UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Foundations of Databases A.Y. 2021-2022
Homework 2 – Conceptual and Logical Design

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: November 26, 2021

Team acronym	TAGMS	
Last Name	**First Name**	**Student Number**
Arslan	Tunahan	2023640
Basso	Marco	2005796
Cimarosto	Pietro	2027173
Collado	Martin	2039907
Esposito	Vittorio	2005795
Giuliani	Amedeo	2005797
Quiroz	Giannina	2041427
Zanini	Samuele	2019038

Conceptual Design

Variations to the Requirement Analysis

An employee must necessarily have one and only one role. A role is typically associated with multiple employees but may not have any. For example, the "intern" role can be created but without having hired anyone in that position yet.

An employee must work in at least one department. Each department can have multiple workers, but none if the department is new and newly created.

A manager is an employee with role equals to "Manager". Only they are allowed to stipulate one or more contracts with many suppliers. A supplier can stipulate many contracts with the manager as well, but a new supplier can also not have stipulated a contract yet. Our system, therefore, is able to also store information relating to suppliers with whom the company has not yet entered into agreements.

Each contract specifies one or more items (e.g., ingredients, packaging materials, etc.) and the respective quantity. An item can be specified in at least one contract, so it can be provided by many different suppliers. Each item belongs to only one item category, and to an item category can belong zero or many items. The stock quantity of each item is tracked. It will therefore be increased upon receipt of the items, after purchase from one or more suppliers, and decreased upon the preparation of lots of products that are made up of the items. The delivery date of the items is specified in each contract, so that the respective quantity in stock is automatically updated only upon arrival of the goods. The company does not use periodic delivery contracts, so every time it becomes necessary to purchase certain ingredients or packaging material, a new contract is signed with the supplier.

A product, which is a finished good ready to be sold, is made up of one or more items (e.g., one glass bottle, a hundred grams of sugar, a hundred milliliters of water, etc.) with the respective quantities. For example, a product called "Coke J" can consist of one aluminum can, 50ml of water, 10g of sugar, etc. Another product, called "Coke B" for example, may have the same ingredients as the previous example but can be packaged with a glass bottle. The expiration date of a product is specified in the various lots (that include that particular product) and it may differ in each lot. The stock quantity of a product is not explicitly specified, but it can be obtained by checking the specified product quantities in each lot not yet sold or shipped. An item can be utilized in many products (also none if, for example, the item is brand new). Each product belongs to one and only one product category, that is used to distinguish them. To a product category can belong zero or many products. The company decides the value of the price increase according to company policies. This increase ("Price_increase") must take on a value greater than or equal to 1. The price of a product is calculated as a multiplication between the cost of production and the price increase. The expiration date of the ingredients (items) is not tracked as the company guarantees to keep them stored for a short period of time because the ingredients are used shortly after their purchase and a FIFO policy is being implemented. It is not necessary to keep track of the current quantity of products in stock because, in those rare cases where such information is required (i.e. yearly cicle counting), it is possible to check the unsold and not expired lots. Furthermore, it is not necessary to keep track, for each product, of its minimum quantity in stock since most of them are produced on request (and not in advance) and therefore these would have a minimum quantity in stock equal to zero.

A package is composed of one or more packaging materials (i.e., an item used for packaging, such as a box, a meter of plastic tape, a kilogram of polystyrene, etc.) with the respective quantities. For example, a package named "PK1" can consist of 4 boxes of dimensions 30cm x 30cm x 10cm, 2 meters of plastic tape and 200 g of polystyrene. A "PK2" package can consist of 6 boxes of dimensions 30cm x 30cm x 10cm, 4 meters of plastic tape and 300 g of polystyrene. An item (e.g., packaging material in this case) can be utilized in many packages (also none if, for example, the package is brand new and not yet used). Each package belongs to one and only one package category, that is used to distinguish them. To a package category can belong zero or many packages. It is not necessary to explicitly track the current and minimum quantity of packages in stock as it is not in the interests of the company to do so.

In a lot there can be stocked a certain amount of only a product and a certain amount of only a package. The quantity of products in each lot depends both on the dimensions of the package of the individual product (e.g., bottle of glass) and on the features of the package (in particular, the size of the box and the number of

boxes that make up the package). Each type of product can be stocked in many lots (in none if, for example, the product is brand new) and the same holds true for packages. The price of a specific lot ("Lot_price") is calculated as the multiplication of the price of the product (stored in the lot) multiplied by its quantity. It is important to underline that the "Lot price" attribute can be derived only at the moment of the production as the fields on which it depends ("Production_Cost" and "Price_increase") can be modified over time and therefore give discordant results at different times. Each lot is also characterized by an expiration date. As some lots may be produced in advance to reduce lead times, some of them may not sell on time and therefore expire. The data analyst will perform half-yearly analysis in this regard to reduce waste. When a lot is produced, the company specifies the current price and VAT. The cost of the packages is not explicitly charged to the customer. A discount can also be associated with each lot. The company decides the total discount to apply to a specific lot. This discount expresses a percentage and is a number between 0 and 100. Furthermore, the company is able to take into account changes in VAT.

