UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

Faculty of Contemporary Sciences and Technologies
Tetovo

Databases

(Sem. 4, 2022/2023)

Project:

The Database Application for the fictional company OneZero

Student(s):
1. Ron Ismaili (130050)
2. Edi Sulo (130320)
3. Omar Amzi (129978)

Mentor:
Prof. Dr. Jaumin Ajdari

May 2023, Tetovo

Abstract

In today's digital age, data management is essential for businesses to operate efficiently and effectively. This is especially true for computer companies that sell computer parts and prebuilt computers, as they deal with a vast amount of data related to their products, customers and suppliers.

OneZero is a fictional computer company based in Tetovo, North Macedonia. We specialize in selling both computer parts and prebuilt computers. Our catalog includes an extensive range of parts such as processors, graphics cards, motherboards, RAM, storage devices and power supplies. We also offer prebuilt computers that are available in various configurations, from budget-friendly options to high-end machines.

To create the database, we plan on using Microsoft SQL Server as our DBMS and for the implementation of the interface we plan on utilizing Microsoft Access. In case we have finished implementing the interface to our database using Microsoft Access and we have enough time left, then we want to develop our own website to act as the interface to our database, where the website will have some extended functionality.

When it comes to technologies we will use to implement the database, we are planning on utilizing the following: Microsoft SQL Server (as the DBMS) and Microsoft Access (as the initial interface of our database).

If time permits, for the interface instead of using Microsoft Access we will develop a website utilizing HTML, CSS, JavaScript and Python.

As a result of our project, we expect to gain a deeper understanding into the intricacies of design, creation, manipulation and overall use of databases. On the flip side, a well-designed database can provide valuable insights and help our fictional company make better-informed decisions.

Key words: Databases, database design, database implementation, SQL.

Group members and their contributions:

1. **Ron Ismaili**
   I was responsible for the overall project (project management) as well as creating the problem description and conceptual design of the database.

2. **Edi Sulo**
   I was responsible for writing the abstract, keywords, introduction and creating the relational design of the database.

3. **Omar Amzi**
   I was responsible for writing the business rules and SQL code of the database.

It goes without saying that this is a group project meaning that there was collaboration on all parts of the project by all group members, the above mentioned contributions are merely there to inform the reader about who was primarily responsible for what.
When it comes to how and why we divided the work, it’s because this is what we thought would be the most efficient and accommodating to everyone’s interests and skills.

Introduction

The reason for creating this database is because OneZero is aiming to expand in the near future and needs to have a better overview of all the data that is necessary for the business operations, such as keeping track of net profit, new customers, demand and much more. Once the data is organized, it can be used for countless operations.

For the database purposes of OneZero we have to keep track of different types of data. Primarily we will need to store data about our customers, employees, products, invoices, stores and suppliers.

All of this data is necessary for the proper functioning of our company. Example uses of this data can be the calculation of the salary for our employees, to check how much we have made and spent in the last month/quarter/year, to find out which products have been sold the most/least and many other data operations.

The following users will have access to the database: Database administrators (owners), managers, employees and customers.

Database Administrators are responsible for the overall management and security of the database. Managers will be responsible for keeping the database up to date with the newest information regarding the customers, products etc. for the store they manage; Employees will have access only to specific data of the database which corresponds to the needs of their position. Customers will have access only to their own data, such as purchase history, as well as data about the products we offer, their price, the amount currently in stock and location, as well as a list of our available services.

THIS IS ONLY THE FIRST PART OF THE PROJECT, THIS MEANS THAT EVERYTHING WHICH IS SEEN HERE IS SUBJECT TO CHANGE (WITHIN REASON)

Problem description

OneZero is a fictional computer parts and servicing company based in Tetovo, North Macedonia. Co-founded by **Ron Ismaili**, **Edi Sulo** and **Omar Amzi**. This company operates in many locations of North Macedonia and it has many stores. OneZero has **3 main activities**: **Purchase** of pre-built computers and computer parts in partnership with our suppliers, **sale** of pre-built computers and computer parts to our customers and finally **maintenance** of desktop computers (cleaning & repair of computers to their original factory state).

All of our **customers** have registered an account with us and new customers have to register for an account before being able to purchase our products or use our services. The reason we do this is so that we can gather more data on customer behavior and improve the quality of our internal and external business operations.

All of our **employees** are assigned different positions. They can be assigned to be a **manager**, **technician** or **clerk** but an employee cannot be assigned more than one position. All of these positions have different salaries and responsibilities.

Managers have the highest salary and their responsibility is to manage the business operations of a specific store (ordering of products for the store, updating of the database for the store, holding meetings with the store employees, etc). One manager can manage only one store.

Technicians have the second highest salary and their responsibility is to service the desktop computers that the customers bring in and issue invoices for their service. One technician works on one computer (until they finish and move to the next one).

Clerks have the lowest salary in the company and their responsibility is to operate the cash register and process payments, where for every purchase an invoice is issued by the clerk.

Our company offers 2 different types of **products**: **Pre-built computers** and **computer parts**. All of the products are not stored in the same store, rather, they are stored in different stores. Every store has its own **warehouse** where it keeps all of the stores' products. We sell our products to our customers and we purchase our products from our suppliers.

Invoices are created by our employees whenever a customer purchases one of our products or uses our services. Alternatively, invoices are also generated whenever a manager purchases products for a specific store from one of our partnered suppliers. Invoices are important for our company as they are used to keep track of all our incomes and expenses.

The **stores** of our company are in many different locations but not in all locations of North Macedonia. OneZero has signed contracts with many different **suppliers** which provide OneZero with pre-built computers and computer parts whenever a manager of a store makes an **order**. Every purchase a manager makes from a supplier is recorded using invoices.

Business rules

OneZero is headed by Ron Ismaili, followed by Edi Sulo and Omar Amzi who are all responsible for overseeing the business aspects of the company. Below them are the Managers who are each responsible for the purchase of computer parts from suppliers, as well as maintaining the day to day operations of their store, as there is only one manager per store, by using their executive power over the other staff in their store, meaning the Technicians and Clerks.

The company's customers must first register an account with us for the purposes of authentication in order to have access to our products and services, afterwards they may purchase a computer, who will issue them an invoice containing the details of the transaction once they have made a purchase, or they can have their computer sent in for maintenance by one of our Technicians. To keep up with the demand for computers, as well as the need for computer parts to conduct repairs with, the company frequently makes purchases from suppliers it has an agreement with and stores them in the warehouses of a location's stores, while also making an invoice to track these transactions.

With all of this in mind, we believe that we need to gather information about:

- The **customers**, such as their name, surname, the credentials they provide during the registration process etc. so that we can properly authenticate them;

- The **employees**, such as their specific role in the company, their Social Security Number, their salary etc. in order to be able to pay them appropriately;

- The **products**, such as their prices, quantity in stock, location etc. so we know for example, when to restock;

- The **invoices** that the employees provide to the customers for products or services, as well as the invoices to the suppliers in order to keep track of the company's transactions;

- The **stores**, such as their locations, what kind of products they store in their warehouses and number of employees to know the net amount of products we have in storage as well as where to find them, as well as to know how many employees are in each store;

- The **suppliers**, such as the parts we purchase from them as well as their prices and quantities in order to adjust our own prices accordingly.

The way this data will be gathered depends on the data itself. Data about the customer is gathered upon registration and purchases of the company's products and services. Data about employees is periodically refreshed but is initially gathered upon hiring. Data on our products, services and invoices will be gathered each time a customer makes a purchase. Store data is gathered every time there is a new hire or the store's warehouses are restocked, along with the information on our suppliers every time we purchase parts. All of this data will be stored within our database, spread out into their own spreadsheets.

The data we collect will be used exclusively for optimizing the workflow of the company by keeping track of its daily expenditures and profits, as well as for striking a balance between profitability and providing the best service to its customers.

With all of this said, our database will have to be able to store new data, fetch already existing data, and in case of a mistake it will need to be able to modify existing data and/or remove it from the database altogether. The storage of new data will be done by filling out the relevant fields in the interface that we will set up for the database, which will also contain buttons that allow you to go back and forth between several data entries so that you can select an older entry to print out or modify, or remove it with another button.

Of course it is important to create business rules for our company, which will serve as the guiding instructions for creating the database. The following are the business rules which we have derived from our problem description:

- For every product a customer purchases, a product invoice is issued (this invoice belongs only to this customer).
- For every computer a customer brings for service, a service invoice is issued (this invoice belongs only to this customer).
- Every employee must have one of the following positions: Manager, technician or clerk.
- Every employee must have only one position.
- Every employee must work in only one store.
- One manager must manage one store (they cannot manage more than one store).
- For every product a manager purchases, a supply invoice is issued (this invoice belongs only to this manager).
- Technicians issue service invoices for the computers they service.
- Technicians issue one invoice for their service (only one technician works on one computer).
- Clerks create product invoices when customers purchase products (an invoice is created by one clerk).
- Products are stored in different stores.
- Product invoices contain a description of what product was purchased.

