



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

COMPUTER SCIENCE
DATABASE SYSTEM – LAB

ML SOLUTIONS

AURORA TOMA – 796719

INDEX

Introduction

1. Specifications on data
2. Requirements analysis
 - 2.1 Choose the right level of abstraction
 - 2.2 Standardize sentence structure
 - 2.3 Linearize phrases and divide those articulated
 - 2.4 Identify homonyms and synonyms
 - 2.5 Making explicit reference between terms
 - 2.6 Building a glossary of terms
 - 2.7 Reorganizing phrases per keywords
3. Conceptual design
 - 3.1 ER Diagram
 - 3.2 Project strategy
4. Logical design
 - 4.1 UML
5. Oracle
 - 5.1 Types' creation
 - 5.2 Containers' creation
 - 5.3 Casual data generation
 - 5.4 Triggers
 - 5.4.1 Considered triggers
 - 5.5 Queries and Indexes
6. Data Warehouse

Conclusion

INTRODUCTION

Databases play a fundamental role in nowadays world.

They are structured systems designed to efficiently store, manage, and retrieve data. Their importance lies in their wide-ranging applications across various industries and sectors. Indeed, databases provide a structured method for organizing vast amounts of data, ensuring data integrity, consistency, and reliability. Moreover, this structured approach facilitates efficient data querying and analysis.

To sum up, databases are foundational to modern computing and have broad applications in business, science, government, healthcare, education, and beyond.

This document aims at analyzing a specific case given by MLSolutions, an online and physical machine learning solutions company, that intends to develop a database to manage its machine learning models and deployment transactions.

1. SPECIFICATIONS ON DATA

The first step to be undertaken is requirements gathering.

The goal is to create a comprehensive understanding of the database's purpose and functionality before moving into the design phase and this is achieved considering as possible sources users or already existing applications in that domain. However, this is not a standardized activity since requirements are expressed using natural language, possibly leading to ambiguity. For this reason, the usage of examples is needed to verify what's relevant and highlight essential aspects.

In ML Solutions' specific case, the database will contain detailed records of models, including model name, algorithm type, training dataset, accuracy, date of creation, and availability. Additionally, it is required to include a comprehensive history for each model, with information such as versions, update dates, training datasets used, performance metrics, and the corresponding documentation. Furthermore, the database will record highlights within models, such as key features or important training steps. This may include information about the data preprocessing steps, main features used, and the key insights derived from the model. The database will need to distinguish between different types of models and deployment environments. For pre-trained models, information such as model size, format, dependencies, and performance metrics

will be recorded. For models deployed on cloud platforms, details such as platform type, resource allocation, access URL, and usage statistics will be registered. Furthermore, the database will collect information on video tutorials and webinars, such as those on model training techniques or deployment strategies. These contents may include title, speaker, genre, duration, and file format details. Regarding transactions, it will be necessary to include details on different types of transactions, including purchases of pre-trained models, downloads of model documents, webinar registrations, and subscriptions to cloud deployment services. Customer data will be recorded, including name, surname, company name, email address, and payment information. Additionally, user account management with credit functionalities and various subscription types is envisaged.

Following operations are carried out on this data:

Operation 1: Registration of a new model deployment (once per day);

Operation 2: Select the accounts with a subscription to cloud deployment services (once a day);

Operation 3: Request information on models without corresponding documentation (1000 per day);

- Operation 4: Print the history and details of models using less than 10 features (100 per day);

Operation 5: Print all the summaries of the models created by the developer of the model with the highest accuracy (once a month).

Considering that the system manages 10000 different models, 100000 customers, each customer deploys an average of 1.8 models per year, define the number of transactions.

2. REQUIREMENTS ANALYSIS

Requirements analysis is a crucial phase that involves examining, understanding, and refining the requirements gathered from stakeholders. This phase focuses on clarifying and organizing the collected requirements to ensure they are complete, consistent, and aligned with the goals of the system being developed.

2.1 CHOOSE THE RIGHT LEVEL OF ABSTRACTION

It refers to a critical decision point in the design process where the appropriate level of detail and complexity is determined for the task at hand.

In this case, as regards the ‘payment information’ of one customer, we can identify the type of the method of payment and the possible commission associated with it. Moreover, as regards the ‘usage statistics’ of deployed models, we can identify them as the number of downloads for those models.

2.2 STANDARDIZE SENTENCE STRUCTURE

It refers to a process of establishing consistent rules and conventions for constructing sentences within a given context or domain, always using the same syntax style.

In this case, it wasn’t necessary.

2.3 LINEARIZE PHRASES AND DIVIDE THOSE ARTICULATED

It involves structuring and organizing information by converting phrases into linear sequences and breaking down complex expressions into simpler, more manageable parts.

In this case, it wasn’t necessary.

2.4 IDENTIFY HOMONYMS AND SYNONYMS

It involves recognizing and distinguishing between words that sound alike (homonyms) or have similar meanings (synonyms) within a given context.

In this case we have the definition of ‘performance metrics’, needed to measure model capabilities. We consider it as its synonym ‘accuracy’ in History’s entity, since in the Model’s one it was the only one considered.

Moreover, in highlights’ definition there’s the presence of ‘main features used’ and ‘key features’, that could be considered as synonyms. In this case, both are called ‘key features’.

2.5 MAKING EXPLICIT REFERENCE BETWEEN TERMS

It involves establishing clear and unambiguous connections or relationships between terms within a specific context or domain.

In this case, it wasn’t necessary.

2.6 BUILDING GLOSSARY OF TERMS

It involves building a comprehensive list of key terms used within a specific domain, along with their precise definitions and contextual meanings.

In this case, the following glossary could be created.

Term	Description	Connections
Model	ML models kept by MLSolutions’ database.	History, Highlight, Transaction, Available
History	Comprehensive history for each model kept by MLSolutions’ database.	Model
Highlight	Highlights within models recorded by MLSolutions’ database.	Model
Explanation	Video tutorials and webinars regarding the use of models or deploying techniques.	Model, Transaction
Transaction	Transactions carried out by customers, of different types.	Model, Customer

Customer	Customers of MLSolutions company, that carry out transactions	Transaction
MethOfPaym	Customer's method of payment information.	Customer
CloudDeplServ	Cloud deployment services offered by MLSolutions' company.	Transaction

Table1 – Glossary of Terms

2.7 REORGANIZING PHRASES PER KEYWORDS

It involves reorganizing phrases within a document or text to optimize their alignment with specific keywords or key concepts.

In this case the following phrases were identified.