The "Quantity" attribute of the "Item" entity is derived. When a new contract is signed, in the delivery date (specified by the attribute "Delivery_date" of "Contract") the quantity of each item in the contract is incremented accordingly based on the "Purchased_Quantity" attribute in the relationship "Specify". This increase will be carried out autonomously by the application connected to the database on the day specified in "Delivery_date". When a new lot is inserted in the system, each quantity of each item, that constitute the product in the lot, is decremented by a number equal to the multiplications between "Product_quantity" in "Stocked" and "Quantity" in "Made_up_of(1)". The same applies to "Package" entity where the decreasing factor is calculated from the product between "Package_quantity" and "Quantity" in "Made_up_of (2)". Immediately after this, for each item involved, the "Quantity" attribute is compared with the "Minimum_quantity" attribute so a manager will get notified to avoid shortages. If the quantity of items were calculated from scratch (i.e., checking all the contracts and the composition of the products in the various lots) there would be no possibility for a worker to correct the quantity in stock of an item in the event of a mismatch with physical inventory control. Furthermore, the worker, before processing the order, verifies that there are sufficient quantities of items to satisfy the request.

The customer decides with the seller regarding the products to be bought. The salesman, then, after communicating the products (with respective quantity) to the warehouse worker, will place the order only when all the lots included in the order are ready. A seller can place zero or many orders for a customer, so a customer can make many orders (none if, for example, the customer is new). An order can be placed by only one salesman for only one customer (i.e., an order for a customer cannot be placed by two or more sellers, but by only one of them). An order includes one or more lots. Each lot can be included by only one order (none if the lot is produced in advanced and waiting to be ordered). The invoice will be automatically generated by the application linked to the system as soon as the order is placed. The total net amount of the order ("Net_price") must be calculated as the sum, for each lot i included, of the Lot_price_i * (1 - Lot_discount_i / 100). The total taxes are calculated as the sum, for each lot i, of the Lot_price_i * VAT_i / 100. When the order is ready, a worker will ship it: a worker can ship zero or many orders, and an order can be shipped by at most one worker (none if the order is waiting to be shipped). The cancellation and modification of the order is not accepted since the goods can be produced on commission and the company wants to minimize the waste caused by the expiry of the products. The customer, which is a business, must necessarily pay within 60 days and can be informed about the status of the order either by contacting the seller. Furthermore, through the tracking number received by email, the customer can monitor the shipping.

Entity-Relationship Schema

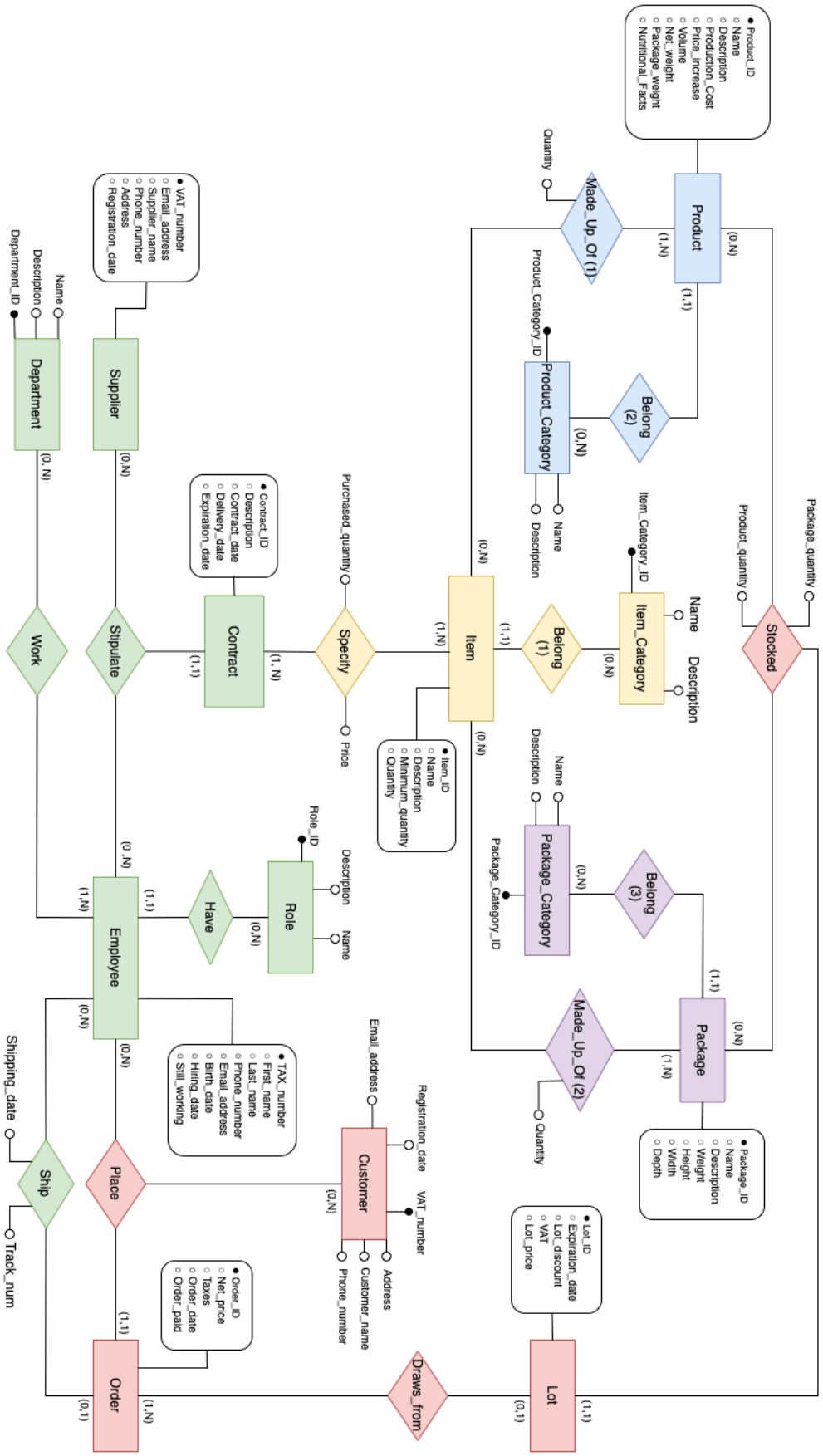

Figure 1: Entity-Relationship schema.