- Supply invoices contain a description of what product was purchased.

We will also have reports for the following:

- A report for retrieving the username and password of a user of the database to authenticate the user’s login.
- A report for retrieving the inventory of each location (Product name, price, quantity etc.).
- A report to retrieve the up-to-date prices of computer parts and parts that are sold by the suppliers we are in a contract with.
- A report to retrieve all business expenses and incomes.
- A report to retrieve the full list of managers as well as the locations they are responsible for.
- A report to check the availability of our services and products, as well as the locations they are available.

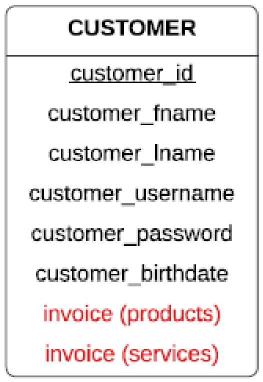
Conceptual design

In this stage of the project we need to take a look at the problem description as well as the business rules and derive the entity types and all of their possible attributes. Once we have found the above mentioned information, we need to draw the entity-relationship diagram (E-R diagram) of the given problem. The conceptual design has been created based on the 4 steps which we have covered during the practical database classes.

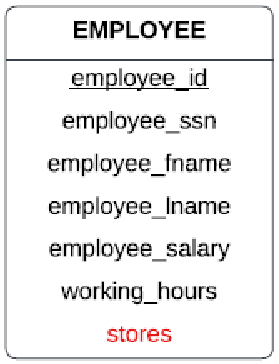
STEP 1: Entity types and attributes

After reading the problem description and business rules, we have a good idea of which entity types and attributes we require for this database. We have derived the following information:

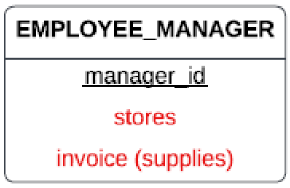
Entity type: CUSTOMER
Attributes: customer_id, customer_fname, customer_lname, customer_username, customer_password, customer_birthdate, invoice (products), invoice (services).



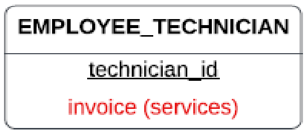
Entity type: EMPLOYEE (Superclass)
Attributes: employee_id, employee_ssn, employee_fname, employee_lname, employee_salary, working_hours, stores.



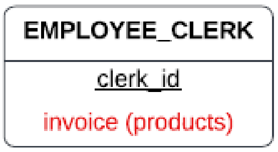
Entity type: EMPLOYEE_MANAGER (Subclass)
Attributes: manager_id, stores, invoice (supplies).



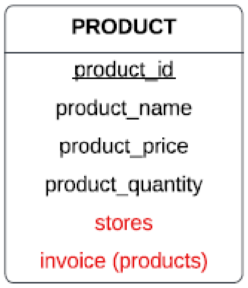
Entity type: EMPLOYEE_TECHNICIAN (Subclass)
Attributes: technician_id, invoice (services).



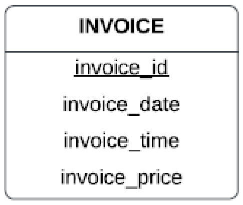
Entity type: EMPLOYEE_CLERK (Subclass)
Attributes: clerk_id, invoice (products).



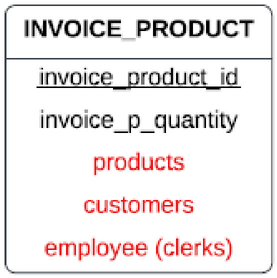
Entity type: PRODUCT
Attributes: product_id, product_name, product_price, product_quantity, stores, invoice (products).



Entity type: INVOICE (Superclass)
Attributes: invoice_id, invoice_date, invoice_time, invoice_price.



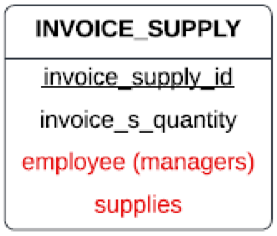
Entity type: INVOICE_PRODUCT (Subclass)
Attributes: invoice_product_id, invoice_p_quantity, products, customers, employee (clerks).



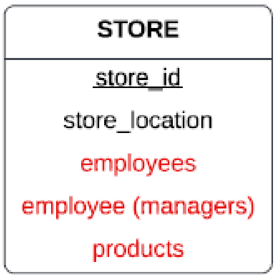
Entity type: INVOICE_SERVICE (Subclass)
Attributes: invoice_service_id, customers, employee (technicians).

INVOICE_SERVICE
invoice_service_id
customers
employee (technicians)

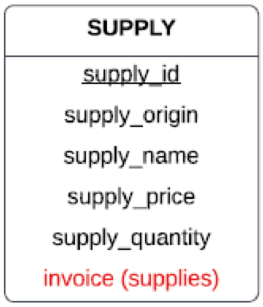
Entity type: INVOICE_SUPPLY (Subclass)
Attributes: invoice_supply_id, invoice_s_quantity, employee (managers), supplies.



Entity type: STORE
Attributes: store_id, store_location, employees, employee (managers), products.



Entity type: SUPPLY
Attributes: supply_id, supply_origin, supply_name, supply_price, supply_quantity, invoice (supplies).

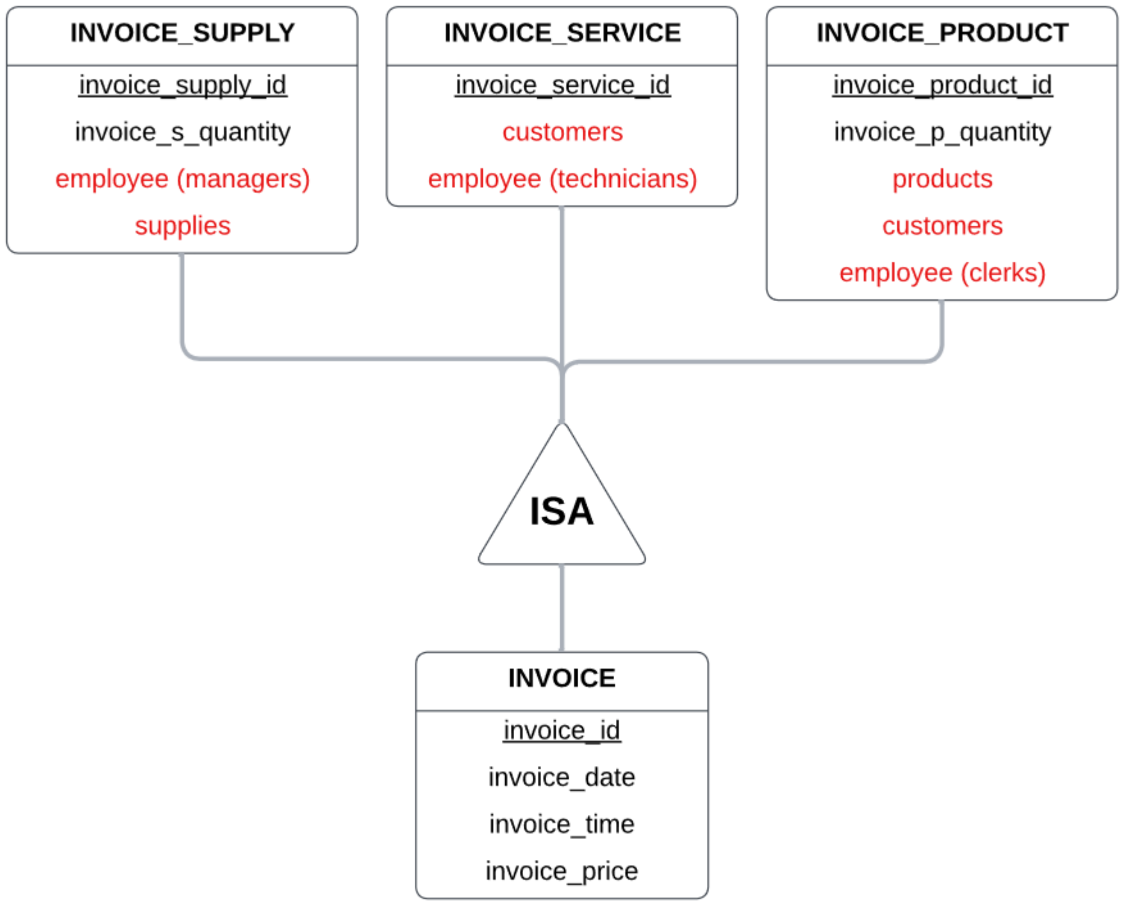


STEP 2: Relationships

Before we start with the relationships between the entity types, we have 2 class hierarchies which need to be defined.

We start with the superclass which has all its attributes directly related to itself. In our case that is the entity type **INVOICE**. It has all its attributes directly related to itself, meaning they will stay unchanged. Also, we can notice that this class hierarchy has 3 subclasses:

INVOICE_PRODUCT, **INVOICE_SERVICE** and **INVOICE_SUPPLY**.



