The Evercreek Company Case Study

The owner of the Evercreek Company maintains a database to keep record of their employees as well as their product sales. It tries to keep track of the company performance, which is split into branches. The database utilizes the information provided to make changes that would produce more profit for the company.

Data Requirements

Employees

The data stored for each employee includes the ID number, name, age, address (street, city, zip), phone number, email, salary, branch, years worked at the company (yearsWorked), the ID of the manager (IDManager), and the name of the manager (nameOfManager). The employee is uniquely defined by the ID number. The employee information is stored to identify them as well as measure their performance in the company based on the performance of the branch they are working in.

Managers

The data stored for each manager includes the ID number, name, age, address (street, city, zip), phone number, email, salary, branch, and years worked at the company (yearsWorked). The manager is uniquely defined by the ID number. The manager works under a specific branch and keeps track of employees that work for the manager.

Branches

The company is composed of several branches. Each branch is uniquely defined by its name. The branch is also composed of the number of employees (numEmployees), number of managers (numManagers), the date the branch was created (dateCreated), the cost of the branch (cost), and the profit of the branch (profit). The company can determine which branch is making the most profit, and they can make decisions regarding employees and managers based on this information.

Investors

Investors of a company are uniquely defined by their ID number. They are also defined by their name, the amount of stock they own (numStocks), phone number, email , how much they invested (contribution), and date joined. Investors purchase stocks in the company, increasing the worth of the company. The more investors willing to buy stock, the more the company grows.

Products

A product is uniquely defined by its product ID. It is also defined by its name, functionality, name, date created, the name of the branch the product was made by (branch), the price the product is being sold at (price), and how much it costs to make the product (cost). The company spends money to create products. Customers buy products, gaining profit for the company.

Stocks

Stocks are uniquely defined by its ID. Stocks also contain the name and ID of the investor, the price of stock (price), and the number of shares (numShares). Stocks are bought by investors, which help grow the company.

Customer

Customers are uniquely defined by the order number (orderNum). They also have a name, address(street, city,zip), phone number, ID of product purchased , name of product purchased, email, and amount spent. Customers purchase products from the company, allowing the company to profit.

Query Transactions

Listed here are some examples of query transactions that should be supported by the Evercreek Company database system:

(a) Show the list of employee names and their id number who work for the manager identified by the id number 95192. **[GO TO Employee -> Works for Specific Manager]**  
(b) Show all the names of the branches at the company. **[GO TO Company -> Get all Branch Names]**

(c) Show the names and id of all the employees and managers that work for the HR Branch of the company. **[GO TO Branch -> Employees of Branch]**

(d) Show the date that a branch was created, the cost the branch requires, and how much profit a branch is generating. **[GO TO Branch -> Branch Info]**

(e) Show the customers of a product, whose product ID is 28739.  
**[GO TO Product -> Get Customer(s) of Product]**

(f) For each stock owned by an investor, whose investor ID is 93193, show the amount of shares and the price. **[GO TO Stock -> Get Stock(s) of Specific Investor]**

(g) Show the customers that bought more than one product from the company. **[GO TO Customers -> Get Customer(s) that bought more than one Product]**

(h) Show the number of employees and managers that have been working at the company for over 10 years. **[GO TO Company -> Years Worked]**

(i) Show the salaries paid to the employees and managers of the Software branch. **[GO TO Company -> Salary per Branch]**

(j) Display the price(s) and cost(s) of the most commonly sold product(s) from the company. **[GO TO Company -> Price and Cost of most Successful Product]**

(k) Display the names and ids of employees and managers that own stock in the company. **[GO TO Company ->Employees/Managers that are also Investors]**

(l) Out of all the least commonly sold product(s) of the company, display the product name and id of the products that had the highest cost to make. **[GO TO Company -> Most costly Unpopular Product]**

(m) Show the product(s) from the Electronics branch that took the lowest cost to make. **[GO TO Branch -> Cheapest Product per Branch]**

(n) Show the order number(s), name(s), and email(s) of customers who had purchased the most expensive product from the Software branch. **[GO TO Branch -> Orders with most expensive Branch product]**

(o) Show the name, email, and ID of distinct Managers, Employees, and Investors of the company. **[GO TO Company -> Distinct Employees, Managers, and Investors]**

# Textual description of the project (Milestone 0) The owner of the Evercreek Company maintains a database to keep record of their employees as well as their product sales. It tries to keep track of the company performance, which is split into branches. The database utilizes the information provided to make changes that would produce more profit for the company.

