# *Plantsdom* - Database Systems Project

by Greco Francesco

a. Y. 2020/21 - Database Systems

Prof. Michelangelo Ceci

## Index

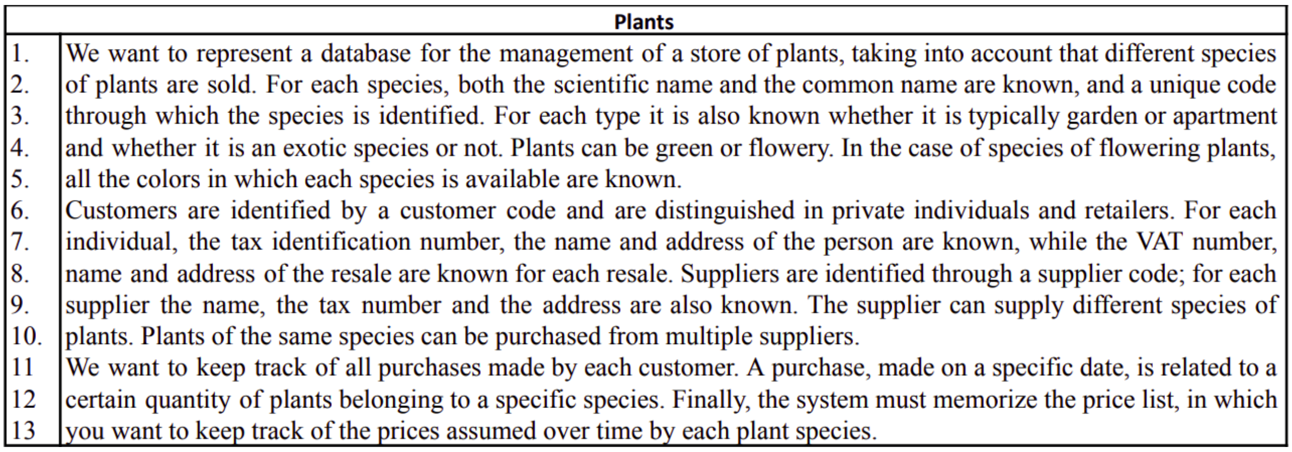
# 1 Introduction

This is the documentation of the final project for the Database Systems course of the “Computer Science” master degree, a.Y. 2020/2021, held by Professor M. Ceci in University of Bari “A. Moro”. The project is based on the specifications given in the written exam of 07/04/2021.

In section 2 the conceptual schema will be explained and the respective diagrams will be reported; in section 3 the logical schema will be reported, as well as the table of accesses and volumes; in section 4 the physical schema will be reported, explaining the choices made; in section 5 the SQL code for the triggers will be reported; in section 6 the additional implemented features will be discussed; in section 7 conclusions will be drawn, also discussing missing features and possible future works on the application.

## 1.1 Specifications

Here are reported the specifications of the database system to develop:



| | Plants |
|---|---|
| 1. | We want to represent a database for the management of a store of plants, taking into account that different species |
| 2. | of plants are sold. For each species, both the scientific name and the common name are known, and a unique code |
| 3. | through which the species is identified. For each type it is also known whether it is typically garden or apartment |
| 4. | and whether it is an exotic species or not. Plants can be green or flowery. In the case of species of flowering plants, |
| 5. | all the colors in which each species is available are known. |
| 6. | Customers are identified by a customer code and are distinguished in private individuals and retailers. For each |
| 7. | individual, the tax identification number, the name and address of the person are known, while the VAT number, |
| 8. | name and address of the resale are known for each resale. Suppliers are identified through a supplier code; for each |
| 9. | supplier the name, the tax number and the address are also known. The supplier can supply different species of |
| 10. | plants. Plants of the same species can be purchased from multiple suppliers. |
| 11 | We want to keep track of all purchases made by each customer. A purchase, made on a specific date, is related to a |
| 12 | certain quantity of plants belonging to a specific species. Finally, the system must memorize the price list, in which |
| 13 | you want to keep track of the prices assumed over time by each plant species. |

*Figure 1: Database specifications*

The operations that regard the database system are the following:

***Op1***: *Register of new purchases (300 times a day)*

***Op2***: *Register a new sale (1000 times a day)*

***Op3***: *Printing of the number of plants and properties of plants available for a given species (10000 times a day)*

***Op4***: *Printing of the purchases of a given customer (150 times a day)*

***Op5***: *Printing of the plants, ordered by the number of items sold (10 times a month)*

# 2 Conceptual Design

## 2.1 Synonyms and homonyms

We begin by identifiying synonyms:

- “**plant**”,“**species**”, “**species of plants**” and “**plant species**” are all synonyms. In line 3 the word “**type**” is used as well to refer to the concept of “plant”. In line 12 “**plants belonging to a specific species**” is just another way to refer to the concept of “plant”.

- “**private individuals**” and “**individual**” are used as synonyms.
- “**retailers**” and “**resale**” are synonyms.

## 2.2 Ambiguities filtering

In line 7, we assume that with “name” is intended the pair *name* and *surname* of a customer.

In line 11, we assume that other than the “date”, also the *time* of the purchase has to be stored.

We assume that a Plant can be exclusively Green or Flowery.

For Customers, as well, we can deduce the division into the two groups is exclusive.

It isn’t clear if a purchase can refer only to a single species of plants or to many of them. With common sense, a customer can buy more than just a single species in a purchase, so we will follow this idea in the design.

To keep track of the prices assumed by each plant over time, we suppose that it’s needed to store, for each plant, a price list containing: the amount (in euros), the start date and the ending date for which that price has been valid for the plant; the ending date is optional so that a price with no ending date is considered being the current price.

## 2.3 Standardized sentences

To have a better understanding of the specifications, we restructure the sentences of the specifications in order to have a standard and organized form for each concept.

- For **plants** we want to represent the *scientific name*, the *common name*, a *unique code* (ID), the *type* (“garden” or “apartment”), the fact that one is *exotic* or not and the *price lists* (containing both the current price and the old prices). A plant can be either *green* or *flowery*.

- For **flowering plants** we want to represent the possible *colors*.
- For **price lists** we want to store the amount (in euros) of the *price* for a single plant, the *start date* and the *ending date* (optional) for which that price has been valid for the plant.
- For **customers** we want to represent the customer code and the *purchases* they make. A customer can be either a *private individual* or a *retailer*.
- For **private individuals** we want to represent the *tax identification number*, the *name* and the *address*.
- For **retailers** we want to represent the *VAT number*, the *name* and the *address*.
- For **purchases** we want to represent the *date* and *time* of the purchase, the *plants* and the respective *quantity* purchased.
- For **suppliers** we want to represent the *supplier code*, the *name*, the *tax identification number*, the *address* and the *plants* they supply.

## 2.4 Glossary

| **Term** | **Description** | **Synonyms** | **Connections** |
|---|---|---|---|
| Plants | A plant sold in the store. A plant is supplied by one or more suppliers and it can be purchased by many customers. A plant can be green or flowery. It has a *scientific name*, a *common name*, a *unique code*, a *type* ("garden" or "apartment") and the *exoticness* ("yes" or "no"). | Species, species of plants, plant species | Listing prices, Green plants, Flowery plants, Suppliers, Purchases |
| Green plants | A type of plant. It has no additional properties with respect to plants. | - | Plants, Purchases, Listing prices |

| | | | |
|---|---|---|---|
| Flowery plants | A type of plant. It has one or more possible *colors*. | - | Plants, Purchases, Listing prices, Colors |
| Customers | A customer of the store. It can be a private individual or a retailer. A customer can make one or more purchases in which it buys one or more plants. It has a *customer code*. | - | Private individuals, Retailers, Purchases |
| Private individuals | A type of customer, which is a physical person who buys plants for him/herself. A private individual has a *tax identification number*, a name, a *surname* and an *address*. | Private | Customer, Purchases |
| Retailers | A type of customer, which is a trader who buys plants to resell them. A retailer has a *VAT number*, a *name* and an *address*. | Resale | Customer, Purchases |
| Suppliers | A supplier, from which the store buys one or more species of plants it sells. A supplier has a *supplier code*, a *name*, a *tax identification number*, an *address* and the *plants* he/she supply. | - | Plants |
| Price lists | The list of prices for a plant. It contains both the current price and the old prices for the plant. | - | Plants |

## 2.5 Conceptual E-R Schema

Here is reported the Conceptual Entity-Relationship schema for the system. It was designed with a bottom up methodology, given that the system doesn't involve a lot of entities and relationships:

indeed, no difficulties were encountered in the merging phase.
A sidenote on notation: the multi-field keys are represented by multiple single primary keys. Thus, if an entity has more than one primary key in the schema, they are not to be considered many candidate keys, but one multi-attribute primary key.

The E-R schemas in this documentation are made with the software JDER1.41 updated by Dr. Gianvito Pio (originally developed by Alessandro Ballini). The original schema files will be available as attachments.

