Reyhan Ayhan
09/07/16
Lab 1: PostgreSQL

1. Data vs. Information

A database used for a healthcare company contains data such as the first and last name, gender, phone number, date of birth, and address of specific recipients. The data organizes all the names, numbers, and addresses by using the appropriate identification fields (i.e. “First Name”, “Last Name”, “Date of Birth (mm/dd/yyyy)”). If, for example, the recipient’s date of birth without the slashes could be seen meaningless numbers or data. It isn’t until those numbers are clearly organized and identified as the date of birth of the recipient. In addition, if the database were to also include the individual’s current annual income but those numbers were not accompanied by the field “Annual Income” and/or “USD”, the numbers would just be data because there’s no context that would make it information. With a clear understanding of the information and not data without context, the healthcare company could remain organized and allow for precision and accuracy.

2. Data Models

The need for physical data independency, the ability to have hardware move from one system to another, led to the hierarchy model. The model has a pyramid-like structure separated between the top and bottom layers. In an example of a video game, the name of game would be the top entry with two nodes beneath it separating the two player entries. Both the “player 1” and “player 2” entry have two branches attached to them. At the end of those branches are entries for items that each player has and doesn’t have. Player 1 for example would have items “A” and “B” and player 2 would have “B” and “C”. These elements belong to the top layer of the model. The bottom layer holds the databases with the Information Management Systems (IMS). This model achieved physical data independency because now with Cobol programming, as a programmer you don’t know which system you’re working on.

Although the hierarchical model allowed for physical data independency, the model is still flawed. Following this model is the networking model. Referring back to the video game example, the networking model allows a user to create a separate entity that isn’t attached to the game entity and neither of the player entities by a node or branch. This entity could be identified as “shop” and this entity could include item “D” which we were unable to include using the hierarchical model because the game wasn’t meant to be designed in a way where every item needed to belong to a player. With this method, the shop entity could have a branch connected to C and D and could remove B from player 2 to get rid of any duplicates.

The relational model differs from both the hierarchical and networking models. Instead of all the other abstractions, a table may be a better, more advanced method. This table includes a collections of rows and columns and the tables relate to each other using keys. Working off of the video game example, three tables would have strong entities identified as players, items, and inventory. Each table contains clear identifiers and various entires. All columns in each table are uniquely named and this method also removes the need for duplicate items. The rows and columns ratio per table will vary depending on the information included. The relational model also allows for easy diagramming for cardinality, the relationship between the number of things belonging to the information within the tables. The relationship model was built more on the basis of set theory and relational algebra– all about mathematics.

I think XML could be useful but only for smaller volumes of data within a system. It could extract data its documents contain and serves well for Were Services but with larger volumes of data, it is gradually becoming outdated so it’s becoming inefficient and unreliable.