Relational Databases - Foundations

Final Project

This relational databases projects will involve the skills learned during the foundation semester.

This project aims at designing a right database structure that corresponds to a real-world project, a parcel delivery management system.

This project is divided in 3 phases, as during the lectures:

1. Finding a right model to match the system needs in terms of data management,
2. Implementing this model using DDL, with PostgreSQL database.
3. Interact with data through SQL Queries.

# Finding the right model

In this section, you must find the right model that corresponds to the system needs. The expected deliverable for this part is **commented ER diagram (using crowfoot)** giving the logical structure of your data. To achieve this, you will have some specifications, from which you’ll have to extract several facts, helpful to consolidate your model.

**Pitch:**

You are a software engineer involved in a team which has no special skill in databases management, so you’re the only one able to deal with the database creation and the data engineering in general! Your company just received a business opportunity to create an application to track parcel for a big DIY stores chain. It will be able to manage special orders from the customers, and to help in dispatching parcels between stores, warehouses, transporters, and providers.

Your team gets the specifications from the functional manager, here is the summary you get from your technical manager:

1. The application will propose to manually create an order, directly in the store, when a customer wants to order a DIY article, not immediately available in the store.
2. An order is constituted of the article reference, the quantity, the unit price, the category of the article, a provider, a date (and time) of creation, a state, an order completion date.
3. Each order is linked to a single customer which has a name, an address, a birthdate and customer number.
4. Each order is to be delivered in the customer’s preferred store.
5. There are 13 article categories :
6. Garden
7. Tools
8. Bathroom
9. Kitchen
10. Storage / arrangement
11. Carpentry
12. Construction materials
13. Tiles & parquet
14. Decoration, Lighting
15. Drugstore & painting
16. Electricity & home automation
17. Heating & Plumbing
18. Hardware
19. Each order is then packed into a parcel, which has a number, a current state, a current location, a current transporter.
20. It is possible to have several transporters between the provider warehouse and the store.
21. It is necessary to trace all the successive states of a parcel, to be able to give information to the customer.

### Deliverables

1. Find the best E-R model to represent the above description. In case of missing information, make a proposal and justify your choices. **You have to provide a crowfoot notation diagram**
2. Once this ER diagram is made, make some comments, and design the physical model (e.g. the tables necessary to implement concretely this database). You have to provide a DDL diagram (same as crowfoot, but with indication of primary and foreign keys, and the final table names).

# Creating the database

## DDL

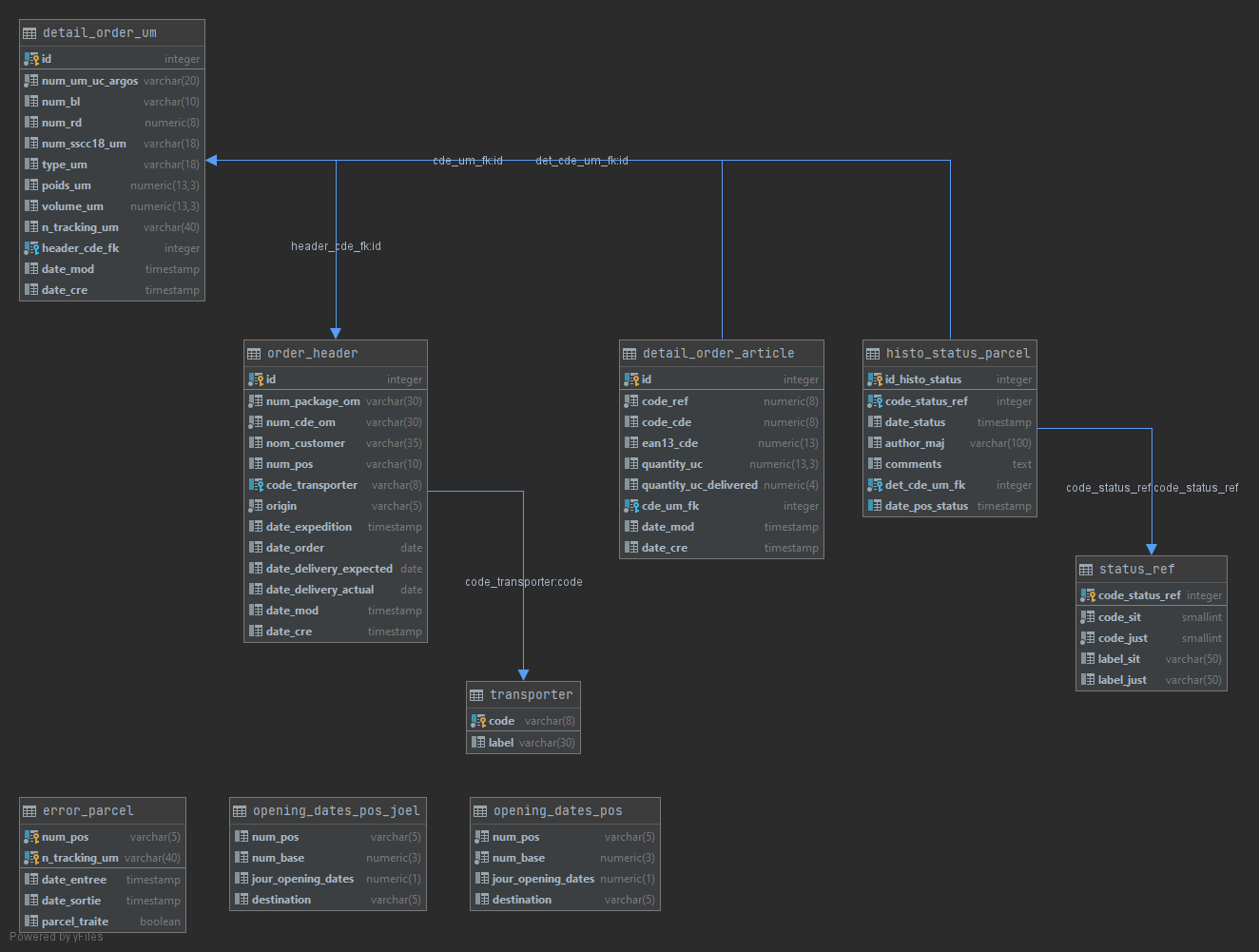
In this phase, create a database and implement the previous found physical model using SQL queries. Insert as much “reference data” you can. Reference data are the data which will not be affected by intensive modification. The DIY articles categories list is a good example of reference data.

