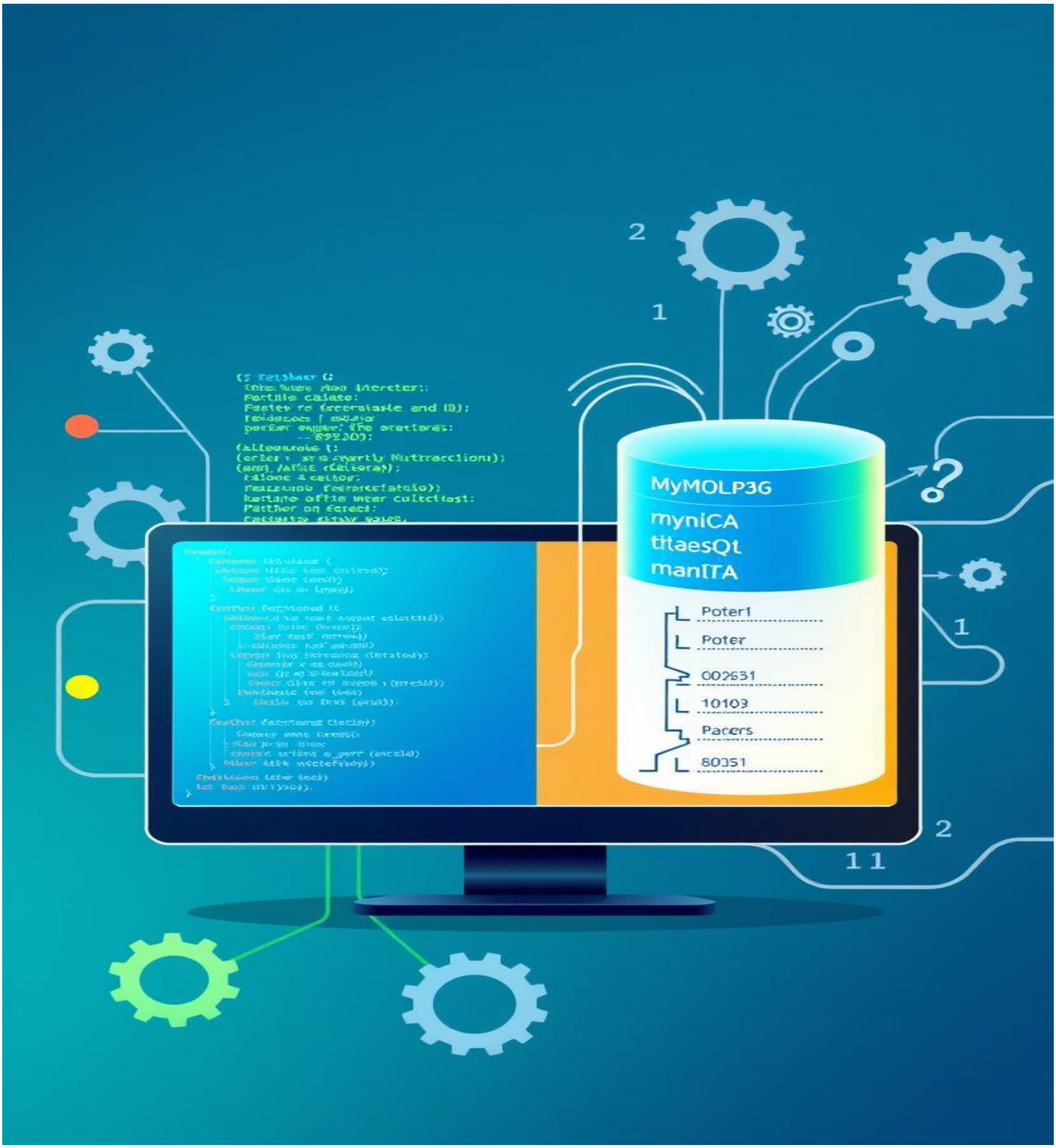
COP 5725: DATABASE SYSTEMS PROJECT



GROUP MEMBERS

Anasah Wawem Christopher

Patrick Ansah

Akhil Bommareddy

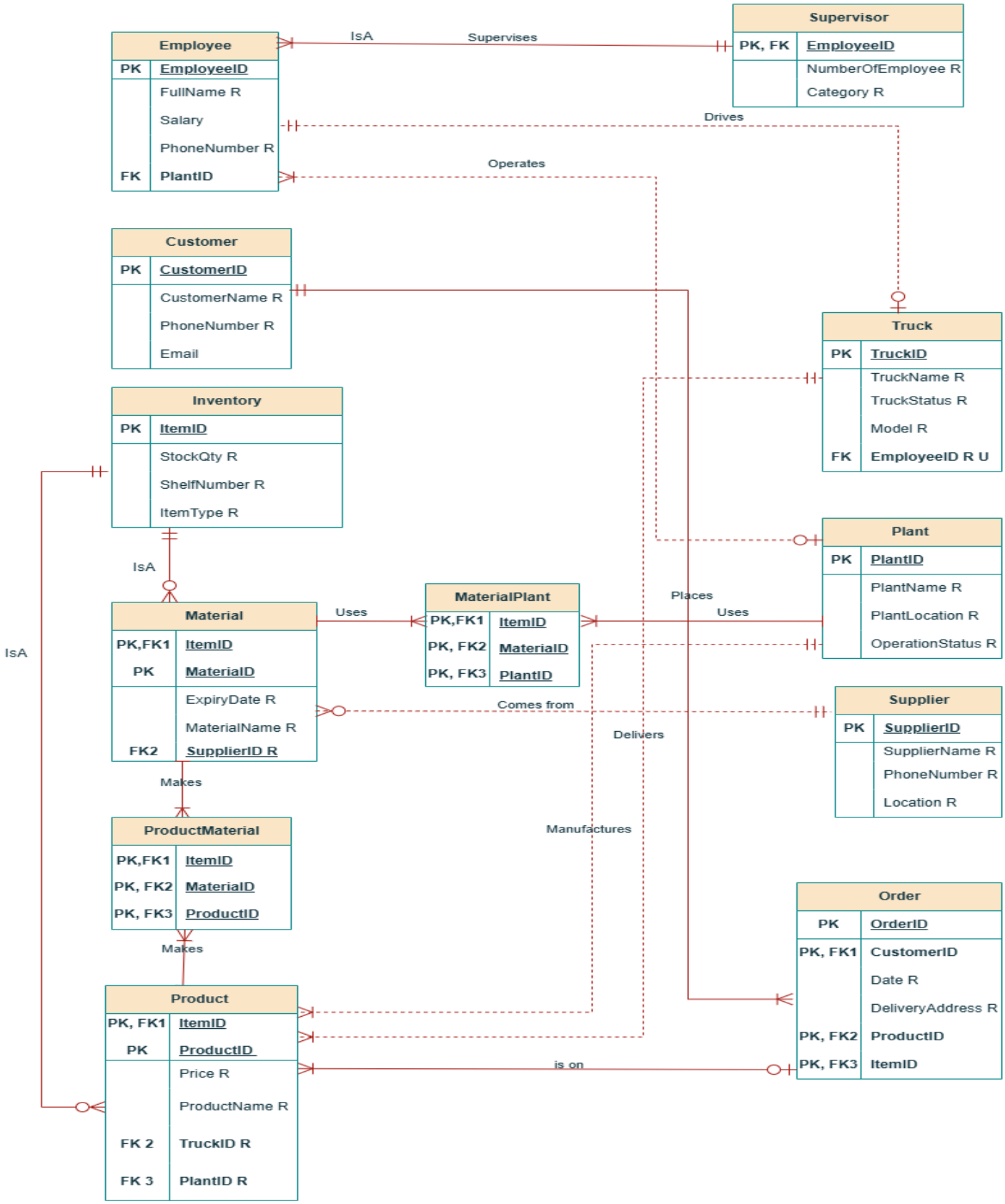
Keithrelle Ferguson

Table of Contents

INTRODUCTION

This project was developed for COP 5725: Database Systems to demonstrate the full lifecycle of database design and implementation. Our team created a relational database for a manufacturing and logistics company, covering key processes such as employee management, inventory tracking, product delivery, and customer orders. We began by identifying entities and relationships, then designed an EER diagram with cardinalities, strong and weak entities, and supertype-subtype structures. After defining constraints and ensuring normalization to 3NF, we implemented the system in MySQL, populated it with data, and wrote complex queries to extract meaningful insights. The project concludes with a reverse-engineered ER diagram, confirming consistency between design and implementation.

PHASE 1 AND 2


Employee
PK EmployeeID
FullName R
Salary
PhoneNumber R
FK PlantID
IsA
Supervises
Supervisor
PK, FK EmployeeID
NumberOfEmployee R
Category R
Drives
Operates
Customer
PK CustomerID
CustomerName R
PhoneNumber R
Email
Truck
PK TruckID
TruckName R
TruckStatus R
Model R
FK EmployeeID R U
Inventory
PK ItemID
StockQty R
ShelfNumber R
ItemType R
IsA
Plant
PK PlantID
PlantName R
PlantLocation R
OperationStatus R
Material
PK,FK1 ItemID
PK MaterialID
ExpiryDate R
MaterialName R
FK2 SupplierID R
Uses
MaterialPlant
PK,FK1 ItemID
PK, FK2 MaterialID
PK, FK3 PlantID
Places
Uses
IsA
Comes from
Supplier
PK SupplierID
SupplierName R
PhoneNumber R
Location R
Delivers
Makes
ProductMaterial
PK,FK1 ItemID