Data Dictionary

Entities Table

Entity	Description	Attributes	Identifier
Employee	Represents data of an employee who works in the company and needs access to the system	• TAX_number: TAX code of the employee, text • First_name: name of the employee, text • Last_name: surname of the employee, text • Phone_number: phone number (prefix included) of the employee, text • Email_address: email address of the employee, text • Birth_date: birthdate of the employee, Datetime • Hiring_date: hiring date of the employee, Datetime • Still_working: flag used to know if employee is still working for the company, boolean	TAX_number
Role	Represents data on the role of employees who work in the company	• Role_ID: role identifier, serial • Name: name of the role, text • Description: technical description of the role, text	Role_ID
Department	Represents data on the departments in which employees work	• Department_ID: department identifier of the company, serial • Name: name of the department, text • Description: description of the department's function, text	Department_ID

Customer	Represents data about a customer of the company	• VAT_number: VAT number of the customer, int • Customer_name: name of the customer, text • Phone_number: phone number (prefix included) of the customer, text • Address: billing address of the customer, text • Email_address: email address of the customer, text • Registration_date: customer registration date in the database, Datetime	VAT_number
Contract	Represents data about a contract stipulated between a supplier and a manager for the supply of items	• Contract_ID: contract identifier, serial • Description: description of the contract, text • Contract_date: date of signature of the contract with the supplier, Datetime • Delivery_date: expected date of delivery of the goods, Datetime • Expiration_date: expiration date of the contract with the supplier, Datetime	Contract_ID

Supplier	Represents data about a supplier of the company	• VAT_number: VAT number of the supplier company, int • Supplier_name: name of the supplier company, text • Phone_number: phone number (prefix included) of the supplier company, text • Email_address: email address of the supplier company, text • Address: address of the supplier company, text • Registration_date: recording date of the supplier company, Datetime	VAT_number
Order	Represents the order placed by a salesman for a customer	• Order_ID: order identifier, serial • Order_date: date in which the order has been processed, Datetime • Order_paid: status of the payment, boolean • Net_price: total net amount including discount (VAT excluded), float • Taxes: total amount of taxes to be payed, float	Order_ID

Lot	Represents a lot in the inventory of the company, containing final products and packaging	• Lot_ID: lot identifier, serial • Expiration_date: expiration date of the included products, Datetime • Lot_price: lot price (without the discount) at the time of sale, float • Lot_discount: percentage of discount of a lot, int • VAT: value added tax (percentage) at the time of sale, int	Lot_ID
Product	Represents the final product that is marketed	• Product_ID: product identifier, serial • Name: name of the product, text • Description: description of the product, text • Nutritional_Facts: description of nutritional facts of the specific product, text • Volume: volume of the product in milliliters, int • Net_weight: net weight of the product in grams, int • Package_weight: package weight of the product in grams, int • Production_cost: cost of the production for a product, float • Price_increase: price increase factor, float	Product_ID

Item	Represents materials provided by suppliers	• Item_ID: identifier of the item, serial • Name: name of the item, text • Description: description of the item, text • Quantity: stock quantity of the item (automatically updates), int • Minimum_quantity: minimum stock quantity of the item, int	Item_ID
Package	Represents packaging of finished products which are made up of boxes, tapes, and other packaging materials	• Package_ID: identifier of the package, serial • Name: name of the package, text • Description: description of the package, text • Weight: weight dimension of the package in grams, int • Height: height dimension of the package in centimeters, int • Width: width dimension of the package in centimeters, int • Depth: depth dimension of the package in centimeters, int	Package_ID
Product_Category	Represents the category of a product	• Product_Category_ID: identifier of the product category, serial • Name: name of the product category, text • Description: description of the product category, text	Product_Category_ID

Item_Category	Represents the category of an item	• Item_Category_ID: identifier of the item category, serial • Name: name of the item category, text • Description: description of the item category, text	Item_Category_ID
Package_Category	Represents the category of a package	• Package_Category_ID: identifier of the package category, serial • Name: name of the package category, text • Description: description of the package category, text	Package_Category_ID

Relationships Table

Relationship	Description	Component Entities	Attributes
Have	Relates each employee to a role	• Employee (1,1) • Role (0,N)	None
Work	Assigns each employee to a department	• Employee (1,N) • Department (0,N)	None
Stipulate	Links the supplier with the company and the contract stipulated	• Supplier (0,N) • Employee (0,N) • Contract (1,1)	None
Place	Links the order made by the employee	• Employee (0,N) • Order (1,1) • Customer (0,N)	None

Ships	Relates the employee shipping the order with the order itself and the shipment details	• Employee (0,N) • Order (0,1)	• Track_num: tracking code of the shipment provided by the external shipment service, int • Shipping_date: date on which the company ships the goods, Datetime
Specify	Describes which items are provided by a contract	• Contract (1,N) • Item (1,N)	• Purchased_quantity: the quantity of items which are purchased, int • Price: the price of the amount of items which are purchased, float
Belong (1)	Links items to the category	• Item (1,1) • Item_Category (0,N)	None
Belong (2)	Links products to the category	• Product (1,1) • Product_Category (0,N)	None
Belong (3)	Links packages to the category	• Package (1,1) • Package_Category (0,N)	None
Made up of (1)	Describes what items are involved in creating the product	• Item (0,N) • Product (1,N)	• Quantity: the quantity of items composing a specific product
Made up of (2)	Describes what items are involved in creating the package	• Item (0,N) • Package (1,N)	• Quantity: the quantity of items composing a specific package

