

Project work

Subject: Database Management Systems II

Topic: Online Market. DBMS for Inventory Control of an Online Market

Suleyman Demirel University

Performed by: Abibullayev Yerzhan, Ozatbek Aidynaskar

Description	2
Introduction	2
End users and functionality	2
Data obsolescence	2
Source of an idea	2
Entity Relationship Diagram (ERD)	3
Entities and attributes	3
Relations	5

Introduction

The topic of this project is Database Management System for Inventory Control of an online market. This particular system is aimed to provide a working base for any online shopping site, because all of them require to establish their goods and have proper control over every action, so that when it's necessary it could be used as a sustain starting point. The system itself consists of 10 working entities each responsible for different mechanisms which are present in most of modern marketing systems. As an example for these systems we chose various consumer electronics' retailer store.

System will be controlled and implemented by managers who are responsible for the warehouse and more superior staff who will be making deals and orders for new supplies. The whole process that we are trying to create the image of, is that whenever the new supplies arrive, it should contain QR-code so that after it is scanned it will be uploaded to the database with all the information about the package it contains. For instance, the market ordered a big amount of supplies and there are exact amounts of stuff they are expecting to arrive, so when they arrive they can scan it and check if there are missing packages by comparing it to the expected data.

Entity Relationship Diagram (ERD)

([Source](#))

Entities and attributes

Warehouse - entity of a warehouse that the market may own and use it to store items.

ID - contains identification numbers of the warehouses given to each by default.

State - contains information about the current state of the warehouses and whether it's functioning properly or in maintenance.

Location - contains information about where the warehouses are located.

Max_capacity - contains information about the maximum amount of items a warehouse can store.

Current_items_stored - contains information about how many items are currently stored in a warehouse

Supplier - entity of a supplier with whom the market worked in the past or plan to work with in future.

ID - unique identification number of each supplier.

Name - contains names of suppliers.

Phone_n - contains business phone numbers of suppliers.

Deal_number - contains exact count of deals that were done between market and suppliers. It is required to define the credibility of each supplier.

Maker - entity of the different makers of products.

ID - unique identification number of each maker(brand).

Name - contains names of makers.

Details - contains various details about the maker. It can be their credibility or specific details about the product they manufacture.

User - entity of a user of this system.. Mostly employees with varying degrees of access and work type.

ID - unique identification number of each user.

Role - contains information about the current position of employee.

Username - usernames of the users used to login into the system. Access to this information is restricted to most users.

Password - passwords of the users used to login into the system. Access to this information is restricted to most users.

Firstname, Lastname - contains first and last names of users.

Phone_number - contains phone numbers of users.

Last_login - contains when the user last logged in to the system.

Item - entity of different items(products) stored in the warehouses or planned to order.

ID - unique identification number of each item.

Name - contains names of items.

Desc - short description of each item.

Maker_id - ID referenced from Maker table used to identify the maker of a particular item.

Category_id - ID referenced from Item_category table used to identify the category of item.

Stock_id - ID referenced from Item_stock table used to identify storage information of a particular item.

Item_category - entity of a categorization of items.

ID - identification number of a category.

Name - name of a category.

Desc - short description of a category.

Item_stock - entity of an inventorization of items.

ID - identification number.

Num_stored - number of an item in store.

Warehouse_id - ID of a warehouse where an item is stored. Referenced from Warehouse table.

Order_details - entity of an order process.

ID - identification number of each ordering process.

Supplier_ID - ID of a supplier.

User_ID - ID of a user supervising the order.

Transaction_ID - ID of a transaction linked to the specific order.

Order_items - entity of an order containing items.

ID - ID of a specific order.

Order_ID - ID of the details for an order.

Item_ID - ID of an item contained in this order.

Transaction_details - entity of a transaction linked to the order.

ID - Identification of a transaction.

Total - Total amount of payment of a transaction.

Status - Status of a transaction. Its either closed or active.

Relations

Ordering - connects user and supplier in the process of ordering with Order_details. The process itself is pretty simple, the user, mostly the employee who is responsible for ordering new products for the store, contacts suppliers to arrange a deal. It is necessary to avoid direct relations between user and supplier in this process, because otherwise the data flow would be a mess. Connections between user/supplier and order are many to one. For instance, many users

can make multiple orders from many suppliers, but in one specific order there is only one user and one supplier.

Payment - connects orders and payments attached to it. The connection is one to one. For instance, one payment is attached to only one specific order.

Has - connects orders and items it contains. The connection between them is one to many, because one order can contain only one set of items, but there can be various sets of items for different orders in order_items.

It also connects order items with items and the connection type there is also one to many, because there could be multiple items in one order, but it is made from sets and sets are constructed from individual items.

Contains - connects warehouses with items and this relation simply means that warehouses contain items. Connection type between them is one to many, because there could be multiple items stored in one warehouse, but one specific item is stored only in one warehouse.