PK, FK2 MaterialID
PK, FK3 ProductID
Manufactures
Makes
Order
PK OrderID
PK, FK1 CustomerID
Date R
DeliveryAddress R
PK, FK2 ProductID
PK, FK3 ItemID
Product
PK, FK1 ItemID
PK ProductID
Price R
ProductName R
FK 2 TruckID R
FK 3 PlantID R
is on

PHASE 3

SPECIFICATION OF THE EMPLOYEE ENTITY.

The database keeps and manages data of our manufacturing and distribution company. The database has 12 useful entities. One of the entities has the following specifications.

The **Employee** entity is related to **Supervisor** entity, the **Truck** entity and **Plant** entity. Some employees drive our company truck, some also operates the plants for manufacturing our products. Since supervisors are also employees, it becomes the subtype of the employee(supertype) since all the supervisors are also employees in our company. The employee entity has the following attributes:

EmployeeID: This is unique integers that identifies our employees hence becomes the Primary Key of the table.

FullName: This is a required attribute that tells the employees' first and last name.

Salary: This is where our employees' salaries can be found and is optional.

PhoneNumber: This is a required attribute about our employees' phone number. Since we will need to contact our employee, they must provide it.

PlantID: Since we will want to know which plant an operator operates, calls for this attribute. This is a foreign key from plant table referencing PlantID.

The employee entity has (one mandatory to mandatory many) relation with supervisor entity which also a subtype entity of the employee entity. The employee entity again has (one mandatory to optional one) relationship with truck entity. It also has (many mandatory to optional one) with plant entity.