- Phrases of **general nature**:

This document aims at analyzing a specific case given by MLSolutions, an online and physical machine learning solutions company, that intends to develop a database to manage its machine learning models and deployment transactions.

- Phrases related to **Model**:

In ML Solutions' specific case, the database will contain detailed records of models, including model name, algorithm type, training dataset, accuracy, date of creation, and availability.

The database will need to distinguish between different types of models and deployment environments. For pre-trained models, information such as model size, format, dependencies, and performance metrics will be recorded. For models deployed on cloud platforms, details such as platform type, resource allocation, access URL, and usage statistics will be registered.

- Phrases related to **History**:

Additionally, it is required to include a comprehensive history for each model, with information such as versions, update dates, training datasets used, performance metrics, and the corresponding documentation.

- Phrases related to **Hightlight**:

Furthermore, the database will record highlights within models, such as key features or important training steps. This may include information about the data preprocessing steps, main features used, and the key insights derived from the model.

- Phrases related to **Explanation**:

Furthermore, the database will collect information on video tutorials and webinars, such as those on model training techniques or deployment strategies. These contents may include title, speaker, genre, duration, and file format details.

- Phrases related to **Transaction**:

Regarding transactions, it will be necessary to include details on different types of transactions, including purchases of pre-trained models, downloads of model documents, webinar registrations, and subscriptions to cloud deployment services.

- Phrases related to **Customer**:

Customer data will be recorded, including name, surname, company name, email address. Additionally, user account management with credit functionalities and various subscription types is envisaged.

- Phrases related to **MethOfPaym**:

Customer's payment information will be recorded.

- Phrases related to **CloudDeplServ**:

Subscriptions to cloud deployment services will be recorded.

3. CONCEPTUAL DESIGN

It refers to the initial phase of database design where the focus is on defining the high-level structure and organization of the database, independent of specific implementation details. This phase aims to capture the essential requirements and logical relationships between data entities, helping to establish a conceptual model of the database system. The latter could be expressed using an ER diagram which can best describe the reality of interest.

3.1 ER DIAGRAM

Conceptual design often involves creating an Entity-Relationship Model (ERM) and there are many possibilities to use it to represent a set of specifications. The general criteria applied defines that if a concept has significant properties and/or describes object with an autonomous existence, it's represented as *Entity*. Then, if a concept has a simple structure and has no properties, it's represented as *Attribute* (of another concept it's referring to). Finally, if two or more entities are identified and there's a concept that associates them, the latter is represented as *Relationship*.

Quality of a conceptual schema is given by a model *syntactically correct*, that means constructs are used properly; *semantically correct* that means using constructs that meet their definition; and *complete* that means all data of interest are represented and all operations are performed.

The ER diagram designed follows.

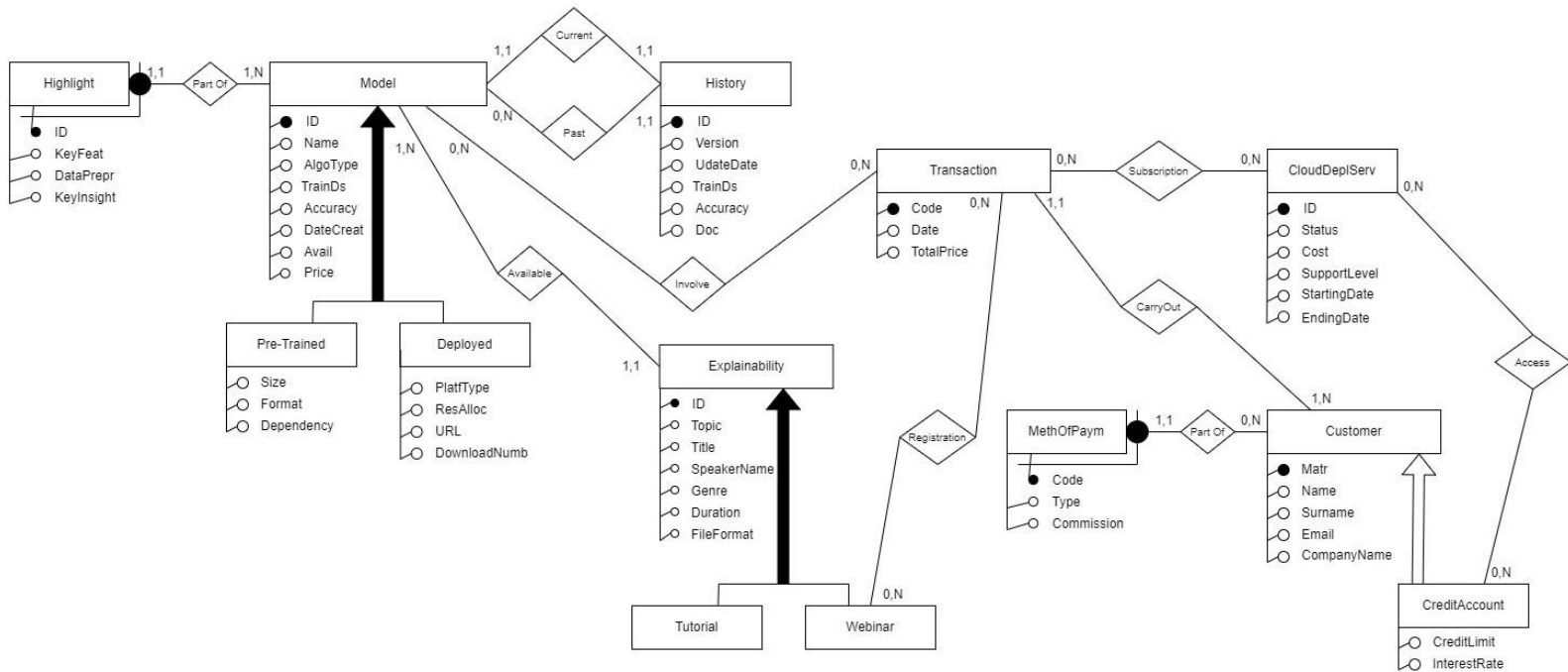


Figure1 – Conceptual schema

As shown in figure 1, represented entities are several. Connections identified among them are:

- **Model**, connected to **Highlight**, considering that one model could have one or more than one highlights as part of it, and one highlight is part of one model;
- **Model**, also connected to **History**, considering that one model could have one current history, or none or more than one past histories, while one history corresponds to one single model;
- **Model**, also connected to **Explainability**, considering that for each model one or more explanations are available, while one explanation is available for a single model;
- **Model**, also connected to **Transaction**, considering that none or more than one models can be involved in a transaction, while one transaction can involve none or more than one models;
- **Transaction**, also connected to **Customer**, considering that one transaction is carried out by one single customer, while one customer can carry out one or more than one transactions;

