Database Systems – Japanese Fruit Agency

Japanese Fruit Agency

A national Fruit Agency would implement a system for storing, maintaining, and monitoring at runtime automatically with several sensors the information of fruits in supermarkets. The system will analyze different types of fruits with edible peel, such as apples, pears, strawberries, or not, such as bananas, kiwis, and oranges.

For each fruit, the system could suggest a low, medium, and high-budget recipe. Each fruit is described through other characteristics such as the name, type, DateTime of arrival at the supermarket, weight, volume, size, hours since it arrived at the supermarket, and ripens level, price.

The price is determined as a function of the ripens level, weight, and name. When the fruit has expired then the system automatically labels the fruit as no longer consumable and marketable. For each fruit, a table of stationary maximum time is provided to guarantee freshness. Each fruit is linked to possible allergies with symptoms.

The operator handles fruit with different activities. He/She is able to put fruit in a bowl, on display for sale, and remove them when necessary, while a client could only remove fruit buying it.

The sensors of the system are different: sensors for analyzing little, medium and big fruit. Each sensor is described via the name, medium energy consumption, cost, and brand.

Solution Architecture

The proposed solution architecture is composed by 3 components:

- The database
- A frontend service
- A backend service

This architecture takes the name of three-tier architecture, and is the most prominent in today's industry.

With this solution, the presentation tier, the logical tier and the data tier are separated both logically and physically.

Database

Conceptual Design

1. Choose the right level of abstraction

- The “type” of a fruit refers to the type of its peel (EDIBLE, NOT EDIBLE)
- Ripens level is a number ranging from 0 to 1, where 0 is “unripe” and 1 is “rotten”
- The price is based on the name of fruit (some fruits are more expensive than others), the total weight of the fruit considered and its ripens level; ripens level near to 0.5 will be more expensive than those near 0 or 1
- The table of stationary maximum time associates to each fruit a number of hours after which the fruit is considered expired
- An operator can put or remove a fruit from display
- The budget of a recipe is its cost
- An analysis of a sensor on a fruit will produce a textual description

2. Identify synonyms and homonyms

-

3. Standardize the sentences

- For each fruit, we hold its name, its peel, type, its size, its maximum stationary time, its time of arrival, its weight, its volume, its ripens level, its price, if it is on display or expired, the recipes which is involved in and the allergies it can cause.
- For each allergy, we hold its name and symptoms
- For each recipe, we hold its name and its cost
- For each sensor, we hold its name, medium energy consumption, cost and brand
- For each analysis, we hold the time it has been conducted and the textual description
- For each operator, we hold its name

4. Glossary

Term	Description	Synonym	Link
Fruit	The core of the system, represent the items to store and mantain		Allergy Recipe Sensor Operator
Recipe	Recipe that use the fruits registered in the system		Fruit
Allergy	A disease caused by contact of ingestion of one or more fruit, which is manifested through some symptoms		Fruit
Sensor	A piece of equipment used to analize fruits		Fruit, Analysis
Analysis	The textual description resulted from the analysis of a fruit with a sensor		Fruit, Sensor
Operator	The person who use the system and that can operate on fruits		Fruit

5. Reorganizing for Keywords phrases

- **General Sentences**

  A national Fruit Agency would implement a system for storing, maintaining, and monitoring at runtime automatically with several sensors the information of fruits in supermarkets.

- **Sentences referred to fruit**

  The system will analyze different types of fruits with edible peel, such as apples, pears, strawberries, or not, such as bananas, kiwis, and oranges.

  Each fruit is described through other characteristics such as the name, type, DateTime of arrival at the supermarket, weight, volume, size, hours since it arrived at the supermarket, and ripens level, price.

  The price is determined as a function of the ripens level, weight, and name.

  When the fruit has expired then the system automatically labels the fruit as no longer consumable and marketable.

  For each fruit, a table of stationary maximum time is provided to guarantee freshness.

- **Sentences referred to recipe**

  For each fruit, the system could suggest a low, medium, and high-budget recipe.

- **Sentences referred to allergy**

  Each fruit is linked to possible allergies with symptoms.

- **Sentences referred to sensor**

  The sensors of the system are different: sensors for analyzing little, medium and big fruit.