CARDINALITY AND PARTICIPATION RATIO

Cardinality Table

Entity	Cardinality	Description
Employee and Supervisor	Many to One (N:1)	Many employees are managed by one supervisor
Employee and Truck	One to One (1:1)	One employee drives only one truck
Employee and Plant	Many to One (N:1)	Many employees can work or operate one plant

Customer and Order	One to Many (1: N)	One customer can make a lot of orders
Inventory and Material	One to Many (1: N)	One inventory number can consist of many materials
Inventory and Product	One to Many (1: N)	One inventory number can consist of many products
Material and Product	Many to Many (M: N)	Many materials can be used to produce different products.
Product and Order	Many to One (N:1)	One order can consist of many products
Product and Truck	Many to One (N:1)	One truck can deliver many products
Product and Plant	Many to One (N:1)	Many products can be produced from one plant
Material and Plant	Many to Many (M: N)	Many materials can be used by many plants for manufacturing or production
Material and Supplier	Many to One (N:1)	A single supplier can supplier many and different materials.

Participation Ratio Table

Entity	Participation Ratio	Description
Employee and Supervisor	Total/Mandatory (1) Total/ Mandatory (1)	One supervisor must at least manage one employee and an employee must be under only one supervision.
Employee and Truck	Total/Mandatory (1) Partial/Optional (0)	Each truck is assigned to only one employee and not all employees drives truck.
Employee and Plant	Total/ Mandatory (1) Partial/Optional (0)	A plant needs at least an employee for production and not all employees work on plant.
Customer and Order	Total/ Mandatory (1) Total/Mandatory (1)	There is no sale without order and an order can be assigned to only one customer.
Inventory and Material	Total/Mandatory (1) Partial/Optional (0)	Each material is classified under only one inventory. There could be an inventory classification without a material.
Inventory and Product	Total/Mandatory (1) Partial/Optional (0)	Each product is classified under only one inventory. There could be an inventory classification without a product.

Material and Product	Total/ Mandatory (1) Total/ Mandatory (1)	Every product is made of at least one material and if there a product, it is definitely made from a material.
Product and Order	Total/ Mandatory (1) Partial/Optional (0)	There can be a product without order and an order is traced to at least one product.
Product and Truck	Total/ Mandatory (1) Total/ Mandatory (1)	A truck delivers at least one product and a delivered product is traced to only one truck.
Product and Plant	Total/ Mandatory (1) Total/ Mandatory (1)	A plant produces at least one product and a product is traced to only one plant.
Material and Plant	Total/ Mandatory (1) Total/ Mandatory (1)	A plant must be provided with at least one material for production.
Material and Supplier	Partial/Optional (0) Total/ Mandatory (1)	A supplier may or may not supply us material and a material in stock is traced to exactly one supplier.

STRONG AND WEAK ENTITIES

There is an identifying relationship between customer and order likewise product and order. Per our business rule, an order cannot be placed when there is no customer. Similarly, when there is no product there will not be an order, therefore the **Order** entity is identified by **Customer** and **Product.** The primary keys of the customer and product tables are also primary keys in the order table as well as foreign keys.

SUPERTYPE AND SUBTYPE

Supervisors in our company are employees which makes **Supervisor** entity a subtype of **Employee** entity (supertype). The primary key of the employee table becomes the primary key of the supervisor table as well as foreign key.

Products and materials are classified as inventory hence the **Product** entity and **Material** entity are subtype of the **Inventory** entity (supertype). The primary key of the inventory table becomes the primary key in the product and material tables as well as foreign keys in same tables.

TABLE AND COLUMNS CONSTRAINTS

Employee Table

Table Constraint: **Primary Key** (EmployeeID)

Foreign Key (PlantID) **[On Delete Set Null On Update Set Null]**

Column Constraint: **Not Null** (FullName, PhoneNumber)

Check (PhoneNumber)

Supervisor Table

Table Constraint**: Primary Key** (EmployeeID)

Foreign Key (EmployeeID) **[On Delete Cascade On Update Cascade]**

Column Constraint: **Not Null** (Category, NumberOfEmployee)

Truck Table

Table Constraint: **Primary Key** (TruckID)

Foreign Key (EmployeeID) **[On Delete Cascade On Update Cascade]**

Column Constraint: **Not Null** (Category, NumberOfEmployee, EmployeeID)

Unique (EmployeeID)

Plant Table

Table Constraint**: Primary Key** (PlantID)

Column Constraint: **Not Null** (PlantName, PlantLocation, OperationStatus)

Customer Table

Table Constraint**: Primary Key** (CustomerID)

Column Constraint: **Not Null** (CustomerName, PhoneNumber)

Check (PhoneNumber)

Order Table

Table Constraint**: Primary Key** (OrderID, CustomerID, ProductID, ItemID)

Foreign Key (CustomerID, ProductID, ItemID) **[On Delete Cascade On Update Cascade]**

Column Constraint: **Not Null** (Date, DeliveryAddress)

Inventory Table

Table Constraint**: Primary Key** (ItemID)

Column Constraint: **Not Null** (StockQty, ShelfNumber, ItemType)

Material Table

Table Constraint**: Primary Key** (ItemID, MaterialID)

Foreign Key (ItemID, SupplierID) **[On Delete Cascade On Update Cascade]**

Column Constraint: **Not Null** (ExpiryDate, MaterialName, SupplierID)

Product Table

Table Constraint**: Primary Key** (ProductID, ItemID)

Foreign Key (TruckID, PlantID) **[On Delete Restrict On Update Restrict]**

Foreign Key (ItemID) **[On Delete Cascade On Update Cascade]**

Column Constraint: **Not Null** (Price, ProductName, TruckID, PlantID)

Supplier Table

Table Constraint**: Primary Key** (SupplierID)

Column Constraint: **Not Null** (SupplierName, PhoneNumber, Location)

Check (PhoneNumber)

ProductMaterial Table

Table Constraint**: Primary Key** (ItemID, MaterialID, ProductID)

Foreign Key (ItemID, MaterialID, ProductID)) **[On Delete Restrict On Update Restrict]**

MaterialPlant Plant

Table Constraint**: Primary Key** (ItemID, MaterialID, PlantID)

Foreign Key (ItemID, MaterialID, PlantID) **[On Delete Restrict On Update Restrict]**

NORMALIZATION

All the tables are in 3NF. This was easily known because there were not many values inserted.