Stocked	Specifies the products and packages stocked in the lots	• Package (0,N) • Product (0,N) • Lot (1,1)	• Product_quantity: quantity of the included product, int • Package_quantity: quantity of the included package, int
Draws from	Associates the lots to an order	• Order (1,N) • Lot (0,1)	None

External Constraints

- All the units of measure used by the company are fixed. Furthermore, for sake of simplicity, they are reported in detail in the entity table. All the items stored in the system follow these conventions.
- Only employees that are sill working in the company can access the system.
- Only the manager can insert new contracts, so the employee who takes part in the "stipulate" relationship must have the role equal to "Manager".
- Only the salesman can insert new orders, so the employee who takes part in the "place" relationship must have the role equal to "Salesman". Only the salesman can update the order status by changing the attribute "Order_paid" which by default is set to false.
- Only the worker can ship orders, so the employee who takes part in the "ship" relationship must have the role equal to "Worker".
- A product is made up of one or more items taken from a certain subset, i.e. having a certain "Item_Category" (e.g., "ingredient" or "glass bottle"). Furthermore, a package is made up of one or more items taken from a certain subset, i.e. having a certain "Item_Category" (e.g., "box", "plastic tape" or "polystyrene"). As an example, in the list of items there can be both "Sugar" and "Box", but the former can appear only in relationships with the "Product" entity, while the latter only with the "Package" entity.
- An expired lot cannot be sold.

Functional Requirements Satisfaction Check

The DBMS has to be able to:

- **store all the details of the employees, customers and suppliers in the organization:** Employee entity stores data related to the employees. Customer entity has details about the customers and Supplier entity has data related to suppliers.
- **allow the employees to update their personal information:** Employee entity has some attributes as Email_address, Password or Phone_number that can be changed. Employees can access the system using their credentials and change this data.
- **store details of all on-hand products in the inventory such as item code, item description, quantity and expiration date:** The inventory is represented by the entities Item, Product, Package and Lot. The Item entity contains information relating to materials (e.g., ingredients, packaging materials, ...) with their respective descriptions and quantities. Likewise, the Product entity contains information relating to

finished products and the Package entity contains information relating to packaging. The products are packed in lots. Each lot is also characterized by an expiry date.

- **allow the employees to log into the system and enter the inbound items they received with information item code, item description, quantity, expiration date and supplier:** Employees can log in the database and insert data about new items in the system. An employee can also update an existing Item and its respective quantity.
- **show and generate the list of inbound and outbound transactions:** The inbound transactions can be generated by inspecting the instances of the Contract entity, while the outbound transactions can be obtained by inspecting the instances of the Order entity.
- **allow the employees to log into the system and enter the outbound transaction needed for the issuance of the products in the production and shipment to the customers; inventory stocks will be automatically updated whenever there are inbound and outbound transactions; show and generate the current inventory balance or stock inquiries:** Regarding items, the update is executed automatically when an inbound transaction occurs by inspecting the new Contract: for each Item the quantity "Item_Quantity" is increased accordingly. Regarding the outbound transaction, the value of "Item_Quantity" is decreased when a new lot (which stocks a product that is made up of that Item) is prepared. Regarding products, the stock quantity can be obtained by inspecting the lots produced but not yet ordered or expired. The same holds true for packages.
- **receive and process the Customers order, specifying which products they want and respective quantity:** Salesmen are able to access the database and enter an instance of the Order entity reporting the lots containing the desired products only when all lots are ready.
- **allow users to view order and shipment status of finished products; create tracking code for orders:** With the unique tracking number (attribute "Track_num" of the Ship relationship), and the unique ID attribute of the Order entity, the users can get information about the order and shipment.
- **generate invoice whenever payment has been made:** When an order is placed, the invoice is automatically generated by the application connected to the system. The data is extracted from the entities Order and Lot, and from the "Draws from" relationship. The total amount, net price, taxes and the list of ordered Lots are specified.
- **grant Cycle Counting in order to validate the accuracy of inventory:** Cycle counting is a periodic check done by a warehouse worker on the items in the physical inventory. After acquiring the real quantities for each item, a check will be made on the system. In case of mismatch, the "Quantity" attribute of the Item is updated.
- **re-ordering the previous orders is allowed:** The system allows salesmen to access past orders and lots using the ID attribute and retrieve information about the lots, the products, and their quantities. In this way, the customer can order the same goods.

Logical Design

Transformation of the Entity-Relationship Schema

Redundancy Analysis

The Employee and Order entities, related to each other through the relationships Place and Ship, do not form a cycle because:

- Only the Employee with the Salesman role can place the order;
- Only the Employee with the Worker role can ship the order;
- Eliminating the Place or Ship relationship implies a violation of functional requirements and a loss of information (e.g., who is the seller who places the order or who is the worker who ships it).

The Lot, Product, Package, and Item entities do not form a loop because:

- Eliminating a "Made_up_of" relationship implies a violation of functional requirements and a loss of information (eg, from which items, and in what quantity, a product or package is composed);
- Eliminating the "Stocked" relationship implies a violation of functional requirements and a loss of information (e.g., it is not possible to know which products and packages are contained in a lot).

The ER schema presents the derivate attributes:

- "Quantity" (of Item): used to keep in memory the current quantity of items;
- "Net_price" (of Order): used to keep in memory the total net price of an order;
- "Taxes" (of Order): used to keep in memory the total taxes of an order.

We report below the analysis of the database load to check whether keeping these attributes or not.

