CREATE TABLE WAREHOUSES (

ID NUMBER(\*,0) NOT NULL,

NAME VARCHAR2(50 BYTE) NOT NULL,

QUANTITY NUMBER(\*, 0) NOT NULL,

AMOUNT NUMBER(18,2) NOT NULL,

CONSTRAINT WAREHOUSES\_PK PRIMARY KEY(ID)

);

CREATE SEQUENCE WAREHOUSES\_AUTO\_INCREMENT START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER WAREHOUSES\_ON\_INSERT

BEFORE INSERT ON WAREHOUSES

FOR EACH ROW

BEGIN

:NEW.ID := WAREHOUSES\_AUTO\_INCREMENT.NEXTVAL;

END;

CREATE TABLE SALES (

ID NUMBER(\*,0) NOT NULL,

AMOUNT NUMBER(18,2) NOT NULL,

QUANTITY NUMBER(\*, 0) NOT NULL,

SALE\_DATE TIMESTAMP(3) NOT NULL,

WAREHOUSE\_ID NUMBER(\*, 0) NOT NULL,

CONSTRAINT SALES\_PK PRIMARY KEY(ID),

CONSTRAINT FK\_SALES\_WAREHOUSES FOREIGN KEY (WAREHOUSE\_ID) REFERENCES WAREHOUSES(ID)

);

CREATE SEQUENCE SALES\_AUTO\_INCREMENT START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER SALES\_ON\_INSERT

BEFORE INSERT ON SALES

FOR EACH ROW

BEGIN

:NEW.ID := SALES\_AUTO\_INCREMENT.NEXTVAL;

END;

/

CREATE TABLE EXPENSE\_ITEMS (

ID NUMBER(\*,0) NOT NULL,

NAME VARCHAR2(100 BYTE) NOT NULL,

CONSTRAINT EXPENSE\_ITEMS\_PK PRIMARY KEY(ID)

);

CREATE SEQUENCE EXPENSE\_ITEMS\_AUTO\_INCREMENT START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER EXPENSE\_ITEMS\_ON\_INSERT

BEFORE INSERT ON EXPENSE\_ITEMS

FOR EACH ROW

BEGIN

:NEW.ID := EXPENSE\_ITEMS\_AUTO\_INCREMENT.NEXTVAL;

END;

/

CREATE TABLE CHARGES (

ID NUMBER(\*,0) NOT NULL,

AMOUNT NUMBER(18,2) NOT NULL,

CHARGE\_DATE TIMESTAMP(3) NOT NULL,

EXPENSE\_ITEM\_ID NUMBER(\*, 0) NOT NULL,

CONSTRAINT CHARGES\_PK PRIMARY KEY(ID),

CONSTRAINT FK\_CHARGES\_EXPENSE\_ITEMS FOREIGN KEY (EXPENSE\_ITEM\_ID) REFERENCES EXPENSE\_ITEMS(ID)

);

CREATE SEQUENCE CHARGES\_AUTO\_INCREMENT START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER CHARGES\_ON\_INSERT

BEFORE INSERT ON CHARGES

FOR EACH ROW

BEGIN

:NEW.ID := CHARGES\_AUTO\_INCREMENT.NEXTVAL;

END;

-- ------------------------------------------------------------------------------------------------------------

--1 Вычислить общее количество проданных товаров и сумму продаж за все время работы магазина.

SELECT COUNT(id) AS sold\_goods\_quantity, SUM(amount \* quantity) AS total\_amount FROM sales

--2 Вывести все расходы за последний месяц.

SELECT \* FROM charges WHERE extract(month from charge\_date) = extract(month from sysdate)-1

--3 Вывести все товары, которые в данный момент есть на складе и по которым за последний месяц были продажи.

SELECT DISTINCT warehouses.id,name,warehouses.QUANTITY,warehouses.AMOUNT FROM warehouses

JOIN sales ON warehouses.id = sales.warehouse\_id

WHERE extract(month from sale\_date) = extract(month from sysdate)-1

ORDER BY warehouses.id

--4 Вывести стоимость каждой статьи расхода за последний год, упорядочив по убыванию стоимости.

SELECT expense\_item\_id, expense\_items.name,charges.CHARGE\_DATE, SUM(amount) FROM charges

JOIN expense\_items ON expense\_items.id = charges.expense\_item\_id

WHERE extract(year from charge\_date) = extract(year from sysdate)-1

GROUP BY expense\_item\_id, expense\_items.name,charges.CHARGE\_DATE

ORDER BY SUM(amount) DESC

--5 Вычислить прибыль магазина за последний месяц.