PHASE 4

IMPLEMENTATION OF ER DIAGRAM IN MYSQL

CREATE DATABASE DBMS_PROJECT;

SHOW DATABASES;

USE DBMS_PROJECT;

Plant Table

Table Creation

```
CREATE TABLE Plant (
    PlantID SMALLINT UNSIGNED PRIMARY KEY,
    PlantName VARCHAR (50) NOT NULL,
    PlantLocation VARCHAR (50) NOT NULL,
    OperationStatus VARCHAR (50) NOT NULL
);
```

Table Description

This table stores basic information about individual plants, such as their name, location, and current operational status.

PlantID: A small unsigned integer that uniquely identifies a plant. This is the **primary key** of the table, meaning each PlantID must be unique and non-null. As it's UNSIGNED, it can only store positive values (0 to 65,535).

PlantName: A variable-length string (up to 50 characters) representing the name of the plant. This field is marked NOT NULL, so every plant must have a name.

PlantLocation: A variable-length string (up to 50 characters) that indicates the physical or geographical location of the plant. This is also a NOT NULL field, meaning each plant must have a known location.

OperationStatus: A variable-length string (up to 50 characters) indicating the current operational state of the plant. This field is also NOT NULL.

The table does **not** currently include any foreign keys, but it could be linked to other tables (like company regions or statuses) depending on how the broader database is designed. There are no duplicate rules or relational constraints in this version beyond the primary key on PlantID.

Values Insertion

```
INSERT INTO Plant
VALUES (100, "Sagali", "Texas", "Mining"), (200, "Tema", "Atlanta", "Refinery"),
(300, "Mashda", "New York", "Cannary"), (400, "Segal", "Washington", "Mixer"),
(500, "Pinto", "Pensacola", "Grinder");
```

Customer Table

Table Creation

```
CREATE TABLE Customer (
        CustomerID SMALLINT UNSIGNED PRIMARY KEY,
        CustomerName VARCHAR (50) NOT NULL,
        PhoneNumber VARCHAR (10) NOT NULL,
   CHECK (
      LENGTH(PhoneNumber) = 10 AND
      PhoneNumber LIKE '0%' AND
      PhoneNumber NOT LIKE '%[^0-9]%'  ),
      Email VARCHAR (50)
);
```

Table Description

This table stores information about individual customers, including contact details such as phone numbers and email addresses.

CustomerID: A small unsigned integer that uniquely identifies each customer. It serves as the **primary key**, ensuring each customer record is unique. Since it's UNSIGNED, values must be positive (0 to 65,535).

CustomerName: A variable-length string (up to 50 characters) representing the full name of the customer. It is marked as NOT NULL, meaning the name is required for every customer.

PhoneNumber: A fixed-length string of exactly 10 digits representing the customer's phone number. The field has a CHECK constraint to ensure:

- The phone number is exactly **10 characters** long,
- It **starts with '0'**,
- It contains **only numeric digits** (no letters or symbols). This helps enforce a standardized phone number format.

Email: A variable-length string (up to 50 characters) representing the customer's email address. This field is **optional** (not marked NOT NULL), so a customer may or may not have an email on file.

This table includes basic data validation using a CHECK constraint on the PhoneNumber field. There are no foreign keys or relationships to other tables in this version, but it could be extended to connect with order histories, accounts, or customer types if needed.

Values Insertion

INSERT INTO Customer
VALUES
(221, "Ratish Prakash", "0245678610", "ratish@gmail.com"),
(222, "Raman Krishna", "0255678610", "raman@gmail.com"),
(223, "Keeks Lady", "0245670610", "keeks@gmail.com"),
(224, "Christopher Anasah", "0545678610", "chris@gmail.com"),
(225, "Felix Addae", "0245678230", "felix@gmail.com");

Employee Table

Table Creation

CREATE TABLE Employee (
EmployeeID SMALLINT UNSIGNED PRIMARY KEY,
FullName VARCHAR (50) NOT NULL,
Salary DECIMAL (10,2),
PlantID SMALLINT UNSIGNED,

```
    PhoneNumber VARCHAR (10) NOT NULL,
    CHECK (
        LENGTH(PhoneNumber) = 10 AND
        PhoneNumber LIKE '0%' AND
        PhoneNumber NOT LIKE '%[^0-9]%' ),
FOREIGN KEY (PlantID) REFERENCES Plant (PlantID) ON DELETE SET NULL ON UPDATE SET
NULL
);
```

Table Description

This table stores information about employees, including their personal details, contact information, salary, and work location (linked to a plant).

EmployeeID: A small unsigned integer that uniquely identifies each employee. This is the **primary key**, so values must be unique and non-null. Being UNSIGNED, only positive values (0 to 65,535) are allowed.

FullName: A variable-length string (up to 50 characters) representing the employee's full name. This field is marked NOT NULL, so a name is required for every employee.

Salary: A decimal number with up to 10 digits in total, including 2 digits after the decimal point (e.g., 50000.00). This stores the employee's salary. It is **optional** (nullable), so not all employees must have a salary value defined.

PlantID: A small unsigned integer representing the plant where the employee is assigned. This is a **foreign key** that references the Plant table's PlantID.

- **ON DELETE SET NULL**: If the referenced plant is deleted, the PlantID in the Employee record will be set to NULL (preserving the employee record without breaking the reference).
- **ON UPDATE SET NULL**: If the referenced PlantID changes in the Plant table, it will also be set to NULL here to avoid inconsistency.