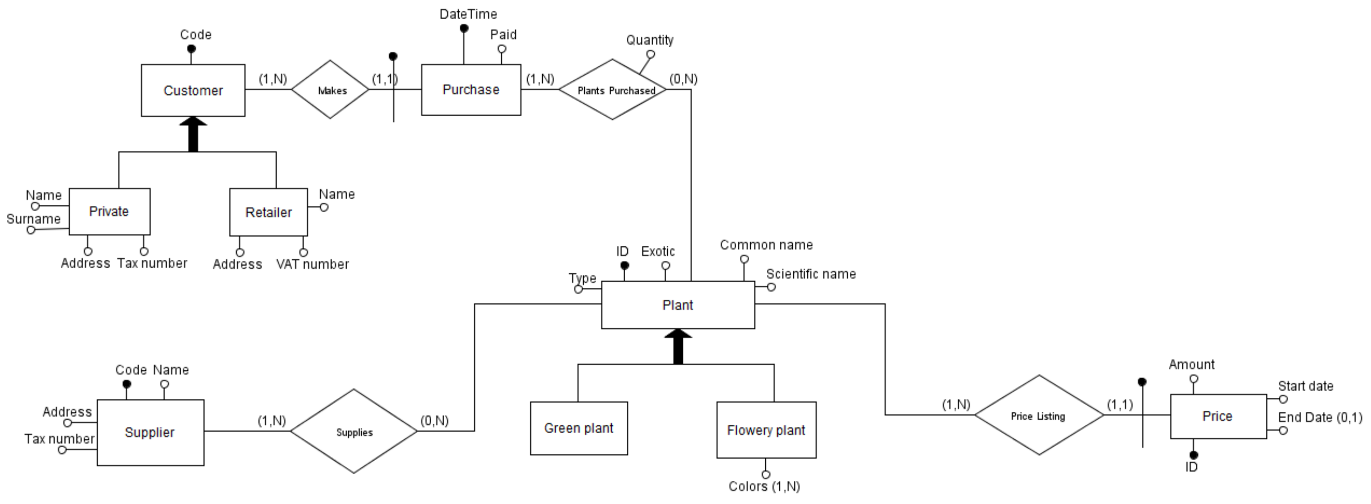


*Figure 2: Conceptual E-R Schema*

## 2.6 Project Extension: Supplies

In the specification the role of the Supplier is defined as a static concept, being like just a list of suppliers that supply certain species of plants to the store. As a project extension, this concept is going to be revised to be more in line with the Operation 1 that will be seen in Section 3: in fact we will introduce some entities, relationships and attributes that will better capture the dynamicity of the supplies. Some concepts will be managed in the system with this extension:

1. The Plants stock: the Plant entity will have a *Stock* attribute which represents its quantity in stock;
2. The Supply concept: a Supplier sells to the store a stock of plants, containing one or more differente species of plants; a Supply can be considered as an order placed by the plants store.
3. Regarding a Supply, the *price* at which the plants are bought by the store: each supply contains a certain quanitity of a specified plant, which is bought at a certain price. Also the total cost of the supply can be stored (this redundancy will be better analyzed in Section 3).
4. The *date of purchase* of a Supply, that is the date (and time) in which the store makes the order for the supply.
5. The *date of arrival* of a Supply, which is the estimate date in which the supply will physically arrive at the store.

In the Figure below is reported the Conceptual E-R schema including these extensions. We can observe in particular the new *In stock* attribute of Plants and the reification of the *Supplies* relationship that was present in the first Conceptual schema.

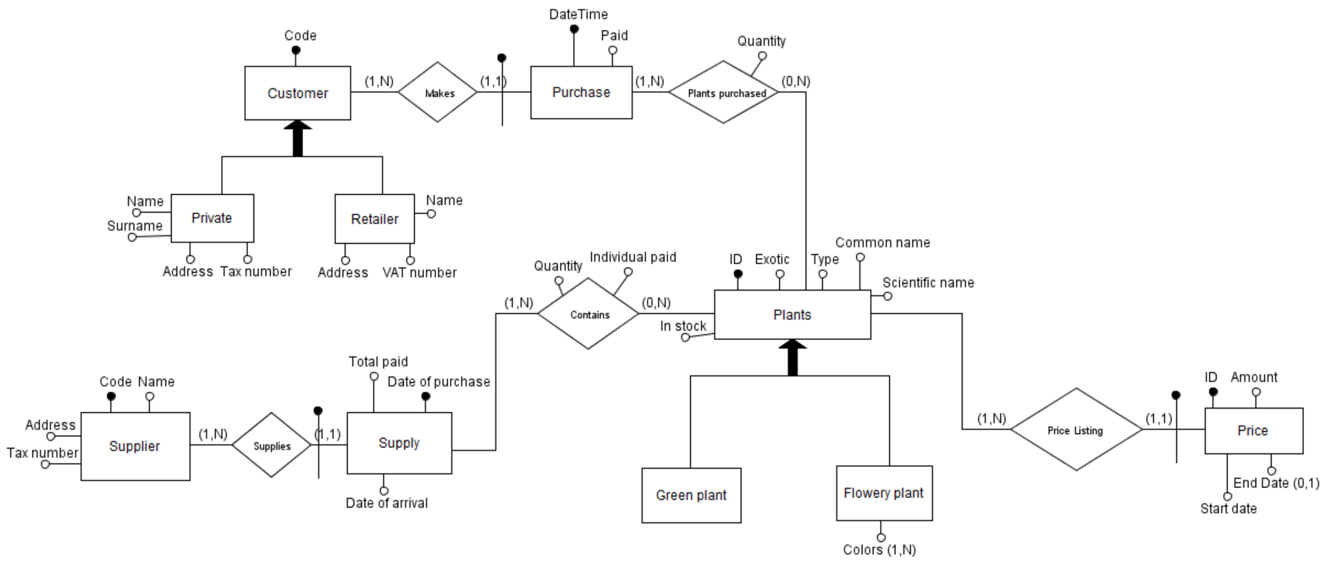


*Figure 3: Conceptual E-R Schema with the "Supply" Extension*

## 2.7 Other constraints not expressed by the ER model

- In Supply we must have Date of arrival > Date of purchase;
- In Price we must have End Date > Start Date, and each Price must have and End Date, apart from 1 and 1 only, which is the current price. Moreover, apart from the first (in order by date) price, the Start date of each price must be the end date of another one.
- The Total paid of Supply must be equal to the sum of the Individual paid of each record in Contains, multiplied by the respective quantity.
- Quantities cannot be <1.

# 3 Logical Design

In this section we are going to consider the translation of the Conceptual E-R schema defined in the previous section with a Logical schema (made in UML) for an Object-Relational model. Since we are not opting for a pure relational model, there's no need to restructure the Conceptual schema by removing generalizations and composite attributes. However, we decide to remove the generalization on Plant by collapsing the child entities into the parent, since all the operations involve both the children and the waste of memory should not be very impactful. Moreover, the Green Plant entity doesn't have any additional attribute. The attributes will be modified in this way:

- the attribute *Garden*: which is the old *type* attribute of Plant; it indicates if it is a garden plant or an apartment plant;
- the attribute *Plant type* is introduced to distinguish between flowery and green plants was introduced;
- the *Colors* attribute now has a 0..N cardinality, so that green plants can have no value for that attribute.

The other generalization, on Customer, can be solved in the same way as before, since every operation involve both the types of Customer. In this way the join operations between Private Individuals and Retailers are minimized. We can also find a way to minimize the waste of memory by using the same attributes for representing slightly different concepts:

- attribute *Name*: will be long enough to represent both the name of a Retailer and the first and last name of a Private Individual;
- attribute *Address*: is the same for both children entities;
- attribute *Identification Number*: will contain either the Tax number for Privates, or the VAT number for Retailers;
- finally an attribute *Type* will be introduced to distinguish between the two.

## 3.1 Volumes table

We assume **1 year of activity** of the system.

| **Concept** | **Type** | **Volume** | **Notes** |
|---|---|---|---|
| Plant | E | 800 | Volume given in the specifications |
| Green Plant | E | 400 | Assuming 50% of Plants |
| Flowery Plant | E | 400 | Assuming 50% of Plants |
| Supplier | E | 500 | Assumed by hypothesis |
| Supplies | R | 108’000 | From Op.1 we know that each day we have 300 purchases from suppliers. Assuming 1 year of life of the system: 300*30*12 |
| Supply | E | 108’000 | 1-1 relationship with Supplies |
| Contains | R | 324’000 | Assuming each Supply contains 3 different plants in average |
| Price | E | 8’000 | Assuming that after 1 year the price for a plant has changed in average 10 times. |
| Price Listing | R | 8’000 | 1-1 relationship with Price |

|  |  |  |  |
|---|---|---|---|
| Customer | E | 1’200 | Volume given in the specifications |
| Private | E | 400 | Assuming 1/3 of the Customers |
| Retailer | E | 800 | Assuming 2/3 of the Customers |
| Makes | R | 360’000 | From Op.2 we know that each day we have 1000 purchases. Assuming 1 year of activity of the system: 1000*30*12 |
| Purchase | E | 360’000 | 1-1 relationship with Makes |
| Plants purchased | R | 1’080’000 | Assuming that in average a purchase involves 3 plants |

## 3.2 Access tables

Here, for each of the queries given in the specifications, the relative access table will be reported. We assume that a Write operation weights double the cost of a Read operation.