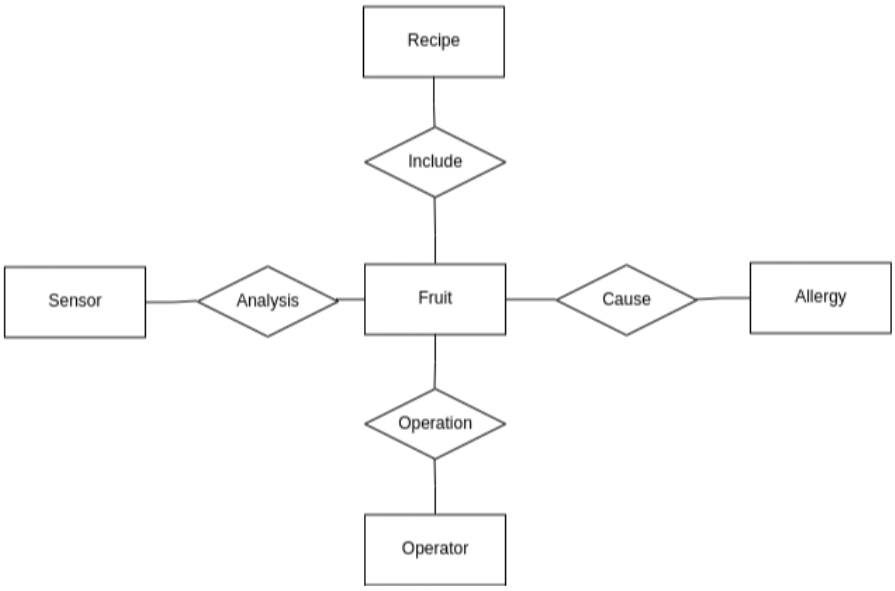
  Each sensor is described via the name, medium energy consumption, cost, and brand.

- **Sentences referred to operator**

  The operator handles fruit with different activities. He/She is able to put fruit in a bowl, on display for sale, and remove them when necessary, while a client could only remove fruit buying it.

6. ER Design

Because the fruit is the central item of the data system, a skeleton of the system is proposed, and then it is detailed starting from the fruit entity.



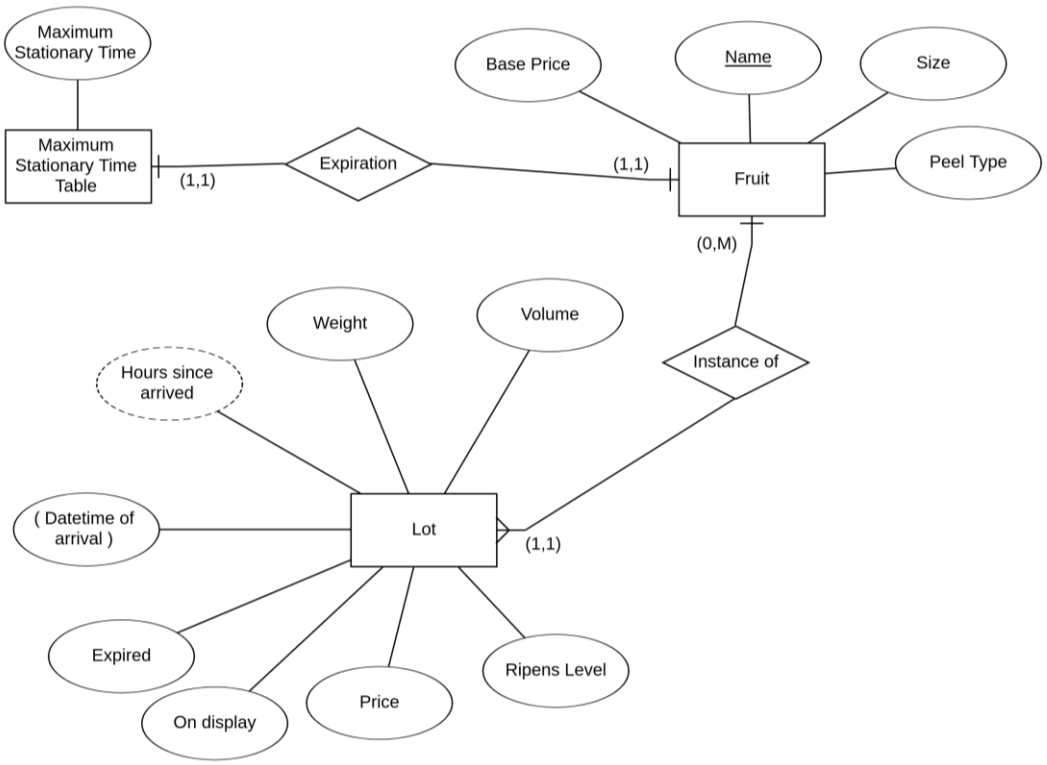
Because a lot of information refer to a fruit in general (bananas) and not a specific instance of a fruit (the 4kg of bananas arrived yesterday), it makes sense to split the fruit entity in two entities:

