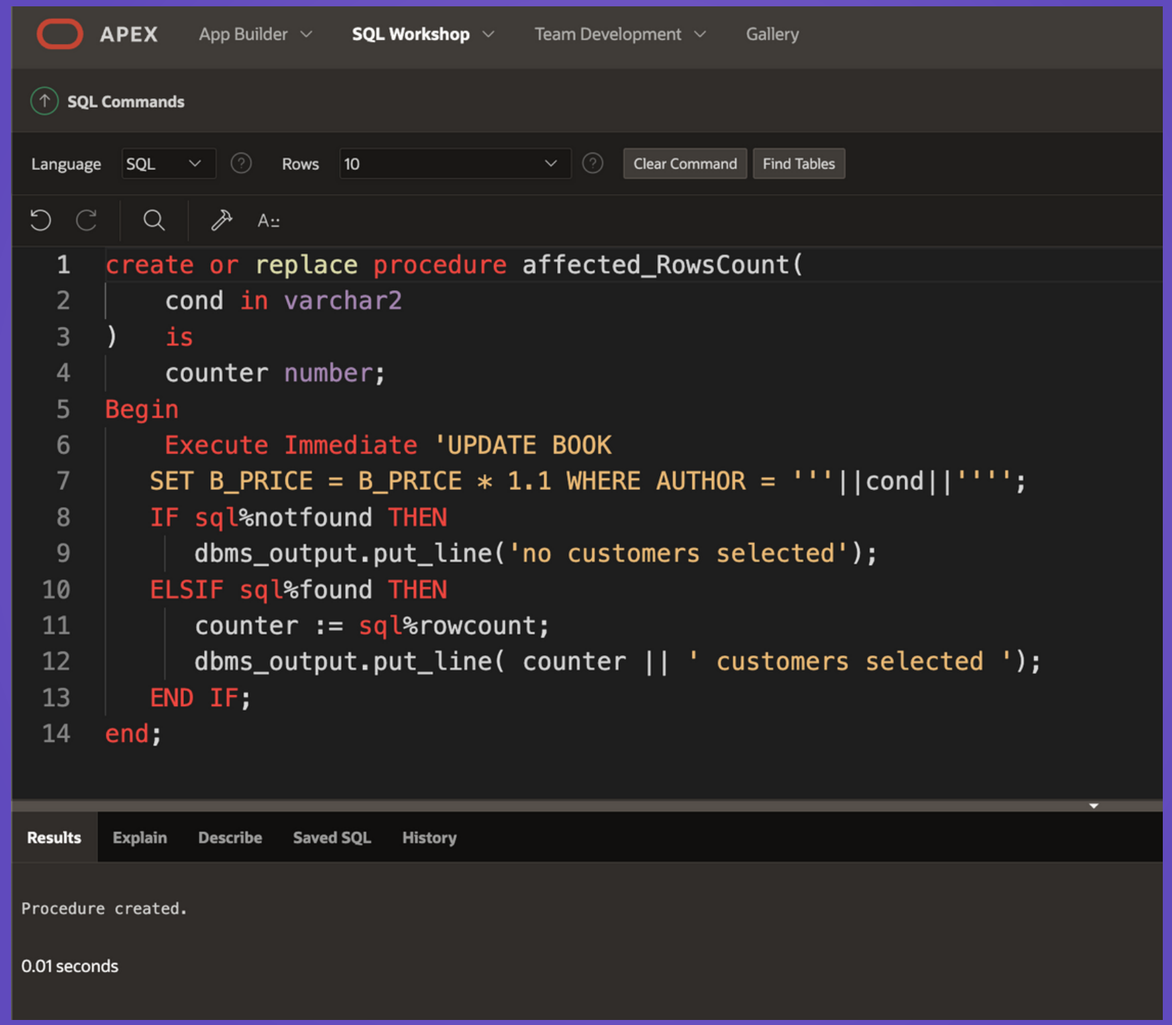
This code is an function that takes two string parameters t_name and t_cond, representing the name of a table and a condition to be used in a select statement, respectively. ***The function returns the count of rows in the table that match the given condition.***

