CS270 Assignment 1 - Ski Lifts and Pistes

Siôn Griffiths - Sig2

February 23, 2014

Contents

1 Introduction

2 Un-Normalized Structures

Studying the dataset provided in the assignment brief, I have determined the un-normalized structure of the database (along with all repeating groups) to be as follows:

Piste(<u>PisteName</u>, Grade, Length, Fall, Lifts, Open, LiftName, Type, Summit, Rise, LiftLength, Operating)

(LiftName, Type, Summit, Rise, LiftLength, Operating) repeats for each lift

The data is separated out into two tables which implies some logical structuring has already been carried out. I have determined that Piste if the main entity of interest and all other attributes are in some way derivative from it.

2.1 Candidate Keys

The following candidate keys have been identified:

2.1.1 Grade, Length, Fall

A composite of the Grade, Length and Fall attributes would serve to uniquely identify a piste according to the data provided in the brief. Even though a composite of Grade + Fall or Length + Fall would also appear to uniquely identify a piste I have chosen to use a composite of all three as a candidate since all three together provide a key which is more unique.
The uniqueness of this composite key lies in the assumption that mountain slopes are of differing heights and gradients, and that the complexity of terrain on a mountain slope varies (Grade).
This candidate has not been chosen as primary key. It is usually best to avoid the use of composite keys as primary key if possible and there's always the possibility that 2 pistes could share similar slopes such that this candidate key would not be sufficient to differentiate between them.

2.1.2 Lifts

The Lifts attribute appears to be unique to each piste according to the dataset provided in the assignment brief. Each piste is served by a unique collection of lifts or lift, this would provide sufficient information to uniquely identify a piste.
This candidate key has not been chosen as primary key. Since some lifts serve a number of pistes it is possible to assume that a lift could be removed in the future, such that a number of pistes have the exact same set of lifts serving them, in this case the Lifts attribute would no longer be enough to uniquely identify a piste.

2.1.3 A composite of all attributes

Looking at the dataset, it is fairly safe to assume that the entire entry for a piste will always be unique. However, there are more suitable and concise choices available so this candidate has not been chosen as primary key.

2.1.4 PisteName

Each piste has a unique name. The PisteName is a single, concise entry that serves to uniquely identify a piste. Since this is the most concise and intuitive choice, it has been selected as the primary key for the un-normalized representation of the dataset.

2.2 Functional Dependencies

The following functional dependencies have been identified:

PisteName → Grade, Length, Fall, Lifts, Open

The attributes Grade, Length, Fall, Lifts and Open can be considered functionally dependant on PisteName since they make no sense outside of the context of a Piste. None of these attributes on it's own is enough to identify a Piste. However, a PisteName is sufficient to derive any or all of these attributes.

LiftName → Type, Summit, Rise, LiftLength, Operating

Type, Summit, Rise, LiftLength and Operating can all be considered to be functionally dependant on LiftName. Similar to the situation with PistName and it's dependant attributes, LiftName can be used to resolve any of these attributes where as the attributes themselves in isolation cannot identify a Lift, or indeed any other attribute.

3 Assumptions

4 First Normal Form

Having identified functional dependencies and chosen a primary key the structure can be normalised.

In order to bring the structure to first normal form, repeating groups must be removed from the relation as shown in the un-normalized structure in section 2 of this document. The primary key of the un-normalized structure is included in the key of the new relation. Doing so results in the following relations:

Piste(PisteName, Grade, Length, Fall, Open)
Lift(LiftName, PisteName*, Type, Summit, Rise, LiftLength, Operating)

Because of the functional dependency : **PisteName → Grade, Length, Fall, Lifts, Open**, all attributes in the Piste relation are derived via the PisteName. Similarly, all attributes in Lift are derived via the LiftName, since **LiftName → Type, Summit, Rise, LiftLength, Operating**. The structure is fairly close to the functional dependencies that were specified in the earlier section, but further work is required.

5 Second normal Form

In order to bring this representation to second normal form we must ensure that no attributes are partially dependant on parts of a composite primary key. The lift relation in the First Normal Form section above has it's attributes only dependant on the LiftName attribute and not on the PisteName foreign key that also makes up it's key.

PisteName has to be removed from the Lift relation and in order to represent the many-to-many relationship between Pistes and Lifts, a new table is required. This new table will hold only a foreign key reference to both Lift and Piste via their primary keys. This results in the following relations:

Piste(PisteName, Grade, Length, Fall, Open)
Lift(LiftName, Type, Summit, Rise, LiftLength, Operating)
Serves(PisteName*, LiftName*)

The structures are now fully compliant with the functional dependencies that were outlined in the previous sections.

6 Third normal Form

In order for a representation to be in third normal form, it must conform to first and second normal form and also transitive dependencies must be removed. No attributes should depend on anything but the primary key.

With the dataset used in this particular example third normal form requires no extra steps from second normal form since there are no transitive dependencies.

7 PostgreSQL Implementation

```sql
CREATE TYPE difficulty as ENUM('Easy', 'Medium', 'Hard', 'Difficult');

CREATE TABLE piste(
        piste_name varchar(50) PRIMARY KEY,
        grade difficulty,
        length decimal(4,2),
        fall smallint,
        open boolean
);

CREATE TYPE type_of_lift as ENUM('Gondola', 'Chair', 'Tow');

CREATE TABLE lift(
        lift_name varchar(50) PRIMARY KEY,
        lift_type type_of_lift,
        summit smallint,
        rise smallint,
        lift_length smallint,
        operating boolean
);
```