Database Systems Project

Implementation of a database by starting from specifications

For this project the following have been realized:

Conceptual schema, logical analysis, logical schema, oracle implementation, triggers, physical analysis, data warehouse and client application

Student:

**Mattia COLUCCI**

Academic Year 2024/2025

Summary

[Specifications 3](#_Toc188470514)

[Glossary 3](#_Toc188470515)

[Specification filtering and grouping 4](#_Toc188470516)

[Conceptual schema 5](#_Toc188470517)

[Project decisions 7](#_Toc188470518)

[Logic design 8](#_Toc188470519)

[Volume table 8](#_Toc188470520)

[Access table 9](#_Toc188470521)

[Analysis of redundancies 10](#_Toc188470522)

[Partitioning, merging and selection of main identifiers 11](#_Toc188470523)

[Reconstructed conceptual schema 11](#_Toc188470524)

[Constraints not expressed in the conceptual schema 12](#_Toc188470525)

[Logical schema 12](#_Toc188470526)

[Implementation of the database 13](#_Toc188470527)

[Implementation of all types and tables 14](#_Toc188470528)

[Implementation of triggers 15](#_Toc188470529)

[Implementation of table filling procedures 17](#_Toc188470530)

[Trigger checking and implementation of required operations 18](#_Toc188470531)

[Physical analysis 18](#_Toc188470532)

[Algebraic optimization and intermediate results calculation 19](#_Toc188470533)

[Physical decisions 20](#_Toc188470534)

[Data warehouse 22](#_Toc188470535)

[Identification of facts, measures and dimensions 22](#_Toc188470536)

[Reconstruction of the E-R schema 23](#_Toc188470537)

[Logical relational schema 24](#_Toc188470538)

[Client application 25](#_Toc188470539)

# Specifications

The company "Brightway" manages decentralized logistics operations through several operational centers distributed across regions, each responsible for handling local storage and shipments. Each operational center is characterized by a name, address, city/province, and number of employees. The company offers customized warehouse management services, including long-term storage and expedited shipping. Orders can be placed by customers via phone, email, or directly through the company’s online platform. Each customer may have one or more business accounts, each identified by a unique code. Every order is associated with a single business account and includes details such as type, date, cost, and customer information. Orders can be of three types: regular, urgent, or bulk (large quantities). Operational centers have management teams, each identified by a unique code, name, and the number of operations handled. Teams consist of specialized personnel, and the number of members may vary depending on the required workload. Additionally, the company maintains a performance evaluation system that assigns a score to each team based on delivery times and customer feedback. Customers can be classified as individual or business, each identified by a unique alphanumeric code, with contact details and order history

**OPERATIONS**

**Operation 1**: Register a new customer (10 times per day).

**Operation 2**: Add a new order (1,000 times per day).

**Operation 3**: Assign an order to a management team (500 times per day).

**Operation 4**: View the total number of operations handled by a specific team (200 times per day).

**Operation 5**: Print a list of teams sorted by their performance score (20 times per day).

# Glossary

The first step is to build a **glossary** to represent all the identified **terms** and for each of them presenting: a description, synonyms and homonyms and other terms with which the current one is related.

Here is the glossary table:

|  |  |  |  |
| --- | --- | --- | --- |
| Glossary | | | |
| Term | Description | Synonyms and Homonyms | Relationships |
| Operational Center | It is the center that plans to handle orders (through teams) regarding a specific province of a region | x | Personnel |
| Team | Group of personnel that handles the shipment of orders of a specific operational center | management team | Order  Personnel |
| Order | Orders made by a account and handled by a specified team | shipments, operations | Team  Account  Customer |
| Account | An account related to a specific client, with which is possible to place an order | account | Order  Customer |
| Customer | Is the individual or business entity that performs orders through its account | x | Account  Order |
| Personnel | Components of a team | employee | Team  Operational Center |

# Specification filtering and grouping

In the next step, specifications provided above are written again to filterambiguities and avoid the use of synonyms and homonyms, standardizingphrases making them easier to be read.

In order to **filter** specifications, all repetitions, synonyms and homonyms have been substituted with only one term, equals for all the situations. All the requirements for each term have been written again by using the same words every time, standardizing all the specifications.

After the filter phase, specifications have been **grouped** by terms. These terms are important concepts that have been found in the specifications, and on which the database will be based. By grouping requirements for each term, we have a cleaner idea of the characteristics associated to each term and of the relationship between a term and the others.

Identified terms are the ones represented in the glossary table.

Here are the resulting specifications after the **filtering** and **grouping** phase:

---------------------------------------------------------------------------------------------------------------------

The company "Brightway" manages decentralized logistics operations through several operational centers distributed across regions. Each region is responsible for handling local storage and shipments. The company offers two main warehouse management services: long-term storage and expedited shipping.

Each **operational center** is characterized by a name, address, city/province, and number of employees that work there.

Each customer is associated with one or more **accounts**. An account is characterized by a unique code (identifier).

Each **customer** can be an individual or business. Each customer is characterized by a unique alphanumeric code (identifier), phone number. Each customer is associated with all the order it made thorough its accounts.

Every **order** is associated with a single account. Each order is characterized by a code (identifier), a type (regular, urgent, or bulk (large quantities)), placement date, shipping date, arrival date, cost, status (placed, shipped, arrived) and customer’s code of the account that made the order. Orders can be placed by an account via phone, email, or directly through the company’s online website.

Each operational center is associated with one or more **management teams**. Each management team is associated with one or more orders. Each management team is characterized by a unique code (identifier), name, and the number of orders handled. Additionally, the company maintains a performance evaluation system that assigns a score to each team based on delivery times (of orders handled by the team) and customer feedback (given at orders made by that team).

Teams are composed by **specialized** **personnel**, and the number of members may vary depending on the required workload.

Note: the first group regards general requirements not linked to a specific term. Each group is separated by an empty line

Note: in the new requirements, some attributes for some terms have been added in order to eliminate the ambiguities present in the original requirements (eg. Customer information has been substituted with customer’s code)

# Conceptual schema

As a project strategy for this case study, I have chosen the **hybrid** methodology. It plans to build an initial **skeleton schema** that represents all identified terms and relationships between them, without representing details (such as the attributes or the cardinalities). Then each entity is detailed creating multiple sub-schemas. Finally, all the sub-schemas are merged, obtaining the conceptual schema.

This technique is good for complex scenarios, since it allows to model each entity independently from the other, and do to require to have an initial global abstract visualization of the entire problem.