### 3.2.1 Operation 1

*Register of new purchases (300 times a day)*

| Concept | E/R | #Accesses | Read/Write | Notes |
|---|---|---|---|---|
| Supplier | E | 1 | R | Read the supplier from which the store is buying |
| Supplies | R | 1 | W | Connect the Supplier to the Supply it supplies |
| Plant | E | 3 | R | Read the plants that must be supplied (the “In stock” attribute will not be written in this moment) |
| Contains | R | 3 | W | Writes the plants that are being purchased |
| Supply | E | 1 | W | Finally writes the Supply entity |

- Cost (Op1) = 1 + 1*2 + 3 + 3*2 + 1*2 = 14 accesses

- Daily cost (Op1) = 300 * cost = 4200 accesses/day

### 3.2.2 Operation 2

*Register a new sale (1000 times a day)*

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer that is making the purchase |
| Makes | R | 1 | W | Connect the Customer to its Purchase |
| Plant | E | 3 | R | Read the plants that must be purchased (we assumed that in average a purchase involves 3 plants) |
| Price Listing | R | 30 | R | In average there are 10 prices per plant |
| Price | E | 30 | R | 1-1 relationship with Price Listing |
| Plant | E | 3 | W | Update the “In stock” attribute of the purchased plants |
| Plants purchased | R | 3 | W | Connect the Plants to the Purchase |
| Purchase | E | 1 | W | Finally write the purchase |

- Cost (Op2) = 1 + 1*2 + 3 + 30 + 30 + 3*2 + 3*2 + 1*2 = 80 accesses
- Daily cost (Op2) = 1000 * cost = 80’000 accesses/day

Let’s try including the redundant attribute *Price* on Plant with the current price of the plant: it is a redundancy because it is also present in the *Price* entity (is the one without the attribute *End Date*).

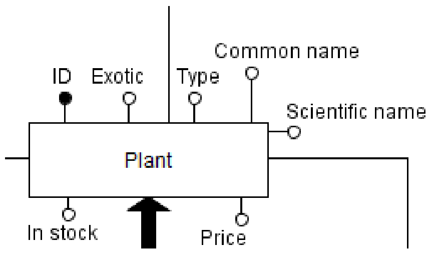


*Figure 4: Plant entity with the Price attribute*

The accesses table for Operation 2 would become:

| Concept | E/R | #Accesses | Read/Write | Notes |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer that is making the purchase |
| Makes | R | 1 | W | Connect the Customer to its Purchase |
| Plant | E | 3 | R | Read the plants that must be purchased (we assumed that in average a purchase involves 3 plants). It has now also the current *Price* |
| Plant | E | 3 | W | Update the “In stock” attribute of the purchased plants |
| Plants purchased | R | 3 | W | Connect the Plants to the Purchase |
| Purchase | E | 1 | W | Finally write the purchase |

With the redundancies the costs are:

- Cost (Op2) = 1 + 1*2 + 3 + 3*2 + 3*2 + 1*2 = 20 accesses
- Daily cost (Op2) = 1000 * cost = 20’000 accesses/day

### 3.2.3 Operation 3

*Printing of the number of plants and properties of plants available for a given species (10000 times a day)*

| Concept | E/R | #Accesses | Read/Write | Notes |
|---|---|---|---|---|

|  |  |  |  |  |
|---|---|---|---|---|
| Plant | E | 1 | R | Read the species of plants to print, with all of these attributes |
| Price Listing | R | 10 | R | Assuming that each plant has in average 10 prices |
| Price | E | 10 | R | Accessing the prices of the plant |

- Cost (Op3) = 1 + 10 + 10 = 21 accesses
- Daily cost (Op3) = 10’000 * cost = 210’000 accesses/day

Considering the previous redundancy of the *Price* attribute on Plant, we would have:

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Plant | E | 1 | R | Read the species of green plant to print, with all of these attributes |

The costs would be:

- Cost (Op3) = 1 access
- Daily cost (Op3) = 10’000 * cost = 10’000 accesses/day

### 3.2.4 Operation 4

*Printing of the purchases of a given customer (150 times a day)*

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer data |
| Makes | R | 3’000 | R | From the table of volumes we have that in average a customer has made 360’000/1200 = 3000 purchases. |
| Purchase | E | 3’000 | R | Reading each purchase for the given customer |
| Plants purchased | R | 9’000 | R | Accessing the purchased plants (in average 3 per purchase) |

- Cost (Op4) = 1 + 3000 + 3000 + 9000 = 15’001 accesses
- Daily cost (Op4) = 150 * cost = 2’250’150 accesses/day

This operation is truly problematic, given its high cost and high daily frequence. We can try to break down the N to N relationship between Purchase and Plant to make every Purchase regard a single Plant; a Purchase with more than one plant bought will be split in two or more purchases with the same date and customer, but different plant.

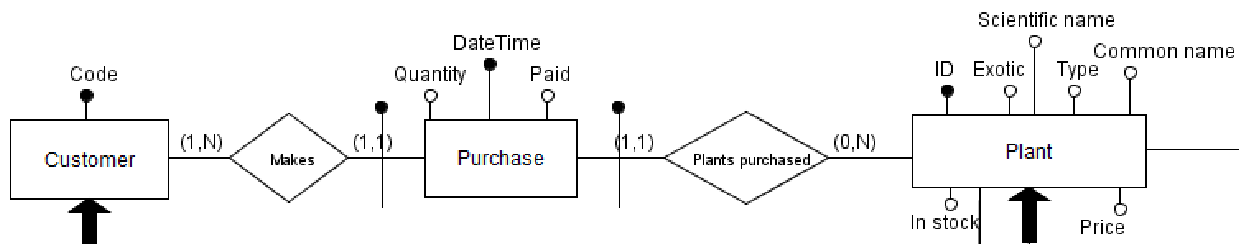


*Figure 5: Restructured Purchase in the E-R schema*

Now we have more records in Purchase:

| **Concept** | **Type** | **Volume** | **Notes** |
|---|---|---|---|
| Makes | R | 1’080’000 | From Op.2 we know that each day we have 1000 purchases. Assuming 1 year of activity of the system: 1000*30*12. Assuming that in average a purchase involves 3 plants: 1000*30*12* 3 |
| Purchase | E | 1’080’000 | 1-1 relationship with Makes |
| Plants purchased | R | 1’080’000 | 1-1 relationship with Purchase |

In the accesses table for Operation 4, however we would have:

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer data |
| Makes | R | 9’000 | R | Reading each purchase for the given customer |
| Purchase | E | 9’000 | R | Accessing the purchase data |

And the cost would still be high (actually even higher than before).

But since we are using an Object-Relational model, we can take advantage of the aggregation concept and incorporate the Purchases of one Customer to the Customer itself: in this way Operation 4 will become much more efficient. We would have the following accesses table:

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer data, including their Purchases |
| Purchase | E | 9’000 | R | Reading each purchase for the given customer |

We wouldn’t have to navigate the *Makes* relationship and the cost would be of:

- Cost (Op4) = 1 + 9000 = 9’001 accesses
- Daily cost (Op4) = 150 * cost = 1’350’150 accesses/day

Aggregating at yet another level, that is, clumpsing together the records involving the same purchase (but regarding different plants), theoretically we could make the cost 3 times lower: this is due to the fact that, considering that a purchase involves in average 3 different plants, we could just access to the purchase and get all 3 different plants references together.

| **Concept** | **E/R** | **#Accesses** | **Read/Write** | **Notes** |
|---|---|---|---|---|
| Customer | E | 1 | R | Read the customer data, including their Purchases |
| Purchase | E | 3’000 | R | Reading each purchase for the given customer: this time each purchase includes the |

| | | | | |
|---|---|---|---|---|
| | | | | full transaction (each plant purchased with the quantity and the price) |

- Cost (Op4) = 1 + 3000 = 3’001 accesses
- Daily cost (Op4) = 150 * cost = 450’150 accesses/day

The attribute of the Purchase entity would be:

- DateTime
- List of plants purchased: each element of the list would have the quantity, the price and the plant purchased.

The total cost of the purchase could be computed by summing each of the element in the list of plants. If frequently accessed, a redundant attribute Total paid could be introduced in the Purchase entity.

