

# **DBMS - 2**

# Our Team



**Temirlan Arystan**

Student



**Nurkhat Muratkan**

Student



**Dias Gaziz**

Student



**Sabyrzhan Rustembekov**

Student

**Dias Gaziz**

**Sabyrzhan Rustembekov**

**Nurkhat Muratkhan**

**Temirlan Arystan**

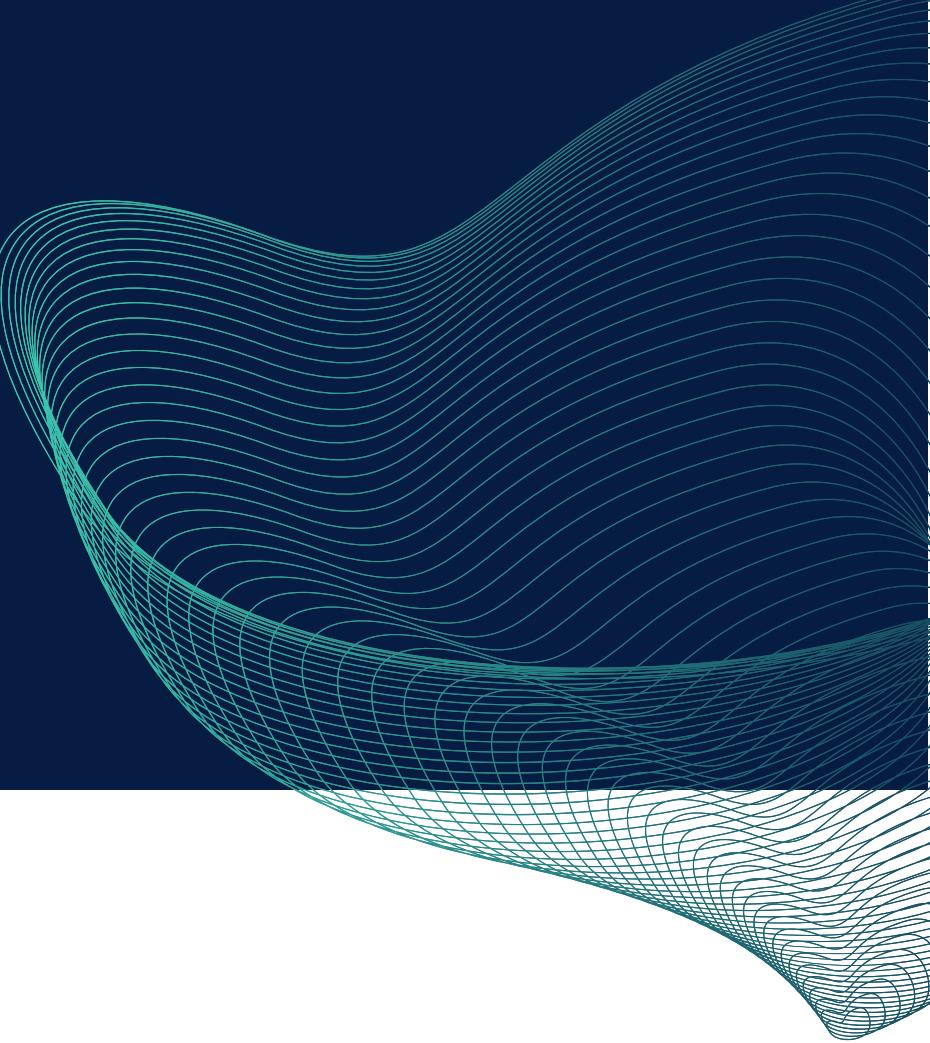
# Online Book Store

## Description

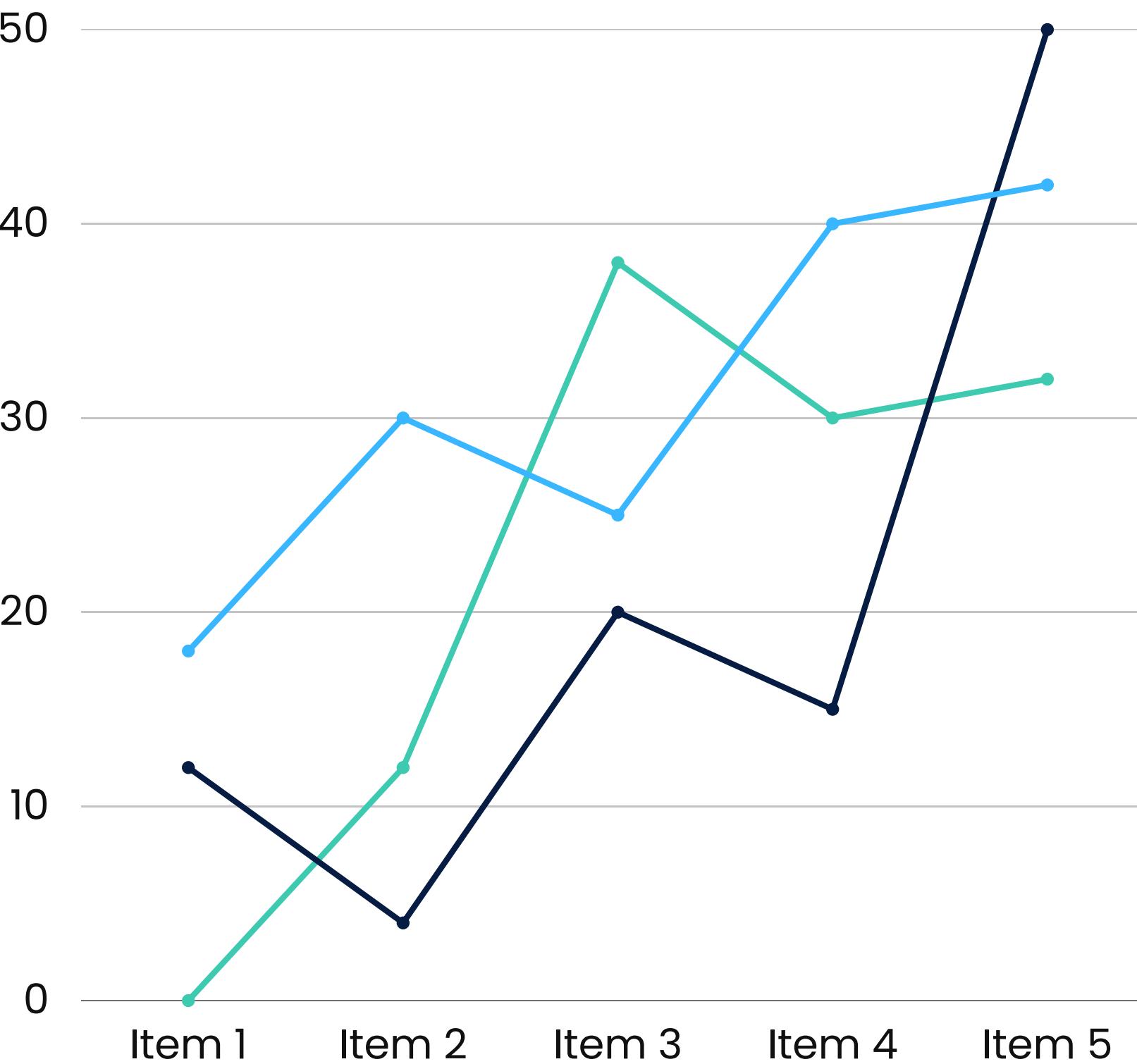
Our database allows the online bookstore to improve its activities. Facilitates management. The database tracks all events.