- **Transaction**, also connected to **CloudDeplServ**, considering that in one transaction none or more than one cloud services can be subscribed, while one cloud service can be subscribed in none or more than one transactions;
- **Customer**, also connected to **MethOfPaym**, considering that one customer could have one or more than one methods of payment, while one method of payment is part of one single customer;

Note: differences with respect to written exam are:

- CloudDeplServ entity added with its attribute. Needed because it was considered among different types of transactions, and credit account with subscription types were considered;
- Attributes of CreditAccount added;
- Attribute 'commission' in MethOfPaym added;
- 'UsageStats' attribute in Deployed modified with 'downloadNumb' attribute;
- Attribute 'keyInsight' in Highlight added;
- Primary key elimination from PreTrained, Deployed and CreditAccount entities thanks to inheritance.

3.2 PROJECT STRATEGY

While building conceptual schema, the *inside-out* strategy was performed.

It consists in starting a project by identifying some essential concepts first, and then progressively expanding them to incorporate additional features or components, moving from closer to more distant concepts.

This strategy is characterized by a continuous iterative approach where project teams iterate through development cycles, adding new features and enhancements in each iteration. This iterative process enables flexibility and adaptability to changing requirements and market conditions. Moreover, it doesn't require steps of integration of operations from different conceptual schemas, avoiding difficulties in case of complex schemas. On the other hand, it could lead to lack of abstraction because it's necessary time to time to look for non-already represented concepts.

4. LOGICAL DESIGN

Logical design involves translating the conceptual data model (established during conceptual design) into a more detailed and structured representation that can be implemented using a specific database management system (DBMS). Indeed, it generates logical structures that accommodate data.

UML (Unified Modeling Language) was used for this purpose, representing an object-relational system.

4.1. UML

UML, which stands for Unified Modeling Language, is a standardized visual modeling language used in software engineering and systems design to define, visualize, achieve, and document the artifacts of a software system, using an object-oriented approach.

It's considered as universal thanks to its capability to represent heterogeneous system architectures without getting lost in details of programming languages.

It provides a common set of graphical notations with a well-defined semantics, making it easier for software developers, designers, and stakeholders to communicate and understand system requirements, designs, and behaviours.

Its main objective is to produce a better software environment through the use of modeling. Indeed, it abstracts and simplifies the most complex projects, allowing to visualize how a system is or how it should be; it specifies the structure and behaviour of it, defining guidelines for its building and documenting decisions taken.

The designed logical schema follows.

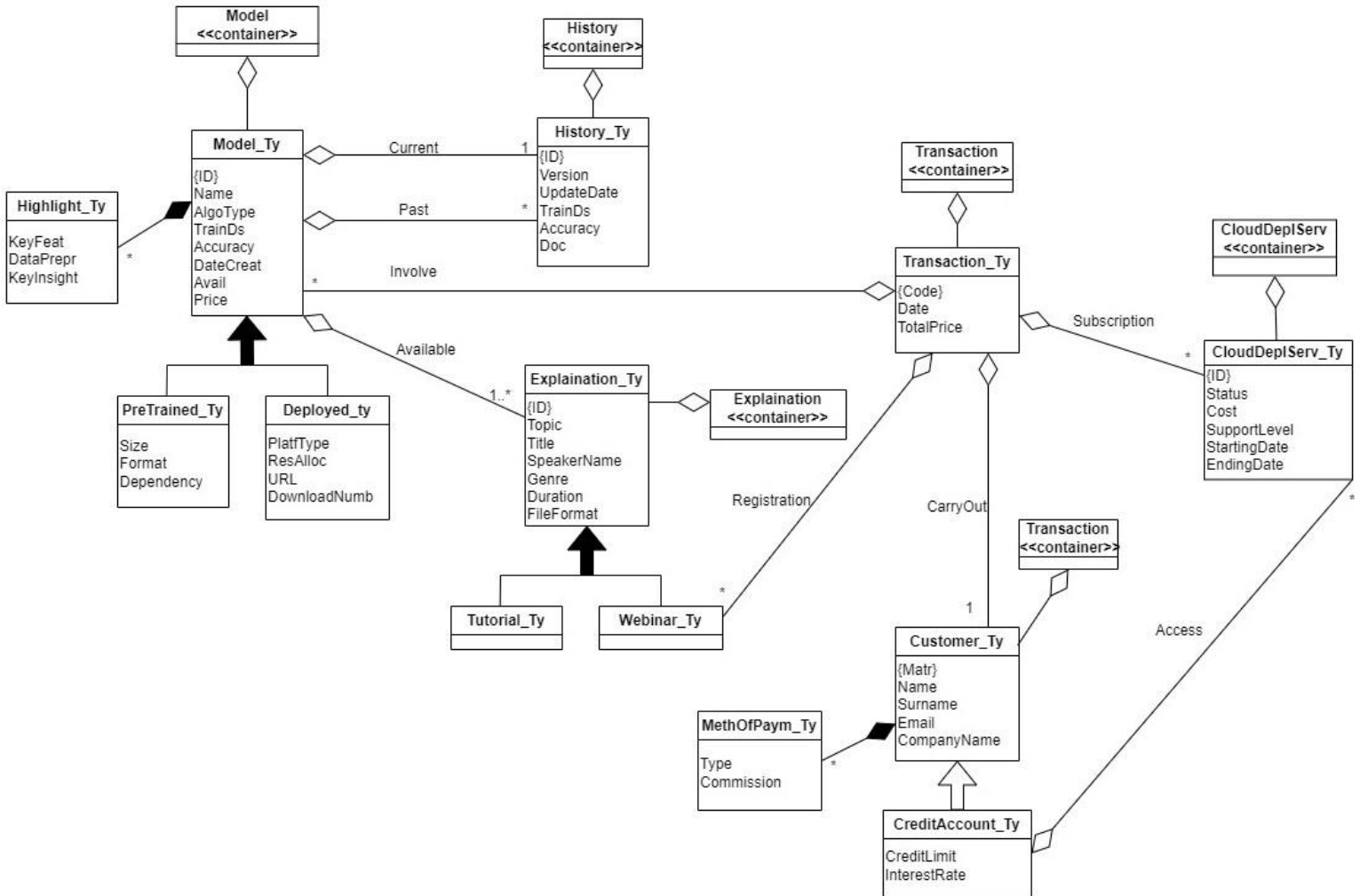


Figure2 – Logical schema

As shown in figure 2, entities have been turned into types; generalizations have been kept, because the selected paradigm allows for types to be extended into sub-types; relationships have been turned into compositions and aggregations of objects; and standalone types are contained into containers, which represent tables.