# E/R Diagram

# Description of Entities.

Description of what each entity represents.

|  |  |
| --- | --- |
| Entity name | Description |
| Employees | The data stored for each employee includes the ID number, name, age, address (street, city, zip), phone number, email, salary, branch, years worked at the company (yearsWorked), and the ID of the manager (IDManager). The employee is uniquely defined by the ID number. The employee information is stored to identify them as well as measure their performance in the company based on the performance of the branch they are working in. |
| Managers | The data stored for each manager includes the managerID, name, age, address (street, city, zip), phone number, email, salary, branch, and years worked at the company (yearsWorked). The manager is uniquely defined by the ID number. The manager works under a specific branch and keeps track of employees that work for the manager. |
| Branches | The company is composed of several branches. Each branch is uniquely defined by its name. The branch is also composed of the number of employees (numEmployees), number of managers (numManagers), the date the branch was created (dateCreated), the cost of the branch (cost), and the profit of the branch (profit). The company can determine which branch is making the most profit, and they can make decisions regarding employees and managers based on this information. |
| Investors | Investors of a company are uniquely defined by their ID number. They are also defined by their name, the amount of stock they own (numStocks), phone number, email , how much they invested (contribution), and date joined. Investors purchase stocks in the company, increasing the worth of the company. The more investors willing to buy stock, the more the company grows. |
| Products | A product is uniquely defined by its product ID. It is also defined by its name, functionality, date created, the name of the branch the product was made by (branch), the price the product is being sold at (price), and how much it costs to make the product (cost). The company spends money to create products. Customers buy products, gaining profit for the company. |
| Stocks | Stocks are uniquely defined by its ID. Stocks also contain the ID of the investor, the price of stock (price), and the number of shares (numShares). Stocks are bought by investors, which help grow the company. |
| Customers | Customers are uniquely defined by the customerID. They also have a name, address(street, city,zip), phone number, and email. Customers purchase products from the company, allowing the company to profit. |
| Orders | Orders are uniquely defined by order number (orderNum). They also have the ID of the product purchased, customerID, and amount spent. Orders show which products are the most profitable to the company. |

# Description of relationships on E/R Diagram.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Entity1 | Entity2 | Entity 1 -> Entity 2 Role Cardinality | Entity 2 -> Entity 1 Role Cardinality | Description |
| Buys | Customers | Products | Many | Many | Many customers can buy many products |
| Has | Customers | Orders | Many | One | Many different orders can be made by exactly one customer |
| MadeOf | Products | Orders | Many | Many | Many orders can be made up of many products |
| Makes | Products | Branches | One | Many | One branch can make many products |
| Owns | Branches | Stocks | Many | One | Many stocks can be owned by exactly one branch |
| InvestsIn | Stocks | Investors | Many | One | Many stocks can be invested by exactly one investor |
| Contains | Employees | Branches | One | Many | Many employees work for exactly one branch |
| RunBy | Branches | Managers | Many | One | Many managers run exactly one branch |
| Manages | Employees | Managers | Many | One | Many employees are managed by exactly one manager |