Follows the skeleton and the conceptual schema created by using the hybrid technique:

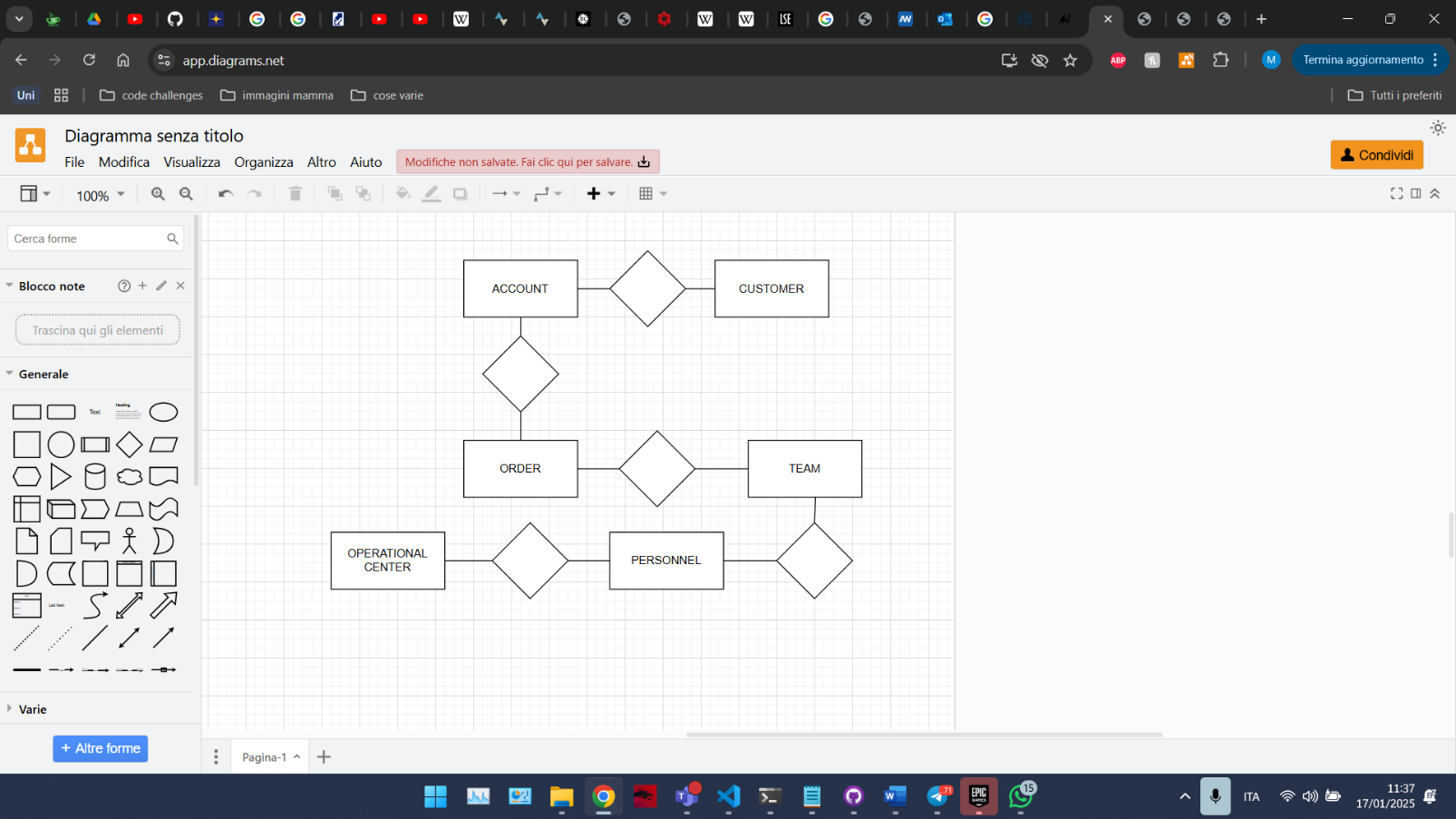


Figure 1. Skeleton schema

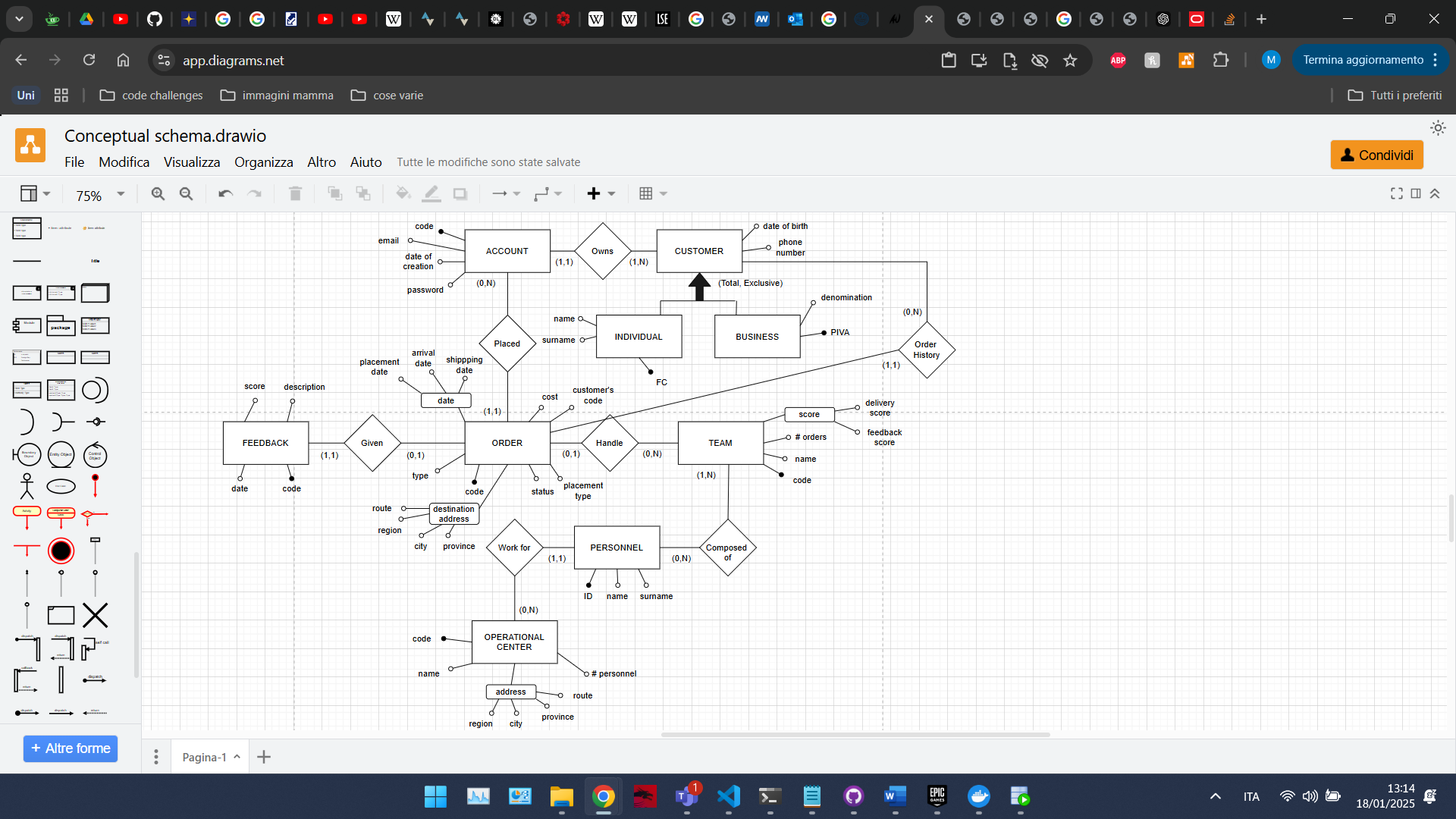


Figure 2. Conceptual schema

# Project decisions

Here there is exposed some information about how some requirements have been implemented in the conceptual schema, in order to have cleaner idea about how the system works. Furthermore, in the following schema, some **project decisions** have been motivated and justified. Decisions are grouped by entities found in the project.

|  |
| --- |
| Order |
| The orders represent the effective shipping service offered by the company and places by the customers.  Each order has 3 possible status: placed (the order has been placed by an account), shipped (the order has been shipped and assigned to a team), arrived (the order has arrived at destination)  Each order with status ”arrived” has a delivery time, that indicates the number of days that have been passed from the shipment to the arrival at destination. The delivery time is calculated as the difference in days between the “shipping date” and the “arrival date”  When an order is created, it is not assigned to a team, and it’s status is “placed”.  When an order becomes assigned to a team, then its status is set to “shipped”, considering the order as shipped effectively. |
| Team |
| Each team is composed by personnel which comes from the same operational center. So, each team has all members that works from the same operational center, and so it’s like it is assigned to a specific operational center.  The score related to a team is represented as 2 separated scores:   * Feedback score: with the score given by a new feedback. * Delivery score: with the delivery time of the new order that has been arrived.   The complete score can be obtained by making the mean between both the scores. For each score, the recursive formula to calculate it is provided. |
| Feedback |
| Each feedback refers to a specific order and it is made by the customer of which account has placed the order. The choice of link Feedback only with the Order entity is made to save space respect to link feedback to both Account and Order. This implementation would require 3 joins to access the customer from the feedback, but since there is no operation that regards feedbacks, this lack in performance can be accepted.  A feedback is optional for an order. |
| Storage service |
| The storage service offered by the company is not handled in this project. Only the shipment service is considered. This is due to the lack of entities and details about the storage service, that seems not to be managed by none of the identified entities. The only entity for which the storage service is mentioned is the operational center, but there are not further requirements for the implementation of this type of service. |

# Logic design

The next step is the **logic design**. In this phase the conceptual project is **refined** by using different techniques; this is done to remove further ambiguities and redundancies that there are in the original schema, obtaining an updated version.

In this phase, the following steps are conducted:

* Making of the volume table
* Making of the access table for each operation
* Analysis of redundancies
* Partitioning and merging
* Selection of the main identifiers

