****

**Faculty of Contemporary Sciences and Technologies**

**Tetovo**

**Databases**

**(Sem. 4, 2022/2023)**

**Project:**

***The Database Application for the fictional company OneZero***

**Student(s)**: **Mentor**:

1. Ron Ismaili (130050) Prof. Dr. Jaumin Ajdari
2. Edi Sulo (130320)
3. Omar Amzi (129978)

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.

1. **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**

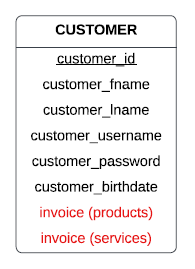
In this stage of the project we need to take a look at the problem description as well as the business rules and derive the entity types and all of their possible attributes. Once we have found the above mentioned information, we need to draw the entity-relationship diagram (E-R diagram) of the given 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:

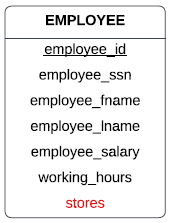
**Entity type:** CUSTOMER

**Attributes:** customer\_id, customer\_fname, customer\_lname, customer\_username, customer\_password, customer\_birthdate, invoice (products), invoice (services).

****

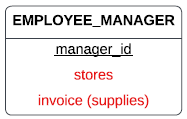
**Entity type:** EMPLOYEE (Superclass)

**Attributes:** employee\_id, employee\_ssn, employee\_fname, employee\_lname, employee\_salary, working\_hours, stores.



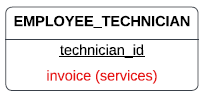
**Entity type:** EMPLOYEE\_MANAGER (Subclass)

**Attributes:** manager\_id, stores, invoice (supplies).

****

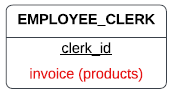
**Entity type:** EMPLOYEE\_TECHNICIAN (Subclass)

**Attributes:** technician\_id, invoice (services).



**Entity type:** EMPLOYEE\_CLERK (Subclass)

**Attributes:** clerk\_id, invoice (products).

****

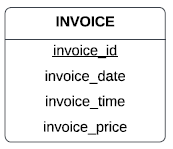
**Entity type:** PRODUCT

**Attributes:** product\_id, product\_name, product\_price, product\_quantity, stores, invoice (products).

****

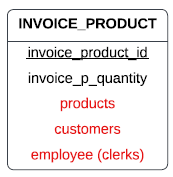
**Entity type:** INVOICE (Superclass)

**Attributes:** invoice\_id, invoice\_date, invoice\_time, invoice\_price.

****

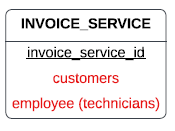
**Entity type:** INVOICE\_PRODUCT (Subclass)

**Attributes:** invoice\_product\_id, invoice\_p\_quantity, products, customers, employee (clerks).



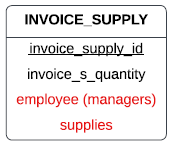
**Entity type:** INVOICE\_SERVICE (Subclass)

**Attributes:** invoice\_service\_id, customers, employee (technicians).

****

**Entity type:** INVOICE\_SUPPLY (Subclass)

**Attributes:** invoice\_supply\_id, invoice\_s\_quantity, employee (managers), supplies.

****

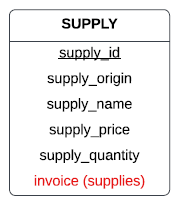
**Entity type:** STORE

**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).

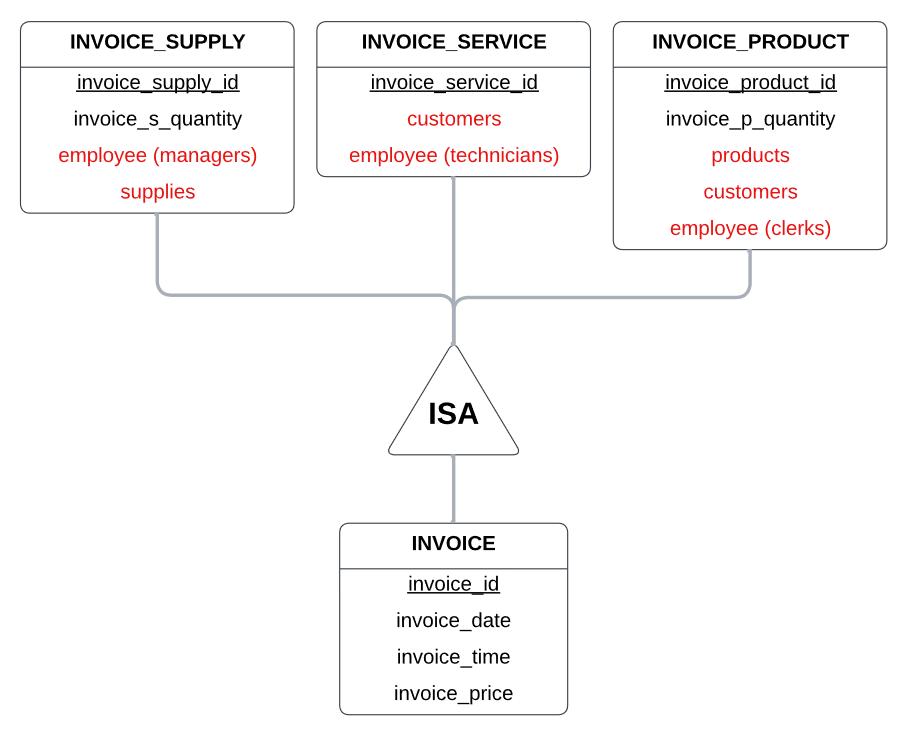


**STEP 2: Relationships**

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.

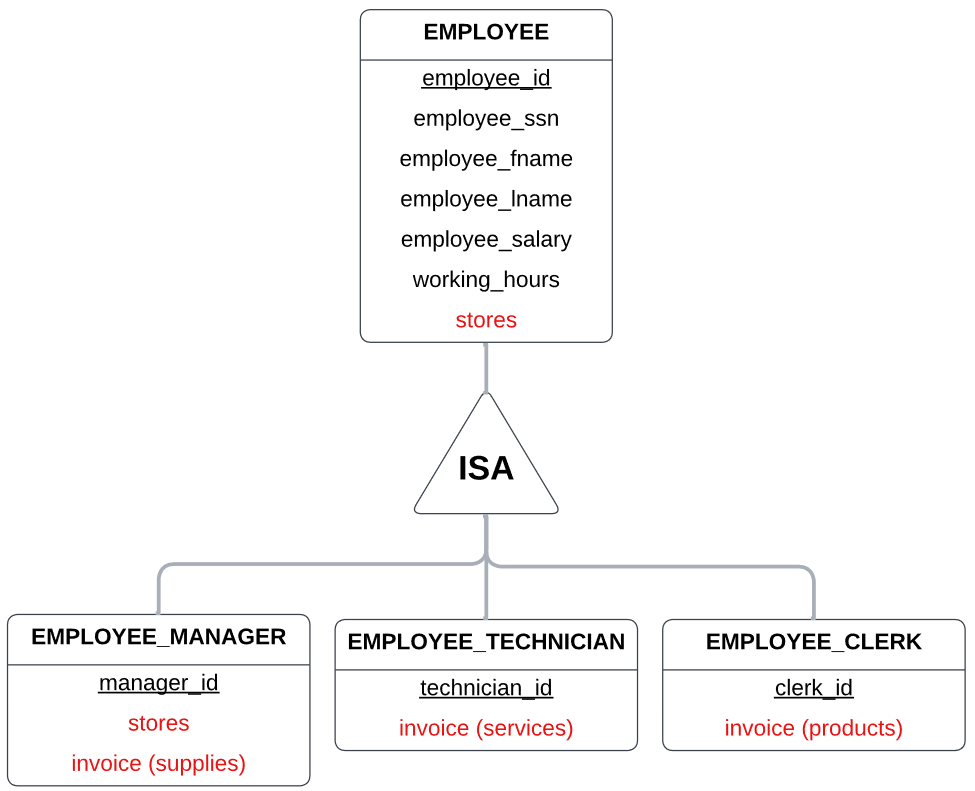
The second table **INVOICE\_SERVICE** will be created for the invoices which are issued by the technicians when they service computers.