Choice of Principal Identifiers

The schema does not contain external identification cycles and the main identifiers comply with the selection criteria.

Analysis of Database Load

The load analysis is divided in two parts: the first, to show that storing the stock quantity of items in the Quantity attribute of Item requests less operations than computing it when needed; the second, to show that storing the net price and taxes of an order is more convenient than computing them when needed. The operations O_1, O_2, and O_3 are used to describe the first part, while the operation O_4 and O_5 are about the second part.

Operation	Description	Frequency	Type
O_1 : Insert contract	Store data about a new contract.	1/week	Online
O_2 : Insert new lot	Store data about a newly packaged lot.	25/week	Online
O_3 : Get item quantity	Get quantity for one type of item	25/week	Online
O_4 : Insert new order	Create a new order, which is made of several lots	5/week	Online
O_5 : Get order price	Get the price for one order	5/week	Online

To solve the first part, it is necessary to calculate the cost of entering the contract plus the cost of entering the lot plus the cost of obtaining the quantity of an item. The calculation must be done in two cases: in the first case (redundant) the Quantity in Item attribute is present, while in the second case (not redundant) the attribute is not present.

Regarding the redundant case, it is necessary to take into account the updating of the attribute, the cost of entering the contract and the cost of entering the lot. During the insertion of the contract a write operation in Contract is made and then an access to insert different instances on Specify is performed. For each instance of Specify, a write access is made to the Item entity to increment the value of the Quantity attribute. Compared to the non-redundant case, the insertion of the contract is more expensive.

During the insertion of the lot, one write operation in Lot and one in Stocked are carried out. We look for the product in Made_up_of (1) to understand which items it consists of. For each instance of Made_up_of (1), using the quantity specified in it, the Item is searched and its quantity is decremented. Similar procedure for packages with Made_up_of (2). Compared to the non-redundant case, lot insertion is more expensive.

To obtain the quantity in stock of an Item, it is necessary to perform only one read on the Item. Compared to the non-redundant case, this operation is much less expensive.

Regarding the non-redundant case, the updating of the attribute does not need to be taken into account. Furthermore, the cost of entering the contract and that of entering the lot are lower. The cost of obtaining the quantity of an item in stock is much higher.

During the insertion of the contract a write operation in Contract is made and then an access to insert different instances on Specify is performed. Compared to the previous case, it is not necessary to make one or more accesses to the Item entity. The insertion of the contract is therefore less expensive. During the insertion of the lot, one write operation in Lot and one in Stocked are carried out. Compared to the redundant case, lot insertion is less expensive.

To obtain the quantity in stock of an item, two operations are necessary: the first allows to trace the total quantity purchased of a specific item. The second instead allows you to calculate the quantity of the item that was used for the production of the lots.

To trace the total purchased quantity of an Item, it is necessary to access all instances of the relationship Specify to understand in which contracts the Item was purchased. Then, for each Specify instance, the Contract entity is accessed to verify that the contract has "Delivery_date" prior to today's date. In this case the Item has been delivered and the purchased quantity "Purchased_quantity" in Specify is taken into account.

To calculate the quantity of Item that has been used for the production of the lots, it is necessary to access the relationship Stocked to understand which products and packages (with relative quantities) a lot is made up of. For each instance of Stocked, the Product_ID is used to check (through Made_up_of (1)) the composition of the product and Package_ID to check (through Made_up_of (2)) the composition of the package. Then, for each instance of Stocked, an access is made to the relationship Made_up_of (1) and, if the Product consists of the Item of interest, the product between "Quantity" in Made_up_of (1) and "Product_quantity" is taken into account in Stocked. Furthermore, for each Stocked instance, an access is made to the relationship Made_up_of

(2) and, if the Package consists of the Item of interest, the product between "Quantity" in Made_up_of (2) and "Package_quantity" is taken into account in Stocked.

As for the second part (show that storing the net price and taxes of an order is more convenient than computing them when needed) proceed as follows. In both cases, redundant or otherwise, as soon as the order is placed it is necessary to calculate both the net price and taxes so that they can be viewed by the customer. The necessary operation is to show the net sales and the total taxes of an order.

In the non-redundant case, the "Net_price" and "Taxes" attributes are not available in the Order entity. It is necessary, for an instance of the Order entity, to access the relationship Draws_from to understand which lots the order is made up of. For each lot, and therefore for each instance of Draws_from, the entity Lot is accessed to obtain the price and any discount, but also the taxes.

In the redundant case, there are the "Net_price" and "Taxes" attributes in the Order entity. When the order is placed, these attributes are calculated as follows. Having the Order_ID, it is necessary to access the relationship Draws_from to understand which lots the order is made up of. For each instance of Draws_from, if the instance is referred to the Order of interest, then the Lot instance is accessed to obtain the price and any discount, but also the taxes To perform the desired operation, for each Order instance, simply access the attributes described above.

Table 4: O_1 Without redundancy

Concept	**Construct**	**Access**	**Type**	**Average Access**
Contract	Entity	1	W	1 x 1 x 2 = 2
Specify	Relationship	2	W	2 x 1 x 2 = 4
Total Access				**6**

Where the access value ("2") in "Specify" is the average number of (different) items in each contract.