PhoneNumber: A string of exactly 10 digits representing the employee's phone number. It has a CHECK constraint that enforces:

- Must be **10 characters** long,
- Must **start with '0'**,
- Must contain **only numeric digits** (no letters or symbols).

This table links employees to their respective plants while enforcing standardized phone number formatting. Nullable PlantID allows flexibility for employees who might not be assigned to a specific plant yet.

Values Insertion

INSERT INTO Employe**e**
VALUES
(111, "Patrick Ansah", 2003, 100, '0245678678'), (112, "Felicia Amana", 4506, 200, '0255678678'),
(113, "Davis Asoma", 4587, 100, '0245678678'), (114, "Gideon Bright", 7892, 400, '0245678678'),
(115, "Jeff Abban", 6902.34, 300, '0555678678'), (116, "John Andai", 5670.43, 500, '0675678678'),
(117, "Wendy Eshun", 6792.33, 400, '0255678600'), (118, "Cecilia Ackon", 5612.88, 300, '0245678654')**;**

Supervisor Table

Table Creation

CREATE TABLE Supervisor (
EmployeeID SMALLINT UNSIGNED PRIMARY KEY,
NumberOfEmployee INT NOT NULL,
Category VARCHAR (50) NOT NULL,
FOREIGN KEY (EmployeeID) REFERENCES Employee (EmployeeID) ON DELETE CASCADE ON UPDATE CASCADE
);

Table Description

The statement creates a table named **Supervisor**, which is used to store information about employees who hold supervisory roles. Each supervisor is identified by a unique **EmployeeID**, which also serves as the **primary key** for the table. This EmployeeID is a foreign key that references the EmployeeID in the Employee table, ensuring that only existing employees can be recorded as supervisors.

Additionally, the table includes a column called **NumberOfEmployee**, which stores the number of employees a supervisor oversees. This field is mandatory and cannot be left empty. Another required field is **Category**, which specifies the type or category of the supervisor, such as "Technical", "Administrative", or "Managerial".

The table also defines two important constraints: **ON DELETE CASCADE** and **ON UPDATE CASCADE**. This means that if an employee is deleted from the Employee table, the corresponding supervisor record will also be deleted. Similarly, if an employee's ID is updated, that update will be automatically reflected in the Supervisor table.

Values Insertion

INSERT INTO Supervisor
VALUES (111, 40, "Senior"), (112, 60, "Supretendent"), (114, 34, "Junior"), (116, 78, "Senior"), (118, 30, "Officer");

Truck Table

Table Creation

```
CREATE TABLE Truck (
    TruckID SMALLINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    TruckName VARCHAR (50) NOT NULL,
    TruckStatus VARCHAR (50) NOT NULL,
    Model VARCHAR (50) NOT NULL,
    EmployeeID SMALLINT UNSIGNED UNIQUE NOT NULL,
    FOREIGN KEY (EmployeeID) REFERENCES Employee (EmployeeID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table Description

The statement creates a table named **Truck**, which is designed to store details about trucks used in an organization.

Each truck has a unique identifier called **TruckID**, which is the **primary key** of the table. It is an **auto-incrementing** small integer, meaning each time a new truck is added, this ID is automatically generated.

The table also stores the following required information:

TruckName: the name or label of the truck.

TruckStatus: the current condition or availability status of the truck (e.g., "Active", "Under Maintenance").

Model: the model or type of the truck.

EmployeeID: the unique ID of the employee who is assigned to or drives the truck.

The **EmployeeID** column must be unique and cannot be null, meaning that each truck is assigned to exactly one employee, and no two trucks can be assigned to the same employee.

Additionally, **EmployeeID** is defined as a **foreign key** that references the **EmployeeID** in the Employee table. This ensures that only existing employees can be associated with a truck. The **ON DELETE CASCADE** and **ON UPDATE CASCADE** constraints ensure that if the associated employee is deleted or their ID is changed in the Employee table, those changes will automatically reflect in the Truck table.

Values Insertion

INSERT INTO Truck (TruckName,TruckStatus,Model,EmployeeID)
VALUES ("Toyota", "Active", "Corrolla", 111), ("Benz", "Active", "A-300", 118),
("Benz", "Active", "Sprinter", 112), ("Ram", "Repair", "C-Class", 116),
("Pigeout", "Faulty", "Apotro", 114);

Inventory Table

Table Creation

CREATE TABLE Inventory (
    ItemID SMALLINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    StockQty INT NOT NULL,
    ShelfNumber CHAR (3) NOT NULL,
    ItemType VARCHAR (50) NOT NULL
);

Table Description

The CREATE TABLE Inventory statement defines a table called **Inventory**, which is used to store information about items held in stock within a storage or warehouse system.

The table has a **primary key** called **ItemID**, which uniquely identifies each item in the inventory. It is a **small unsigned integer** and is **auto-incremented**, meaning its value will automatically increase for each new record added.

StockQty represents the **quantity of the item currently in stock**. It is an integer and must not be null, ensuring that every inventory record has a quantity value.

ShelfNumber is a **3-character field** that indicates the **specific shelf or location** where the item is stored. For example, it could be something like "A12" or "B07".

ItemType is a **text field (up to 50 characters)** describing the **type or category of the item** (e.g., "Tool", "Material", "Equipment"), and it must not be null.

This table is foundational for tracking stock levels and item locations in a structured inventory management system.

Values Insertion

INSERT INTO Inventory (StockQty,ShelfNumber,ItemType)

VALUES (102, "AAA", "Raw Materials"), (349, "AAC", "Detergent"), (40, "BCD", "Finished Products"), (600, "ZSA", "Plant Parts"), (20, "AXC", "Car Parts");

Supplier Table

Table Creation