- Function which counts the number of records



ROWCOUNT

This code is a procedure that takes a string parameter named 'cond', which represents a condition to be used in an update statement. The procedure updates the 'B_PRICE' field of the "BOOK" table, increasing it by 10%, for all rows where the "AUTHOR" field matches the 'cond' parameter. ***The number of rows affected by the update is then counted and displayed to the user using the dbms_output package.***



Exception

```
declare
  ss_name varchar2(50);
  sdr sender.s_name%type;
  impermissible_ex exception;

begin
  ss_name:= :s_name;
  if  length(ss_name) < 5 then
  raise impermissible_ex;
else
  select s_name into sdr from sender where s_name = ss_name;
  dbms_output.put_line('Sender with this name exists.');
end if;

exception
  when impermissible_ex then
   raise_application_error(-20001, 'The name of sender must contain at least 5 characters.');
  when no_data_found then
   dbms_output.put_line('Invalid name of sender.');
  when others then
   dbms_output.put_line('Try again');
end;
```

REPORT

This is a PL/SQL code block that declares a variable ss_name of type varchar2 and a variable sdr of the same data type as the s_name column in the sender table. The code block also defines user-defined exception called impermissible_ex.

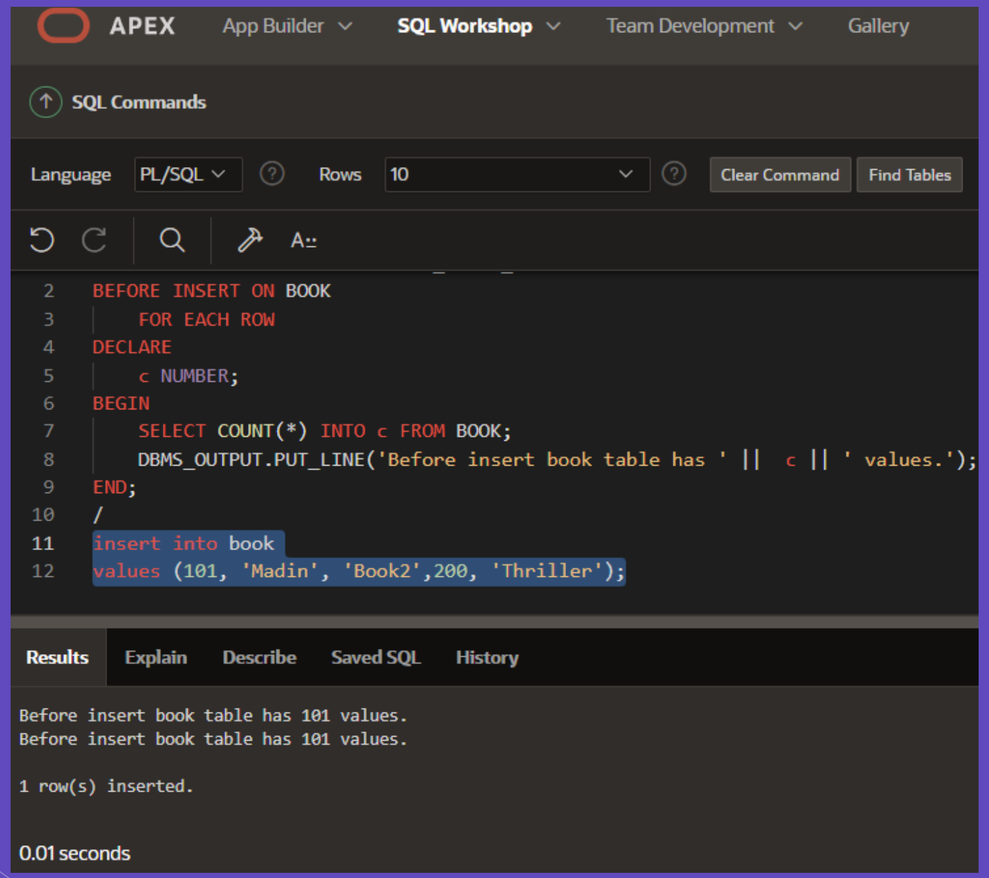
To achieve this, the code block first assigns the value of the input parameter :s_name to the variable ss_name. Then it checks if the length of ss_name is less than 5. If it is, the code block raises impermissible_ex exception with an error message indicating that the name of the sender must contain at least 5 characters.