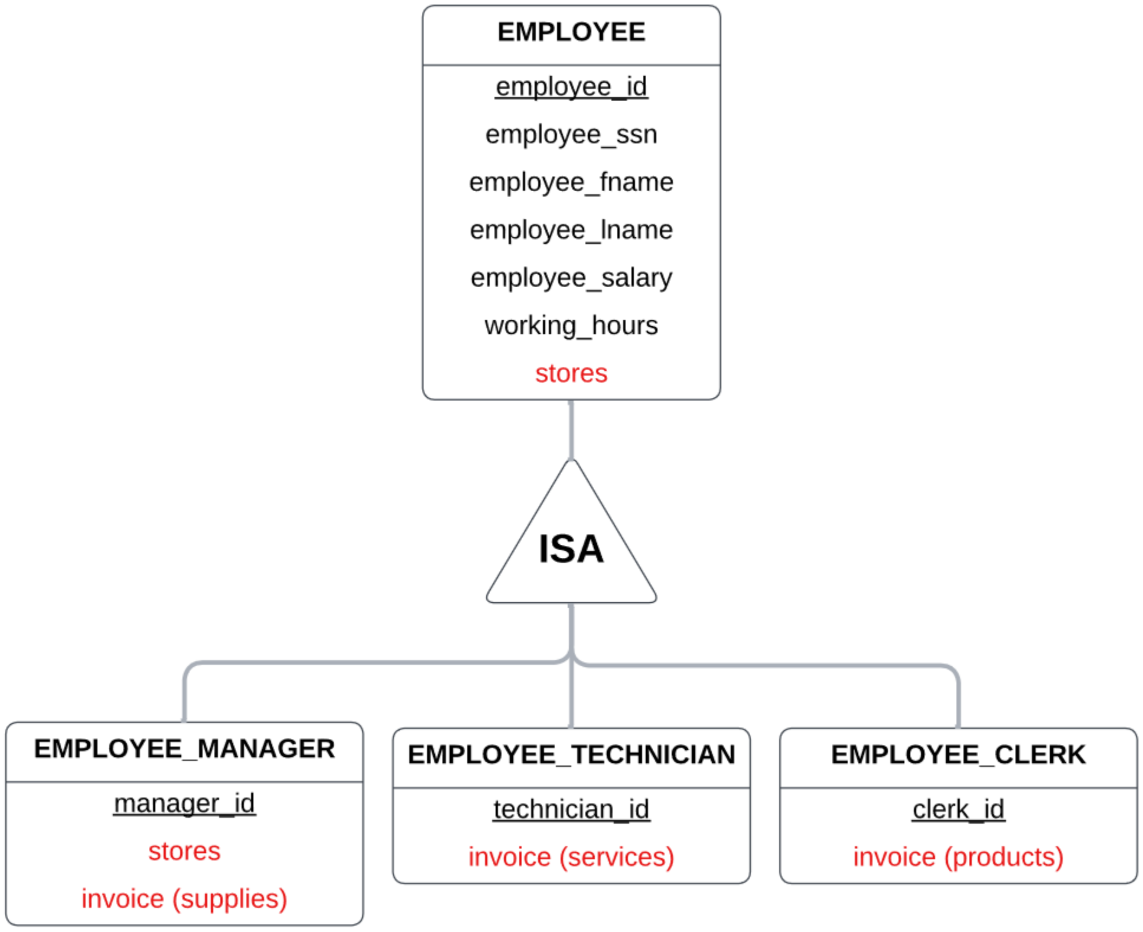
The reason why we split this entity type into 1 superclass and 3 subclasses is because we want to have 3 different tables for the different types of invoices which can be created in our database so that we have a clearer overview of the incomes and expenditures of our fictional company. We also believe that this would be more efficient when querying, also, if we had only one table there would be a lot of redundant data.

The first table **INVOICE_SUPPLY** will be created for the invoices which a manager receives when he purchases supplies for the store which he manages.
The second table **INVOICE_SERVICE** will be created for the invoices which are issued by the technicians when they service computers.
The third table **INVOICE_PRODUCT** will be created for the invoices which are issued by the clerks for the customers when they purchase our products.

All of these entity types share some common attributes such as date, time and price and that is why they have all been grouped into a class hierarchy. Important to note is that for this class hierarchy we have the **COVERING** constraint.

The second superclass in our database is the **EMPLOYEE** entity type. Since we know that all of our employees must have a position but they can't have more than one position, this means that for this class hierarchy we have the **COVERING** constraint. This class hierarchy also has 3 subclasses, they are the following:

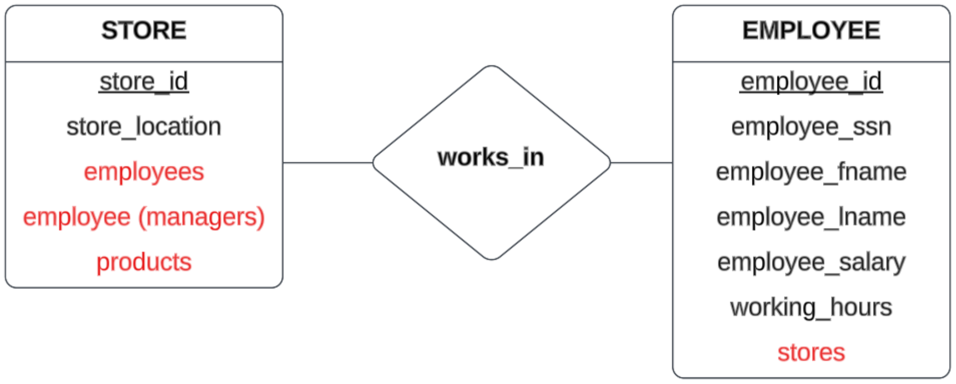
EMPLOYEE_MANAGER, EMPLOYEE_TECHNICIAN and **EMPLOYEE_CLERK**.



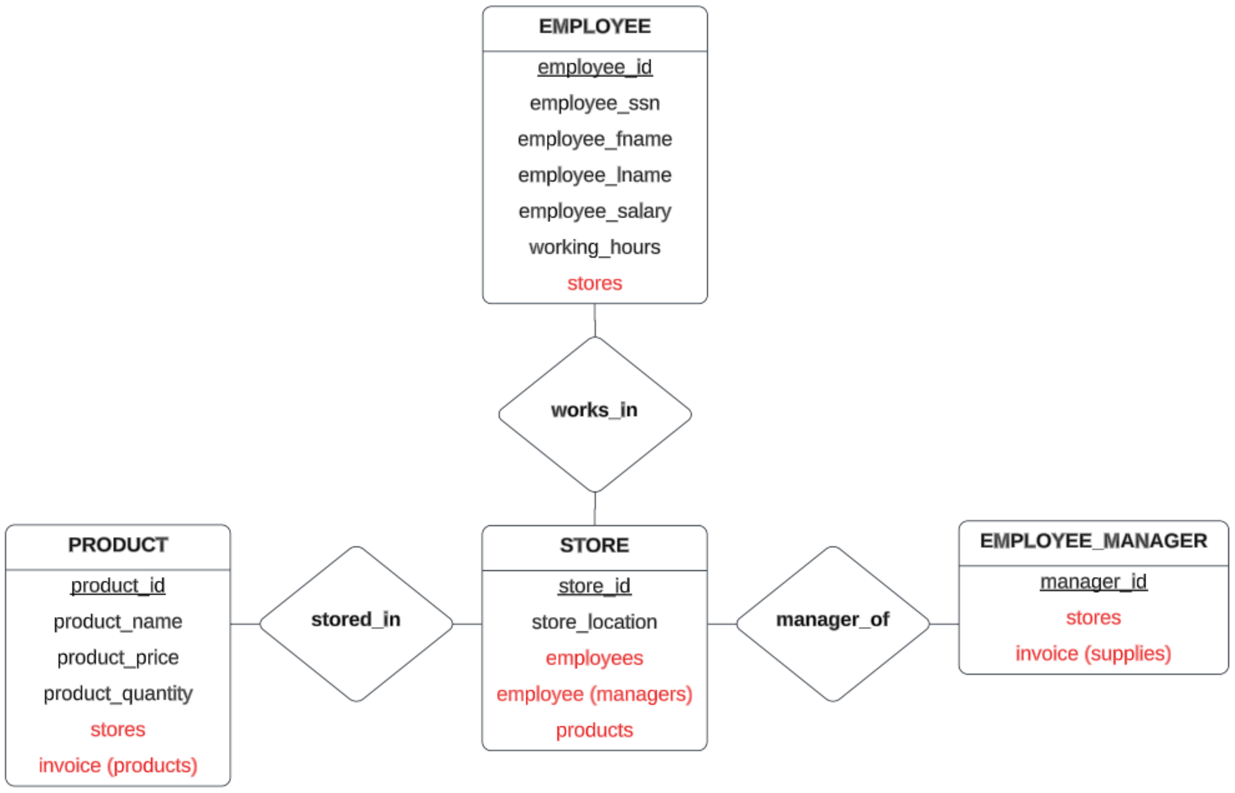
The reason why we split the employee entity type into subclasses is for almost the same reasons as stated for the INVOICE class hierarchy: It is easier to manage the data, for avoiding data redundancy and to have better querying.