## Volume table

In this step the **volume table** is made. This table plans to specify an estimated number of tuples that will be stored into each entity and relationship represented in the conceptual schema, considering the whole life period of the database (in this case 10 years).

Here is represented the volume table for the current project:

|  |  |  |
| --- | --- | --- |
| VOLUME TABLE | | |
| ENTITY | VOLUME | NOTE |
| Team | 150 |  |
| Personnel | 600 |  |
| Composed by | 900 | 6 members x team |
| Handle | 5 400 000 | 36 000 orders x team |
| Order | 5 400 000 |  |
| Feedback | 5 000 000 |  |
| Given | 5 000 000 | 92% of orders have a feedback |
| Operational Center | 50 |  |
| Works for | 600 | 12 employees x center |
| Customer | 500 |  |
| Business | 400 |  |
| Individual | 100 |  |
| Accounts | 1000 |  |
| Owns | 1000 | 2 accounts x customer |
| Placed | 5 400 000 | 5400 orders x account |
| Order history | 5 400 000 | 10 800 orders x customer |

## Access table

Now we proceed to build the **access table** for each operation. The access table highlights the number of reads/write that needs to be done on each entity/relationship, to perform a certain operation.

Here are the access tables for each operation

|  |  |
| --- | --- |
| ACCESS TABLE | |
| TABLE | ACCESSES |
| OPERATION 1 (10x day) | |
| Customer | 1 write |
| Individual/Business | 1 write |
| Owns | 1 write |
| Account | 1 write |
|  | TOTAL: 8 accesses  TOTAL x day: 10\*8 = 80 accesses |
| OPERATION 2 (1000x day) | |
| Order | 1 write |
| Placed | 1 write |
| Account | 1 read |
| Owns | 1 read |
| Order history | 1 write |
| Customer | 1 read |
|  | TOTAL: 9 accesses  TOTAL x day: 1000\*9 = 9000 accesses |
| OPERATION 3 (500x day) | |
| Team | 1 read |
| Team | 1 write (update of #orders) |
| Handle | 1 write |
| Order | 1 read |
| Order | 1 write (update of status to “shipped”) |
|  | TOTAL: 8 accesses  TOTAL x day: 500\*8 = 4000 accesses |
| OPERATION 4 (200x day) | |
| Team | 1 read |
|  | TOTAL: 1 access  TOTAL x day: 200 accesses |
| OPERATION 5 (x20 day) | |
| Team | 150 reads |
|  | TOTAL: 150 accesses  TOTAL x day: 20\*150 = 3000 accesses |

All operations are iterative, since they are all executed with daily frequency.

## Analysis of redundancies

Now we conduct the **analysis of redundancies**. In this step we will consider all redundancies there are represented in the conceptual schema and we need to decide if maintain them or not, based on them influence on the operations (according to the number of accesses that each operation makes with and without the redundancy).

1. Number of orders in Team entity

The number of orders in the Team entity is a redundancy because we can get this number by making a count operation on the orders, filtering for those assigned to the considered team. This redundancy affects operation 3 and operation 4.

The number of accesses with redundancy for operation 3 is 4000 a day, while for operation 4 is 200 a day.

The number of accesses without redundancy for operation 3 is 3000 a day, while for operation 4 is 36 001\*200 = 7 200 200 (because we need to read all orders assigned to the team, which are 36 000 according to the volume table).

So, with redundancy we have 4000 + 200 = 4200 accesses a day, while without redundancy we have 3000 + 7 200 200 = 7 203 200 accesses a day. We decide to maintain the redundancy.

