

# CS270 Assignment 1 - Ski Lifts and Pistes

Siôn Griffiths - Sig2

February 24, 2014

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Un-Normalized Structures</b>	<b>2</b>
2.1	Candidate Keys . . . . .	2
2.1.1	Grade, Length, Fall . . . . .	2
2.1.2	Lifts . . . . .	2
2.1.3	A composite of all attributes . . . . .	3
2.1.4	PisteName . . . . .	3
2.2	Functional Dependencies . . . . .	3
<b>3</b>	<b>First Normal Form</b>	<b>3</b>
<b>4</b>	<b>Second normal Form</b>	<b>4</b>
<b>5</b>	<b>Third normal Form</b>	<b>4</b>
<b>6</b>	<b>PostgreSQL Implementation</b>	<b>5</b>
6.1	Attribute Types . . . . .	5
6.2	SQL . . . . .	6
6.3	Create Tables . . . . .	6
6.4	Inset Data . . . . .	7
6.5	Queries . . . . .	8
6.6	Drop Tables . . . . .	8
6.7	Output . . . . .	9
6.7.1	Create.sql . . . . .	9
6.7.2	Insert.sql . . . . .	9
6.7.3	Query.sql . . . . .	10
6.7.4	Drop.sql . . . . .	10

# 1 Introduction

This document provides an overview of my solution for the CS270 Ski Lifts and Pistes Assignment. The problem given in the assignment brief is to analyse a set of data, normalise this data to a form suitable to be implemented in a relational database system, and provide a PostgreSQL implementation of this dataset.

## 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**(PisteName, 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 that some logical structuring has already been carried out. I have determined that Piste is the main entity of interest and all other attributes are in some way derivable 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  $\rightarrow$  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  $\rightarrow$  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 PisteName 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 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**  $\rightarrow$  **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**  $\rightarrow$  **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.

## 4 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.

## 5 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.

## 6 PostgreSQL Implementation

### 6.1 Attribute Types

Attributes	Type	Justification
PisteName	Varchar[50]	Using Varchar[50] to represent the PisteName attribute allows enough space to represent all pistes in the dataset.
Grade	Enum	Since there are only a small number of discrete values possible for this attribute an Enum seems like the most logical choice.
Length	Decimal(4,2)	A decimal type with precision of 4 and scale of 2 is sufficient to represent this attribute. If the dataset is to be considered representative then a Decimal(2,1) could be used but I have chosen to use a (4,2) representation to allow for longer runs or more precise measurements in the future.
Fall	Smallint	The numeric range required for the Fall attribute is such that a Smallint type is more than sufficient to store it.
Open	Boolean	Since the Open attribute only has 2 possible values a boolean type is the most suitable choice to represent it.
LiftName	Varchar[50]	Using Varchar[50] to represent the LiftName attribute allows enough space to represent all lifts in the dataset.
Type	Enum	Since there are only a small number of discrete values possible for this attribute an Enum seems like the most logical choice.
Summit	Smallint	It is safe to assume that a Summit will never be more than 32k meters, using a Smallint to represent this type is more than sufficient.
Rise	Smallint	Since it is unlikely that a Rise would have a higher value than a Summit, a smallint is sufficient to represent this attribute.
LiftLength	Smallint	From studying the given dataset a Smallint should be more than sufficient to represent this attribute.
Operating	Boolean	Since the Operating attribute only has 2 possible values a boolean type is the most suitable choice to represent it.

## 6.2 SQL

Implementing the database on the departmental computer system was achieved by running the following command for each of the files specified below. The SQL scripts themselves were uploaded to my university filestore. :

```
psql -h db.dcs.aber.ac.uk -U sig2 -d cs27020_13_14 < sql/filename.sql
```

## 6.3 Create Tables

```
1 CREATE TYPE difficulty as ENUM('Easy', 'Medium', 'Hard', 'Difficult');
2
3 CREATE TABLE piste(
4     piste_name varchar(50) PRIMARY KEY,
5     grade difficulty,
6     length decimal(4,2),
7     fall smallint,
8     open boolean
9 );
10
11 CREATE TYPE type_of_lift as ENUM('Gondola', 'Chair', 'Tow');
12
13 CREATE TABLE lift(
14     lift_name varchar(50) PRIMARY KEY,
15     lift_type type_of_lift,
16     summit smallint,
17     rise smallint,
18     lift_length smallint,
19     operating boolean
20 );
21
22 CREATE TABLE serves(
23     piste_name varchar(50) references piste(piste_name),
24     lift_name varchar(50) references lift(lift_name),
25     CONSTRAINT serves_key PRIMARY KEY (piste_name, lift_name)
26 );
```