Now we continue with the relationships between the entity types.

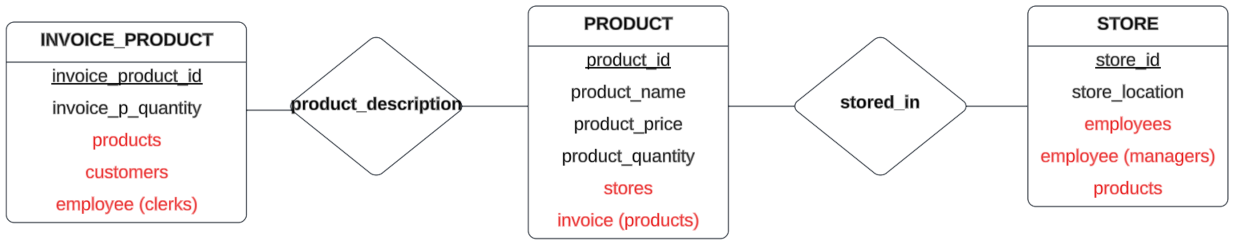
In the **EMPLOYEE** entity type we have an attribute which is not directly related to EMPLOYEE called stores. Since all employees must work in stores, we create a relationship called **works_in** between EMPLOYEE and STORE.



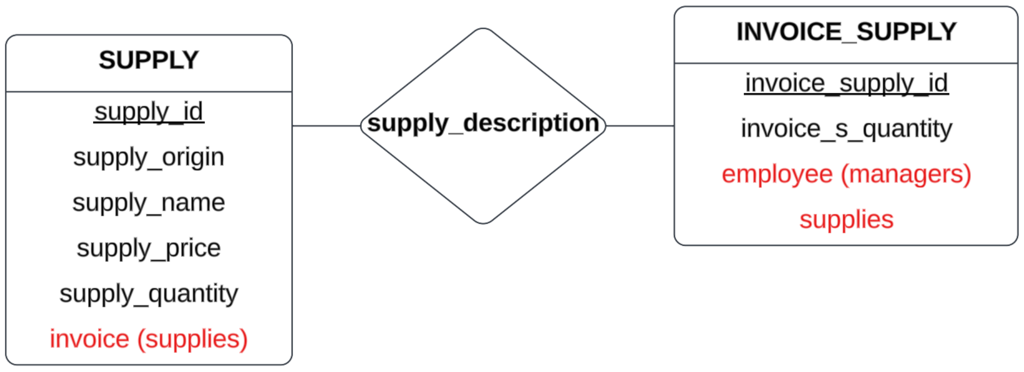
In the **STORE** entity type we see that besides employees we have 2 other attributes which are not directly related to STORE. They are employee (managers) and products. Since managers manage one store, we create the relationship called **manager_of** between EMPLOYEE_MANAGER and STORE. Also, since products are stored in stores, we create the relationship called **stored_in** between PRODUCT and STORE.



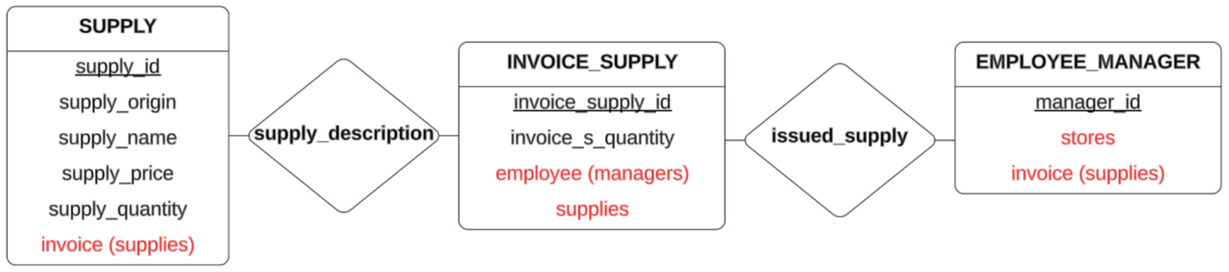
In the **PRODUCT** entity type we see that besides stores we have one other attribute which is not directly related to PRODUCT called invoice (products). Since product invoices should contain what products have been purchased, we create a relationship called **product_description** between INVOICE_PRODUCT and PRODUCT.



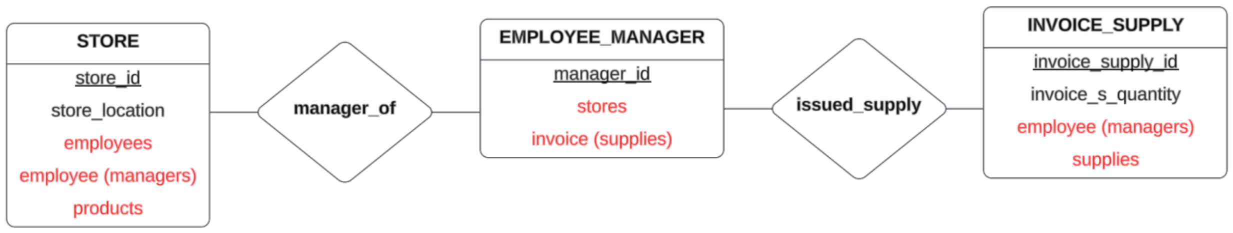
In the **SUPPLY** entity type we see that there is only one attribute which is not directly related to SUPPLY called invoice (supplies). Since supply invoices should contain what products (supplies) have been purchased, we create a relationship called **supply_description** between SUPPLY and INVOICE_SUPPLY.



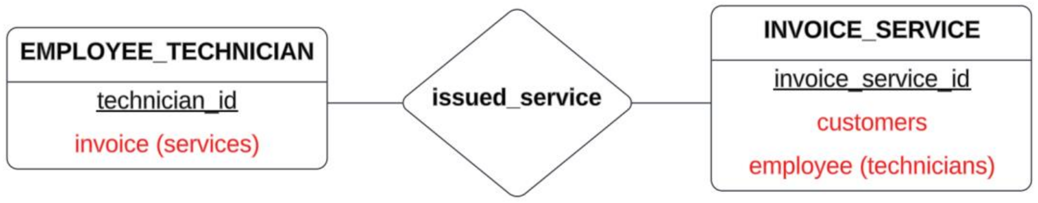
In the **INVOICE_SUPPLY** entity type we see that besides supplies we have one other attribute which is not directly related to INVOICE_SUPPLY called employee (managers). Since supply invoices should be issued by managers, we create a relationship called **issued_supply** between INVOICE_SUPPLY and EMPLOYEE_MANAGER.



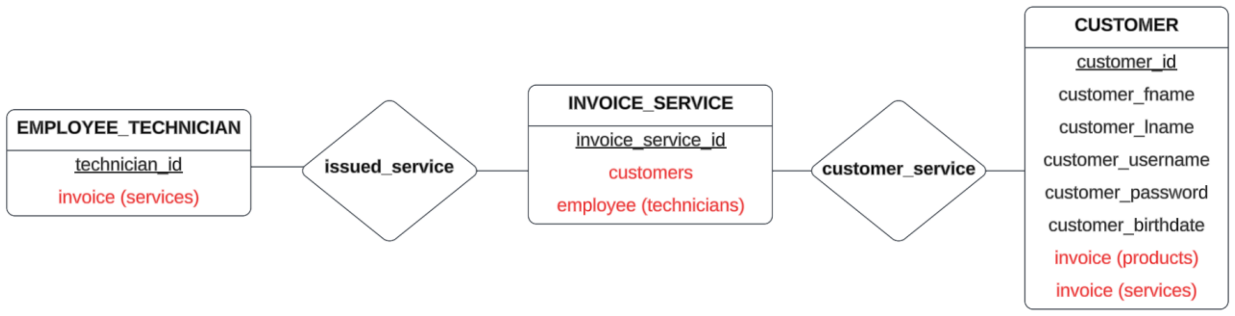
In the **EMPLOYEE_MANAGER** entity type we have 2 attributes, stores and invoice (supplies), which are not directly related to EMPLOYEE_MANAGER. Since the relationships **manager_of** and **issued_supply** have already been defined, we will only provide the following diagram:



