AppStore case study

The AppStore is a typical online store for mobile software applications (Apps). It sells Apps that belong to one of the available categories in which the AppStore specializes.

In particular, the available categories are distinguished by an indicative code (Cat_Name), a description of the category (Cat_Description) and the indication of the population group for which the category is indicated (Cat_Age_Group).

Apps are declared in the AppStore with a unique alphanumeric identifier (App_ID), the name of the App (App_Name), a description of it (App_Description), its creation date (App_Date), its current version (App_Version) and the number of App installations (App_Installations) that have been achieved through the online store.

When an App is registered in the online store, it is declared that it belongs / is classified in a single category from those defined in the online store.

The AppStore also registers:

- popular operating systems / platforms, for each of which there is an indicative (unique) name (OS_Name), model indication (Source_Model) and a website (Website)
- mobile device manufacturers (Vendor) that have a unique identifier (Vendor_Name), geographical location indication (Vendor_Location) and reference to its founder (Vendor_Founder)
- entities (Developer) that develop Apps for which their unique code (Dev_ID), name (Dev_Name), contact email (Dev_email) and address (Dev_Address) are recorded.

It should be noted that an app is designed and developed by a single entity that also has overall responsibility for the online store. Finally, an operating system/platform can be used by multiple Apps and mobile device manufacturers, while an App and a mobile device manufacturer can support multiple operating systems/platforms.

Assignment 1 (Functional dependencies and relational schema)

1A (Functional dependencies and relational schema): The first exercise aims to familiarize students with the design of relational schemas using functional dependencies. Specifically, you are asked to filter the utterance so that you identify (and record) the functional dependencies that you identify. Then, based on the functional dependencies, the data and scripts available (see eclass material) finalize the relational schema.

1B (Schema modifications): You are now informed that a series of modifications need to be implemented. The first concerns the support of ratings for the Apps of the online store. The data of the ratings that must be recorded are as follows. A rating concerns a single app and has a unique

code (Rating_ID), a value (App_Rating_Value) on a scale (0 to 5) and the date of the rating (App_Rating_Date).

The second requirement concerns the recording of consumers who use the AppStore to purchase Apps. In the current version of the AppStore, only certified users - customers of the manufacturers (Vendor) present in the database are considered consumers. For them, the AppStore must record a unique username, first name, last name, contact phone number and home address. This data is derived from services offered by the manufacturers (APIs) and which function as a form of third-party log in. The individual details should not concern you beyond the fact that your database must record what we mentioned above regardless of their origin.

Based on the above, you should first identify the new functional dependencies that arise from the new requirements and then you should update your database appropriately. Indicative data with Ratings of some Apps and a query that you can formulate in SQL for confirmation have been posted. For users - consumers, enter at least two tuples of your choice.

Assignment 2 (New data types)

2A (User-defined data types): In the AppStore database script (which you are working on) you need to implement the following modifications:

- Create a new data type S_Model_Type and use it appropriately to avoid the CHECK statement in the OPERATING_SYSTEM table
- Create a new data type 'Affiliation' and use it appropriately to separate DEVELOPER into external (E) and internal (I)
- Create a new composite data type Address with parameters Line VARCHAR(90) and Zip_Code VARCHAR(30) for the DEVELOPER addresses
- Convert the email in the DEVELOPER table to a multidimensional array so that it is possible to insert two types of email (e.g. institutional and personal) with appropriate encoding in the array
- Assume that each App has a Blog where messages about the App are posted (by DEVELOPERs or VENDORS of the AppStore). Then consider using a new complex data type MESSAGETYPE with parameters PERIEXOMENO VARCHAR(4000), HMEROMHNIA DATE and TAGS TEXT [] that allows posting messages to Blogs of the AppStore Apps
- Create a new complex data type RATING with parameters Rating_ID INT, App_ID TEXT, App_Rating_Value TEXT, App_Rating_Date TEXT to post the ratings that already exist in your database

2B (Querying): After completing the modifications, you should answer the following SQL queries (Note: You should be careful because depending on the modifications you make, some of the following queries may need to be changed. It is recommended that, as soon as some

modifications to the schema are completed, the corresponding queries be formulated and then continue):

- For all DEVELOPERS, print all personal emails that they declare
- Print the details of DEVELOPERS who declare a ZIP code that includes the characters 'CA' and have an affiliation 'E'
- Print the details of DEVELOPERS who do not declare a ZIP code
- Print the authors of the messages and the contents of the messages that have a tag of our choice (e.g., 'Messaging')
- Count the number of messages that include a tag of your choice (e.g. 'Messaging')
- Print the authors of the messages and the contents of the messages that include a snippet of your choice (e.g. 'Sales')
- Print the contents of the messages that do not include a tag of your choice, their author and the Blog
- Print the details of the APPs that have a rating value of 2 (the question must be answered in the modified script where RATING is a complex type)
- Print the details of APPs that have not been rated (the query must be answered in the modified script where RATING is a complex type)
- Print the APP_ID and the rating value of all APPs except those that have not been rated (the query must be answered in the modified script where RATING is a complex type)