### Your deliverable:

* Your DDL SQL Script
* Your reference data insertion scripts.

## Migrate from previous application database

Create another schema in the same database from the previous application databases scripts (attached). You will have existing data to transform using sql keywords (ALTER, UPDATE). Here is the physical model of the previous application database:



Your deliverables:

* Your migration SQL script.

# Query the new database.

For this part, if you did not manage the end of the second part, you can import and take data from the old database to design the following queries.

Providing the following use cases, your goal will be to implement the sql queries allowing to handle data as expected by the application.

## List of all the arrived orders

* 1. Give the total number of arrived orders in a Point of Sale
  2. Give the total number of arrived orders for one specific customer

## Customer order pick-up

* 1. Build a search query that will be able to look for customer orders by date, name, or article (like in a simple search query : [demo](https://codepen.io/huange/pen/rbqsD))
  2. Build a search query that can take at least 2 of the parameters cited in the previous question, leaving the other search parameters optional (like in an advanced search : [demo](https://www.google.com/search?q=advanced+search+form&rlz=1C1FKPE_frFR929FR929&sxsrf=ALeKk03SbzilSQqs0TdG0bhK8kPdfgXJgw:1613660934551&tbm=isch&source=iu&ictx=1&fir=-aoj9sDulZWOJM%252CQ4_F6gR4MrCxrM%252C_&vet=1&usg=AI4_-kTR6PMqh84Whnwyco3Tg7YdGFT9Bw&sa=X&ved=2ahUKEwjisaq22_PuAhWHy4UKHd0YASQQ9QF6BAgKEAE#imgrc=-aoj9sDulZWOJM))
  3. For a specific customer, display the list of orders with the following information :
     + Parcel number
     + Order number
     + Customer name
     + Last Status Date (arrival date)
     + Parcel Status
  4. For a selected order : update the order with a text comment and a status with a value option among :
     + Delivered to customer.
     + Not delivered, to customer wants an item exchange.
     + Not delivered, the customer wants a refund.

## Orders that have not been picked-up

1. Find all the unclaimed orders, although they arrived in the PoS. Restrict the list for orders arrivals older than 30 days.
2. Calculate the 3 categories the most affected by unclaimed orders
3. Calculate the overall cost of unclaimed orders

### Deliverables

For this part, you should deliver your queries **and** their associated results as follows in a word (or equivalent) document.

|  |
| --- |
| -- 1.a total number of …  **select \* from …**  **<screenshot of your result>**  Add some comments if necessary |