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:

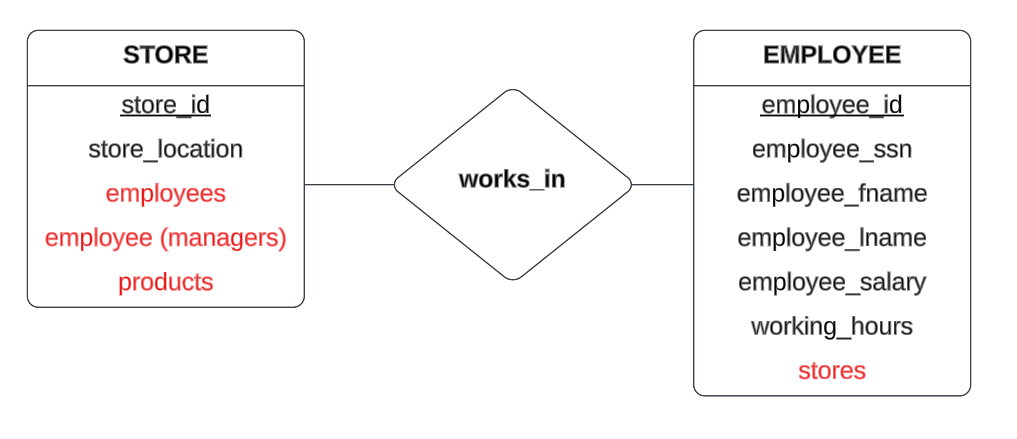
**EMPLOYEE\_MANAGER, EMPLOYEE\_TECHNICIAN** and **EMPLOYEE\_CLERK**.



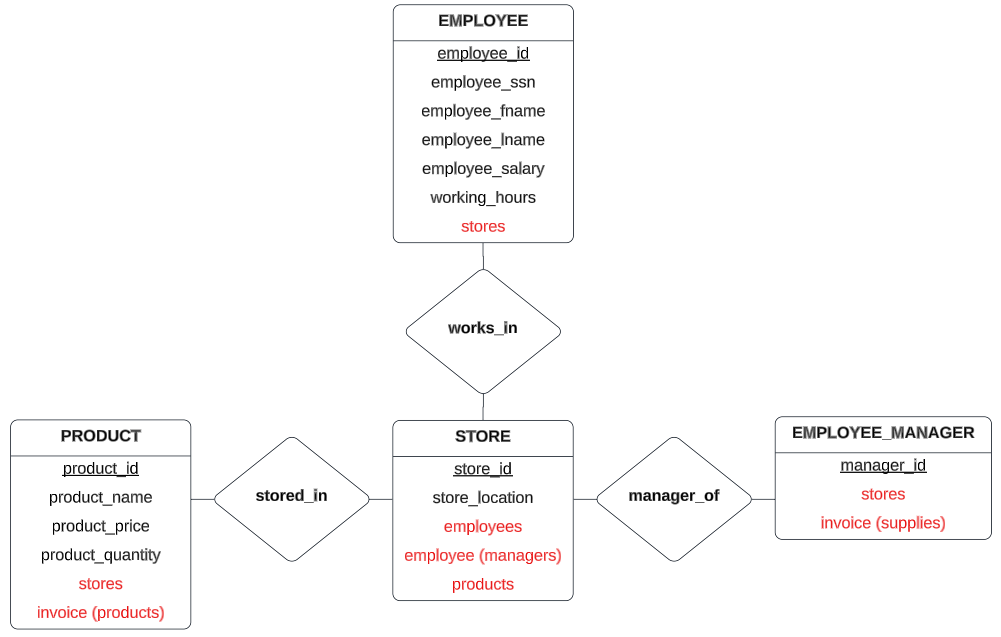
The reason why we split the employee entity type into subclasses is for almost the same reasons as stated for the INVOICE class hierarchy: It is easier to manage the data, for avoiding data redundancy and to have better querying.

Now we continue with the relationships between the entity types.

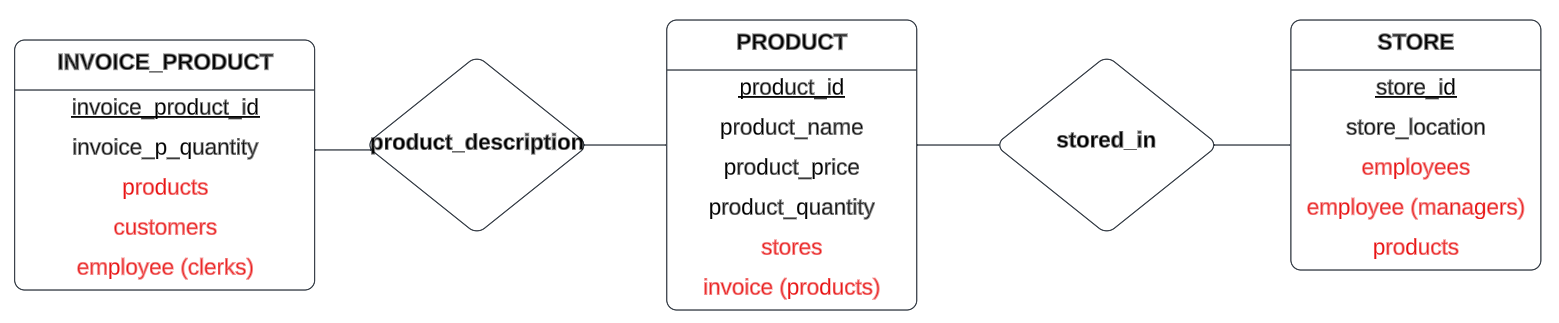
In the **EMPLOYEE** entity type we have an attribute which is not directly related to EMPLOYEE called stores. Since all employees must work in stores, we create a relationship called **works\_in** between EMPLOYEE and STORE.