- Fruit: representing the generic fruit, and holding the information appliable to all of the instances of that fruit
- Lot: representing a specific lot of a type of fruit.

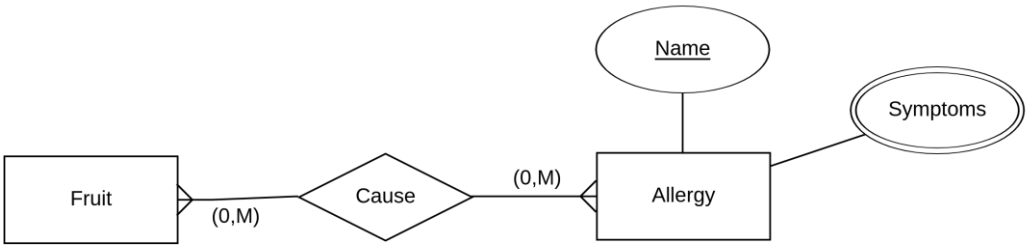
There can be more lot of the same fruit (a lot of oranges arrived yesterday and one arrived today).

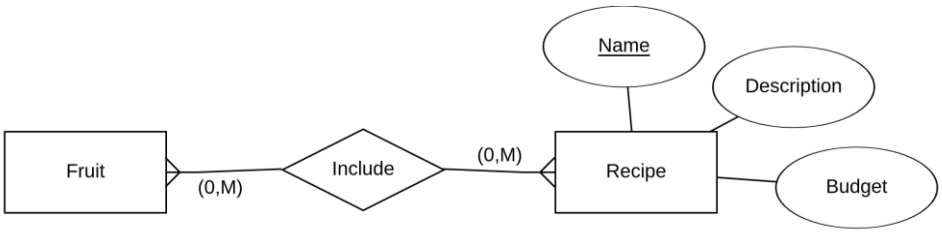
This is a fundamental choice: if we kept everything in a single relation we would introduce a lot of data redundancy (for example the information about peel type), along with anomalies and the need for tricks to do the simplest operations (for example, to register more lots of the same fruit).

The fruit and the lot entity are connected by an instance_of relation.

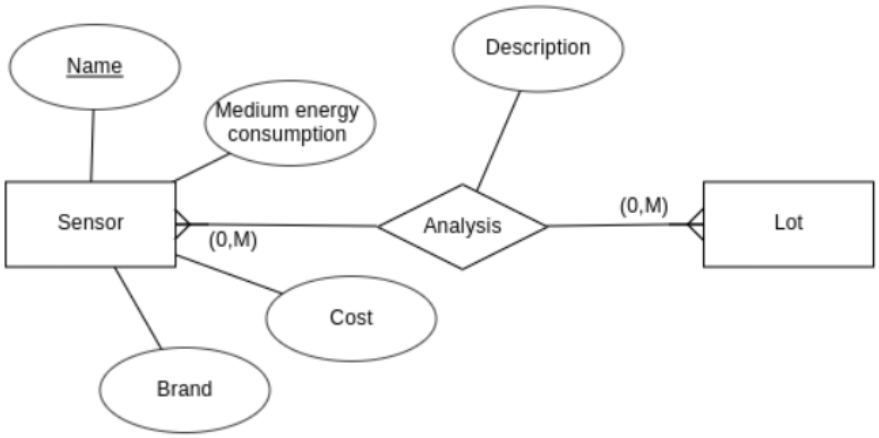

Maximum Stationary Time
Maximum Stationary Time Table
(1,1)
Expiration
(1,1)
Base Price
Name
Size
Fruit
Peel Type
(0,M)
Instance of
Weight
Volume
Hours since arrived
(Datetime of arrival)
Lot
(1,1)
Expired
On display
Price
Ripens Level

