Database Systems

2020/2021

Documentation of sql create database script

Šesták Pavel(xsesta07)
Kroupa Dominik(xkroup12)

Brno, August 14, 2021

Contents

1 Assignment

Design an IS for a cheese store, which should allow to record the goods in the store and its warehouse. Further it should allow employees to order cheeses into store or warehouse from various suppliers. One order can contain several types of goods. Cheeses are categorized according to several criteria: country of origin, animal (cow, sheep, goat, yak, ...), type (Emmental, mold, smoked, ...). For cheese, it is necessary to record the fat percentage. Cheeses are delivered into store as loaves, where for each loaf it is necessary to record its initial weight, current weight for sale, delivery date, shelf life. Employees can also search for cheeses by country of origin, for which further information is available on cheeses from a given country, or according to its species or animal. Store employees can order different types of cheese from different suppliers (not all suppliers are offering the same cheese types), two orders can contain more types of cheese and for each type it is necessary to specify the quantity (weight). For delivered cheeses, it is necessary to be able to monitor which order it belongs to and from which supplier they are.

2 Entity relationship diagram

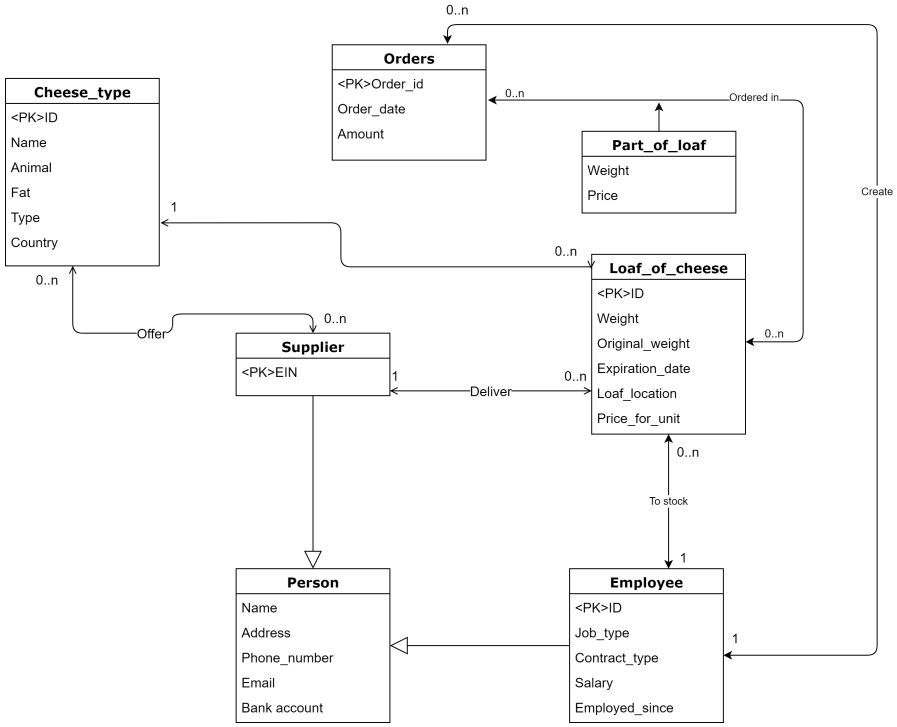

Figure 1: ERD diagram of IS

2.1 Text description of entity relationship diagram

The main entity of the data model is a loaf of cheese, where we store information about its original and current weight, expiration date and unit price. The location of the loaf represents whether the loaf is already in stock, in the store or at the supplier. Each loaf of cheese must be delivered by just one supplier and the loaf must have specific type. After delivery, just one warehouse worker must store it. In the process, the cheese can be moved to the store, where it is then sold out in parts. Employees are modeled as a specialized entity of a person. We keep information about the Employee regarding his position and whether he is a warehouse worker or a seller. Other stored employee informations are also the type of employment contract, payment and employee's entry into the company. Each employee is a salesman or warehouse worker. Each warehouse worker took over from the supplier 0..n loaves of cheese and each vendor created 0..n orders that consisted of 0..n pieces of cheese. About everyone in addition, we keep the weight and price of the cheese in the order. Another entity in the system is the supplier, which is also a specialized entity of the person. About the supplier in addition we store the EIN (Employer Identification Number). Each supplier offers 0..n types of cheese for sale and supplies 0..n specific loaves of cheese. The type of cheese is a separate entity. We keep the name of the cheese, its place of origin, from which animal stems, fat content and type. A specific type of cheese is supplied 0..n to the supplier, and within the company there are 0..n loaves of the given type. The constraints are 0..n because when creating a given entity, not all of its constraints may be known, or none may exist yet.