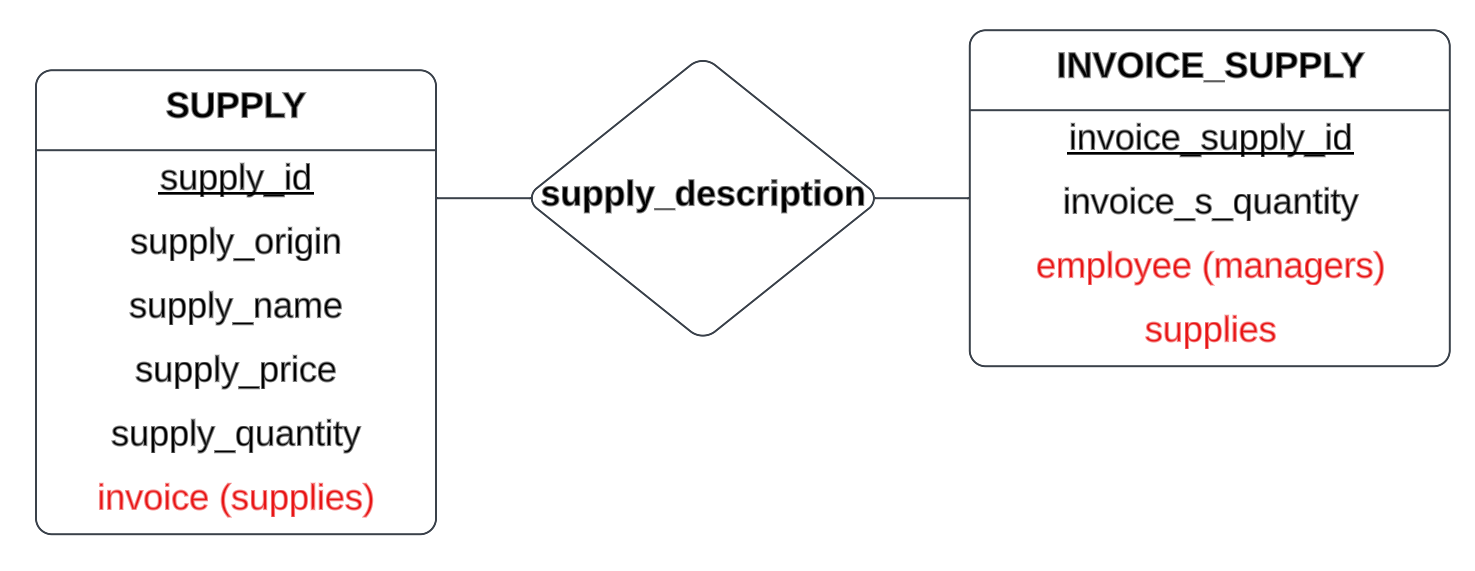
In the **STORE** entity type we see that besides employees we have 2 other attributes which are not directly related to STORE. They are employee (managers) and products. Since managers manage one store, we create the relationship called **manager\_of** between EMPLOYEE\_MANAGER and STORE. Also, since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



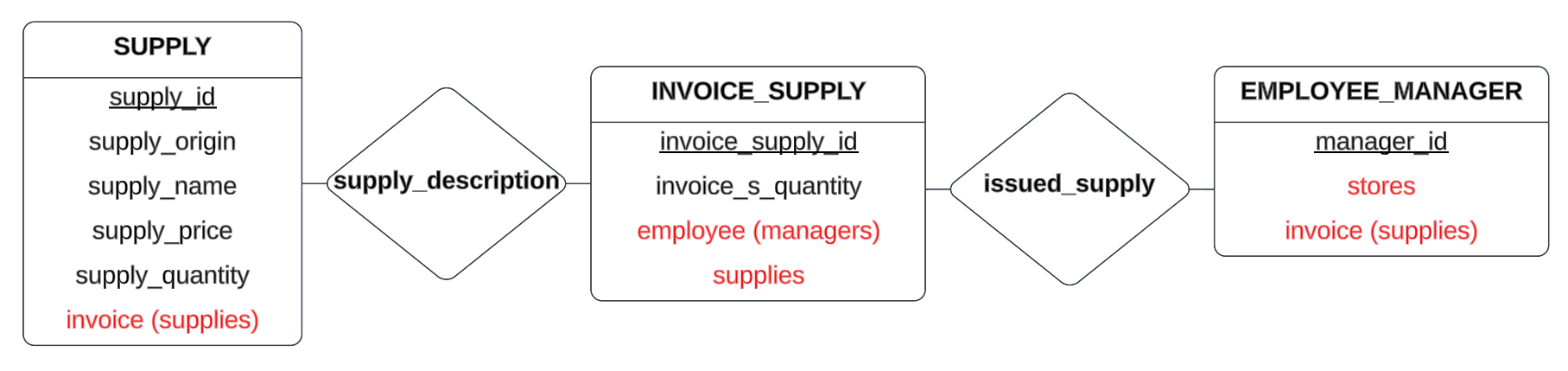
In the **PRODUCT** entity type we see that besides stores we have one other attribute which is not directly related to PRODUCT called invoice (products). Since product invoices should contain what products have been purchased, we create a relationship called **product\_description** between INVOICE\_PRODUCT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



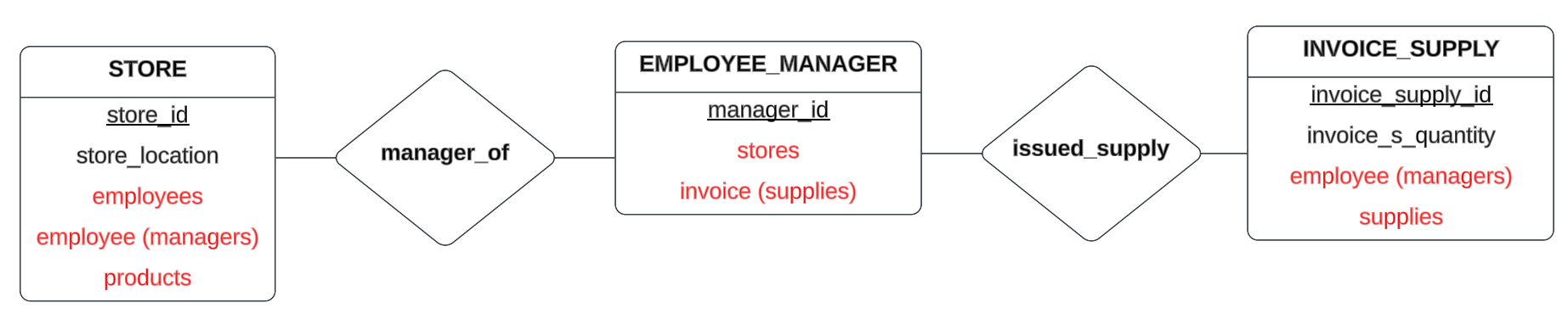
In the **SUPPLY** entity type we see that there is only one attribute which is not directly related to SUPPLY called invoice (supplies). Since supply invoices should contain what products (supplies) have been purchased, we create a relationship called **supply\_description** between SUPPLY and INVOICE\_SUPPLY.



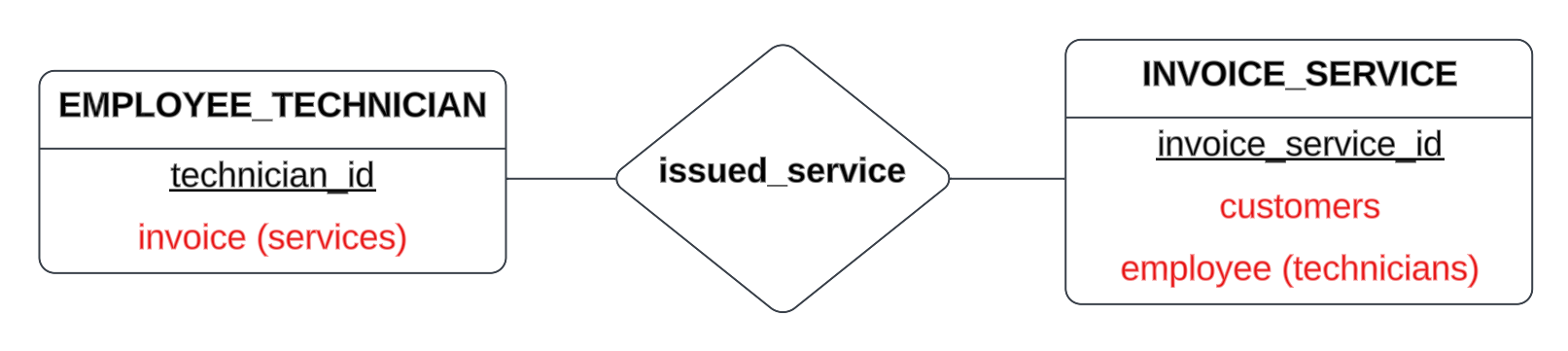
In the **INVOICE\_SUPPLY** entity type we see that besides supplies we have one other attribute which is not directly related to INVOICE\_SUPPLY called employee (managers). Since supply invoices should be issued by managers, we create a relationship called **issued\_supply** between INVOICE\_SUPPLY and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