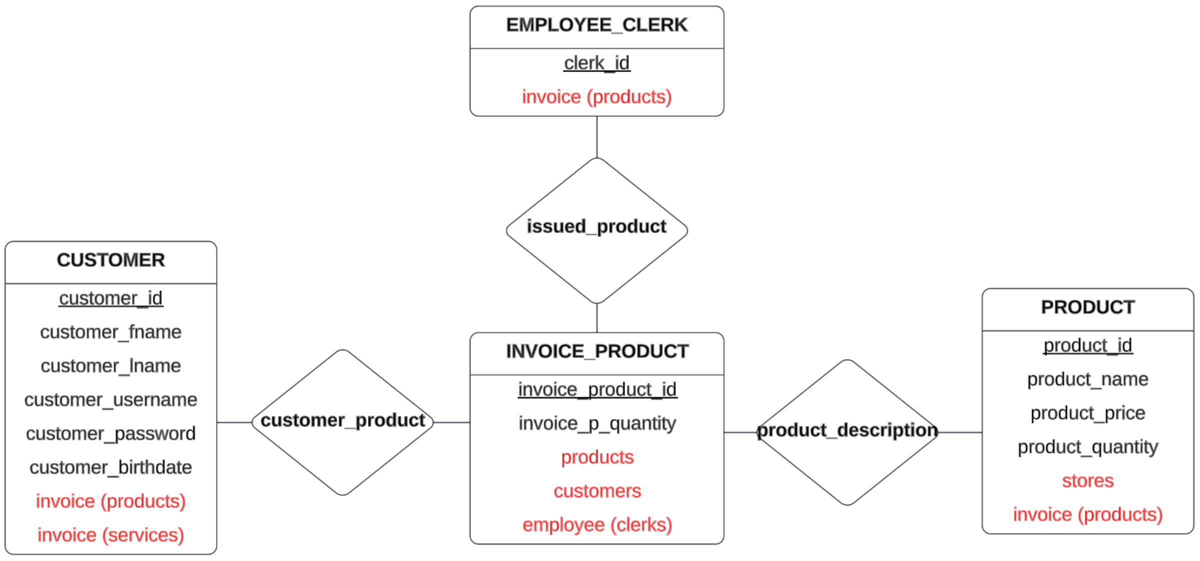
In the **EMPLOYEE_TECHNICIAN** entity type we see that there is only one attribute which is not directly related to EMPLOYEE_TECHNICIAN called invoice (services). Since technicians have to issue invoices when they service a computer, we create a relationship called **issued_service** between EMPLOYEE_TECHNICIAN and INVOICE_SERVICE.



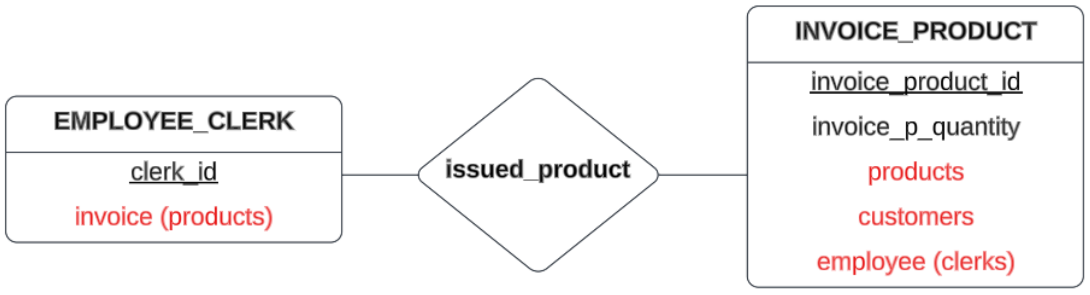
In the **INVOICE_SERVICE** entity type we see that besides employee (technicians) we have one other attribute which is not directly related to INVOICE_SERVICE called customers. Since customers receive invoices for their serviced computers, we create a relationship called **customer_service** between INVOICE_SERVICE and CUSTOMER.



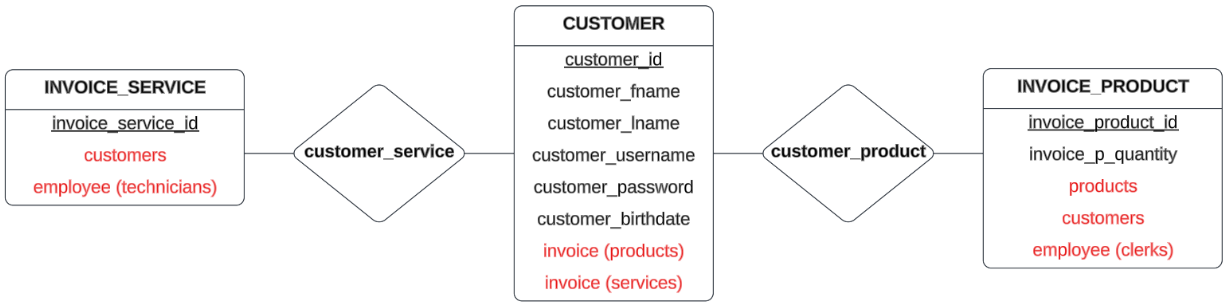
In the **INVOICE_PRODUCT** entity type we see that besides products we have 2 other attributes which are not directly related to INVOICE_PRODUCT called customers and employee (clerks). Since customers receive invoices for their purchased computers, we create a relationship called **customer_product** between INVOICE_PRODUCT and CUSTOMER. Also, since clerks issue product invoices, we create a relationship called **issued_product** between INVOICE_PRODUCT and EMPLOYEE_CLERK.



In the **EMPLOYEE_CLERK** entity type we see that there is only one attribute which is not directly related to EMPLOYEE_CLERK called invoice (products). Since clerks have to issue invoices when they sell products, we create a relationship called **issued_product** between EMPLOYEE_CLERK and INVOICE_PRODUCT.



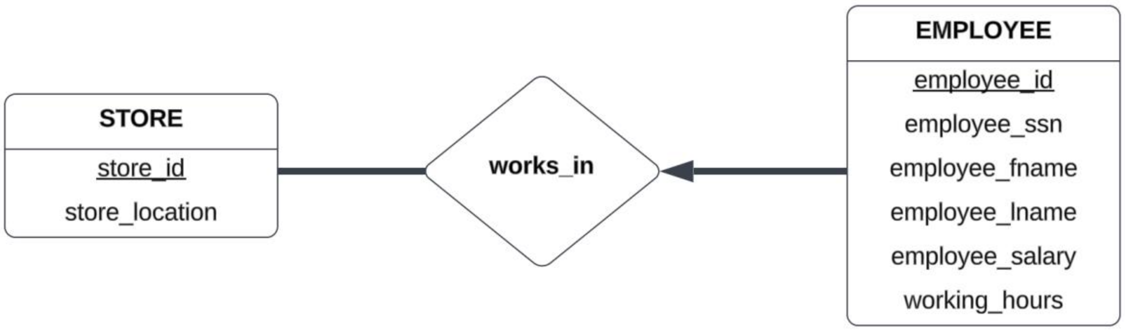
In the **CUSTOMER** entity type we have 2 attributes, invoice (products) and invoice (services), which are not directly related to CUSTOMER. Since the relationships **customer_service** and **customer_product** have already been defined, we will only provide the following diagram:



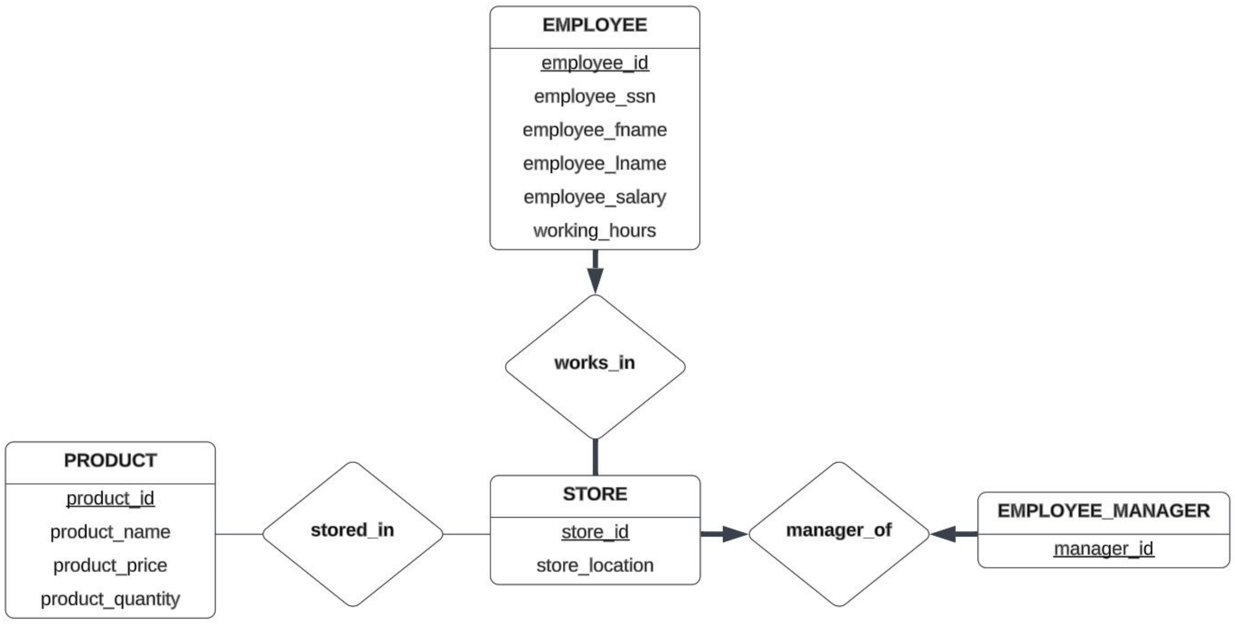
<u>STEP 3</u>: Cardinality and participation

In this step of the conceptual design we define the constraints of the relationships by determining the cardinality and participation.