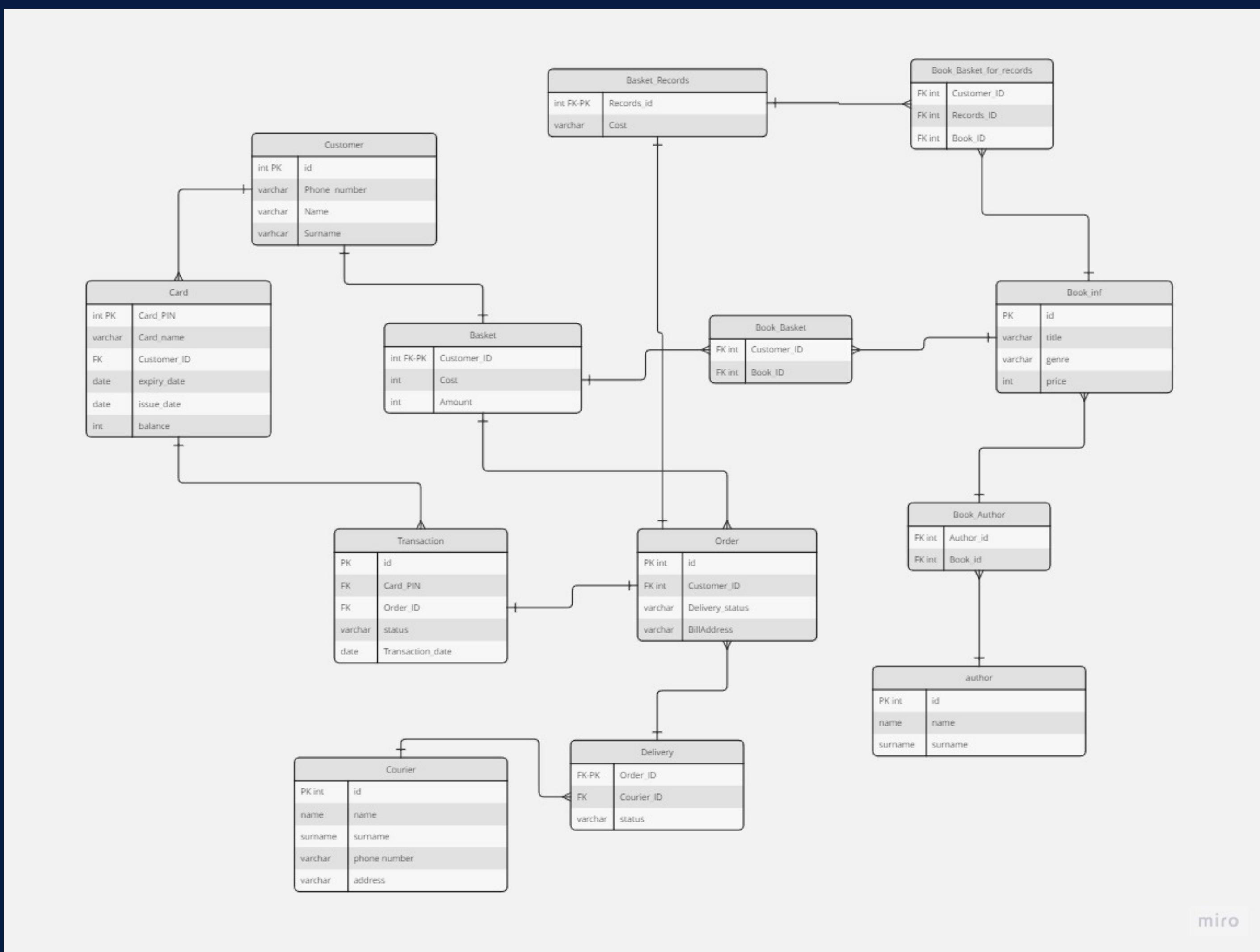
To do this, we have tables: Book\_info, Customers, Basket, Book\_Basket, Order, Delivery, Courier, Author, Book\_Author, Card, Payment(Transactions), Basket\_Records, Book\_Basket\_for\_records.

These tables perform the functions of courier delivery pickup or courier, income, sort by book authors, and much more. The end user of the database is the customer who buys the book. The problem of data obsolescence will be solved with the help of triggers. We came up with the idea of the project, and the Internet became the main source of resources.