If the length of ss_name is greater than or equal to 5, the code block queries the sender table to see if a sender with that name exists. If it exists, the code block outputs a message stating that the sender with that name exists. If it does not exist, the no_data_found exception is raised and an appropriate message is output.

If any other exception occurs during the execution of the code block, a generic message is output.

TRIGGERS

REPORT

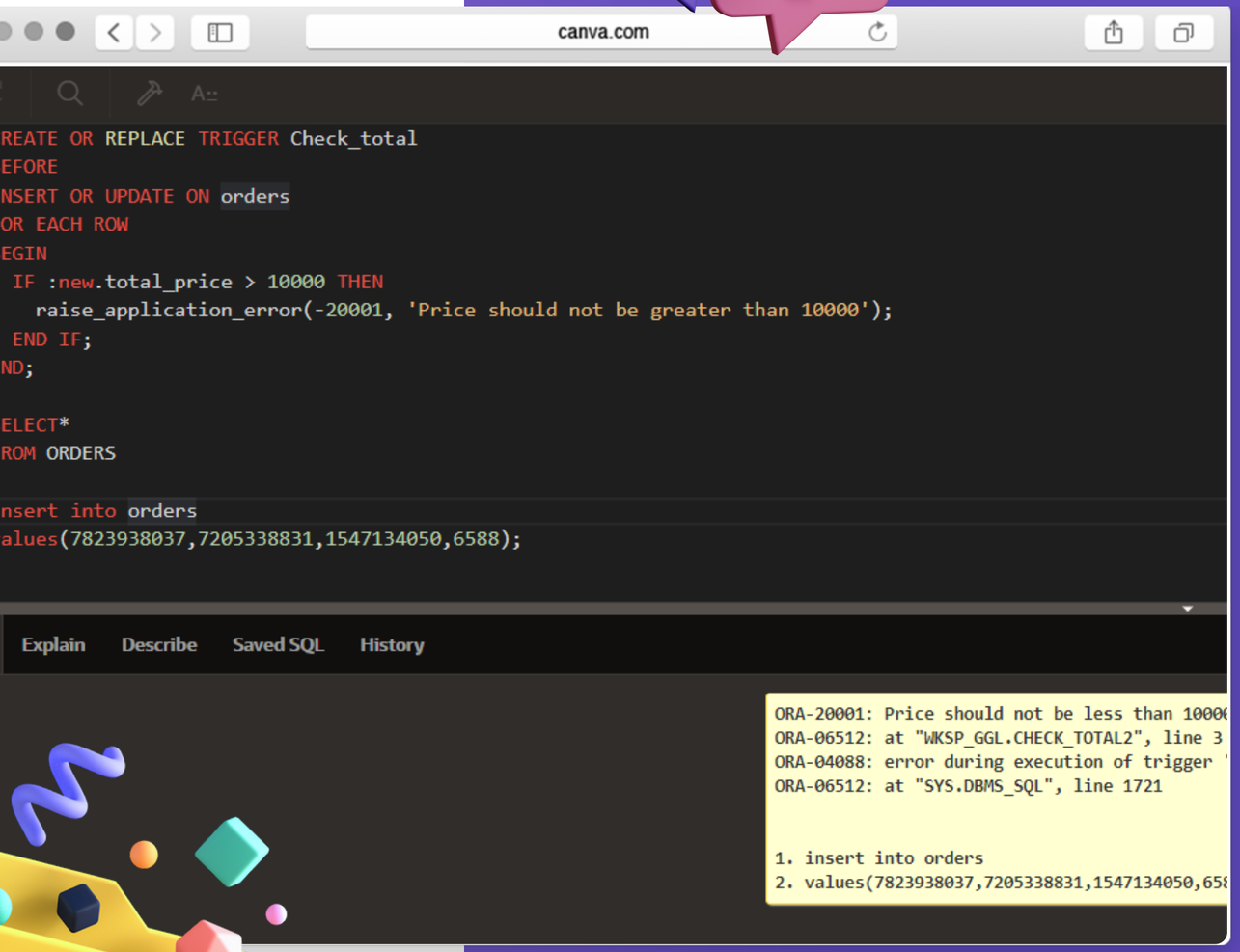


This is a trigger in the Apex that is designed to execute automatically before an insert operation is performed on the BOOK table. The trigger is defined to execute for each row that is inserted into the BOOK table.The trigger starts with a declaration section that defines a variable c of type NUMBER. Then,Inside the BEGIN and END block of the trigger, a SQL statement is used to select the count of rows in the BOOK table and assign it to the variable c. Finally, the DBMS_OUTPUT.PUT_LINE statement is used to display the number of rows in the BOOK table before a new row is inserted.

When a new row is inserted into the BOOK table, this trigger will fire and display the count of rows in the BOOK table before the insertion is performed. In this case, the INSERT statement inserts a new row into the BOOK table and sets the values for the columns B_ID,AUTHOR, B_NAME,B_PRICE and GENRE .As a result, before the new row is inserted into the BOOK table, the trigger will execute and display the number of rows in the BOOK table at that time.

Back to Agenda

TRIGGERS



REPORT

