CS 4347.502 - Database Systems
Jalal Omer
Revision Date: December 7, 2022

# Database Project

by



| **Name** | **NetID** | **Email** |
|---|---|---|
| Kenan Stredic | kls190005 | kls190005@utdallas.edu |
| Yubin Wang | yxw190012 | yxw190012@utdallas.edu |
| Aryaan Athwaal | axa180175 | axa180175@utdallas.edu |
| Daniel Joung | duj180000 | duj180000@utdallas.edu |

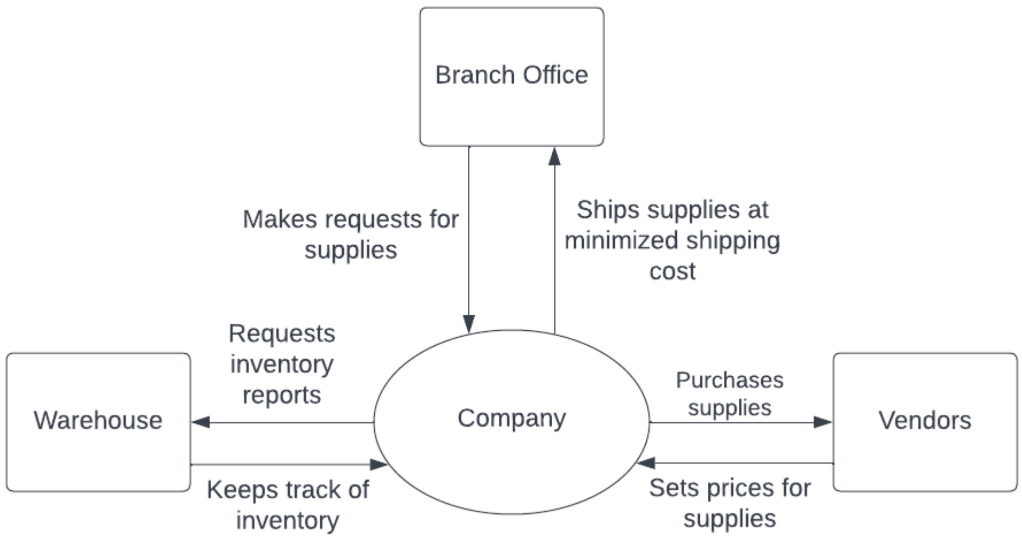
# Contents

# Requirements Analysis

## Introduction / System Description

Our company is creating a database for maintaining supplies in its branch offices. This database will consist of the stock of company supplies, the requests of supplies from the branch offices, and the vendors that stock the supplies. The database containing the stock of the supplies will also contain the product ID of the supplies, and the amount of stock of the item. Requests of the supplies will contain the product ID requested, and the branch office the request is coming from. The database showing vendors that stock the supplies will contain the product ID the item is listed as, the vendor that is selling the item, and the price of the item. Items stored in the database which are listed with a product ID will also be accompanied by a product name, which must be consistent across all product ID entries.

## Context Diagram




## Functional Requirements

| |
|---|
| Branch Office must contain request by product ID as a key |
| Warehouse must contain product by ID as a key |
| Branch Office must contain the branch office sending the product request |
| Vendors must contain the item by product ID |
| Warehouse must contain product ID"s name |
| Vendors must contain the price of the item |
| Vendors will use product ID and vendor name as a key |
| Branch Office must contain product ID"s name for requests |
| Warehouse must contain current stock of product |
| Warehouse must contain the last shipment time |
| Product ID should be consistent with the same product across databases |
| Vendors will contain the shipping time for each product |
| Vendors must contain the vendor name selling the item |

