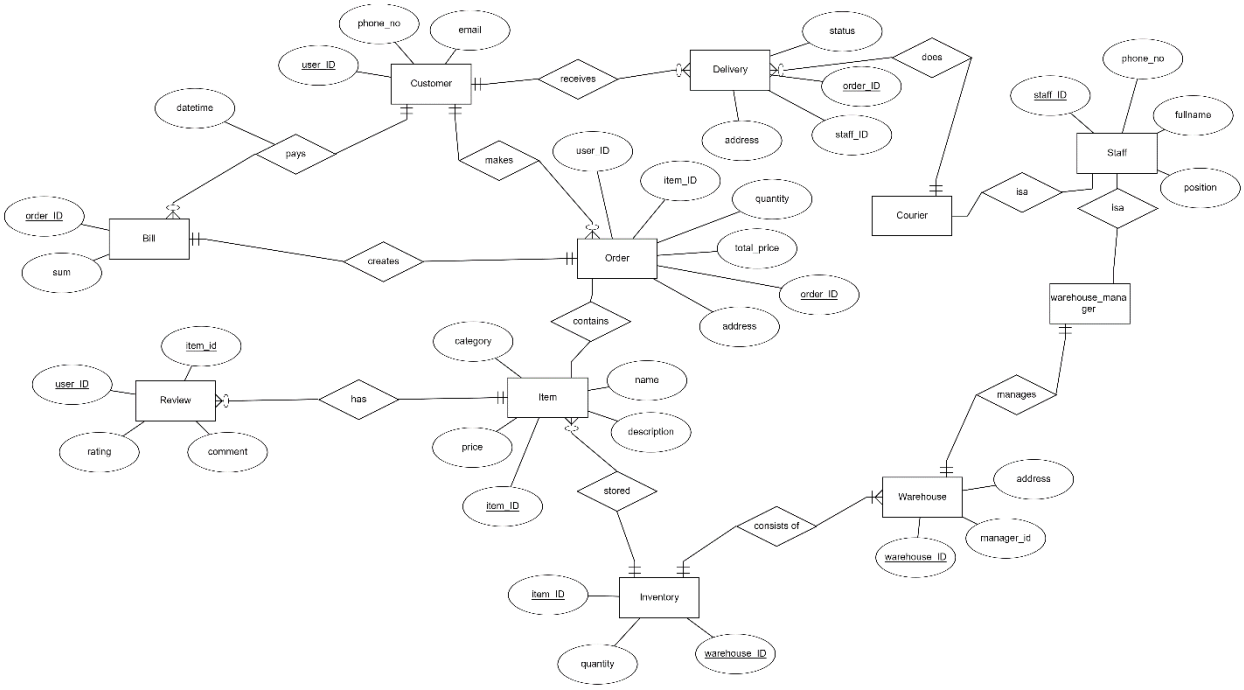
REPORT

Introduction:

This project is a simple design of an e-commerce system, in this example of a furniture store. But the design can also be suitable for selling books, parachutes or even hoods. The system includes tools for inventory, personnel management, contains a basic transaction system. The system is in the third normal form, which means minimal load on the hardware: no redundancy, improved data integrity, and efficient storage and retrieval of data.

ER diagram for the system (also included on the github page):



On the diagram you can see entities, its relationships and attributes.

To elaborate on the normalization of the structure:

As I mentioned above, structure follows 3NF, which means it follows 1NF and 2NF too. But what does it mean?

First Normal Form (1NF) requires that each table cell should contain a single value, and each record must be unique. You can examine the structure and see that that it follows this form. Although staff and customers can only have one phone number, but who cares anyway.

Second Normal Form (2NF) requires that structure is in 1NF already, and has no partial dependencies. That is, all non-key attributes are fully dependent on a primary key. Also check, moving on…

Finally, Third Normal Form (3NF) says that no table should have transitive partial dependency. In other words, a table satisfies 3NF, if and only if for any non-trivial X depends on Y either X is a superkey or each attribute in Y is contained in a key.

Now to coding part:

Procedures and Functions are available on .txt file.

First task was to create a procedure which does group by information. Code:

```
CREATE OR REPLACE PROCEDURE warehouse_capacity
AS
CURSOR cursorr IS SELECT warehouse_id, sum(quantity) total_quant FROM inventory group by warehouse_id order by warehouse_id;
BEGIN
    for roww in cursorr LOOP
        dbms_output.put_line('Warehouse no' || roww.warehouse_id || ': ' || roww.total_quant);
    end loop;
END;
/
```

This procedure returns number of items stored in each warehouse. First, I wrote query that does what I need. And then used cursor to save the query output, after which used loop to iterate through every row displaying id and total quantity.

Function which counts number of records:

```
1   CREATE OR REPLACE FUNCTION cnt_rec(table_name varchar)
2   RETURN NUMBER
3   IS
4       cnt number;
5   begin
6       execute immediate 'SELECT COUNT(*)
7       FROM ' || table_name INTO cnt;
8
9       return cnt;
10  end;
11  /
```

As the name suggests this function counts number of rows given the name of the table. I used ‘execute immediate’ keywords so that query will be dynamic, giving output depending on user input.

Procedure which uses SQL%ROWCOUNT to determine the number of rows affected:

```
CREATE OR REPLACE PROCEDURE upd_costbycat(
    CAT in varchar,
    diff in number
)
is
BEGIN
    UPDATE ITEMS
    SET PRICE = PRICE + DIFF
    WHERE CATEGORYY LIKE CAT;

    DBMS_OUTPUT.PUT_LINE('Number of rows affected: ' || SQL%ROWCOUNT);
END;
/
```

This procedure is used to increment prices of items based on the category. Procedure takes category name, and value of increment. SQL%ROWCOUNT is automatically created, so I didn't need to initialize or give value to it. Easy as that

Add user-defined exception which disallows to enter title of item (e.g. book) to be less than 5 characters:

```
CREATE OR REPLACE PROCEDURE CHNAMEBYID(
    NEWNAME VARCHAR,
    IID NUMBER
)
AS
    too_short exception;
BEGIN
    IF LENGTH(NEWNAME) < 5 THEN RAISE TOO_SHORT;
    ELSE
        UPDATE ITEMS
        SET NAMEE = NEWNAME
        WHERE ITEM_ID = IID;
    END IF;

    COMMIT;

    dbms_output.put_line('Item name changed to ' || NEWNAME );
EXCEPTION
    WHEN TOO_SHORT THEN
        DBMS_OUTPUT.PUT_LINE('New name is just too short');
END;
```

Procedure called CHNAMEBYID changes name of the item given its id and new name. But if new name is too short (less than 5 symbols) it will raise an exception.

Trigger before insert on any entity which will show the current number of rows in the table:

```
CREATE OR REPLACE TRIGGER rowcnt
BEFORE INSERT ON REVIEWS
DECLARE
    ROW_NUM NUMBER;
BEGIN
    SELECT COUNT(*) INTO ROW_NUM FROM REVIEWS;
    DBMS_OUTPUT.PUT_LINE('Number of rows before insert ' || row_num);
END;
/
```

This one is really simple. We create a trigger that executes COUNT(*) and displays it before insertion on given table.