Normalization

Functional dependencies

Warehouse

ID→State; ID→Location; ID→State; ID→Max_capacity; ID→Current_items_stored;

Primary key: ID

Supplier

ID→Name, phone_n; ID→deal_number; *Primary key: ID*

Maker

ID→name,details; *Primary key: ID*

User

ID→Role, username, password, firstname, lastname, phone_number, last_login;

Username→ID, Role, password, firstname, lastname, phone_number, last_login;

Candidate keys: ID, Username

Order_items

ID→order_id, item_id; *Primary key: ID*

Order_details

ID→supplier_id, user_id, transaction_id; *Primary key: ID*

Transaction_details

ID→total, status; *Primary key: ID*

Item

ID→name, desc, maker_id, category_id, stock_id; *Primary key: ID;*

Item_category

ID→name, desc; *Primary key: ID;*

Item_stock

ID→num_stored, warehouse_id; *Primary key: ID;*

BCNF check

Instead of 3NF we decided that BCNF would be a much better solution for our system in particular. In order for the table to be in BCNF, first it should be in 1NF, 2NF and 3NF. The tables are in 1NF, because none of them have repeating attribute names and they all contain a single value. The tables are also in 2NF, because there is no partial-dependency. 3NF implies that FDs should not contain transitive dependencies. Secondly, non prime attributes should not define a prime attribute or part of it and as we can see there are none of it as well. So we can conclude that all the tables are in BCNF.

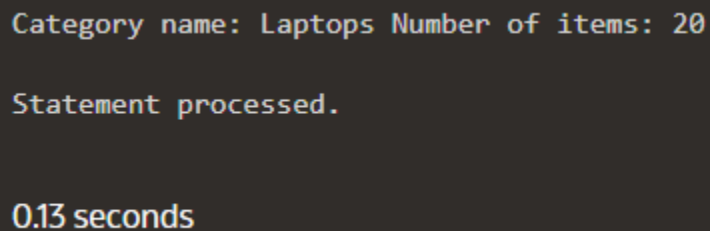
PL/SQL Part**Procedures**

```
[CREATE OR REPLACE PROCEDURE proc_group(c_name VARCHAR2) IS
item_no NUMBER;
BEGIN
SELECT COUNT(item_id)
INTO item_no FROM item WHERE CATEGORY_ID = (SELECT ID FROM item_category WHERE
NAME = c_name);
DBMS_OUTPUT.PUT_LINE('Category name: ' || c_name || ' Number of items: ' || item_no);
END;]
```

This procedure displays the number of items in the item table for each category chosen by a user.

```
[BEGIN  
proc_group('Laptops');  
END;]
```

This line of code calls the procedure with “Laptops” as variable and output is the following:



```
Category name: Laptops Number of items: 20  
  
Statement processed.  
  
0.13 seconds
```

Triggers

```
[create or replace TRIGGER check_name_length  
BEFORE UPDATE OR INSERT ON item  
FOR EACH ROW  
declare  
    NAME_L EXCEPTION;  
    name_length NUMBER;  
BEGIN  
    name_length := LENGTH(:NEW.name);  
    IF name_length < 5 THEN  
        raise NAME_L;  
    END IF;  
    exception  
    when NAME_L then  
        RAISE_APPLICATION_ERROR(-20001, 'Item name should contain at least 5 characters');  
    when others then  
        dbms_output.put_line('Error!');  
END;]
```


This trigger prevents users from inserting or updating item's name. Whenever it's updated or inserted name is shorter than 5 characters the user defined exception is raised and the error message along with the error code is displayed.

If we'll try to insert new row with the following structure:

```
INSERT INTO item(item_id, name, description) VALUES (201, 'AC', 'Sample text');
```

The output will be as follows:

```
ORA-20001: Item name should contain at least 5 characters
ORA-06512: at "WKSP_FREQYBOO.CHECK_NAME_LENGTH", line 11
ORA-04088: error during execution of trigger 'WKSP_FREQYBOO.CHECK_NAME_LENGTH'
ORA-06512: at "SYS.DBMS_SQL", line 1721

1. INSERT INTO item(item_id, name, description) VALUES (201, 'AC', 'Sample text');
```

```
[CREATE OR REPLACE TRIGGER show_row
BEFORE INSERT ON maker
FOR EACH ROW
declare
row_count NUMBER;
BEGIN
    SELECT COUNT(*) into row_count FROM maker;
    DBMS_OUTPUT.PUT_LINE('Current row count is ' || row_count);
END;]
```

This trigger shows the current number of rows that are in the maker's table before the insert was done.

If we will insert new row to the maker's table with this structure:

```
1  INSERT INTO MAKER VALUES (201, 'Realtek', 'Sample text');
```

The outcome will look like this:

```
Current row count is 100
```

```
1 row(s) inserted.
```

```
0.01 seconds
```

Functions

```
[CREATE OR REPLACE FUNCTION count_records(table_name IN VARCHAR2)
RETURN NUMBER
IS
    record_count NUMBER;
BEGIN
    EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || table_name INTO record_count;
    RETURN record_count;
END;]
```

When executed this function returns the number of records in a table chosen by a user.

If we execute this function with the following variables :

```
1  DECLARE
2  |   num number;
3  BEGIN
4  |   num := count_records('maker');
5  |   dbms_output.put_line('Total number of records: ' || num);
6  END; |
```

DBMS for Inventory Control of an Online Market
The output will look like this:

10

```
Total number of records: 101
```

```
Statement processed.
```

```
0.01 seconds
```