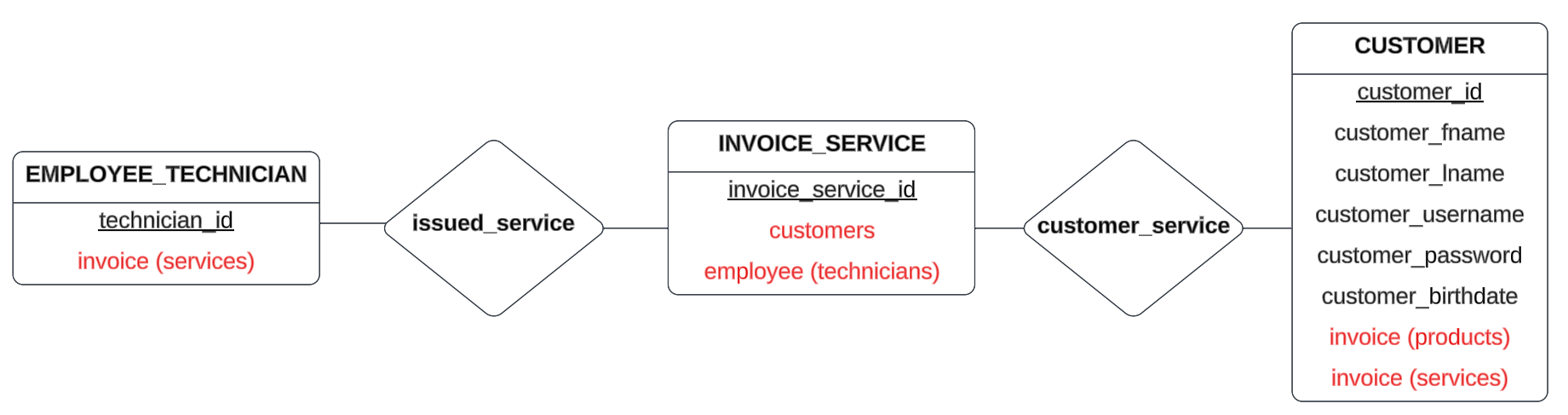
In the **EMPLOYEE\_MANAGER** entity type we have 2 attributes, stores and invoice (supplies), which are not directly related to EMPLOYEE\_MANAGER. Since the relationships **manager\_of** and **issued\_supply** have already been defined, we will only provide the following diagram:



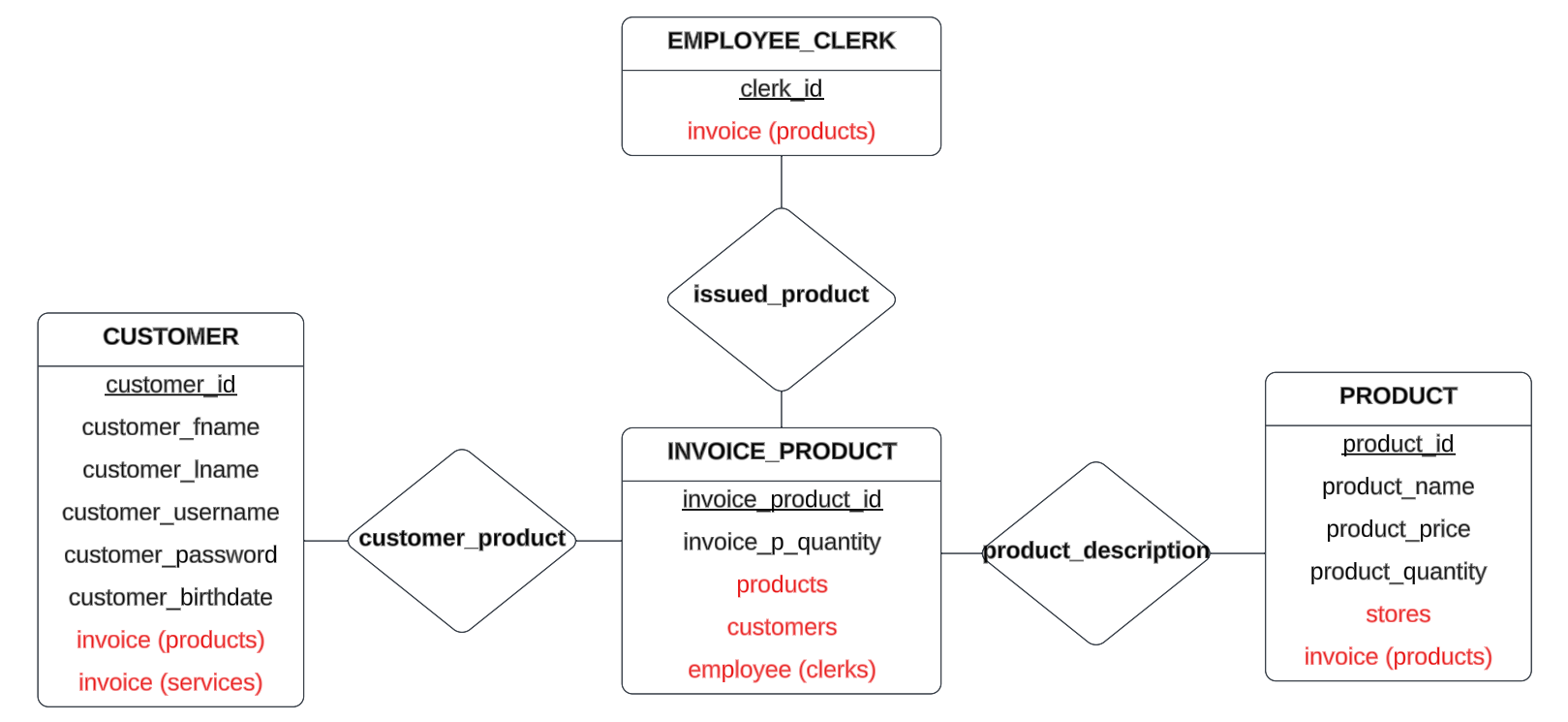
In the **EMPLOYEE\_TECHNICIAN** entity type we see that there is only one attribute which is not directly related to EMPLOYEE\_TECHNICIAN called invoice (services). Since technicians have to issue invoices when they service a computer, we create a relationship called **issued\_service** between EMPLOYEE\_TECHNICIAN and INVOICE\_SERVICE.