This is a PL/SQL code that defines a trigger named "Check_total". The trigger is defined to fire "BEFORE" each "INSERT" or "UPDATE" operation on the "orders" table. The purpose of the trigger is to check the value of the "total_price" column in the "orders" table. If the value is greater than 10000, it will raise an application error with the message "Price should not be greater than 10000".The "insert into orders" statement at the end is an example of how to insert a new row into the "orders" table.

However, since the trigger is defined to check the "total_price" value before any insert or update operation, it will prevent the insertion of the new row if the value of the "total_price" column is greater than 10000.

Back to Agenda

REPORT

TRIGGERS

This PL/SQL trigger named "prevent_high_price_deletion" is created or replaced on the "BOOK" table. The trigger is set to fire before a delete operation is performed on the "BOOK" table, for each row being deleted.

The trigger has a condition that checks if the value of the "B_PRICE" column of the old row (the row being deleted) is greater than 4000. If the condition is true, then it raises an error with the message "Cannot delete books with high price" and a code of -20001.

This means that if someone tries to delete a row from the "BOOK" table with a "B_PRICE" value greater than 4000, the trigger will prevent the deletion and return an error message.

In summary, this trigger prevents the deletion of high-priced books from the "BOOK" table to maintain data integrity.

```
CREATE OR REPLACE TRIGGER prevent_high_price_deletion
BEFORE DELETE ON BOOK
FOR EACH ROW
BEGIN
IF :OLD.B_PRICE > 4000 THEN
RAISE_APPLICATION_ERROR(-20001, 'Cannot delete books with high price');
END IF;
END;
DELETE FROM book WHERE b_id = 843835877;
```

ts Explain Describe Saved SQL History

```
ORA-20001: Cannot delete books with high pric
ORA-06512: at "WKSP_GGL.PREVENT_HIGH_PRICE_DE
ORA-04088: error during execution of trigger
ORA-06512: at "SYS.DBMS_SQL", line 1721
```