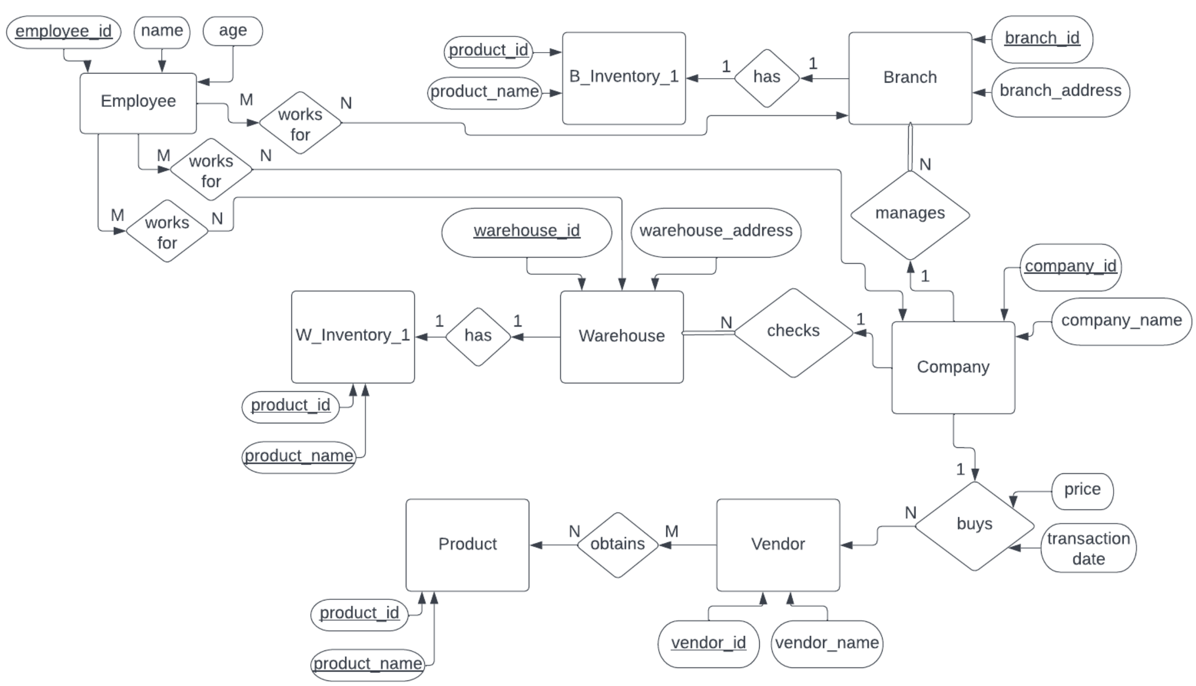
| |
|---|
| Vendors must contain product ID"s name |

## Non-functional Requirements

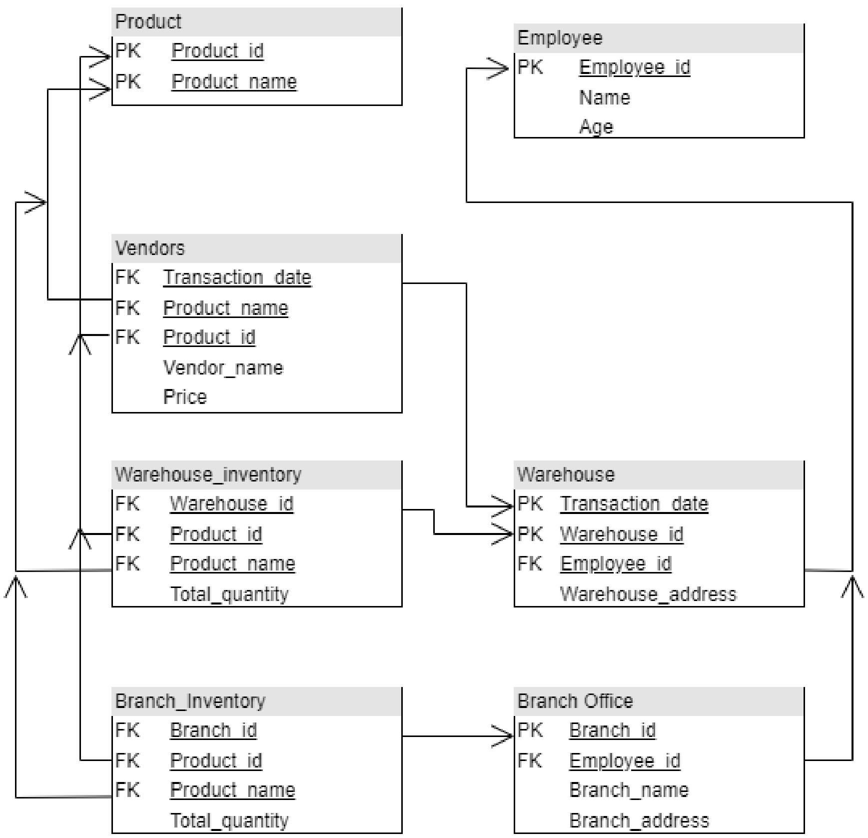
| |
|---|
| Average data queries take no longer than a few seconds |
| Database is secure |
| Database updates are secure |
| Database updates should be close to instant |
| Database capacity can hold all the data of the entities |
| Database system is always online, unless there"s a system upgrade |
| Database system is easy for users to navigate through it |
| Database system should be able to sustain many concurrent users |