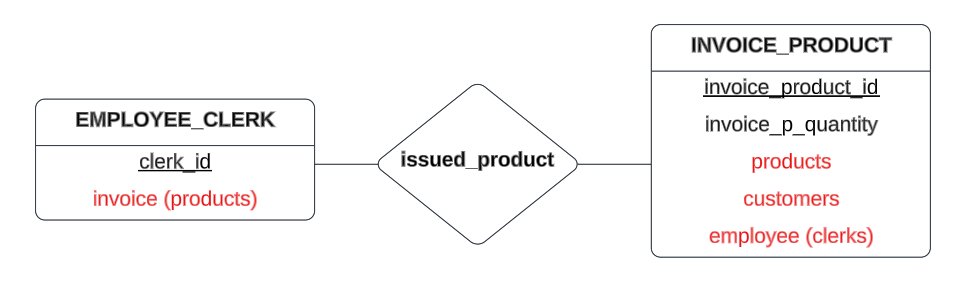
In the **INVOICE\_SERVICE** entity type we see that besides employee (technicians) we have one other attribute which is not directly related to INVOICE\_SERVICE called customers. Since customers receive invoices for their serviced computers, we create a relationship called **customer\_service** between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



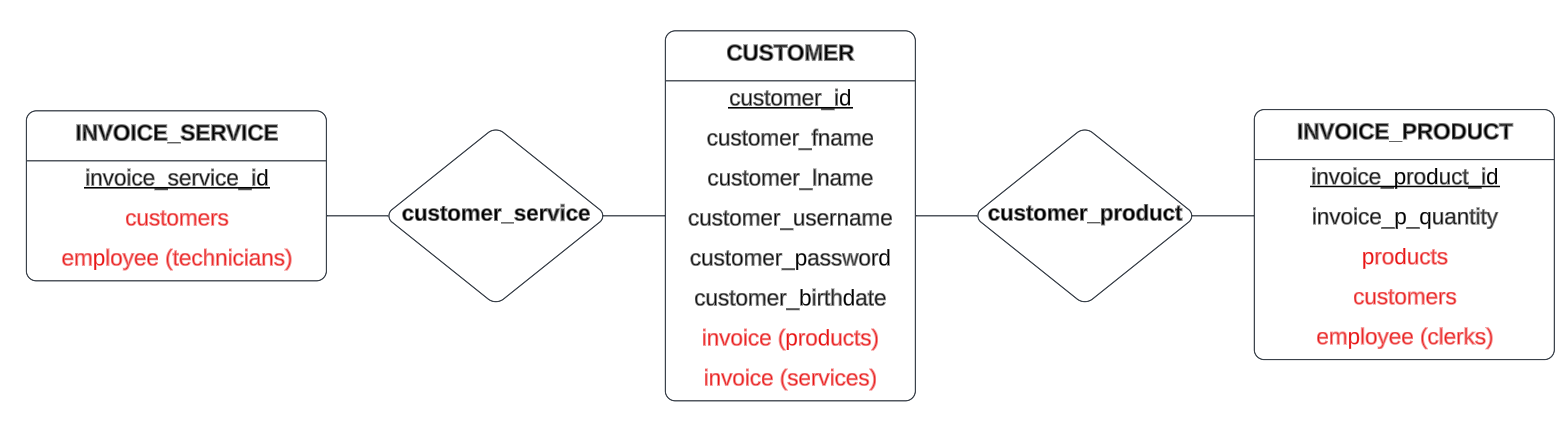
In the **INVOICE\_PRODUCT** entity type we see that besides products we have 2 other attributes which are not directly related to INVOICE\_PRODUCT called customers and employee (clerks). Since customers receive invoices for their purchased computers, we create a relationship called **customer\_product** 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.



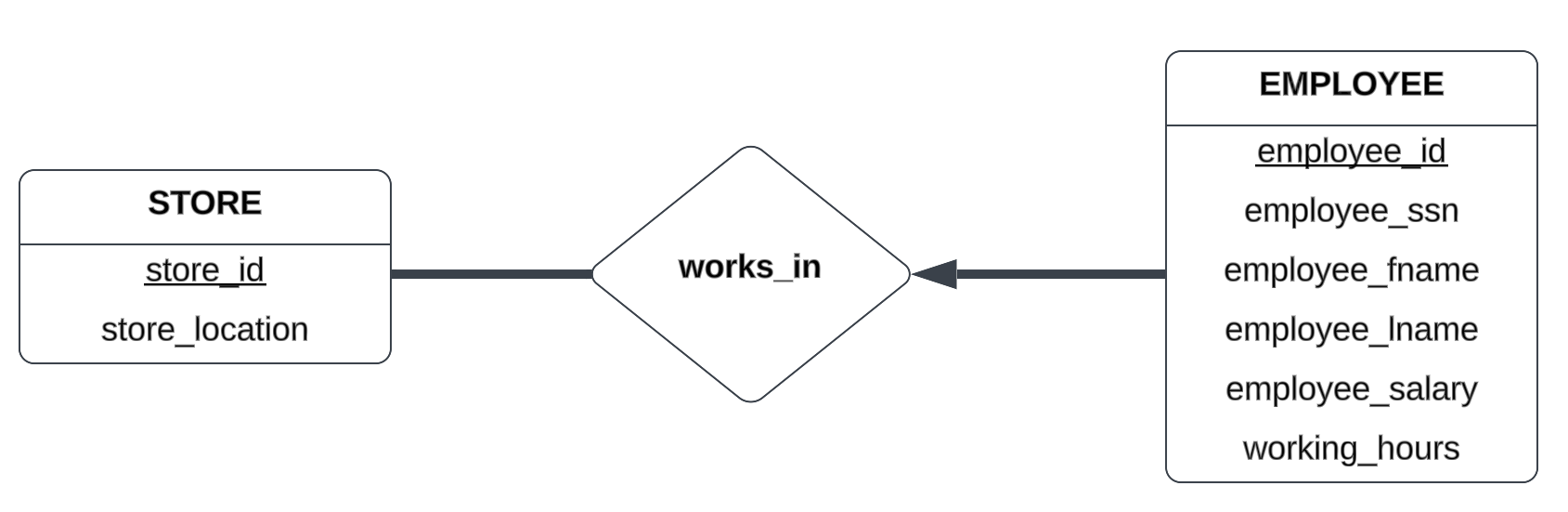
In the **CUSTOMER** entity type we have 2 attributes, invoice (products) and invoice (services), which are not directly related to CUSTOMER. Since the relationships **customer\_service** and **customer\_product** have already been defined, we will only provide the following diagram:



**STEP 3: Cardinality and participation**

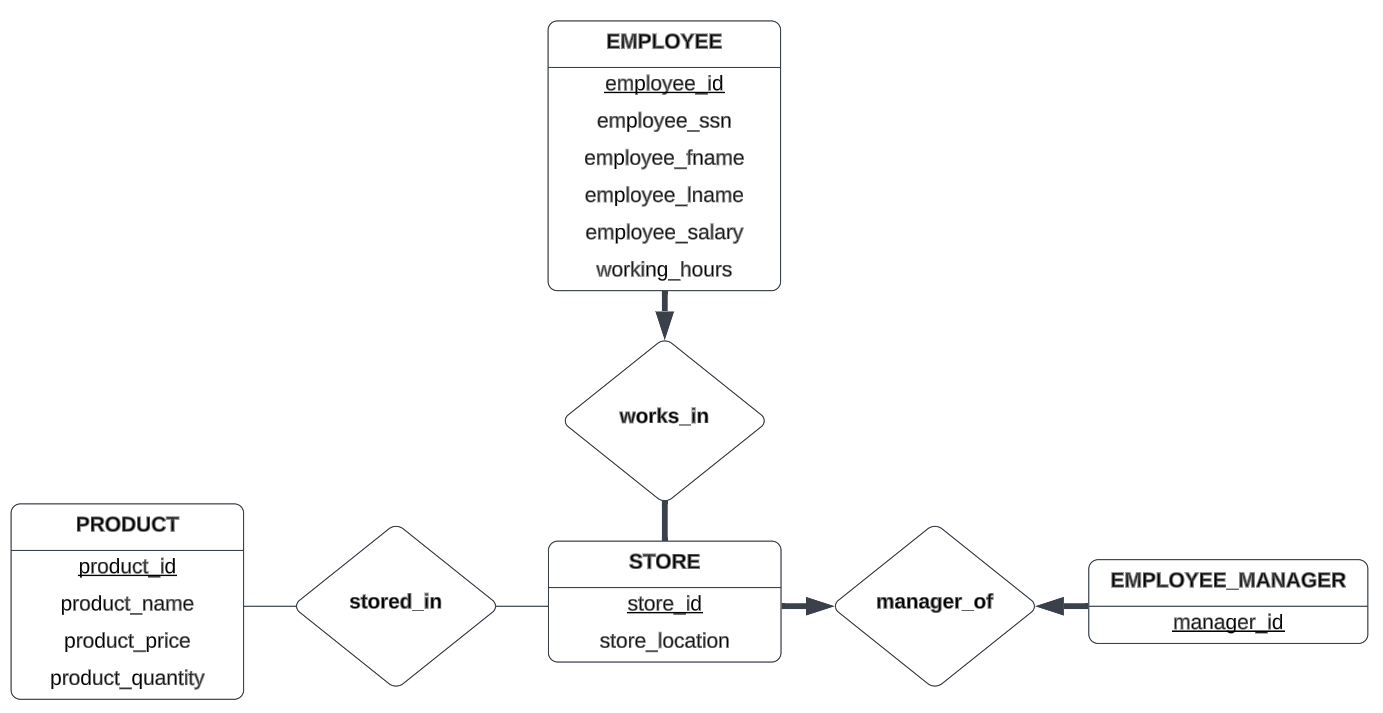
In this step of the conceptual design we define the constraints of the relationships by determining the cardinality and participation.

