University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2

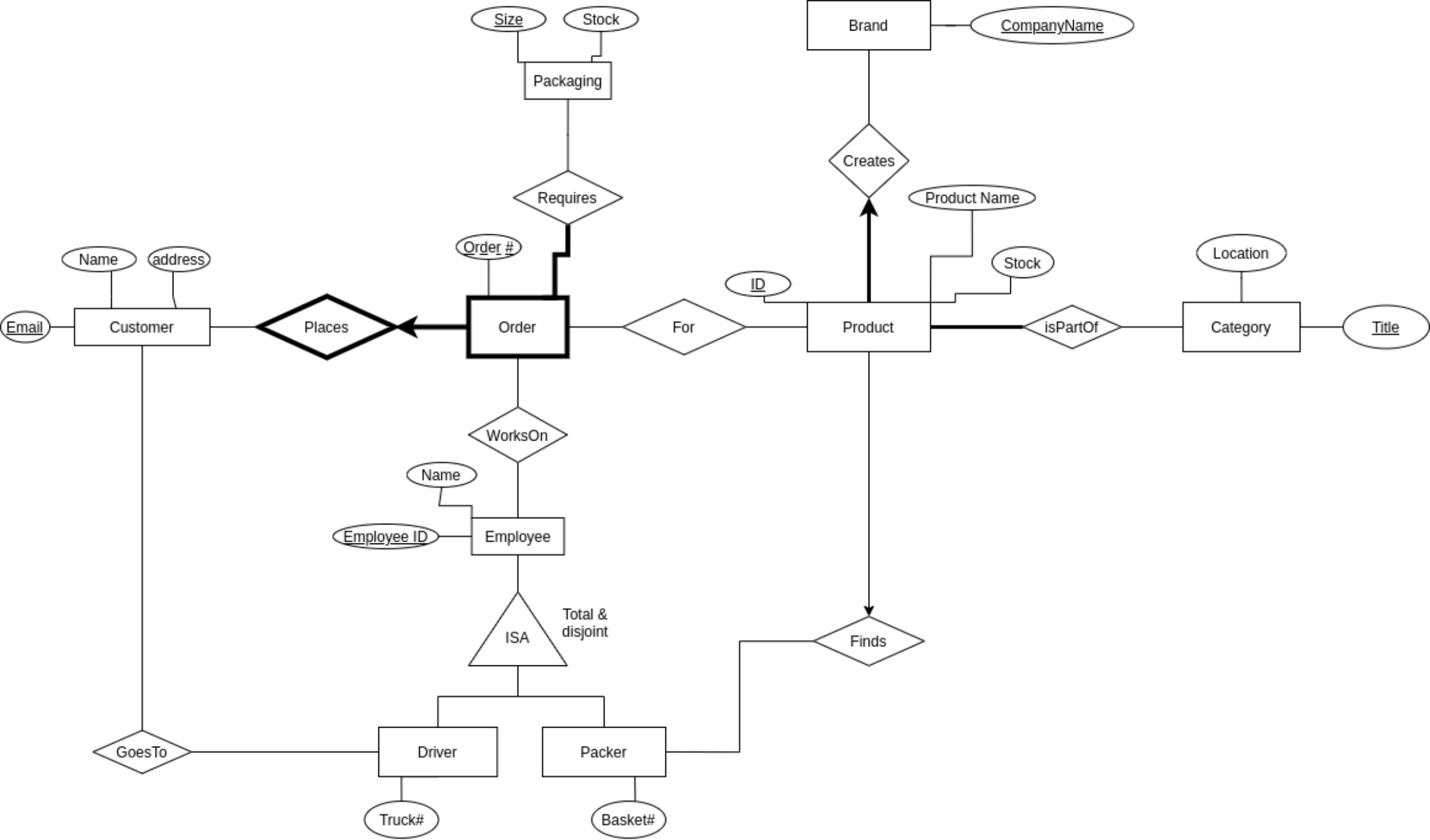
Date: 10/25/2021

Group Number: 39

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Jason Han	94528833	c4g3b	hanjasn@gmail.com
Caeleb Koharjo	18539551	a8a3b	khocaeleb@gmail.com
Claudia Wu	94863289	a3a3b	claudiawu_@hotmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia


Size
Stock
Packaging
Requires
Brand
CompanyName
Creates
Product Name
Stock
Location
Order #
Name
address
ID
Email
Customer
Places
Order
For
Product
isPartOf
Category
Title
WorksOn
Name
Employee ID
Employee
ISA
Total &
disjoint
Finds
GoesTo
Driver
Packer
Truck#
Basket#

