**MOBILE PHONE STORE**

Our store started selling phones in three locations around the world in October, 2023 - in the US (Apple phone store), in Vienna (Samsung phone store), and in Madrid (All Phone store - Samsung, Apple, and Huawei), planning to expand more in the US and Europe.

For all the tables that are likely to have a lot of rows, the primary key was an auto-incrementing BIGINT (CUSTOMER, CC\_PAYMENT, TICKET), because the auto-increment function increases simplicity, performance, and scalability, especially when there are many transactions. However, NUMSEQ was not set to be auto-incremented in order not to overcomplicate the task by resetting the auto-increment for every new ticket ID, but it could be potentially implemented with a trigger.

In the PRODUCT table, the ID is a VARCHAR(13) but limited to 12 or 13 characters in length (to adapt to different barcode standards around the world).

In other tables, where much fewer data would have to be added (and less often) and the primary key is referred to in another table, the primary key is not auto-incremented to simplify further insert statements, and it was set to only allow positive values.

**Table description (in the order of creation):***\*The most important entities are highlighted in yellow*

**Stores-related tables:**

* **ADDRESS\_TYPE**: Stores types of addresses (e.g., street, avenue, etc.).
  + ADDRESS\_TYPE\_ID was set to CHAR(1) because there is a limited number of address types.
* **POSTAL\_CODE**: Contains postal codes and their associated geographical details.
  + POSTAL\_CODE VARCHAR(10) accommodates global postal code formats including letters and special characters.
* **ADDRESS**: Holds address information for various stores.
  + We use SMALLINT for NUMBER, FLOOR, DOOR considering the usual size range of these attributes.
* **LOCATION\_TYPE**: Categorizes types of stores (e.g., Samsung store, Apple store, etc.).
  + TYPE\_ID CHAR(2) presumes a small, fixed set of location types which is flexible enough for different kinds of phone stores.
* **LOCATION**: links physical locations (through a GLN number) to addresses and their types.
  + LOCATION\_ID CHAR(13) is designed for GLN numbers which are always 13 digits.
* **STORE**: Represents individual store locations.
  + OPENING\_DATE ensures stores' opening dates are not set before Oct. 1st 2023, the official date of opening our first store.
* **CURRENCY**: Lists different types of currencies with their codes.
  + CURRENCY\_ID CHAR(3) follows standard three-letter currency codes.
* **SALESPERSON**: Contains information about salespeople.
  + BIRTH\_DATE and HIRE\_DATE checks ensure realistic hiring and age ranges (e.g., we cannot hire very old people and children less than 14 years old).
  + SALARY is ensured to not take negative numbers.
* **SALESPERSON\_BONUS**: Tracks bonuses awarded to salespeople in the respective currency based on their sales for the previous month - salespeople receive a 5% bonus from phone sales as a way of encouraging sales (ONE-TO-MANY RELATIONSHIP)
  + BONUS\_DATE ensures bonuses are recorded not before November 1st 2023, the first time salespeople can receive their bonuses.