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.

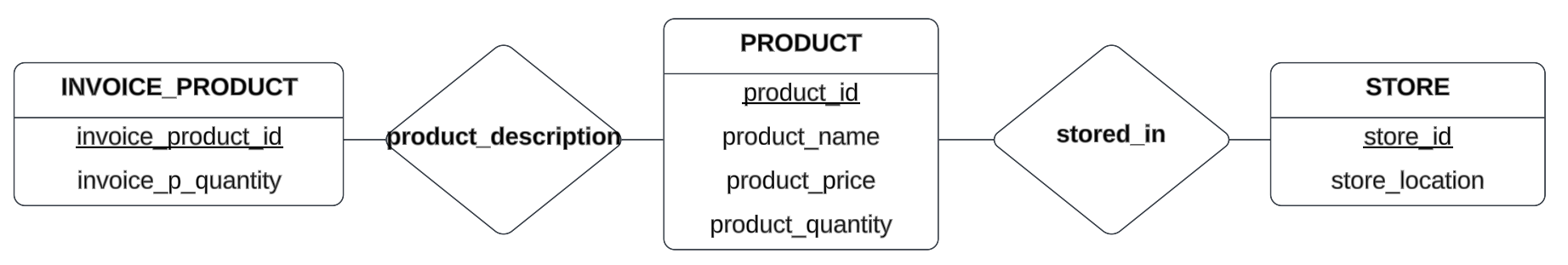


In the **stored\_in** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one many products can be stored in many stores and many stores can have many products.

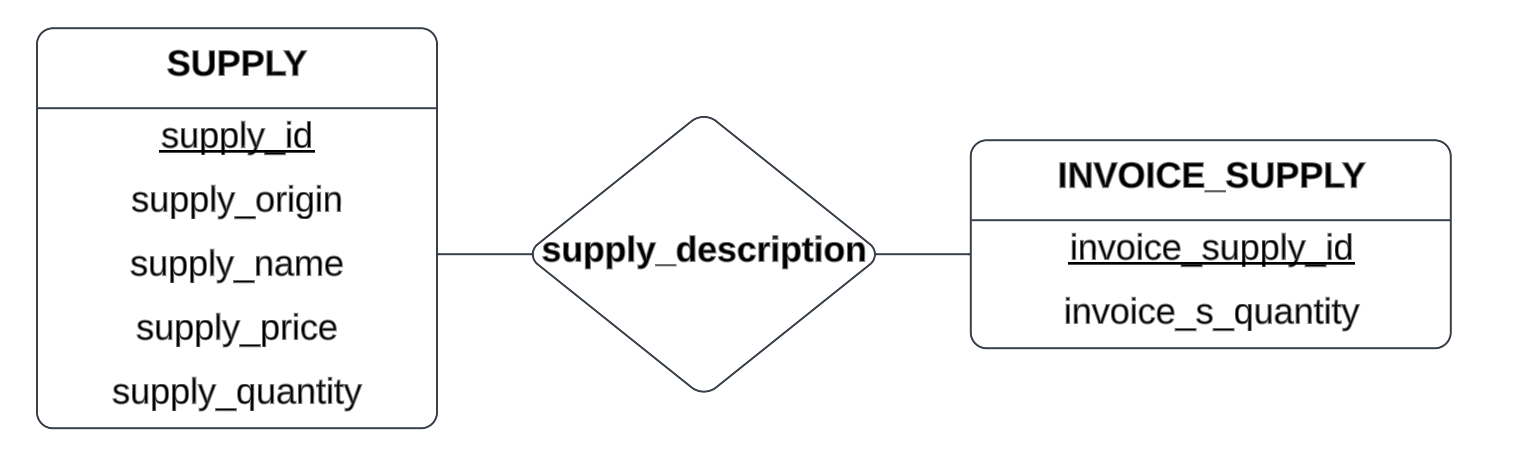
In the **manager\_of** relationship we have a one to one cardinality (1:1) with total participation on both sides. The reasoning behind this is that one store must have only one manager (can’t have no manager) and one manager can manage only one store (can’t manage no stores).



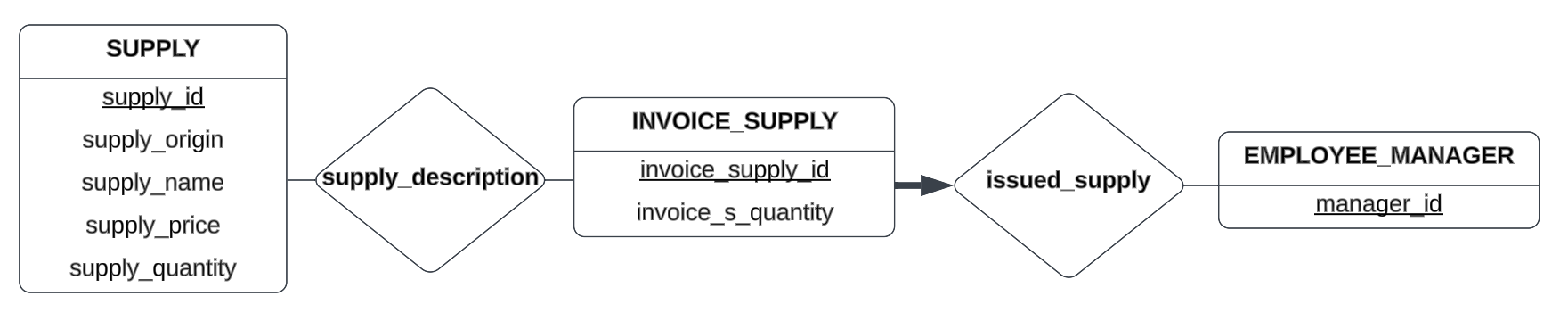
In the **product\_description** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one product can appear on many product invoices and one product invoice can have many products.



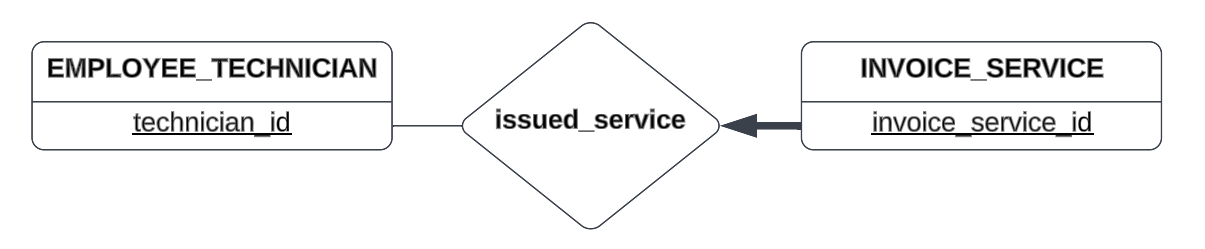
In the **supply\_description** relationship we have a many to many cardinality (M:N) with partial participation on both sides. The reasoning behind this is that one supply (product which the manager purchases) can appear on many supply invoices and one invoice can have many supplies.



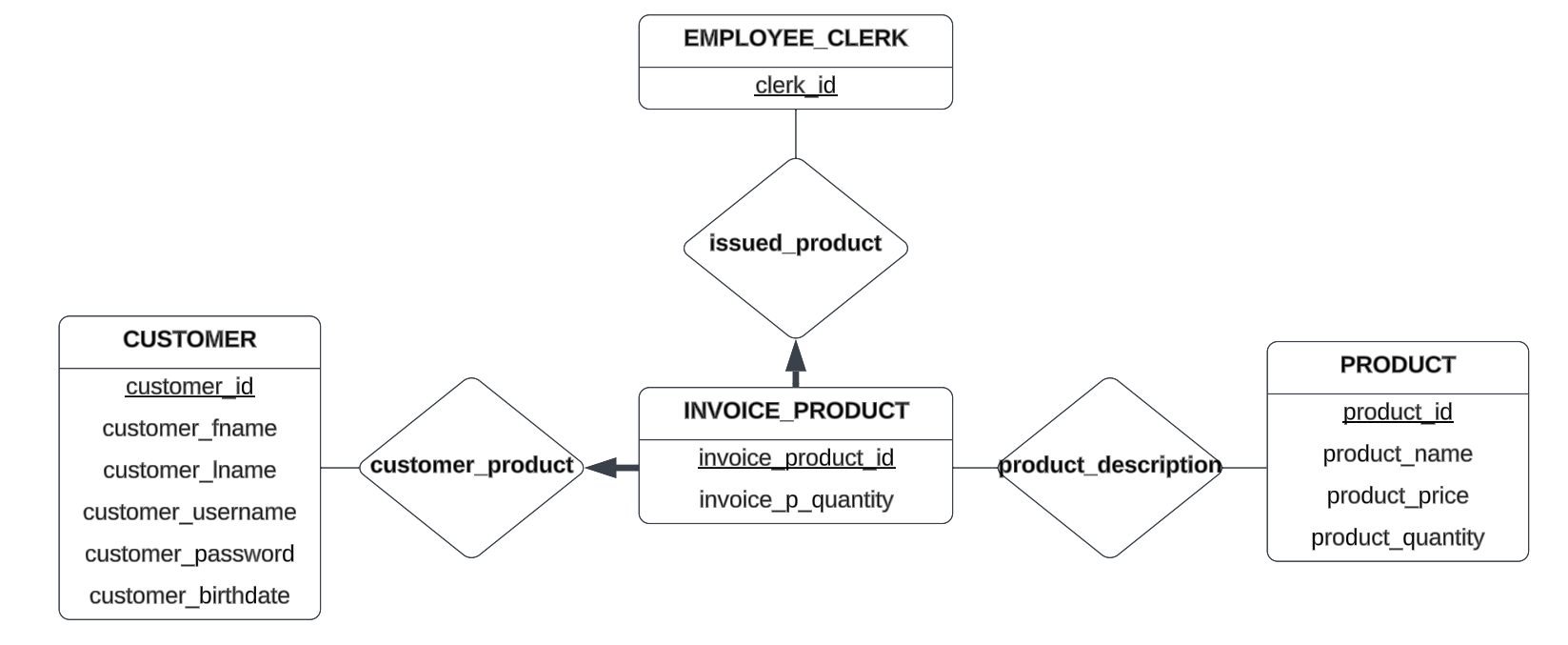
In the **issued\_supply** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE\_SUPPLY side. This means that one supply invoice can be issued by (belong to) one manager, whereas one manager can issue many supply invoices.



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.

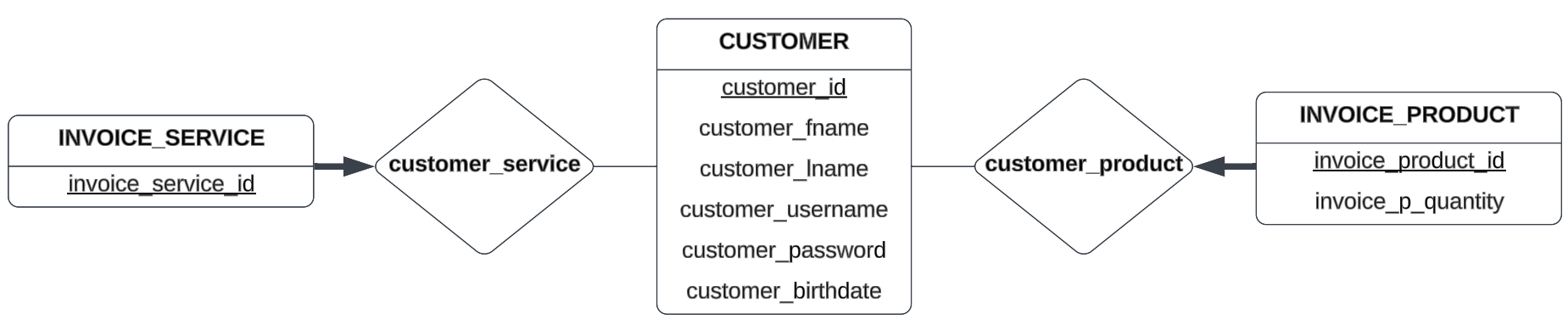


In the **issued\_product** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE\_PRODUCT side. This means that one product invoice can be issued by one clerk, whereas one clerk can issue many product invoices.



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).

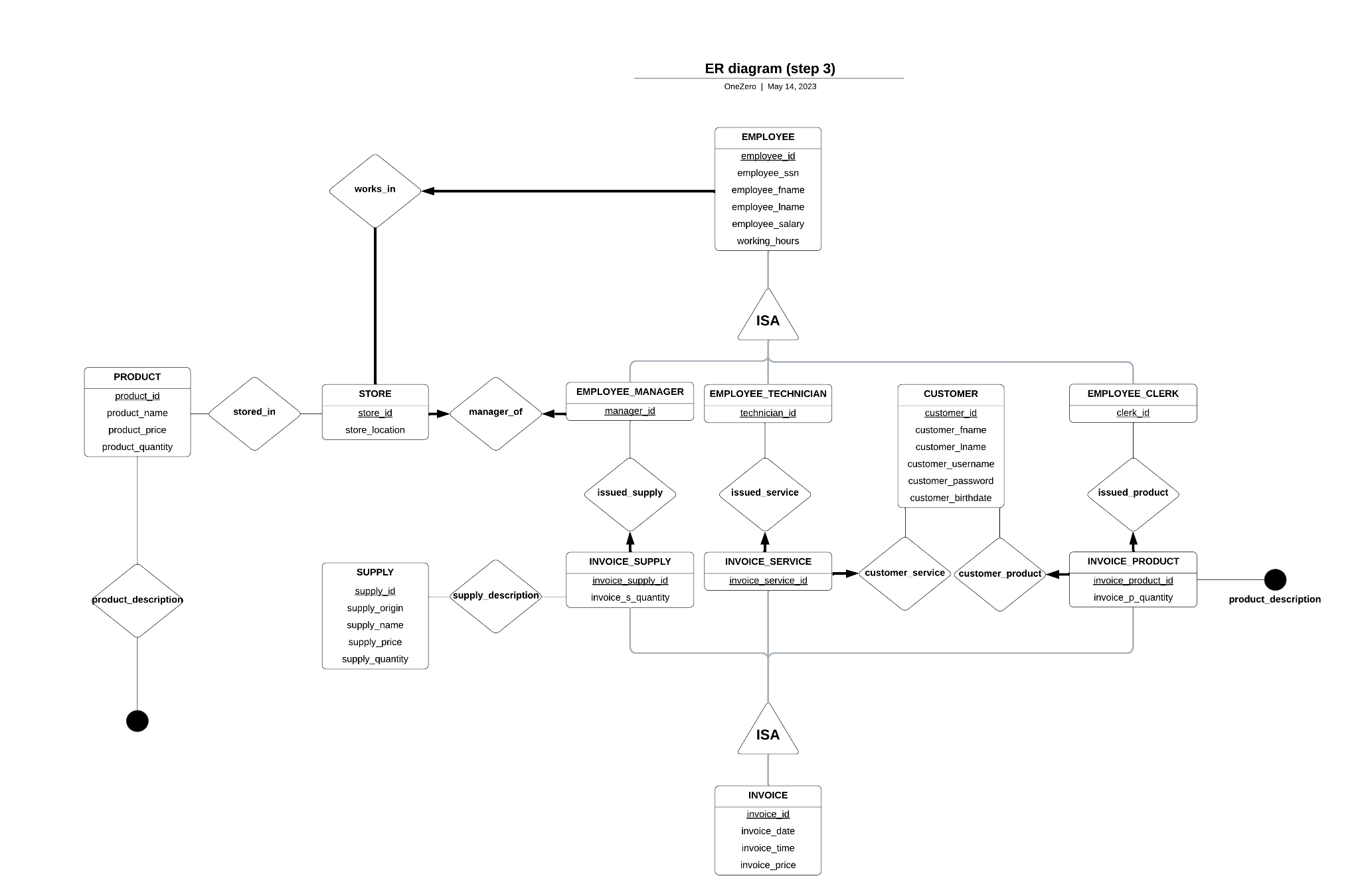
In the **customer\_product** relationship we have a one to many cardinality (1:N) with the key constraint and total participation on the INVOICE\_PRODUCT side. This means that one product invoice can belong only to one customer, whereas one customer can issue many product invoices (buy many products).



**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**

**Step 1: Strong entity types**

EMPLOYEE

| employee\_id | employee\_ssn | employee\_fname | Employee\_lname | employee\_salary | working\_hours |
| --- | --- | --- | --- | --- | --- |

STORE

| store\_id | store\_location |
| --- | --- |

PRODUCT

| product\_id | product\_name | product\_price | product\_quantity |
| --- | --- | --- | --- |

SUPPLY

| supply\_id | supply\_origin | supply\_name | supply\_price | supply\_quantity |
| --- | --- | --- | --- | --- |

CUSTOMER

| customer\_id | customer\_fname | customer\_lname | customer\_username | customer\_password | customer\_birthdate |
| --- | --- | --- | --- | --- | --- |

INVOICE

| invoice\_id | 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

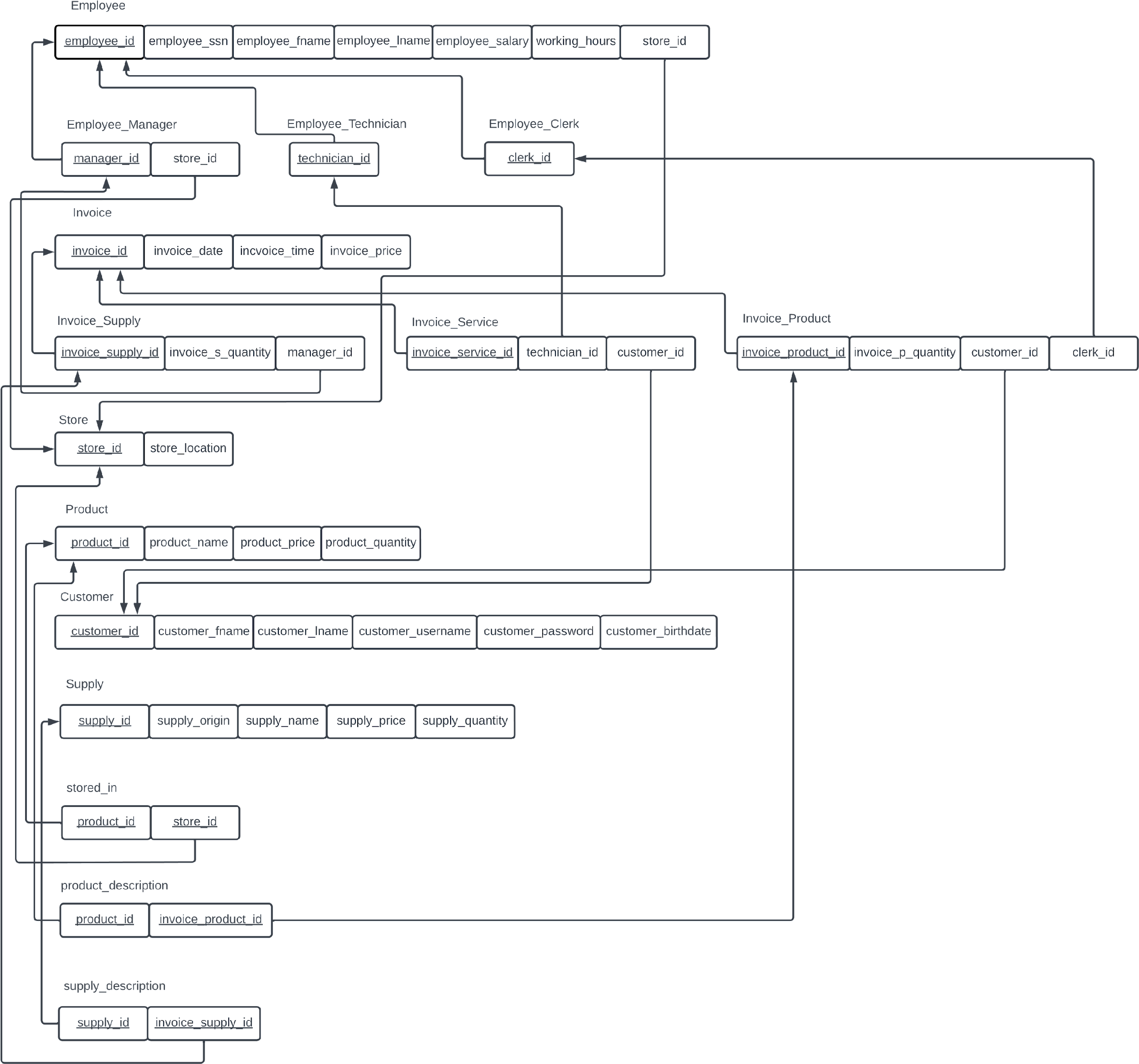
| invoice\_product\_id | invoice\_p\_quantity | customer\_id |
| --- | --- | --- |

**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**



**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 |
| --- |