# Description of Entity attributes.

Description of attributes in each entity:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Used By | Used By An Identifier | Data Type | Description |
| IDNumber | Employees | Yes | INT | Used to uniquely identify an individual associated with the company |
| firstName | Employees | No | VARCHAR(50) | Shows the name of an individual |
| lastName | Employees | No | VARCHAR(50) | Shows the name of an individual |
| DOB | Employees | No | DATE | Shows the date of birth of an individual |
| street | Employees | No | VARCHAR(25) | Shows the street where the individual lives |
| city | Employees | No | VARCHAR(25) | Shows the city where the individual lives |
| zip | Employees | No | INT | Shows the zip code in which the individual lives in |
| phoneNumber | Employees | No | VARCHAR(25) | Shows the individual’s phone number |
| email | Employees | No | VARCHAR(50) | Shows the individual’s email address |
| salary | Employees | No | INT | Shows the individual’s salary |
| startDate | Employees | No | INT | Used to show the date the employee started working |
| IDNumber | Managers | Yes | INT | Used to uniquely identify a manager |
| firstName | Managers | No | VARCHAR(50) | Shows the first name of an individual |
| lastName | Managers | No | VARCHAR(50) | Shows the last name of an individual |
| DOB | Managers | No | INT | Shows the date of birth of an individual |
| street | Managers | No | VARCHAR(25) | Shows the street where the individual lives |
| city | Managers | No | VARCHAR(25) | Shows the city where the individual lives |
| zip | Managers | No | INT | Shows the zip code in which the individual lives in |
| phoneNumber | Managers | No | VARCHAR(25) | Shows the individual’s phone number |
| email | Managers | No | VARCHAR(50) | Shows the individual’s email address |
| salary | Managers | No | INT | Shows the individual’s salary |
| branch | Managers | No | VARCHAR(25) | Shows the branch in which the individual works at |
| startDate | Managers | No | INT | Used to show when manager started working |
| branchName | Branches | Yes | VARCHAR(25) | Uniquely identifies a branch |
| dateCreated | Branches | No | DATE | Shows the date in which the branch was created |
| cost | Branches | No | INT | Shows the cost needed to maintain the branch |
| profit | Branches | No | INT | Shows the profit generated by the branch |
| IDNumber | Investors | Yes | INT | Uniquely identifies an investor |
| firstName | Investors | No | VARCHAR(50) | Shows the first name of the individual |
| lastName | Investors | No | VARCHAR(50) | Shows the last name of the individual |
| phoneNumber | Investors | No | VARCHAR(25) | Shows the individual’s phone number |
| email | Investors | No | VARCHAR(50) | Shows the individual’s email address |
| contribution | Investors | No | INT | Shows how much the investor has invested in the company |
| dateJoined | Investors | No | DATE | Shows the date that the investor joined |
| productID | Products | Yes | INT | Uniquely identifies a product |
| name | Products | No | VARCHAR(50) | Name of the product |
| functionality | Products | No | VARCHAR(50) | Description of a product and what it does |
| dateCreated | Products | No | DATE | Date at which the product was made |
| priceSold | Products | No | INT | Price at which product is sold |
| costToMake | Products | No | INT | Cost for company to make product |
| stockID | Stocks | Yes | INT | Shows the ID of the stock |
| price | Stocks | No | INT | Shows the price of the stock |
| numShared | Stocks | No | INT | Showed the number of shares in a stock |
| customerID | Customers | Yes | INT | Used to uniquely identify a customer |
| firstName | Customers | No | VARCHAR(50) | Shows the first name of an individual |
| lastName | Customers | No | VARCHAR(50) | Shows the last name of an individual |
| DOB | Customers | No | DATE | Shows the date of birth of an individual |
| street | Customers | No | VARCHAR(25) | Shows the street where the individual lives |
| city | Customers | No | VARCHAR(25) | Shows the city where the individual lives |
| zip | Customers | No | INT | Shows the zip code in which the individual lives in |
| phoneNumber | Customers | No | VARCHAR(25) | Shows the individual’s phone number |
| email | Customers | No | VARCHAR(50) | Shows the individual’s email address |
| orderNum | Orders | Yes | INT | Used to uniquely identify an order |

# Analysis of functional and non-functional requirements.

Description of design assumptions and detailing the textual description of project.

**Functional:**  
Listed here are some examples of query transactions that should be supported by the Evercreek Company database system:

(a) Show the list of employee names and their id number who work for the manager identified by the id number 95192.  
  
(b) Show all the names of the branches at the company.

(c) Show the names and id of all the employees and managers that work for the HR Branch of the company.

(d) Show the date that a branch was created, the cost the branch requires, and how much profit a branch is generating.

(e) Show the customers of a product, whose product ID is 28739.

(f) For each stock owned by an investor, whose investor ID is 93193, show the amount of shares and the price.

(g) Show the customers that bought more than one product from the company.

(h) Show the number of employees and managers that have been working at the company for over 10 years.

(i) Show the salaries paid to the employees and managers of the Software branch.

(j) Display the price(s) and cost(s) of the most commonly sold product(s) from the company.

(k) Display the names and ids of employees and managers that own stock in the company.  
  
(l) Out of all the least commonly sold product(s) of the company, display the product name and id of the products that had the highest cost to make.  
  
(m) Show the product(s) from the Electronics branch that took the lowest cost to make.

(n) Show the order number(s), name(s), and email(s) of customers who had purchased the most expensive product from the Software branch.  
  
(o) Show the name, email, and ID of distinct Managers, Employees, and Investors of the company.

**Non-functional** – examples: scalability, flexibility, extensibility, efficiency of storage, efficiency of processing. Describe how your application will meet these requirements.  
a) Searching the database will take less than 2 seconds.  
b) Modified data will be updated for all users within 2 seconds.  
c) Data input will take less than 2 seconds  
d) Customers and Investors should be able to receive info within 2 minutes.  
e) Branches can access all information; Managers can access Employees, Customer, and Order records; Employees can access Customer and Order records.  
f) Less than 1 hour of downtime per month.  
g) Database can handle high load with no more than 10 seconds of lag.  
h) In case of system crash, database will reboot within 1 hour.