### 3.2.5 Operation 5

*Printing of the plants, ordered by the number of items sold (10 times a month)*

| Concept | E/R | #Accesses | Read/Write | Notes |
|---|---|---|---|---|
| Plant | E | 800 | R | Read the plants data |
| Plants purchased | R | 1’080’000 | R | Read each one of the records in the relationship to count the total number of sales |

- Cost (Op5) = 800 + 1080000 = 1’800’800 accesses
- Monthly cost (Op5) = 10 * cost = 10’800’800 accesses/month

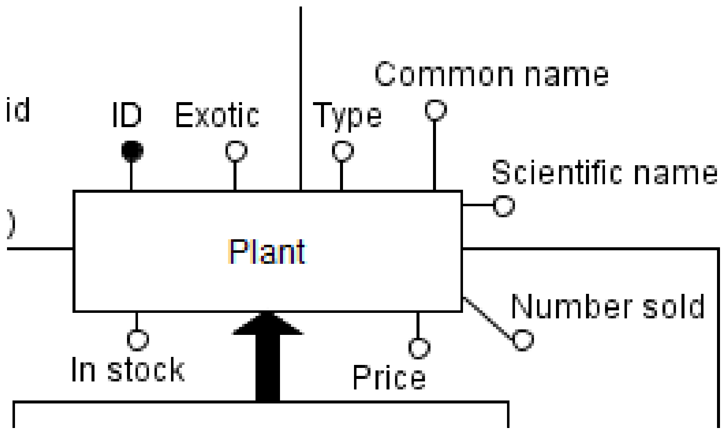


*Figure 6: Plant entity with the redundant attribute Number sold*

By introducing a redundant attribute *Number sold* on the Plant attribute we could drastically reduce the number of accesses in this operation.

In fact, we would only need in total 800 accesses to the Plant entity:

| Concept | E/R | #Accesses | Read/Write | Notes |
|---|---|---|---|---|
| Plant | E | 800 | R | Read the plants data (and the counter of sold items) |

- Cost (Op5) = 800 accesses
- Monthly cost (Op5) = 10 * cost = 8’000 accesses/month

## 3.3 Redundancies analysis

1. If we add the redundancy on Plant with the *Price* attribute both Operation 2 and Operation 3 become much faster: in Operation 2 we have 80’000 accesses/day (without redundancies) and 20’000 accesses/day (with redundancies); in Operation 3 we have 210’000 accesses/day (without

redundancies) and 10’000 accesses/day (with redundancies). The downside is the little loss of memory of 4 bytes (size of a float) * 800 (#plants) = 3,2 KiB, which is truly irrelevant. The other downside is the additional cost for those rare operations of updating the price of a plant, but benefits outweight the disadvantages.

2. The attribute *Paid* in the Purchase entity can be retrieved by accessing to the Plant price and making a product between the price and the quantity of the Purchase. This would require too many accesses and it’s better just to spend 4 bytes * 1.080.000 = about 4 MB to avoid this operation. Also, this value is never modified, so there’s no situation in which we need to keep this redundancy updated.
3. If we add the redundancy on Plant with the *Number sold* attribute we have great advantages on Operation 5 as we have seen: 10’800’800 accesses/month (without redundancies) vs. 8’000 (with redundancies). On the other hand, the updating of this value is not even problematic, since in Operation 2 we already write on the Plant entity when a purchase is made (to update the *In stock* attribute). So there are no disadvantages other than the spacial one: 4 bytes (standard size of int) * 800 (#plants) = 3,2 KiB. So the redundancy is kept.

## 3.4 Logical schema

The Object-Relational Logical schema at the end of the day would be as follows:

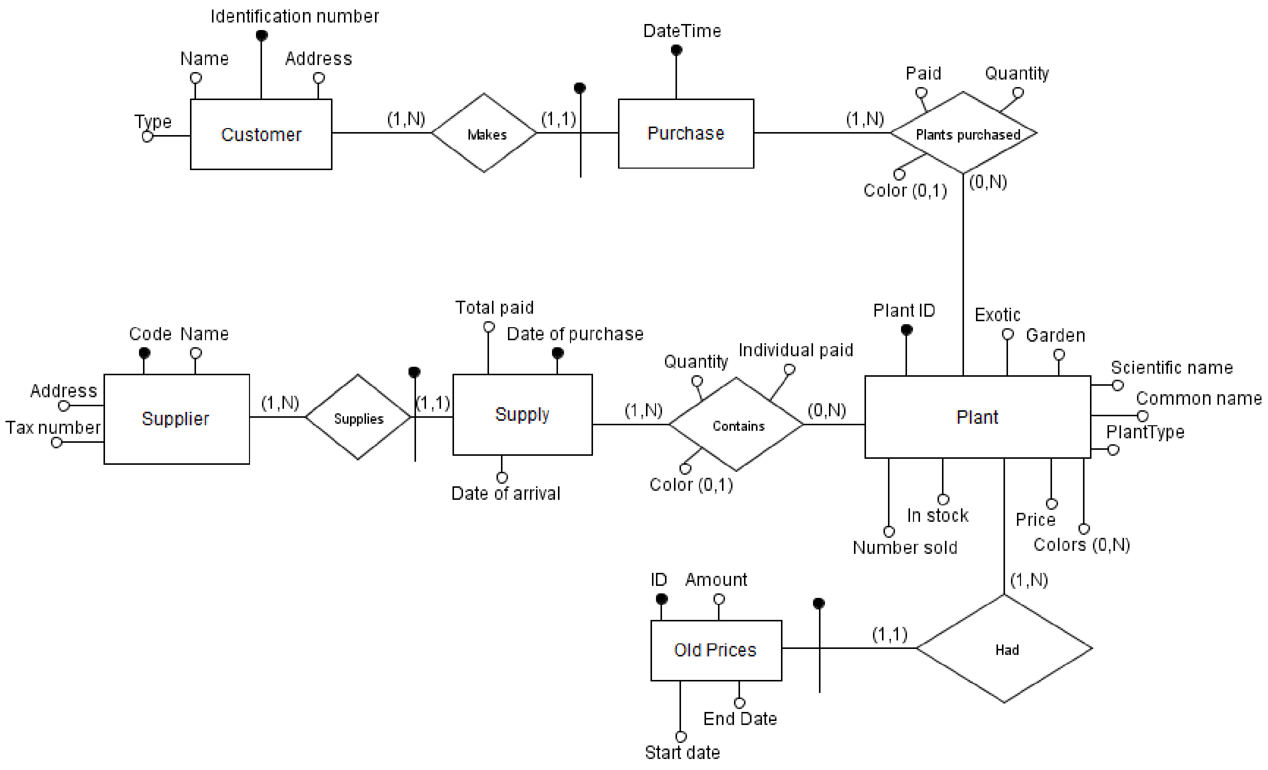


*Figure 7: Logical schema with the E-R model*

As it can be seen, there are a couple of modifications we have not talked about:

- the Customer *code* was dropped, as the *Identification number* will be sufficient to uniquely distinguish each customer;
- the *Plant ID* which is simply the renaming of the *ID* attribute on Plant;

- the *Color* attribute was added in the Contains and Plants purchased relations: in fact, it's necessary to specify the color for each plant purchased/supplied. This leads to the problem of not having more than one pair Purchase-Plant, as we represented this concept in the ER model above. This will be fixed with the introduction of OIDs.

We want the Plants Purchased concept to be represented as a *composition* entity that depends on Purchases: in fact the former has meaning only in relation with the latter. In turn, also the Purchases exist only if in relation with some Customer, so the same logic applies.

Furthermore, we also want to have the composition paradigm to a Supply and the Plants supplied.

As we can see, it is better to leave the E-R model in favour of an UML diagram, in which composition relations can be represented.

Finally, we can also see the Old prices entity as an entity which exists only if associated with some Plant.

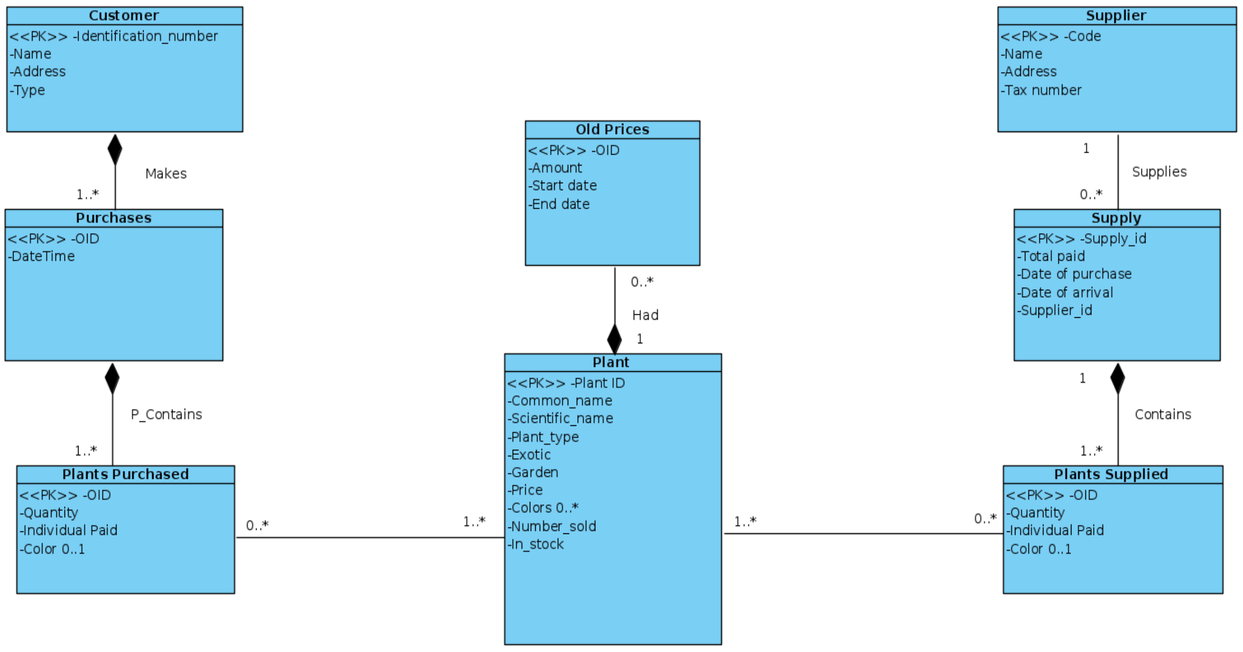


*Figure 8: Logical model with an UML diagram*

The logical schema has been translated in UML using the software Visual Paradigm.