```
CREATE TABLE Supplier (
  SupplierID SMALLINT UNSIGNED PRIMARY KEY,
   SupplierName VARCHAR (50) NOT NULL,
   PhoneNumber VARCHAR (10) NOT NULL,
    CHECK (
       LENGTH(PhoneNumber) = 10 AND
       PhoneNumber LIKE '0%' AND
       PhoneNumber NOT LIKE '%[^0-9]%' ),
   Location VARCHAR (50) NOT NULL
);
```

Table Description

The statement creates a table called **Supplier**, which is used to store information about suppliers that provide goods or services to the organization.

SupplierID is the primary key of the table and uniquely identifies each supplier. It is a small unsigned integer, ensuring that each supplier has a unique numerical ID.

SupplierName is a required text field (up to 50 characters) that stores the name of the supplier.

PhoneNumber is a 10-character text field that stores the supplier's contact phone number. It is not allowed to be null, meaning every supplier must have a phone number.

A **CHECK** constraint is added to ensure the phone number is in a valid format:

- The phone number must be exactly 10 digits long.
- It must start with a 0.
- It must contain only numeric characters, i.e., no letters or special characters are allowed.

Location is a required field (maximum of 50 characters) that stores the geographical location or address of the supplier.

Values Insertion

INSERT INTO Supplier
VALUE (610, "Sophia Aswaorth", "0542760273", "Florida"),
(611, "Grace Bryant", "0542760265", "Pensylvania"),
(612, "Evette Walker", "0542760323", "North Carolina"),
(613, "Harry Lui", "0542760453", "Texas"),
(614, "Keiva Slim", "0545660273", "Dallas");

Material Table

Table Creation

CREATE TABLE Material (
MaterialID SMALLINT UNSIGNED,

ExpiryDate DATE NOT NULL,

MaterialName VARCHAR (50) NOT NULL,

SupplierID SMALLINT UNSIGNED NOT NULL,

ItemID SMALLINT UNSIGNED,

FOREIGN KEY (SupplierID) REFERENCES Supplier (SupplierID) ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY (ItemID) REFERENCES Inventory (ItemID) ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY (ItemID, MaterialID)

);

Table Description

The table **Material** has a **composite primary key** made up of **ItemID** and **MaterialID**, which means each combination of item and material must be unique.

MaterialID is an identifier for the material. It's a **small unsigned integer** but **not unique by itself** — uniqueness is enforced only when combined with ItemID.

ExpiryDate is a required field that records the **date the material will expire**, helping in inventory control and safety checks.

MaterialName stores the **name or label of the material** (e.g., "Aluminum", "Plastic Sheets") and is required.

SupplierID links the material to the **supplier that provided it**. It must reference a valid SupplierID from the Supplier table, and this relationship includes:

- **ON UPDATE CASCADE**: If a supplier's ID changes, it will be updated here automatically.
- **ON DELETE CASCADE**: If a supplier is deleted, all related material records are also removed.

ItemID links the material to a specific **item in the inventory**, referencing ItemID from the Inventory table. It has the same cascading rules for updates and deletions.

Values Insertion

INSERT INTO Material

VALUE (101, "2025-09-12", "Sugar", 610, 1), (102, "2026-09-12", "Flour", 612, 3),

(103, "2025-12-12", "Oil", 610, 5), (104, "2025-04-16", "Greese", 610, 1),

(105, "2025-11-02", "Water", 610, 2);

MaterialPlant Table

Table Creation

CREATE TABLE MaterialPlant (

ItemID SMALLINT UNSIGNED,

MaterialID SMALLINT UNSIGNED,

PlantID SMALLINT UNSIGNED,

PRIMARY KEY (PlantID, ItemID, MaterialID),

FOREIGN KEY (PlantID) REFERENCES Plant (PlantID) ON DELETE RESTRICT ON UPDATE RESTRICT,

FOREIGN KEY (ItemID, MaterialID) REFERENCES Material (ItemID, MaterialID) ON DELETE RESTRICT ON UPDATE RESTRICT

);

Table Description

The **MaterialPlant** table is designed to establish a relationship between materials and the plants where they are utilized or stored. This association is crucial for tracking material distribution across various plant locations. This functions as a junction table, facilitating a many-to-many relationship between materials and plants. This design allows for comprehensive tracking of which materials are available or used at specific plant locations, which is essential for inventory management, logistics, and operational planning. The table has the following columns:

ItemID: An unsigned small integer that, in combination with MaterialID, uniquely identifies a material.

MaterialID: An unsigned small integer that, along with ItemID, specifies a particular material.

PlantID: An unsigned small integer representing the unique identifier of a plant.

Primary Key (PlantID, ItemID, MaterialID): This composite primary key ensures that each combination of plant, item, and material is unique within the table, preventing duplicate entries for the same material at the same plant.

Foreign Key (PlantID): This constraint establishes a relationship with the Plant table, ensuring that each PlantID in the MaterialPlant table corresponds to a valid entry in the Plant table. The ON DELETE

RESTRICT and ON UPDATE RESTRICT actions prevent deletion or modification of a plant record if it is referenced in the MaterialPlant table.

Foreign Key (ItemID, MaterialID): This composite foreign key references the Material table, ensuring that each combination of ItemID and MaterialID exists in the Material table. Similar to the previous constraint, the ON DELETE RESTRICT and ON UPDATE RESTRICT actions maintain referential integrity by restricting deletions or updates that would affect related records in the MaterialPlant table.

Values Insertion

INSERT INTO MaterialPlant

VALUES (1, 101, 100), (2, 105, 300), (3, 102, 200), (1, 104, 200), (1, 101,500);

Product Table

Table Creation

CREATE TABLE Product (

ProductID SMALLINT UNSIGNED,

ItemID SMALLINT UNSIGNED,

Price DECIMAL (20,2) NOT NULL,

ProductName VARCHAR (50) NOT NULL,

TruckID SMALLINT UNSIGNED,

PlantID SMALLINT UNSIGNED,

PRIMARY KEY (ItemID, ProductID),

FOREIGN KEY (TruckID) REFERENCES Truck (TruckID) ON DELETE RESTRICT ON UPDATE RESTRICT,

FOREIGN KEY (PlantID) REFERENCES Plant (PlantID) ON DELETE CASCADE ON UPDATE CASCADE

);

Table Description

The statement creates a table named **Product**, designed to store information about products associated with specific inventory items, transportation vehicles, and manufacturing plants. The table uses a composite

primary key consisting of ItemID and ProductID. This ensures that each combination of item and product is unique within the table.

ProductID: A small unsigned integer representing the unique identifier for the product within the context of a specific item.

ItemID: A small unsigned integer linking the product to a specific item in the inventory.

Price: A decimal value (up to 20 digits, with 2 decimal places) indicating the price of the product. This field is mandatory.

ProductName: A variable character field (up to 50 characters) storing the name of the product. This field is mandatory.

TruckID: A small unsigned integer referencing the truck responsible for transporting the product.

PlantID: A small unsigned integer referencing the plant where the product is manufactured or processed.

TruckID references the TruckID in the Truck table. The ON DELETE RESTRICT and ON UPDATE RESTRICT clauses ensure that a truck cannot be deleted or its ID changed if it is associated with any product.

PlantID references the PlantID in the Plant table. The ON DELETE CASCADE and ON UPDATE CASCADE clauses ensure that if a plant is deleted or its ID updated, the corresponding changes cascade to the Product table.

Values Insertion

INSERT INTO Product
VALUE (10, 1, 300.11, "Washing Machine", 1, 100), (20, 3, 400.56, "Biscuit", 3, 200),
(30, 5, 800.00, "Zinc cover", 2, 300), (40, 1, 567.45, "Paint", 2, 400),
(50, 2, 123.12, "Oil Machine", 5, 500);

ProductMaterial Table

Table Creation

CREATE TABLE ProductMaterial (

ProductID SMALLINT UNSIGNED,

ItemID SMALLINT UNSIGNED,

MaterialID SMALLINT UNSIGNED,

PRIMARY KEY (ItemID, MaterialID, ProductID),

FOREIGN KEY (ItemID, MaterialID) REFERENCES Material (ItemID, MaterialID) ON DELETE RESTRICT ON UPDATE RESTRICT,

FOREIGN KEY (ItemID, ProductID) REFERENCES Product (ItemID, ProductID) ON DELETE RESTRICT ON UPDATE RESTRICT

);

Table Description

This table a junction (or bridge) table that connects Products and Materials representing which materials are used to make which products. The has the following columns:

ProductID: A small unsigned integer that identifies a product.

ItemID: A small unsigned integer (possibly a shared key used across tables).

MaterialID: A small unsigned integer that identifies a material.

All three are likely identifiers linking to other tables. This means that each unique combination of ItemID, MaterialID, and ProductID is used only once in this table. No duplicates of the same combo are allowed.This ensures the ItemID and MaterialID in this table must exist in the Material table.

RESTRICT means you cannot delete or update a material if it's already linked in this table. This ensures the ItemID and ProductID must exist in the Product table. Again, RESTRICT prevents deletions/updates that would break this link.

Values Insertion

INSERT INTO ProductMaterial

VALUE (10,1,101), (20,3,102), (30,5,103), (40,1,104), (50,2,105);

Order Table

Table Creation

CREATE TABLE `Order` (

OrderID SMALLINT UNSIGNED,