Table 5: O_1 With redundancy

Concept	**Construct**	**Access**	**Type**	**Average Access**
Contract	Entity	1	W	1 x 1 x 2 = 2
Specify	Relationship	2	W	2 x 1 x 2 = 4
Item	Entity	2	W	2 x 1 x 2 = 4
Total Access				**10**

Where the access value ("2") in "Specify" and in "Item" is the average number of (different) items in each contract

Table 6: O_2 Without redundancy

Concept	**Construct**	**Access**	**Type**	**Average Access**
Lot	Entity	1	W	1 x 25 x 2 = 50
Stocked	Relationship	1	W	1 x 25 x 2 = 50
Total Access				**100**

Table 7: O_2 With redundancy

Concept	Construct	Access	Type	Average Access
Lot	Entity	1	W	1 x 25 x 2 = 50
Stocked	Relationship	1	W	1 x 25 x 2 = 50
Made_up_of (1)	Relationship	300	R	300 x 25 x 1 = 7500
Item	Entity	10	W	10 x 25 x 2 = 500
Made_up_of (2)	Relationship	100	R	100 x 25 x 1 = 2500
Item	Entity	3	W	3 x 25 x 2 = 150
Total Access				**10750**

Table 8: O_3 Without redundancy

Concept	Construct	Access	Type	Average Access
Specify	Relationship	200	R	200 x 25 x 1 = 5000
Contract	Entity	5	R	5 x 25 x 1 = 125
Stocked	Relationship	300	R	300 x 25 x 1 = 7500
Made_up_of (1)	Relationship	300	R	300 x 25 x 1 = 7500
Made_up_of (2)	Relationship	100	R	100 x 25 x 1 = 2500
Total Access				**22625**

Table 9: O_3 With redundancy

Concept	Construct	Access	Type	Average Access
Item	Entity	1	R	1 x 25 x 1 = 25
Total Access				**25**

Table 10: O_4 Without redundancy

Concept	Construct	Access	Type	Average Access
Draws from	Relationship	5	W	5 x 25 x 2 = 250
Order	Entity	1	W	1 x 100 x 2 = 200
Total Access				**450**

Table 11: O_4 With redundancy

Concept	Construct	Access	Type	Average Access
Draws from	Relationship	5	W	5 x 25 x 2 = 250
Order	Entity	1	W	1 x 100 x 2 = 200
Total Access				**450**

Table 12: O_5 Without redundancy

Concept	Construct	Access	Type	Average Access
Draws_from	Relationship	10000	R	10000 x 1 x 1 = 10000
Lot	Entity	5	R	5 x 1 x 1 = 5
Total Access				**10005**

Table 13: O_5 With redundancy

Concept	Construct	Access	Type	Average Access
Order	Entity	1	R	1 x 25 x 1 = 1
Total Access				**25**

Table 14: Comparison of item quantity w/ and w/o redundancy

Operation	With Redundancy	Without Redundancy
O_1	10	6
O_2	10750	100
O_3	25	22625
Total access/week	**10785**	**22731**

Table 15: Comparison of the order price w/ and w/o redundancy

Operation	With Redundancy	Without Redundancy
O_4	450	450
O_5	25	10005
Total access/week	**475**	**10455**

We can see that, from the sum of O_1, O_2, and O_3 operations, the Quantity attribute of Item is required in order to perform fewer operations per week. Also for the sum of O_4 and O_5 operations we have an additional benefit with the redundancy with respect to the case without it.

