1)   Makhanbetova Gulzhainash 180107159

2)   As a inspiration of our project I chose the “Coffee shop”

The process of using of my shop:

1.    Client opens the shop

2.    Signing up (fill in the information about themselves and add a bank card)

3.    Loging in into their account

4.    Scrolls through the menu

5.    Client can choose the items and put them into the shopping card

6.    After client is able to make order (in process of buying the cost of the order will be withdrawn from the clients’ card)

7.    Client has 2 choices: Picking up themselves / Delivery from the nearest branch (if client chose ‘delivery’ after payment, the courier will start transporting the order)

8.    Also, client can book table in any branch

9.    When ‘delivery’ status is successful, the transaction will also be ended, otherwise the order will be canceled and money will be returned to the client.

3)   User, Orders, Bank card, Transactions, Shopping card, Menu, Reservation, Delivery, Branches.

4)   I have created the tables listed below:

* User
* Shopping card
* Orders
* Menu
* Bank Card
* Transactions
* Branches
* Reseration
* Delivery
* Staff
* Feedstock
* Storage
* Shipments
* Addresses

5)   User chooses products from menu; add them into the shopping card; Make order of items that in the shopping card by using their bank card; Client has 2 choices: Picking up themselves / Delivery from the nearest branch; If it first option, clients goes to chosen branch, else client receive the order from the courier

6)From the Business process side:

1.    Shopping card – Menu = M:M

2.    Orders – Menu = M:M

3.    User – Orders = 1:M

4.    User – Shopping card = 1:M

5.    User – Reservation = 1:1

6.    User – Bank card = 1:M

7.    Bank card – Transaction = 1:M

8.    User – Delivery = 1:M

9.    Delivery - Branches = M:1

Relations left:

1.    Branches – Staff = 1:M

2.    Branches – Storage = 1:1

3.    Storage – Feedstock = 1:M

4.    Storage – Shipments = 1:M

Explanation why the structure follows normal forms:

1NF: The structure follows normal form, because every row contains unique information. Every element of tables (Branch address, Delivery address, User address) in the cell is atomic and cannot be divided, used without regardless one another.

2NF: The table must be in first normal form. Any of its fields that are not part of the primary key are functionally fully dependent on the primary key. So, we avoid half-dependance on the primary key by creating a separate table and store a foreign key for the order table there.

3NF: A relation that fulfills 3NF is a relation in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key. For example, we have attributes A,B and C of a relation such that if A -> B, and B -> C, so C is transitively depends on A via B.

ER Diagram



CREATE OR REPLACE TRIGGER show\_row\_count\_menu

BEFORE INSERT ON "MENU"

FOR EACH ROW

DECLARE

  row\_count NUMBER;

BEGIN

  SELECT COUNT(\*) INTO row\_count FROM "MENU";

  DBMS\_OUTPUT.PUT\_LINE('Current number of rows in the table: ' || row\_count);

END;

!Считает сколько row имел table "MENU" до Insert-a в этот же table

Полезна тем что можно увидеть сколько позиции имеет Menu на настоящий момент не считая последнего insert-a!

CREATE OR REPLACE TRIGGER update\_storage\_quantity

AFTER INSERT ON "SHIPMENTS"

FOR EACH ROW

BEGIN

  UPDATE "STORAGE" s

  SET s."QUANTITY" = s."QUANTITY" + :new.QUANTITY

  WHERE s."BRANCH ID" = :new."BRANCH ID" AND s."FEEDSTOCK ID" = :new."FEEDSTOCK ID";

END;

После поставки в какой либо филиал (то есть после добавления информации в "SHIPMENTS") обновляет количество тех или иных вещей

DECLARE

    m\_name menu.name%type;

    m\_cost menu.cost%type;

CURSOR m\_menu is

    SELECT name, cost FROM menu;

BEGIN

    OPEN m\_menu;

    LOOP

    FETCH m\_menu into m\_name, m\_cost;

        EXIT WHEN m\_menu%notfound;

        dbms\_output.put\_line(m\_name||' '||m\_cost);

    END LOOP;

    CLOSE m\_menu;

END;

cursor показывает весь состав меню вместе с ценой

DECLARE

    staff\_rec staff%rowtype;

    code "STAFF"."STAFF ID"%type := :code;

