CS27020
Modelling Persistent Data
Assignment: Ski Lifts and Pistes

Author:
James Euesden (jee22)

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

Date: February 21, 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, or ready to be put safely and logically into a relational database. 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 there appears to be some correlation between the data in 'rise' in Lift and 'fall' in Piste. However, this only happens for a few of the entries, and with multiple values in attributes, is not something to be taken into consideration.

2.1.1 Piste

piste_name
grade
length_km
fall_m
lif_name
piste_open

2.1.2 Lift

lift_name
type
summit_m
rise_m
length_m
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

piste_name ->grade, length_km, fall_m, piste_open

We assume that this information about a Piste is functionally dependant upon the piste_name, and this is how it should be accessed. Each attribute provides a 'fact' about the Piste, based on the piste_name.

piste_name ->lift_name

In order to find the name of Lifts (lift_name) servicing a Piste, we must know the piste_name. However, it should be noted that Lift is in itself it's own relation. On top of this, a Lift can serve many Pistes, and so it should be noted that this could also be expressed as:

piste_name <-> lift_name

2.2.2 Lift

lift_name ->type, summit_m, rise_m, length_m, operating

Similar to Piste, we assume that the data about a Lift is functionally dependant upon lift_name. Each of these items is functionally dependant on the lift_name, and gives data, a 'fact', about a Lift.

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, it must conform to being unique to that relation, while also being as unchaging as possible (immutable - could change, but shouldn't) and there may also be Candidate Keys.

When deciding the Primary Keys for these relations, I found it challenging to decide the correct course of action. My initial thought was to use the piste_name and lift_name. However, while these are likely to be unique, it is also possible that in the future, a user may want to change or update the names, causing potential update issues. With this in mind, my choice was to make an auto-incremented integer value for both Piste and Lift in order to represent a unique identification number. These are Piste(piste_uid) and Lift(lift_uid).

This choice to use a UID means that a user has the freedom of adding a number of different Pistes, while still giving them the ability to alter the names of a Piste at a later date. With this considered, I decided to continue using piste_name and lift_name as Candidate Keys. This means that while they are not the true Primary Key, they are a Candidate for it, and it should be noted along with the functional dependencies when running the normalization process.

It could be considered that now piste_name and all it's functionally dependant attributes are now all functionally dependent upon piste_uid, and the same for Lift. Since the name values are staying as Candidate keys, and the UID is merely for unique representaton purposes though, the original functional dependencies still hold.

As an extra precaution when creating the database, I will constrain the lift_name and piste_name to be unique, enforcing the rule that no two Pistes or Lifts should be named the same, but their names are free for change.

3 Normalizing the Data

When normalizing this data, I will be dealing with each relation (Piste, Lift) separately, 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 'lift_name'. This violates 1NF rule.

In order to solve this, we can move lift_name out of the Piste and into a new relation, 'Connection'. This new relation contains the Primary Key of Piste and the attribute lift_name. Piste no longer contains lift_name, 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 <u>underlined</u> 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

<u>piste_uid</u>
piste_name
grade
length_km
fall_m
open

3.1.2 Lift

<u>lift_uid</u>
liftName
type
summit_m
rise_m
length_m
operating

3.1.3 Connection

<u>pisteName*</u>
<u>length*</u>
<u>fall*</u>
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

<u>pisteName</u>
<u>length</u>
<u>fall</u>
grade
open

3.2.2 Lift

<u>liftName</u>
<u>summit</u>
<u>rise</u>
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 suitable to be 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. Below are the commands I used to create these, and screenshots to show the creation process.

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.