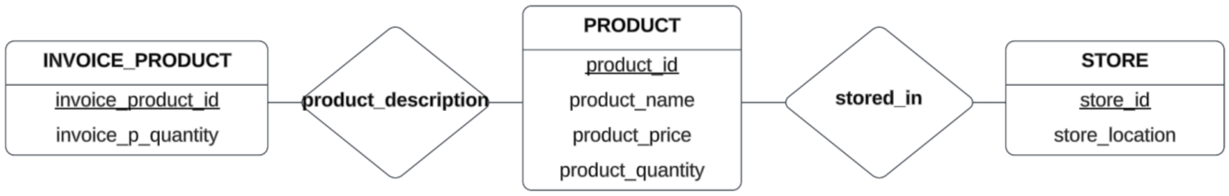
In the **works_in** relationship we have the entity types STORE and EMPLOYEE. Since one employee must work in only one store we say that this is total participation with cardinality one to many (1:N). Since every store must have employees, this means total participation on both sides.



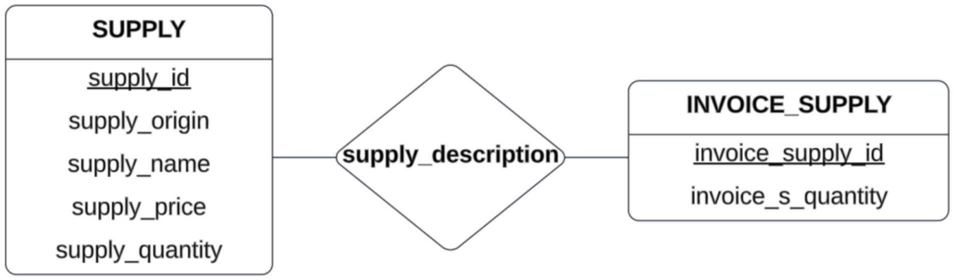
In the **stored_in** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one many products can be stored in many stores and many stores can have many products.
In the **manager_of** relationship we have a one to one cardinality (1:1) with total participation on both sides. The reasoning behind this is that one store must have only one manager (can't have no manager) and one manager can manage only one store (can't manage no stores).



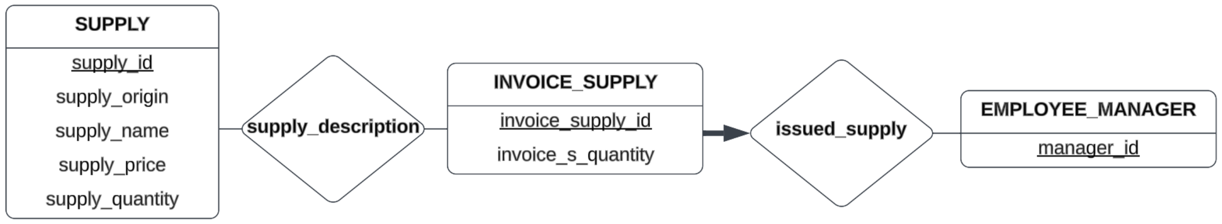
In the **product_description** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one product can appear on many product invoices and one product invoice can have many products.



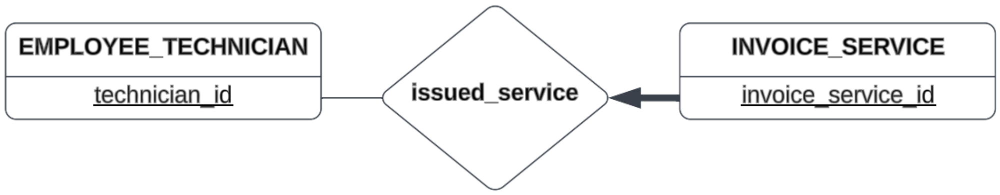
In the **supply_description** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one supply (product which the manager purchases) can appear on many supply invoices and one invoice can have many supplies.



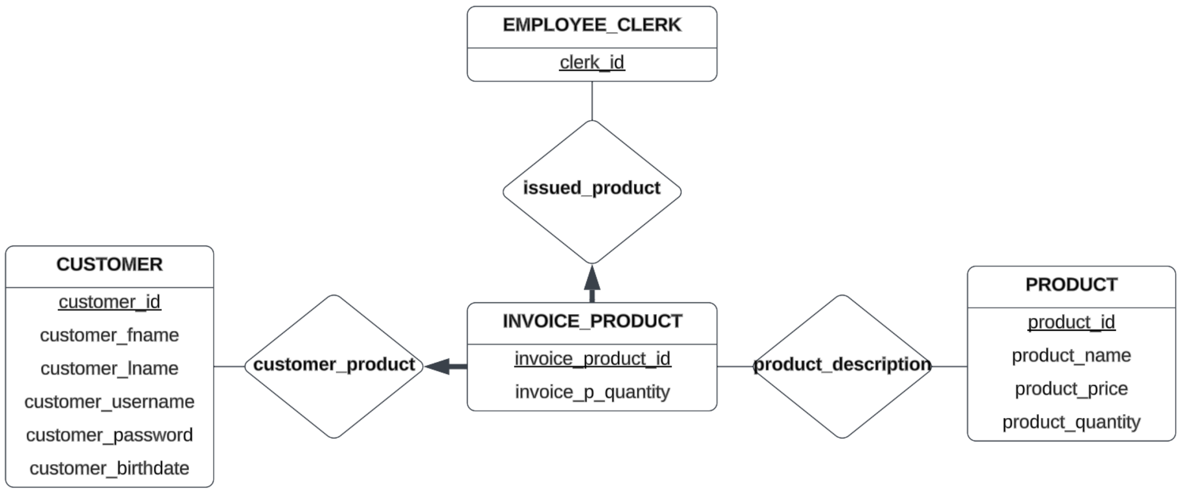
In the **issued_supply** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE_SUPPLY side. This means that one supply invoice can be issued by (belong to) one manager, whereas one manager can issue many supply invoices.



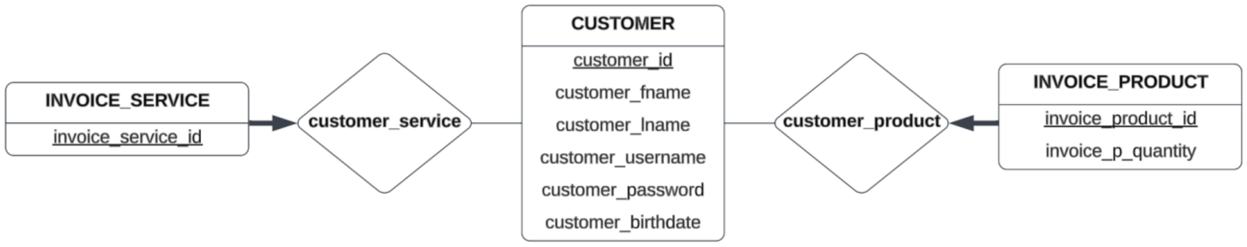
In the **issued_service** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE_SERVICE side. This means that one service invoice can be created by one technician, whereas one technician can issue many service invoices.



In the **issued_product** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE_PRODUCT side. This means that one product invoice can be issued by one clerk, whereas one clerk can issue many product invoices.