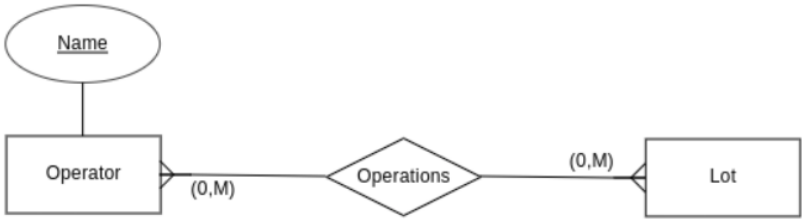
Then we detail the relationships of fruit with allergy and recipe



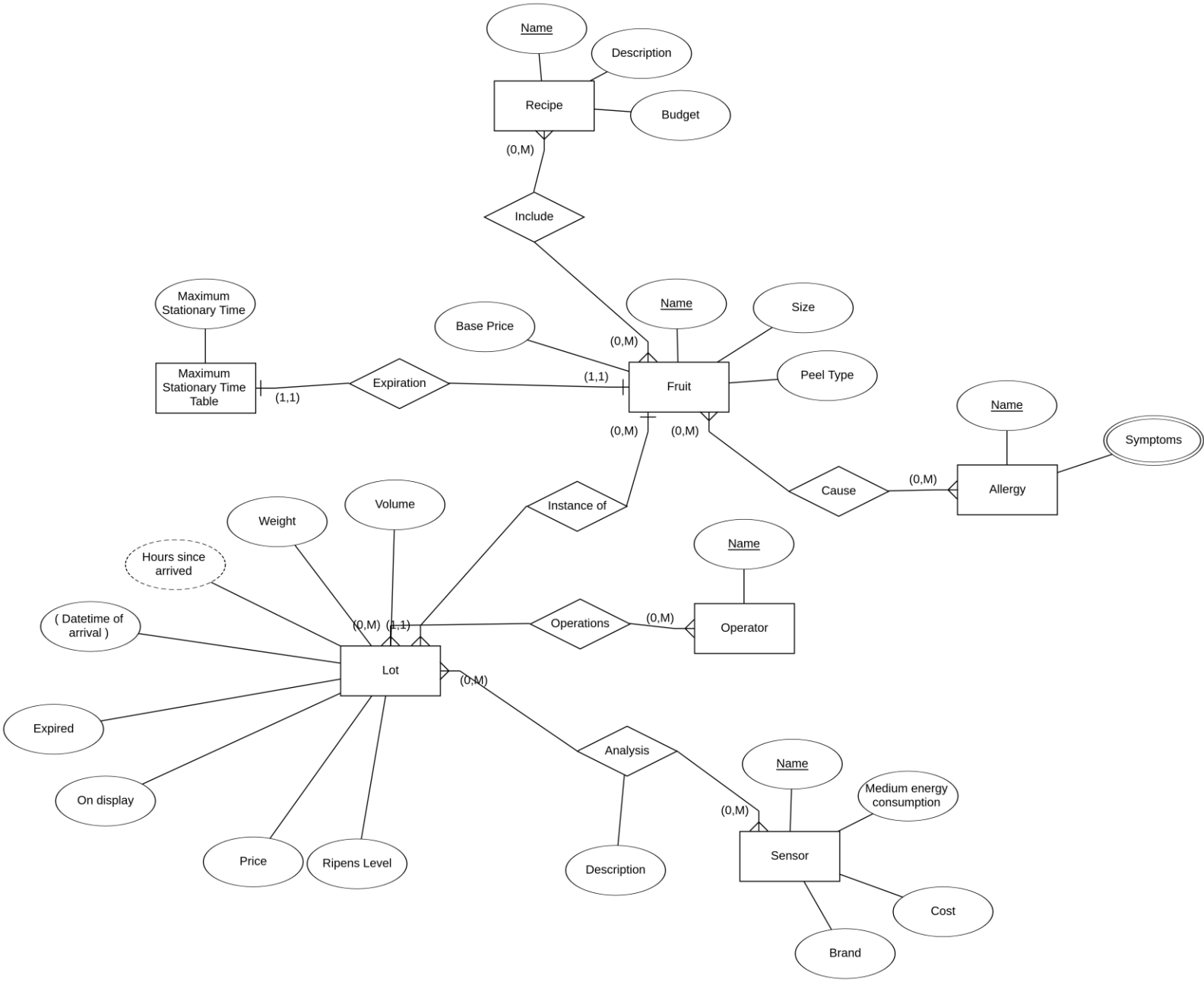


Lastly, we specify that a sensor analyzes a specific lot of fruits, and that an operator operates on a specific lot.





The final ER Diagram is the following:



Some constraints couldn’t be represented using the ER model.

These include:

- The fact that a sensor can analyze only fruits of a specific size
- The fact that the price is a calculated price and how it has to be calculated
- The fact that a fruit must expire when enough time is passed
- The fact that an operator cannot put on display an expired lot

Logical Design

1. Analysis of redundancies

- The attribute Hours since arrival of the entity Lot is redundant and it is difficult to mantain updated. For this reason, it is removed and will be calculated when needed, with the following specification:

  Hours since arrived = NOW – Datetime of Arrival

2. Removal of composite and multivalued attribute

- The composite attribute Datetime of Arrival of the entity Lot is kept since the chosen DBMS (ORACLE) supports Datetime attributes
- The multivalued attribute Symptoms of the entity Allergy is transformed into a textual field, where each value is separated by a comma, because there is no need to add detail for the symptoms.