## 4 Physical Design

### 4.1 Database Implementation

The database has been implemented in Oracle 10g.

To not clog this document, the types and tables definitions are reported as an attachment of this documentation, respectively in the ***create_tables.sql*** and ***create_types.sql*** files.

## 4.2 Operations implementation

Operation 1 and 2 were implemented as PL/SQL stored procedures in the ***operations.sql*** file, which can be found in attachment. In this file also Operations 3,4 and 5 are reported as queries, which are later called by the Java Servlet to retrieve the result sets from the database.

## 4.3 Queries optimization

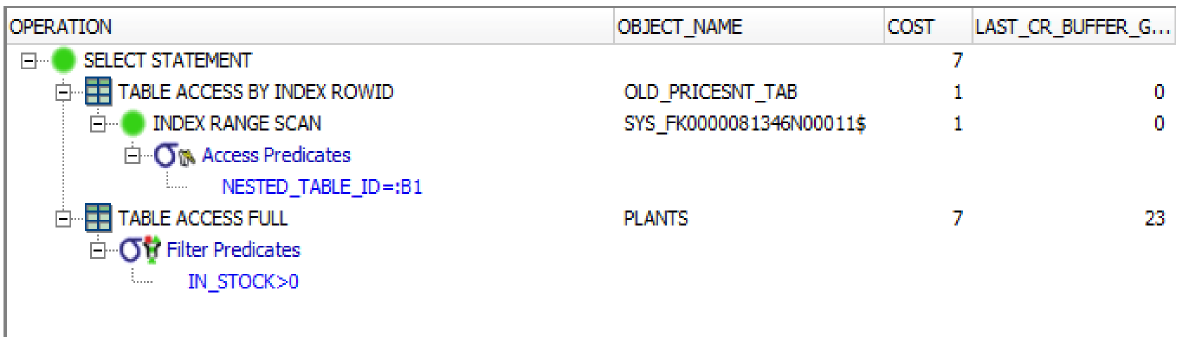
Looking at the query operations, we can try hands-on to insert some index in order to optimize the operations 3-4-5 (since operations 1 and 2 are insertions).

### 4.3.1 Operation 3

*Printing of the number of plants and properties of plants available for a given species (10000 times a day)*

For operation 3, which is the most frequent (10000 times a day), we have to try to squeeze out the most performance we can; the query looks like this:

```
 SELECT * FROM Plants WHERE in_stock>0;
```



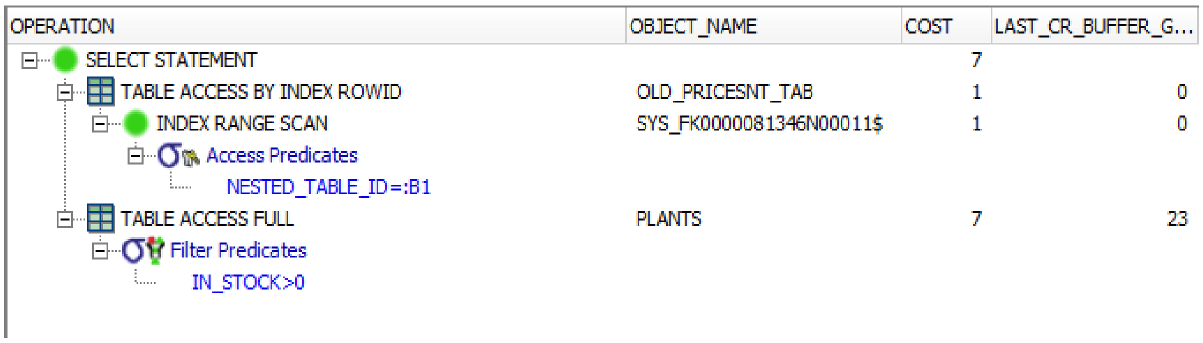
```
OPERATION                                  OBJECT_NAME                COST   LAST_CR_BUFFER_G...
SELECT STATEMENT                                                         7
  TABLE ACCESS BY INDEX ROWID              OLD_PRICESNT_TAB              1                  0
    INDEX RANGE SCAN                       SYS_FK0000081346N00011$       1                  0
      Access Predicates
        NESTED_TABLE_ID=:B1
  TABLE ACCESS FULL                        PLANTS                        7                 23
    Filter Predicates
      IN_STOCK>0
```

*Figure 9: Operation 3 autotrace - without index on in_stock*

This is the autotrace took from Oracle SQL Developer. As we can see we have a cost of 7, where the biggest load comes from the full scan of the table Plants. Let's try to create a test b-tree index on the attribute in_stock:

```
CREATE INDEX ix_plants_instock ON plants(in_stock);
```

and let's see how it performs:



```
OPERATION                                  OBJECT_NAME                COST   LAST_CR_BUFFER_G...
SELECT STATEMENT                                                         7
  TABLE ACCESS BY INDEX ROWID              OLD_PRICESNT_TAB              1                  0
    INDEX RANGE SCAN                       SYS_FK0000081346N00011$       1                  0
      Access Predicates
        NESTED_TABLE_ID=:B1
  TABLE ACCESS FULL                        PLANTS                        7                 23
    Filter Predicates
      IN_STOCK>0
```

*Figure 10: Operation 3 autotrace - with index on in_stock*

We can see that the cost is the same, so that's no use in having an index and we can delete it.

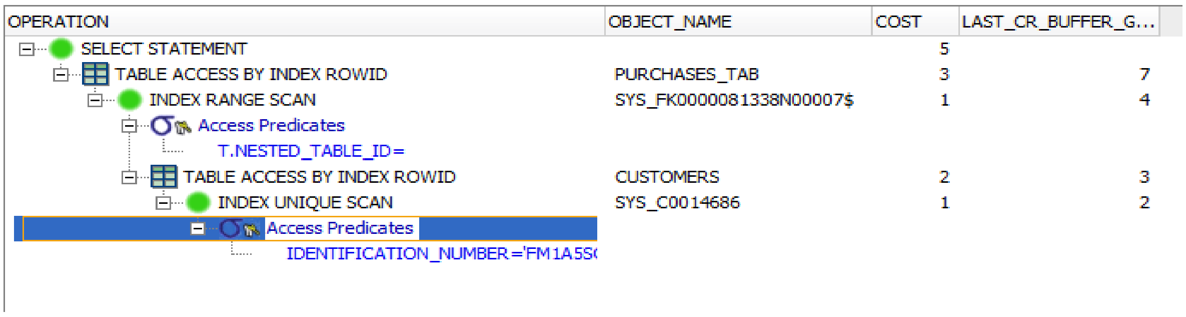
### 4.3.2 Operation 4

*Printing of the purchases of a given customer (150 times a day)*

Operation 4 involves a simple selection of a customer row with a given *identification number* (for example, *'FM1A5SQ0ESBAKS4').*

```
SELECT t.* FROM TABLE (SELECT Purchases FROM Customers
WHERE identification_number = 'FM1A5SQ0ESBAKS4') t
```

As we can see from the autotrace of Oracle SQL Developer, the cost of Operation 4 is only of 5 and thus we can state that is a very cheap operation to perform. Therefore, there's no need to try to optimize this query.



```
OPERATION                                   OBJECT_NAME                 COST   LAST_CR_BUFFER_G...
SELECT STATEMENT                                                           5
  TABLE ACCESS BY INDEX ROWID               PURCHASES_TAB                  3                     7
    INDEX RANGE SCAN                        SYS_FK0000081338N00007$        1                     4
      Access Predicates
        T.NESTED_TABLE_ID=
      TABLE ACCESS BY INDEX ROWID           CUSTOMERS                      2                     3
        INDEX UNIQUE SCAN                   SYS_C0014686                   1                     2
          Access Predicates
            IDENTIFICATION_NUMBER='FM1A5S
```