Some considerations could be:

- Model_Ty is connected to Highlight_Ty through a composition, indicating that if one model is deleted, also its highlights will be, since the latter could not exist without the former;
- The same reasoning is applied to Customer_Ty and MethOfPaym_Ty;

- Model_Ty is connected to History through two aggregations, both for current and past relationships. It was decided to keep the reference of History_Ty inside Model_Ty since it would be easier to compute operations three and four, starting from models, looking for their details inside their history references.
- Transaction_Ty is connected to Model_Ty, Webinar_Ty, Customer_Ty and CloudDeplServ_Ty through aggregations. It was decided to keep the reference of connected types inside Transaction_Ty because there's no operation working on it, and a possible one in which elements involved in one transaction are requested was thought.
- CreditAccount_Ty is connected to CloudDeplServ_Ty through an aggregation, indicating that one customer with credit account could have access to cloud deployment services. It was decided to keep the reference of CloudDeplServ_Ty inside CreditAccount_Ty since it would be easier to compute operation two, starting from the customer account, looking for those with the required subscription.

5. ORACLE

Oracle Corporation is an American multinational computer technology company, founded in 1977 by Larry Ellison, Bob Miner, and Ed Oates. The company is primarily known for its object relational database management systems (ORDBMS), but it also offers a wide range of other enterprise software and cloud computing solutions. Oracle database is one of Oracle Corporation's flagship products and is one of the most widely used database solutions in the world.

Using Oracle databases, it is possible to integrate objects in a relational model. This means being able to relax the first normal form, denoting that attributes must be of primitive types, while introducing the possibility of nested relationships. Therefore, there's the presence of a set of tuples as before, but attributes can also be of a set type. The notion of 'tuple' is replaced by the notion of 'object identity' (OID), corresponding to a unique reference for the whole database used. In conclusion, when we speak about an 'object', we can refer to it as a pair: identifier and value.

Object relational schema is a collection of definitions of object types and schemas of tables, each associated with a type. This association can be modeled by aggregation or composition, depending on the case. Therefore, in the following section we're going to discuss about the definition of types used for tuples; the definition of tables, considered as containers, that will contain the previous types; the instantiation of casual data in order to fullfill the database created; the definition of possible triggers that could be applied, and finally, the creation of indexes that could be considered as useful during the execution of operations outlined in specification section.

5.1 TYPES' CREATION

Within Oracle, it is possible to create sub-types using inheritance; however, this is not possible with tables. The substitution principle is allowed, but only with single inheritance at the type level (a type can be a sub-type of only one super-type, not multiple).

Types can be defined as 'Final', when objects of that type cannot be further specialized, or 'Not Final', otherwise. And they could also be defined as 'Instatiable', when an object of that type can be instatiated, or 'Not Instatiable', otherwise.

Regarding associations between types, aggregations have been implemented referencing appropriate OIDs into composite objects, using VARRAY when it allows multiple

elements. Whereas compositions have been implemented by instantiating object inside other objects, using TABLE when it allows multiple elements.

Types created can be found at AuroraTomaProject\Scripts\1.types.sql.

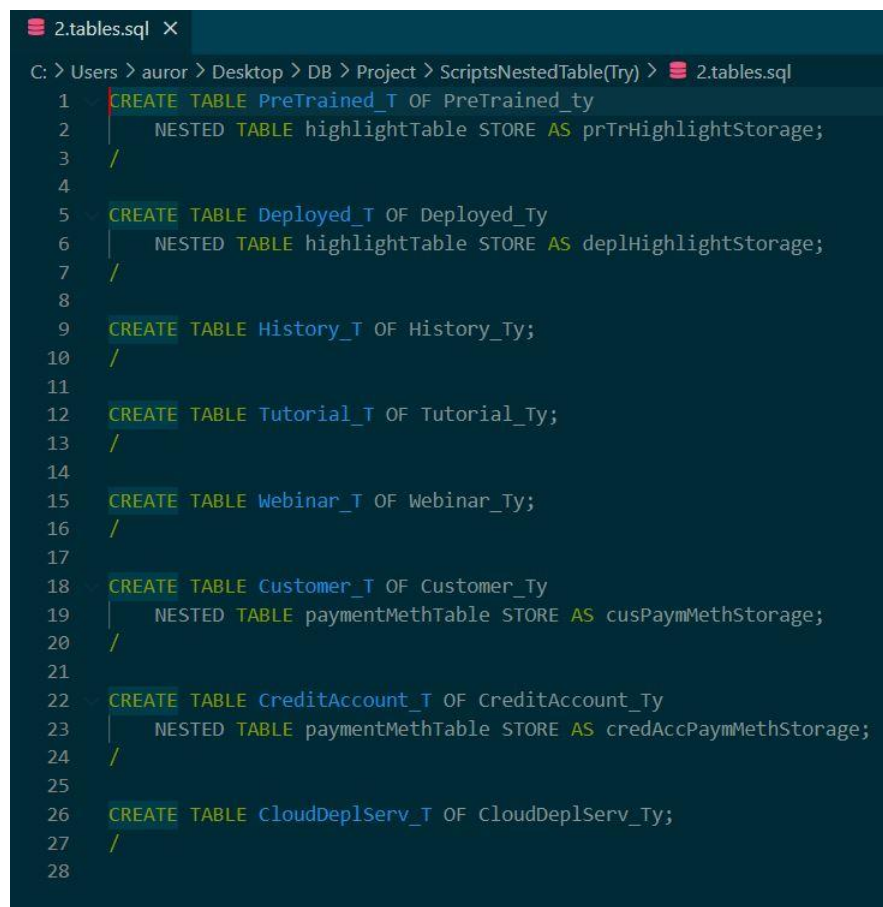
Note: in this section screenshots weren't taken due to their too big number.

5.2 CONTAINERS' CREATION

As previously mentioned, in this environment tables are considered as containers containing the defined types.

Each table consists of a set of tuples, and the type of each tuple is said *row type*, characterized by being a non atomic value. It could be implicitly or explicitly defined, and in this case the latter was chosen in order to use the defined type for different typed tables.