3. Removal of hierarchies

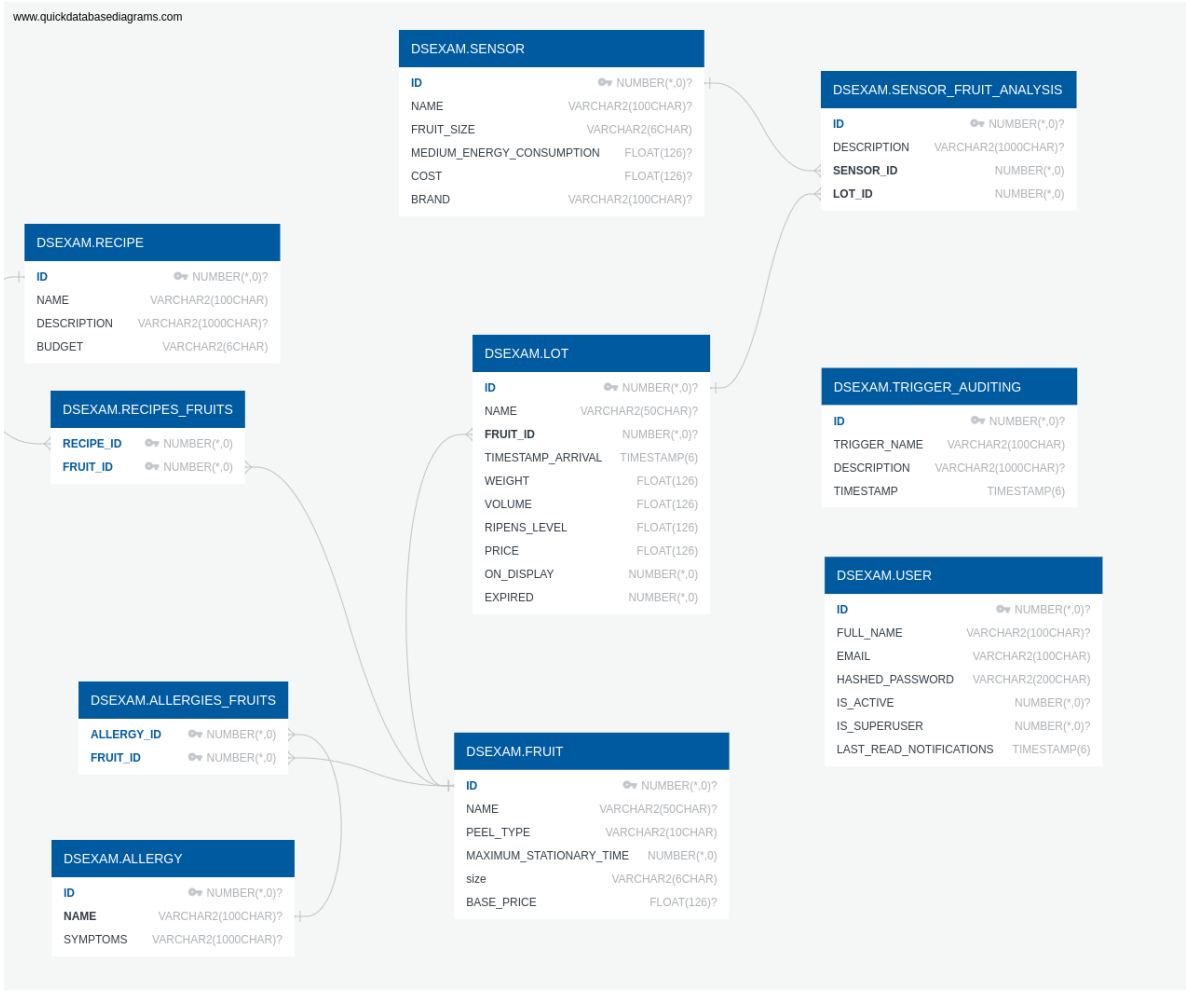
-

4. Partitioning/Merging of concepts

-

5. Choice of primary keys

For each entity, an id attribute is added. However, any constraint which was imposed by the keys is kept (e.g. for the entity Analysis, a unique constraint is imposed on both the sensor_id and the lot_id so that each sensor can analyze only one time each lot).


www.quickdatabasediagrams.com
DSEXAM.SENSOR
ID NUMBER(*,0)?
NAME VARCHAR2(100CHAR)?
FRUIT_SIZE VARCHAR2(6CHAR)
MEDIUM_ENERGY_CONSUMPTION FLOAT(126)?
COST FLOAT(126)?
BRAND VARCHAR2(100CHAR)?
DSEXAM.SENSOR_FRUIT_ANALYSIS
ID NUMBER(*,0)?
DESCRIPTION VARCHAR2(1000CHAR)?
SENSOR_ID NUMBER(*,0)
LOT_ID NUMBER(*,0)
DSEXAM.RECIPE
ID NUMBER(*,0)?
NAME VARCHAR2(100CHAR)
DESCRIPTION VARCHAR2(1000CHAR)?
BUDGET VARCHAR2(6CHAR)
DSEXAM.LOT
ID NUMBER(*,0)?
NAME VARCHAR2(50CHAR)?