Assignment 3 (JSON, specialization hierarchies)

The exercise focuses on modifying the AppStore script to incorporate JSON data and create a specialization hierarchy. Specifically, the following should be done:

3A (JSON data type): Modify the script so that the RATING table is defined as follows:

```
CREATE TABLE RATING (
Rating_ID serial NOT NULL PRIMARY KEY,
info JSON NOT NULL);
```

After updating the table with indicative data (from the already available data) so that it is compatible with the new definition, you will need to formulate SQL statements for the queries:

- Which Apps have at least one rating?
- For each App that has received a rating, find the rating code, date and rating value
- Find Apps that have ratings > 3
- For Apps that have ratings > 3, find the installations that have
- Find the operating systems that run Apps that have ratings > 3

3B (Inheritance): Use the following script (as a guide) to modify the version of AppStotre you previously implemented to support this specific specialization hierarchy.

```
CREATE TABLE CATEGORY (
Name TEXT NOT NULL,
PRIMARY KEY (Name));

CREATE TABLE HEALTH_AND_FITNESS (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE PRODUCTIVITY (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE TRAVEL_AND_LOCAL (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE COMMUNICATION (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE WEATHER (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE PHOTOGRAPHY (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE GAMES (
Description TEXT,
SUB_CATEGORIES TEXT [][][]
) INHERITS (CATEGORY);

CREATE TABLE GAME_SIMULATION (
Description TEXT,
SUB_CATEGORIES TEXT [][]
) INHERITS (CATEGORY);
```

```
CREATE TABLE GAME_ACTION (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE SOCIAL (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE ENTERTAINMENT (
Description TEXT
) INHERITS (CATEGORY);

CREATE TABLE ART_AND_DESIGN (
Description TEXT
) INHERITS (CATEGORY);
```

Where you encounter the SUB_CATEGORIES attribute that is of type array (simple or multi-dimensional), consider that it is used to register characteristics of each App in the form of pairs such as e.g. {"Field", "War affair"} and/or {"Period", "Second World War"}. After modifying some of your data appropriately, you should formulate the SQL statement that calculates the App_ID, the App name, the App manufacturer's name and the number of its installations for all Apps of a subcategory (e.g. GAME_SIMULATION) that are related to some characteristic of your choice (e.g. they have the Second World War as a reference period, which is formulated as {"Period", "Second World War"}.

Assignment 4 (Graphs in Postgres and recursive queries)

Exercise 4 has the dual goal of firstly familiarizing students with graph development and secondly to allow the analysis of alternative ways of representing graphs using relational technology. Progressively, this analysis will allow the understanding and practice of some advanced PostgreSQL techniques such as the recursive function for traversal and transitive closure of graphs. Students are still working on the current implementation of the AppStore database where the following will need to be implemented:

4A (Property graph in PostgreSQL): You will start by developing (initially on paper or in a tool) a property graph (see theory) that will capture the following:

- Follows relationships between developers who develop Apps (i.e. who follows whom, in the sense that the following is found in the Twitter service)
- Interest relationships (i.e. interest) of developers who develop Apps in operating systems (note that this relationship could be reflected in the current version of your implementation with the tags in the content of the messages that developers post on a wall)

- Ownership relationship (i.e. belongs_to) for developers posting messages on a wall.

The final version of the property graph that will be developed should definitely capture data that already exists in your database and any new data that you may want to add

Having designed the indicative property graph (with a small number of nodes so that at least two representations of each node are recorded), you should then represent the property graph using relational technology. It is at your discretion to choose a technique from those you have learned so far, i.e. the use of normalized relations, unnormalized relations, adjacency lists, etc., while you should study the way in which the alternative nodes and edges that will exist in your graph will be supported (see PostgreSQL inheritance mechanism).

4B (Recursive path queries): Having represented the property graph with relationships, you should become familiar with modern graph traversal techniques using recursive SQL calls. Specifically, in the current version of the AppStore database you have implemented, you should answer the following questions:

1. For each / one node calculate the next ones (in a direction of your choice)
2. For each / one node calculate the previous ones (in a direction of your choice)
3. For each / one node calculate the next ones regardless of direction
4. For each / one node calculate the previous ones regardless of direction
5. Calculate all possible paths connecting the listed developers in the property graph you implemented

Assignment 5 (NoSQL)

The last assignment requires you to choose a NoSQL system of your choice and review (a subset of) the AppStore database so that you gain hands-on experience with a system of a different type and philosophy than the one you used to develop the database. Your goal should not be to create yet another different implementation of the database in another computing environment (hence the reference to the subset above) but to transfer a portion of your data to a NoSQL environment so as to create a new data source complementary to PostgreSQL. The data set you choose should be realistic and reflect real-world conditions. For example, it could be a set of reviews and ratings for Apps from an open data set. Students who can present an illustrative scenario of utilizing the data from all the sources they create will receive an additional bonus during the final grading.