Screenshot of created tables follows.



```
2.tables.sql X
C: > Users > auror > Desktop > DB > Project > ScriptsNestedTable(Try) > 2.tables.sql
1 CREATE TABLE PreTrained_T OF PreTrained_ty
2 |   NESTED TABLE highlightTable STORE AS prTrHighlightStorage;
3 /
4
5 CREATE TABLE Deployed_T OF Deployed_Ty
6 |   NESTED TABLE highlightTable STORE AS deplHighlightStorage;
7 /
8
9 CREATE TABLE History_T OF History_Ty;
10 /
11
12 CREATE TABLE Tutorial_T OF Tutorial_Ty;
13 /
14
15 CREATE TABLE Webinar_T OF Webinar_Ty;
16 /
17
18 CREATE TABLE Customer_T OF Customer_Ty
19 |   NESTED TABLE paymentMethTable STORE AS cusPaymMethStorage;
20 /
21
22 CREATE TABLE CreditAccount_T OF CreditAccount_Ty
23 |   NESTED TABLE paymentMethTable STORE AS credAccPaymMethStorage;
24 /
25
26 CREATE TABLE CloudDeplServ_T OF CloudDeplServ_Ty;
27 /
28
```

Figure3 – Tables created

As shown in figure3, only instantiable tables were created in order to contain casual data that will be generated in the next step. Moreover, nested table were defined where needed, following the previous definition of types.

Note: transaction table wasn't created since it isn't needed in operations defined by specifications.

5.3 CASUAL DATA GENERATION

Before generating casual data, the definition of *sequences* with relative triggers for each instantiable table was needed, in order to realize a sequence of elements ordered by their primary key, guaranteeing their unicity.

Script containing sequences creation is located at AuroraTomaProject\Scripts\3.sequences.sql.

Next, database was fullfilled with casual data by creating some procedures for randomizing great amounts of content in the tables.

Unfortunately, specifications' indication about number of instances used for each table led to a too long time needed during the instantiation of them, due to low computational capacity given by the used computer. In particular, the instantiation of pre-trained and deployed models took a long time due to their need of instantiating elements also for nested tables contained within them. Therefore, number of models stored by the considered company was reduced from 10000 to 4000, respectively divided in 2000 pre-trained and 2000 deployed models.

During their instatiation, for each model it was decided to store five highlights per models, two possible explanations (tutorial or webinar) and two possible past histories. Whereas, during the instatiation of customer and creditAccount, it was decided to store three methods of payment, and for the second one, it was decided to store two cloud services accessed.

Script containing casual data add is located at AuroraTomaProject\Scripts\4.population.sql.

Note: in this section screenshots weren't taken due to their too big number. Nevertheless, resulting number of instances generated per table follows.

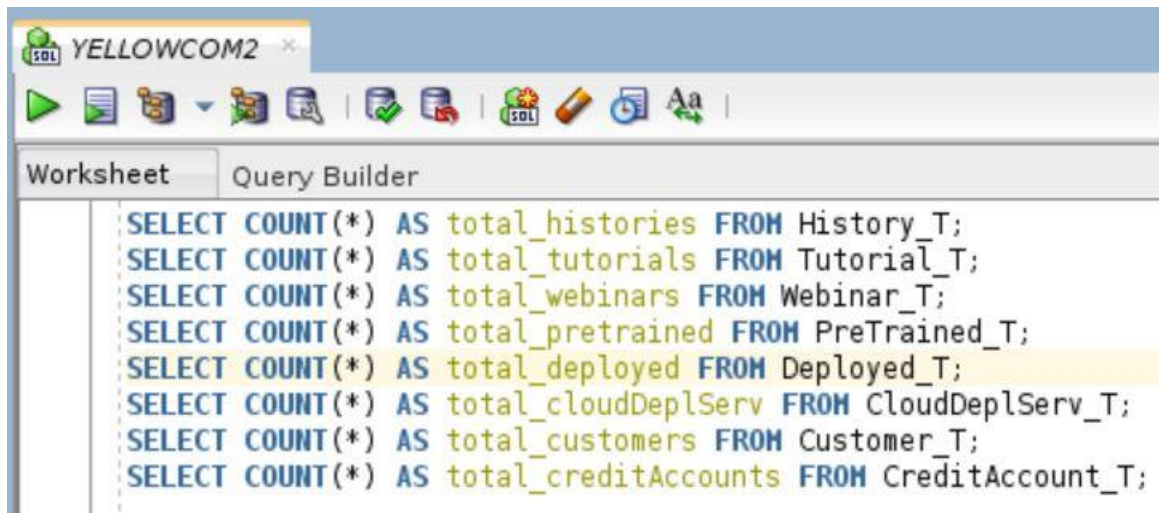


Figure4 – Queries visualizing number of instances per table

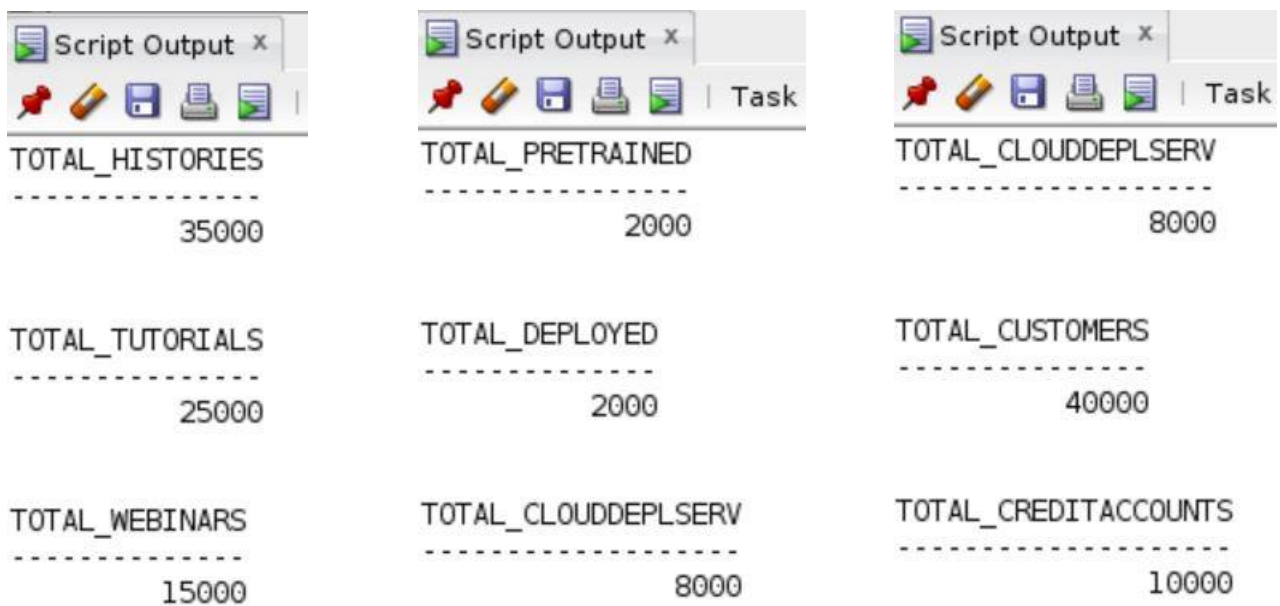


Figure5 – Visualization of number of instances per table

5.4 TRIGGERS

Triggers are useful in ensuring that when certain events or conditions are identified, specific operations are executed automatically.

