CS27020
Assignment: Ski Lifts and Pistes

Author:
James Euesden (jee22)

Address:
Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

Date: February 20, 2014

1 Introduction

This task set is to build a relational database, using PostgreSQL. To begin this task, I have been provided with sample data of Pistes and Lifts, which must be viewed in Unnormalized Form (UNF), and then taken through the normalization process to reach Third Normal Form (3NF). This will be done by determining the Functional Dependencies, constructing Primary Keys and understanding other Candidate Keys, then using these to help with the normalization steps. With the resulting model, I will then create the Database, and provide suitable commands for conducting a series of queries on the database, with screenshot evidence.

2 Analysis

2.1 Unnormalized Structure

An Unnormalized Structure is data that has not been Normalized. This is data that we might find in the 'real world' during situations where we are provided large quantities of data that, while it might make logical sense to a client, may not be appropriate to be implemented into a relational database.

Based upon the sample data provided[1], the unnormalized structure of the Database is as follows. A thing to initially note, is how 'liftName' is contained in the sample data for Piste, yet I have not included it within my functional dependencies. This is because of repeating data, where multiple Lifts can be in multiple Pistes, and visa versa. This will be dealt with during the normalization process.

2.1.1 Piste

pisteName
grade
length
fall
liftName
open

2.1.2 Lift

liftName
type
summit
rise
length
operating

2.2 Functional Dependencies

A functional dependency is where data relies depends upon another piece of data in order to be determined. This can be expressed as FD: X -> Y. To see the functional dependencies, I made a number of assumptions about the data, based upon the sample data provided.

2.2.1 Piste

pisteName ->grade, length, fall, open

We assume that this information about a Piste is functionally dependant upon the pisteName, and this is how it should be accessed.

pisteName ->liftName

In order to find the name of Lifts (liftName) of a Piste, we must know the pisteName.

As a side note, there is an interesting relationship between 'Fall' in Piste and 'Rise' in Lift.

2.2.2 Lift

liftName ->type, summit, rise, length, operating

Similar to Piste, we assume that the data about a Lift is functionally dependant upon liftName.

2.3 Primary & Candidate Keys

A Primary Key (PK) is a key that is unique to each record in a relation, and that will never be repeated in the data set. This key can be a single attribute, or a composite key, comprised of multiple attributes. This key is used as the unique identifier of a relation. When picking a Primary Key, there will be a number of Candidate Keys (CK) that could also be used, but may not be as suitable.

2.3.1 Piste

Primary Key selection: pisteName, length, fall
As a key, these three values are strong, as they are the most likely be unique and unlikely to be changed. It could be safely assumed that even if another Piste was named the same, it would not have the same length and fall.

Candidate Keys, and why they are inappropriate:

pisteName - Potential for more than one Piste to be named the same.

piesteName, length - As above, and could have the same length. Unlikely, but possible.

pisteName, fall - As above.

pisteName, grade - As above.

pisteName, grade, length, fall, open - Strong, but 'open' may change often. a PK should attempt to be static and unchanging. This many attributes is also not required when a smaller key can be made.

2.3.2 Lift

Primary Key selection: liftName, summit, rise
As with Piste, I have selected three attributes for the List Primary Key, based on similar reasons. It is possible, yet unlikely, that two Lifts could be named the same, or named the same and have the same summit or rise, but unlikely to share all three attributes. These are attributes are also unlikely to change.

Candidate Keys, and why they are inappropriate:

liftName - Potential for more than one Piste to be named the same.

liftName, type - As above, they could be of the same type.

liftName, summit - Two same named Lifts could have the same summit. Unlikely but possible.

liftName, rise - As above.

liftName, length - As above.

liftName, type, summit, rise, length, operating - All attributes as a composite PK is a potential choice, and will likely be unique, but is also perhaps too many attributes. It is also possible that 'operating' will change during the course of time. Continuing with trying to keep PKs static and unchanging, it would be best to not use this.

3 Normalizing the Data

When normalizing this data, I will be dealing with each relation (Piste, Lift) separately, without an account of what the other relation's state is, and will bring them together when it seems appropriate.

3.1 First Normal Form

The act of taking data from UNF into 1NF is by disallowing attributes to have multiple values. This means that an attribute could not contain two values, such as 2 phone numbers for one person. In the sample data provided, it can be seen that within 'Piste', there are multiple values in 'liftName'. This violates 1NF rule.

In order to solve this, we can move liftName out of the Piste and into a new relation, 'Connection'. This new relation contains the Primary Key of Piste and the attribute liftName. Piste no longer contains liftName, while otherwise staying the same. This brings Piste into 1NF.

Lift does not have any multiple values within it's attributes, nor could it be assigned any in the future. This means that Lift is already in 1NF and does not need anything doing. There are no more sets of multiple values in Lift or Piste, and Connection is also acceptable at this stage.

The end result of the 1NF operations are below, with the current three relations shown. Those attributes underlined represent Primary Key components. Those attributes with an asterisk () are foreign keys. These relations still have some anomalies however, that will be dealt with in 2NF.

3.1.1 Piste

pisteName
length
fall
grade
open

3.1.2 Lift

liftName
summit
rise
type
length
operating

3.1.3 Connection

pisteName*
length*
fall*
liftName*
(liftName references liftName from relation Lift)

3.2 Second Normal Form

Achieving Second Normal Form relies upon two things, the first being that 1NF is already achieved, the second being that every non-Primary Key attribute of the relation is dependent on the whole of a candidate key. As we can see from the new relation 'Connection', and our functional dependencies, liftName only depends on pisteName, not the length or fall.

In this circumstance, we would separate out piesteName and liftName into their own relation. However, since there are multiple Lifts to multiple Pistes, where a Lift can service many Pistes and a Piste can have multiple Lifts, we bring the Primary Key of each relation into a new relation, which, considering that now there is no need for Connection, we will also call Connection. This relation, Connection, is now of the Primary Keys of both Lift and Pieste, representing the Many-to-Many relationship.

This is valid with 2NF, as we can assume that the full PK of Lift and the full PK of Piste are dependent upon one another in this relation. The attributes left within Lift and Piste are wholly dependant on the Primary Key, and so are also valid. The current relationships are:

3.2.1 Piste

pisteName
length
fall
grade
open

3.2.2 Lift

liftName
summit
rise
type
length
operating

3.2.3 Connection

pisteName, length, fall *
liftName, summit, rise*

(pisteName, length and fall refer to attributes in relation Piste. liftName, summit and rise refer to attributes in relation Lift)

3.3 Third Normal Form

For a database to be valid for Third Normal Form, it must first conform to 2NF, and also have no transitive dependencies. This requires that all non-key attributes rely upon only the PK, and nothing but the key, providing a fact about the PK and nothing else. If we look at our current relations, we can see that this is already the case.

Each attributes relies soley upon the Primary Key of its relation, and provides a fact about that Primary Key, providing no information about any other aspect of the database or of itself. From all this, we can see that Lift has been in 3NF throughout the whole process.

4 PostgreSQL

With the data now in 3NF, it is able to be suitable put into a database. For this task, it must be placed into a PostgreSQL table. I have created this on my personal filestore at Aberystwyth University.

4.1 Creating the tables

4.2 Quering the Database

References

[1] Edel Sherratt, *CS27020 Assignment: Ski Lifts and Pistes*. Computer Science Department, Aberystwyth University, 2014.