FRUIT_ID NUMBER(*,0)?
TIMESTAMP_ARRIVAL TIMESTAMP(6)
WEIGHT FLOAT(126)
VOLUME FLOAT(126)
RIPENS_LEVEL FLOAT(126)
PRICE FLOAT(126)
ON_DISPLAY NUMBER(*,0)
EXPIRED NUMBER(*,0)
DSEXAM.TRIGGER_AUDITING
ID NUMBER(*,0)?
TRIGGER_NAME VARCHAR2(100CHAR)
DESCRIPTION VARCHAR2(1000CHAR)?
TIMESTAMP TIMESTAMP(6)
DSEXAM.RECIPES_FRUITS
RECIPE_ID NUMBER(*,0)
FRUIT_ID NUMBER(*,0)
DSEXAM.USER
ID NUMBER(*,0)?
FULL_NAME VARCHAR2(100CHAR)?
EMAIL VARCHAR2(100CHAR)
HASHED_PASSWORD VARCHAR2(200CHAR)
IS_ACTIVE NUMBER(*,0)?
IS_SUPERUSER NUMBER(*,0)?
LAST_READ_NOTIFICATIONS TIMESTAMP(6)
DSEXAM.ALLERGIES_FRUITS
ALLERGY_ID NUMBER(*,0)
FRUIT_ID NUMBER(*,0)
DSEXAM.FRUIT
ID NUMBER(*,0)?
NAME VARCHAR2(50CHAR)?
PEEL_TYPE VARCHAR2(10CHAR)
MAXIMUM_STATIONARY_TIME NUMBER(*,0)
size VARCHAR2(6CHAR)
BASE_PRICE FLOAT(126)?
DSEXAM.ALLERGY
ID NUMBER(*,0)?
NAME VARCHAR2(100CHAR)?
SYMPTOMS VARCHAR2(1000CHAR)?