Triggers adhere to the ECA paradigm, which stands for Event Condition Action. This paradigm involves their activation (Event), influenced by one of the defined events; their evaluation (Condition), where the specified conditions are assessed; and their execution (Action), applied if the previous evaluation is positive.

The presence of triggers offers several advantages: it is possible to have multiple actions for different events with the same condition, and identifying an event in large databases becomes simpler, thus avoiding CPU wastage.

Then, triggers can be classified as *Passive*, typically used to raise exceptions or define constraints; or *Active*, typically used to change the status of the database. They can also be defined as *Conditioners* (Before triggers) if used to condition the values utilized by an editing operation, or as *Re-Installers* (After triggers) if used to react to certain changes made by other operations.

As regards Oracle triggers, they can be activated by more than one event, differently from others DBMS like SQL3, in which one single event can activate them. Moreover, within them DBMS can work only with *new* and *old*, denoting the new row inserted or modified, or the old row before deletion or modification, differently from others DBMS like SQL3, in which also *new table* and *old table* were considered. Finally, another difference with SQL3 is that Oracle can operate on tables and views, while the former only with tables.

5.4.1 CONSIDERED TRIGGERS

In the case studied, possible examples of triggers that could be considered as usefull are:

- When a new pre-trained model is inserted, its size must be greater than zero;
- When a new cloud deployed service is inserted, its starting date must be less than its ending date;
- When a model (pre-pretrained or deployed) is inserted or modified, its accuracy must be in the range of 0-1;
- When a cloud deployed service cost is updated, we can assume that if it exceeds a certain threshold the support level of the considered cloud deployed service is

updated. In this case we could assume that with a cost less than 500, support level corresponds to standard; premium otherwise;

- When a customer is inserted or updated, we could assume that it can have at most 5 methods of payment.

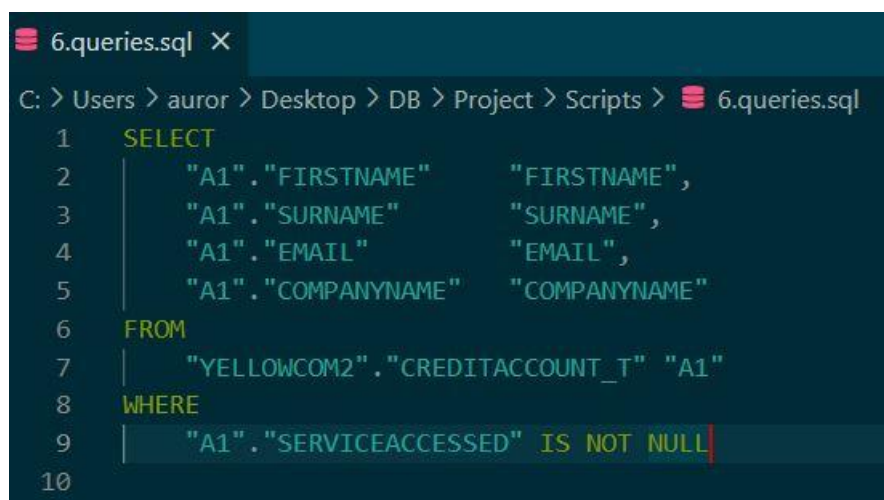
Script containing triggers definition is located at AuroraTomaProject\Scripts\5.triggers.sql.

Note: in this section screenshots weren't taken due to their too big number.

5.5 QUERIES AND INDEXES

In this section, queries have been defined, corresponding to the operations outlined in the specifications.

- **Operation 2:** Select the accounts with a subscription to cloud deployment services.



```
6.queries.sql X
C: > Users > auror > Desktop > DB > Project > Scripts > 6.queries.sql
1  SELECT
2      "A1"."FIRSTNAME"      "FIRSTNAME",
3      "A1"."SURNAME"        "SURNAME",
4      "A1"."EMAIL"          "EMAIL",
5      "A1"."COMPANYNAME"    "COMPANYNAME"
6  FROM
7      "YELLOWCOM2"."CREDITACCOUNT_T" "A1"
8  WHERE
9      "A1"."SERVICEACCESSED" IS NOT NULL
10
```

Figure6 – Operation2

Since the insertion of instances was random, no null row was defined and resulting rows corresponds to all possible visible rows, as shown in figure 7.

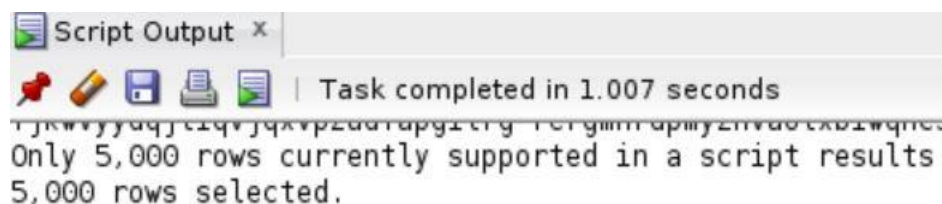


Figure7 – Resulting rows of operation2

- **Operation 3:** Request information on models without corresponding documentation.

```
10 (SELECT
11     "A1"."MODELNAME"    "MODELNAME",
12     "A1"."ALGOTYPE"     "ALGOTYPE",
13     "A1"."TRAINDS"      "TRAINDS",
14     "A1"."ACCURACY"      "ACCURACY",
15     "A1"."DATECREAT"    "DATECREAT",
16     "A1"."AVAIL"        "AVAIL",
17     "A1"."PRICE"        "PRICE"
18 FROM
19     "YELLOWCOM2"."PRETRAINED_T" "A1"
20 WHERE
21     "A1"."CURRENTHIST"."DOC" IS NULL
22 )
23 UNION ALL
24 (SELECT
25     "A1"."MODELNAME"    "MODELNAME",
26     "A1"."ALGOTYPE"     "ALGOTYPE",
27     "A1"."TRAINDS"      "TRAINDS",
28     "A1"."ACCURACY"      "ACCURACY",
29     "A1"."DATECREAT"    "DATECREAT",
30     "A1"."AVAIL"        "AVAIL",
31     "A1"."PRICE"        "PRICE"
32 FROM
33     "YELLOWCOM2"."DEPLOYED_T" "A1"
34 WHERE
35     "A1"."CURRENTHIST"."DOC" IS NULL
36 )
```

Figure8 – Operation 3

As shown in figure 9, also in this case it happened that all rows related to models were returned, due to casual generation data.