2.2 Use case diagram

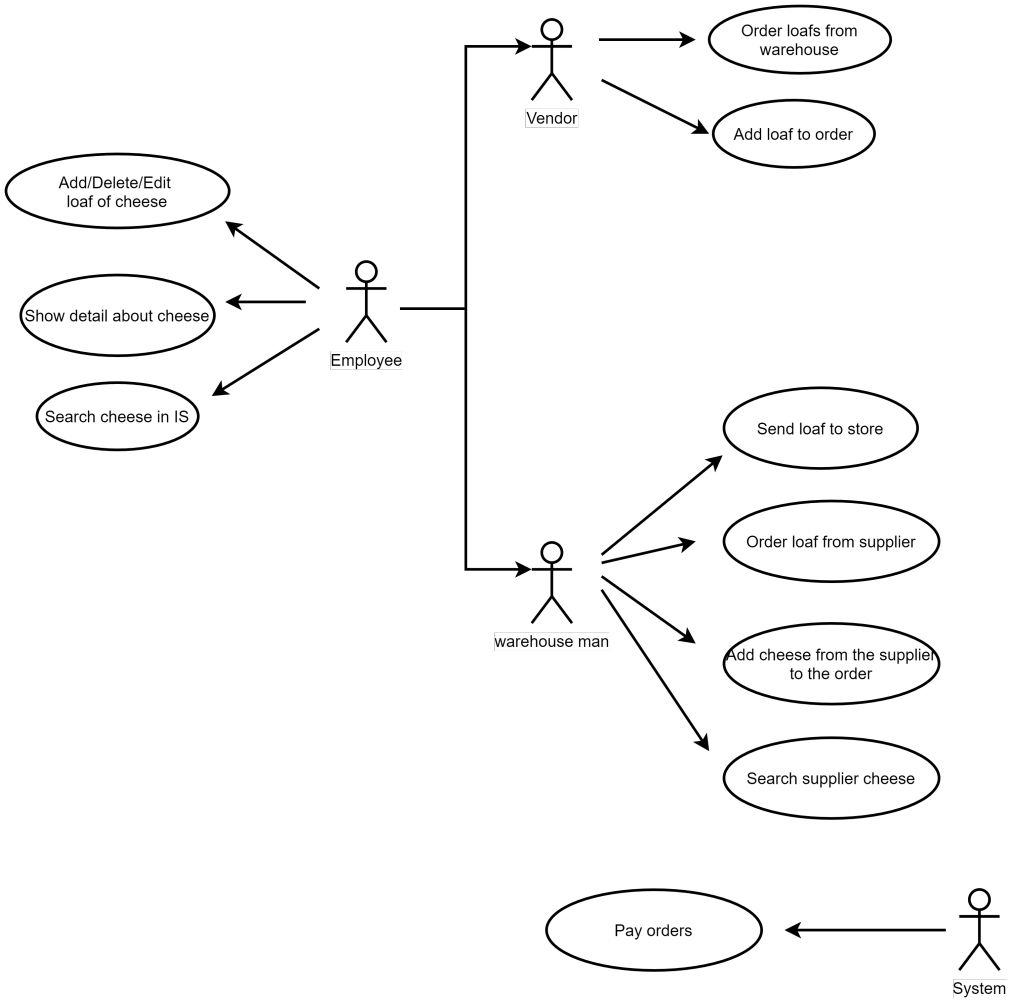

Figure 2: Use case diagram

3 Generating tables in SQL

3.1 Drop tables

Before creating tables in script, we need to drop tables. Oracle SQL doesn't support check if tables exist, this implies first launch of script ends with error: table doesn't exist.

3.2 Create tables

Each table is defined in sql:
CREATE TABLE table_name(
attribut_name$_1$ data_type$_1$ constrains$_1$,
attribut_name$_n$ data_type$_n$ constrains$_n$
);

$$n \in < 1, \infty)$$

After definition of basic attributes is list of references which represent relationships between tables:
FOREIGN KEY (key_name)
REFERENCES table (key_name_in_referenced_table)
Identifiers are set as GENERATED AS IDENTITY which means Oracle DB automatically generates unique id for each row.

3.2.1 Table Employee

In this table we check type of job if is shop or stock. With regular expression we can validate correct email address and phone number.
Phone number regular expression:

```
^(([+]|00)\d{3})?\d{9}$
```

Email regular expression: This regular expression must begin with + or 0 and three digits. This is optional part of phone number continued with mandatory sequence of nine digits.

```
^.+@.+[.][a-zA-Z]+$
```

This regular expression must begin with sequence of characters followed by at-sign. Next part is sequence of characters represent domain of email followed by dot-sign. Last part of regular expression is sequence of characters which represent top level domain.