CPSC 304 - Milestone 2

1. *A completed cover page.*
2. *The ER diagram you are basing your item #3 (below) on. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.*
    - We changed the underlining of the partial key for the weak entity. The line was originally solid, but is now a dotted line to conform with formatting.
    - Our ISA hierarchy is a total and disjoint relationship, so this was added to the ER diagram. We chose total because our database will only represent employees that are either drivers or packers. We will not consider other employees (such as human resources or maintenance) because they are not directly relevant to our stock database. In our database situations, only drivers and packers handle orders and the corresponding product stock. The relationship is partial so that drivers only do deliveries and packers only process orders.
    - We changed the primary key of category from location to title. Different categories can be in the same location, but they cannot have the same title, otherwise they would be the same category.
    - We removed the "Desc." attribute from "Product" because it does not add useful information to our Database. The other attributes of product describe the product well enough.
    - We change the relationship between "Package" and "Order" from one-to-many to many-to-many. While we originally reasoned a one-to-many relation as meaning each order can only have 1 package, our "package" entity doesn't work like this. Since we want the "package" entity to mean all the packaging boxes in the warehouse, you can have many packages to many orders.

3. *The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:*
    a. *List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...))*
    b. *Specify the primary key, candidate key, foreign keys, and other constraints that the table must maintain.*

Specification:

Underlined = Primary Key
Bold = Foreign key

Customer(Email: Char(20), Name: Char(20), Address: Char(50))

- Candidate Key: (Email)

Employee(EmployeeID: Int, Name: Char(20))

- Candidate Key: (EmployeeID)

Driver(**EmployeeID**: Int, TruckNum: Int)

- Candidate Key: (EmployeeID), (TruckNum)

Packer(**EmployeeID**: Int, BasketNum: Int)

- Candidate Key: (EmployeeID), (BasketNum)

GoesTo(**EmployeeID**: Int, **Email**: Char(20))

- Candidate Key: (EmployeeID, Email)

Place-Order(OrderID: Int, **Email**: Char(20))

- Candidate Key: (OrderID, Email)

Package(Size: Char(10), Stock: Int)

- Candidate Key: (Size)

Requires(**Size**: Char(10), **OrderID**: Int, **Email**: Char(20))

- Candidate key: (Size, OrderID, Email)

Category(Title: Char(20), Location: Char(3))

- Candidate key: (Title)

Product(ProductID: Int, ProductName: Char(80), Stock: Int, **CompanyName**: Char(30) NOT NULL, **EmployeeID**: Int)

- Candidate Key: (ProductID)

Brand(CompanyName: Char(30))

- Candidate key: (CompanyName)

IsPartOf(**ProductID:** Int, **Title**: Char(3))

- Candidate key: (ProductID, Title)

WorksOn(**OrderID**: Int, **Email**: Char(20), **EmployeeID**: Int)

- Candidate key: (OrderID, Email, EmployeeID)

4. *Functional Dependencies (FDs)*
    a. *Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).*

Customer(Email: Char(20), Name: Char(20), Address: Char(50))

- FD:
    - **PK:** Email → Name, Address
    - Name, Address → Email

Employee(EmployeeID: Int, Name: Char(20))

- FD:
    - **PK:** EmployeeID → Name

Driver(**EmployeeID**: Int, TruckNum: Int) *Adding temporary "Meaningful" Attributes for normalization* → (ParkingSpot: Char(3), Birthdate: Char(10), ExperienceLevel: Char(10))

- FD:
    - **PK:** EmployeeID → TruckNum, ParkingSpot, Birthdate, ExperienceLevel
    - **CK:** TruckNum → EmployeeID, Birthdate, ExperienceLevel, ParkingSpot
    - Birthdate, ExperienceLevel → EmployeeID
    - ParkingSpot → TruckNum

Packer(**EmployeeID**: Int, BasketNum: Int)

- FD:
    - **PK:** EmployeeID → BasketNum
    - **CK:** BasketNum → EmployeeID

GoesTo(**EmployeeID**: Int, **Email**: Char(20))

- FD:
    - **PK:** EmployeeID, Email → Email, EmployeeID

Place-Order(OrderID: Int, **Email**: Char(20))

- FD:
    - **PK:** (Trivial) OrderID, Email → OrderID, Email

Package(Size: Char(10), Stock: Int)

- FD:
    - **PK:** Size → Stock

Requires(**Size**: Char(10), **OrderID**: Int, **Email**: Char(20))