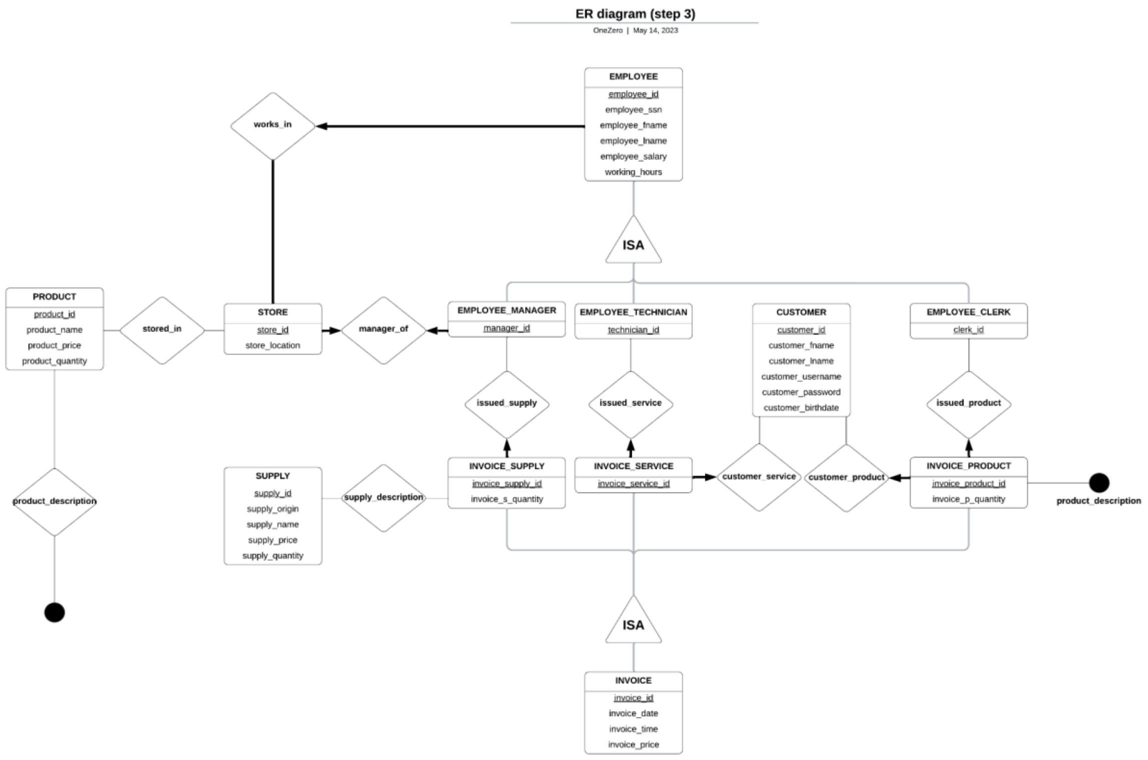
In the **customer_service** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE_SERVICE side. This means that one service invoice can belong only to one customer, whereas one customer can issue many service invoices (bring many computers for servicing).
In the **customer_product** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE_PRODUCT side. This means that one product invoice can belong only to one customer, whereas one customer can issue many product invoices (buy many products).



STEP 4: Refinement

In this stage of the project we haven’t done any refining of the database. We will do this step in the final part of the project.

The final (current) conceptual design (E-R diagram) of the database:



For a better quality image check out the attached files

Relational design

<u>Step 1</u>: Strong entity types

EMPLOYEE

<u>employee_id</u>	employee_ssn	employee_fname	Employee_lname	employee_salary	working_hours

STORE

<u>store_id</u>	store_location

PRODUCT

<u>product_id</u>	product_name	product_price	product_quantity

SUPPLY

<u>supply_id</u>	supply_origin	supply_name	supply_price	supply_quantity

CUSTOMER

<u>customer_id</u>	customer_fname	customer_lname	customer_username	customer_password	customer_birthdate

INVOICE

<u>invoice_id</u>	invoice_date	invoice_time	invoice_price

Step 2: Sub classes

Subclasses of Employee:

The subclass inherits all attributes of its superclass. The inheritance is done by using the primary key of the superclass as primary key of the subclass and this primary key also at the same time will be a foreign key into the subclass. **Manager_id**, **technician_id**, **clerk_id** are the same as **employee_id.**

Employee_Manager

manager_id

Employee_Technician

technician_id

Employee_Clerk

clerk_id

Subclasses of Invoice:

Invoice_Supply

Invoice_supply_id	invoice_s_quantity

Invoice_Service

Invoice_service_id

Invoice_Product

Invoice_product_id	invoice_p_quantity

Step 3: Relationships

Works_In: 1:N relationship between **Store** and **Employee**, **Employee** and **Store** have total participation and **Employee** has a key constraint on the relationship.

Employee

employee_id	employee_ssn	employee_fname	employee_lname	employee_salary	working_hours	store_id

Stored_In: M:N between **Product** and **Store** a new table stored_in is created which contains the primary key of the **Product** entity and **Store** entity.

stored_in

product_id	store_id

product_Description: M:N between **Product** and **Invocie_Product** a new table product_description is created which contains the primary key of the **Product** entity and **Invoice_Product** entity.

product_description

product_id	invoice_product_id

manager_of: 1:1 relationship between **Store** and **Employee_Manager** we can take the primary key of either entity and make it a foreign key for the other entity.

Employee_Manager

manager_id	store_id

supply_description: M:N between **Supply** and **Invoice_Supply** a new table supply_description is created which contains the primary key of the Supply entity and Invoice_Supply entity.

supply_description

supply_id	Invoice_supply_id

issued_supply: 1:N relationship between **Employee_Manager** and **Invoice_Supply**, the primary key of **Employee_Manager** goes to **Invoice_Supply** as a foreign key.

Invoice_Supply

Invoice_supply_id	Invoice_s_quantity	manager_id

issued_service: 1:N relationship between **Employee_Technician** and **Invoice_Service**, the primary key of **Employee_Technician** goes to **Invoice_Service** as a foreign key.

Invoice_Service

Invoice_service_id	technician_id

customer_service: 1:N relationship between **Customer** and **Invoice_Service**, the primary key of **Customer** goes to **Invoice_Service** as a foreign key.

Invoice_Service

Invoice_service_id	technician_id	customer_id

customer_product: 1:N relationship between **Customer** and **Invoice_Product**, the primary key of **Customer** goes to I**nvoice_Product** as a foreign key.

Invoice_Product

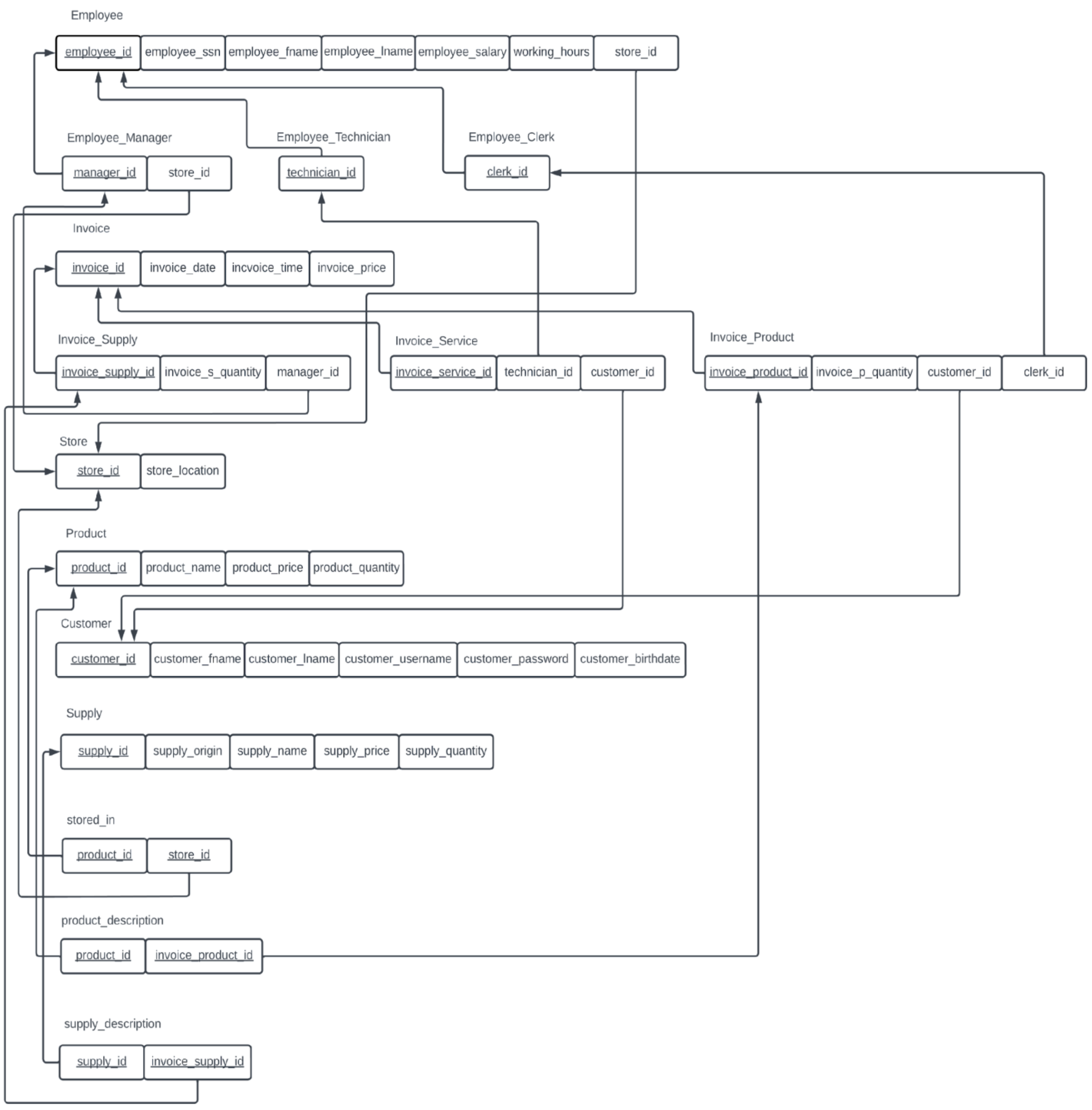
invoice_product_id	invoice_p_quantity	customer_id

Issued_product: 1:N relationship between **Employee_Clerk** and **Invoice_Product**, the primary key of **Employee_Clerk** goes to **Invoice_Product** as a foreign key.