# Relational model (translating E/R into table model). Entities: Employees(IDNumber, firstName, lastName, DOB, street, city, zip, phoneNumber, email, salary, startDate) Managers(IDNumber, firstName, lastName, DOB, street, city, zip, phoneNumber, email, salary, branch, startDate) Branches(branchName, dateCreated, cost, profit) Investors(IDNumber, firstName, lastName, phoneNumber, email, contribution, dateJoined) Products(productID, name, functionality, dateCreated, priceSold, costToMake) Stocks(stockID, price, numShared) Customers(customerID, firstName, lastName, DOB, street, city, zip, phoneNumber, email) Orders(orderedNum) Relations: Buys(customerID,productID) Has(customerID,orderID) MadeOf(productID,orderID, quantity) Makes(productID,branchName) Owns(branchName,stockID) InvestsIn(stockID,investorID) Contains(employeeID,branchName) RunBy(branchName,managerID) Manages(employeeID,managerID)

# Normalization – 3NF

All tables in the design have been normalized. No normalization required.

# DDL Script that creates a database

CREATE TABLE Employees(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

DOB DATE,

street VARCHAR(25),

city VARCHAR(25),

zip INT,

phoneNumber VARCHAR(25),

email VARCHAR(50),

salary INT,

startDate DATE

);

CREATE TABLE Managers(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

DOB DATE,

street VARCHAR(25),

city VARCHAR(25),

zip INT,

phoneNumber VARCHAR(25),

email VARCHAR(50),

salary INT,

branch VARCHAR(25),

yearsWorked INT

);

CREATE TABLE Branches(

branchName VARCHAR(25) PRIMARY KEY,

dateCreated DATE,

cost INT,

profit INT

);

CREATE TABLE Investors(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

phoneNumber VARCHAR(25),

email VARCHAR(50),

contribution INT,

dateJoined DATE

);

CREATE TABLE Products(

productID INT PRIMARY KEY,

name VARCHAR(50),

functionality VARCHAR(50),

dateCreated DATE,

priceSold INT,

costToMake INT

);

CREATE TABLE Stocks(

stockID INT PRIMARY KEY,

price INT,

numShared INT

);

CREATE TABLE Customers(

customerID INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

DOB DATE,

street VARCHAR(25),

city VARCHAR(25),

zip INT,

phoneNumber VARCHAR(25),

email VARCHAR(50)

);

CREATE TABLE Orders(

orderNum INT PRIMARY KEY

);

CREATE TABLE Buys(

customerID INT,

productID INT,

PRIMARY KEY(customerID, productID)

);

CREATE TABLE Has(

customerID INT,

orderID INT,

PRIMARY KEY(customerID, orderID)

);

CREATE TABLE MadeOf(

productID INT,

orderID INT,  
quantity INT,

PRIMARY KEY(productID, orderID)

);

CREATE TABLE Makes(

productID INT,

branchName VARCHAR(25),

PRIMARY KEY(productID, branchName)

);

CREATE TABLE Owns(

branchName VARCHAR(25),

stockID INT,

PRIMARY KEY(branchName, stockID)

);

CREATE TABLE InvestsIn(

stockID INT,

investorID INT,

PRIMARY KEY(stockID, investorID)

);

CREATE TABLE Contains(

employeeID INT,

branchName VARCHAR(25),

PRIMARY KEY(employeeID, branchName)

);

CREATE TABLE RunBy(

branchName VARCHAR(25),

managerID INT,

PRIMARY KEY(branchName, managerID)

);

CREATE TABLE Manages(

employeeID INT,

managerID INT,

PRIMARY KEY(employeeID, managerID)

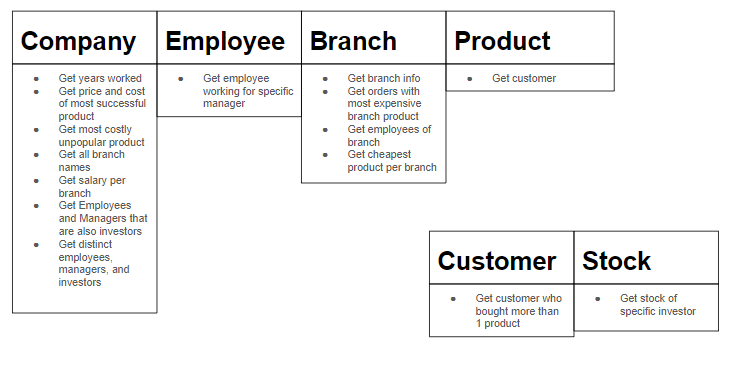
);