Relational Schema

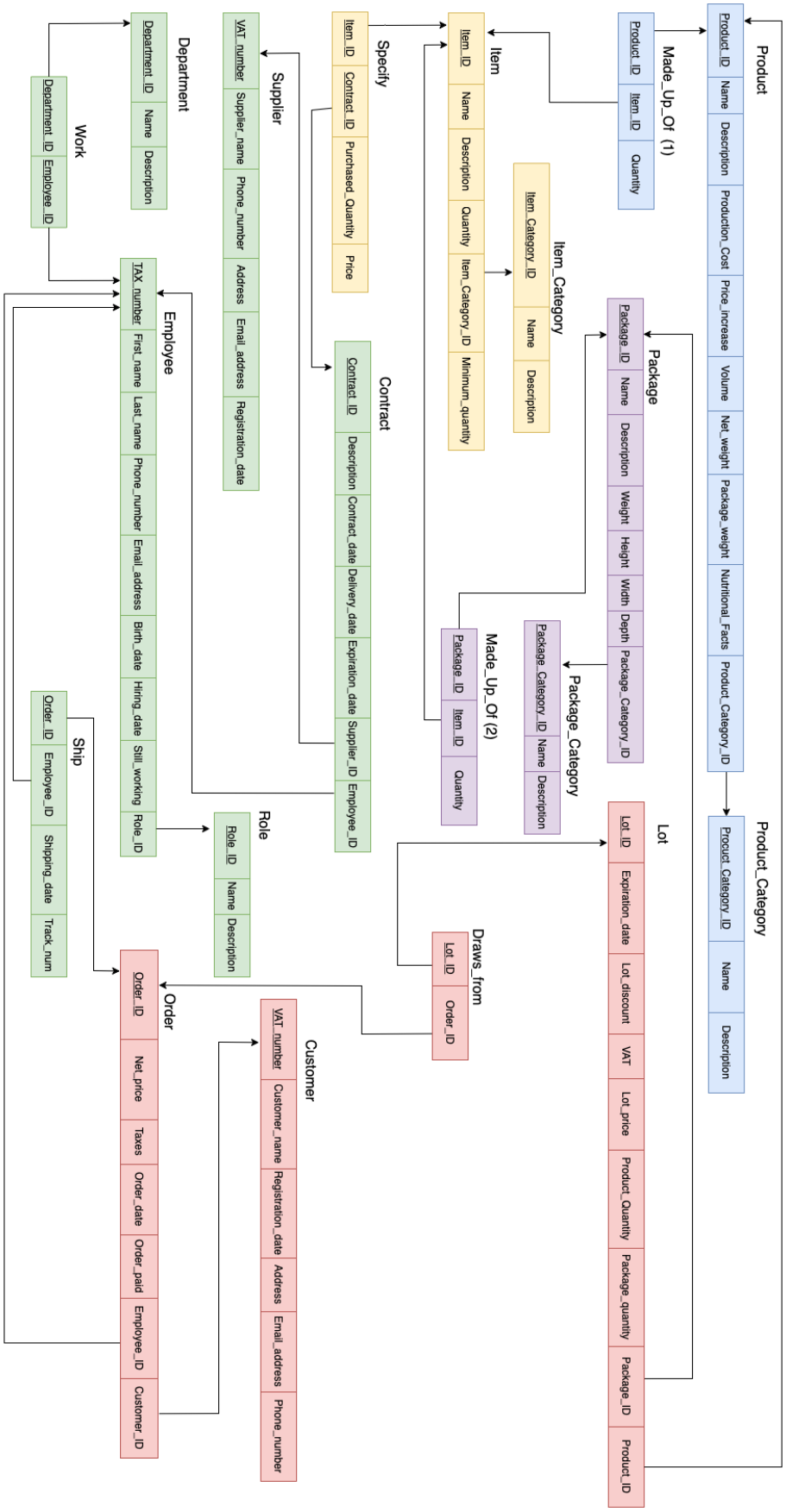

Figure 2: Relational schema.

Ship is a one to many relationship with optional participation either of Order and of Employee, due to the fact that an order can be ready to ship but not been shipped yet and a Worker might not have shipped an order yet. We have created the "Ship" relation, instead of incorporating its attributes into "Order", because if an order is ready but not yet shipped, the "Shipping_date" and "Track_num" attributes (as well as the "Employee_ID" foreing key) would be NULL, and there could be many orders in this situation. Draws_from is a one to many relationship with optional participation of Lot, due to the fact that a Lot can be produced in advance (and so it wouldn't be assigned to any order yet). We have created the "Draws_from" relation, instead of corporating the "Order_ID" foreing key into Lot because if a Lot is not assigned to any order yet, the "Order_ID" foreing key would be null, and there could be many lots in this situation.

Data Dictionary

Relation	Attribute	Description	Domain	Constraints
Product	Product_ID	Identifier of a product	Serial	PRIMARY KEY
	Name	Name of a product	Text	NOT NULL
	Description	Description of a product	Text	
	Production_Cost	Production cost of a product	Float	NOT NULL
	Price_increase	Price increase factor of a product	Float	NOT NULL
	Volume	Volume of a product in milliliters	Int	NOT NULL
	Net_Weight	Net weight of a product in grams	Int	NOT NULL
	Package_weight	Package weight of a product in grams	Int	NOT NULL
	Nutritional_Facts	Description of nutritional facts of a product	Text	NOT NULL
	Product_Category_ID	Identifier of a Product Category	Serial	NOT NULL, Foreign Key to Product_Category
Product_Category	Product_Category_ID	Identifier of a Product Category	Serial	PRIMARY KEY
	Name	Name of a Product Category	Text	NOT NULL
	Description	Description of a Product Category	Text	
Item	Item_ID	Identifier of an item	Serial	PRIMARY KEY
	Name	Name of an item	Text	NOT NULL
	Description	Description of an item	Text	
	Quantity	Stock quantity of an item	Int	NOT NULL
	Minimum_quantity	Minimum stock quantity of the item	Int	NOT NULL
	Item_Category_ID	Identifier of an Item Category	Serial	NOT NULL, Foreign Key to Item_Category
Item_Category	Item_Category_ID	Identifier of an Item Category	Serial	PRIMARY KEY
	Name	Name of an Item Category	Text	NOT NULL
	Description	Description of an Item Category	Text	
Specify	Item_ID	Identifier of an item	Serial	Foreign Key to Item, Primary key with Contract_ID
	Contract_ID	Identifier of a contract	Serial	Foreign Key to Contract, Primary key with Item_ID
	Price	The price of the amount of items which are purchased	Float	NOT NULL

	Purchased_Quantity	Amount of each item which is purchased	Int	NOT NULL
Contract	Contract_ID	Identifier of a contract	Serial	PRIMARY KEY
	Description	Description of a contract	Text	
	Contract_date	Date of signature of a contract with the supplier	Datetime	NOT NULL
	Delivery_date	Expected date of delivery of the goods	Datetime	NOT NULL
	Expiration_date	Expiration date of a contract	Datetime	NOT NULL
	Supplier_ID	Identifier of a supplier	Int	NOT NULL, Foreign Key to Supplier
	Employee_ID	Identifier of an employee	Text	NOT NULL, Foreign Key to Employee
Package	Package_ID	Identifier of a package	Serial	PRIMARY KEY
	Name	Name of a package	Text	NOT NULL
	Description	Description of a package	Text	
	Weight	Weight of a package in grams	Int	NOT NULL
	Height	Height of a package in centimeters	Int	NOT NULL
	Width	Width of a package in centimeters	Int	NOT NULL
	Depth	Depth of a package in centimeters	Int	NOT NULL
	Package_Category_ID	Identifier of a Package Category	Serial	NOT NULL, Foreign Key to Package_Category
Package_Category	Package_Category_ID	Identifier of a Package Category	Serial	PRIMARY KEY
	Name	Name of a Package Category	Text	NOT NULL
	Description	Description of a Package Category	Text	
Lot	Lot_ID	Identifier of a lot	Serial	PRIMARY KEY
	Expiration_date	Expiration date of the included products	Datetime	NOT NULL
	Product_Quantity	Amount of a product in each lot	Int	NOT NULL
	Package_Quantity	Amount of a package in each lot	Int	NOT NULL
	Lot_Discount	Percentage of discount of a lot	Int	NOT NULL
	VAT	Value added tax (percentage)	Int	NOT NULL
	Lot_price	Lot price without discount	Float	NOT NULL
	Package_ID	Identifier of a package in each lot	Serial	NOT NULL, Foreign Key to Package
	Product_ID	Identifier of a product in each lot	Serial	NOT NULL, Foreign Key to Product
Order	Order_ID	Identifier of an order	Serial	PRIMARY KEY
	Net_price	Total net amount including discount (VAT excluded)	Float	NOT NULL
	Taxes	Amount of taxes to be paid	Float	NOT NULL
	Order_date	Date in which the order has been processed	Datetime	NOT NULL
	Order_paid	Status of the order payment	Boolean	NOT NULL
	Employee_ID	Identifier of the employee that places the order	Text	NOT NULL, Foreign key to Employee
	Costumer_ID	Identifier of the customer that places the order	Int	NOT NULL, Foreign key to Customer

Customer	VAT_number	VAT_number of the customer	Text	PRIMARY KEY
	Customer_name	Name of the customer	Text	NOT NULL
	Phone_number	Phone number (prefix included) of the customer	Text	NOT NULL
	Address	Billing address of the customer	Text	NOT NULL
	Email_address	Email address of the customer	Text	NOT NULL
	Registration_date_ID	Customer registration date	Datetime	NOT NULL
Draws_from	Lot_ID	Identifier of a lot	Serial	Foreign Key to Lot, Primary key to Lot
	Order_ID	Identifier of an order	Serial	NOT NULL, Foreign Key to Order
Made_Up_Of (1)	Product_ID	Identifier of a product	Serial	Foreign key to Product, Primary key with Item_Id
	Item_ID	Identifier of an item	Serial	Foreign key to Item, Primary key with Product_Id
	Quantity	Amount of each item in a product	Int	NOT NULL
Made_Up_Of (2)	Package_ID	Identifier of a package	Serial	Foreign key to Package, Primary key with Item_Id
	Item_ID	Identifier of an item	Serial	Foreign key to Item, Primary key with Package_Id
	Quantity	Amount of each item in a package	Int	NOT NULL
Ship	Order_ID	Identifier of the shipped order	Serial	Foreign key to Order, Primary key
	Employee_ID	Identifier of employee that ships the order	Serial	NOT NULL, Foreign key to Employee
	Shipping_date	Date of when the order leaves to be shipped	Datetime	NOT NULL
	Track_num	Shipment code given by the third-party shipping company	Text	
Employee	TAX_number	TAX code of the employee	Text	PRIMARY KEY
	First_name	Name of the employee	Text	NOT NULL
	Last_name	Surname of the employee	Text	NOT NULL
	Phone_number	Phone number (prefix included) of the employee	Text	NOT NULL
	Email_address	Email address of the employee	Text	NOT NULL
	Birth_date	Birthdate of the employee	Datetime	
	Hiring_date	Hiring date of the employee	Datetime	NOT NULL
	Still_working	Flag used to know if employee is still working for the company	Boolean	NOT NULL
	Role_ID	Identifier of the role that an employee has	Serial	NOT NULL, Foreign Key to Role
Role	Role_ID	Role identifier	Serial	PRIMARY KEY
	Name	Name of the role	Text	NOT NULL
	Description	Technical description of the role	Text	
Work	Department_ID	Identifier of a department	Serial	Foreign key to Department, Primary key with Employee_ID

	Employee_ID	Identifier of an employee	Text	Foreign key to Employee, Primary key with Department_ID
Department	Department_ID	Department identifier of the company	Serial	PRIMARY KEY
	Name	Name of the department	Text	NOT NULL
	Description	Description of the department's function	Text	
Supplier	VAT_number	VAT number of the supplier company	Int	PRIMARY KEY
	Supplier_name	Name of the supplier company	Text	NOT NULL
	Phone_number	Phone number (prefix included) of the supplier company	Text	NOT NULL
	Email_address	Email address of the supplier company	Text	NOT NULL
	Address	Address of the supplier company	Text	NOT NULL
	Registration_date	Recording date of the supplier company	Datetime	NOT NULL

External Constraints

- Only employees that are sill working (i.e. those who have the "Still_working" attribute set to "True") in the company can access the system. So Employees that take part in any operation of insertion or modification must have True as value of attribute Still_working in its own Employee Relation.
- Only the salesman can insert new orders, so the Employee_ID in the Order relation must have the role equal to "Salesman". Only the salesman can update the order status.
- Only the manager can insert new contracts, so the Employee_ID of the Contract relation must have the role equal to "Manager".
- Only the Worker can ship orders, so the Employee_ID in the Ship relation must have the role equal to "Worker".

Group Members Contributions

Conceptual Design

- **Variations to the Requirement Analysis:** Esposito, Basso
- **Entity-Relationship Schema:** Esposito, Basso, Zanini, Collado, Giuliani
- **Entities Table:** Esposito, Basso
- **Relationships Table:** Zanini, Giuliani
- **External Constraints:** Quiroz, Collado, Esposito, Basso, Cimarosto
- **Functional Requirements Satisfaction Check:** Cimarosto, Collado, Arslan, Esposito, Basso

Logical Design

- **Transformation of the Entity-Relationship Schema:** Esposito, Basso
- **Analysis of Database Load:** Giuliani, Zanini
- **Relational Schema:** Esposito, Basso, Quiroz, Collado
- **Data Dictionary:** Collado, Arslan, Cimarosto, Basso, Esposito
- **External Constraints:** Collado, Basso, Esposito