BEGIN

    SELECT \* into staff\_rec

    FROM staff

    WHERE "STAFF ID" = code;

    dbms\_output.put\_line('Staff ID: '||staff\_rec."STAFF ID");

    dbms\_output.put\_line('Name: '||staff\_rec."FIRST NAME");

    dbms\_output.put\_line('Surname: '||staff\_rec."LAST NAME");

    dbms\_output.put\_line('Phone number: '||staff\_rec."PHONE NUMBER");

    dbms\_output.put\_line('Email: '||staff\_rec."EMAIL");

    dbms\_output.put\_line('Date of birth: '||staff\_rec."DATE OF BIRTH");

    dbms\_output.put\_line('Date of joining: '||staff\_rec."DATE OF JOINING");

    dbms\_output.put\_line('Salary: '||staff\_rec."SALARY");

END;

table-based records выдает информацию о работнике чью ID мы ввели

DECLARE

    b\_login "BANK CARD"."LOGIN"%type:= :b\_login;

    counter integer := 0;

    CURSOR bankcard\_cur IS

        SELECT bank, type, "CARD NUMBER", cvv

        FROM "BANK CARD"

        WHERE login = b\_login;

    bankcard\_rec bankcard\_cur%rowtype;

BEGIN

    OPEN bankcard\_cur;

    LOOP

        FETCH bankcard\_cur INTO bankcard\_rec;

        EXIT WHEN bankcard\_cur%notfound;

        counter := counter + 1;

        dbms\_output.put\_line(counter||' '||bankcard\_rec.bank||''||bankcard\_rec.type||''||bankcard\_rec."CARD NUMBER"||''||bankcard\_rec.cvv);

    END LOOP;

END;

cursor-based records по логину пользователя выдает все банковские карточки которые он имеет

DECLARE

    b\_address branches.address%type;

    b\_phone branches."PHONE NUMBER"%type;

    counter integer := 0;

CURSOR b\_branches is

    SELECT address, "PHONE NUMBER" FROM branches;

BEGIN

    OPEN b\_branches;

    LOOP

    FETCH b\_branches into b\_address, b\_phone;

        EXIT WHEN b\_branches%notfound;

        counter := counter + 1;

        dbms\_output.put\_line(counter||' '||b\_address||' '||b\_phone);

    END LOOP;

    CLOSE b\_branches;

END;

cursor выводит все филлиалы заведения, а также их персональный номер

DECLARE

    f\_id feedstock."FEEDSTOCK ID"%type;

    f\_name feedstock.name%type;

    CURSOR f\_feedstock IS

        SELECT "FEEDSTOCK ID", name FROM feedstock;

    TYPE f\_list IS TABLE of feedstock.name%type INDEX BY binary\_integer;

    name\_list f\_list;

    counter integer := 0;

BEGIN

    FOR n IN f\_feedstock LOOP

        counter := counter + 1;

        name\_list(counter) := n.name;

        dbms\_output.put\_line(counter||': '||name\_list(counter));

    END LOOP;

END;

collections + cursor все сырье имеющееся на складе

DECLARE

    reservation\_rec reservation%rowtype;

    r\_login reservation.login%type := :r\_login;

BEGIN

    SELECT \* INTO reservation\_rec

    FROM reservation

    WHERE login = r\_login;

    dbms\_output.put\_line('Login: '||reservation\_rec.login);

    dbms\_output.put\_line('Table: '||reservation\_rec."TABLE NUM");

    dbms\_output.put\_line('Time/Date: '||reservation\_rec."RES DATE TIME");

    dbms\_output.put\_line('Order ID: '||reservation\_rec."ORDER ID");

END;

table-based records узнать информацию о резервированном столике через логин пользователя

CREATE OR REPLACE TRIGGER PRODUCT\_NAME\_LENGTH\_CHECK

BEFORE INSERT OR UPDATE ON MENU

FOR EACH ROW

DECLARE

 MIN\_PRODUCT\_NAME\_LENGTH EXCEPTION;

 PRAGMA EXCEPTION\_INIT(MIN\_PRODUCT\_NAME\_LENGTH, -20009);

BEGIN

 IF LENGTH(:NEW.name) < 5 THEN

  RAISE MIN\_PRODUCT\_NAME\_LENGTH;

   END IF;

   EXCEPTION

   WHEN MIN\_PRODUCT\_NAME\_LENGTH THEN

    DBMS\_OUTPUT.PUT\_LINE('Error: Product name must be at least 5 characters long.');

END;

свой эксепшн и триггер при попытке ввести в меню название  товара меньше чем 5 символами в названии

CREATE OR REPLACE PROCEDURE get\_cart\_total\_prices\_by\_user AS

    CURSOR c\_cart\_totals IS

        SELECT login, SUM("TOTAL PRICE") AS total\_sum

        FROM "SHOPPING CART"

