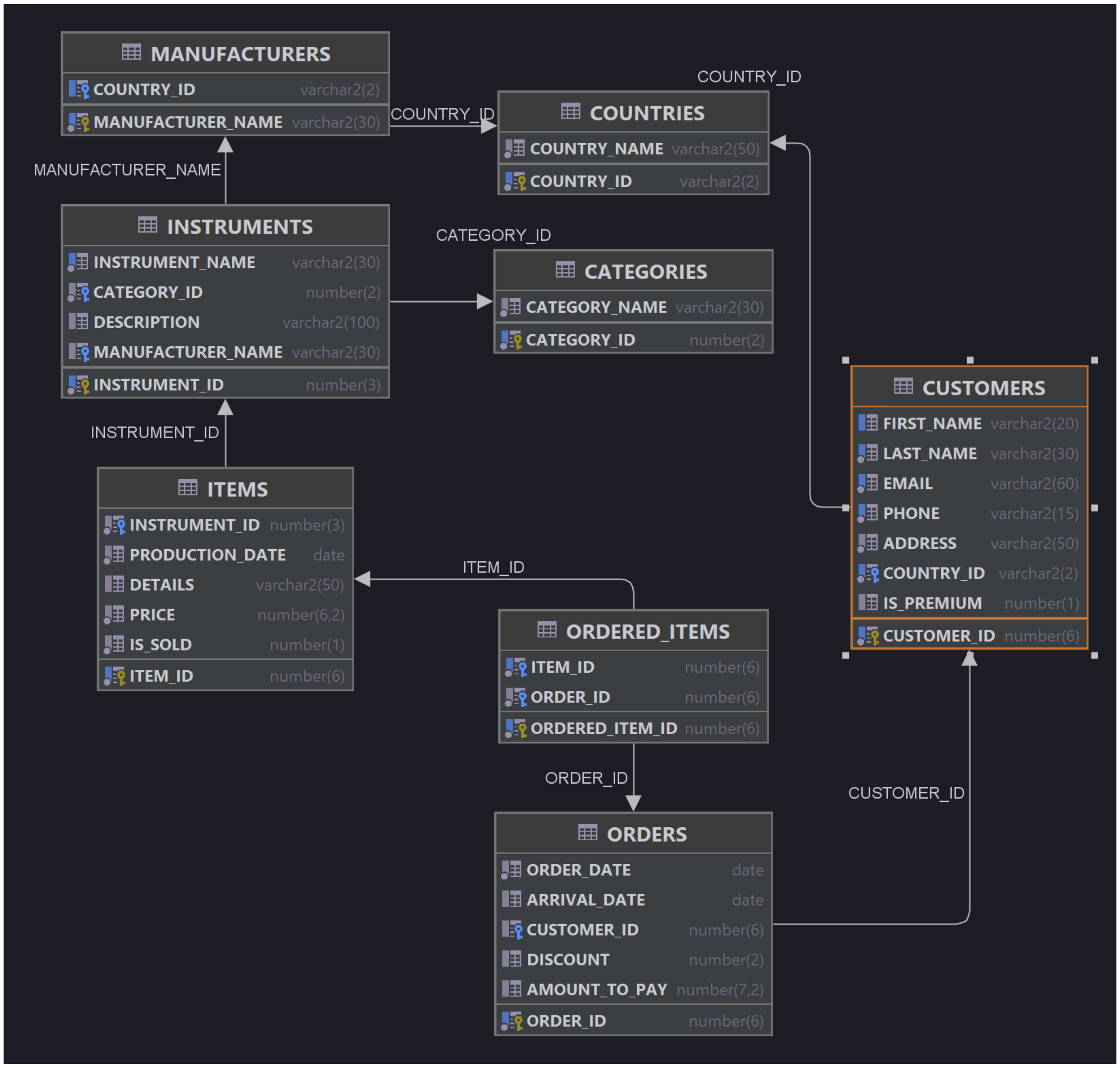
Database for musical instruments store

Ungureanu Anda

The database

In this project I created and populated a relational database for a store that sells musical instruments using Oracle SQL and PL/SQL. I designed 8 tables (with fitting constraints and triggers) and wrote 3 procedures. Their results will be represented through a Python application.

The diagram and the tables' structure



Looking at this diagram you can see the names of the tables, the names chosen for their fields, their data types and also the relationships between them, established using primary (gold) and foreign (blue) keys.

The constraints

The '**not null**' constraints – I have used these for fields that I considered mandatory, for instance the email address of a customer, the phone number, the price of an item, etc.

The '**unique**' constraints – Used for the email and the phone number of a customer. Other data that is supposed to be unique is already a primary key.

The '**check**' constraints – I used this type of constraint in order to check if the inserted price for an item is always positive. Another use case was for the 'boolean' data type (is_sold and is_premium). In order to mimic that type I used a one-digit number and always checked if the inserted value is 0 or 1.