1. Number of personnel per operational center

The number of personnel that work for an operational center is a redundancy, because we can get this number by making a count operation on Personnel filtering for those that work in the desired operational center. This redundancy does not affect any operation, so we decide to remove the redundancy, since it causes only a waste of space.

1. The relationship “Order History” between Order and Customer

This relationship is a redundancy because we can get the order history of a customer by selecting the orders made by all the accounts that the customer has. This redundancy affects operation 2.

The number of accesses with redundancy for operation 2 is 9000 a day. The number of accesses without redundancy for operation 2 is 5000 a day.

We decide to remove the redundancy because we make more daily accesses with it.

1. Customer’s code in Order

The customer’s code in Order is a redundancy, since we can get it by accessing the account that made the order and then arrive to the customer through the Owns relationship. This redundancy affects operation 2.

By making the same analysis made in the point (3), we decide to remove the redundancy, also because there is not operation that would benefit from this redundancy.

## Partitioning, merging and selection of main identifiers

About partitioning and merging, we think that we do not have to merge or partition any entity or relationship. The only doubt was if to let Feedback become an attribute of Order. We decided to remain Feedback as a table because if we let all attributes of Feedback being attributes of Order, and so we would have too much null values in the case that an order has not received a feedback (and 8% of orders do not have a feedback).

The identifiers shown in the original conceptual schema (Figure 2) are all confirmed, and no modifications have been made on them.

# Reconstructed conceptual schema

Once conducted the whole analysis of the conceptual schema (Figure 2), we reconstruct again the conceptual schema, obtaining its final version with all the modifications expressed in the previous sections.

Here is shown the updated conceptual schema:

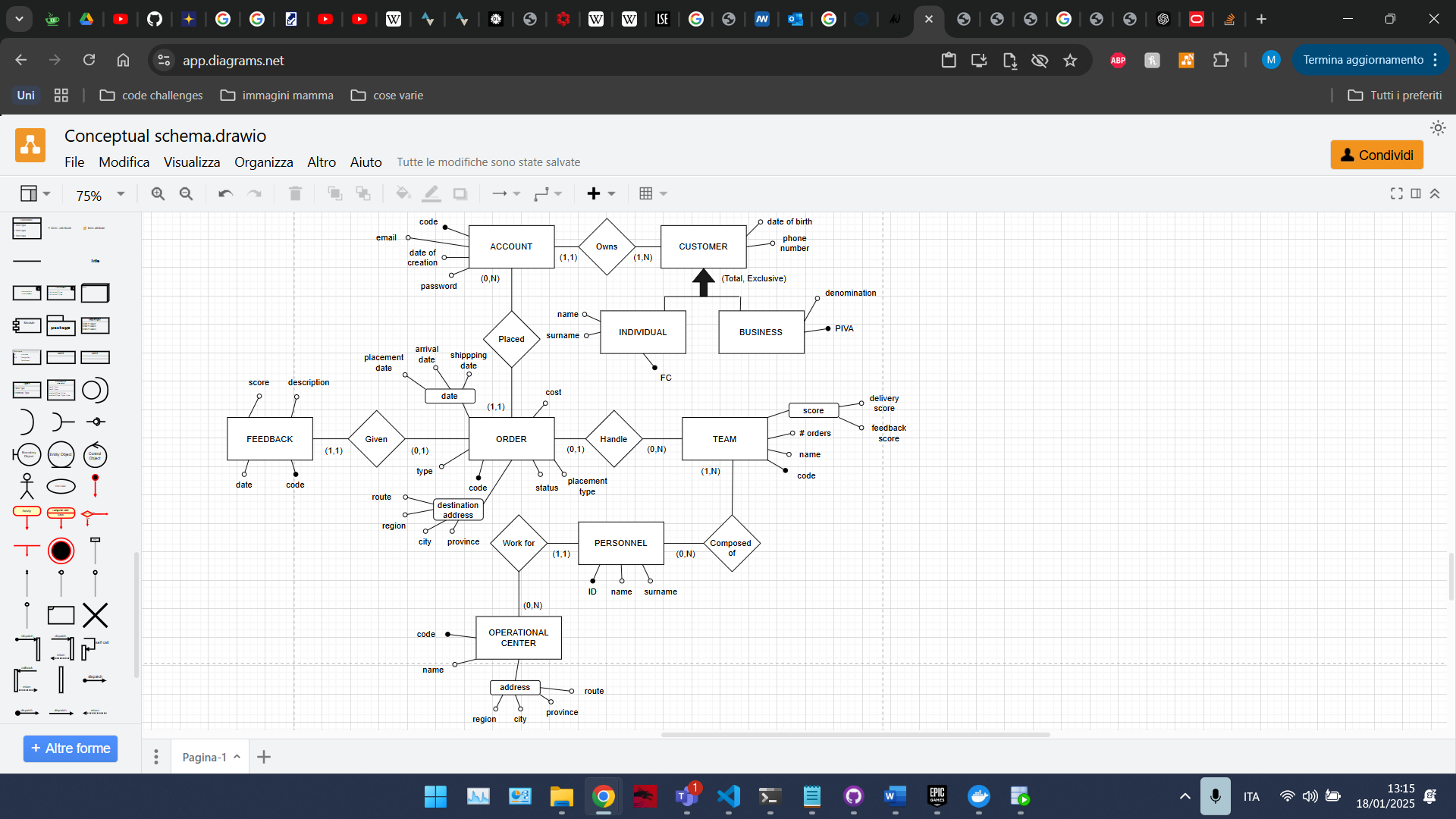


Figure 3. Refactored conceptual schema

## Constraints not expressed in the conceptual schema

In this section we list all **constraints** that need to be implemented in the database and that are not represented in the conceptual schema. Among these constraints we have the need to keep redundancies updated, limitation of values that an attribute can assume, and so on.

These constraints can be implemented by using the following constructs: **trigger**, **assertion** or **check** (supposing that the DBMS admits them).

Here are exposed all the constraints that we need to implement to maintain the database in a consistent state, by expressing (between brackets) the construct used to implement these constraints:

1. Check that the score attribute in Feedback is between 1 and 5 (check)
2. Change the status of the order to “shipped” when it is assigned to a team (trigger)
3. Check that all members of a team work for the same operational center (trigger)
4. Keep updated the redundancy “#orders” in Team (trigger)
5. Not allowing the insertion of a member in a team if it has already 8 members (trigger)
6. Keep updated the “feedback score” attribute in Team when a feedback is inserted for a order handled by that team (trigger)
7. Keep updated the “delivery score” attribute in Team when an order handled by that team changes its status to “arrived” (trigger)
8. Check that when an order is assigned to a team, the team works for an operational center that is in the same province as the province of destination address of the order (trigger)

Furthermore, we plan to implement the attribute “type” and “placement type” as enumerations. All the date attributes are implemented as timestamps.