# ER-diagram





# Procedures



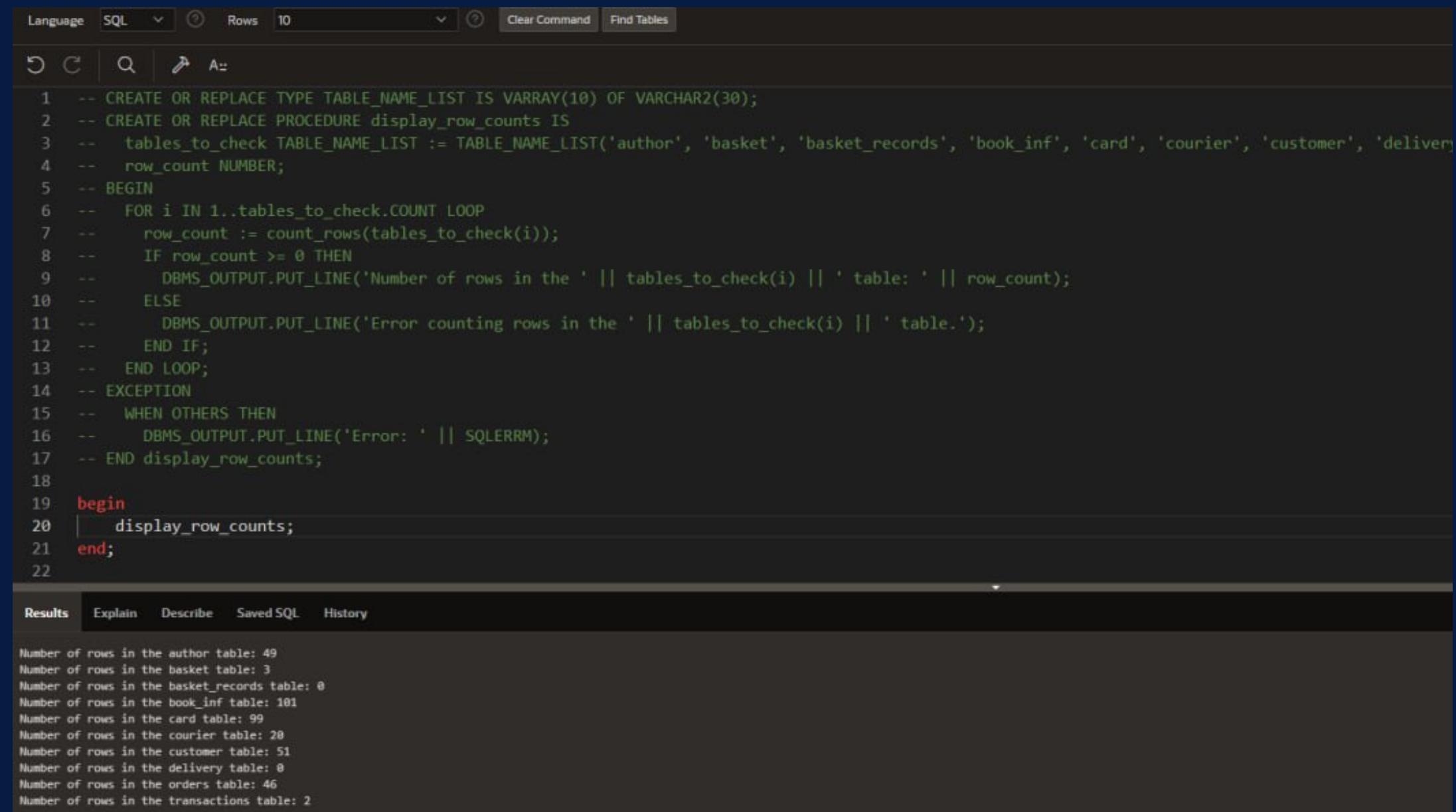
This procedure removes row from Transaction, Basket, Book\_basket tables if the transaction status is set to "In expectation" and the time of adding to the basket has exceeded 3 days.

---

```
create or replace PROCEDURE delete_unpaid_orderss AS
  CURSOR c_order IS
    SELECT tr.order_id, tr.status, od.delivery_status, od.customer_id
      FROM Transactions tr
      INNER JOIN Delivery dl
        ON dl.order_id = tr.order_id
      INNER JOIN Orders od
        ON od.id = dl.order_id
    WHERE tr.transaction_date < (SYSDATE - 3)
      AND tr.status = 'In expectation';
  BEGIN
    FOR order_rec IN c_order LOOP
      DELETE FROM Book_basket WHERE customer_id = order_rec.customer_id;
      DELETE FROM Basket WHERE customer_id = order_rec.customer_id;
      IF order_rec.delivery_status = 'courier' then
        DELETE FROM Delivery WHERE order_id = order_rec.order_id;
      END IF;
      DELETE FROM Transactions WHERE order_id = order_rec.order_id;
      DBMS_OUTPUT.PUT_LINE('Order deleted is id: ' || order_rec.order_id);
    END LOOP;
  END;
```

# Displays the row sum of all tables

```
create or replace PROCEDURE display_row_counts IS
    tables_to_check TABLE_NAME_LIST := TABLE_NAME_LIST('author', 'basket', 'basket_records',
'book_inf', 'card', 'courier', 'customer', 'delivery', 'orders',
'transactions');
    row_count NUMBER;
    BEGIN
        FOR i IN 1..tables_to_check.COUNT LOOP
            row_count := count_rows(tables_to_check(i));
            IF row_count >= 0 THEN
                DBMS_OUTPUT.PUT_LINE('Number of rows in the ' ||
                    tables_to_check(i) || ' table: ' || row_count);
            ELSE
                DBMS_OUTPUT.PUT_LINE('Error counting rows in the ' ||
                    tables_to_check(i) || ' table.');
            END IF;
        END LOOP;
        EXCEPTION
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        END display_row_counts;
```



The screenshot shows the Oracle SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Clear Command, Find Tables
- Code Area:** Displays the PL/SQL code for the `display_row_counts` procedure.
- Results Area:** Displays the output of the procedure execution, listing the number of rows for each table in the specified list.

Table	Number of Rows
author	49
basket	3
basket_records	0
book_inf	101
card	99
courier	28
customer	51
delivery	0
orders	46
transactions	2

The procedure is done on sql%rowcount. This procedure does update by specifying customer\_id, we do update amount book