```
 OrderDate DATE,
 DeliveryAddress VARCHAR (150) NOT NULL,
 ItemID SMALLINT UNSIGNED,
 ProductID SMALLINT UNSIGNED,
 CustomerID SMALLINT UNSIGNED,
 PRIMARY KEY (OrderID, ItemID, ProductID, CustomerID),
FOREIGN KEY (ItemID, ProductID) REFERENCES Product (ItemID, ProductID) ON DELETE CASCADE ON UPDATE CASCADE,
 FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table Description

The statement creates a table named **Order**, which records customer orders by associating them with specific products and customers. It has the following columns:

OrderID: A small unsigned integer serving as the unique identifier for each order.

OrderDate: A date field indicating when the order was placed.

DeliveryAddress: A variable character field (up to 150 characters) specifying where the order should be delivered. This field is mandatory.

ItemID: A small unsigned integer referencing the specific item associated with the product in the order.

ProductID: A small unsigned integer identifying the specific product being ordered.

CustomerID: A small unsigned integer referencing the customer who placed the order.

The table uses a **composite primary key** consisting of OrderID, ItemID, ProductID, and CustomerID. This ensures that each combination of these fields is unique, preventing duplicate entries for the same order, product, and customer combination.

- **(ItemID, ProductID)**: This composite foreign key references the Product table's primary key, establishing a relationship between the order and the specific product being ordered. The ON DELETE CASCADE and ON UPDATE CASCADE clauses ensure that changes in the Product table are propagated to the Order table, maintaining referential integrity.

- **CustomerID**: This foreign key references the Customer table's primary key, linking each order to the customer who placed it. Similar to the product relationship, the ON DELETE CASCADE and ON UPDATE CASCADE clauses ensure that changes in the Customer table are reflected in the Order table.