# Logical schema

Once created and refactored the conceptual schema, the next step is to create the **logical** **schema**, by using the UML notation, which is independent from the programming language or platform used to implement the system.

We are going to implement an object relational database, that so plans to use typed tables to represent tables and aggregations to represent relationships, implementing them by using references or nested tables.

The transformation of the conceptual schema to the logical schema plans to represent entities as tables, typed tables of types and represent relationships as aggregations between the tables involved into it.

Here there is the UML schema:

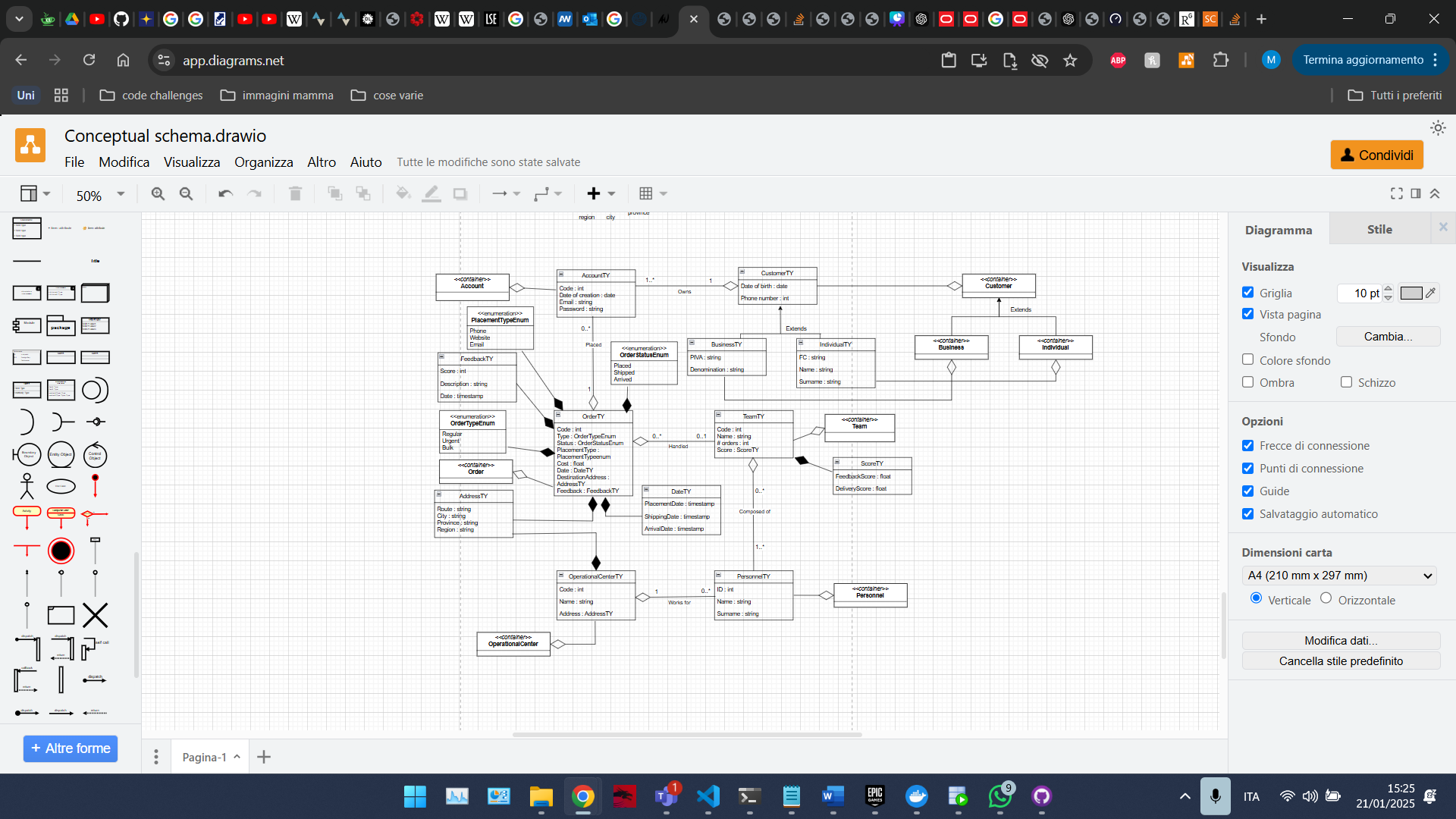


Figure 4. UML schema

NOTE: methods are not reported in the UML schema due to a lack of space.

NOTE: composition between types A and B indicates that type A contains a structured attribute of type B.

NOTE: The relationship between Accounts and Customer has not been implemented as a composition since Account has a mean also outside the Customer (infact it is used also in the orders). By putting a composition, if I delete the Customer, also all its Accounts would be removed and with them all the Orders made by them (otherwise they would have a dandling reference to an account that does not exists anymore), losing all historical data that could be useful in the case of making a data warehouse.

The same reasoning has been applied on the relationship between Operational Center and Personnel (they have a mean outside the center, because they are used in the teams).

# Implementation of the database

Once obtained the logical schema, the next step is to make the code to implement the database in Oracle, according to the logical schema previously defined.

The first thing to do is create a new user that will represent our DBA. The created user is named “C##ExamDBA” and has different privileges such as all the ones needed to implement the database and the one needed to connect to it. The role of the user is “CONNECT” and “DBA”. Once created the user, we create a new connection that uses it to authenticate. The new connection is named “Exam”.

The implementation phase is split into the following steps:

* Implementation of all types and tables expressed in the UML schema.
* Implementation of all the triggers necessary to ensure the consistency of the data and to respect all constraints not expressed in the conceptual schema.
* Implementation of procedures to automatically fill all the tables with some random data.
* Checking that the triggers work correctly, by making operations that fire them.
* Implement all the 5 operations expressed in the specifications.

## Implementation of all types and tables

The first step is to implement all the entities expressed in the logical schema. We first implement all the **types** and enumerations required and then we implement all the **typed tables** and tables, finishing the implementation of the database according to the logical schema.

Since Oracle does not provide a type ENUM to implement enumerations natively, we have to implement them by using the CHECK clause. For example, the enumeration of the status of the order (that plans 3 values: “placed”, ”shipped”, ”arrived”) has been implemented via a VARCHAR with a CHECK constraint, checking that its value is one of (“placed”, ”shipped”, ”arrived”).

In the following list we describe some **implementation decisions** that have been taken, providing reasons for each of them:

* Relationship between Team and Personnel has been implemented as an attribute of types VARRAY(8) of REF PersonnelTY, put in the type TeamTY. This way we automatically satisfy **constraint 5** that a team can have at most 8 members (since it is the maximum number of admitted members).
* The table Customer has not been implemented, since it is involved into a total generalization, that means that all customers are or business or individuals, and so there won't be any customer that is not specialized.