- FD:
    - **PK:** (Trivial) Size, OrderID, Email → Size, OrderID, Email

Category(Title: Char(20), Location: Char(3))

- FD:
    - **PK:** Title → Location

Product(ProductID: Int, ProductName: Char(80), Stock: Int, **CompanyName**: Char(30) NOT NULL, **EmployeeID**: Int)

- FD:
    - **PK:** ProductID → ProductName, Stock, CompanyName, EmployeeID
    - EmployeeID → ProductID
    - ProductName, CompanyName → Stock

Brand(CompanyName: Char(30))

- FD:
    - **PK:** (Trivial) CompanyName -> CompanyName

IsPartOf(**ProductID:** Int, **Title**: Char(3))

- FD:
    - **PK:** (Trivial) ProductID, Title → ProductID, Title

WorksOn(**OrderID**: Int, **Email**: Char(20), **EmployeeID**: Int)

- FD:
    - **PK:** (Trivial) OrderID, Email, EmployeeID → OrderID, Email, EmployeeID

5. *Normalization*
    a. *Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, …). ALL Tables must be listed, not only the ones post normalization. You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown.*

Customer(Email: Char(20), Name: Char(20), Address: Char(50))

- FD:
    - **PK:** Email → Name, Address
    - Name, Address → Email

Employee(EmployeeID: Int, Name: Char(20))

- FD:
    - **PK:** EmployeeID → Name

Driver(**EmployeeID**: Int, TruckNum: Int) *Adding temporary "Meaningful" Attributes for normalization* → (ParkingSpot: Char(3), Birthdate: Char(10), ExperienceLevel: Char(10))

- FD:
    - **PK:** EmployeeID → TruckNum, ParkingSpot, Birthdate, ExperienceLevel
    - **CK:** TruckNum → EmployeeID, Birthdate, ExperienceLevel, ParkingSpot
    - Birthdate, ExperienceLevel → EmployeeID
    - ParkingSpot → TruckNum
- Decomposition:
    - (Birthdate, ExperienceLevel)+ = {Birthdate, ExperienceLevel, EmployeeID}
    - (ParkingSpot) += {TruckNum, ParkingSpot}

- For the non-trivial FD, (Birthdate, ExperienceLevel) → EmployeeID, (Birthdate, ExperienceLevel) is not a superkey of R (Violates BCNF) so we decompose into:
  - R_1(ExperienceLevel, Birthdate, EmployeeID) and R_2(ParkingSpot, TruckNumber, Birthdate, ExperienceLevel).
- R_1 is in BCNF because there are no remaining functional dependencies that violate BCNF. R_2 violates BCNF because for the remaining non-trivial FD, (ParkingSpot) → TruckNum, (ParkingSpot) is not a superkey for R_2, violating BCNF. Therefore we decompose R_2 into:
  - R_3(ParkingSpot, TruckNum) and R_4(ParkingSpot, Birthdate, ExperienceLevel).
- R_3 is in BCNF because (ParkingSpot) is a superkey of R_3 and there are no more remaining FDs. R_4 is also in BCNF because there are no more remaining functional dependencies.
- The final decomposition is R_1(ExperienceLevel, Birthdate, EmployeeID), R_3(ParkingSpot, TruckNum), and R_4(ParkingSpot, Birthdate, ExperienceLevel).
- The new tables(relations) are:
  - EmployeeParking(Birthdate: Char(10), ParkingSpot: Char(3), ExperienceLevel: Char(10))
  - Truck(ParkingSpot: Char(3), TruckNumber: Int)
  - Driver(Birthdate: Char(10), ExperienceLevel: Char(10), **EmployeeID**: Int)

Packer(**EmployeeID**: Int, BasketNum: Int)

- FD:
  - **PK:** EmployeeID → BasketNum
  - **CK:** BasketNum → EmployeeID

GoesTo(**EmployeeID**: Int, **Email**: Char(20))

- FD:
  - **PK:** EmployeeID, Email → Email, EmployeeID

Place-Order(OrderID: Int, **Email**: Char(20))

- FD:
  - **PK:** (Trivial) OrderID, Email → OrderID, Email

Package(Size: Char(10), Stock: Int)

- FD:

- **PK:** Size → Stock

Requires(**Size**: Char(10), **OrderID**: Int, **Email**: Char(20))

- FD:
  - **PK:** (Trivial) Size, OrderID, Email → Size, OrderID, Email

Category(Title: Char(20), Location: Char(3))

