Online Shop “Mura treats”.

Credits for:

Yessetova Ulday

Ashirova Rabina

Uzbayeva Aruzhan

Bekmuratov Kairat

Andybayev Mukhammat

Introduction.

Welcome to the zoo shop “Mura treats” database project! The target of this project is to simplify accounting and analytical processes for the manager or the head of the store. It will help with tracking inventory, high-low demanded product accounting, customer personal data, such as address, preferred items, phone number, transactions, etc. Eventually, users (product managers, store owners etc.) will get only quite useful advantages that will optimize work of the business and help to enhance customer service.

Let’s talk about figures! We have **8 entity sets** for providing the quality of our database, as Customer, Order, Payment, Address, Item, Supplier, Basket, Courier. We implemented **n number** of *triggers*, **n number** of *exceptions* and **n number** of *procedures*.

Customer - data about our customers to track the preferences of our customers in order to update customer service, and also we have the phone number of our clients in case of emergency or news-announcing.

Address - data about location of our customers, sole purpose is for the delivery process.

Order - data about all orders that were made recently, and proper data for informing couriers about delivery data.

Basket - data about goods that were picked by customers, however have not been paid yet.

Item - data about all available and sold items.

Supplier - data about suppliers who made the items, in order to call or etc later.

Payment- data about transactions.

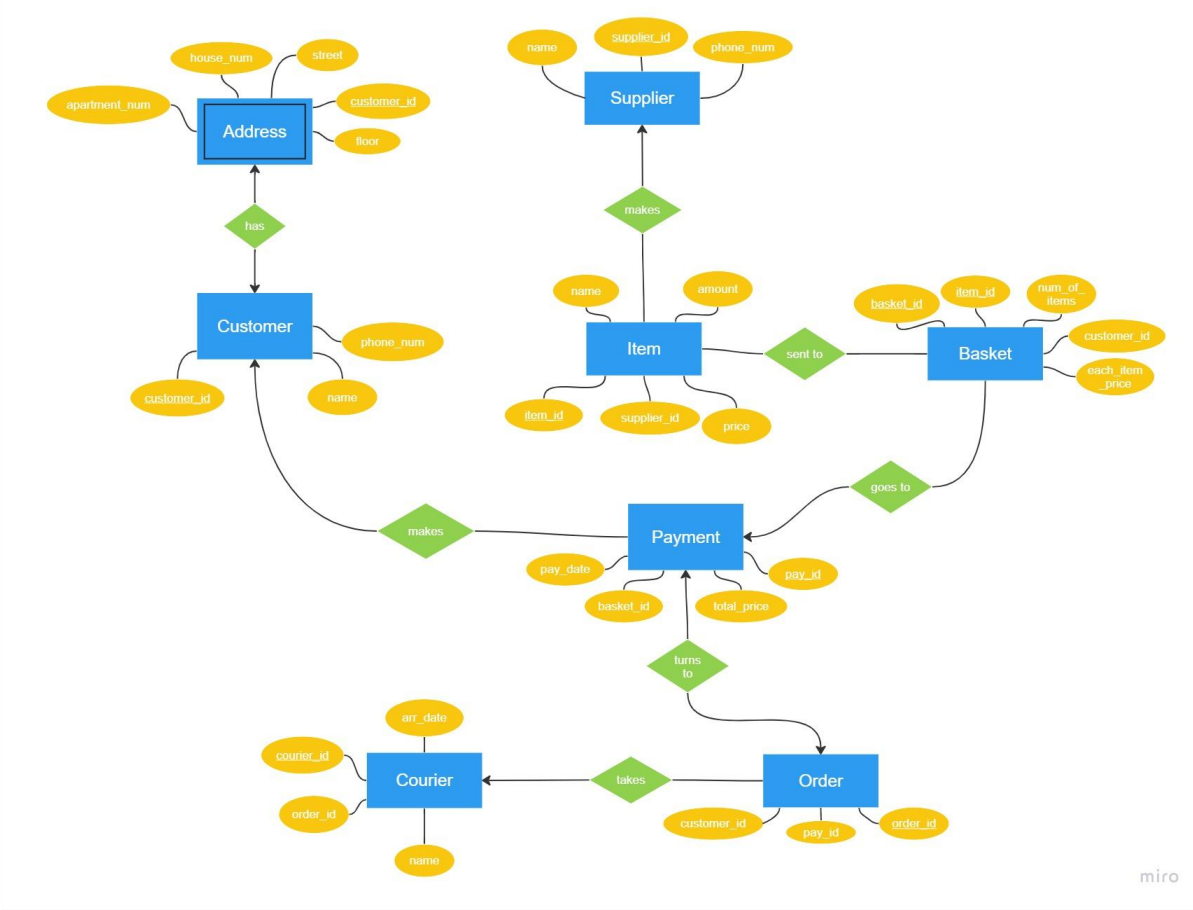
Courier - data about all of our couriers and delivery processes.

For better quality, the database is entirely normalized in **3NF,** that's why users are not likely to face anomalies and mistakes during the work.

In addition, we maximized the database for **life-like-experience**, in order to simplify and optimize the work for users.