**Product-related tables:**

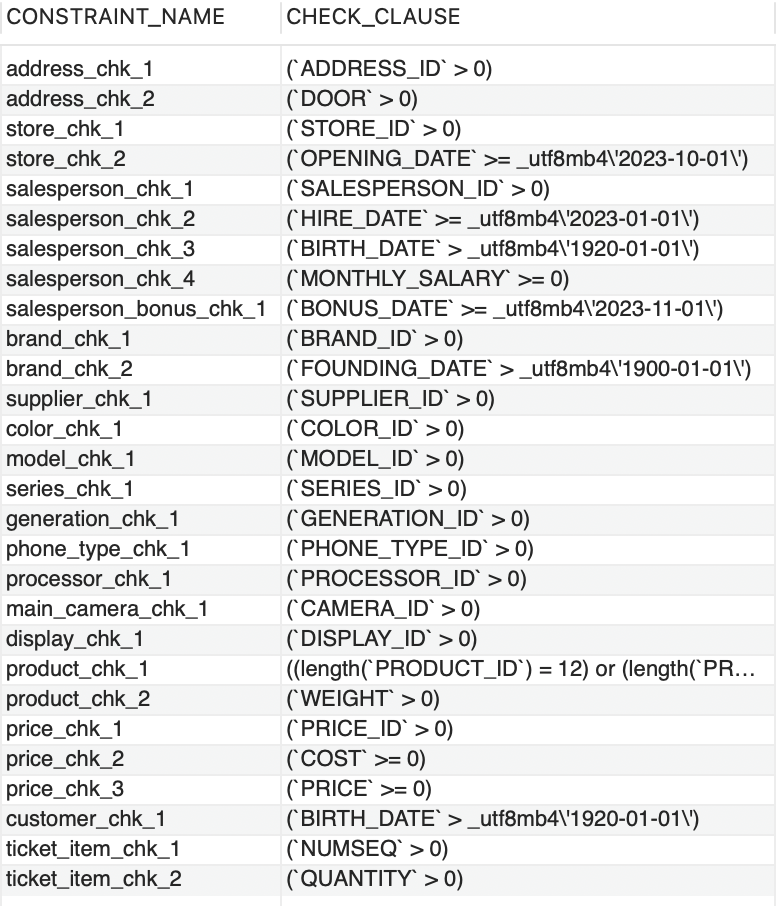
* **BRAND**: Stores information about product brands.
  + Justification: FOUNDING\_DATE to not record brands founded before 1900, considering historical relevance.
* **SUPPLIER**: Details suppliers of products.
  + In our stores, the same kind of phone can only be supplied by one supplier, but a given supplier can supply multiple kinds of phones (ONE-TO-MANY RELATIONSHIP)
* **COLOR**: Lists available colors for products.
  + Justification: COLOR\_ID as a BIGINT might be excessive for the number of colors.
* **MODEL**: Represents different models of flagships of products (Galaxy, iPhone, etc.).
* **SERIES**: Represents series within product lines.
  + Useful for products with multiple series under a single brand (S or A series for Samsung phones).
* **GENERATION**: Defines generations of products (23 for Galaxy S or 15 for iPhone).
* **PHONE\_TYPE[[1]](#footnote-1)**: Categorizes types of phones (Ultra, Pro, Pro Max, etc.).
  + Justification: Designed for varied phone types, assuming limited categories.
* **PROCESSOR**: Contains information about different phone processors.
  + CPU\_CORES and CLOCK\_SPEED\_GHz are included as key performance indicators for processors.
* **MAIN\_CAMERA**: Details the main camera specifications in phones.
  + CAMERA\_MP for megapixels and VIDEO\_RESOLUTION are included because they are crucial for camera quality.
* **WEIGHT**: Classifies weight categories.
  + WEIGHT\_CODE and WEIGHT\_LABEL offer a standardized way to categorize product weight (grams and ounces, for example).
* **DISPLAY**: Stores display specifications of products.
  + Covers key aspects of displays like size in inches and cm, resolution, and screen type (e.g. AMOLED).
* **PRODUCT**: Central table for products, linking phones with various attributes.
  + Justification: PRODUCT\_ID accommodates barcodes of 12 or 13 digits.
* **PRICE**: Tracks cost and selling price of products.
  + Separate fields for cost and selling price in different currencies were added given the global nature of our stores and supply system.
  + The PRICE\_ID is used instead of the PRODUCT\_ID as the primary key because it is a ONE-TO-MANY relationship: the same kind of phone can have a different price in a different country (in terms of currency and amount) and in different stores (different amount, even if the currency is the same). That is why, the PRICE\_ID is used as a foreign key in the TICKET\_ITEM table to identify the product in that particular transaction and be able to connect it to exactly to the corresponding price in the store where it is bought.
* **GPC\_SEGMENT**: Storing the product segment(s) based on GPC standards.
  + Only one segment in our case.
* **GPC\_FAMILY**: Storing the product family/families based on GPC standards.
  + Only one family in our case.
* **GPC\_CLASS**: Storing the product class(es) based on GPC standards.
  + Only one class in our case.
* **GPC\_CATEGORY**: Storing the product brick(s) / category based on GPC standards.
  + Only one category in our case.
* **GPC\_CATEGORY\_ATTRIBUTES**: Storing the product category attributes based on GPC standards.
* **GPC\_PRODUCT\_ATTRIBUTES**: Connecting the attributes to the products because one attribute can be present in many phones, and each phone can have several attributes. No ATTRIBUTE\_VALUE table was created because there are no attribute values for mobile phones in the GPC.
* **INVENTORY**: Storing the inventory of every product in every store
  + STORE\_ID and PRODUCT\_ID are used as a composite primary key because the cannot have different inventories for the same product)