By implementing the Customer table, we would insert all tuples of Business and Individual also into Customer; furthermore, the nested table of the accounts on Customer needs to be implemented also on Business and Individual, having lots of duplicates.

Said this, Customer has been implemented as type, in order to benefit from its inheritance respect to Business and Individual (that will be defined as types under Customer).

* The relationship between Account and Business/Individual has been implemented by using a nested table of refs of accounts put in Business/Individual (since there is not MULTISET in Oracle and this is a one-to-many relationship). We put refs and not effective instances in the nested table, because this is an aggregation relationship (and so the container contains references to the contained items, not items themselves).
* The relationship between Operational Center and Personnel has been implemented by using a nested table of refs of personnel put in Operational Center. We put refs and not effective instances in the nested table, because this is an aggregation relationship.
* The relationship between Orders and Team/Account has been implemented by putting in Order an attribute of type ref of Team/Account.

Also, the following constraints have been used during the implementation:

* UNIQUE on the email of table Account.
* Each nested table contains references with scope to the table involved in the relationships.
* CHECK( order type IN ('regular', 'urgent', 'bulk') )
* CHECK( order status IN ('placed', 'shipped', 'arrived') )
* CHECK( order placement type IN ('phone', 'email', 'website') )
* CHECK (order feedback score BETWEEN 1 AND 5) **(Constraint 1)**

## Implementation of triggers

Before exposing all the implemented triggers, here are reported some constraints that have been added to those that cannot be represented by the conceptual schema:

1. Order’s dates cannot be modified by an UPDATE operation.
2. Cannot insert a customer that has an account owned by another customer
3. Set shipping/arrival date when status turns to ‘shipped’/’arrived’
4. Do not let insert a feedback into an order that is not in the status ‘arrived’

Here is the list of implemented **triggers** and for each of them, there is a description of what the trigger does:

|  |  |  |  |
| --- | --- | --- | --- |
| TRIGGER | TYPE | DESCRIPTION | CONSTRAINT IMPLEMENTED |
| **Check\_business\_insert** | AFTER INSERT ON Business | Checks that when a business is inserted, at least one account is provided, and the accounts provided are not owned by another business or individual | **10** |
| **Check\_individual\_insert** | AFTER INSERT ON Individual | Checks that when an individual is inserted, at least one account is provided, and the accounts provided are not owned by another business or individual | **10** |
| **Check\_team\_members** | AFTER UPDATE OF members OR INSERT ON Team | Checks that:   * A team has at least one member * A team has all members that work for the same operational center * A team has no duplicated members | **3** |
| **Update\_order\_not\_arrived** | BEFORE UPDATE OF feedback ON Orders | Let not insert a feedback if the order has not arrived | **12** |
| **Update\_feedback\_score** | AFTER UPDATE OF feedback ON Orders | When a feedback is inserted in an order, it updates the feedback score of the team related to the order | **6** |
| **Set\_order\_shipped** | BEFORE INSERT OR UPDATE OF team\_ref ON Orders | When a team is assigned to an order, it sets the order status to ‘shipped’ and sets the shipping date to now.  It does not allow the team assignment if the team works for an operational center of which province is different from the one expressed in the destination address of the order, or if the order status is ‘arrived’ | **2, 8, 11** |
| **Update\_num\_orders\_in\_**  **team** | AFTER INSERT OR UPDATE OF team\_ref ON Orders | When an order is assigned to a team, it increments the number of orders in the team assigned, and decrements the number of orders of the team that was previously assigned to the order (if there was some) | **4** |
| **Set\_order\_date** | BEFORE UPDATE OF status ON Orders | Prevents to change the status back to ‘placed’ and plans to set the shipping date when the status is changed to ‘shipped’ and the arrival date when the status is changed to ‘arrived’ | **11** |
| **Update\_delivery\_score** | AFTER UPDATE OF status ON Orders | When an order status is changed to ‘arrived’ calculates its delivery time and updates the delivery score of the team related to the order (if there is some) | **7** |
| **Prevent\_order\_date\_**  **update** | BEFORE UPDATE OF or\_date ON Orders | Let the date of the orders not being changed by UPDATE operations (unless the previous date is null) | **9** |
| **Update\_num\_orders\_on\_**  **delete** | AFTER DELETE ON Orders | When an order is deleted, it decrements the num of orders in the team related to the order (if there was some) | **4** |

## Implementation of table filling procedures

In order to facilitate the insertion of tuples in the tables, some **procedures** to automatically fill the tables have been created. These procedures plan to generate random values for each of the attributes of a specific table and allow the insertion of tuples respecting all the implemented triggers.

More specifically, tuples are inserted in a way that:

* Every customer has one account associated with it. The account is also inserted in the Account table
* Every operational center has 5 personnels and an address with 'province' as province. Each personnel is also inserted in the Personnel table
* Each team has 5 members, took from the personnel table
* Even coded orders have status 'shipped' and a random team related to them. Odd coded orders have status 'placed' and no team related to them. No order has a feedback.

Here there is a table summarizing the number of tuples created and the range of them identifiers for each table:

|  |  |  |
| --- | --- | --- |
| TABLE | # TUPLES | RANGE OF IDENTIFIER |
| Business | 100 | - |
| Individuals | 100 | - |
| Accounts | 200 | 1-100 (Business) 101-200 (Individual) |
| Operational centers | 20 | 1,5,10,15,…,100 |
| Personnel | 100 | 1-100 |
| Team | 50 | 1-50 |
| Orders | 500 | 1-500 |

## Trigger checking and implementation of required operations

In order to check that triggers work well, some blocks of code have been created. These blocks plan to simulate a situation in which the trigger fires, checking if it does its job well or not. These blocks use the tuples inserted by the previously explained procedures.

Lastly, all the 5 operations expressed in the specifications have been implemented and checked if works correctly respect to the implemented triggers.

# Physical analysis

The **physical analysis** of the database is conducted in order to optimize the operations that are planned to be made on the database, by implementing auxiliary structures such as indexes or clusters. Furthermore, this analysis allows us to get an idea of the volumes of intermediate results that are calculated during the execution of a query. Based on these volumes, we are able to take better physical decisions.

By using Oracle as DBMS, we have all secondary structures needed to implement hash, bitmap and b-tree indexes. Furthermore, Oracle offers the possibility to build clusters.