The default schema name is DSEXAM.

The Operator table has been renamed in User and contain access data.

A table Trigger Auditing has been added to manage notifications to the users.

Physical Design

Indexes have been added for all of the primary keys.

Triggers

Several triggers are defined to implement those constraints that cannot be included into the ER model.

UPDATE_LOT_PRICE

```
CREATE OR REPLACE
TRIGGER UPDATE_LOT_PRICE
BEFORE INSERT OR UPDATE OF WEIGHT, RIPENS_LEVEL ON LOT
FOR EACH ROW
DECLARE
base_price NUMBER;
BEGIN
SELECT base_price INTO base_price FROM FRUIT WHERE :new.fruit_id = FRUIT.id;
:new.price := base_price * :new.weight * (1 - ABS(0.5 - :new.ripens_level));
INSERT INTO TRIGGER_AUDITING (TRIGGER_NAME, DESCRIPTION) VALUES (
  'UPDATE_LOT_PRICE',
  CONCAT(CONCAT(CONCAT('Run trigger UPDATE_LOT_PRICE on lot ', \:new.name), ' new price '),
:new.price));
END;
```

This trigger calculates the price of a lot each time:

- A lot is inserted
- The attributes weight or ripens_level of a lot are updated

The price is calculated based on the price, the ripens level and the fruit's base price.

It also adds a notification in the TRIGGER_AUDITING table.

UPDATE_LOT_PRICE_ON_BASE_PRICE_CHANGE

```
CREATE OR REPLACE TRIGGER UPDATE_LOT_PRICE_ON_BASE_PRICE_CHANGE
AFTER UPDATE OF base_price ON FRUIT
FOR EACH ROW
BEGIN
UPDATE LOT
    SET PRICE = :new.base_price * LOT.weight * (1 - ABS(0.5 - LOT.ripens_level))
    WHERE LOT.fruit_id = :new.id;
INSERT INTO TRIGGER_AUDITING (TRIGGER_NAME, DESCRIPTION)
    VALUES (
      'UPDATE_LOT_PRICE_ON_BASE_PRICE_CHANGE',
      CONCAT('Updated all prices for lots of ', :new.name)
    );
END;
```

This trigger updates the price of all corresponding lots when the base price of their fruit changes.

It also adds a notification in the TRIGGER_AUDITING table.

CHECK_SENSOR_TYPE

```
CREATE OR REPLACE TRIGGER CHECK_SENSOR_TYPE
BEFORE UPDATE OR INSERT ON SENSOR_FRUIT_ANALYSIS
FOR EACH ROW
DECLARE
fruit_id NUMBER;
lot_fruit_size VARCHAR2(10);
sensor_fruit_size VARCHAR2(10);
BEGIN
SELECT fruit_id INTO fruit_id FROM LOT WHERE LOT.id = :new.lot_id;
SELECT "size" INTO lot_fruit_size FROM FRUIT WHERE FRUIT.id = fruit_id;
SELECT fruit_size INTO sensor_fruit_size FROM SENSOR WHERE SENSOR.id = :new.sensor_id;
IF sensor_fruit_size != lot_fruit_size THEN
    raise_application_error(-20042, 'Sensor fruit size and fruit size mismatch');
END IF;
END;
```

This trigger raises an application error and stops the operation if there is an attempt to insert or update an analysis were the size of the fruit of the analyzed lot is different from the size of fruit that the sensor can analyze.