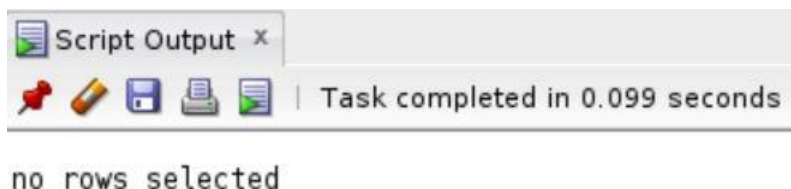


Figure9 – Resulting rows of operation3

Nevertheless, in this case a counter proof was made, in order to verify if all models actually have documentation inserted. Results of this proof are shown in figure 10.

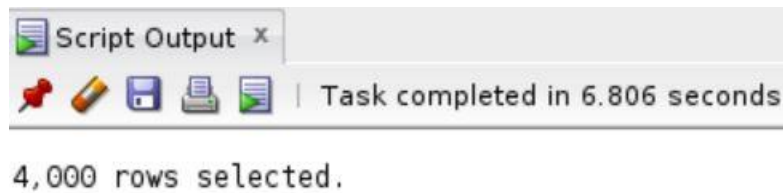


Figure10 – Resulting rows of counter operation3

- **Operation 4:** Print the history and details of models using less than 10 features.

```

40  ( SELECT
41      "A3"."MODELNAME"      "MODELNAME",
42      "A3"."ALGOTYPE"       "ALGOTYPE",
43      "A3"."TRAINDS"        "TRAINDS",
44      "A3"."ACCURACY"        "ACCURACY",
45      "A3"."DATECREAT"      "DATECREAT",
46      "A3"."AVAIL"          "AVAIL",
47      "A3"."PRICE"          "PRICE",
48      "A3"."HIGHLIGHTTABLE" "HIGHLIGHTTABLE",
49      "A3"."EXPLANATION"    "EXPLANATION",
50      "A3"."CURRENTHIST"    "CURRENTHIST"
51  FROM
52      "YELLOWCOM2"."DEPLOYED_T" "A3"
53  WHERE
54      EXISTS (
55          SELECT
56              "A5"."KEYFEAT" "KEYFEAT"
57          FROM TABLE(A3.HIGHLIGHTTABLE) A5
58          WHERE
59              "A5"."NESTED_TABLE_ID" = "A3"."SYS_NC0001100012$"
60              AND "A5"."KEYFEAT" < 10
61      )
62  )
63  UNION ALL
64  ( SELECT
65      "A2"."MODELNAME"      "MODELNAME",
66      "A2"."ALGOTYPE"       "ALGOTYPE",
67      "A2"."TRAINDS"        "TRAINDS",
68      "A2"."ACCURACY"        "ACCURACY",
69      "A2"."DATECREAT"      "DATECREAT",
70      "A2"."AVAIL"          "AVAIL",
71      "A2"."PRICE"          "PRICE",
72      "A2"."HIGHLIGHTTABLE" "HIGHLIGHTTABLE",
73      "A2"."EXPLANATION"    "EXPLANATION",
74      "A2"."CURRENTHIST"    "CURRENTHIST"
75  FROM
76      "YELLOWCOM2"."PRETRAINED_T" "A2"
77  WHERE
78      EXISTS (
79          SELECT
80              "A7"."KEYFEAT" "KEYFEAT"
81          FROM TABLE(A2.HIGHLIGHTTABLE) A7
82          WHERE
83              "A7"."NESTED_TABLE_ID" = "A2"."SYS_NC0001100012$"
84              AND "A7"."KEYFEAT" < 10
85      )
86  );

```

Figure11 – Operation4

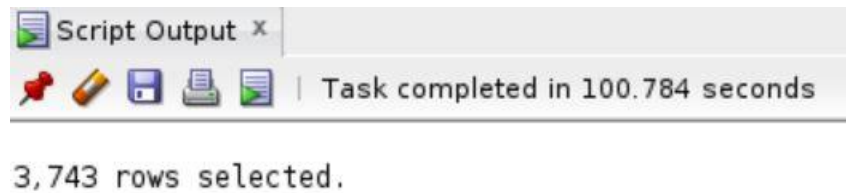


Figure12 – Resulting rows of operation4

- **Operation 5:** Print all the summaries of the models created by the developer of the model with the highest accuracy.

```

88  ( SELECT
89      "A5"."MODELNAME"      "MODELNAME",
90      "A4"."KEYFEAT"        "KEYFEAT",
91      "A4"."DATAPREPR"      "DATAPREPR",
92      "A4"."KEYINSIGHT"     "KEYINSIGHT"
93  FROM
94      "YELLOWCOM2"."DEPLOYED_T"          "A5",
95      TABLE (A5.HIGHLIGHTTABLE)        A4
96  WHERE
97      "A4"."NESTED_TABLE_ID" = "A5"."SYS_NC0001100012$" AND
98      "A5"."ACCURACY" = 1
99  )
100 UNION ALL
101 ( SELECT
102     "A3"."MODELNAME"      "MODELNAME",
103     "A2"."KEYFEAT"        "KEYFEAT",
104     "A2"."DATAPREPR"      "DATAPREPR",
105     "A2"."KEYINSIGHT"     "KEYINSIGHT"
106  FROM
107      "YELLOWCOM2"."PRETRAINED_T"        "A3",
108      TABLE (A3.HIGHLIGHTTABLE)          A2
109  WHERE
110      "A2"."NESTED_TABLE_ID" = "A3"."SYS_NC0001100012$" AND
111      "A3"."ACCURACY" = 1
112 );

```

Figure13 – Operation5

In this case, ‘summaries’ correspond to information of highlights of models having the highest possible accuracy (exactly one).