Triggers

PREMIUM_DISCOUNT_FOR_ORDER – Whenever a new row is inserted into the ORDERS table, I check if the customer that placed the order has a premium account. If so, a discount for the last order placed will be calculated based on the number of previous orders. For example, if the customer had placed 2 orders before he placed the last one, then his discount will be 2 * 2 = 4%. The calculated value will be updated in his last order.

WAS_SOLD – Whenever a new row is inserted into the ORDERED_ITEMS table, it means that that particular item from the ITEM table is not available anymore. The trigger changes the value of the is_sold field of that item to 1.

Procedures

GET_FAVORITE_MANUFACTURERS – Gives us a ranking of the first 5 'best-seller' manufacturers based on the customers' orders. I decided to do that by taking every customer and calculating from whom they ordered the most items. Then, for each manufacturer resulted earlier, I got how many customers chose that brand. The manufacturer with the biggest result will be number 1 in the ranking.

GET_FAVORITE_INSTRUMENTS – Gives us a ranking based on the times the same instrument has been bought by the same person. The person with the most instruments of the same type will be number 1 in the ranking.

GET_MOST_EXPENSIVE_ORDERS – This one is pretty straight forward. It is a procedure used for the calculation of the orders' prices based on the items that were bought, their value and also on the discount (for premium accounts). In the end, it gives us a ranking of the most expensive orders.

