Triggers:

1)

```
create or replace TRIGGER book_name_validation
before INSERT ON book_inf
REFERENCING NEW AS NEW

FOR EACH ROW
DECLARE
   invalidprice_ex EXCEPTION;
   invalidname_ex EXCEPTION;
BEGIN
   if length(:new.title) <= 3 then
      raise invalidname_ex;
   elsif :new.price < 0 then
      raise invalidprice_ex;
   end if;

   EXCEPTION
   when invalidname_ex then
      RAISE_APPLICATION_ERROR(-20001, 'Invalid book title, title must be more than 3');
   when invalidprice_ex then
      RAISE_APPLICATION_ERROR(-20002, 'Invalid price, price cannot be negative');
END;
```

2)

```
create or replace TRIGGER count_row_inBook_table
BEFORE INSERT ON Book_inf

DECLARE
  row_count NUMBER;
BEGIN
   select count(*) into row_count from book_inf;
   dbms_output.put_line('Number of rows: ' || row_count);
END;
```

3)

```
create or replace TRIGGER count_row_inCustomer_table
BEFORE INSERT ON customer

DECLARE
  row_count NUMBER;
BEGIN
   select count(*) into row_count from customer;
   dbms_output.put_line('Number of rows: ' || row_count);
END;
```

4)

```
create or replace TRIGGER insert_basket
BEFORE INSERT ON BASKET
FOR EACH ROW
```

```
DECLARE
  v_bas_cost NUMBER;
  v_bas_amount NUMBER;
BEGIN
  SELECT SUM(bi.price), COUNT(bb.customer_id)
  INTO v_bas_cost, v_bas_amount
  FROM book_basket bb
  JOIN book_inf bi ON bb.book_id = bi.id
  WHERE bb.customer_id = :NEW.customer_id;

  :NEW.cost := v_bas_cost;
  :NEW.amount := v_bas_amount;
END;
```

5)

```
create or replace TRIGGER update_status_del
BEFORE INSERT ON DELIVERY
FOR EACH ROW
DECLARE
v_tran_status varchar2(10);
v_del_status varchar2(10);


BEGIN
SELECT tr.status, ord.delivery_status into v_tran_status, v_del_status from transactions tr  join orders ord on
tr.order_id = ord.id where tr.order_id = :new.order_id;
IF v_tran_status = 'Paid' AND v_del_status = 'Courier' THEN
   :new.status := 'Confirmed';
ELSIF v_tran_status = 'Unpaid' AND v_del_status = 'Courier' THEN
   :new.status := 'In expectation';
ELSE
   :new.status := 'Cancelled';
END IF;
END;
```

6)

```
create or replace TRIGGER update_status_tr
BEFORE INSERT ON TRANSACTIONS
FOR EACH ROW
DECLARE
v_bas_cost NUMBER;
v_card_bal NUMBER;
v_date date;
v_card NUMBER;
v_id NUMBER := 0;

BEGIN
SELECT customer_id into v_id from orders where orders.id = :new.order_id;

SELECT basket.cost INTO  v_bas_cost
FROM basket
WHERE basket.customer_id = v_id;

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
where c.customer_id = v_id) and card.customer_id = v_id;
```

```
IF v_bas_cost <= v_card_bal AND :NEW.transaction_date < v_date THEN
   :NEW.status := 'Paid';
   FOR i IN (select book_id from book_basket where book_basket.customer_id = v_id) LOOP
   insert into book_basket_for_records values(v_id,:new.order_id,i.book_id);
   END LOOP;
   insert into basket_records values(:new.order_id,v_bas_cost);

   delete from basket where basket.customer_id = v_id;
   delete from book_basket where book_basket.customer_id = v_id;

ELSE
   :NEW.status := 'Unpaid';

END IF;

EXCEPTION
   WHEN no_data_found then
   dbms_output.put_line('Please add something to the Basket');
   RAISE_APPLICATION_ERROR(-20001, 'No data found. Transaction cannot be inserted.');
END;
```

Procedures:

1)

```
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;
```

2)
```
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;
```

```
begin
    display_row_counts;
end;
```

3)
```
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:

1)
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;
```

2) Specification:

```
create or replace PACKAGE BookManagment AS

  -- Функция для получения списка всех книг определенного автора
  FUNCTION get_books_by_author(p_author_id NUMBER) RETURN SYS_REFCURSOR;

  -- Функция для получения списка всех заказов для определенного клиента
  FUNCTION get_orders_by_customer(p_customer_id NUMBER) RETURN SYS_REFCURSOR;

  -- Процедура для добавления новой книги
  PROCEDURE add_book(p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER, p_author_ids
SYS.ODCINUMBERLIST);

  -- Процедура для обновления информации о книге
  PROCEDURE update_book(p_book_id NUMBER, p_title VARCHAR2, p_genre VARCHAR2, p_price
NUMBER);

  -- Процедура для удаления книги
  PROCEDURE delete_book(p_book_id NUMBER);

END BookManagment;
```

BODY:

```
create or replace PACKAGE BODY BookManagment AS

  FUNCTION get_books_by_author(p_author_id NUMBER) RETURN SYS_REFCURSOR IS
```

```
    book_cursor SYS_REFCURSOR;
  BEGIN
    OPEN book_cursor FOR
      SELECT b.*
      FROM BOOK_INF b
      JOIN BOOK_AUTHOR ba ON b.ID = ba.BOOK_ID
      WHERE ba.AUTHOR_ID = p_author_id;
    RETURN book_cursor;
  END get_books_by_author;

  FUNCTION get_orders_by_customer(p_customer_id NUMBER) RETURN SYS_REFCURSOR IS
    order_cursor SYS_REFCURSOR;
  BEGIN
    OPEN order_cursor FOR
      SELECT o.*
      FROM ORDERS o
      WHERE o.CUSTOMER_ID = p_customer_id;
    RETURN order_cursor;
  END get_orders_by_customer;

  PROCEDURE add_book(p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER, p_author_ids
SYS.ODCINUMBERLIST) IS
    new_book_id NUMBER;
  BEGIN
    INSERT INTO BOOK_INF (TITLE, GENRE, PRICE)
    VALUES (p_title, p_genre, p_price)
    RETURNING ID INTO new_book_id;

    FOR i IN 1 .. p_author_ids.COUNT LOOP
      INSERT INTO BOOK_AUTHOR (AUTHOR_ID, BOOK_ID)
      VALUES (p_author_ids(i), new_book_id);
    END LOOP;

    COMMIT;
  END add_book;

  PROCEDURE update_book(p_book_id NUMBER, p_title VARCHAR2, p_genre VARCHAR2, p_price NUMBER)
IS
  BEGIN
    UPDATE BOOK_INF
    SET TITLE      = p_title,
        GENRE      = p_genre,
        PRICE      = p_price
    WHERE ID = p_book_id;

    COMMIT;
  END update_book;

  PROCEDURE delete_book(p_book_id NUMBER) IS
  BEGIN
    DELETE FROM BOOK_AUTHOR WHERE BOOK_ID = p_book_id;
    DELETE FROM BOOK_INF WHERE ID = p_book_id;

    COMMIT;
  END delete_book;

END BookManagment;
/
```

FUNCTIONS:

1)

```
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;
```

2)

```
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;
```