CHECK_SENSOR_TYPE

```
CREATE OR REPLACE TRIGGER CHECK_ON_DISPLAY_ON_EXPIRED_LOT
BEFORE UPDATE OR INSERT ON LOT
FOR EACH ROW
BEGIN
IF :new.expired = 1 AND :new.on_display = 1 THEN
    raise_application_error(-20042, 'Cannot put on display an expired lot');
END IF;
END;
```

This trigger raises an application error and stops the operation if there is an attempt to insert or update a lot, setting it on display when it is expired.

Jobs

CHECK_AND_UPDATE_LOT_EXPIRATION

To check for fruits expiration, an Oracle job is used, which is a program that is executed recurrently, after the specified interval.

```
CREATE OR REPLACE PROCEDURE CHECK_AND_UPDATE_LOT_EXPIRATION AS COUNT_UPDATED_LOT NUMBER;
BEGIN

SELECT COUNT(*) INTO COUNT_UPDATED_LOT
FROM LOT
INNER JOIN FRUIT ON FRUIT.ID = LOT.FRUIT_ID
WHERE (TIMESTAMP_ARRIVAL + FRUIT.MAXIMUM_STATIONARY_TIME * interval '1' hour) < CURRENT_DATE AND
LOT.EXPIRED != 1;

UPDATE
(
SELECT * FROM LOT
INNER JOIN FRUIT ON FRUIT.ID = LOT.FRUIT_ID
WHERE (TIMESTAMP_ARRIVAL + FRUIT.MAXIMUM_STATIONARY_TIME * interval '1' hour) < CURRENT_DATE AND
LOT.EXPIRED != 1
) T
SET T.EXPIRED = 1, T.ON_DISPLAY = 0;

IF COUNT_UPDATED_LOT > 0 THEN
    INSERT INTO TRIGGER_AUDITING (TRIGGER_NAME, DESCRIPTION) VALUES ('CHECK_AND_UPDATE_LOT_EXPIRATION',
'Update expired lots.');
END IF;
END;
```

This procedure checks all the lots present in the database.

If they are arrived by more than the number of hours specified on the corresponding fruit as MAXIMUM_STATIONARY_TIME, they are set as expired and if they were on display, they are removed.

Moreover, if and only if any lot has been updated, a notification in the TRIGGER_AUDITING table is added.

The following PL/SQL block is run to create a JOB which runs the procedure below every minute.

```
BEGIN
dbms_scheduler.create_job (
    job_name => 'CHECK_AND_UPDATE_LOT_EXPIRATION_JOB',
    job_type => 'PLSQL_BLOCK',
    job_action => 'CHECK_AND_UPDATE_LOT_EXPIRATION();',
    enabled => true,
    start_date        => SYSTIMESTAMP,
    repeat_interval => 'FREQ=MINUTELY;INTERVAL=1'
);
END;
```

Frontend

The frontend of the application has been written using Vue.js, which is a modern Javscript framework for the creation of web applications and Single Page Applications.

Backend

The backend of the application has been written in Python, using FastAPI as a web framework.

ORM and data versioning

SQLAlchemy as been used on the backend. It is an Object-Relational Mapper (ORM), which allows to map Database tables with Python classes.

This simplifies the integration between the application and the database, and makes more fast and reliable the development of CRUD operations.

A library named Alembic has been used as versioning and migration tool for the database.

Database versioning allows to version all of the DDL changes in the databae, and to manage the upgrades and downgrades of the databases in multiple environments.

Setup

The application has been developed in a fully containerazed environment, which allows to develop and deploy applications in a consistent way, across multiple environments, operating systems and development setups.

The containerization technology employed is Docker, and it is the only dependency required to be installed on the host machine to run the project.

To run the project, download the code and run the following command in the root folder:

```
docker-compose up
```

All of the necessary images will be downloaded and/or built (*this may take a while since the Oracle image is heavy).*

At the startup, the following services will be available:

- Backend: *localhost:8081*
- Frontend: *localhost:8080*
- Database: *localhost:1521*
- *CloudBeaver: localhost:8978 (CloudBeaver is a lightweight web-based SQL client)*

On the database, the user **DSEXAM** will be created and some data will be inserted.

Additional notes

All of the relevant code for the project is available at this link.

The backend and frontend projects are based on this boilerplate, which comes with the relevant software architecture, a user management system (login) and some basic views.