Automatically, at the moment of the creation of tables and definition of them primary keys, Oracle implements **primary sparse B+tree** indexes on all these primary keys. The following indexes have been implemented automatically:

* Index on ‘code’ in the Account table
* Index on ‘FC’ in the Individual table
* Index on ‘Piva’ in the Business table
* Index on ‘code’ in the Team table
* Index on ‘code’ in the Operational Center table
* Index on ‘id' in Personnel table
* Index on ‘code' in Order table

Furthermore, Oracle has created indexes on an ‘id’ attribute that is present in the nested tables, that acts as a foreign key linking to a specific row of the parent table. Also, Oracle implements primary B+tree indexes on fields with UNIQUE constraint.

The physical analysis is divided into the following steps:

* Calculate the volume table and the access table for each operation (already done)
* Conduct the algebraic optimization for each operation
* Calculates all the volumes of intermediate results for each operation
* Decide the auxiliary structures to implement

## Algebraic optimization and intermediate results calculation

Starting from a query, the **algebraic optimization** plans to create another query, equivalent to the initial one, that is faster and more efficient. The main idea is to anticipate selections and projections on the product and anticipate selections over the projections.

This optimization is made by using some properties of algebraic operators and by following this algorithm:

* Anticipate selections on projections
* Group selections
* Anticipate selections on the product
* Repeat previous steps until possible
* Eliminate unnecessary projections
* Anticipate projections on the product

For each query we represent its algebraic expression as a tree, in which nodes are algebraic operations and leaves are tables on which the operation act. For each node we show the volume of the intermediate results. These volumes are calculated by using **relation profiles**.

Here is the algebraic optimization for each query that is planned to be performed on the database

**OPERATION 4**

Original algebraic query:

Optimized algebraic query:

Tree representation of the algebraic optimized query (near each node there is the volume of the intermediate result, represented In red):

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

**OPERATION 5**

Original algebraic query:

Optimized algebraic query:

Tree representation of the algebraic optimized query (near each node there is the volume of the intermediate result, represented In red):

Immagine che contiene testo, schermata, software, numero

Descrizione generata automaticamente

## Physical decisions

In this section there are exposed all **physical decisions** made about inserting or removing auxiliary structures, in order to optimize the operations planned to be performed on the table.

To check the effective performance of the operations we have used to **autotrace** feature of Oracle, that shows which structures and components are used during the execution of a specific query, by effectively executing it. To do so we have granted following permissions to our DBA user: GRANT SELECT\_CATALOG\_ROLE TO C##EXAMDBA; and GRANT SELECT ANY DICTIONARY TO C##EXAMDBA;

* **Operation 4**
  + Performance of operation 4

According to autotrace, the operation 4 has following costs and time execution:

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

As we can see, the primary index made on ‘code’ of Team is used during the execution of the query

* + Analysis of the B+tree on ‘code’ of Team

The index on ‘code’ of the table Team influences the operation 4.

Since this index is a B+tree, we can calculate the fan out of the tree as: with the size of a block (that in our case is 8192), the size of a pointer (in our case Is 8) and the length of the attribute on which the index has been made (in our case, the ‘code’ attribute has length 22).

So, the fan out of the tree is

So, considering that the volume of the table Team is , we can calculate the depth of the B+tree as

So, the cost of operation 4 is accesses a day.

* + Join complexity

In case of joints between a table A and the table Team, with a condition on the ‘code’ of Team, thanks to the B+tree, we can execute the join by using:

* + - Nested loop

with a complexity of

* + - Merge scan (In the case that we have another B+tree on the attribute, involved in the join condition, of the table A)

with a complexity of

with the cardinality (number of tuples) of the table A and the cardinality of Team. Also is the blocking factor that depends on the nodes of the tree.

* + Alternatives

As an alternative to the already present B+tree index on ‘code’, we could use an hash index. An hash index could bring various advantages like guarantee cost 1 (1 access) for a selection on the ‘code’ of Team, also in possible future cases in which the volume of Team increases (and the depth of the B+tree increases too, increasing the number of required accesses). Furthermore, among the operations there are no queries with inequalities conditions on the ‘code’, that would make the hash index, a bad choice. The problem is that hash primary indexes are not implemented in Oracle.

Another option could be the hash cluster on ‘code’ of Team, but this structure cannot be implemented in typed tables or in tables that uses user-defined types as attributes (as happened in our case).

* + Final decision

We decide to maintain the B+tree primary index on ‘code’ of Team.

* **Operation 5**
  + Performance of operation 5

According to autotrace, the operation 5 has following costs and time execution:

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

* + Inefficiency of a possible Index on ‘score’ of Team

If we implement an index on ‘score’ of the table Team, this index is not used by the operation 5 (according to the autotrace) and so it would not faster its execution. Furthermore, having an index is not good for update operations on the attribute ‘score’ of the Team, because it let these operations being slower than before the implementation of the index.

This means that implementing an index on ‘score’ of Team would be a bad choice.

* + Final decision

So, operation 5 cannot be optimized by taking physical decisions. And so we remain the situation as it is.

# Data warehouse

This this section we discuss a possible implementation of a data warehouse in order to conduct analysis of historical data stored in the database that we are developing.

A **data warehouse** is useful in the case of **analysis** on **historical data**, mainly for complex studies on some data that needs to be **consistent**. Data in a data warehouse is always **static**; this means that it is not updated by online users and delete operations are not allowed. Data in a data warehouse is periodically updated when necessary.

Data in a data warehouse is taken from different **data sources** that can use different schemas, and so that can be heterogeneous to each other. Due to this situation, data taken from these data sources needs to be pre-processed by conducting a reconciliation phase that plans to correct all heterogeneities between these data.

Keep in mind that the creation of a data warehouse is so **expensive** due to this required data reconciliation and to the support used to store and host all the data.

In our case, we suppose that we have one only data source represented by our database.

The creation of a data warehouse is divided into the following phases:

* Identification of facts and measures
* Identification of dimensions
* Reconstruction of the E-R schema
* Construction of the logical relational schema

## Identification of facts, measures and dimensions

In this section we discuss the analysis that we decided to conduct for the creation of the data warehouse, in terms of facts, measures and dimensions.

Facts represent the concepts on which the analysis is made. Measures represent atomic properties of a fact on which the analysis is conducted. Dimensions represent different points of view along which the analysis is conducted (concepts that group instances of facts).

We want to conduct an analysis of the orders (**fact**) made by the customers of our system.

This analysis regards the following **measures**: delivery time of the order (difference between the arrival and shipping date), cost of the order and type of the order (which can be: ‘regular’, ‘urgent’ or ‘bulk’).

The **dimensions** along which the analysis will be conducted are: the account that made the order (and its related customer), the time at which the order has been placed and the team that handled the order (with the location of the operational center for which the team works).

## Reconstruction of the E-R schema

In this section we show the reconstructed **E-R schema** according to the selected facts, measures and dimensions.