**See attached file(s)**

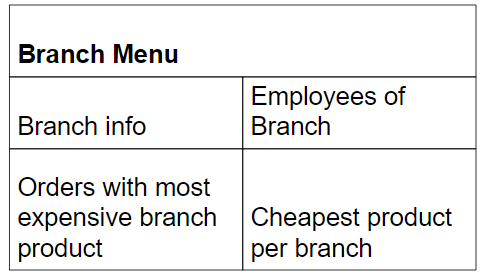
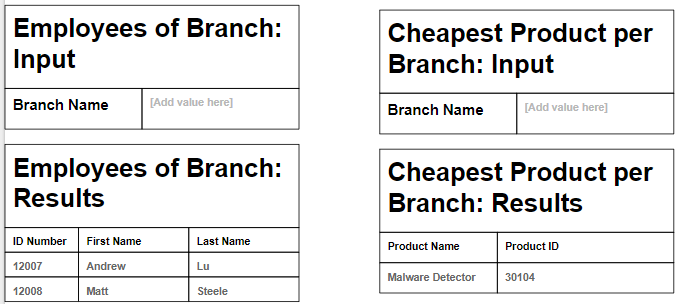
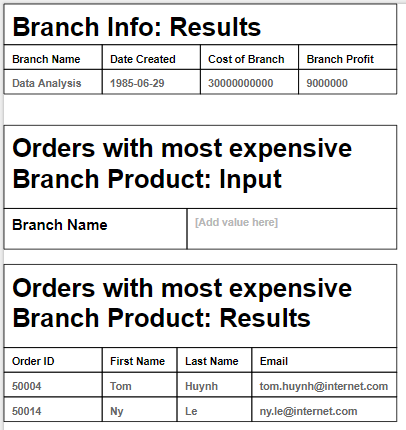
10. Populating tables with sample data

Prepare text files with data and use one of the loading methods presented in the tutorials/lecture. Depending on the tables and problem, 7-30 rows per each table.  
  
**See attached file(s)  
  
1. Design of the application menu**

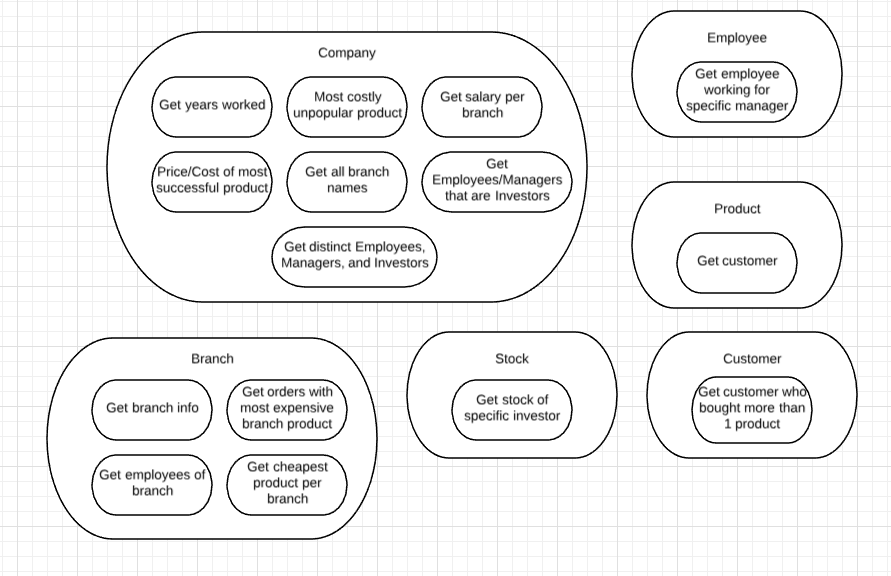
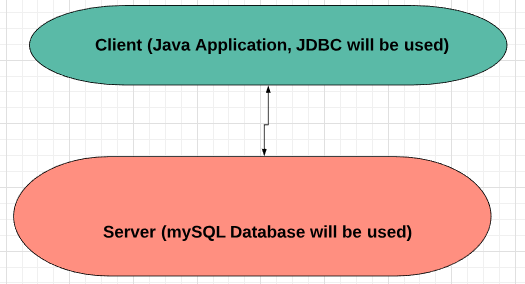
Propose and design a menu for your application (use ‘mock’ menu).



