University of British Columbia
Department of Computer Science
CPSC 304 2020
Summer Term1

Group Project - Implementation of a Relational Database

Project Title:	Hospital Inventory Management System
Project Milestone:	Milestone 4a (Implementation)

#	Student Name	Student Number	Email Address
1	Jin Kyu Lee	33850165	JKL9515@gmail.com
2	Yujia Yang	82719170	yujia_yang1999@sina.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Description of Final Project:

Our final project is a database management system for the inventory of hospitals in the area. One is able to log in as either an inventory manager or a worker at the distribution center. It is the job of the inventory manager to keep the hospitals' inventories updated and maintained. As items are ordered to the hospital, the inventory manager will be able to add elements to the database, adding stock as they arrive. They'll also be able to update and delete elements from the inventory as they get used by other hospital workers among other useful features. This way, the application will be an effective way to constantly manage the hospital's inventory from multiple levels. As there are different logs for deletions and additions, the stock of the database will always remain accurate, and looking up stock of certain equipment will also be easy as each item has a unique medical id and amount in storage.

The distribution center workers must also log into the system. They are able to insert new orders, delete orders, and look at all of the current orders.

With manufacturers, distributors, workers, hospitals, orders, medical items all having specific primary keys, and good normalization, there is no redundancy and it is simple to look up anything that you wish to.

When you enter our app, there is a login screen in which you have to enter credentials to access the system itself. From within the system, there is a UI prompt in which it asks if you wish to add, delete, or update the inventory, as well as having options to search for hospital items of a certain type, providing an interface for the user to query what they need, whether that be all the values of an item or a listing of a certain aspect of those items. One can also combine tables with common elements with a join query, perform aggregate functions on the data to return numbers of drugs, and even division queries for finding details you may need about the hospital's inventory and its manager.

If selecting the distributor option, after logging in, you are able to manage the orders of medical supplies to the hospital: with the ability to add and delete from the list of orders which you may also view.

As such, you can see that our app provides a variety of functions for the hospital's inventory manager and the distributor workers to do their job, which is to manage the inventory and supply orders easily and efficiently.

Final Schema Difference:

The biggest difference in between the schema we turned in and our final schema was the fact that we removed the utilises table and the ShippingCompany_ShipsTo tables. This was because we realized that having a utilises relationship between distributor and shipping company was not the cleanest implementation of our idea. We could simply have a shipped-with

relationship between order and shipping company which would accomplish the same idea but simpler. It also made more sense from a logistical standpoint to connect the shipping company with the order rather than with the distribution company. Other than that, another difference was that we removed all ON UPDATE CASCADE statements. This was firstly because we used Oracle and oracle does not have that functionality built in. We could have implemented it by deferring the foreign key check, updating parent and child, then committing. However, thinking about it further, primary keys should be immutable anyway, so we shouldn't have to update them. We also added drop table statements at the start of the script so that pre-existing values wouldn't interfere with the next insertions. This was done making sure that the child tables were deleted first in order to delete foreign keys before their primary keys were deleted. Similarly, we changed the order in which we created tables so that the primary keys which foreign keys later reference are already in the tables when foreign keys reference them later.

We were also initially planning on using PostgreSQL, but decided later to use Oracle, so there were some formatting things that we had to switch such as having to insert rows one at a time rather than at once and also removing the ON UPDATE CASCADEs. Also, since we used oracle on the school server, we had to make sure to drop any tables used in tutorials so that we had enough room in our database to work with everything in our project.