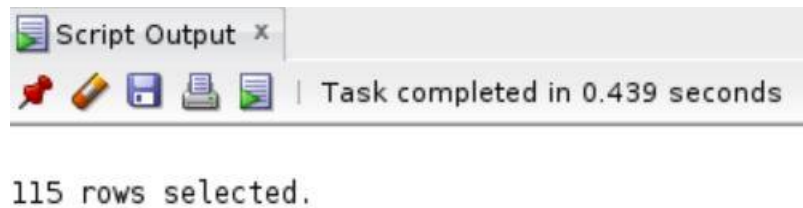


Figure14 – Resulting rows of operation5

Finally, the definition of indexes has been applied in certain cases in order to improve performances.

Indexes can be considered as auxiliary structures used to efficiently access to tuples on the basis of values of a field called *key*. They can be dense, containing one record for each value of the key field; or sparse, containing a lower number of records than the number of values of the key field.

We can distinguish *primary structure* physically containing data, based on *multilevel primary index*, using a sparse index (because they reflect the way data are stored inside primary block); from *secondary structure*, implemented to facilitate access to data without containing them, based on *multilevel secondary index*, using dense index (because they don't reflect the way data are stored inside primary block, but on the basis of the previously mentioned key field).

As regards Oracle's primary structure, it's based on heap files, using hash functions only with clusters, and the latter are ordered with a dense B-Tree. Whereas, for secondary structure indexes of various types are used like B-Tree, bitmap, hash function.

Unfortunately, as said in section 5.3, little data were inserted due to low computational capabilities of computer used. Therefore, DBMS can't exploit any possible defined indexes because they're used when millions or billions of data are inserted. Nevertheless, they were created for any possible future use.

Regarding **operation2**, we could think of applying an index on the ID of cloud deployed services aggregated with credit accounts, since it checks directly if it's null or not. Nevertheless, ID corresponds to the primary key in cloud deployed service, and then it will already be sparsely indexed with a primary structure.

Regarding **operation3**, we could think of applying an index on documentation attribute of history table. Since this operation checks directly if it's null or not, an index on it could improve the access on models without documentation.

Regarding **operation4**, we could think of applying an index on key feature attribute of highlight. Since this operation checks directly if its value is less than ten, an index on it could improve the access on models having less than ten key features.

Finally, regarding **operation5**, we could think of applying an index on accuracy attribute of model table. Since this operation checks directly if its value is equal to one (maximum accuracy), an index on it could improve the access on models having maximum accuracy.

6. DATA WAREHOUSE

A data warehouse is a collection of data exploited during management's decisions. Within it, different *data charts* could be defined, depending on the specific sector considered. It's also possible to define the notion of *fact*, corresponding to the main concept on which analyses are performed. Moreover, they're characterized by some *features* called measures describing them. Eventually, another notion that could be specified is *dimension*, corresponding to particular perspectives of analyses considered, associated with the chosen fact.

In this case, a possible example of fact could be a model, characterized with some measures like id, price and accuracy. Moreover, possible dimensions to take into account could be the date of creation of the model, the type of the model (pre-trained or deployed), and the algorithm type.

Script containing data warehouse definition is located at AuroraTomaProject\Scripts\8.dataWarehouse.sql.

Note: in this section screenshots weren't taken due to their too big number. Nevertheless, a star schema used to depict the designed data warehouse follows.

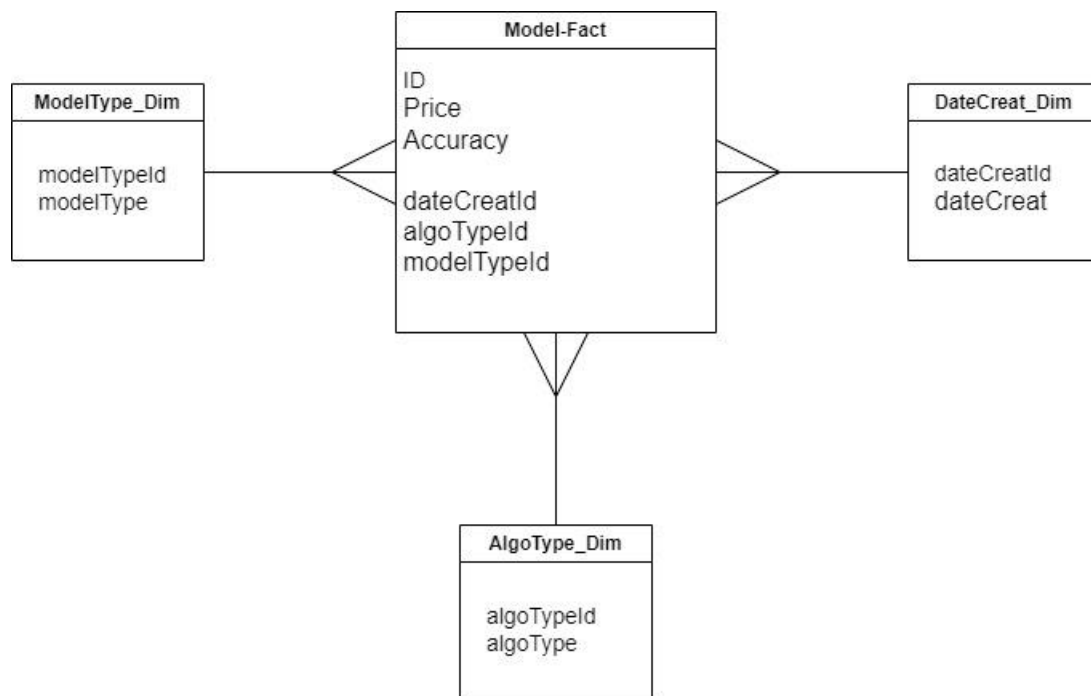


Figure15 – Data warehouse star schema

CONCLUSION

In conclusion, for the company MLSolutions, an initial requirements analysis was conducted, following the necessary steps to gain a clear understanding of the specifications provided by the company.

Subsequently, a conceptual schema was developed, depicting a preliminary design of the database to be used by the company. This conceptual schema was then converted into a logical schema using UML, adhering to the relevant rules.

Finally, the design was implemented using an Oracle ORDBMS. Types and corresponding instantiable tables deemed useful were created. Random data were instantiated for each of these tables, although due to the limited computational capacity of the computer used for the work, only a few instances were created.

Next, queries that met the operations defined in the company's specifications were created, and an idea of possible applicable indexes was formed, although these were not actually implemented due to the limited data available.

Finally, a data warehouse was designed and implemented, considering a model as a fact with id, price and accuracy as measures, while associated to dimensions like algorithm type, model type and creation date.