# **2. GUI design for the chosen module**

Propose and design a ‘mock’ interface for the chosen module of your application.  
  


# **3. Application architecture**

Modules + descriptions of the modules. Include diagram of the Client-Server architecture. The project must be implemented using Client-Server architecture and using JDBC.  
  
**Company:** Overall/General information about company  
**Employee:** Gets information about employees  
**Branch:** Gets information about branches and their responsibilities  
**Product:** Gets information about product and who buys them  
**Customer:** Check customer purchases  
**Stock:** Get information about stocks and their investors  
  
  
  


# **4. Server-side and Client-side functionality.**

Describe which functionalities will be implemented on the Client side and which will be implemented on the server side.  
  
**Client:**   
**(a) Show the list of employee names and their id number who work for the manager identified by the id number 95192.   
  
(b) Show all the names of the branches at the company.**

**(c) Show the names and id of all the employees and managers that work for the HR Branch of the company.  
  
(d) Show the date that a branch was created, the cost the branch requires, and how much profit a branch is generating.**

**(e) Show the customers of a product, whose product ID is 28739.**

**(f) For each stock owned by an investor, whose investor ID is 93193, show the amount of shares and the price.**

**(g) Show the customers that bought more than one product from the company.**

**(i) Show the salaries paid to the employees and managers of the Software branch.**

**(j) Display the price(s) and cost(s) of the most commonly sold product(s) from the company.**

**(k) Display the names and ids of employees and managers that own stock in the company.  
  
(o) Show the name, email, and ID of distinct Managers, Employees, and Investors of the company.   
  
Server:   
(h) Show the number of employees and managers that have been working at the company for over 10 years.  
  
(l) Out of all the least commonly sold product(s) of the company, display the product name and id of the products that had the highest cost to make.  
  
(m) Show the product(s) from the Electronics branch that took the lowest cost to make.  
  
(n) Show the order number(s), name(s), and email(s) of customers who had purchased the most expensive product from the Software branch.**

**5. Integrity: constraints and triggers**

Describe how integrity constraints will be enforced in your database system application (in the table definition and programmatically with assertion/triggers). Include code for defining constraints and code for triggers.  
  
**CONSTRAINTS:**   
CREATE TABLE Employees(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50) NOT NULL,

lastName VARCHAR(50) NOT NULL,

DOB DATE NOT NULL,

street VARCHAR(25) NOT NULL,

city VARCHAR(25) NOT NULL,

zip INT NOT NULL,

phoneNumber VARCHAR(25) NOT NULL,

email VARCHAR(50) NOT NULL,

salary INT NOT NULL,

startDate DATE NOT NULL

);

CREATE TABLE Managers(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50) NOT NULL,

lastName VARCHAR(50) NOT NULL,

DOB DATE NOT NULL,

street VARCHAR(25) NOT NULL,

city VARCHAR(25) NOT NULL,

zip INT NOT NULL,

phoneNumber VARCHAR(25) NOT NULL,

email VARCHAR(50) NOT NULL,

salary INT NOT NULL,

branch VARCHAR(25) NOT NULL,

startDate DATE NOT NULL

);

CREATE TABLE Branches(

branchName VARCHAR(25) PRIMARY KEY,

dateCreated DATE NOT NULL,

cost INT NOT NULL,

profit INT NOT NULL

);

CREATE TABLE Investors(

IDNumber INT PRIMARY KEY,

firstName VARCHAR(50) NOT NULL,

lastName VARCHAR(50) NOT NULL,

phoneNumber VARCHAR(25) NOT NULL,

email VARCHAR(50) NOT NULL,

contribution INT NOT NULL,

dateJoined DATE NOT NULL

);

CREATE TABLE Products(

productID INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

functionality VARCHAR(50) NOT NULL,

dateCreated DATE NOT NULL,

priceSold INT NOT NULL,

costToMake INT NOT NULL

);

CREATE TABLE Stocks(

stockID INT PRIMARY KEY,

price INT NOT NULL,

numShared INT NOT NULL

);

CREATE TABLE Customers(

customerID INT PRIMARY KEY,

firstName VARCHAR(50) NOT NULL,

lastName VARCHAR(50) NOT NULL,

DOB DATE NOT NULL,

street VARCHAR(25) NOT NULL,

city VARCHAR(25) NOT NULL,

zip INT NOT NULL,

phoneNumber VARCHAR(25) NOT NULL,

email VARCHAR(50) NOT NULL

);  
  