Invoice_Product

invoice_product_id	invoice_p_quantity	customer_id	clerk_id

Step 4: Final Solution


Employee
employee_id
employee_ssn
employee_fname
employee_lname
employee_salary
working_hours
store_id
Employee_Manager
manager_id
store_id
Employee_Technician
technician_id
Employee_Clerk
clerk_id
Invoice
invoice_id
invoice_date
incvoice_time
invoice_price
Invoice_Supply
invoice_supply_id
invoice_s_quantity
manager_id
Invoice_Service
invoice_service_id
technician_id
customer_id
Invoice_Product
invoice_product_id
invoice_p_quantity
customer_id
clerk_id
Store
store_id
store_location
Product
product_id
product_name
product_price
product_quantity
Customer
customer_id
customer_fname
customer_lname
customer_username
customer_password
customer_birthdate
Supply
supply_id
supply_origin
supply_name
supply_price
supply_quantity
stored_in
product_id
store_id
product_description
product_id
invoice_product_id
supply_description
supply_id
invoice_supply_id

SQL code and implementation

```
--CREATION OF DATABASE
CREATE DATABASE OneZero
GO
USE OneZero
GO

--CREATION OF TABLES
CREATE TABLE Employee
(
employee_id int not null,
employee_ssn int,
employee_fname varchar(20),
employee_lname varchar(20),
employee_salary money,
working_hours int,
store_id int not null
)
GO

CREATE TABLE Employee_Manager
(
manager_id int not null,
store_id int not null
)
GO

CREATE TABLE Employee_Technician
(
technician_id int not null
)
GO

CREATE TABLE Employee_Clerk
(
clerk_id int not null
)
GO

CREATE TABLE Invoice
(
invoice_id int not null,
invoice_datetime date,
invoice_time time,
invoice_price money
)
GO

CREATE TABLE Invoice_Supply
(
invoice_supply_id int not null,
invoice_s_quantity int,
manager_id int not null
)
GO

CREATE TABLE Invoice_Service
(
invoice_service_id int not null,
technician_id int not null,
customer_id int not null
)
GO

CREATE TABLE Invoice_Product
(
invoice_product_id int not null,
invoice_product_quantity int,
customer_id int not null,
clerk_id int not null
)
GO

CREATE TABLE Store
(
store_id int not null,
store_location varchar(50)
)
GO

CREATE TABLE Product
(
product_id int not null,
product_name varchar(50),
product_price money,
product_quantity int
)
GO
```

```
CREATE TABLE Customer
(
customer_id int not null,
customer_fname varchar(20),
customer_lname varchar(20),
customer_username varchar(20),
customer_password varchar(20),
customer_birthdate date
)
GO

CREATE TABLE Supply
(
supply_id int not null,
supply_origin varchar(20),
supply_name varchar(20),
supply_price money,
supply_quantity int
)
GO

CREATE TABLE stored_in
(
product_id int not null,
store_id int not null
)
GO

CREATE TABLE product_description
(
product_id int not null,
invoice_product_id int not null
)
GO

CREATE TABLE supply_description
(
supply_id int not null,
invoice_supply_id int not null
)
GO
--ASSIGNMENT OF CONSTRAINTS
--PRIMARY KEYS
ALTER TABLE Employee
ADD CONSTRAINT PK_Employee PRIMARY KEY (employee_id);
```

```
GO
ALTER TABLE Employee_Manager
ADD CONSTRAINT PK_Employee_Manager PRIMARY KEY (manager_id);
GO
ALTER TABLE Employee_Technician
ADD CONSTRAINT PK_Employee_Technician PRIMARY KEY (technician_id);
GO
ALTER TABLE Employee_Clerk
ADD CONSTRAINT PK_Employee_Clerk PRIMARY KEY (clerk_id);
GO

ALTER TABLE Invoice
ADD CONSTRAINT PK_Invoice PRIMARY KEY (invoice_id);
GO
ALTER TABLE Invoice_Supply
ADD CONSTRAINT PK_Invoice_Supply PRIMARY KEY (invoice_supply_id);
GO
ALTER TABLE Invoice_Service
ADD CONSTRAINT PK_Invoice_Service PRIMARY KEY (invoice_service_id);
GO
ALTER TABLE Invoice_Product
ADD CONSTRAINT PK_Invoice_Product PRIMARY KEY (invoice_product_id);
GO

ALTER TABLE Store
ADD CONSTRAINT PK_Store PRIMARY KEY (store_id);
GO

ALTER TABLE Product
ADD CONSTRAINT PK_Product PRIMARY KEY (product_id);
GO

ALTER TABLE Customer
ADD CONSTRAINT PK_Customer PRIMARY KEY (customer_id);
GO

ALTER TABLE Supply
ADD CONSTRAINT PK_Supply PRIMARY KEY (supply_id);
GO

ALTER TABLE stored_in
ADD CONSTRAINT PK_stored_in PRIMARY KEY (product_id, store_id);
GO

ALTER TABLE product_description
ADD CONSTRAINT PK_product_description PRIMARY KEY (product_id,
invoice_product_id);
GO

ALTER TABLE supply_description
ADD CONSTRAINT PK_supply_description PRIMARY KEY (supply_id,
invoice_supply_id);
GO

--FOREIGN KEYS
ALTER TABLE Employee ADD CONSTRAINT FK_EmployeeStore
FOREIGN KEY (store_id) REFERENCES Store;
GO
ALTER TABLE Employee_Manager ADD CONSTRAINT FK_ManagerEmployee
FOREIGN KEY (manager_id) REFERENCES Employee;
GO
ALTER TABLE Employee_Manager ADD CONSTRAINT FK_ManagerStore
FOREIGN KEY (store_id) REFERENCES Store;
GO
ALTER TABLE Employee_Technician ADD CONSTRAINT FK_TechnicianEmployee
FOREIGN KEY (technician_id) REFERENCES Employee;
GO
ALTER TABLE Employee_Clerk ADD CONSTRAINT FK_ClerkEmployee
FOREIGN KEY (clerk_id) REFERENCES Employee;
GO

ALTER TABLE Invoice_Supply ADD CONSTRAINT FK_Invoice_SupplyInvoice
FOREIGN KEY (invoice_supply_id) REFERENCES Invoice;
GO
ALTER TABLE Invoice_Supply ADD CONSTRAINT FK_Invoice_SupplyManager
FOREIGN KEY (manager_id) REFERENCES Employee_Manager;
GO
ALTER TABLE Invoice_Service ADD CONSTRAINT FK_Invoice_ServiceInvoice
FOREIGN KEY (invoice_service_id) REFERENCES Invoice;
GO
ALTER TABLE Invoice_Service ADD CONSTRAINT FK_Invoice_ServiceTechnician
FOREIGN KEY (technician_id) REFERENCES Employee_Technician;
GO
ALTER TABLE Invoice_Service ADD CONSTRAINT FK_Invoice_ServiceCustomer
FOREIGN KEY (customer_id) REFERENCES Customer;
GO
ALTER TABLE Invoice_Product ADD CONSTRAINT FK_Invoice_ProductInvoice
FOREIGN KEY (invoice_product_id) REFERENCES Invoice;
GO
ALTER TABLE Invoice_Product ADD CONSTRAINT FK_Invoice_ProductCustomer
FOREIGN KEY (customer_id) REFERENCES Customer;
GO
```

```
ALTER TABLE Invoice_Product ADD CONSTRAINT FK_Invoice_ProductClerk
FOREIGN KEY (clerk_id) REFERENCES Employee_Clerk;
GO

ALTER TABLE stored_in ADD CONSTRAINT FK_stored_inProduct
FOREIGN KEY (product_id) REFERENCES Product;
GO
ALTER TABLE stored_in ADD CONSTRAINT FK_stored_inStore
FOREIGN KEY (store_id) REFERENCES Store;
GO

ALTER TABLE product_description ADD CONSTRAINT
FK_product_descriptionProduct
FOREIGN KEY (product_id) REFERENCES Product;
GO
ALTER TABLE product_description ADD CONSTRAINT
FK_product_descriptionInvoice
FOREIGN KEY (invoice_product_id) REFERENCES Invoice_Product;
GO

ALTER TABLE supply_description ADD CONSTRAINT FK_supply_descriptionSupply
FOREIGN KEY (supply_id) REFERENCES Supply;
GO
ALTER TABLE supply_description ADD CONSTRAINT FK_supply_descriptionInvoice
FOREIGN KEY (invoice_supply_id) REFERENCES Invoice_Supply;
GO
--UNIQUE
ALTER TABLE Employee_Manager ADD CONSTRAINT
UQ_Employee_ManagerStore
UNIQUE (store_id);
GO
```