# Conceptual and Logical Database Design

## ER Diagram



## Database Schema




## Business Rules and Integrity Constraints

- Constraints:
    - Having a NOT NULL constraint will prevent the null values from being entered into a column.
    - Having a *Unique constraint* will ensure that the values in a set of columns are unique and not null for all rows in the table. And the columns specified in a unique constraint must be defined as NOT NULL.
    - Having a Primary key constraint because you can use primary key and foreign key constraints to define relationships between tables.
    - Having a *Check constraint* because that specifies the values allowed in one or more columns of every row of a table
    - Having a *foreign key* constraint because it is a logical rule about the values in one or more columns in one or more tables

## Interface Requirements

1. The interface will need to present the user with a menu that will lead to a request.
2. Pull-down menus will need to be implemented as they are often used with web-based database systems.
3. The interface will have forms for the user to fill in order to make new entries to the database or some to query the other ones.
4. The interface will not allow unnecessary inputs into the forms.
5. The interface will display a structure of searched results for the user to save and query.
6. The interface will contain logs of previous updates and changes made to the database and will allow users to access those logs.
7. The log menu will not allow the user to make changes but only allow viewing of the statistical information.
8. The interface will have different access points for the various branch offices and each of the vendors that stock the supplies.
9. Each access point will allow the user to enter in text boxes to search for specific strings contained in the database.

# Normalization and Database Implementation and Testing

## Functional Dependencies and Database Normalization

Functional Dependencies:
Company_id → Company_name
Employee_id → Name, Age
Product_id → Product_name
Warehouse_id → Warehouse_address, Total_quantity, Transaction_date
Branch_id → Branch_name, Branch_address, Total_quantity
Vendor_id → Vendor_name, Price
Warehouse_id, Product_id, Product_name → Warehouse_Inventory.Total_quantity
Branch_id, Product_id, Product_name → Branch_Inventory.Total_quantity