Values Insertion

INSERT INTO Order

VALUE

(711, "2020-12-10", "10300 Pensacola 52321 Fl", 1, 10, 221),

(712, "2020-04-20", "10300 Pensacola 52321 Fl", 3, 20, 222),

(713, "2025-12-10", "10300 Texas 52321 TX", 5, 30, 223),

(714, "2011-12-20", "2034 Amari 52321 WA", 1, 40, 224),

(715, "2022-06-14", "359 England 52321 CA", 2, 50, 225);

QUERIES AND OUTPUTS

1. Many customers have many orders and are in different locations, we want to know the name of the customer and the number of each product they have ordered and how we can contact them.

```
SELECT
        c.CustomerName,
        c.PhoneNumber,
        p.ProductName,
        o.DeliveryAddress,
        COUNT(o.OrderID) AS TotalOrders
FROM
   Customer c
JOIN
   `Order` o ON c.CustomerID = o.CustomerID
JOIN
   Product p ON o.ProductID = p.ProductID
```

```
GROUP BY
    c.CustomerName, c.PhoneNumber, p.ProductName, o.DeliveryAddress
;
```

CustomerName	PhoneNumber	ProductName	DeliveryAddress	TotalOrders
Ratish Prakash	0245678610	Washing Machine	10300 Pensacola 52321 Fl	1
Raman Krishna	0255678610	Biscuit	10300 Pensacola 52321 Fl	1
Keeks Lady	0245670610	Zinc cover	10300 Texas 52321 TX	1
Christopher Anasah	0545678610	Paint	2034 Amari 52321 WA	1
Felix Addae	0245678230	Oil Machine	359 England 52321 CA	1

2. We were told by one of our supplier's that the material we received from them must be used as soon as possible. The logistics officer clearly remembers that those items were tagged with "raw" in their name. We want to find out those material name, their quantity and possibly the supplier and their location.

```
SELECT s.SupplierName  AS Supplier,
          m.MaterialName AS Substance,
          i.ItemType,
          i.StockQty,
          s.Location
FROM Inventory i
JOIN
    Material m ON i.ItemID = m.ItemID
JOIN
    Supplier s ON m.SupplierID = s.SupplierID
WHERE i.ItemID = (SELECT ItemID FROM Inventory WHERE ItemType LIKE '%Raw%')
ORDER BY   i.StockQty
;
```

Supplier	Substance	ItemType	StockQty	Location
Sophia Aswaorth	Sugar	Raw Materials	102	Florida
Sophia Aswaorth	Greese	Raw Materials	102	Florida

3. A customer reported that his company fired him for purchasing a bad product from us. The quality control manager analysed the situation and suggested the product might have been produced using materials of less time life. Hence, we want to find all products and their related materials produced for this situation.

```
SELECT
        m.MaterialName,
        p.ProductName,
        COUNT(DISTINCT p.ProductID) AS Total_Products
FROM
    Material m
JOIN
    ProductMaterial pm ON m.MaterialID = pm.MaterialID
JOIN
     Product p ON pm.ProductID = p.ProductID
WHERE
    m.ExpiryDate = (SELECT MIN(ExpiryDate) FROM Material)
GROUP BY
MaterialName, ProductName
;
```

MaterialName	ProductName	Total_Products
Greese	Paint	1

4. For the purpose of monthly inspection, evaluation and rotation, the production manager wants to know supervisors and employees and the specific plant they have been assigned to, the purpose of the plant and the specific venue of the plant.

```
SELECT
        e.FullName AS EmployeeName,
        p.PlantName,
        p.PlantLocation,
        p.OperationStatus,
        sup.EmployeeID AS SupervisorID
FROM
```

```
    Employee e
JOIN
    Plant p ON e.PlantID = p.PlantID
LEFT JOIN
    Supervisor sup ON e.EmployeeID = sup.EmployeeID
;
```

EmployeeName	PlantName	PlantLocation	OperationStatus	SupervisorID
Patrick Ansah	Sagali	Texas	Mining	111
Davis Asoma	Sagali	Texas	Mining	NULL
Felicia Amana	Tema	Atlanta	Refinery	112
Jeff Abban	Mashda	New York	Cannary	NULL
Cecilia Ackon	Mashda	New York	Cannary	118
Gideon Bright	Segal	Washington	Mixer	114
Wendy Eshun	Segal	Washington	Mixer	NULL
John Andai	Pinto	Pensacola	Grinder	116

5. One of our customers reported of a missing product with ID 10 and 30 he ordered. We want to trace the truck that was suppose to deliver those products and the employees who drives that truck.

```
SELECT e.FullName,
    t.TruckName,
    t.Model,
    p.ProductID
FROM Employee e
JOIN Truck t ON e.EmployeeID = t.EmployeeID
JOIN Product p ON t.TruckID = p.TruckID
WHERE ProductID IN (10, 20)
;
```

FullName	TruckName	Model	ProductID
Patrick Ansah	Toyota	Corrolla	10
Cecilia Ackon	Benz	A-300	30

6. For each material used in production, we want to know the name of the material, the plant where it is processed, the location of that plant, and the total number of distinct products that material is used to produce

```
SELECT
    m.MaterialName,
    pl.PlantName,
    pl.Location,
    COUNT(DISTINCT pm.ProductID) AS Total_Products
FROM
  Material m
JOIN
  MaterialPlant mp ON m.MaterialID = mp.MaterialID
JOIN
  Plant pl ON mp.PlantID = pl.PlantID
JOIN
  ProductMaterial pm ON m.MaterialID = pm.MaterialID
GROUP BY
  m.MaterialName, pl.PlantName, pl.Location
;
```

MaterialName	PlantName	PlantLocation	Total_Products
Flour	Tema	Atlanta	1
Greese	Tema	Atlanta	1
Sugar	Pinto	Pensacola	1
Sugar	Sagali	Texas	1
Water	Mashda	New York	1

REVERSE ENGINEERING DIAGRAM