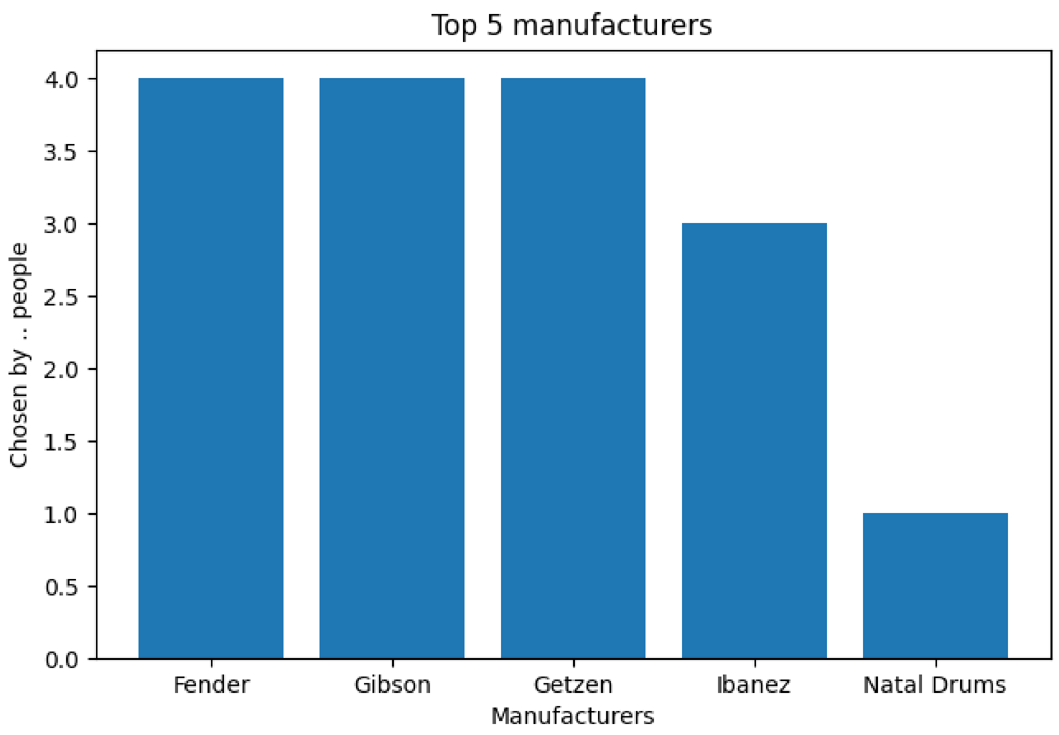
The application

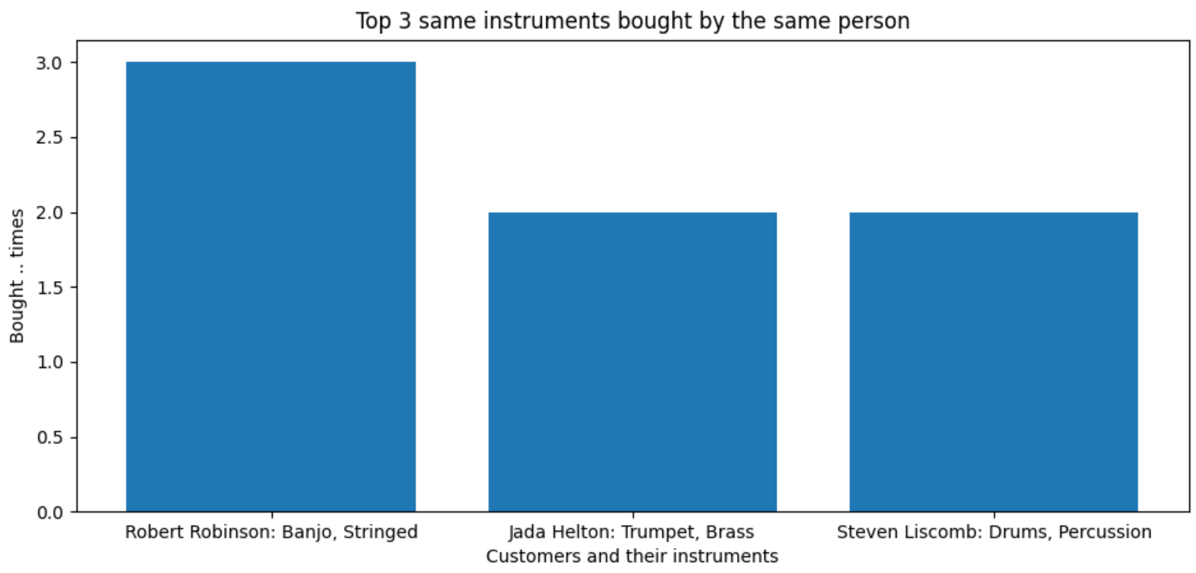
I wrote a simple application in Python. Its sole purpose is to connect to the database and provide me with graphic representations of the procedures described above. I have a single class defined called OracleConnection with 5 implemented methods: one for opening the connection to the database, one for closing it and one for the representation of each procedure.

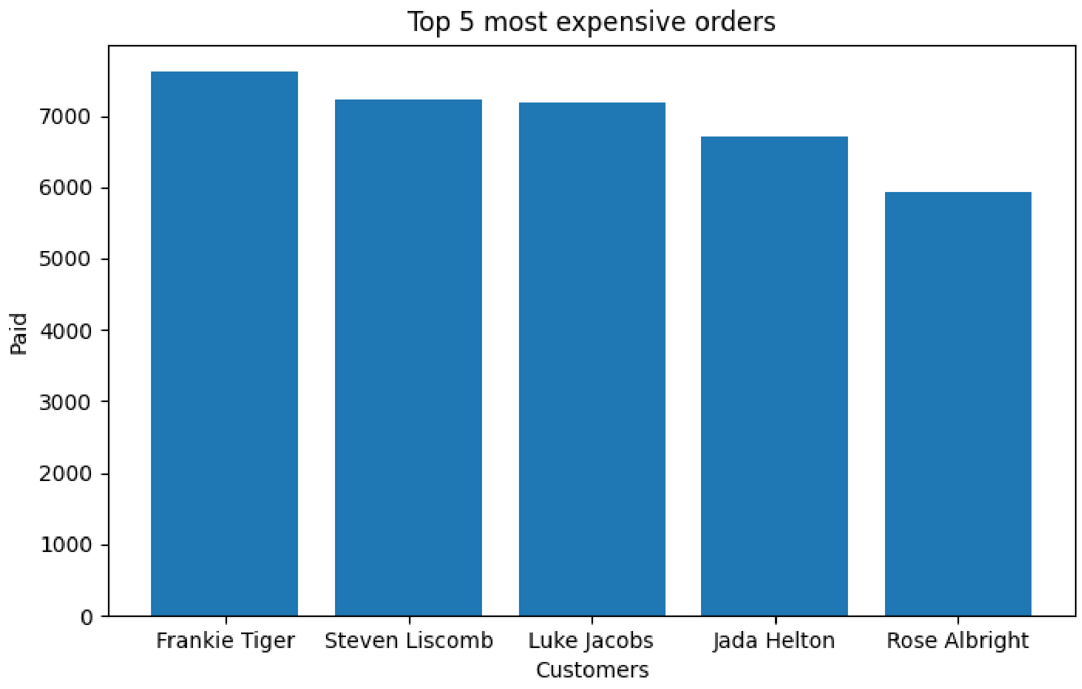
In order to connect to the database, I use the cx_Oracle library for mainly 2 functions: init_oracle_customer (so I can have the path to the Instant Customer) and connect.

The OracleConnection class will have all the attributes I need in order to build the input for the connect function: username, password, host, port and sid.

For the graphic representations I used matplotlib.pyplot and these are the results:




Top 3 same instruments bought by the same person
Bought .. times
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Robert Robinson: Banjo, Stringed
Jada Helton: Trumpet, Brass
Steven Liscomb: Drums, Percussion
Customers and their instruments


Top 5 most expensive orders
Paid
7000
6000
5000
4000
3000
2000
1000
0
Frankie Tiger
Steven Liscomb
Luke Jacobs
Jada Helton
Rose Albright
Customers