Overall, we made a flawless database intended for owners and managers of stores, in order to account and track business processes efficiently. You can check our work by using **GITHub** or **Youtube**, since we uploaded our project there as well for commercial purposes. All additional and practical information, such as coding and links, you can see below. Thanks for your attention!

ER Diagram.



[Link to the board.]

Normalization form.

1NF requires that:

- The table must have a primary key defined.
- Each column in the table must contain only one value per row.
- There should be no repeating groups of columns in the table.

2NF - requires that:

- The table must have a primary key defined.
- Each non-key attribute (i.e., column) in the table must be dependent on the entire primary key, not just part of it.

3NF - requires that:

- Each column in a table should only contain atomic (indivisible) values.
- There should be no repeating groups of columns in the table.

- Each non-key column in a table should be dependent only on the primary key of the table, and not on any other non-key column.

Every criteria is considered in our database system.

Functional Dependency.

---

order_id > courier_id

courier_id > arr_date, name

order_id > courier_id, arr_date, name

---

supplier_id > item_id, name(Supplier), phone_num(Supplier)

item_id > name(Item), amount, price

supplier_id > item_id, name(Supplier), phone_num(Supplier), name(Item), amount, price

---

basket_id > num_of_items, each_item_price, pay_id

pay_id > pay_date, total_price

basket_id > num_of_items, each_item_price, pay_id, pay_date, total_price

---

customer_id > phone_num(Customer), name(Customer), apartment_num, house_num, street, floor, *order_id*, *basket_id*.

customer_id > phone_num(Customer), name(Customer), apartment_num, house_num, street, floor, *order_id*, *basket_id*, courier_id, arr_date, name, num_of_items, each_item_price, pay_id, pay_date, total_price.

Relationships.

Weak entity - Address.
Referential integrity - Customer <> Address, Item <> Supplier.
Many-to-one - Order <> Courier, Basket <> Payment, Payment <> Customer, Item <> Supplier.
Many-to-many - Item <> Basket.
One-to-one - Customer <> Order, Payment <> Order.

Queries.

```
-- 1)
CREATE OR REPLACE PROCEDURE basket_total_price AS
BEGIN
  FOR rec IN (
    SELECT basket_id, SUM(each_item_price) AS total_price
    FROM basket
    GROUP BY basket_id
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('Basket ID: ' || rec.basket_id || ',
Total Price: ' || rec.total_price);
  END LOOP;
END;

begin
basket_total_price;
end;
```

```
-- 2)
create or replace function count_records(tablee IN VARCHAR2)
return number
IS
  record_count number;
begin
  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || tablee INTO
record_count;
  return record_count;
end;
```

```
DECLARE
  item_count number;
begin
  item_count := count_records('Item');
  DBMS_OUTPUT.PUT_LINE('Number of records in Item: ' ||
item_count);
end;

-----------------------------------------
-- 3)
create or replace procedure rowcount(bas_id IN INT)
IS
BEGIN
delete from BASKET where basket_id = bas_id;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows deleted from
basket.');
END;

create or replace trigger callprocedure
AFTER INSERT ON payment
for each row
begin
    rowcount(:new.basket_id);
end;

-----------------------------------------
-- 4)
CREATE OR REPLACE TRIGGER exc
BEFORE INSERT ON item
FOR EACH ROW
DECLARE
    ex Exception;
BEGIN
    IF LENGTH(:new.item_name) < 5 THEN
        RAISE ex;
    END IF;
EXCEPTION
    WHEN ex THEN
        RAISE_APPLICATION_ERROR(-20001, 'Name cannot contain
less than 5 letters');
END exc;

-----------------------------------------
```

```
-- 5)
CREATE OR REPLACE TRIGGER row_count
BEFORE INSERT ON OORDER
FOR EACH ROW
DECLARE
  row_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO row_count FROM oorder;
  DBMS_OUTPUT.PUT_LINE('Current number of rows in Order table:
' || row_count);
END;
```

Triggers.

```
create or replace trigger order_exist
before insert or update of order_id
on courier
for each row
DECLARE
    oorder_id oorder.order_id%type;
BEGIN
    select order_id into oorder_id from oorder where order_id =
:new.order_id;

EXCEPTION
    when no_data_found then
        dbms_output.put_line('There is no such order!');
END;
```

```
create or replace trigger order_exist
before insert or update of order_id
on courier
for each row
DECLARE
```

```
    oorder_id oorder.order_id%type;
BEGIN
    select order_id into oorder_id from oorder where order_id =
:new.order_id;

EXCEPTION
    when no_data_found then
        dbms_output.put_line('There is no such order!');
        delete from courier where order_id = :new.order_id;
END;
```

```
create or replace trigger basket_exist
before insert
on payment
for each row
DECLARE
    bas_id basket.basket_id%type;
BEGIN
    select basket_id into bas_id from basket where basket_id =
:new.basket_id;

EXCEPTION
    when no_data_found then
        dbms_output.put_line('There is no such basket id!');
        delete from payment where basket_id = :new.basket_id;
END;
```