**Tables related to Payments and Purchases:**

* **CC\_TYPE**: Lists types of credit cards.
  + CC\_TYPE CHAR(2) assumes a concise range of credit card types (e.g., MC for MasterCard).
* **CC\_PAYMENT**: Contains credit card payment details of a purchase.
* **CUSTOMER**: Stores customer information - when customers buy a phone, they are offered to sign up to the store to receive updates.
  + PHONE\_NUMBER is mandatory to register to the system.
* **TICKET**: Represents sales transactions.
  + Includes all the fields related to sales transactions, including the STORE\_ID, SALESPERSON\_ID, CUSTOMER\_ID (if the customer agrees to sign up), PAYMENT\_ID, TOTAL\_AMOUNTS before and after tax, and CURRENCY of the purchase.
* **TICKET\_ITEM**: Details items within a ticket.
  + TICKET\_ID and NUMSEQ ensure ordered listing of items in a ticket when customers buy multiple phones or kinds of phones.
  + It refers to each project through its PRICE\_ID, stores the PRICE and the QUANTITY of the respective product purchased.

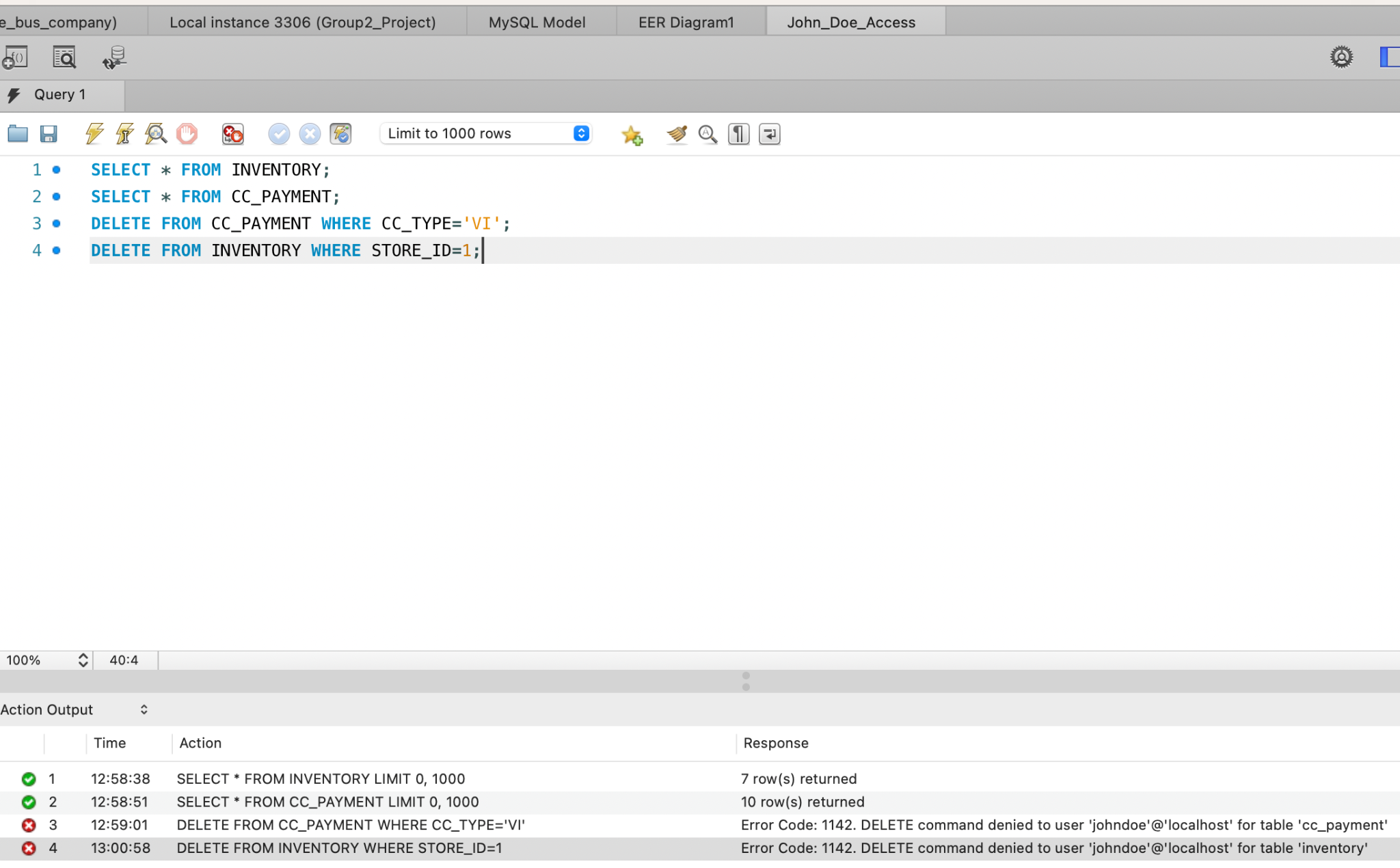
**Additional comments:**

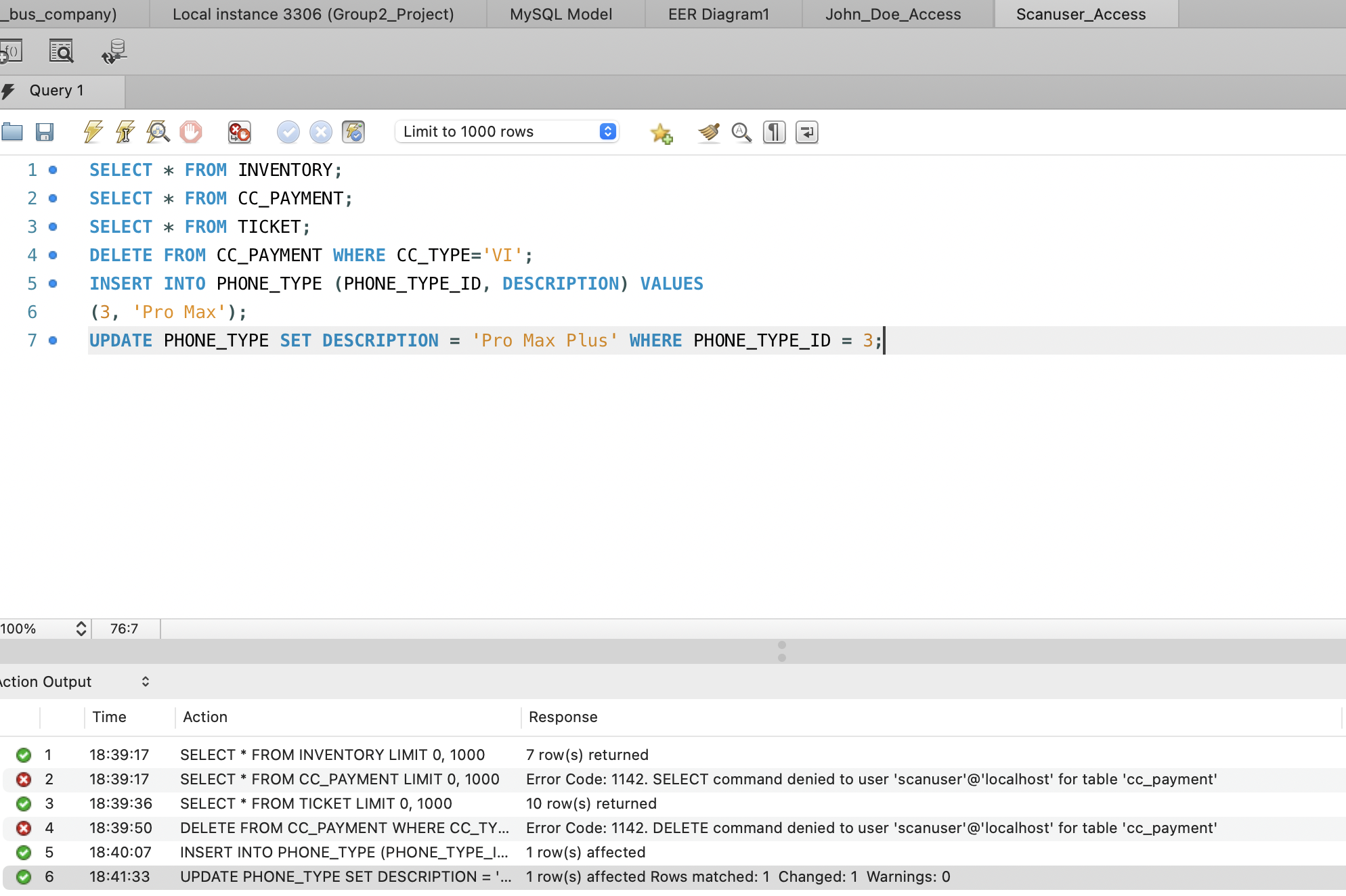
* The following constraints / validation rules were added to the different table attributes to reduce the number of possible errors if inserted manually:



* Some constraints to the date/timestamp attributes were added with triggers to take into account non-fixed current date/time and not allow future dates in certain fields (see DDL file)
* We do not want any data to be deleted from the system, so all foreign keys are ON DELETE RESTRICT by default.
* To simulate transaction inserts, LAST\_INSERT\_ID() was used to store the ID generated by the last INSERT statement in the current connection (which is session-specific and is not affected by other concurrent sessions opened by other users) - see SQL INSERTS file.[[2]](#footnote-2)
* For updating the inventory for the purchased product(s), we wanted to reduce the number of manually entered values to avoid mistakes and by performing the update through joining the necessary tables (see SQL INSERTS file).
* In our INSERT statements, we only added new customers if they wanted to sign up. In the future, when the same customers come back to the story to buy a new phone, the INSERT statements could be made more complex and flexible to look-up the customer first and insert a customer only if they are not already in the system.
* ***Control***: two users were defined in this model: **scanuser** and **johndoe.** Nobody is allowed to delete data from the system. The scan user can select, insert, and update every table but he cannot select any data from the CC\_PAYMENT table, since it contains credit card information and we don’t trust scan users. Only **johndoe** can view the data in every table, including the CC\_PAYMENT because he is the only person we trust.

The examples of test DML statements made in the respective sessions for **scanuser** and **johndoe** with the corresponding success/error messages can be found below:





1. Model, series, generation, and phone type are made nullable in the product table below because different companies have different elements in their phone names. [↑](#footnote-ref-1)
2. The series of INSERTS could be improved in the future with the help of MySQL procedures. [↑](#footnote-ref-2)