The creation of a data warehouse plans to rebuild the E-R schema by starting from the conceptual schemas of the single data sources. In our case we will consider the conceptual schema in the Figure 3.

In the rebuilding of the E-R schema, the fact is represented as a table of which attributes are the selected measures. For each dimension we need to identify different **aggregation levels** (attributes of the dimension) each of which is represented as a table, and these tables are related to each other.

Here is the reconstructed E-R schema:

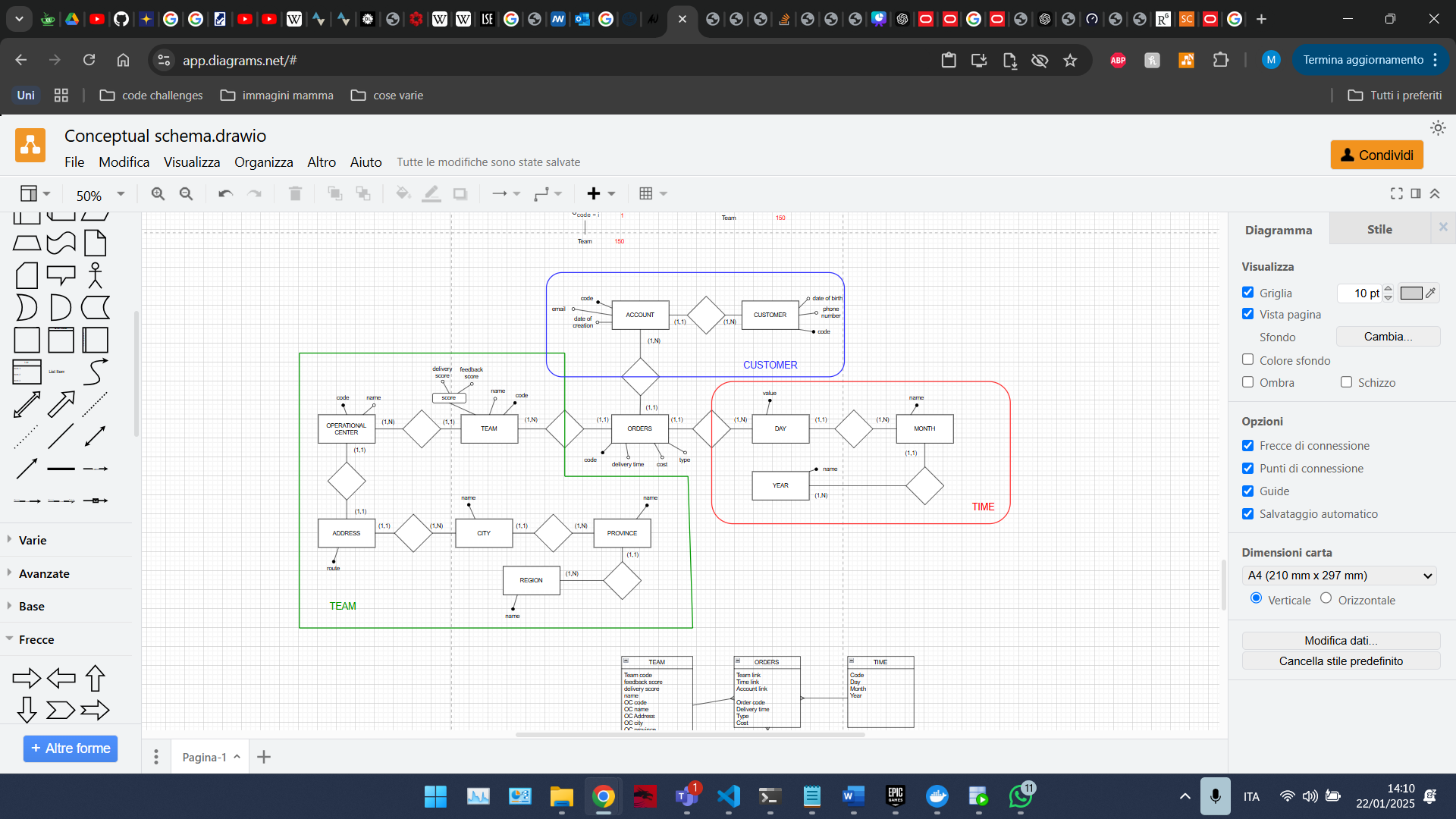


Figure 5. Reconstructed E-R schema for the data warehouse

## Logical relational schema

In this section it is described which logical schema has been selected and the motivations behind this selection. Also, the logical schema is shown.

As logical schema we have two main possibilities: **star schema** and **snowflake schema.** The **star schema** is the simpler one and is used in the case in which we need faster joints for our analysis, and we have no many-to-many relationship. The **snowflake schema** is used in the case we have many-to-many relationships, and we need that all dimensions tables are normalized.

We have selected to implement a **star schema**, since there are no many-to-many relationships that would justify the usage of a snowflake schema. Furthermore, with the snowflake schema we would have a gain in memory saving, but this gain is not enough to overcome the loss in speed and efficiency of joints operations, because we have more dimensions tables respect to the star schema (since in the snowflake, each aggregation level is represented as a table). Also, most of the time, the gain in memory saving that we have with the snowflake schema is almost insignificant.

By selecting a star schema, we have that the fact table is in Boyce-Codd normal form while all the dimensions tables (one for each dimension) are not normalized, in order to **fast joints** operations between them. This denormalization does not cause **anomalies** and does not cause problems for guaranteeing **ACID properties**, because data into a data warehouse is static.

Here is the star schema for the data warehouse:

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

Figure 6. Star schema of the data warehouse

NOTE: in order to use a star schema we need all measures to be numeric. So, we have converted the ‘type’ of the order into a numeric attribute with the following association to the original values: 1->’regular’, 2->’bulk’, 3->’urgent’.

# Client application

The last part of the project is the development of a **client application**, more specifically of a web application. Thanks to this application, a user can perform different operations on the database, without connecting directly to it.

The web application is composed by two parts: a frontend and a backend. The **frontend** part is written by using ReactJS and represents our client. It allows the user to make queries and data manipulation operations on the database. The **backend** part is written in NodeJS and represents the application server, that plans to get all the requests made by the client and forward them to the server (DBMS), by connecting to it. Results of the computation made by the server are sent to the backend and then sent back to the front end, where these results are presented and shown.

The simple application gives the possibility to perform all the 5 operations expressed in the specifications, and to easily read them results. Each operation will be made by pressing a button on the website. Then a loading screen appears, inviting the user to wait for the server’s response. Once the server has performed the requested operation, its results are shown on the website.

NOTE: the goal of this application is just to simulate an interaction between a web application and our database, trying the communication between them by performing the operations expressed in the specifications.

Here is how the client application looks like:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente