Database Naming Conventions, Documents and Scripts

This document serves as the aggregation of all the small reports related to the database that are scattered along this project. This project will have into account the following topics:

1. Database Naming Conventions
2. Docs
3. Scripts
4. Database Physical Structure
5. Database Logic

Database Naming Conventions

- Check isolated file

Rules

- **Table names** must not include protected keywords for Oracle SQL (ex. User, Dual, Start, etc.)
- **Table names** must follow the proper Camel case with the first letter of name capitalized (ex. Sensor, WaterSensor, Fertilizer, etc. What not to follow: sensor, waterSensor, PoTassicfertilizer,etc.)
- **Table Attributes** must follow proper camel case with first character not capitalized (ex. amount, numberOfSensors, cultivationType, etc.),
- **Table Attributes Constraints**, if inside table creation, may (or may not) have a dedicated name (ex. check (regex_like(code,"\d{8}\w{3}"))).
- **Table Attributes Constraints**, if outside table, as an alter table, it must have a name as the following form CC[TABLE_NAME]_[DESCRIPTION], where CC is the type of constraint (see constraint table in use), [TABLE_NAME] is the name of the table and [DESCRIPTION] is a description to identify what the constraint aims to achieve
- **Primary Key[s]** must be as simple as possible, using camel case, with first character uncapitalized, and, in preference, one word long (ex. code, name, id. What not to follow: idOfTeam, teamId, teamID, ID, Id, iD, idTeam, etc.)
- **Foreign Key[s]** must follow the same rules as **Primary Key[s]** and the name must be related to the relation that results in the **Foreign Key[s]** (ex. Assume entity *Music (M)* has multiple *CD (C)*, *M* "1" -> "1..N" *C*, then the **Foreign Key[s]** name in entity *Music* must be **cd**, and not cdCode, codeOfCd, fkCD, cdFK, etc.)
- **Functions** must follow the convention fncUS[NNN][Designation], where NNN is the number of the US stated in the requirements and the description the function main goal, in Camel Case
- **Procedures** must follow the convention prcUS[NNN][Designation], where NNN is the number of the US stated in the requirements and the description the procedure main goal, in Camel Case

- **Triggers** must follow the convention trg[Designation], where designation is the triggers main goal, in Camel Case

Table Resume

Database Entity	Rule
Table Name	Must not include protected keywords for Oracle SQL (ex. User, Dual, Start, etc.)
	Must follow the proper Camel case with the first letter of name capitalized (ex. Sensor, WaterSensor, Fertilizer, etc. What not to follow: sensor, waterSensor, PoTassicfertilizer,etc.)
Table Attribute	Must follow proper camel case with first character not capitalized (ex. amount, numberOfSensors, cultivationType, etc.)
Table Attributes Constraint	If inside table creation, may (or may not) have a dedicated name (ex. check (regex_like(code,"\d{8}\w{3}"))).
	If outside table, as an alter table, it must have a name as the following form CC[TABLE_NAME]_[DESCRIPTION], where CC is the type of constraint (see constraint table in use), [TABLE_NAME] is the name of the table and [DESCRIPTION] is a description to identify what the constraint aims to achieve
Primary Key[s]	Must be as simple as possible, using camel case, with first character uncapitalized, and, in preference, one word long (ex. code, name, id. What not to follow: idOfTeam, teamId, teamID, ID, Id, iD, idTeam, etc.)
Foreign Key[s]	Must follow the same rules as **Primary Key[s]** and the name must be related to the relation that results in the **Foreign Key[s]** (ex. Assume entity *Music (M)* has multiple *CD (C)*, *M* "1" -> "1..N" *C*, then the **Foreign Key[s]** name in entity *Music* must be **cd**, and not cdCode, codeOfCd, fkCD, cdFK, etc.)
Function	Functions must follow the convention fncUS[NNN][Designation], where NNN is the number of the US stated in the requirements and the description the function main goal, in Camel Case
Procedure	Procedures must follow the convention prcUS[NNN][Designation], where NNN is the number of the US stated in the requirements and the description the procedure main goal, in Camel Case
Trigger	Triggers must follow the convention trg[Designation], where designation is the triggers main goal, in Camel Case

Constraints Table

Constraint Name	Constraint Key
Foreign Key	**FK**
Primary Key	**PK**
Not Null	**NN**
Unique	**UQ**
Default	**DF**
Index	**ID**

Note: The SQL syntax is case insensitive, so for everything that is related to Camel case, it only applies to diagrams and documentation.

Docs

- Check isolated file

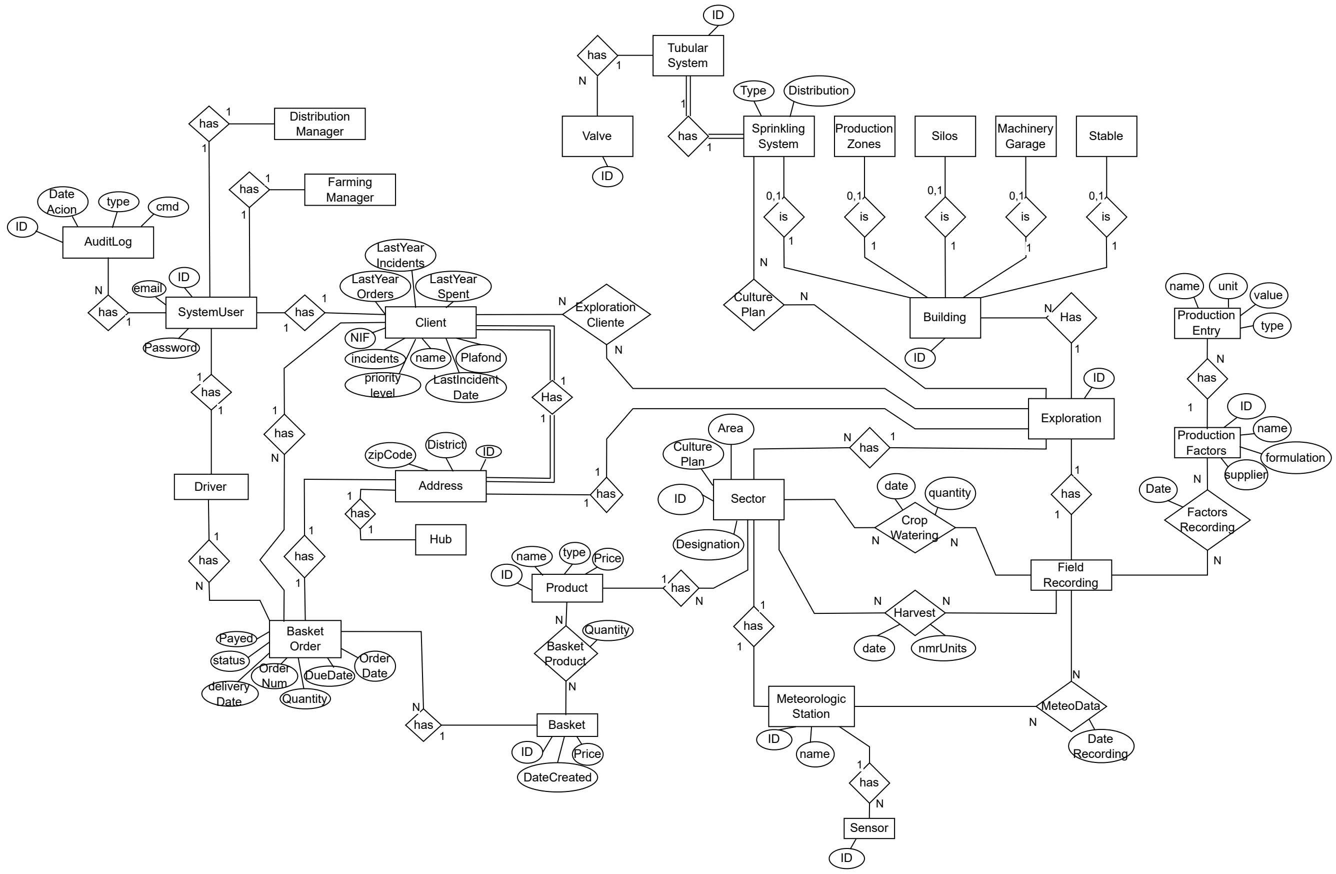
Dictionary

Entry Name	Definition
Address	Represents an address of an individual/user and/or building/hub/location
AuditLog	Table responsible for recording any table alterations by a given system user
Basket	The aggregate of products of a given basket
BasketOrder	The order of a given basket by a client
BasketProduct	The product that is included on a given basket, and their quantities
Building	Place of an exploration that has constructions
Client	System user that can order basket
CropWatering	The registration of the watering of the crops of a given sector
CulturePlan	The plan used by a given sprinkling system to coordinate how to water the crops of a given exploration
DistributionManager	System user responsible for the operations of distribution of the products and baskets

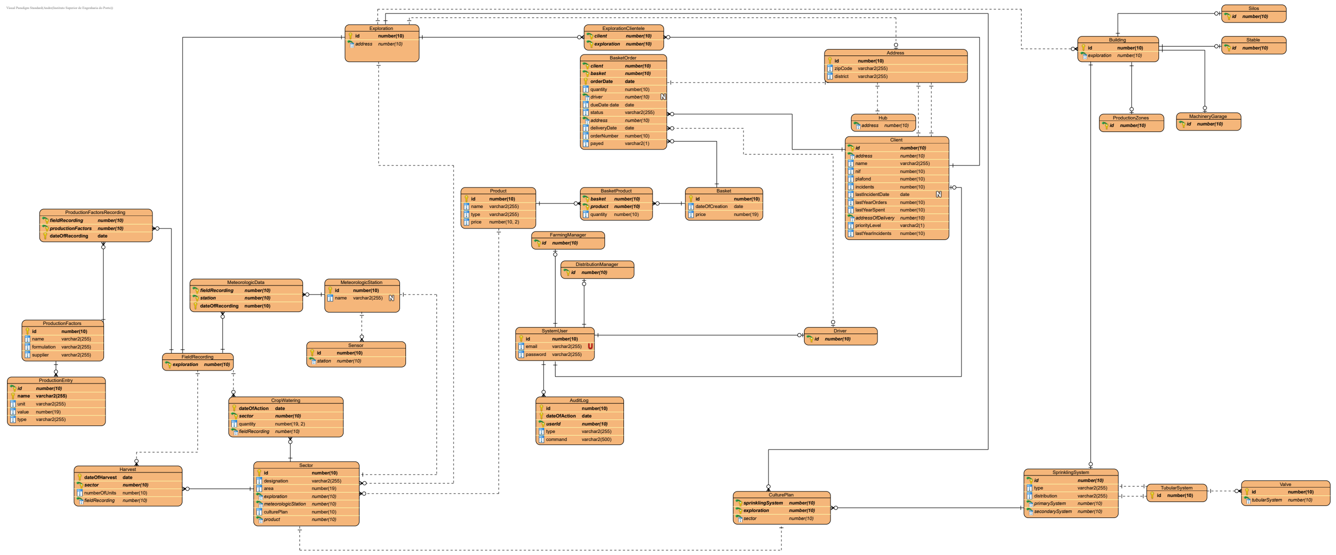
Entry Name	Definition
Driver	System user responsible for the transport of a given order
Exploration	The representation of the area responsible for producing products
ExplorationClientele	The regular clients of a given exploration
FarmingManager	System user responsible for the production operations of a given exploration
FieldRecording	Registry responsible for documenting all the actions of a given exploration
Harvest	The registration of all the important information of a given yield of a sector
Hub	Center for trade and commerce
MachineryGarage	Type of building where machines are stored
MeteorologicData	The data that a given station collects
MeteorologicStation	The aggregate of scientific and technological components that are responsible for the conditions of the soil, atmosphere and/or water of a given sector
Product	The item that is being produced on a given sector
ProductionEntry	The components/composition/ingredients of a given production factor
ProductionFactors	A item that helps the exploration to grow crops
ProductionsFactorsRecording	The registration production factors used in the exploration
ProductionZones	Buildings where products are produced
Sector	Plot of land where crops are grown and cultivated
Sensor	The equipment responsible for collecting the information of the soil, atmosphere and water of a sector
Silos	Type of a building that stores products
SprinklingSystem	The infrastructure system responsible for watering the crops of a given exploration
Stable	Type of building where live animals are stored and live
SystemUser	A person that is registered in this application and executes a given function on said application
TubularSystem	The aggregate of components that are part of a sprinkling system

Entry Name	Definition
Valve	Component of a tubular system that controls the flow of water

Conceptual Model



Logical Model



Scripts

- Check isolated file (Database Structure)
- Check isolated file (Database Logic)

Database Physical Structure

Technology

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data. Popular examples are:

- Oracle XE
- My SQL
- SQL Server
- PostgreSQL

Nevertheless, there are other types of databases that deviate from such specification, non-relational (or no SQL) databases. A NoSQL, or non relational database, allows unstructured and semi structured (making use of schemas) data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into the database must be composed). NoSQL databases grew popular as web applications became more common and more complex. Popular examples are:

- Mongo DB
- Apache Cassandra
- Neo4J
- Redis

Reference

Relational databases work with structured data. They support ACID transactional consistency and provide a flexible way to structure data that is not possible with other database technologies. Key features of relational databases include the ability to make two tables look like one, join multiple tables together on key fields, create complex indexes that perform well and are easy to manage, and maintain data integrity for maximum data accuracy.

The relational database is a system of storing and retrieving data in which the content of the data is stored in tables, rows, columns, or fields. When you have multiple pieces of information that need to be related to one another then it is important to store them in this type of format; otherwise, you would just end up with a bunch of unrelated facts and figures without any ties between them.

There are many benefits associated with using a relational database for managing your data needs. For instance, if you want to view all the contacts in your phone book (or other types) then all you would need to do is enter one query into the search bar and instantly see every contact listed there. This saves time from having to manually go through.

The relational database benefits are discussed briefly.

1. **Simplicity of Model**

In contrast to other types of database models, the relational database model is much simpler. It does not require any complex queries because it has no query processing or structuring so simple SQL queries are enough to handle the data.

2. **Ease of Use**

Users can easily access/retrieve their required information within seconds without indulging in the complexity of the database. Structured Query Language (SQL) is used to execute complex queries.

3. **Accuracy**

A key feature of relational databases is that they're strictly defined and well-organized, so data doesn't get duplicated. Relational databases have accuracy because of their structure with no data duplication.

4. **Data Integrity**

RDBMS databases are also widely used for data integrity as they provide consistency across all tables. The data integrity ensures the features like accuracy and ease of use.

5. **Normalization** As data becomes more and more complex, the need for efficient ways of storing it increases. Normalization is a method that breaks down information into manageable chunks to reduce storage size. Data can be broken up into different levels with any level requiring preparation before moving onto another level of normalizing your data.

Database normalization also ensures that a relational database has no variety or variance in its structure and can be manipulated accurately. This ensures that integrity is maintained when using data from this database for your business decisions.

6. **Collaboration**

Multiple users can access the database to retrieve information at the same time and even if data is being updated.

7. **Security**

Data is secure as Relational Database Management System allows only authorized users to directly access the data. No unauthorized user can access the information.

Although there are more benefits of using relational databases, it has some limitations also. Let’s see the limitations or disadvantages of using the relational database.

1. **Maintenance Problem**

The maintenance of the relational database becomes difficult over time due to the increase in the data. Developers and programmers have to spend a lot of time maintaining the database.

2. **Cost**

The relational database system is costly to set up and maintain. The initial cost of the software alone can be quite pricey for smaller businesses, but it gets worse when you factor in hiring a professional technician who must also have expertise with that specific kind of program.

3. **Physical Storage**

A relational database is comprised of rows and columns, which requires a lot of physical memory because each operation performed depends on separate storage. The requirements of physical memory may increase along with the increase of data.

4. **Lack of Scalability**

While using the relational database over multiple servers, its structure changes and becomes difficult to handle, especially when the quantity of the data is large. Due to this, the data is not scalable on different physical storage servers. Ultimately, its performance is affected i.e. lack of availability of data and load time etc. As the database becomes larger or more distributed with a greater number of servers, this will have negative effects like latency and availability issues affecting overall performance.

5. **Complexity in Structure**

Relational databases can only store data in tabular form which makes it difficult to represent complex relationships between objects. This is an issue because many applications require more than one table to store all the necessary data required by their application logic.

6. **Decrease in performance over time**

The relational database can become slower, not just because of its reliance on multiple tables. When there is a large number of tables and data in the system, it causes an increase in complexity. It can lead to slow response times over queries or even complete failure for them depending on how many people are logged into the server at a given time.

Reference

For this project a **Relational Database (SQL Database)** was chosen due to having more upsides than downsides and for the downsides.

Files to Include

- CREATE TABLES

- INITIAL BOOT
- DELETE DATABASE

The database that will be used in this project will be Oracle 18c

For clarification on the naming conventions used on this database see this document

Table Creation and Alters

```
-- TABLES --
CREATE TABLE Address
(
    id       number(10) GENERATED BY DEFAULT AS IDENTITY,
    zipcode  VARCHAR2(255),
    district VARCHAR2(255) DEFAULT 'PORTO',
    PRIMARY KEY (id)
);
CREATE TABLE AuditLog
(
    id           number(10) GENERATED BY DEFAULT AS IDENTITY,
    dateOfAction date DEFAULT SYSDATE,
    userId       number(10)    NOT NULL,
    type         varchar2(255) NOT NULL,
    command      varchar2(500) NOT NULL,
    PRIMARY KEY (id,
                 dateOfAction,
                 userId)
);
CREATE TABLE Basket
(
    id             number(10) GENERATED BY DEFAULT AS IDENTITY,
    dateOfCreation date DEFAULT SYSDATE,
    price          number(15, 2) NOT NULL CHECK ( price > 0 ),
    PRIMARY KEY (id)
);
CREATE TABLE BasketOrder
(
    client       number(10) NOT NULL,
    basket       number(10) NOT NULL,
    quantity     number(10) NOT NULL CHECK ( quantity > 0 ),
    driver       number(10),
    orderDate    date          DEFAULT SYSDATE,
    dueDate      date          DEFAULT SYSDATE + 10,
    deliveryDate date          DEFAULT SYSDATE + 30,
    status       VARCHAR2(255) DEFAULT 'REGISTERED',
    address      number(10) NOT NULL,
    orderNumber  number(10) GENERATED ALWAYS AS IDENTITY,
    payed        VARCHAR2(1)   DEFAULT 'N',
    PRIMARY KEY (client,
                 basket, orderDate)
);
CREATE TABLE BasketProduct
(
    basket   number(10) NOT NULL,
    product  number(10) NOT NULL,
    quantity number(10) DEFAULT 1 CHECK ( quantity > 0 ),
    PRIMARY KEY (basket,
    PRIMARY KEY (basket,
                 product)
);
CREATE TABLE Building
(
    id          number(10) GENERATED BY DEFAULT AS IDENTITY,
    exploration number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE Client
(
    id                number(10)                   NOT NULL,
    address           number(10)                   NOT NULL,
    name              varchar2(255)                NOT NULL,
    nif               number(9)                    NOT NULL CHECK ( REGEXP_LIKE(nif, '^[1-4]\d{8}')
    plafond           number(10)    DEFAULT 100000 NOT NULL CHECK ( plafond >= 0 ),
    incidents         number(10)    DEFAULT 0      NOT NULL CHECK ( incidents >= 0 ),
    lastIncidentDate  date          DEFAULT SYSDATE,
    lastYearOrders    number(10)    DEFAULT 0      NOT NULL CHECK ( lastYearOrders >= 0 ),
    lastYearSpent     number(20, 2) DEFAULT 0      NOT NULL CHECK ( lastYearSpent >= 0 ),
    addressOfDelivery number(10)                   NOT NULL,
    priorityLevel     varchar2(1)   DEFAULT 'B'    NOT NULL CHECK ( REGEXP_LIKE(priorityLevel, '[AB
    lastYearIncidents number(10)    DEFAULT 0      NOT NULL CHECK ( lastYearIncidents >= 0 ),
    PRIMARY KEY (id)
);
CREATE TABLE CropWatering
(
    dateOfAction   date                    NOT NULL,
    sector         number(10)              NOT NULL,
    quantity       number(19, 2) DEFAULT 0 NOT NULL CHECK ( quantity >= 0 ),
    fieldRecording number(10)              NOT NULL,
    PRIMARY KEY (dateOfAction,
                 sector)
);
CREATE TABLE CulturePlan
(
    sprinklingSystem number(10) NOT NULL,
    exploration      number(10) NOT NULL,
    sector           number(10) NOT NULL,
    PRIMARY KEY (sprinklingSystem,
                 exploration)
);
CREATE TABLE DistributionManager
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE Driver
(
    id number(10) NOT NULL,

    PRIMARY KEY (id)
);
CREATE TABLE Exploration
(
    id      number(10) GENERATED BY DEFAULT AS IDENTITY,
    address number(10),
    PRIMARY KEY (id)
);
CREATE TABLE ExplorationClientele
(
    client      number(10) NOT NULL,
    exploration number(10) NOT NULL,
    PRIMARY KEY (client,
                 exploration)
);
CREATE TABLE FarmingManager
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE FieldRecording
(
    exploration number(10) NOT NULL,
    PRIMARY KEY (exploration)
);
CREATE TABLE ProductionFactorsRecording
(
    fieldRecording    number(10) NOT NULL,
    productionFactors number(10) NOT NULL,
    dateOfRecording   date DEFAULT SYSDATE,
    PRIMARY KEY (fieldRecording,
                 productionFactors,
                 dateOfRecording)
);
CREATE TABLE Harvest
(
    dateOfHarvest  date       NOT NULL,
    sector         number(10) NOT NULL,
    numberOfUnits  number(10) NOT NULL,
    fieldRecording number(10) NOT NULL,
    PRIMARY KEY (dateOfHarvest,
                 sector)
);
CREATE TABLE Hub
(
    address number(10) NOT NULL
);
CREATE TABLE MachineryGarage
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE MeteorologicData
(
    fieldRecording  number(10) NOT NULL,
    station         number(10) NOT NULL,
    dateOfRecording number(10) NOT NULL,
    PRIMARY KEY (fieldRecording,
                 station,
                 dateOfRecording)

);
CREATE TABLE MeteorologicStation
(
    id   number(10) GENERATED AS IDENTITY,
    name VARCHAR2(255),
    PRIMARY KEY (id)
);
CREATE TABLE Product
(
    name  varchar2(255)           NOT NULL,
    type  varchar2(255)           NOT NULL,
    id    number(10) GENERATED BY DEFAULT AS IDENTITY,
    price number(10, 2) DEFAULT 1 NOT NULL CHECK ( price > 0 ),
    PRIMARY KEY (id)
);
CREATE TABLE ProductionEntry
(
    id    number(10)    NOT NULL,
    value number(10)    NOT NULL CHECK ( value >= 0 ),
    unit  varchar2(255) NOT NULL,
    type  varchar2(255) NOT NULL,
    name  varchar2(255) NOT NULL,
    PRIMARY KEY (id, name)
);
CREATE TABLE ProductionFactors
(
    id          number(10) GENERATED BY DEFAULT AS IDENTITY,
    name        varchar2(255) NOT NULL,
    formulation varchar2(255) NOT NULL,
    supplier    varchar2(255) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE ProductionZones
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE Sector
(
    id                  number(10) GENERATED BY DEFAULT AS IDENTITY,
    designation         varchar2(255) NOT NULL,
    area                number(19)    NOT NULL,
    exploration         number(10)    NOT NULL,
    meteorologicStation number(10)    NOT NULL,
    culturePlan         number(10)    NOT NULL,
    product             number(10)    NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE Sensor
(
    id      number(10) GENERATED BY DEFAULT AS IDENTITY,
    station number(10) NOT NULL,
    CONSTRAINT id
        PRIMARY KEY (id)
);
CREATE TABLE Silos
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE SprinklingSystem
(
    id              number(10)    NOT NULL,
    type            varchar2(255) NOT NULL,
    distribution    varchar2(255) NOT NULL,
    primarySystem   number(10)    NOT NULL,
    secondarySystem number(10)    NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE Stable
(
(
    id number(10) NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE SystemUser
(
    id       number(10) GENERATED BY DEFAULT AS IDENTITY,
    email    varchar2(255) NOT NULL UNIQUE,
    password varchar2(255) DEFAULT 'Qw&rty12345678',
    PRIMARY KEY (id)
);
CREATE TABLE TubularSystem
(
    id number(10),
    PRIMARY KEY (id)
);
CREATE TABLE Valve
(
    id            number(10) GENERATED BY DEFAULT AS IDENTITY,
    tubularSystem number(10) NOT NULL,
    PRIMARY KEY (id)
);

-- Alter --

ALTER TABLE Driver
    ADD CONSTRAINT FKDriverUserId FOREIGN KEY (id) REFERENCES SystemUser (id);
ALTER TABLE FarmingManager
    ADD CONSTRAINT FKFarmingManagerUserId FOREIGN KEY (id) REFERENCES SystemUser (id);
ALTER TABLE Client
    ADD CONSTRAINT FKClientUserId FOREIGN KEY (id) REFERENCES SystemUser (id);
ALTER TABLE Client
    ADD CONSTRAINT FKClientUserId FOREIGN KEY (addressOfDelivery) REFERENCES Address (id);
ALTER TABLE DistributionManager
    ADD CONSTRAINT FKDistributionManagerUserId FOREIGN KEY (id) REFERENCES SystemUser (id);
ALTER TABLE Stable
    ADD CONSTRAINT FKStableBuildingId FOREIGN KEY (id) REFERENCES Building (id);
ALTER TABLE Silos
    ADD CONSTRAINT FKSilosBuildingId FOREIGN KEY (id) REFERENCES Building (id);
ALTER TABLE MachineryGarage
    ADD CONSTRAINT FKMachineryGarageBuildingId FOREIGN KEY (id) REFERENCES Building (id);
ALTER TABLE ProductionZones
    ADD CONSTRAINT FKProductionZonesBuildingId FOREIGN KEY (id) REFERENCES Building (id);
ALTER TABLE SprinklingSystem
    ADD CONSTRAINT FKSprinklingSystemBuildingId FOREIGN KEY (id) REFERENCES Building (id);
ALTER TABLE Sector
    ADD CONSTRAINT FKSectorExplorationId FOREIGN KEY (exploration) REFERENCES Exploration (id);
ALTER TABLE Building
    ADD CONSTRAINT FKBuildingExplorationId FOREIGN KEY (exploration) REFERENCES Exploration (id);
ALTER TABLE ProductionEntry

    ADD CONSTRAINT FKProductionEntryProductionFactorsId FOREIGN KEY (id) REFERENCES ProductionFacto
ALTER TABLE SprinklingSystem
    ADD CONSTRAINT FKSprinklingSystemTubularSystemPrimary FOREIGN KEY (primarySystem) REFERENCES Tu
ALTER TABLE SprinklingSystem
    ADD CONSTRAINT FKSprinklingSystemTubularSystemSecondary FOREIGN KEY (secondarySystem) REFERENCE
ALTER TABLE Sensor
    ADD CONSTRAINT FKSensorMeteorologicStationId FOREIGN KEY (station) REFERENCES MeteorologicStati
ALTER TABLE Sector
    ADD CONSTRAINT FKSectorMeteorologicStationId FOREIGN KEY (meteorologicStation) REFERENCES Meteo
ALTER TABLE ExplorationClientele
```

```
    ADD CONSTRAINT FKExplorationClienteleClientId FOREIGN KEY (client) REFERENCES Client (id);
ALTER TABLE ExplorationClientele
    ADD CONSTRAINT FKExplorationClienteleExplorationId FOREIGN KEY (exploration) REFERENCES Explora
ALTER TABLE Exploration
    ADD CONSTRAINT FKExplorationAddressId FOREIGN KEY (address) REFERENCES Address (id);
ALTER TABLE Client
    ADD CONSTRAINT FKClientAddressId FOREIGN KEY (address) REFERENCES Address (id);
ALTER TABLE Hub
    ADD CONSTRAINT FKHubAddressId FOREIGN KEY (address) REFERENCES Address (id);
ALTER TABLE BasketOrder
    ADD CONSTRAINT FKBasketOrderClientId FOREIGN KEY (client) REFERENCES Client (id);
ALTER TABLE BasketOrder
    ADD CONSTRAINT FKBasketOrderBasketId FOREIGN KEY (basket) REFERENCES Basket (id);
ALTER TABLE CulturePlan
    ADD CONSTRAINT FKCulturePlanSprinklingSystemId FOREIGN KEY (sprinklingSystem) REFERENCES Sprink
ALTER TABLE CulturePlan
    ADD CONSTRAINT FKCulturePlanExplorationId FOREIGN KEY (exploration) REFERENCES Exploration (id)
ALTER TABLE CulturePlan
    ADD CONSTRAINT FKCulturePlanExplorationId FOREIGN KEY (sector) REFERENCES Sector (id);
ALTER TABLE Valve
    ADD CONSTRAINT FKValveTubularSystemId FOREIGN KEY (tubularSystem) REFERENCES TubularSystem (id)
ALTER TABLE FieldRecording
    ADD CONSTRAINT FKFieldRecordingExplorationId FOREIGN KEY (exploration) REFERENCES Exploration (
ALTER TABLE CropWatering
    ADD CONSTRAINT FKCropWateringSectorId FOREIGN KEY (sector) REFERENCES Sector (id);
ALTER TABLE Sector
    ADD CONSTRAINT FKSectorProductId FOREIGN KEY (product) REFERENCES Product (id);
ALTER TABLE Harvest
    ADD CONSTRAINT FKHarvestSectorId FOREIGN KEY (sector) REFERENCES Sector (id);
ALTER TABLE CropWatering
    ADD CONSTRAINT FKCropWateringFieldRecordingId FOREIGN KEY (fieldRecording) REFERENCES FieldReco
ALTER TABLE Harvest
    ADD CONSTRAINT FKHarvestFieldRecordingId FOREIGN KEY (fieldRecording) REFERENCES FieldRecording
ALTER TABLE MeteorologicData
    ADD CONSTRAINT FKMeteorologicDataFieldRecordingId FOREIGN KEY (fieldRecording) REFERENCES Field
ALTER TABLE MeteorologicData
    ADD CONSTRAINT FKMeteorologicDataMeteorologicStationId FOREIGN KEY (station) REFERENCES Meteoro
ALTER TABLE ProductionFactorsRecording
    ADD CONSTRAINT FKProductionFactorsRecordingFieldRecordingId FOREIGN KEY (fieldRecording) REFERE
ALTER TABLE ProductionFactorsRecording
    ADD CONSTRAINT FKProductionFactorsRecordingProductionFactorsId FOREIGN KEY (productionFactors)
ALTER TABLE BasketOrder
    ADD CONSTRAINT FKBasketOrderDriverId FOREIGN KEY (driver) REFERENCES Driver (id);
ALTER TABLE BasketProduct
    ADD CONSTRAINT FKBasketProductBasketId FOREIGN KEY (basket) REFERENCES Basket (id);
ALTER TABLE BasketProduct
    ADD CONSTRAINT FKBasketProductProductId FOREIGN KEY (product) REFERENCES Product (id);
ALTER TABLE AuditLog
    ADD CONSTRAINT FKAuditLogSystemUserId FOREIGN KEY (userId) REFERENCES SystemUser (id);


ALTER TABLE BasketOrder
    ADD CONSTRAINT FKBasketOrderAddressId FOREIGN KEY (address) REFERENCES Address (id);
```

Delete Database

```
--DELETE DATABASE--
DROP TABLE Address CASCADE CONSTRAINTS PURGE;
DROP TABLE AuditLog CASCADE CONSTRAINTS PURGE;
DROP TABLE Basket CASCADE CONSTRAINTS PURGE;
DROP TABLE BasketOrder CASCADE CONSTRAINTS PURGE;
DROP TABLE BasketProduct CASCADE CONSTRAINTS PURGE;
DROP TABLE Building CASCADE CONSTRAINTS PURGE;
DROP TABLE Client CASCADE CONSTRAINTS PURGE;
DROP TABLE CropWatering CASCADE CONSTRAINTS PURGE;
DROP TABLE CulturePlan CASCADE CONSTRAINTS PURGE;
DROP TABLE DistributionManager CASCADE CONSTRAINTS PURGE;
DROP TABLE Driver CASCADE CONSTRAINTS PURGE;
DROP TABLE Exploration CASCADE CONSTRAINTS PURGE;
DROP TABLE ExplorationClientele CASCADE CONSTRAINTS PURGE;
DROP TABLE FarmingManager CASCADE CONSTRAINTS PURGE;
DROP TABLE FieldRecording CASCADE CONSTRAINTS PURGE;
DROP TABLE PRODUCTIONFACTORSRECORDING CASCADE CONSTRAINTS PURGE;
DROP TABLE Harvest CASCADE CONSTRAINTS PURGE;
DROP TABLE Hub CASCADE CONSTRAINTS PURGE;
DROP TABLE MachineryGarage CASCADE CONSTRAINTS PURGE;
DROP TABLE MeteorologicData CASCADE CONSTRAINTS PURGE;
DROP TABLE MeteorologicStation CASCADE CONSTRAINTS PURGE;
DROP TABLE Product CASCADE CONSTRAINTS PURGE;
DROP TABLE ProductionEntry CASCADE CONSTRAINTS PURGE;
DROP TABLE ProductionFactors CASCADE CONSTRAINTS PURGE;
DROP TABLE ProductionZones CASCADE CONSTRAINTS PURGE;
DROP TABLE Sector CASCADE CONSTRAINTS PURGE;
DROP TABLE Sensor CASCADE CONSTRAINTS PURGE;
DROP TABLE Silos CASCADE CONSTRAINTS PURGE;
DROP TABLE SprinklingSystem CASCADE CONSTRAINTS PURGE;
DROP TABLE Stable CASCADE CONSTRAINTS PURGE;
DROP TABLE SystemUser CASCADE CONSTRAINTS PURGE;
DROP TABLE TubularSystem CASCADE CONSTRAINTS PURGE;
DROP TABLE Valve CASCADE CONSTRAINTS PURGE;
```

Initial Boot

```
DECLARE
    systemId     Systemuser.ID%type;
    large        BASKET.ID%type;
    average      BASKET.ID%type;
    small        BASKET.ID%type;
    addressResId ADDRESS.ID%type;
    addressDelId ADDRESS.ID%type;
    cId          SYSTEMUSER.ID%type;
BEGIN
    INSERT INTO SYSTEMUSER(EMAIL, PASSWORD)
    VALUES ('system@system.sys', 'qwerty123')
    returning ID into systemId;

    INSERT INTO EXPLORATION(ID) VALUES (1);
    COMMIT;

    INSERT INTO PRODUCT(ID, NAME, TYPE, PRICE) VALUES (1, 'Carrot', 'TEMPORARY', 1);
    INSERT INTO PRODUCT(ID, NAME, TYPE, PRICE) VALUES (2, 'Apple', 'PERMANENT', .80);
    INSERT INTO PRODUCT(ID, NAME, TYPE, PRICE) VALUES (3, 'Honey', 'TEMPORARY', 3);
    INSERT INTO PRODUCT(ID, NAME, TYPE, PRICE) VALUES (4, 'Pears', 'PERMANENT', .75);
    COMMIT;
    INSERT INTO SECTOR(ID, DESIGNATION, AREA, EXPLORATION, CULTUREPLAN, PRODUCT)
    VALUES (1, 'Carrot Filed', 1500, 1, 0, 1);
    INSERT INTO SECTOR(ID, DESIGNATION, AREA, EXPLORATION, CULTUREPLAN, PRODUCT)
    VALUES (2, 'Apple Filed', 150000, 1, 0, 2);
    INSERT INTO SECTOR(ID, DESIGNATION, AREA, EXPLORATION, CULTUREPLAN, PRODUCT) VALUES (3, 'Beehiv
    INSERT INTO SECTOR(ID, DESIGNATION, AREA, EXPLORATION, CULTUREPLAN, PRODUCT)
    VALUES (4, 'Pears Field', 10200, 1, 0, 4);
    COMMIT;
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (SYSDATE, 1, 100);
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('8/10/2022', 'DD/MM/Y
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('10/10/2022', 'DD/MM/
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('9/10/2022', 'DD/MM/Y
    COMMIT;
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (SYSDATE, 2, 1000);
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('8/10/2022', 'DD/MM/Y
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('10/10/2022', 'DD/MM/
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('9/10/2022', 'DD/MM/Y
    COMMIT;
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (SYSDATE, 3, 100);
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('8/10/2022', 'DD/MM/Y
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('10/10/2022', 'DD/MM/
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('9/10/2022', 'DD/MM/Y
    COMMIT;
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (SYSDATE, 4, 150);
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('8/10/2022', 'DD/MM/Y
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('10/10/2022', 'DD/MM/
    INSERT INTO HARVEST(DATEOFHARVEST, SECTOR, NUMBEROFUNITS) VALUES (TO_DATE('9/10/2022', 'DD/MM/Y
    COMMIT;
    INSERT INTO BASKET(PRICE) VALUES (100) RETURNING ID INTO average;
    INSERT INTO BASKET(PRICE) VALUES (10) RETURNING ID INTO small;
    INSERT INTO BASKET(PRICE) VALUES (10000) RETURNING ID INTO large;
    COMMIT;
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (small, 1, 3);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (small, 2, 5);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (small, 4, 2);
    COMMIT;
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (average, 1, 10);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (average, 2, 15);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (average, 4, 10);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (average, 3, 15);
    COMMIT;
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (large, 1, 100);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (large, 2, 150);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (large, 4, 100);
    INSERT INTO BASKETPRODUCT(BASKET, PRODUCT, QUANTITY) VALUES (large, 3, 150);
    COMMIT;
    INSERT INTO ADDRESS(ZIPCODE, DISTRICT)
    VALUES ('Rua da funda 400, 4445-245 Alfena', 'Porto')
    RETURNING ID into addressResId;
    INSERT INTO ADDRESS(ZIPCODE, DISTRICT)
    VALUES ('Rua primeiro de maio 960, 4445-245 Alfena', 'Porto')
    RETURNING ID into addressDelId;
    COMMIT;
    INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES ('tomcat@java.com', 'Catalina') RETURNING ID INT
    INSERT INTO CLIENT(ID, ADDRESS, NAME, NIF, PLAFOND, INCIDENTS, LASTINCIDENTDATE, LASTYEARORDERS
                       ADDRESSOFDELIVERY, PRIORITYLEVEL, LASTYEARINCIDENTS)
    VALUES (cID, addressResId, 'Apache Tomcat', 212345678, 100000, 0, null, 1, 100, addressDelId, '
    COMMIT;
```

```
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE)
    VALUES (cId, small, 2, 'DELIVERED', addressDelId, 'Y', SYSDATE - 3);
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE)
    VALUES (cId, average, 3, 'DELIVERED', addressDelId, 'Y', SYSDATE - 23);
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE)
    VALUES (cId, large, 10, 'REGISTERED', addressDelId, 'Y', SYSDATE);
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE)
    VALUES (cId, small, 1, 'REGISTERED', addressDelId, 'Y', SYSDATE - 10);
    COMMIT;

    INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES ('gradle@copy-maven.org', 'IwishIwasMav€n') RETU
    INSERT INTO CLIENT(ID, ADDRESS, NAME, NIF, PLAFOND, INCIDENTS, LASTINCIDENTDATE, LASTYEARORDERS
                       ADDRESSOFDELIVERY, PRIORITYLEVEL, LASTYEARINCIDENTS)
    VALUES (cID, addressResId, 'Apache Mav... I mean, Gradle', 112345678, 1, 10, SYSDATE - 10, 10,
            addressDelId, 'C', 10);
    COMMIT;
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE, DUEDATE)
    VALUES (cId, small, 2, 'DELIVERED', addressDelId, 'N', SYSDATE - 100, SYSDATE - 90);
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE, DUEDATE)
    VALUES (cId, average, 3, 'DELIVERED', addressDelId, 'N', SYSDATE - 23, SYSDATE - 13);

    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE)
    VALUES (cId, large, 10, 'REGISTERED', addressDelId, 'N', SYSDATE);
    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, STATUS, ADDRESS, PAYED, ORDERDATE, DUEDATE)
    VALUES (cId, small, 1, 'REGISTERED', addressDelId, 'N', SYSDATE - 10, SYSDATE);
    COMMIT;

end;


DECLARE
    val NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Data Report:');
    DBMS_OUTPUT.PUT_LINE('Table ==> Number of Entries');
    DBMS_OUTPUT.PUT_LINE('===========================');
    FOR I IN (SELECT TABLE_NAME FROM USER_TABLES ORDER BY TABLE_NAME)
        LOOP
            EXECUTE IMMEDIATE 'SELECT count(*) FROM ' || i.table_name INTO val;
            DBMS_OUTPUT.PUT_LINE(i.table_name || ' ==> ' || val);
        END LOOP;
END;
```

Result Report

ADDRESS

ID	ZIPCODE	DISTRICT
31	Rua da funda 400, 4445-245 Alfena	Porto
32	Rua primeiro de maio 960, 4445-245 Alfena	Porto

BASKET

ID	DATEOFCREATION	PRICE

19	2022-12-04 17:06:19	100
20	2022-12-04 17:06:19	10
21	2022-12-04 17:06:19	10000

BASKETORDER

CLIENT	BASKET	QUANTITY	DRIVER	ORDERDATE	DUEDATE	DELIVERYDAT
40	19	3	null	2022-11-11 17:06:20	2022-11-21 17:06:20	2023-01-03 17:06:20
40	21	10	null	2022-12-04 17:06:20	2022-12-14 17:06:20	2023-01-03 17:06:20
40	20	1	null	2022-11-24 17:06:20	2022-12-04 17:06:20	2023-01-03 17:06:20
39	20	2	null	2022-12-01 17:06:20	2022-12-14 17:06:20	2023-01-03 17:06:20
39	19	3	null	2022-11-11 17:06:20	2022-12-14 17:06:20	2023-01-03 17:06:20
39	21	10	null	2022-12-04 17:06:20	2022-12-14 17:06:20	2023-01-03 17:06:20
39	20	1	null	2022-11-24 17:06:20	2022-12-14 17:06:20	2023-01-03 17:06:20
40	20	2	null	2022-08-26 17:06:20	2022-09-05 17:06:20	2023-01-03 17:06:20

BASKETPRODUCT

BASKET	PRODUCT	QUANTITY
19	2	15
19	4	10
19	3	15

21	1	100
21	2	150
21	4	100
21	3	150
20	1	3
20	2	5
20	4	2
19	1	10

Client

ID	ADDRESS	NAME	NIF	PLAFOND	INCIDENTS	LASTINCIDENTDATE
39	31	Apache Tomcat	212345678	100000	0	null
40	31	Apache Mav... I mean, Gradle	112345678	1	10	2022-11-24 17:06:20

EXPLORATION

ID	ADDRESS
1	null

FIELDRECORDING

EXPLORATION
1

HARVEST

DATEOFHARVEST	SECTOR	NUMBEROFUNITS	FIELDRECORDING
2022-10-08	2	80	1
2022-10-10	2	300	1
2022-10-09	2	870	1
2022-12-04 17:06:19	3	100	1
2022-10-08	3	8	1

2022-10-10	3	200	1
2022-10-09	3	87	1
2022-12-04 17:06:19	4	150	1
2022-10-08	4	86	1
2022-10-10	4	2	1
2022-10-09	4	0	1
2022-12-04 17:06:19	1	100	1
2022-10-08	1	8	1
2022-10-10	1	30	1
2022-10-09	1	87	1
2022-12-04 17:06:19	2	1000	1

METEOROLOGICSTATION

ID	NAME
45	Station
46	Station
47	Station
48	Station

PRODUCT

NAME	TYPE	ID	PRICE
Carrot	TEMPORARY	1	1.00
Apple	PERMANENT	2	0.80
Honey	TEMPORARY	3	3.00
Pears	PERMANENT	4	0.75

SECTOR

ID	DESIGNATION	AREA	EXPLORATION	METEOROLOGICSTATION	CULTUREPL
1	Carrot Filed	1500	1	45	0
2	Apple Filed	150000	1	46	0
3	Beehive	15	1	47	0
4	Pears Field	10200	1	48	0

4	Pears Field	10200	1	48	0

SYSTEMUSER

ID	EMAIL	PASSWORD
40	gradle@copy-maven.org	IwishIwasMav€n
38	system@system.sys	qwerty123
39	tomcat@java.com	Catalina

NOTE: Some Ids may alter because of usage of Identity on Insertion!

Database Logic

Files to Include

- CREATE PROCEDURES
- DELETE PROCEDURES
- CREATE FUNCTIONS
- DELETE FUNCTIONS
- CREATE TRIGGERS
- DELETE TRIGGERS
- CREATE VIEWS
- DELETE VIEWS

Procedures

US205

prcUS205AlterClientLastYearInfo

This procedure has the objective of altering the client's information regarding last year operations, recieving the id of the client, the number of orders and the amount spent on said orders, updating said information.

```
CREATE OR REPLACE PROCEDURE prcUS205AlterClientLastYearInfo(clientId IN SYSTEMUSER.ID%type,
                                                            numberOfOrders IN CLIENT.LASTYEARORDERS
                                                            spentOnOrders IN CLIENT.LASTYEARSPENT%t


    newOrders CLIENT.LASTYEARORDERS%type;
    newSpent  CLIENT.LASTYEARSPENT%type;
BEGIN

    SELECT LASTYEARORDERS, LASTYEARSPENT INTO newOrders,newSpent FROM CLIENT WHERE ID = clientId;
    SELECT LASTYEARORDERS, LASTYEARSPENT INTO newOrders,newSpent FROM CLIENT WHERE ID = clientId;

    if (numberOfOrders IS NOT NULL) THEN
        newOrders := numberOfOrders;
    end if;

    if (spentOnOrders IS NOT NULL) THEN
        newSpent := spentOnOrders;
    end if;




    UPDATE CLIENT SET LASTYEARORDERS = newOrders, LASTYEARSPENT = newSpent WHERE CLIENT.ID = client
end;
```

```
DROP PROCEDURE prcUS205AlterClientLastYearInfo;
```

US206

prcUS206CreateSector

This procedure has the objective to create a sector in the database, receiving the necessary parameters for such functionality and archiving the command on the AuditLog Table. This procedure will also return an out only variable with the sector id created.

```
CREATE OR REPLACE PROCEDURE prcUS206CreateSector(userCallerId IN SYSTEMUSER.ID%type,
                                                 designationParam IN Sector.DESIGNATION%type,
                                                 areaParam IN SECTOR.AREA%type,
                                                 explorationId IN SECTOR.EXPLORATION%type,
                                                 productId IN SECTOR.PRODUCT%type, sectorId out SEC
begin
    SAVEPOINT BeforeCall;
    INSERT INTO SECTOR(DESIGNATION, AREA, EXPLORATION, CULTUREPLAN, PRODUCT)
    VALUES (designationParam, areaParam, explorationId, 0, productId) RETURNING ID INTO sectorId;
    INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
    VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO SECTOR(DESIGNATION, AREA, EXPLORATION, CU
    VALUES (designationParam, areaParam, explorationId, 0, productId);');
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Added sector to database');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Could not create entry to the database');
        ROLLBACK TO SAVEPOINT BeforeCall;
end;
```

```
DROP PROCEDURE PRCUS206CREATESECTOR;
```

US208

Como Gestor Agrícola, quero manter os fatores de produção classificados por tipo (fertilizante, correctivo mineral, produto fitofármaco, etc.), incluindo a sua ficha técnica – que deve ser persistida na base de dados.

Critério de Aceitação:

1. Um utilizador pode configurar fatores de produção.
2. É possível persistir na base de dados uma ficha técnica semelhante à da Fig. 3.
   i. O modelo de dados inclui as tabelas necessárias para persistir fichas técnicas
   ii. Está disponível o código para persistir uma ficha técnica (nome comercial, fornecedor, tipo de fator de produção) e cada um dos seus elementos (categoria, como por exemplo SUSTÂNCIA ORGÂNICAS, substância, quantidade e unidade)

prcUS208AddProductionFactor

This function will add an entry to the production factors used in the exploration, receiving the id of the user who called this function, the id of the exploration, the commercial name of the product, the formulation of the product, the name of the supplier chain or enterprise and will return the id of said factor.

```
CREATE OR REPLACE PROCEDURE prcUS208AddProductionFactor(userCallerId in SYSTEMUSER.ID%type,
                                                        fieldRecordingId IN FIELDRECORDING.EXPLORAT
                                                        productName IN PRODUCTIONFACTORS.NAME%type,
                                                        productFormulation IN PRODUCTIONFACTORS.FOR
                                                        supplierName IN PRODUCTIONFACTORS.SUPPLIER%
                                                        productFactorId OUT PRODUCTIONFACTORS.ID%ty
    dateToUse DATE := sysdate;
BEGIN
    SAVEPOINT BeforeCall;
    INSERT INTO PRODUCTIONFACTORS(NAME, FORMULATION, SUPPLIER)
    VALUES (productName, productFormulation, supplierName)
    returning ID into productFactorId;

    INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
    VALUES (dateToUse, userCallerId, 'INSERT', 'INSERT INTO PRODUCTIONFACTORS(NAME, FORMULATION,SUP
    VALUES (' || productName || ',' || productFormulation || ',' || supplierName || ')');

    INSERT INTO PRODUCTIONFACTORSRECORDING(FIELDRECORDING, PRODUCTIONFACTORS, DATEOFRECORDING)
    VALUES (fieldRecordingId, productFactorId, dateToUse);
    INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
    VALUES (dateToUse, userCallerId, 'INSERT', 'INSERT INTO PRODUCTIONFACTORSRECORDING(FIELDRECORDI
    VALUES (' || fieldRecordingId || ',' || productFactorId || ',' || dateToUse || ')');
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Added factor to the database');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Could not create the entry for the product');
        ROLLBACK TO SAVEPOINT BeforeCall;
end;
```

```
DROP FUNCTION prcUS208AddProductionFactor;
```

prcUS208AddEntryToProductionFactor

This function will add an entry to the composition of a certain production factor used in the exploration, receiving the id of the user who called this function, the id of the factor, the name of the entry, the unit of the entry (ex. mL, Kg/m^3, etc.), the amount present on the product and the type of the entry.

```
CREATE OR REPLACE PROCEDURE prcUS208AddEntryToProductionFactor(userCallerId in SYSTEMUSER.ID%type,
                                                               productFactorId in PRODUCTIONFACTORS
                                                               entryName IN PRODUCTIONENTRY.NAME%ty
                                                               unitName IN PRODUCTIONENTRY.UNIT%typ
                                                               unitValue IN PRODUCTIONENTRY.VALUE%t
                                                               unitType IN PRODUCTIONENTRY.TYPE%typ
BEGIN
    SAVEPOINT BeforeCall;
    INSERT INTO PRODUCTIONENTRY(ID, VALUE, UNIT, TYPE, NAME)
    VALUES (productFactorId, unitValue, unitName, unitType, entryName);
    INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
    VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO PRODUCTIONENTRY(ID, VALUE, UNIT, TYPE, NA
    VALUES (' || productFactorId || ',' || unitValue || ',' || unitName || ',' || unitType || ',' |
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Added entry to the database');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Could add the entry for the product');
        ROLLBACK TO SAVEPOINT BeforeCall;
end;
```

```
DROP PROCEDURE prcUS208AddEntryToProductionFactor;
```

prcUS209OrderBasket

This procedure will order a certain amount of a basket to an user; it will receive the id of the client, the id of the basket, the amount of baskets, the due date to pay the order, the address to deliver the basket, the probable date of deliver of the product. For that, this procedure will validate if the order (plus all the unpaid orders) surpasses the plafond of the client, proceeding with the order if the plafond is not exceeded.

```
CREATE OR REPLACE PROCEDURE prcUS209OrderBasket(clientId IN SYSTEMUSER.ID%type, basketId IN BASKETC
                                                numberOfBaskets IN BASKETORDER.QUANTITY%type,
                                                orderDueDate IN BASKETORDER.DUEDATE%type DEFAULT SY
                                                deliveryAddress IN BASKETORDER.ADDRESS%type DEFAULT
                                                orderDeliveryDate IN BASKETORDER.DELIVERYDATE%type

    unpaidValue   NUMERIC;
    basketPrice   NUMERIC;
    orderPrice    NUMERIC;
    clientPlafond NUMERIC;
BEGIN

    SELECT (SELECT sum(P.PRICE)
            FROM BASKET
                     JOIN BASKETPRODUCT B on BASKET.ID = B.BASKET
                     JOIN PRODUCT P on P.ID = B.PRODUCT
            WHERE BASKET.ID = PARENT.BASKET) * PARENT.QUANTITY
    into unpaidValue
    FROM BASKETORDER PARENT
    WHERE CLIENT = clientId
      AND PAYED='N';

    SELECT sum(P.PRICE)
    into basketPrice
    FROM BASKETPRODUCT B
             JOIN PRODUCT P on P.ID = B.PRODUCT
    WHERE B.BASKET = basketId;

    orderPrice := basketPrice * numberOfBaskets;
    SELECT PLAFOND into clientPlafond from CLIENT where ID = clientId;

    if (clientPlafond < orderPrice + unpaidValue) then
        raise_application_error(-20005, 'Order exceeds client plafond limit!');
    end if;

    INSERT INTO BASKETORDER(CLIENT, BASKET, QUANTITY, DUEDATE, DELIVERYDATE, ADDRESS)
    VALUES (clientId, basketId, numberOfBaskets, orderDueDate, orderDeliveryDate, deliveryAddress);

end;



DROP PROCEDURE prcUS209OrderBasket;
```

Functions

US205

Como Gestor Agrícola, quero gerir os meus clientes, empresas ou particulares, que compram os bens produzidos na minha exploração agrícola. Um cliente é caracterizado por um código interno, nome, número fiscal, email, morada de correspondência, morada de entrega, plafond, número de incidentes, data do último incidente, número de encomendas colocadas no último ano, valor total das encomendas colocadas no último ano. A morada deve incluir o código postal que é utilizado para análises de vendas. O plafond é o limite máximo de crédito atribuído o cliente – os clientes não podem ter um valor total de encomendas pendentes de pagamento superior ao seu plafond. Os incidentes – pagamentos de encomendas que não foram efetuados na data de vencimento, são caracterizados por cliente, valor, data em que ocorreram e data em que foram sanados e devem ser registados. A cada cliente é atribuído um nível (A, B, C) que caracteriza o seu valor para o negócio. Clientes que não tenham incidentes reportados nos últimos 12 meses e que tenham um volume total de vendas (encomendas pagas) no mesmo período superior a 10000€ são do nível A; clientes sem incidentes reportados nos últimos 12 meses e que tenham um volume total de vendas (encomendas pagas) no mesmo período superior a 5000€ são do nível B; clientes que tenham incidentes reportados nos últimos 12 meses são do nível C independentemente do volume de vendas.

Critério de Aceitação:

1. Um utilizador pode inserir um novo Cliente na Base de Dados, com os dados que descrevem um cliente, sem a necessidade de escrever código SQL. Se a inserção for bem-sucedida, o utilizador é informado sobre o valor da chave primária do novo cliente
2. Quando o processo de inserção falha, o utilizador é informado sobre o erro que pode ter ocorrido.

3. O administrador pode executar um procedimento que atualiza o número e o valor total das encomendas colocadas no último ano por cada cliente
4. Criar uma View que agregue para cada cliente:
    i. o seu nível (A, B, C),
    ii. a data do último incidente – ou a menção “Sem incidentes à data” caso não tenha incidentes reportados
    iii. o volume total de vendas (encomendas pagas) nos últimos 12 meses e
    iv. o volume total das encomendas já entregues mas ainda pendentes de pagamento.
5. implemente uma função que retorna o fator de risco de um cliente. O fator de risco de um cliente é dado pelo rácio entre o valor total dos incidentes observados nos últimos 12 meses e o número de encomendas colocadas depois do último incidente e ainda pendentes de pagamento. Por exemplo, um cliente que tenha um total de incidentes de 2400€ e tenha feito 3 encomendas depois do último incidente que ainda não pagou tem um fator de risco de 800€ (2400/3)

fncUS205CreateUser

To create an SystemUser into the database, this function will receive all the necessary information for its functionality and will try to create the user, archiving as records the results. Firstly, the function will validate if the chosen email is not yet taken, if that is not the case, the function will raise an error.

Then, the function will register the SystemUser, associating with the correct type of SystemUser via the *userType* variable creating the record into the correct table. The function will print the user id and will return such value.

```
--DEPRECATED FOR CLIENTS--
CREATE OR REPLACE FUNCTION fncUS205CreateUser(userCallerId IN SYSTEMUSER.ID%type, userType IN VARCH
                                              userEmail IN SYSTEMUSER.EMAIL%TYPE,
                                              userPassword IN SYSTEMUSER.PASSWORD%TYPE) RETURN SYST
    userId    SYSTEMUSER.ID%TYPE;
    nullEmail SYSTEMUSER.ID%TYPE;
BEGIN
    SAVEPOINT BeforeCall;
    SELECT EMAIL into nullEmail FROM SYSTEMUSER WHERE EMAIL = userEmail;

    if (nullEmail is not null) then
        RAISE_APPLICATION_ERROR(-20001, 'Email already exists in database!');
    end if;

    INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES (userEmail, userPassword);
    INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
    VALUES (sysdate, userCallerId, 'INSERT',
            'INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES (' || userEmail || ',' || userPassword
    SELECT ID into userId FROM SYSTEMUSER WHERE EMAIL = userEmail;
    if (lower(userType) = 'client') then
        INSERT INTO CLIENT(ID) VALUES (userId);
        INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
        VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO CLIENT(ID) VALUES (' || userId || ');
    elsif (lower(userType) = 'driver') then
        INSERT INTO DRIVER(ID) VALUES (userId);
        INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
        VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO DRIVER(ID) VALUES (' || userId || ');
    elsif (lower(userType) = 'farm') then
        INSERT INTO FARMINGMANAGER(ID) VALUES (userId);
        INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
        VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO FARMINGMANAGER(ID) VALUES (' || userI
    elsif (lower(userType) = 'distribution') then
        INSERT INTO DISTRIBUTIONMANAGER(ID) VALUES (userId);
        INSERT INTO AUDITLOG(DATEOFACTION, USERID, TYPE, COMMAND)
        VALUES (sysdate, userCallerId, 'INSERT', 'INSERT INTO DISTRIBUTIONMANAGER(ID) VALUES (' ||
    else
        ROLLBACK;
        RAISE_APPLICATION_ERROR(-20002,
                                'User type is incorrect! It should be one of the following: [client
    end if;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('New System User ID: ' || userId);
    return userId;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK TO SAVEPOINT BeforeCall;
        RAISE;
end;



DROP FUNCTION FNCUS205CREATEUSER;
```

fncUS205ClientRiskFactor

This function calculates the risk factor of a certain user. For that, it requires the id of the user; firstly, it finds all the orders that are late an then calculates the missing amount. Then it counts the number of incidents in the last 365 days and returns the ratio between the missing amount and the number of incidents in the last 365 days.

```
CREATE OR REPLACE FUNCTION fncUS205ClientRiskFactor(clientId IN CLIENT.ID%TYPE) RETURN NUMERIC AS
    result     NUMERIC;
    tmp        NUMERIC;
    itr        Sys_Refcursor;
    basketId   BASKET.ID%type;
    amount     BASKETORDER.QUANTITY%type;
    incidentsN NUMERIC;
BEGIN
    OPEN itr FOR SELECT BASKETORDER.BASKET, BASKETORDER.QUANTITY
                 FROM BASKETORDER
                          JOIN CLIENT C2 on C2.ID = BASKETORDER.CLIENT
                 WHERE ORDERDATE >= COALESCE(LASTINCIDENTDATE, TO_DATE('01/01/0001', 'DD/MM/YYYY'))
                   AND PAYED = 'N' AND CLIENT=clientId;
    result := 0;
    LOOP
        FETCH itr INTO basketId,amount;
        EXIT WHEN itr%notfound;
        SELECT BASKET.PRICE
        into tmp
        FROM BASKET;
        result := result + tmp * amount;
    end loop;

    SELECT count(*)
    into incidentsN
    FROM BASKETORDER
    WHERE PAYED = 'N'
      AND CLIENT = clientId;
      AND CLIENT = clientId
      AND ORDERDATE >= SYSDATE - 365
      AND DUEDATE < SYSDATE;
    return result / incidentsN;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        return 0;
end;



DROP FUNCTION fncUS205ClientRiskFactor;
```

fncUS205CreateClient

This function will create a client in the database. For that, the function will receive all the necessary information for validating if the password is not null and if it is, will use the default one, verify if both addresses are null, if no more that one is null, the function will override the null one with the value of the not null, then, finally, will create the user and the client taking into account all the information, logging any database alteration.

```
CREATE OR REPLACE FUNCTION fncUS205CreateClient(userCallerId IN SYSTEMUSER.ID%type, userEmail IN SYST
                                                addressOfResidence IN ADDRESS.ZIPCODE%type,
                                                addressOfDelivery IN ADDRESS.ZIPCODE%type,
                                                clientName IN CLIENT.NAME%type, clientNIF IN CLIENT.N
                                                userPassword in SYSTEMUSER.PASSWORD%type DEFAULT NULL
                                                clientPlafond IN CLIENT.PLAFOND%type DEFAULT 100000,
                                                clientIncidents IN CLIENT.INCIDENTS%type DEFAULT 0,
                                                clientLastIncidentDate IN CLIENT.LASTINCIDENTDATE%typ
                                                clientLastYearOrders IN CLIENT.LASTYEARORDERS%type DE
                                                clientLastYearSpent IN CLIENT.LASTYEARSPENT%type DEFA
                                                clientPriority IN CLIENT.PRIORITYLEVEL%type DEFAULT '
                                                clientLastYearIncidents IN CLIENT.LASTYEARINCIDENTS%t

    clientId           SYSTEMUSER.ID%type;
    tmpDistrict        ADDRESS.DISTRICT%type;
    idAddressResidence ADDRESS.ID%type;
    idAddressDelivery  ADDRESS.ID%type;
    realPassword       SYSTEMUSER.PASSWORD%type;
    resAddr            ADDRESS.ZIPCODE%type;
    devAddr            ADDRESS.ZIPCODE%type;
BEGIN

    if (userPassword IS NULL) then
        realPassword := 'Qwerty123';
    else
        realPassword := userPassword;
    end if;

    if (COALESCE(addressOfDelivery, addressOfResidence) IS NULL) then
        RAISE_APPLICATION_ERROR(-20003, 'Zipcodes cannot be null');
    end if;
    devAddr := addressOfDelivery;
    resAddr := addressOfResidence;
    if (addressOfDelivery IS NULL) THEN
        devAddr := addressOfResidence;
    ELSIF (addressOfResidence IS NULL) THEN
        resAddr := addressOfDelivery;
        resAddr := addressOfDelivery;
    end if;
    INSERT INTO ADDRESS(zipcode) VALUES (devAddr) returning ID into idAddressDelivery;
    INSERT INTO ADDRESS(zipcode) VALUES (resAddr) returning ID into idAddressResidence;
    INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES (userEmail, realPassword) returning ID INTO client
    PRCUS000LOG(userCallerId, 'INSERT',
                'INSERT INTO SYSTEMUSER(EMAIL, PASSWORD) VALUES (' || userEmail || ',' || userPasswor
                ') returning ID INTO clientId');

    INSERT INTO CLIENT(ID, ADDRESS, NAME, NIF, PLAFOND, INCIDENTS, LASTINCIDENTDATE, LASTYEARORDERS,
                       ADDRESSOFDELIVERY, PRIORITYLEVEL, LASTYEARINCIDENTS)
    VALUES (clientId, idAddressResidence, clientName, clientNIF, clientPlafond, clientIncidents, clie
            clientLastYearOrders, clientLastYearSpent, idAddressDelivery, clientPriority, clientLastY
    return clientId;
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        RAISE_APPLICATION_ERROR(-20001, 'Email already exists in database!');
        return null;
    WHEN OTHERS THEN
        RAISE;
end;
```

```
DROP FUNCTION fncUS205CreateClient;
```

US206

Como Gestor Agrícola, quero manter a estrutura da minha exploração agrícola – contendo um conjunto de Setores – atualizada, ou seja, quero especificar cada um dos Setores. As suas características, como tipo de cultivo e cultivo, devem ser configuradas.

Critério de Aceitação:

1. Um utilizador pode criar Setores numa exploração agrícola Biológica especificando suas características.
2. É possível definir novos tipos de características parametrizadas, como tipo de cultura ou cultura entre outras.
3. Um utilizador podem listar os Setores de sua exploração agrícola ordenados por ordem alfabética.
4. Um utilizador podem listar os Setores de sua exploração agrícola ordenados por tamanho, em ordem crescente ou decrescente.
5. Um utilizador podem listar os Setores de sua exploração agrícola ordenados por tipo de cultura e cultura.

fncUS206OrderSectorByDesignation

To order the sectors in the most convenient way possible, this function will open a cursor for the selection of the sectors from the exploration with *explorationId* and order them (the sectors) by their designation, alphabetically.

After having the result, the function returns the cursor containing a list of elements with the `Sector%ROWTYPE` profile.

```
CREATE OR REPLACE FUNCTION fncUS206OrderSectorByDesignation(explorationId IN EXPLORATION.ID%type)
CREATE OR REPLACE FUNCTION fncUS206OrderSectorByDesignation(explorationId IN EXPLORATION.ID%type)
    RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    OPEN result for SELECT * FROM SECTOR WHERE EXPLORATION = explorationId ORDER BY DESIGNATION;
    return result;
end;



DROP FUNCTION fncUS206OrderSectorByDesignation;
```

fncUS206OrderSectorBySize

To order the sectors, by size, in the most convenient way possible, this function will open a cursor for the selection of the sectors from the exploration with *explorationId* and order them (the sectors) by their area by, depending on the criteria passed as parameter by *orderType*, ascending or descending order.

After having the result, the function returns the cursor containing a list of elements with the `Sector%ROWTYPE` profile.

```
CREATE OR REPLACE FUNCTION fncUS206OrderSectorBySize(explorationId IN EXPLORATION.ID%type,
                                                     orderType IN VARCHAR2 DEFAULT 'ASC')
    RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    if (orderType = 'DESC') then
        OPEN result for SELECT * FROM SECTOR WHERE EXPLORATION = explorationId ORDER BY AREA DESC;
    else
        OPEN result for SELECT * FROM SECTOR WHERE EXPLORATION = explorationId ORDER BY AREA;
    end if;
    return result;
end;



DROP FUNCTION fncUS206OrderSectorBySize;
```

fncUS206OrderSectorByCrop

To order the sectors, by crop type or name (depending on the argument *arg*), in the most convenient way possible, this function will open a cursor for the selection of the sectors from the exploration with *explorationId*, inner joining the sectors with the products on their productId and order such results, depending on the criteria passed as parameter by *orderType*, ascending or descending order.

After having the result, the function returns the cursor containing a list of elements with the `{SECTOR.ID%type, SECTOR.DESIGNATION%type, PRODUCT.NAME%type, PRODUCT.TYPE%type}` profile.

```
CREATE OR REPLACE FUNCTION fncUS206OrderSectorByCrop(explorationId IN EXPLORATION.ID%type, arg IN V
                                                     orderType IN VARCHAR2 DEFAULT 'ASC')
    RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    if (arg = 'TYPE') then
    if (arg = 'TYPE') then
        if (orderType = 'DESC') then
            OPEN result for SELECT SECTOR.ID, DESIGNATION, P.NAME, P.TYPE
                            FROM SECTOR
                                     JOIN PRODUCT P on P.ID = SECTOR.PRODUCT
                            WHERE EXPLORATION = explorationId
                            ORDER BY P.TYPE DESC;
        else
            OPEN result for SELECT SECTOR.ID, DESIGNATION, P.NAME, P.TYPE
                            FROM SECTOR
                                     JOIN PRODUCT P on P.ID = SECTOR.PRODUCT
                            WHERE EXPLORATION = explorationId
                            ORDER BY P.TYPE;
        end if;

    else
        if (orderType = 'DESC') then
            OPEN result for SELECT SECTOR.ID, DESIGNATION, P.NAME, P.TYPE
                            FROM SECTOR
                                     JOIN PRODUCT P on P.ID = SECTOR.PRODUCT
                            WHERE EXPLORATION = explorationId
                            ORDER BY P.NAME DESC;
        else
            OPEN result for SELECT SECTOR.ID, DESIGNATION, P.NAME, P.TYPE
                            FROM SECTOR
                                     JOIN PRODUCT P on P.ID = SECTOR.PRODUCT
                            WHERE EXPLORATION = explorationId
                            ORDER BY P.NAME;
        end if;
    end if;
    return result;
end;
```

```
DROP FUNCTION fncUS206OrderSectorByCrop;
```

US207

Como Gestor Agrícola, quero saber o quão rentáveis são os setores da minha exploração agrícola.

Critério de Aceitação:

1. Um utilizador pode listar os Setores de sua exploração agrícola ordenados por ordem decrescente da quantidade de produção em uma determinada safra, medida em toneladas por hectare.
2. Um utilizador pode listar os Setores de sua exploração agrícola ordenados por ordem decrescente do lucro por hectare em uma determinada safra, medido em K€ por hectare.

fncUS207OrderSectorByMaxHarvest

To order the sectors, by harvest, in the most convenient way possible, this function will open a cursor for the selection of the sectors from the exploration with *explorationId*, inner joining the sectors with the harvests on their sectorId, grouping the results by sectorId and sectorDesignation, finding the maximum harvest of said group and order such results, depending on the criteria passed as parameter by *orderType*, ascending or descending order.

After having the result, the function returns the cursor containing a list of elements with the `{SECTOR.DESIGNATION%type, HARVEST.NUMBEROFUNITS%type}` profile.

```
CREATE OR REPLACE FUNCTION fncUS207OrderSectorByMaxHarvest(explorationId IN EXPLORATION.ID%type,
                                                           orderType IN VARCHAR2 DEFAULT 'ASC')
    RETURN SYS_REFCURSOR AS
    result SYS_REFCURSOR;
BEGIN
    if (orderType = 'DESC') then
        OPEN result FOR SELECT S.DESIGNATION, max(H.NUMBEROFUNITS) as HARVEST
                        FROM SECTOR S
                                 JOIN HARVEST H on S.ID = H.SECTOR
                        WHERE S.EXPLORATION = explorationId
                        GROUP BY S.ID, S.DESIGNATION
                        ORDER BY HARVEST DESC;
    else
        OPEN result FOR SELECT S.DESIGNATION, max(H.NUMBEROFUNITS) as HARVEST
                        FROM SECTOR S
                                 JOIN HARVEST H on S.ID = H.SECTOR
                        WHERE S.EXPLORATION = explorationId
                        GROUP BY S.ID, S.DESIGNATION
                        ORDER BY HARVEST;
    end if;
    return result;
end;
```

```
DROP FUNCTION fncUS207OrderSectorByMaxHarvest;
```

fncUS207OrderSectorByRentability

To order the sectors, by harvest, in the most convenient way possible, this function will open a cursor for the selection of the sectors from the exploration with *explorationId*, inner joining the sectors with the harvests on their sectorId and inner joining, yet again, with the products on productId, grouping the results by sectorDesignation and productPrice, finding the average harvest of said group, multiplying the average amount by the price by unit of each product ordering such results, depending on the criteria passed as parameter by *orderType*, ascending or descending order.

After having the result, the function returns the cursor containing a list of elements with the `{SECTOR.DESIGNATION%type, NUMERIC}` profile.

```
CREATE OR REPLACE FUNCTION fncUS207OrderSectorByRentability(explorationId IN EXPLORATION.ID%type,
                                                            orderType IN VARCHAR2 DEFAULT 'ASC')
    RETURN SYS_REFCURSOR as
    result Sys_Refcursor;
BEGIN
    IF (orderType = 'DESC') then
        OPEN result FOR SELECT S.DESIGNATION, avg(H.NUMBEROFUNITS) * P.PRICE
                        FROM SECTOR S
                                 JOIN PRODUCT P on P.ID = S.PRODUCT
                                 JOIN HARVEST H on S.ID = H.SECTOR
                        WHERE S.EXPLORATION = explorationId
                        GROUP BY S.DESIGNATION, P.PRICE
                        ORDER BY 2 DESC;
    else
        OPEN result FOR SELECT S.DESIGNATION, avg(H.NUMBEROFUNITS) * P.PRICE
                        FROM SECTOR S
                                 JOIN PRODUCT P on P.ID = S.PRODUCT
                                 JOIN HARVEST H on S.ID = H.SECTOR
                        WHERE S.EXPLORATION = explorationId
                        GROUP BY S.DESIGNATION, P.PRICE
                        ORDER BY 2;
    end if;
    return result;
end;
```

```
DROP FUNCTION fncUS207OrderSectorByRentability;
```

US209

fncUS209ListOrdersByStatus

This function will simply return a cursor with the result of all the orders with a certain status

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersByStatus(orderStatus BASKETORDER.STATUS%type) RETURN S
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT * FROM BASKETORDER WHERE STATUS = orderStatus;
    return result;
end;
```

```
DROP FUNCTION fncUS209ListOrdersByStatus;
```

fncUS209ListOrdersByDateOfOrder

This function will simply return a cursor with the result of all the orders sorted by order by their ordering date

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersByDateOfOrder RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT * FROM BASKETORDER ORDER BY ORDERDATE;
    return result;
end;
```

```
DROP FUNCTION fncUS209ListOrdersByDateOfOrder;
```

fncUS209ListOrdersByClient

This function will simply return a cursor with the result of all the orders of a certain client sorted by order by their ordering date

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersByClient(idClient BASKETORDER.CLIENT%type) RETURN SYS_
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT * FROM BASKETORDER WHERE CLIENT = idClient ORDER BY ORDERDATE;
    return result;
end;



DROP FUNCTION fncUS209ListOrdersByClient;
```

fncUS209ListOrdersById

This function will simply return a cursor with the result of all the orders sorted by order by their number

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersById RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT * FROM BASKETORDER ORDER BY BASKETORDER.ORDERNUMBER;
    return result;
end;



DROP FUNCTION fncUS209ListOrdersById;
```

fncUS209ListOrdersByOrderNumber

This function will simply return a cursor with the result of all the orders sorted by order number

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersByOrderNumber RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT * FROM BASKETORDER ORDER BY BASKETORDER.ORDERNUMBER;
    return result;
end;



DROP FUNCTION fncUS209ListOrdersByOrderNumber;
```

fncUS209ListOrdersByPrice

This function will list all orders by their price. For that, it is necessary to join the table of orders with the table of baskets to obtain the price of the basket, multiplying by the number of ordered baskets.

```
CREATE OR REPLACE FUNCTION fncUS209ListOrdersByPrice RETURN SYS_REFCURSOR AS
    result Sys_Refcursor;
BEGIN
    OPEN result FOR SELECT CLIENT,
                           BASKET,
                           QUANTITY,
                           DRIVER,
                           ORDERDATE,
                           DUEDATE,
                           DELIVERYDATE,
                           STATUS,
                           ADDRESS,
                           ORDERNUMBER,
                           B.PRICE * PA.QUANTITY as PRICE
                    FROM BASKETORDER PA
                             JOIN BASKET B on B.ID = PA.BASKET
                    ORDER BY PRICE DESC;
    return result;
end;
```

```
DROP FUNCTION fncUS209ListOrdersByPrice;
```

Triggers

trgCreateMeteorologicStation

This triggers guarantees that any sector that is inserted into the database will receive its own meteorologic station with the generic name 'Station' that can be altered in the future.

```
CREATE OR REPLACE TRIGGER trgCreateMeteorologicStation
    BEFORE INSERT
    ON SECTOR
    FOR EACH ROW
    WHEN ( new.METEOROLOGICSTATION is NULL )
BEGIN
    INSERT INTO METEOROLOGICSTATION(NAME) VALUES ('Station') returning ID into :new.METEOROLOGICSTA
end;
```

```
DROP TRIGGER trgCreateMeteorologicStation;
```

trgCreateFieldRecording

This triggers auto creates the field recording of an exploration upon its creation

```
CREATE OR REPLACE TRIGGER trgCreateFieldRecording
    AFTER INSERT
    ON EXPLORATION
    FOR EACH ROW
BEGIN
    INSERT INTO FIELDRECORDING VALUES (:NEW.ID);
end;
```

```
DROP TRIGGER trgCreateFieldRecording;
```

trgFindFieldRecording

This triggers guarantees that an harvest is connected to the correct field report

```
CREATE OR REPLACE TRIGGER trgFindFieldRecording
    BEFORE INSERT
    ON HARVEST
    FOR EACH ROW
    WHEN ( new.FIELDRECORDING IS NULL )

BEGIN
    SELECT EXPLORATION into :new.FIELDRECORDING FROM SECTOR WHERE ID = :new.SECTOR;
end;
```

```
DROP TRIGGER trgFindFieldRecording;
```

Views

ClientView

This view has the objective of presenting all the information about all the clients in a convenient way. To calculate the number of orders that have been payed by each client, a sub-query that counts every entry on the orders by client, is payed and whose date is after 365 days in the past was used. To calculate the number of orders still awaiting payment, but delivered, a sub-query that counts every entry on the orders by client, is not payed and is delivered.

```
CREATE OR REPLACE VIEW ClientView AS
SELECT ID                                                          AS "Client's ID",
       NAME                                                        AS "Client's Name",
       PRIORITYLEVEL                                               AS "Client Level",
       COALESCE(TO_CHAR(LASTINCIDENTDATE), 'No incidents to date') AS "Reported Incidents",
       (SELECT count(*)
        FROM BASKETORDER B
        WHERE CParent.ID = B.CLIENT
          AND B.PAYED = 'Y'
          AND B.ORDERDATE > SYSDATE - 365)                         AS "Number of payed orders",
       (SELECT count(*)
        FROM BASKETORDER
        WHERE CLIENT = CParent.ID
          AND STATUS = 'DELIVERED'
          AND PAYED = 'N')                                         AS "Number of orders awaiting pa
FROM CLIENT CParent;
```

```
DROP VIEW CLIENTVIEW;
```