- FD:
  - **PK:** Title → Location

Product(ProductID: Int, ProductName: Char(80), Stock: Int, **CompanyName**: Char(30) NOT NULL, **EmployeeID**: Int)

- FD:
  - **PK:** ProductID → ProductName, Stock, CompanyName, EmployeeID
  - EmployeeID → ProductID
  - ProductName, CompanyName → Stock
- Decomposition:
  - EmployeeID+ = {EmployeeID, ProductID}
  - (ProductName, CompanyName)+ = {ProductName, CompanyName, Stock}
  - Using the non-trivial FD, EmployeeID → ProductID, EmployeeID is not a superkey (Violates BCNF) of R so we decompose:
    - R_1(EID, PID) and R_2(EID, PName, CName, Stock).
  - R_1 is in BCNF since there are only 2 attributes. R_2 is not in BCNF because for the remaining non-trivial FD, (PName, CName) → Stock, (PName, CName) is not a superkey of R_2. So we decompose on R_2 to get:
    - R_3(PName, CName, Stock) & R_4(PName, CName, EID).
  - Since there are no more non-trivial functional dependencies for either R_3 or R_4, they are in BCNF.
  - The final decomposition is R_1(EID, PID); R_3(PName, CName, Stock); R_4(PName, CName, EID). Therefore the new tables (Relations) are:
    - Employee-Product(**EmployeeID**: Int, ProductID: Int)
    - Product-Stock(ProductName: Char(80), Stock: Int, **CompanyName**: Char(30) NOT NULL)
    - Product(ProductName: Char(80), **CompanyName**: Char(30) NOT NULL, EmployeeID: Int)

Brand(CompanyName: Char(30))

- FD:
    - **PK:** (Trivial) CompanyName -> CompanyName

IsPartOf(**ProductID:** Int, **Title**: Char(3))

- FD:
    - **PK:** (Trivial) ProductID, Title → ProductID, Title

WorksOn(**OrderID**: Int, **Email**: Char(20), **EmployeeID**: Int)

- FD:
    - **PK:** (Trivial) OrderID, Email, EmployeeID → OrderID, Email, EmployeeID

6. *The SQL DDL to create all the tables in SQL. All primary keys and foreign keys must be declared appropriately. Code the SQL CREATE TABLE statements with the appropriate foreign keys, primary keys, UNIQUE constraints, etc.*

Customer(Email: Char(20), Name Char(20), Address Char(50))

```
CREATE TABLE Customer(
        email CHAR(20),
        name CHAR(20),
        address CHAR(50),
        PRIMARY KEY email);
```

Employee(EmployeeID: Int, Name: Char(20))

```
CREATE TABLE Employee(
        employeeID Int,
        name CHAR(20),
        PRIMARY KEY(employeeID));
```

EmployeeParking(Birthdate: Char(10), ParkingSpot: Char(3), ExperienceLevel: Char(10))

```
CREATE TABLE EmployeeParking (
        birthdate CHAR(10),
        parkingSpot CHAR(3),
        experienceLevel CHAR(10),
        PRIMARY KEY (birthDate, parkingSpot, experienceLevel));
```

Truck(ParkingSpot: Char(3), TruckNumber: Int)

```
CREATE TABLE Truck (
        parkingSpot CHAR(3),
        truckNumber INTEGER,
        PRIMARY KEY (parkingSpot));
```

Driver(Birthdate: Char(10), ExperienceLevel: Char(10), **EmployeeID**: Int)

```
CREATE TABLE Driver (
        birthdate CHAR(10),
        experienceLevel CHAR(10),
        employeeID Integer,
        FOREIGN KEY (employeeID) REFERENCES Employee,
        PRIMARY KEY (birthdate, experienceLevel));
```

Packer(**EmployeeID**: Int, BasketNum: Int)

```
CREATE TABLE Packer
        (employeeID INTEGER,
         basketNum INTEGER,
         FOREIGN KEY (employeeID) REFERENCES Employee,
         PRIMARY KEY employeeID);

CREATE TABLE GoesTo
        (employeeID INTEGER,
         email CHAR(20),
         PRIMARY KEY (employeeID, email),
         FOREIGN KEY (employeeID) REFERENCES Driver,
         FOREIGN KEY (email) REFERENCES Customer);

CREATE TABLE Place-Order
        (orderID INTEGER,
         email CHAR(20),
         PRIMARY KEY (orderID, email),
         FOREIGN KEY (email) REFERENCES Customer,
```