## The Database System

```
CREATE TABLE Company (
	Company_id int UNSIGNED AUTO_INCREMENT,
	Company_name varchar(255),
	PRIMARY KEY (Company_id)
);

CREATE TABLE Employee (
	Employee_id int UNSIGNED AUTO_INCREMENT,
	Name varchar(255),
	Age int,
	PRIMARY KEY (Employee_id)
);

CREATE TABLE Product (
	Product_id int,
	Product_name varchar(255),
	PRIMARY KEY (Product_id, Product_name)
);

CREATE TABLE Branch_Office (
	Branch_id int UNSIGNED AUTO_INCREMENT,
	Branch_name varchar(255),
	Branch_address varchar(255),
	PRIMARY KEY (Branch_id)
);

CREATE TABLE Branch_inventory (
	Branch_id int UNSIGNED AUTO_INCREMENT,
	Product_id int,
	Product_name varchar(255),
	Total_quantity int,
	PRIMARY KEY (Branch_id, Product_id),
	FOREIGN KEY (Branch_id) REFERENCES Branch_office(Branch_id),
```

```
	FOREIGN KEY (Product_id, Product_name) REFERENCES Product(Product_id,
	Product_name)
);



CREATE TABLE Warehouse (
	Warehouse_id int UNSIGNED AUTO_INCREMENT,
	Warehouse_address varchar(255),
	Transaction_date varchar(255),
	PRIMARY KEY (Warehouse_id)
);

CREATE TABLE Warehouse_inventory (
	Warehouse_id int UNSIGNED AUTO_INCREMENT,
	Product_id int,
	Product_name varchar(255),
	Total_quantity int,
	PRIMARY KEY (Warehouse_id, Product_id),
	FOREIGN KEY (Warehouse_id) REFERENCES Warehouse(Warehouse_id),
	FOREIGN KEY (Product_id, Product_name) REFERENCES Product(Product_id,
	Product_name)
);

CREATE TABLE Vendors (
	Vendor_id int UNSIGNED AUTO_INCREMENT PRIMARY KEY,
	Transaction_date varchar(255),
	Product_name varchar(255),
	Product_id int,
	Vendor_name varchar(255),
	Price int,
	FOREIGN KEY (Product_id, Product_name) REFERENCES Product(Product_id,
Product_name)
);

INSERT INTO Company (Company_name)
VALUES (
	“FirstCompany”
);
```

| Company_id | Company_name |
|---|---|
| Filter... | Filter... |
| 1 | FirstCompany |

INSERT INTO employee (Name, Age)
VALUES (
  “FirstGuy”,
   21
 );

INSERT INTO employee (Name, Age)
VALUES (
   “SecondGuy”,
   42
 );

| Employee_id | Name | Age |
|---|---|---|
| Filter... | Filter... | Filter... |
| 1 | FirstGuy | 21 |
| 2 | SecondGuy | 42 |

INSERT INTO branch_office (branch_name, branch_address)
VALUES (
   “FirstBranch”,
   “FirstStreet”
);

| Branch_id | Branch_name | Branch_address |
|---|---|---|
| Filter... | Filter... | Filter... |
| 1 | FirstBranch' | FirstStreet |

INSERT INTO warehouse (
   Warehouse_address,
   Transaction_date
 )

```
VALUES (
    “FirstWarehouseStreet”,
    “November11”
);
```

| Warehouse_id | Warehouse_add... | Transaction_date |
|---|---|---|
| Filter... | Filter... | Filter... |
| 1 | FirstWarehouseStreet | November11 |

```
INSERT INTO product (Product_id, Product_name)
VALUES (
    1,
    “Apple”
  );

INSERT INTO product (Product_id, Product_name)
VALUES (
    2,
    “Banana”
  );

INSERT INTO product (Product_id, Product_name)
VALUES (
    3,
    “Orange”
  );
```

| Product_id | Product_name |
|---|---|
| Filter... | Filter... |
| 1 | Apple |
| 2 | Banana |
| 3 | Orange |

```
INSERT INTO vendors (
    Transaction_date,
    Product_name,
    Product_id,
    Vendor_name,
    Price
```

```
 )
VALUES (
   “November13”,
   “Apple”,
   1,
   “AppleVendor”,
   1
 );

INSERT INTO vendors (
   Transaction_date,
   Product_name,
   Product_id,
   Vendor_name,
   Price
 )
VALUES (
   “November13”,
   “Banana”,
   2,
   “BananaVendor”,
   1
 );

INSERT INTO vendors (
   Transaction_date,
   Product_name,
   Product_id,
   Vendor_name,
   Price
 )
VALUES (
   “November13”,
   “Orange”,
   3,
   “OrangeVendor”,
   2
 );
```

