**Ski Resort Database Design**

**Ben Mintz**

**MET CS579**

**May 6, 2020**

**Introduction:**

The goal of this project is to design a database to oversee a local ski resort’s rental shop. The shop wishes to keep track of all rental receipts – that is to say the actual payment transactions. Additionally, the shop wants to be able to keep track of all pieces of equipment leaving and entering the shop as well as to keep track of how much stock is left in their inventory throughout the day. Ultimately, this database will be connected to a POS (point of sales) system. This means that the database will have high user traffic. Specifically, it will have many insert operations performed in the morning as customers are purchasing rentals, and will have many update operations performed in the afternoon as the agreement status is updated to reflected the items returned to the shop.

**Entity-Relation Diagram:**

A close up of a map

Description automatically generated

As seen above, this database will contain 5 entities. CUSTOMER and SKIER are listed as two separate entities, as these two have different information. CUSTOMER is defined as the person purchasing the rental agreement whereas the SKIER is the person actually uses the equipment. Therefore, CUSTOMER will strictly contain contact and payment and information and SKIER will contain physical attributes such as height and weight, which are needed to skiers to be sized properly by shop staff. However, a SKIER and a CUSTOMER may be the same person. In most cases, the person who is purchasing rentals will also be using the rentals themselves. There will be an INVENTORY table where each size variation of each item will receive a unique identifier. The AGREEMENT entity is an associative entity displaying who purchased the rental, how much it cost, what items were included in that rental, and which skier is using which item.

**Logical Design and Normalization:**

A screenshot of a cell phone

Description automatically generated

Logical Design

A screenshot of a cell phone

Description automatically generated

Normalization

The final logical design is a result of the relations being converted to 3rd normal form (3NF). All of the entities in the original ERD were already in 3NF with the exception of the AGREEMENT entities. This is because two of the attributes in that relation are multi-valued attributes. This led to the creation of the SKIER\_ITEM table with a triple composite primary key. With this new table, there will be one row added per skier, per item used. That is to say if a skier has rented three pieces of equipment before, they will appear in the table three times. The attributes “agreement”, “skier”, and “item\_used” – all of which are foreign keys – jointly contribute to the table’s primary key.

**Physical Design:**

**A screenshot of a cell phone

Description automatically generated**

Notice how on the INVENTORY table I elected to create the attribute “size” as a VARCHAR. Although many items such as skis and snowboards are sized by their length in centimeters, a few items such as helmets are sized by an arbitrary small, medium, large designation. As a result, the entire column must be a VARCHAR. If we wish to do any arithmetic with the numeric sizes, we will simply need to convert the entry to an INTEGER first.

It is also worth noting that within the SKIER\_ITEM table the “item\_status” attribute includes a default value. The only time a new row is added to the SKIER\_ITEM table is when an item is being taken out of the shop for a new rental. Therefore, the only possible status that the item could be in is “In Use”, indicating that it is currently on the mountain. Upon return, the status of the item will be updated to “Returned”. It is also possible for the item status to be updated to “lost” or “damaged” if they were used by a reckless skier.

**Index Creation:**

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

There was a total of three indices created. Two were on the agreement table (the date of rental, and due date), and the third was created on the SKIER\_ITEM table. These were selected because they contain information that the rental shop may wish to receive frequent reports on. For example, the shop will want to know which agreements are delinquent.

**Trigger Creation:**

**A screenshot of a cell phone

Description automatically generated**

The two triggers shown above were created to help keep track of the item quantity. Every time an item is rented out, it is removed from the shop, so it is necessary to update the quantity accordingly. The same can be said for when the item is returned. The first trigger is the situation in which an item is rented and removed from the shop. This occurs when a new row is entered into the SKIER\_ITEM table. When this occurs, the trigger will fire and will deduct one from the current quantity of that specific item.

At the end of the rental period, the items will be returned, and the inventory will once again need to be updated. The second trigger created will fire after the “item\_status” column has been changed to “Returned”. When this occurs, the trigger will execute another update statement to add one to the quantity of that specific item.

**Sample Queries and Output:**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**Conclusion: An Alternative Approach:**

No project is completed a single step. It is an iterative process filled with many modifications as the project progresses and evolves. The creation of this database was no different. Problems with data generation and table creation forced me to rethink the logical design. Even after these issues were resolved, there was a fair amount of hindsight as the project neared completion. Shortly after generating my data, I had thought of a different way of structuring my entities, specifically the CUSTOMER and SKIER tables. Although I stand by my original reasoning for having these as two separate entities, I realized there is no good way to show how skiers and customers may often be the same person. With this in mind, I redesigned my ERD to demonstrate how the same objective can be reached with a different approach.

A close up of text on a white background

Description automatically generated

In the above diagram, we see the ski rental shop as an enhanced ERD. Instead of SKIER and CUSTOMER as two separate entities altogether, they could have been designed as subtypes to the supertype PERSON entity. This would be total, overlapping specialization as a person, could be either a skier, a customer, or both at the same time. However, they must be at least one of them. Otherwise, there would no point in them being listed in the database in the first place. The advantage to this approach is that the CUSTOMER and SKIER tables would share an attribute: person ID. Additionally, there will not be a need to duplicate the names of a person who is both a customer and a skier.

Finally, although I had a SUPPLIER table, I did not make use of it. Part of my alternative approach would have been to restructure the INVENTORY table such that each individual item – not just an item type – would have its own unique serial number, and the item status attribute would live with the INVENTORY table. This way, items can truly be “assigned” to each rental agreement. Furthermore, there is the possibility that an item could be returned damaged. In this case, the item would need to be sent back to the supplier for repairs. In the future, I could include a REPAIR table to show items that have been temporarily sent back the supplier.