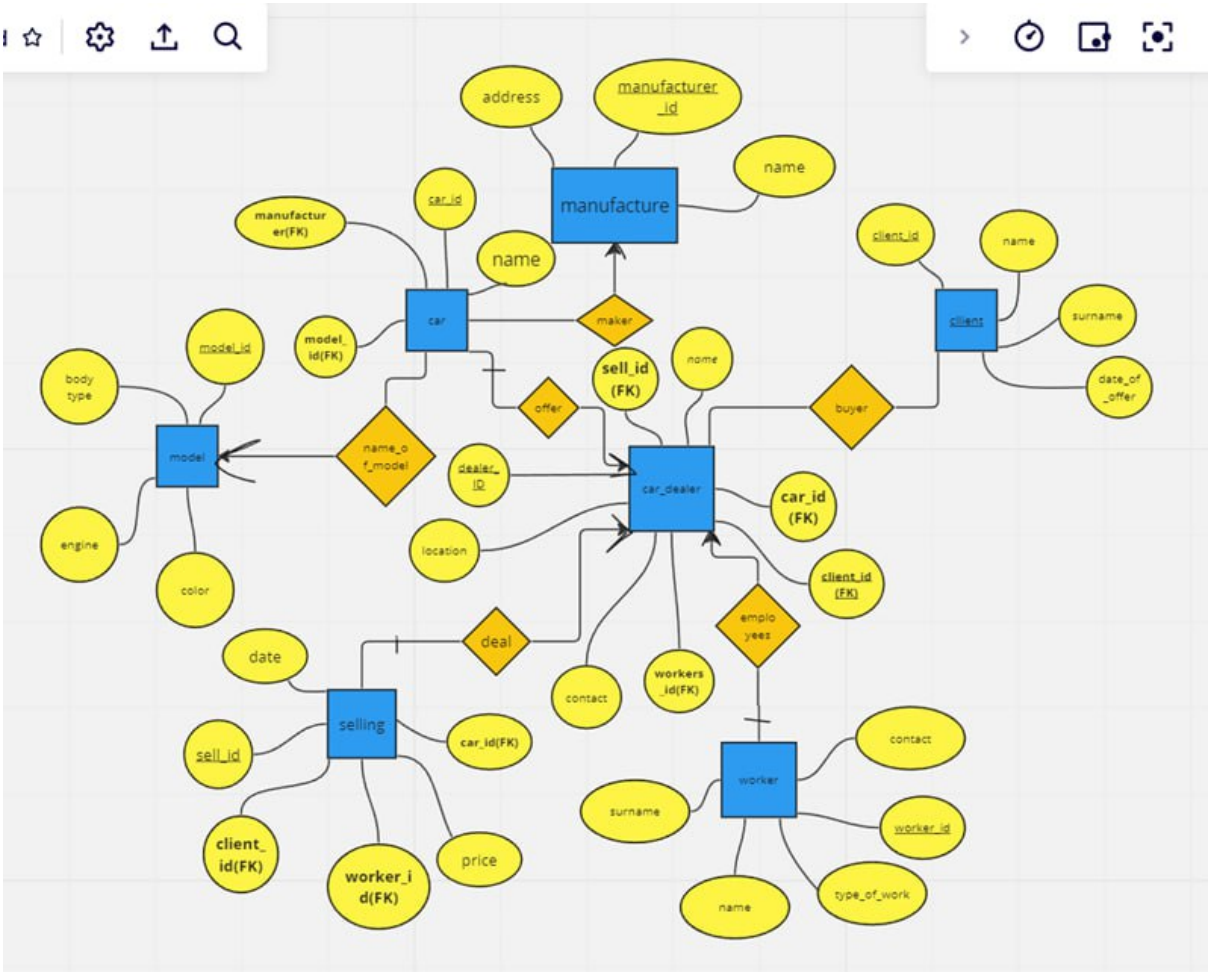
Database management Systems INF 305
Report on team project

Members of team: Aidaruly Rakhman,
Orynbek Nurkeldi,
Dias Satemirov

Our system is a simple explanation of a car selling process. The main focus stays on the role of a car dealer, especially his connection to a client, employees, deals, and cars that he is selling.

To be clear, here is an ER diagram, where we have 7 entities.



We have car_dealer, car, manufacture, model, selling, worker, and client entities. Each of them has its unique properties, which we call attributes (in oval). For example, car_dealer has 8 attributes,dealer_id, location, contact, workers_id, client_id, car_id, name, and sell_ id where 4 of them are foreign keys - sell_id, car_id, client_id, and workers_id (foreign key is a field or a set of fields in a table that refers to the primary key of another table. It establishes a relationship between two tables and helps maintain referential integrity between them) and one primary key - dealer_id (primary key is a unique identifier for a particular record in a

table. It is used to identify and retrieve a specific row of data in the table. A primary key must be unique and cannot contain null values).