*Figure 11: Operation 4 autotrace - without indexes*

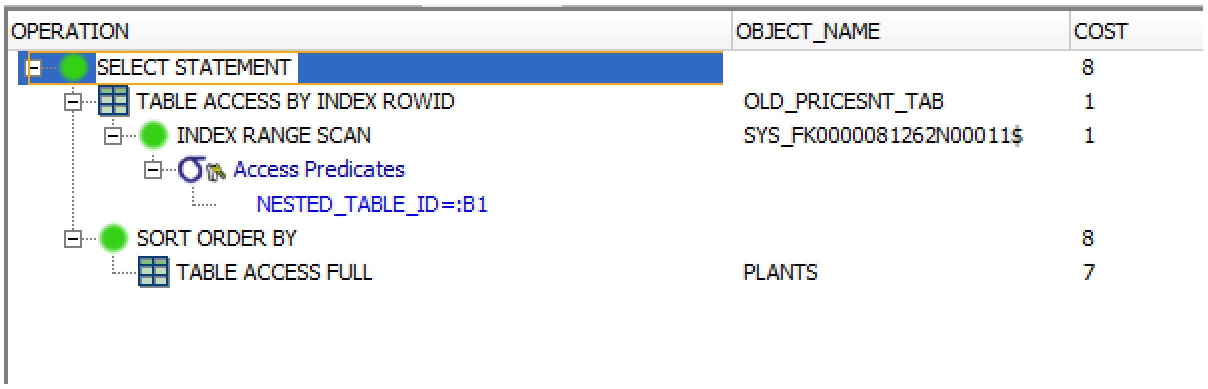
### 4.3.3 Operation 5

*Printing of the plants, ordered by the number of items sold (10 times a month)*

Regarding operation 5, we need to execute an ordering of the result set in order to answer to the query.

```
SELECT * FROM plants ORDER BY (number_sold) DESC;
```

Let's look at its cost:



| OPERATION | OBJECT_NAME | COST |
|---|---|---|
| SELECT STATEMENT | | 8 |
| TABLE ACCESS BY INDEX ROWID | OLD_PRICESNT_TAB | 1 |
| INDEX RANGE SCAN | SYS_FK0000081262N00011$ | 1 |
| Access Predicates | | |
| NESTED_TABLE_ID=:B1 | | |
| SORT ORDER BY | | 8 |
| TABLE ACCESS FULL | PLANTS | 7 |

*Figure 12: Operation 5 autotrace - without indexes*

As we could expect, the highest cost comes from the ordering operation. We can try and optimize this query by inserting a b-tree index on the attribute upon which the ordering has to be executed; but, even if we would manage to reduce the cost, operation 5 is so seldomly executed (only 10 times a month), that the gain in efficiency would be unnoticeable. Thus, we decide not to create an index, because that would only lead to higher costs of data manipulation operations.

## 5 Additional features

### 5.1 Populate Procedures

To populate the Database with plausible (kinda) data, a set of procedures was created:

- ***POPULATE_CUSTOMERS***: inserts into the Customers table a variable number (given in input) of records; a third of the data will be of *retailers* customers, the remainder will be of *private individuals*.
- ***POPULATE_PLANTS***: inserts into the Plants table a variable number of records (the number is given as an input of the procedure); half of the data will belong to the *"green"* type of

plants (i.e. without a color), and the other half to *"flowery"* plants, which have an array of three colors (between red, yellow, blue, violet, white, pink, orange and NULL) each. All plants come with 5 old prices.

- ***POPULATE_PURCHASES***: inserts into the Customer's Purchases nested table a number of record limited to a maximum value (given in input). Each of the purchase will have 3 plants purchased.
- ***POPULATE_SUPPLIERS***: populates the Suppliers table with a number of record given in input.
- ***POPULATE_SUPPLIES***: populates the Supplies table with a given number (given in input) of rows; each supply has a reference to an existing Supplier; each record has 2 plants purchased subrows.

## 5.2 Triggers

Some triggers have been implemented in the systems:

- ***supplies_auto_increment_id***: keeps updated an auto-increment value which is used as a primary key in the Supplies table; it also prevents inserts of supplies regarding plants which do not exist.
- ***increase_plants_in_stock***: increases the *in_stock* attribute of a Plant when a new supply is made;
- ***decrease_plants_in_stock****:* decreases the *in_stock* attribute of a Plant when a new purchase is made by a customer. It also prevents the insertion of a purchase involving a quantity which is greater than the in-stock availability of that plant.
- [currently doesn't work] *plants_insert_check_type:* prevents the inserting of a plant of type 'f' (flowery), but with no colors assigned, and the inserting of a plant of type 'g' (green) but with one or more colors assigned;

## 5.3 Other functionalities

Other than the extensions in the database structure, two more operations were introduced:

- Op 6: *Update a plant price*

From the website an operator can update the price of a plant by its ID. The current price will be replaced by the new one and will be inserted in the plants' old prices, with the appropriate dates.

This operation is performed by the ***UPDATE_PRICE*** procedure, which is attached to the documentation in the *operations.sql* file.

- Op 7: *Search of a plant by its name*

From the website an operator can search for a plant's data by making a research based on the plants common name. The matching results, which include the typed string anywhere in the common_name field, will be shown to the user.

The query to perform for this operation is the following:

```
SELECT * FROM plants WHERE common_name LIKE
('%partial_name%');
```

# 6 Deployment, Website and Servlet

## 6.1 System Deployment

The Oracle database is installed in a Windows 7 virtual machine, in the version J2EE 10g (10.1.3.4.0).

To create the database and to populate it, the file *SetDatabase.sql* was created: it is sufficient to run the entire script to have it all ready; it is worth noting that the populate procedures execution will take a while to be performed.

A script *EmptyDatabase.sql* has been created to delete all the tables, triggers and procedures from the database and leave it empty.

## 6.2 Website front-end

A website was created ad hoc to interact with the Oracle database. HTML, CSS and Javascript were used to implement the client-side, in particular JQuery and Bootstrap libraries, while for the server-side Java was used to create a servlet. The latter receives requests from the users and queries the database to return the desired data.
The website has been called “Plantsdom” and here we will report some screens of it.

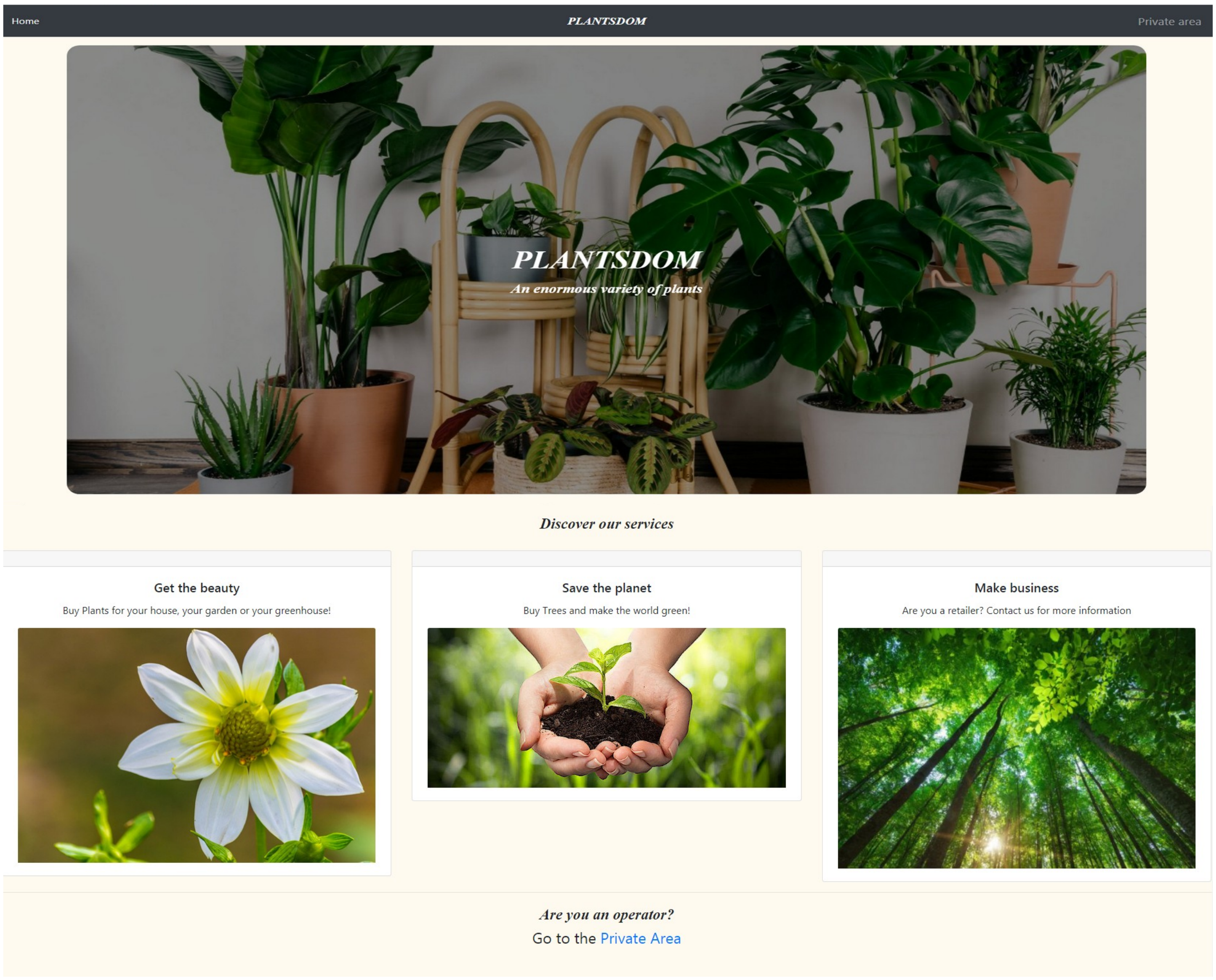
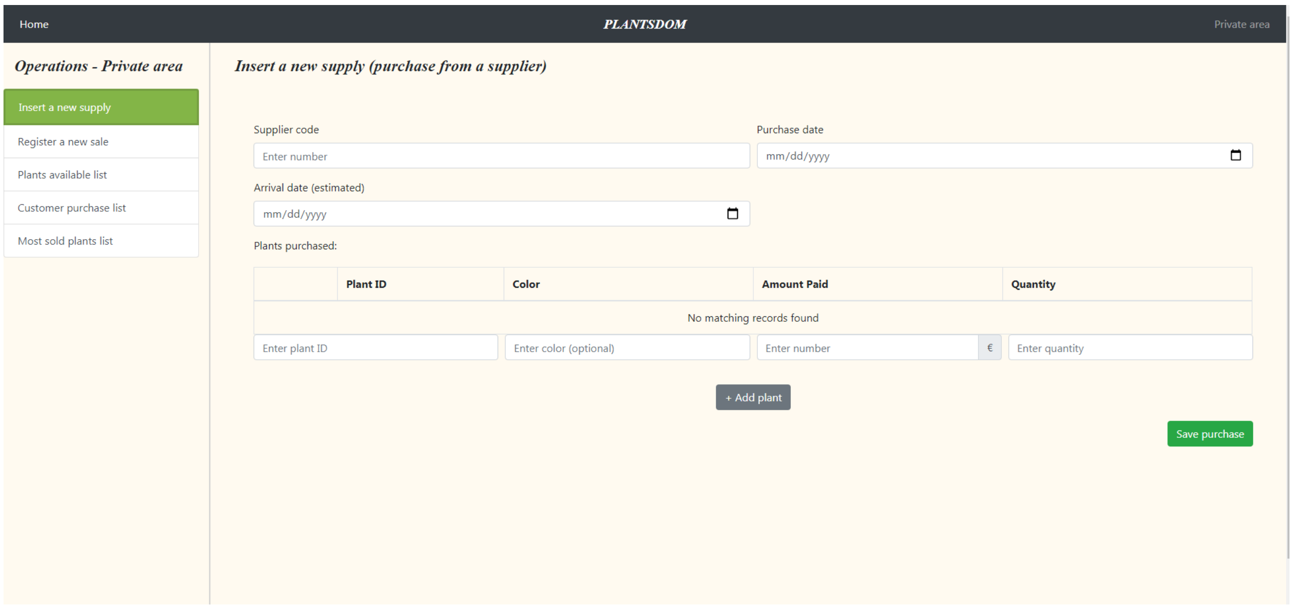