        GROUP BY login;

    v\_login "SHOPPING CART".login%TYPE;

    v\_total\_sum NUMBER;

BEGIN

    OPEN c\_cart\_totals;

    LOOP

        FETCH c\_cart\_totals INTO v\_login, v\_total\_sum;

        EXIT WHEN c\_cart\_totals%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('User ' || v\_login || ' has a total cart price of ' || v\_total\_sum);

    END LOOP;

    CLOSE c\_cart\_totals;

END;

процедура выводит юзеров  и общую сумму их корзины

CREATE OR REPLACE TRIGGER cart\_total\_price\_autocount

BEFORE INSERT OR UPDATE OF cost, quantity ON "SHOPPING CART"

FOR EACH ROW

BEGIN

 :NEW."TOTAL PRICE" := :NEW.cost \* :NEW.quantity;

END;

автоматически считает общую сумму за n товаров

CREATE OR REPLACE TRIGGER count\_rows\_before\_insert

BEFORE INSERT ON "SHOPPING CART"

DECLARE

 counter NUMBER;

BEGIN

 SELECT COUNT(\*) INTO counter FROM "SHOPPING CART";

 DBMS\_OUTPUT.PUT\_LINE('Number of products in shopping cart before adding tis one is: ' || counter );

END;

выводит количество продуктов в корзину до добавление нового

CREATE OR REPLACE PROCEDURE group\_data1 IS

  CURSOR c\_data IS

    SELECT "NAME", "LOGIN", SUM ("TYPE") as sum\_TYPE

    FROM "USER"

    GROUP BY "TYPE";

BEGIN

  FOR r\_data IN c\_data LOOP

    DBMS\_OUTPUT.PUT\_LINE(r\_data.sum\_TYPE || ': ' || r\_data.sum\_TYPE);

  END LOOP;

END;

процедура называется group\_data1, и она извлекает данные из таблицы с именем "USER". Оператор SELECT использует GROUP BY для группировки на основе "TYPE" и вычисляет amount стоблцов.

CREATE OR REPLACE PROCEDURE group\_data2 IS

  CURSOR c\_data IS

    SELECT "BRANCH ID", "FEEDSTOCK ID", SUM("QUANTITY") as sum\_QUANTITY

    FROM "STORAGE"

    GROUP BY "BRANCH ID", "FEEDSTOCK ID";

BEGIN

  FOR r\_data IN c\_data LOOP

    DBMS\_OUTPUT.PUT\_LINE(r\_data."BRANCH ID" || ', ' ||  r\_data."FEEDSTOCK ID" || ': ' || r\_data.sum\_QUANTITY);

  END LOOP;

END;

процедура называется group\_data2, и она извлекает данные из таблицы с именем "STORAGE". Оператор SELECT использует GROUP BY для группировки на основе BRANCH "ID", "FEEDSTOCK ID" и вычисляет amount стоблцов.

CREATE OR REPLACE PROCEDURE group\_data3 IS

  CURSOR c\_data IS

    SELECT MAX("STAFF ID") AS "STAFF ID", "BRANCH ID"

    FROM "STAFF"

    GROUP BY "BRANCH ID";

BEGIN

  FOR r\_data IN c\_data LOOP

    DBMS\_OUTPUT.PUT\_LINE(r\_data."BRANCH ID");

  END LOOP;

END;

 процедура называется group\_data3, и она извлекает данные из таблицы с именем "STAFF". Оператор SELECT использует GROUP BY для группировки на основе "BRANCH ID" и вычисляет максимальное занчение и группирирует стоблцы.

DECLARE

   record\_count INTEGER;

BEGIN

   SELECT COUNT(\*) INTO record\_count FROM "ORDERS";

   DBMS\_OUTPUT.PUT\_LINE('The number of records is: ' || record\_count);

END;

функция считает количество records

CREATE OR REPLACE PROCEDURE delete\_succeed\_transactions IS

  deleted\_count NUMBER;

BEGIN

  DELETE FROM "TRANSACTIONS"

  WHERE status = 'SUCCESS';

  deleted\_count := SQL%ROWCOUNT;

  DBMS\_OUTPUT.PUT\_LINE('Number of successful transactions deleted: ' || deleted\_count);

END;

процедура обновляет статус заказа при успешной доставке на адрес и удаляет заказ, а SQL%ROWCOUNT высчитывает количество измененных столбцов