There are several entities on the diagram: Car Dealer, Client, Car, Worker, Car Model, Selling, and Manufacture. Each of these entities has its own attributes.

The Car Dealer entity is the main and common entity with the Worker (The company has only a few workers and each of them works for certain salesmen), the Client (the Company has several customers and each customer can contact substances), “Car” (Every car has the same seller), “Sale” (Multiple sales). Has the following attributes: "car_id(fk)", "client_id(fk)", "sell_id(fk)", "name", "contacts" and "workers_id(fk)".

There is also the "Car" entity(attributes: model(fk), name, car_id(pk), manufacturer(fk)), which affects the "Model" entities(attributes: model_id(pk), body type, engine, color), because each machine has several models and "Manufacture"(attributes: address, manufacture_id(pk), name), because. each car was made by the same company.

1NF:
All lines are different. All elements inside cells are atomic (indivisible).

2NF:
All tables are in first normal form. Any of its fields that are not part of the primary key are functionally fully dependent on the primary key.

3NF:
The tables are in second normal form. Any of its non-key attributes are functionally dependent only on the primary key.

BCNF:
The tables are in third normal form. The table has only one primary key.

information about - Procedure which does group by information

```
create or replace procedure info   // Here we give a name to the functions and begin procedure with 'IS'.
is
begin
  for row in(              //Here we select column dealer_id from car_dealer and count number of selles using
    select dealer_id, count(*) as selles      // as counter. It will count by rows for each dealer.
    from car_dealer
    group by dealer_id
    order by dealer_id
  )
```

```
  loop                          // there we output the data which.
    DBMS_OUTPUT.PUT_LINE('dealer '  row.dealer_id  ' sold '  row.selles  ' cars');
  end loop;
end;
/

begin
info;
end;      // it's the command that will show the result

create or replace FUNCTION function_name1
Return number is counter number;
Begin
Select count (*) into counter from worker;
return counter;
End;
```

in the first row we create a function and give name to it
in the second row we will choose data type which will return our function and create variable with this data type
in the third row we count the rows of table worker into the variable counter and return counter

```
declare var number;
begin
    var := function_name1;
    dbms_output.put_line('count of workers '  var);
end;
```

here we declare variable whichs datatype is a number because our function will work only if we declare data type number. Then we assign function name to our variable and outputting it .

```
create or replace PROCEDURE update_prices(new_price IN NUMBER, machine_list IN
VARCHAR2, rows_updated OUT NUMBER) IS
BEGIN
    UPDATE selling
    SET price = new_price
    WHERE car_id IN (SELECT car_id FROM car WHERE name IN (machine_list));

    rows_updated := SQL%ROWCOUNT;
END;
/
```

Firstly we giving the name to our procedure, then give the parameters where new_price and car_list is taken from the table, rows_updated is used to count the rows . After this we are starting a procedure . It will work when we update selling table and putting new price. We put

new price to the line where we declare car_name. Then we assign sql%row_count to row_updated.

```
DECLARE
   num_rows_updated NUMBER;
BEGIN
   update_prices(100000, 'BMW', num_rows_updated);
   DBMS_OUTPUT.PUT_LINE('Number of rows updated: '  num_rows_updated);
END;
```

This query we use for activate the procedure.
We declare numbers_row_updated which data type is number and begin with give to the procedure parameters such as car_price, car_name, numbers_row_updated and output numbers_row_updated.

```
DECLARE
  v_price NUMBER := 12000;
  price_too_low EXCEPTION;
BEGIN
  IF  v_price < 10000 THEN
    RAISE price_too_low;
  ELSE
    INSERT INTO selling (sell_id, price, date1, car_id, worker_id, client_id)
    VALUES (1009, v_price, '1684651489', 901, 1, 482);
  END IF;
EXCEPTION
  WHEN price_too_low THEN
    DBMS_OUTPUT.PUT_LINE('Machine price must be at least 10000');
END;
```

We declare variable which we call v_price and giving a name to exception, then we give condition with this statement if our v_price is lower than 10000 we will call exception where our exception will output some message. If our v_price is more than 10000 we will insert some data into the table selling.

```
create or replace TRIGGER Model_trig
BEFORE INSERT ON MODEL1
FOR EACH ROW
DECLARE
ccount number;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 SELECT COUNT(*) into ccount
 FROM MODEL1;
 commit;
 DBMS_OUTPUT.PUT_LINE('Number of rows in MODEL table before inserting : ' || ccount);
END;
```

/

We create trigger name it and write a condition that will awake trigger in table modul1, then we create variable ccount with data type number, also we did not forget to add "PRAGMA AUTONOMOUS_TRANSACTION;", we are using it to avoid errors because we aare working with only one table. Following this we count and gives this output to ccount and write "commit", then output count of rows before inserting.

insert into model1 values(194,'sedan', '4.0 6-cilinder', 'blue')
this query created for checking trigger.