SELECT SUM (sales.amount \* sales.quantity) AS earnings FROM sales

JOIN warehouses ON warehouses.id = sales.warehouse\_id

WHERE extract(month from sale\_date) = extract(month from sysdate)-1

--6 Пять самых доходных товаров за все время работы магазина.

SELECT \* FROM (SELECT warehouses.name, SUM(sales.amount \* sales.quantity) AS earnings FROM sales

JOIN warehouses ON warehouses.id = sales.warehouse\_id

GROUP BY warehouses.name

ORDER BY earnings DESC)

WHERE ROWNUM <= 5

--1. Добавить новую статью расхода.

INSERT INTO EXPENSE\_ITEMS (ID, NAME) VALUES (4, 'expense\_item');

--2. Добавить в таблицу расходов расход по статье из п.1.

INSERT INTO SALES (ID, AMOUNT, QUANTITY, SALE\_DATE, WAREHOUSE\_ID) VALUES (1000, 410.00, 1,

TO\_TIMESTAMP('2019-04-20', 'YYYY-MM-DD'), 2);

INSERT INTO SALES (ID, AMOUNT, QUANTITY, SALE\_DATE, WAREHOUSE\_ID) VALUES (1001, 415.00, 1,

TO\_TIMESTAMP('2019-04-20', 'YYYY-MM-DD'), 8);

INSERT INTO CHARGES (AMOUNT, CHARGE\_DATE, EXPENSE\_ITEM\_ID) VALUES (150, CURRENT\_TIMESTAMP, (SELECT id FROM expense\_items WHERE name = 'expense\_item'));

INSERT INTO CHARGES (AMOUNT, CHARGE\_DATE, EXPENSE\_ITEM\_ID) VALUES (70, CURRENT\_TIMESTAMP, (SELECT id FROM expense\_items WHERE name = 'expense\_item'));

--1. Добавить в рамках транзакции новый товар на склад и добавить в журнал запись о его продаже. Если количество товара на складе меньше, чем пытаются продать, транзакцию откатить.

DECLARE

quantity\_warehouses NUMBER;

quantity\_sales NUMBER;

BEGIN

SAVEPOINT start\_transaction;

INSERT INTO warehouses (name, quantity, amount) VALUES ('object7178', 90, 147.00);

INSERT INTO sales (amount ,quantity, sale\_date, warehouse\_id) VALUES (167.00, 100 , CURRENT\_TIMESTAMP, (SELECT id FROM warehouses WHERE name='object7178'));

SELECT SUM(quantity) INTO quantity\_warehouses FROM warehouses WHERE NAME = 'object7178';

SELECT SUM(quantity) INTO quantity\_sales FROM sales WHERE warehouse\_id = (SELECT id FROM warehouses WHERE name='object7178');

IF (quantity\_warehouses < quantity\_sales) THEN

ROLLBACK TO start\_transaction;

END IF;

COMMIT;

END;

--1. Удалить статью расхода и все расходы по ней.

BEGIN

DELETE FROM charges WHERE expense\_item\_id = (SELECT id FROM expense\_items WHERE name='expense\_item');

DELETE FROM expense\_items WHERE name='expense\_item';

END;

--1. Удалить в рамках транзакции информацию о продаже определенного товара, заданного по наименованию, с наименьшим количеством этого товара.

DECLARE

min\_quantity NUMBER;

good\_id NUMBER;

BEGIN

SELECT id INTO good\_id FROM warehouses WHERE name='object3';

SELECT MIN(quantity) INTO min\_quantity FROM sales

WHERE warehouse\_id = good\_id;

DELETE FROM sales WHERE quantity = min\_quantity AND warehouse\_id = good\_id;

COMMIT;

END;

--2. то же, что и п.1, но транзакцию откатить.

DECLARE

min\_quantity NUMBER;

good\_id NUMBER;

BEGIN

SAVEPOINT start\_transaction;

SELECT id INTO good\_id FROM warehouses WHERE name='object0';

SELECT MIN(quantity) INTO min\_quantity FROM sales

WHERE warehouse\_id = good\_id;

DELETE FROM sales WHERE quantity = min\_quantity AND warehouse\_id = good\_id;

ROLLBACK TO start\_transaction;

END;

--1. Увеличить цену всех товаров на складе на 10%.

UPDATE warehouses SET amount = amount \* 1.1;

--1. В рамках транзакции в таблице продаж увеличить стоимость последней продажи определенного товара на 5.00 единиц стоимости.

DECLARE

min\_sale\_date TIMESTAMP;

good\_id NUMBER;

BEGIN

SELECT id INTO good\_id FROM warehouses WHERE name='object3';

SELECT sale\_date INTO min\_sale\_date FROM (SELECT \* FROM sales

WHERE warehouse\_id = good\_id

ORDER BY sale\_date DESC)

WHERE ROWNUM = 1;

UPDATE sales SET amount = amount + 5

WHERE warehouse\_id = good\_id AND sale\_date = min\_sale\_date;

END;

--2. то же, что и п.1, но транзакцию откатить.

DECLARE

min\_sale\_date TIMESTAMP;

good\_id NUMBER;

BEGIN

SAVEPOINT start\_transaction;

SELECT id INTO good\_id FROM warehouses WHERE name='object3';

SELECT sale\_date INTO min\_sale\_date FROM (SELECT \* FROM sales

WHERE warehouse\_id = good\_id

ORDER BY sale\_date DESC)

WHERE ROWNUM = 1;

UPDATE sales SET amount = amount + 5

WHERE warehouse\_id = good\_id AND sale\_date = min\_sale\_date;

ROLLBACK TO start\_transaction;

END;

--------------------------------------------------------------------------------

--1. Создать представление, отображающее все статьи расходов,

-- по которым за все время сумма превысила некоторую границу.

CREATE OR REPLACE VIEW charges\_overflows\_view1 AS

SELECT expense\_items.id,expense\_items.name,charges.amount FROM expense\_items

JOIN charges ON charges.expense\_item\_id = expense\_items.id

GROUP BY expense\_items.id,expense\_items.name,charges.amount

HAVING (SUM(charges.amount) > 300);

--2. Создать представление, отображающее общее количество расходов

-- за последний месяц в разрезе статей расходов.

CREATE VIEW charges\_last\_month\_view AS

SELECT DISTINCT name, SUM(charges.amount) AS total\_cost FROM expense\_items

JOIN charges ON expense\_items.id = charges.expense\_item\_id

WHERE extract(month from charge\_date) > extract(month from sysdate)-1

GROUP BY name;

--1. Создать хранимую процедуру, выводящую все товары

-- и среднюю стоимость их продаж за все время.

CREATE OR REPLACE PROCEDURE goods\_avg\_prices IS

CURSOR goods\_cursor IS

SELECT name, AVG(sales.amount) AS AVG FROM warehouses

JOIN sales ON sales.warehouse\_id = warehouses.id

GROUP BY name;

good\_name VARCHAR2(50 char);

good\_avg NUMBER;

BEGIN

OPEN goods\_cursor;

LOOP

FETCH goods\_cursor INTO good\_name, good\_avg;

EXIT WHEN goods\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('NAME=' || good\_name || ', AVG=' || good\_avg);

END LOOP;

END goods\_avg\_prices;

-- go

SET SERVEROUTPUT ON

BEGIN

goods\_avg\_prices();

END;

--1. Создать хранимую процедуру, имеющую два параметра «товар1»

-- и «товар2». Она должна возвращать даты продаж, в которых эти два

-- товара продавались одновременно.

CREATE OR REPLACE PROCEDURE simultaneous\_sale(good\_id\_1 IN NUMBER, good\_id\_2 IN NUMBER) IS

CURSOR sales\_cursor IS

SELECT TO\_CHAR(sales.sale\_date , 'YYYY-MM-DD') AS sale\_date

FROM ( SELECT \* FROM sales

WHERE warehouse\_id = good\_id\_1) res

JOIN sales ON TO\_CHAR(sales.sale\_date , 'YYYY-MM-DD') = TO\_CHAR(res.sale\_date, 'YYYY-MM-DD')

WHERE sales.warehouse\_id = good\_id\_2

GROUP BY sales.sale\_date;

sale\_date VARCHAR2(50);

BEGIN

OPEN sales\_cursor;

LOOP

FETCH sales\_cursor INTO sale\_date;

EXIT WHEN sales\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('DATE=' || sale\_date);

END LOOP;

END simultaneous\_sale;

-- go

SET SERVEROUTPUT ON

BEGIN

simultaneous\_sale(2, 8);

END;

--1. Создать хранимую процедуру с двумя входными параметрами «дата1»

-- и «дата2» и двумя выходными параметрами, возвращающими общую сумму дохода

-- и расхода за данный период с «дата1» по «дата2».

CREATE OR REPLACE PROCEDURE income\_consumption\_in\_interval (date\_1 IN DATE, date\_2 IN DATE, sales\_sum OUT NUMBER, charges\_sum OUT NUMBER) IS

BEGIN

SELECT SUM(amount \* quantity) INTO sales\_sum FROM sales

WHERE sale\_date BETWEEN date\_1 AND date\_2;

SELECT SUM(amount) INTO charges\_sum FROM charges

WHERE charge\_date BETWEEN date\_1 AND date\_2;

END income\_consumption\_in\_interval;

-- go

SET SERVEROUTPUT ON

DECLARE

sales\_sum NUMBER;

charges\_sum NUMBER;

BEGIN

income\_consumption\_in\_interval(TO\_TIMESTAMP('2018-01-01', 'YYYY-MM-DD'), TO\_TIMESTAMP('2021-01-01', 'YYYY-MM-DD'), sales\_sum, charges\_sum);

DBMS\_OUTPUT.ENABLE;

DBMS\_OUTPUT.PUT\_LINE('SALES\_SUM = ' || sales\_sum || ', CHARGES\_SUM = ' || charges\_sum);

END;

--1. Создать триггер, который не позволяет добавлять расход, с суммой большей заданной.

CREATE OR REPLACE TRIGGER check\_charge

BEFORE INSERT ON charges

FOR EACH ROW

DECLARE

max\_charge\_amount NUMBER := 2700;

BEGIN

IF (:NEW.amount >= max\_charge\_amount) THEN RAISE\_APPLICATION\_ERROR(-20000, 'All charges must be less than 2700');

END IF;

END;

-- check

INSERT INTO CHARGES (AMOUNT, CHARGE\_DATE, EXPENSE\_ITEM\_ID) VALUES (17000, CURRENT\_TIMESTAMP, 1);

--1. Создать триггер, который не позволяет изменять данные в таблице продаж задним числом от сегодняшней даты.

CREATE OR REPLACE TRIGGER restrict\_sale\_dates\_update

BEFORE UPDATE ON sales

FOR EACH ROW

BEGIN

IF (:NEW.sale\_date <= sysdate) THEN RAISE\_APPLICATION\_ERROR(-20000, 'You cannot modify previous sales');

END IF;

END;

-- check

UPDATE sales SET amount = amount \* 1.1;

--1. Создать триггер, который при удалении расхода в случае, если расход был более чем месяц назад, откатывает транзакцию.

CREATE OR REPLACE TRIGGER on\_delete\_charge

BEFORE DELETE ON charges

FOR EACH ROW

BEGIN

IF (:OLD.charge\_date < add\_months(sysdate,-1)) THEN

RAISE\_APPLICATION\_ERROR(-20000, 'You cannot delete charges added more than a month ago');

END IF;

END;

-- check

DELETE FROM charges WHERE id = 4;

--Хранимая процедура для расчета суммы предполагаемой прибыли на ближайший месяц.

CREATE OR REPLACE PROCEDURE count\_incomee (result\_income OUT NUMBER) AS

CURSOR cur\_sales IS

SELECT amount \* quantity, sale\_date FROM sales

WHERE extract(month from sale\_date) > extract(month from sysdate)-3;

CURSOR cur\_charges IS

SELECT amount, charge\_date FROM charges

WHERE extract(month from charge\_date) > extract(month from sysdate)-3;

avg\_charges NUMBER;

current\_charge NUMBER;

current\_charge\_date TIMESTAMP;

avg\_sales NUMBER;

current\_sale NUMBER;

current\_sale\_date TIMESTAMP;

BEGIN

OPEN cur\_sales;

LOOP

FETCH cur\_sales INTO current\_sale, current\_sale\_date;

EXIT WHEN cur\_sales%NOTFOUND;

IF ( extract(month from current\_sale\_date) = extract(month from sysdate) )

THEN avg\_sales := current\_sale \* 1;

END IF;

IF ( extract(month from current\_sale\_date) = extract(month from sysdate)-1)

THEN avg\_sales := current\_sale \* 0.5;

END IF;

IF ( extract(month from current\_sale\_date) >= extract(month from sysdate)-3)

THEN avg\_sales := current\_sale \* 0.25;

END IF;

END LOOP;

OPEN cur\_charges;

LOOP

FETCH cur\_charges INTO current\_charge, current\_charge\_date;

EXIT WHEN cur\_charges%NOTFOUND;

IF ( extract(month from current\_charge\_date) = extract(month from sysdate) )

THEN avg\_charges := current\_charge \* 1;

END IF;

IF ( extract(month from current\_charge\_date) = extract(month from sysdate)-1)

THEN avg\_charges := current\_charge \* 0.5;

END IF;

IF ( extract(month from current\_charge\_date) >= extract(month from sysdate)-3)

THEN avg\_charges := current\_charge \* 0.25;

END IF;

END LOOP;

result\_income := avg\_sales - avg\_charges;

END count\_incomee;

-- go

SET SERVEROUTPUT ON

DECLARE

result\_income NUMBER;

BEGIN

count\_incomee(result\_income);

DBMS\_OUTPUT.ENABLE;

DBMS\_OUTPUT.PUT\_LINE('RESULT\_INCOME = ' || result\_income);

END;