4 SQL Insert queries

After we create tables we can fill it up. We didn't specify ID column, because id's are automatically generated by Oracle DB.
Each record is filled by command:

INSERT INTO table_name (property$_1$,property$_2$,..,property$_n$)
VALUES (value$_1$, value$_2$,...,value$_n$);

$$n \in < 1, \infty)$$

5 SQL Select queries

Select in script implement some basic queries to database.
Select basic syntax:

SELECT
table.property$_1$ AS alias$_1$
table.property$_n$ AS alias$_n$
FROM
table_name table_alias;

$$n \in < 1, \infty)$$

Select may be enriched with where clause or with join when we combine more tables in one query.

6 Triggers

Triggers are methods in PL/SQL which are executed on create or update or delete record in specified table. In our script we use triggers to automatic generate id and synchronize weight of loaf.

6.1 Trigger to automatic generate id

In case of inserting **employee** into database we check value of identifier. If identifier is null we assign next value from sequence **employee_id_seq**.

6.2 Trigger to synchronize weight of loaf

In case of insert or update **part_of_loaf** we update value of weight. In this method we use two variables: **Current_weight**, **Update_weight**. In case of insert :OLD.WEIGHT is null and we subtract new weight from current. Otherwise calculate **delta** of weights and subtract from **current weight**.

7 Explain plan – indexes and performance

Explain plan shows how our query will be processed in database. We select employees from table employee that created at least one order or stocked at least one loaf of cheese and are working for the company at least one year.

To get these informations, searching in EMPLOYEE table is not enough. The database also needs to search in tables LOAF_OF_CHEESE and ORDERS. Which can get a bit demanding on computing resources, as can be seen in first table in the picture below.

That is why we applied indexes to index properties in where clause (in this clause the tables LOAF_OF_CHEESE and ORDERS are accessed). After applying these two indexes we can improve CPU cost and lower the needed computing power.

```
PLAN_TABLE_OUTPUT
Plan hash value: 2375530189

-------------------------------------------------------------------------------------
| Id  | Operation          | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                |     3 |   816 |     6   (0)| 00:00:01 |
|*  1 |  FILTER            |                |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| EMPLOYEE       |     1 |   272 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| LOAF_OF_CHEESE |     1 |    13 |     3   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL| ORDERS         |     1 |    13 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( EXISTS (SELECT 0 FROM "LOAF_OF_CHEESE" "LOAF" WHERE
              "LOAF"."STOCKED_EMPLOYEE_ID"=:B1) OR  EXISTS (SELECT 0 FROM "ORDERS" "ORD"
              WHERE "ORD"."EMPLOYEE_ID"=:B2))
   2 - filter(SYSDATE@!-"EMP"."EMPLOYED_SINCE">365)
   3 - filter("LOAF"."STOCKED_EMPLOYEE_ID"=:B1)
   4 - filter("ORD"."EMPLOYEE_ID"=:B1)


PLAN_TABLE_OUTPUT
Index "INDEX_LOAF_OF_CHEESE_STOCKED_EMPLOYEE_ID" created.

Index "INDEX_ORDERS_EMPLOYEE_ID" created.

Plan hash value: 4001104745

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name                                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                                       |     3 |   816 |     4   (0)| 00:00:01 |
|*  1 |  FILTER            |                                       |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| EMPLOYEE                              |     1 |   272 |     3   (0)| 00:00:01 |
|*  3 |   INDEX RANGE SCAN | INDEX_LOAF_OF_CHEESE_STOCKED_EMPLOYEE_ID |     1 |    13 |     1   (0)| 00:00:01 |
|*  4 |   INDEX RANGE SCAN | INDEX_ORDERS_EMPLOYEE_ID              |     1 |    13 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( EXISTS (SELECT 0 FROM "LOAF_OF_CHEESE" "LOAF" WHERE "LOAF"."STOCKED_EMPLOYEE_ID"=:B1)
             OR  EXISTS (SELECT 0 FROM "ORDERS" "ORD" WHERE "ORD"."EMPLOYEE_ID"=:B2))
   2 - filter(SYSDATE@!-"EMP"."EMPLOYED_SINCE">365)
   3 - access("LOAF"."STOCKED_EMPLOYEE_ID"=:B1)
   4 - access("ORD"."EMPLOYEE_ID"=:B1)
```

Figure 3: Screen from oracle with explain plan

8 Materialized view

A materialized view is a database object that contains the results of a query. We create materialized view non expired loafs of cheese with information from cheese type. This will be probably most common query into database.

9 Procedure

Implemented procedure prints number of suppliers and employees stored in database.