CREATE TABLE Orders(

orderNum INT PRIMARY KEY

);

CREATE TABLE Buys(

customerID INT NOT NULL,

productID INT NOT NULL,

PRIMARY KEY(customerID, productID)

);

CREATE TABLE Has(

customerID INT NOT NULL,

orderID INT NOT NULL,

PRIMARY KEY(customerID, orderID)

);

CREATE TABLE MadeOf(

productID INT NOT NULL,

orderID INT NOT NULL,

quantity INT NOT NULL,

PRIMARY KEY(productID, orderID)

);

CREATE TABLE Makes(

productID INT NOT NULL,

branchName VARCHAR(25) NOT NULL,

PRIMARY KEY(productID, branchName)

);

CREATE TABLE Owns(

branchName VARCHAR(25) NOT NULL,

stockID INT NOT NULL,

PRIMARY KEY(branchName, stockID)

);

CREATE TABLE InvestsIn(

stockID INT NOT NULL,

investorID INT NOT NULL,

PRIMARY KEY(stockID, investorID)

);

CREATE TABLE Contains(

employeeID INT NOT NULL,

branchName VARCHAR(25) NOT NULL,

PRIMARY KEY(employeeID, branchName)

);

CREATE TABLE RunBy(

branchName VARCHAR(25) NOT NULL,

managerID INT NOT NULL,

PRIMARY KEY(branchName, managerID)

);

CREATE TABLE Manages(

employeeID INT NOT NULL,

managerID INT NOT NULL,

PRIMARY KEY(employeeID, managerID)

);