---

```
create or replace PROCEDURE update_book_amount (
    p_customer_id IN NUMBER,
    p_new_amount IN NUMBER)
IS BEGIN
    UPDATE basket
    SET amount = p_new_amount
    WHERE customer_id = p_customer_id;

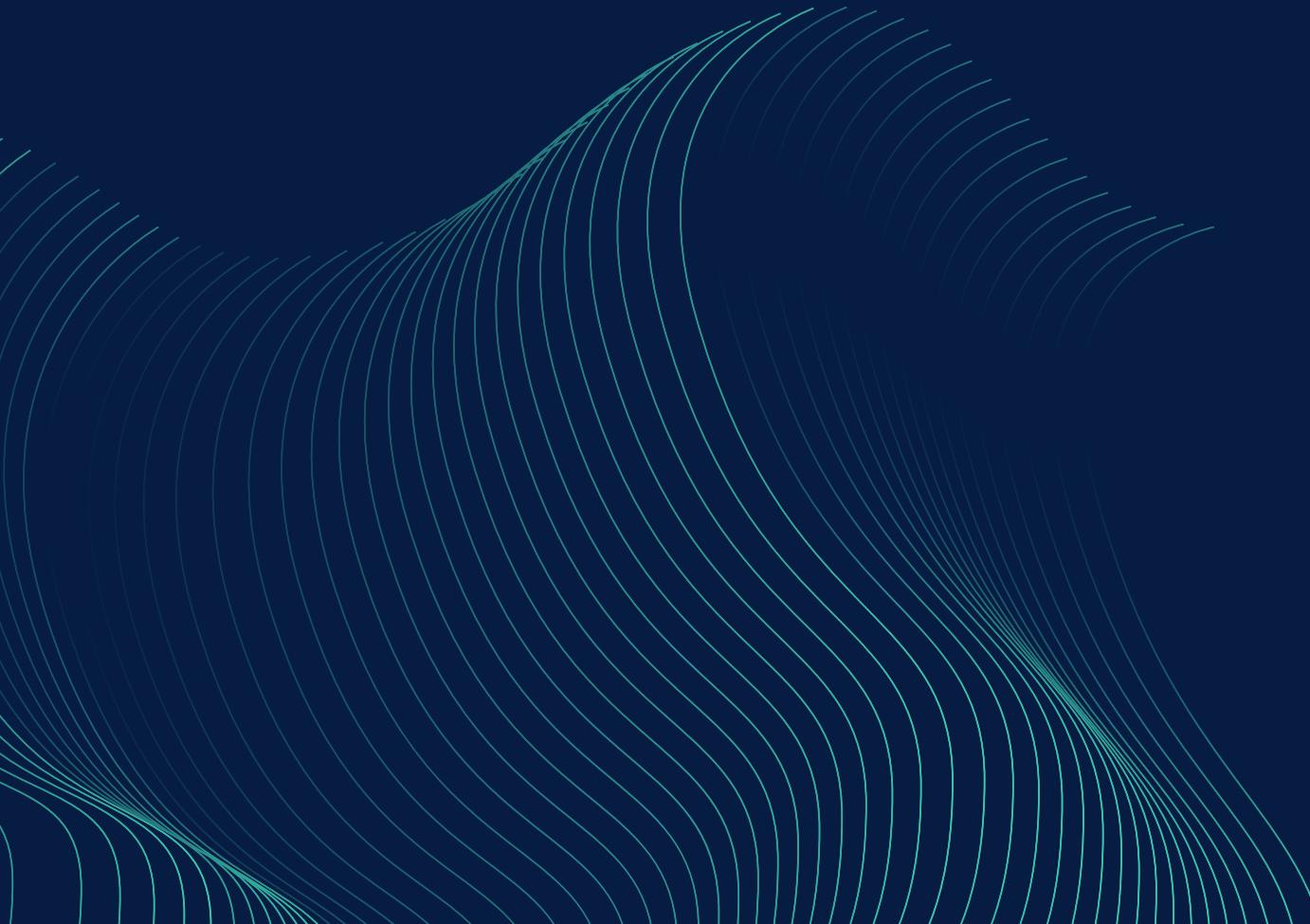
    IF SQL%ROWCOUNT > 0 THEN
        DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No rows updated.');
    END IF;
END;
```

For example:

```
BEGIN
update_book_amount(4, 7);
END;
```

---

# Packages



# This package totals a row of the table

## Specification:

```
create or replace PACKAGE total_profit AS
    PROCEDURE print_total;
END total_profit;
```

## BODY:

```
create or replace PACKAGE BODY total_profit AS
```

```
    PROCEDURE print_total IS
        sum_price NUMBER(10);
    BEGIN
```

```
        SELECT SUM(cost) INTO sum_price FROM basket_records;
        dbms_output.put_line('total profit is: ' || TO_CHAR(sum_price));
    END print_total;
```

```
END total_profit;
```

```
begin
total_profit.print_total;
end;
```

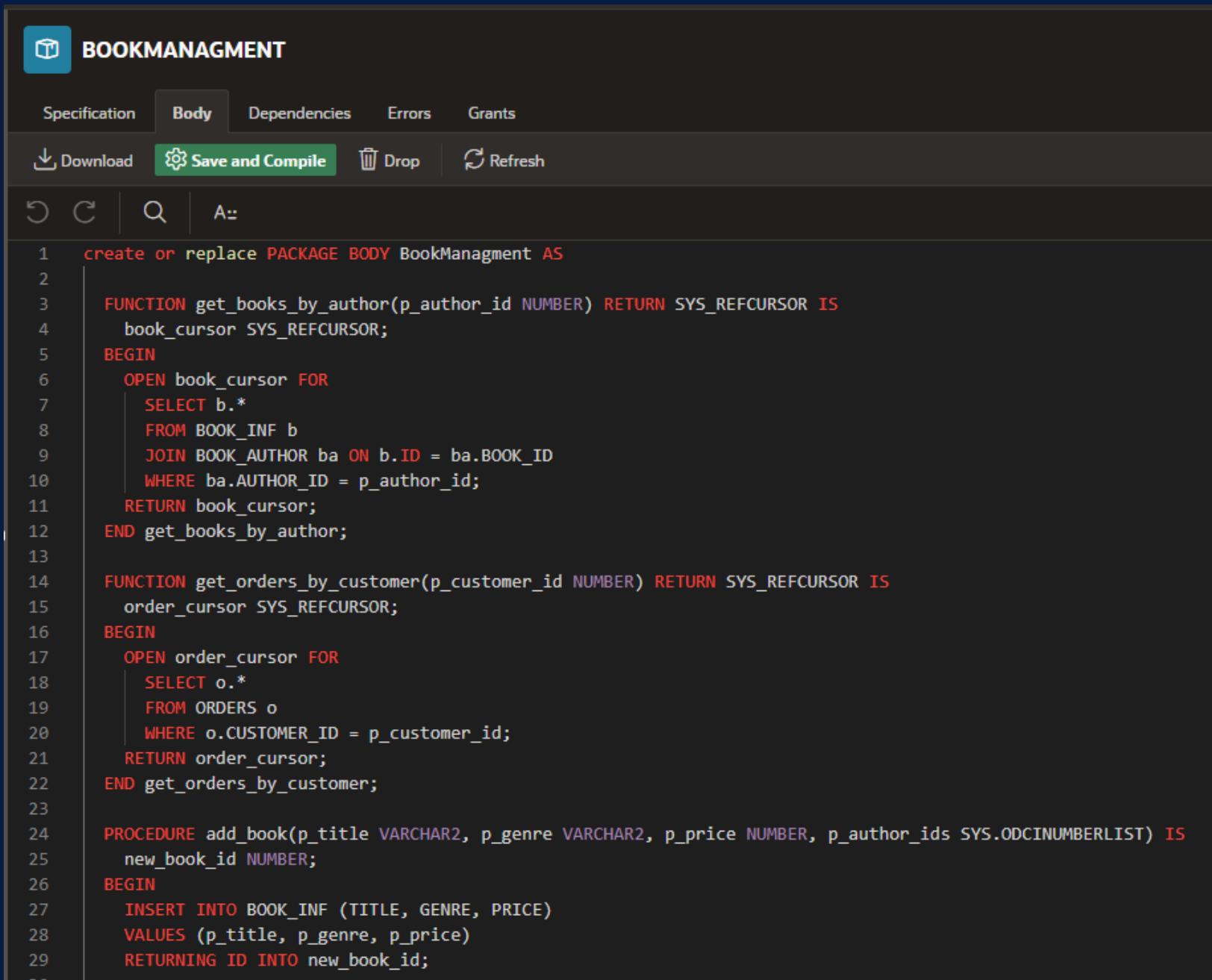
The screenshot shows a SQL developer interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables.
- Code Area:** Contains the following PL/SQL code:

```
1 begin
2     total_profit.print_total;
3 end;
```
- Results Tab:** Shows the output of the executed code:

```
total profit is: 207133
Statement processed.
```
- Timing:** 0.01 seconds.