ON DELETE CASCADE);

CREATE TABLE Package
(size CHAR(10),
stock INTEGER,
PRIMARY KEY size);

CREATE TABLE Requires
(size CHAR(10),
orderID INTEGER,
email CHAR(20),
PRIMARY KEY (size, orderID, email),
FOREIGN KEY (size) REFERENCES Package,
FOREIGN KEY (orderID) REFERENCES Order,
FOREIGN KEY (email) REFERENCES Customer);

CREATE TABLE Category
(title CHAR(20),
location CHAR(3),
PRIMARY KEY title);

Employee-Product(**EmployeeID**: Int, ProductID: Int)
CREATE TABLE Employee-Product
(employeeID INTEGER,
productID INTEGER,
PRIMARY KEY employeeID,
FOREIGN KEY (employeeID) REFERENCES Packer);

Product-Stock(ProductName: Char(80), Stock: Int, **CompanyName**: Char(30))
CREATE TABLE Product-Stock
(productName CHAR(80),
stock INTEGER,
companyName CHAR(30) NOT NULL,

PRIMARY KEY (productName, companyName),
FOREIGN KEY (companyName) REFERENCES Brand);

Product(ProductName: Char(80), **CompanyName**: Char(30), **EmployeeID**: Int)

```
CREATE TABLE Product
        (productName CHAR(80),
         companyName CHAR(30) NOT NULL,
         employeeID INTEGER,
         PRIMARY KEY (productName, companyName, employeeID),
         FOREIGN KEY (employeeID) REFERENCES Packer,
         FOREIGN KEY (companyName) REFERENCES Brand);
```

Brand(CompanyName: Char(30))

```
CREATE TABLE Brand (
        companyName CHAR(30) PRIMARY KEY);
```

IsPartOf(**ProductID:** Int, **Title**: Char(3))

```
CREATE TABLE IsPartOf (
        productID INTEGER,
        title CHAR(3),
        PRIMARY KEY (productID, title),
        FOREIGN KEY (productID) REFERENCES Employee-Product,
        FOREIGN KEY (title) REFERENCES Category);
```

WorksOn(**OrderID**: Int, **Email**: Char(20), **EmployeeID**: Int)

```
CREATE TABLE WorksOn (
        orderID INTEGER,
        email CHAR(20),
        employeeID INTEGER,
        PRIMARY KEY (orderID, email, employeeID),
        FOREIGN KEY (orderID) REFERENCES Order,
        FOREIGN KEY (email) REFERENCES Customer,
        FOREIGN KEY (employeeID) REFERENCES Employee);
```

7. *Populate each table with at least 5 tuples, and probably more so that you can issue meaningful Group By queries later on. Show the instance of each relation after inserting the tuples. We recommend writing INSERT statements and submitting a .SQL file. However, screenshots of an excel tabular like format are accepted as well.*

- Spreadsheet link: https://docs.google.com/spreadsheets/d/1mKtk47Uanz_hpQDCwnwqxmiAgnpCcED8Hwn9D803iwY/edit?usp=sharing

8. *As much as possible, write a list of all the variety of queries that are proposed for your application, in plain English. For example, "Insertion: Add a customer to the supermarket membership list." There may be some concepts that we have not yet covered by the time that the milestone is due. That's okay. For example, if we have not covered projection by a week before the milestone is due, just look at it to get an idea of what is coming and write either something to the best of your ability or say "we have not covered projection yet, so we cannot answer this question."*
    a. *See milestone 5 of this document to see a list of all the types of required queries.*
        i. *You only need to list the queries you plan to use to fulfill the requirements in milestone 5 and not every query for the application*
    b. *Note that you cannot "double dip" on a query (i.e., the same query cannot be used as an answer for multiple categories).*
    c. *Note that our goal for this is to make sure that your schema is complex enough to ask the required questions over. You can certainly change them later*

Queries:

- **Insertion:** Add the customer, Jonathan Looney with email jlooney@gmail.com that lives at 9999 looneytunes avenue v6o 2x3 BC Vancouver, to the customers list (Customer table).
- **Update:** Update the stock of the product (Product-stock table), monkey pants by the company Amazon jungle, from 28 to 58 because the company has restocked the product.
- **Delete:** Delete the truck (Truck table) that has truck number 1 at parking spot A10 because the truck is now totaled.
- **Selection:** Select the columns *productName* and *companyName* from the *Product* table.
- **Selection-Where:** Select all columns from the *Category* table where *title*="Electronics" or *title*="Health and Beauty".