**TRIGGERS:  
  
Employees:  
CREATE TABLE employeeHistory\_table**

**(action VARCHAR(50),**

**IDNumber INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoEmployeeHistory**

**BEFORE INSERT ON Employees**

**FOR EACH ROW**

**INSERT INTO employeeHistory\_table(action, IDNumber) VALUES('Inserting', NEW.IDNumber);**

**CREATE TRIGGER deletionFromEmployeeHistory**

**BEFORE DELETE ON Employees**

**FOR EACH ROW**

**INSERT INTO employeeHistory\_table(action, IDNumber) VALUES('Deleting', OLD.IDNumber);**

**Managers:  
CREATE TABLE managerHistory\_table**

**(action VARCHAR(50),**

**IDNumber INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoManagerHistory**

**BEFORE INSERT ON Managers**

**FOR EACH ROW**

**INSERT INTO managerHistory\_table(action, IDNumber) VALUES('Inserting', NEW.IDNumber);**

**CREATE TRIGGER deletionFromManagerHistory**

**BEFORE DELETE ON Managers**

**FOR EACH ROW**

**INSERT INTO managerHistory\_table(action, IDNumber) VALUES('Deleting', OLD.IDNumber);**

**Branches:  
CREATE TABLE branchesHistory\_table**

**(action VARCHAR(50),**

**branchName VARCHAR(25) DEFAULT NULL);**

**CREATE TRIGGER insertIntoBranchHistory**

**BEFORE INSERT ON Branches**

**FOR EACH ROW**

**INSERT INTO branchesHistory\_table(action, branchName) VALUES('Inserting', NEW.branchName);**

**CREATE TRIGGER deletionFromBranchHistory**

**BEFORE DELETE ON Branches**

**FOR EACH ROW**

**INSERT INTO branchesHistory\_table(action, branchName) VALUES('Deleting', OLD.branchName);**

**Investors:  
CREATE TABLE investorHistory\_table**

**(action VARCHAR(50),**

**IDNumber INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoInvestorHistory**

**BEFORE INSERT ON Investors**

**FOR EACH ROW**

**INSERT INTO investorHistory\_table(action, IDNumber) VALUES('Inserting', NEW.IDNumber);**

**CREATE TRIGGER deletionFromInvestorHistory**

**BEFORE DELETE ON Investors**

**FOR EACH ROW**

**INSERT INTO investorHistory\_table(action, IDNumber) VALUES('Deleting', OLD.IDNumber);**

**Products:  
CREATE TABLE productsHistory\_table**

**(action VARCHAR(50),**

**productID INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoProductsHistory**

**BEFORE INSERT ON Products**

**FOR EACH ROW**

**INSERT INTO productsHistory\_table(action, productID) VALUES('Inserting', NEW.productID);**

**CREATE TRIGGER deletionFromProductsHistory**

**BEFORE DELETE ON Products**

**FOR EACH ROW**

**INSERT INTO productsHistory\_table(action, productID) VALUES('Deleting', OLD.productID);**

**Stocks:  
CREATE TABLE stockHistory\_table**

**(action VARCHAR(50),**

**stockID INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoStockHistory**

**BEFORE INSERT ON Stocks**

**FOR EACH ROW**

**INSERT INTO stockHistory\_table(action, stockID) VALUES('Inserting', NEW.stockID);**

**CREATE TRIGGER deletionFromStockHistory**

**BEFORE DELETE ON Stocks**

**FOR EACH ROW**

**INSERT INTO stockHistory\_table(action, stockID) VALUES('Deleting', OLD.stockID);**

**Customers:**

**CREATE TABLE customersHistory\_table**

**(action VARCHAR(50),**

**customerID INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoCustomersHistory**

**BEFORE INSERT ON Customers**

**FOR EACH ROW**

**INSERT INTO customersHistory\_table(action, customerID) VALUES('Inserting', NEW.customerID);**

**CREATE TRIGGER deletionFromCustomersHistory**

**BEFORE DELETE ON Customers**

**FOR EACH ROW**

**INSERT INTO customersHistory\_table(action, customerID) VALUES('Deleting', OLD.customerID);**

**Orders:  
  
CREATE TABLE ordersHistory\_table**

**(action VARCHAR(50),**

**orderNum INT DEFAULT NULL);**

**CREATE TRIGGER insertIntoOrdersHistory**

**BEFORE INSERT ON Orders**

**FOR EACH ROW**

**INSERT INTO ordersHistory\_table(action, orderNum) VALUES('Inserting', NEW.orderNum);**

**CREATE TRIGGER deletionFromOrdersHistory**

**BEFORE DELETE ON Orders**

**FOR EACH ROW**

**INSERT INTO ordersHistory\_table(action, orderNum) VALUES('Deleting', OLD.orderNum);**

# **6. Performance: Indexes**

Propose and design a set of indexes suitable for your systems, taking into account most common query transactions performed in your application. Give code for creating these indexes. Consider also designing materialized views to speed up certain queries.

CREATE INDEX EmpManInd ON Manages(employeeID, managerID);  
  
CREATE MATERIALIZED VIEW EmpManView AS   
SELECT \*   
FROM Manages;

CREATE INDEX EmpBrnInd ON Contains(employeeID,branchName);

CREATE INDEX ManBrnInd ON RunBy(branchName,managerID);  
  
CREATE INDEX CustBuysInd ON Buys(customerID,productID);  
  
CREATE MATERIALIZED VIEW CustBuysView AS   
SELECT \*   
FROM Buys;  
  
CREATE INDEX StockInvestInd ON InvestsIn(stockID,investorID);  
  
CREATE MATERIALIZED VIEW StockInvestView AS   
SELECT \*   
FROM InvestsIn;  
  
CREATE INDEX BrnProdInd ON Makes(productID,branchName);

# **7. Concurrency: Transactions for the chosen module**

Describe atomic transactions in the chosen module and how concurrency will be supported.

**Chosen module: Select Employees With Specific Manager**

Assume 2 users are looking up employees for the same manager, identified by 95192. Assume modification queries are in the program.

**SELECT e.IDNumber, e.firstName, e.lastName**

**FROM Employees e, Manages m**

**WHERE m.managerID = 95192**

**AND m.employeeID = e.IDNumber;**

**SAVEPOINT SELECT1\_DONE;**  
  
**INSERT INTO Employees VALUES(12000 ,"James", "Deely", 1984-01-01,"Mulberry St.", "Joseph City", 95121, "(408)123-4560", "james.deely@evercreek.org", 50000, 2001-12-30);  
  
INSERT INTO Manages VALUES(12001,95192);**

**SAVEPOINT INSERT\_DONE;**

**SELECT e.IDNumber, e.firstName, e.lastName**

**FROM Employees e, Manages m**

**WHERE m.managerID = 95192**

**AND m.employeeID = e.IDNumber;**

**SAVEPOINT SELECT2\_DONE;**

**ROLLBACK TO SELECT1\_DONE;**

**We added a new employee and connected it to a manager. Select2 would produce 1 extra employee, but since we rolled back to Select1, the original employees are here. We want to be sure the first user gets the first result set instead of an extra employee.**