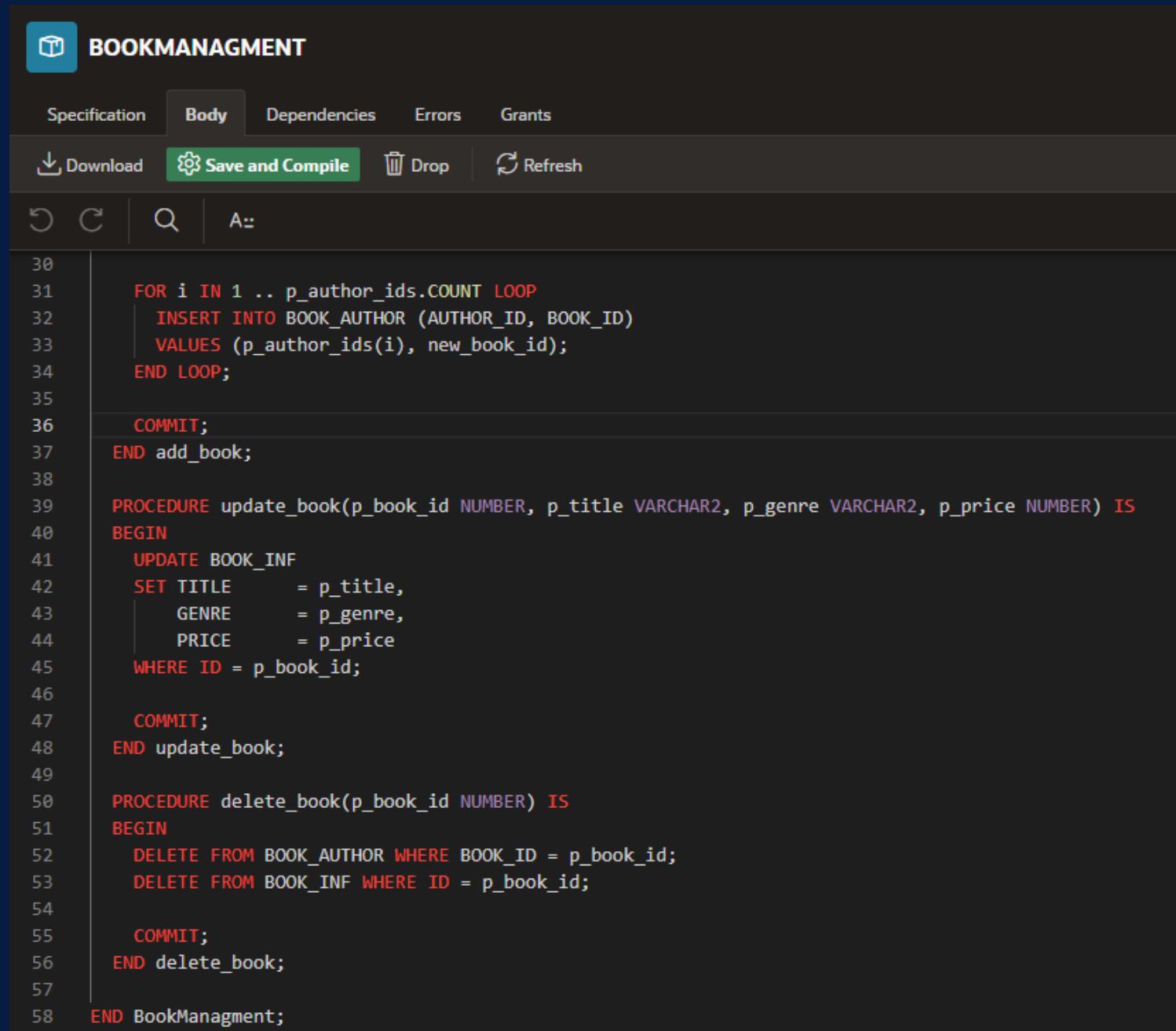
This function combines two functions, the first (to get a list of all books by a certain author), the second (to get a list of all orders for a certain client) and three procedures add\_book, update\_book, delete\_book



```

1  create or replace PACKAGE BODY BookManagement AS
2
3      FUNCTION get_books_by_author(p_author_id NUMBER) RETURN SYS_REFCURSOR IS
4          book_cursor SYS_REFCURSOR;
5      BEGIN
6          OPEN book_cursor FOR
7              SELECT b.*
8                  FROM BOOK_INF b
9                  JOIN BOOK_AUTHOR ba ON b.ID = ba.BOOK_ID
10                 WHERE ba.AUTHOR_ID = p_author_id;
11         RETURN book_cursor;
12     END get_books_by_author;
13
14     FUNCTION get_orders_by_customer(p_customer_id NUMBER) RETURN SYS_REFCURSOR IS
15         order_cursor SYS_REFCURSOR;
16     BEGIN
17         OPEN order_cursor FOR
18             SELECT o.*
19                 FROM ORDERS o
20                 WHERE o.CUSTOMER_ID = p_customer_id;
21         RETURN order_cursor;
22     END get_orders_by_customer;
23
24     PROCEDURE add_book(p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER, p_author_ids SYS.ODCINUMBERLIST) IS
25         new_book_id NUMBER;
26     BEGIN
27         INSERT INTO BOOK_INF (TITLE, GENRE, PRICE)
28             VALUES (p_title, p_genre, p_price)
29             RETURNING ID INTO new_book_id;
30     END add_book;
31
32     PROCEDURE update_book(p_book_id NUMBER, p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER) IS
33     BEGIN
34         UPDATE BOOK_INF
35             SET TITLE      = p_title,
36                 GENRE      = p_genre,
37                 PRICE      = p_price
38             WHERE ID = p_book_id;
39     END update_book;
40
41     PROCEDURE delete_book(p_book_id NUMBER) IS
42     BEGIN
43         DELETE FROM BOOK_AUTHOR WHERE BOOK_ID = p_book_id;
44         DELETE FROM BOOK_INF WHERE ID = p_book_id;
45
46         COMMIT;
47     END delete_book;
48
49     END BookManagement;

```



```

1  create or replace PACKAGE BODY BookManagement AS
2
3      FUNCTION get_books_by_author(p_author_id NUMBER) RETURN SYS_REFCURSOR IS
4          book_cursor SYS_REFCURSOR;
5      BEGIN
6          OPEN book_cursor FOR
7              SELECT b.*
8                  FROM BOOK_INF b
9                  JOIN BOOK_AUTHOR ba ON b.ID = ba.BOOK_ID
10                 WHERE ba.AUTHOR_ID = p_author_id;
11         RETURN book_cursor;
12     END get_books_by_author;
13
14     FUNCTION get_orders_by_customer(p_customer_id NUMBER) RETURN SYS_REFCURSOR IS
15         order_cursor SYS_REFCURSOR;
16     BEGIN
17         OPEN order_cursor FOR
18             SELECT o.*
19                 FROM ORDERS o
20                 WHERE o.CUSTOMER_ID = p_customer_id;
21         RETURN order_cursor;
22     END get_orders_by_customer;
23
24     PROCEDURE add_book(p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER, p_author_ids SYS.ODCINUMBERLIST) IS
25         new_book_id NUMBER;
26     BEGIN
27         INSERT INTO BOOK_INF (TITLE, GENRE, PRICE)
28             VALUES (p_title, p_genre, p_price)
29             RETURNING ID INTO new_book_id;
30     END add_book;
31
32     PROCEDURE update_book(p_book_id NUMBER, p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER) IS
33     BEGIN
34         UPDATE BOOK_INF
35             SET TITLE      = p_title,
36                 GENRE      = p_genre,
37                 PRICE      = p_price
38             WHERE ID = p_book_id;
39     END update_book;
40
41     PROCEDURE delete_book(p_book_id NUMBER) IS
42     BEGIN
43         DELETE FROM BOOK_AUTHOR WHERE BOOK_ID = p_book_id;
44         DELETE FROM BOOK_INF WHERE ID = p_book_id;
45
46         COMMIT;
47     END delete_book;
48
49     END BookManagement;

```

# FUNCTIONS



# This function reads every row of the table

---

```
create or replace FUNCTION count_rows(p_table_name IN VARCHAR2) RETURN NUMBER IS
    row_count NUMBER;
    sql_query VARCHAR2(1000);
    BEGIN
        sql_query := 'SELECT COUNT(*) FROM ' || p_table_name;
        EXECUTE IMMEDIATE sql_query INTO row_count;
        RETURN row_count;
    EXCEPTION
        WHEN OTHERS THEN
            RETURN -1;
    END count_rows;
```

# This function will output all the information about the author which book he wrote if we give the author's ID function

```
create or replace FUNCTION get_books_by_author(p_author_id NUMBER)
  RETURN SYS_REFCURSOR
IS
  books_cursor SYS_REFCURSOR;
BEGIN
  OPEN books_cursor FOR
    SELECT b.id, b.title, b.genre, b.price
      FROM Book_inf b
     JOIN book_author ba ON b.id = ba.book_id
    WHERE ba.author_id = p_author_id;

  RETURN books_cursor;
EXCEPTION
  WHEN OTHERS THEN
    RAISE;
END get_books_by_author;
```

The screenshot shows the Oracle SQL Developer interface with the following details:

- Code Area:** Displays the PL/SQL code for the `get_books_by_author` function.
- Execution Area:** Shows the execution of the function with the parameter `6`.
- Results Tab:** Active tab, showing the output of the query:
  - Book ID: 94, Title: 7 Below (Seven Below), Genre: Horror|Thriller, Price: 1046
  - Book ID: 39, Title: Black Stallion, The, Genre: Adventure|Children|Drama, Price: 288
  - Book ID: 93, Title: Don't Drink the Water, Genre: Comedy, Price: 1220
- Timing:** 6.10 seconds

# TRIGGERS



# Trigger #1

Schema

**INSERT\_BASKET**

Code Errors Object Details DDL

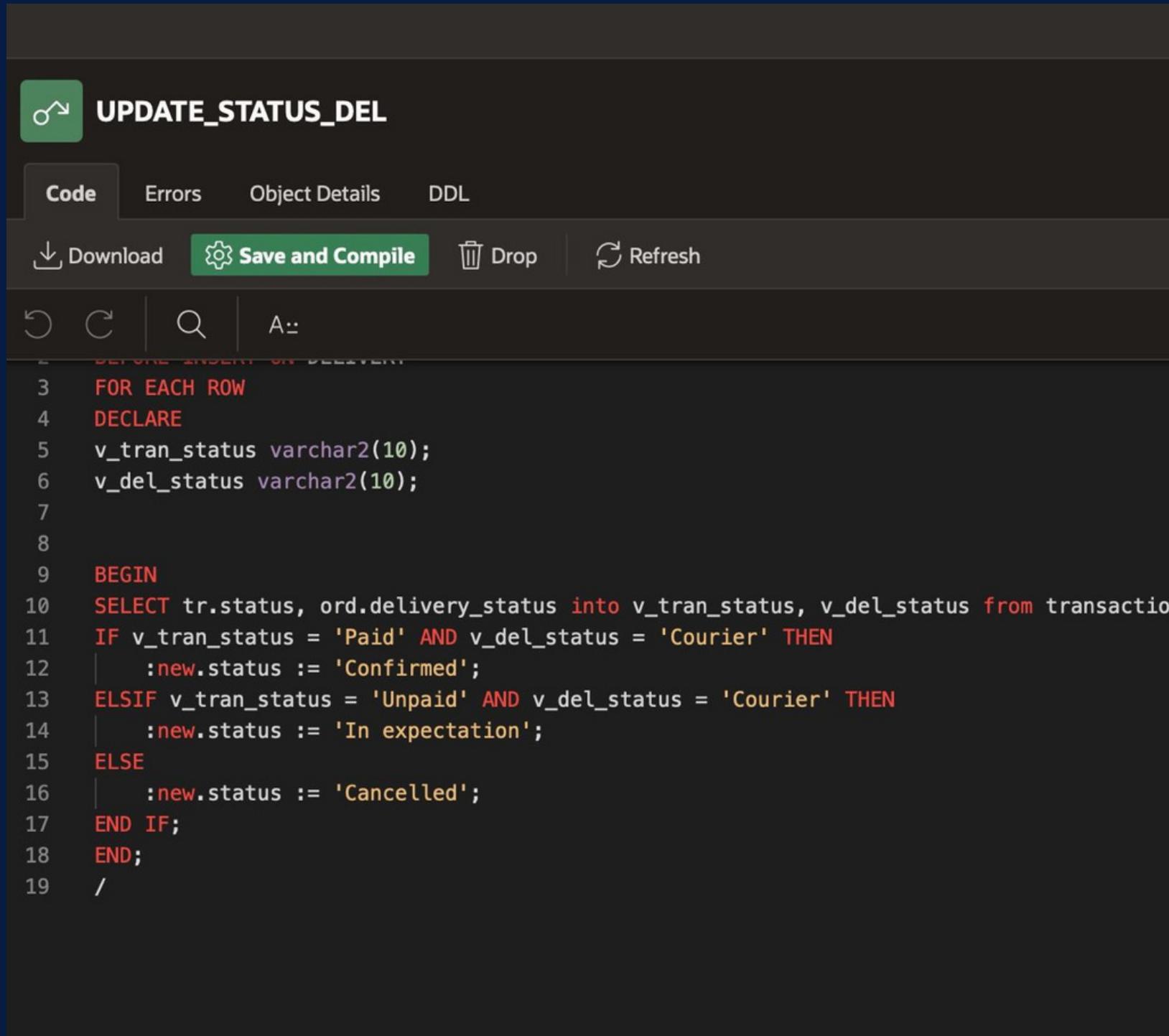
Download Save and Compile Drop Refresh

↻ ↺ Q A::

```
1 create or replace TRIGGER insert_basket
2 BEFORE INSERT ON BASKET
3 FOR EACH ROW
4 DECLARE
5   v_bas_cost NUMBER;
6   v_bas_amount NUMBER;
7 BEGIN
8   SELECT SUM(bi.price), COUNT(bb.customer_id)
9   INTO v_bas_cost, v_bas_amount
10  FROM book_basket bb
11  JOIN book_inf bi ON bb.book_id = bi.id
12  WHERE bb.customer_id = :NEW.customer_id;
13
14  :NEW.cost := v_bas_cost;
15  :NEW.amount := v_bas_amount;
16 END;
17 /
```

**TRIGGER insert\_basket**  
**This trigger is needed for basket, just add the ID and get in cost recalculate all the amounts of the selected books and recalculate how many books the user took**

# Trigger #2



The screenshot shows a database interface for managing triggers. The title bar says "UPDATE\_STATUS\_DEL". Below it, there are tabs for "Code", "Errors", "Object Details", and "DDL". The "Code" tab is selected. A toolbar below the tabs includes "Download", "Save and Compile" (which is highlighted in green), "Drop", and "Refresh". Below the toolbar, there are icons for back, forward, search, and refresh. The main area contains the trigger code:

```
3  FOR EACH ROW
4  DECLARE
5      v_tran_status varchar2(10);
6      v_del_status varchar2(10);
7
8
9      BEGIN
10     SELECT tr.status, ord.delivery_status into v_tran_status, v_del_status from transaction
11     IF v_tran_status = 'Paid' AND v_del_status = 'Courier' THEN
12         :new.status := 'Confirmed';
13     ELSIF v_tran_status = 'Unpaid' AND v_del_status = 'Courier' THEN
14         :new.status := 'In expectation';
15     ELSE
16         :new.status := 'Cancelled';
17     END IF;
18
19 /
```

**TRIGGER update\_status\_del**  
**Checks the delivery table of the courier changes its status that the customer took or is waiting for an order or canceled we check through the table transaction where his status is paid or not paid depends on the trigger and should not be self-expert should be to the address**

# Trigger #3

UPDATE\_STATUS\_TR

Code Errors Object Details DDL

Download Save and Compile Drop Refresh

```
1 create or replace TRIGGER update_status_tr
2 BEFORE INSERT ON TRANSACTIONS
3 FOR EACH ROW
4 DECLARE
5   v_bas_cost NUMBER;
6   v_card_bal NUMBER;
7   v_date date;
8   v_card NUMBER;
9   v_id NUMBER := 0;
10
11 BEGIN
12   SELECT customer_id into v_id from orders where orders.id = :new.order_id;
13
14   SELECT basket.cost INTO v_bas_cost
15   FROM basket
16   WHERE basket.customer_id = v_id;
17
18   SELECT card.pin,card.expiry_date,balance into :NEW.card_pin,v_date, v_card_bal FROM card WHERE balance = (SELECT MAX(balance) FROM card c
19   where c.customer_id = v_id) and card.customer_id = v_id;
20
21   IF v_bas_cost <= v_card_bal AND :NEW.transaction_date < v_date THEN
22     :NEW.status := 'Paid';
23     FOR i IN (select book_id from book_basket where book_basket.customer_id = v_id) LOOP
24       insert into book_basket_for_records values(v_id,:new.order_id,i.book_id);
25     END LOOP;
26     insert into basket_records values(:new.order_id,v_bas_cost);
27
28     delete from basket where basket.customer_id = v_id;
29     delete from book_basket where book_basket.customer_id = v_id;
30
```

**TRIGGER update\_status\_tracking**  
This trigger is needed if the user is to find out what he paid or not if he paid then we remove it from the basket and add it to the records we will check through what he has enough money in the card to buy a book and a card the expiration date has not yet passed

# Trigger #4

BOOK\_NAME\_VALIDATION

Code Errors Object Details DDL

Download Save and Compile Drop Refresh

↻ C Q A:

```
1 create or replace TRIGGER book_name_validation
2 before INSERT ON book_inf
3 REFERENCING NEW AS NEW
4
5 FOR EACH ROW
6 DECLARE
7     invalidprice_ex EXCEPTION;
8     invalidname_ex EXCEPTION;
9 BEGIN
10    if length(:new.title) <= 3 then
11        raise invalidname_ex;
12    elsif :new.price < 0 then
13        raise invalidprice_ex;
14    end if;
15
16    EXCEPTION
17    when invalidname_ex then
18        RAISE_APPLICATION_ERROR(-20001, 'Invalid book title, title must be more than 3');
19    when invalidprice_ex then
20        RAISE_APPLICATION_ERROR(-20002, 'Invalid price, price cannot be negative');
21 END;
```

This trigger check book name and price for validation. If length of book name  $\leq 3$  or price is negative we catch user-defined exception.

# Trigger #5

COUNT\_ROW\_INBOOK\_TABLE

Code Errors Object Details DDL

Download Save and Compile Drop Refresh

↻ C | Q | A:

```
1 create or replace TRIGGER count_row_inBook_table
2 BEFORE INSERT ON Book_inf
3
4 DECLARE
5     row_count NUMBER;
6 BEGIN
7     select count(*) into row_count from book_inf;
8     dbms_output.put_line('Number of rows: ' || row_count);
9 END;
```

This triggers counts  
row numbers before  
inserting new row.

# Trigger #6

COUNT\_ROW\_INCUSTOMER\_TABLE

Code Errors Object Details DDL

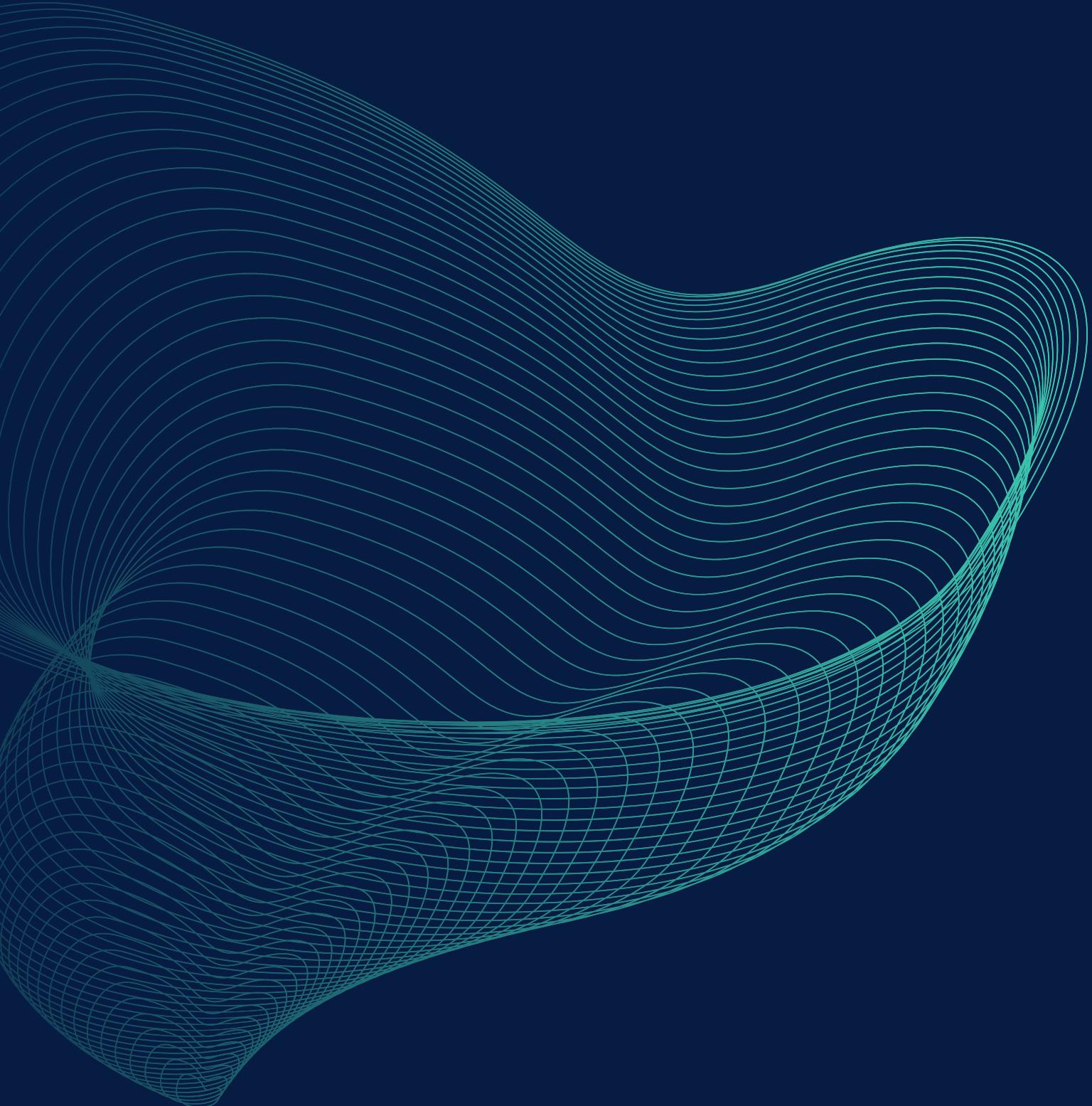
Download Save and Compile Drop Refresh

1 2 3 4 5 6 7 8 9

```
1 create or replace TRIGGER count_row_inCustomer_table
2 BEFORE INSERT ON customer
3
4 DECLARE
5   row_count NUMBER;
6 BEGIN
7   select count(*) into row_count from customer;
8   dbms_output.put_line('Number of rows: ' || row_count);
9 END;
```

This triggers counts  
row numbers before  
inserting new row.

---



**Thank you for  
attention!**