| Vendor_id | Transaction_date | Product_name | Product_id | Vendor_name | Price |
|---|---|---|---|---|---|
| Filter... | Filter... | Filter... | Filter... | Filter... | Filter... |
| 1 | November13 | Apple | 1 | AppleVendor | 1 |
| 2 | November13 | Banana | 2 | BananaVendor | 1 |
| 3 | November13 | Orange | 3 | OrangeVendor | 2 |

```
INSERT INTO branch_inventory (
    Branch_id,
    Product_id,
    Product_name,
    Total_quantity
  )
VALUES (
    1,
    1,
    “Apple”,
    100
  );

INSERT INTO branch_inventory (
    Branch_id,
    Product_id,
    Product_name,
    Total_quantity
  )
VALUES (
    1,
    2,
    "Banana",
    50
  );

INSERT INTO branch_inventory (
    Branch_id,
    Product_id,
    Product_name,
    Total_quantity
  )
VALUES (
    1,
    3,
    "Orange",
```

```
  25
 );
```

| Branch_id | Product_id | Product_name | Total_quantity |
|---|---|---|---|
| Filter... | Filter... | Filter... | Filter... |
| 1 | 1 | Apple | 100 |
| 1 | 2 | Banana | 50 |
| 1 | 3 | Orange | 25 |

```
INSERT INTO warehouse_inventory (
  Warehouse_id,
  Product_id,
  Product_name,
  Total_quantity
 )
VALUES (
  1,
  1,
  "Apple",
  200
 );

INSERT INTO warehouse_inventory (
  Warehouse_id,
  Product_id,
  Product_name,
  Total_quantity
 )
VALUES (
  1,
  2,
  "Banana",
  150
 );
INSERT INTO warehouse_inventory (
  Warehouse_id,
  Product_id,
  Product_name,
```

```
    Total_quantity
  )
VALUES (
    1,
    3,
    "Orange",
    50
  );
```

| Warehouse_id | Product_id | Product_name | Total_quantity |
|---|---|---|---|
| Filter... | Filter... | Filter... | Filter... |
| 1 | 1 | Apple | 200 |
| 1 | 2 | Banana | 150 |
| 1 | 3 | Orange | 50 |

## Additional Queries and Views

```
CREATE VIEW REDUCED_BRANCH_INV AS
        SELECT *
        FROM Branch_inventory
        WHERE Total_quantity < 50;
```

| Branch_id | Product_id | Product_name | Total_quantity |
|---|---|---|---|
| Filter... | Filter... | Filter... | Filter... |
| 1 | 3 | Orange | 25 |

```
CREATE VIEW REDUCED_WAREHOUSE_INV AS
        SELECT *
        FROM Warehouse_inventory
        WHERE Total_quantity < 100;
```

| Warehouse_id | Product_id | Product_name | Total_quantity |
|---|---|---|---|
| Filter... | Filter... | Filter... | Filter... |
| 1 | 3 | Orange | 50 |

# User Application Interface

For this project, our interface for the supply cabinet project was designed to output the data out from the terminal. There are 5 alternative options to choose from and the system asks for user input. Each option references different queries into the data in order to retrieve and write data in and out of the database. These options include viewing the warehouse inventory, request items, purchase items, view product costs, and quit. Each function takes a query and runs it in mysql.

# Conclusions and Future Work

This project has been very helpful for learning how to make a SQL database and creating a frontend with a backend server to connect the three parts. A possible future work could be simulating a school's database using SQL where tables could be created for the departments, students, majors, classes, and much more. We could CRUD the tables and insert or remove information from those tables while building a UI for users to interact with those tables.

# References


https://www.youtube.com/watch?v=wzdCpJY6Y4c&ab_channel=BoostMyTool