Home
PLANTSDOM
Private area
PLANTSDOM
An enormous variety of plants
Discover our services
Get the beauty
Buy Plants for your house, your garden or your greenhouse!
Save the planet
Buy Trees and make the world green!
Make business
Are you a retailer? Contact us for more information
Are you an operator?
Go to the Private Area

*Figure 13: Website Homepage*

We have the homepage, which presents the website, and the "Private area" (reachable from the link in the top-right corner or in the bottom of the page) in which an operator can interact with the Java Servlet. In the latter, the user can select between all the operations (from Operation 1 to 5), insert data and view results.

In particular Bootstrap Tables was used to implement tables for inserting multiple plants of a purchase: we can see them in Operations 1, 2 and 7.


Home PLANTSDOM Private area

*Operations - Private area*

- Insert a new supply
- Register a new sale
- Plants available list
- Customer purchase list
- Most sold plants list

*Insert a new supply (purchase from a supplier)*

Supplier code: Enter number

Purchase date: mm/dd/yyyy

Arrival date (estimated): mm/dd/yyyy

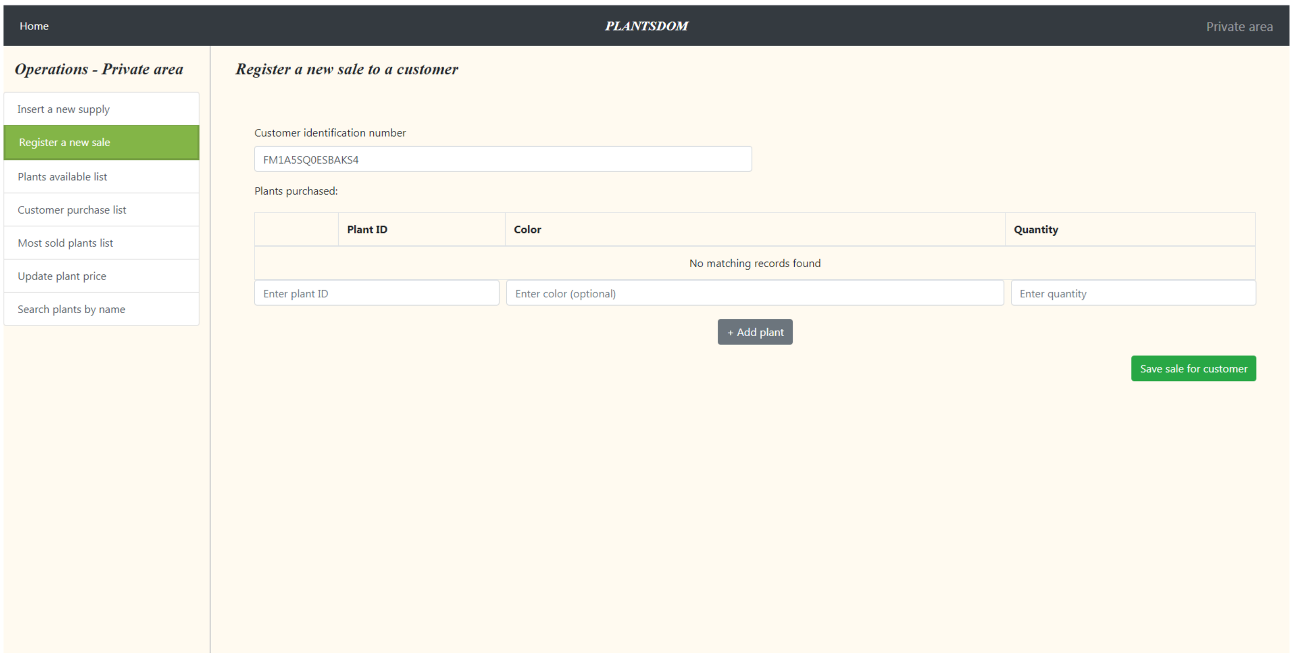
Plants purchased:

| | Plant ID | Color | Amount Paid | Quantity |
|---|---|---|---|---|
| No matching records found | | | | |
| | Enter plant ID | Enter color (optional) | Enter number € | Enter quantity |

+ Add plant

Save purchase

*Figure 14: Website "Private Area" – Operation 1*



*Register a new sale to a customer*

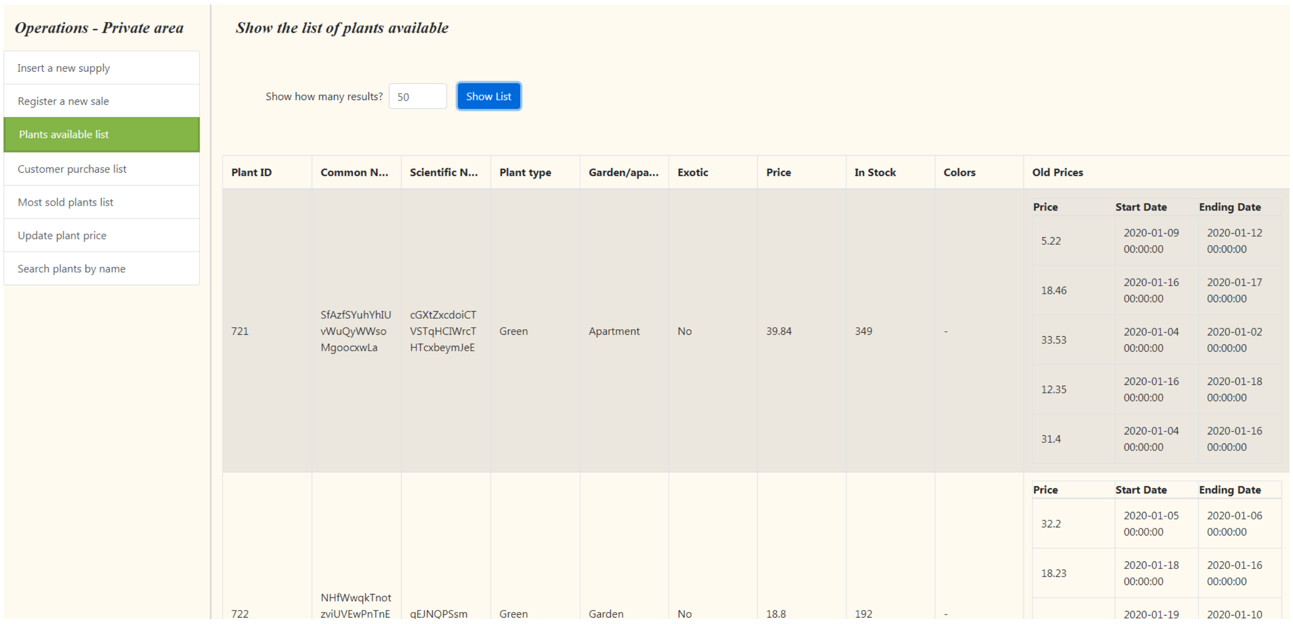
Customer identification number

FM1A5SQ0ESBAKS4

Plants purchased:

| | Plant ID | Color | Quantity |
|---|---|---|---|
| No matching records found | | | |
| | Enter plant ID | Enter color (optional) | Enter quantity |

*Figure 15: Website "Private Area" – Operation 2*

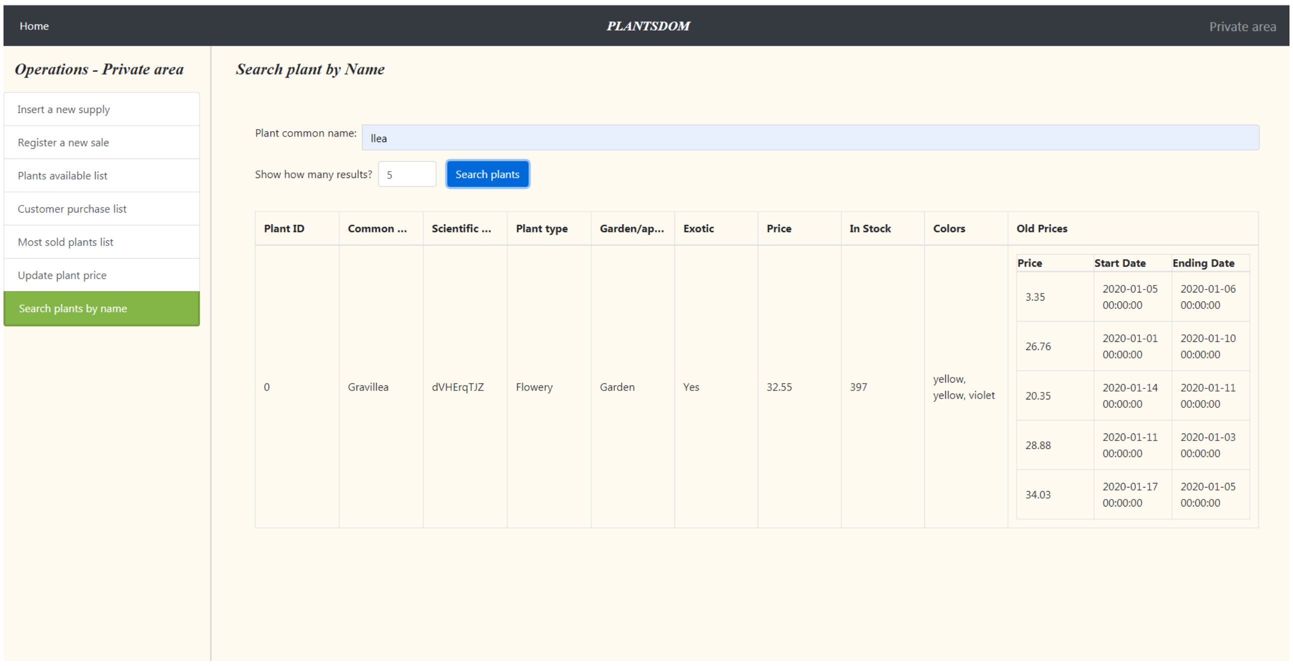


*Show the list of plants available*

Show how many results? 50

| Plant ID | Common N... | Scientific N... | Plant type | Garden/apa... | Exotic | Price | In Stock | Colors | Old Prices |
|---|---|---|---|---|---|---|---|---|---|
| 721 | SfAzfSYuhYhIU vWuQyWWso MgoocxwLa | cGXtZxcdoiCT VSTqHCIWrcT HTcxbeymJeE | Green | Apartment | No | 39.84 | 349 | - | Price / Start Date / Ending Date<br>5.22 / 2020-01-09 00:00:00 / 2020-01-12 00:00:00<br>18.46 / 2020-01-16 00:00:00 / 2020-01-17 00:00:00<br>33.53 / 2020-01-04 00:00:00 / 2020-01-02 00:00:00<br>12.35 / 2020-01-16 00:00:00 / 2020-01-18 00:00:00<br>31.4 / 2020-01-04 00:00:00 / 2020-01-16 00:00:00 |
| 722 | NHfWwqkTnot zviUVEwPnTnE | qEJNQPSsm | Green | Garden | No | 18.8 | 192 | - | Price / Start Date / Ending Date<br>32.2 / 2020-01-05 00:00:00 / 2020-01-06 00:00:00<br>18.23 / 2020-01-18 00:00:00 / 2020-01-16 00:00:00<br>2020-01-19 / 2020-01-10 |

*Figure 16: Website "Private Area" - Operation 3, results of a query*



| Plant ID | Common ... | Scientific ... | Plant type | Garden/ap... | Exotic | Price | In Stock | Colors | Old Prices | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Price | Start Date | Ending Date |
| 0 | Gravillea | dVHErqTJZ | Flowery | Garden | Yes | 32.55 | 397 | yellow, yellow, violet | 3.35 | 2020-01-05 00:00:00 | 2020-01-06 00:00:00 |
| | | | | | | | | | 26.76 | 2020-01-01 00:00:00 | 2020-01-10 00:00:00 |
| | | | | | | | | | 20.35 | 2020-01-14 00:00:00 | 2020-01-11 00:00:00 |
| | | | | | | | | | 28.88 | 2020-01-11 00:00:00 | 2020-01-03 00:00:00 |
| | | | | | | | | | 34.03 | 2020-01-17 00:00:00 | 2020-01-05 00:00:00 |

*Figure 17: Website "Private Area" - Operation 7, results of the search of "llea"*

## 6.3 Website back-end: Servlet

As we've told the back-end is a Java Servlet, which is the controller of the application: it intercepts the users' requests, makes requests to the database and returns the data, which the views will output to the users.

This is the ***Servlet**.java* class, which based on the request method and operation called, will invoke the proper operation: these operations are contained in the ***DBOperations**.java* file.

For POST requests the *doPost()* method selects the right operation between:

- Operation 1: calls the *insertSupply()* method (in the DBOperation class), giving as parameters the supplier_code, the purchase_date, the arrival_date and the plants_purchased (the ids, the quantities, the amounts paid and the colors);

- Operation 2: calls the *insertPurchase()* method, which takes in input the customer_code, and the plants_purchased information (the ids, the quantities, and the colors);
- Operation 6: calls the *updatePrice()* method, which takes the plant_id to update and the new_price to set.

For GET requests the *doGet()* method selects an operation between:

- Operation 3: calls the *getAvailablePlants()* method;
- Operation 4: calls the *getCustomerPurchases()* method, taking in input the customer_code;
- Operation 5: calls the *getMostSoldPlants()* method;
- Operation 7: calls the *getPlantByName()* method, taking in input the common_name string to look for.

The operations functions contained in the *DBOperations* class have dependencies on another class, called ***DatabaseOracle*** (contained in the *DatabaseOracle.java* file): the latter contains all the connectivity, conversions and utility functions to correctly form the requests to the Oracle database. This class mainly uses the *oracle.sql* package.

To avoid SQL injections *prepared statements* were adoperated to securely get the parameters from the users requests before using them into the queries.

Results from the DB were translated into JSON Objects using the *org.json* package: the *Operations.jsp* view, therefore, takes the JSON response and builds the tables with the results to show them to the user.
The user can query up to 500 results and can choose to visualize only a limited part, but this only affects the client side: the server will always return all the data.

# 7 Conclusions

This project has the intent of showing the capability of designing and implementing an Oracle database system. Furthermore, also a Java backend system was implemented to query the database from the final user's point of view. I learnt new technologies and discovered that there were many many more that I could have chosen for the implementation of this project, some of them which are surely more powerful than the ones used. The downside was the learning time that many of them required, even to be understood, so I hope I made some good technical choices. I also tried to get the best usability level as possible with the use of some front-end web technologies, but in a limited period of time it has not been completely possible.
Some future work could surely include the creation of the Model classes with Jpublisher, to have a better flexibility and robustness in the creation of other functionalities; another crucial point is the

improvement of the user interaction with the database, like giving more feedback to the user (since now the feedback given by the system is really poor) and some auto-completement features for the operations that require user inputs.