## 6.4 Inset Data

```
1 INSERT INTO piste (piste_name, grade, length, fall, open) VALUES
2 ('Zwischenholzabfahrt', 'Medium', '3', '440', 'TRUE'),
3 ('Moeseralmabfahrt', 'Medium', '2.5', '400', 'FALSE'),
4 ('Schoenjochabfahrt', 'Medium', '4', '510', 'TRUE'),
5 ('Sattelkopf-Suedabfahrt', 'Medium', '4', '350', 'TRUE'),
6 ('Sattelkopf-Nordabfahrt', 'Difficult', '1.5', '220', 'TRUE'),
7 ('Moeserabfahrt', 'Easy', '0.5', '80', 'TRUE'),
8 ('Wonneabfahrt', 'Medium', '1.5', '280', 'FALSE'),
9 ('Rastabfahrt', 'Medium', '1', '150', 'FALSE'),
10 ('Waldabfahrt', 'Hard', '3', '420', 'TRUE'),
11 ('Ladisabfahrt', 'Easy', '3.5', '290', 'TRUE'),
12 ('Verbindungsabfahrt', 'Easy', '2', '70', 'TRUE'),
13 ('Plazoerabfahrt', 'Medium', '3', '360', 'FALSE'),
14 ('Schoengampabfahrt', 'Medium', '3.5', '420', 'TRUE'),
15 ('Schoenjochpiste', 'Easy', '1', '70', 'TRUE'),
16 ('Almabfahrt', 'Medium', '4', '370', 'FALSE');
17
18
19 INSERT INTO lift (lift_name, lift_type, summit, rise, lift_length, operating)
20 VALUES
21 ('Schoenjochbahn I', 'Gondola', '1920', '440', '1600', 'TRUE'),
22 ('ESL-Fiss-Moeseralm', 'Chair', '1850', '400', '1700', 'FALSE'),
23 ('ESL-Ladis-Fiss', 'Chair', '1510', '290', '2700', 'FALSE'),
24 ('Waldlift', 'Tow', '1850', '420', '1200', 'TRUE'),
25 ('Rastlift', 'Tow', '1900', '150', '400', 'TRUE'),
26 ('Schoenjochbahn II', 'Gondola', '2436', '516', '1350', 'TRUE'),
27 ('Sattelkopflift', 'Tow', '2100', '220', '1000', 'FALSE'),
28 ('Moeserlift', 'Tow', '1930', '80', '400', 'TRUE'),
29 ('Wonnelifft', 'Tow', '2080', '280', '1000', 'TRUE'),
30 ('Plazoerlift', 'Tow', '2450', '360', '1350', 'TRUE'),
31 ('Schoenjochlift', 'Tow', '2509', '70', '420', 'FALSE'),
32 ('Schoengamplift', 'Tow', '2509', '420', '1340', 'TRUE'),
33 ('Almlift', 'Tow', '2250', '370', '1180', 'FALSE');
34
35 INSERT INTO serves (piste_name, lift_name) VALUES
36 ('Zwischenholzabfahrt', 'Schoenjochbahn I'),
37 ('Zwischenholzabfahrt', 'ESL-Fiss-Moeseralm'),
38 ('Moeseralmabfahrt', 'ESL-Fiss-Moeseralm'),
39 ('Moeseralmabfahrt', 'Rastlift'),
40 ('Schoenjochabfahrt', 'Schoenjochbahn II'),
41 ('Schoenjochabfahrt', 'Plazoerlift'),
42 ('Schoenjochabfahrt', 'Schoenjochlift'),
43 ('Sattelkopf-Suedabfahrt', 'Waldlift'),
44 ('Sattelkopf-Suedabfahrt', 'Sattelkopflift'),
45 ('Sattelkopf-Nordabfahrt', 'Sattelkopflift'),
46 ('Moeserabfahrt', 'Moeserlift'),
47 ('Wonneabfahrt', 'Schoenjochbahn I'),
48 ('Wonneabfahrt', 'Wonnelifft'),
49 ('Rastabfahrt', 'Rastlift'),
50 ('Waldabfahrt', 'Waldlift'),
51 ('Ladisabfahrt', 'ESL-Ladis-Fiss'),
52 ('Verbindungsabfahrt', 'Schoenjochbahn I'),
53 ('Verbindungsabfahrt', 'Wonnelifft'),
54 ('Plazoerabfahrt', 'Schoenjochbahn II'),
```



```

54 ('Plazoerabfahrt' , 'Plazoerlift' ),
55 ('Schoengampabfahrt', 'Schoengamplift'),
56 ('Schoenjochpiste' , 'Schoenjochbahn II'),
57 ('Schoenjochpiste' , 'Plazoerlift' ),
58 ('Almabfahrt' , 'Schoenjochbahn II'),
59 ('Almabfahrt' , 'Plazoerlift' ),
60 ('Almabfahrt' , 'Almlift');

```

## 6.5 Queries

```

1 SELECT lift_name FROM lift WHERE operating = TRUE;
2
3 SELECT lift_name FROM serves WHERE piste_name = 'Almabfahrt' ;
4
5 SELECT piste_name FROM serves WHERE lift_name = 'Schoenjochbahn II';
6
7 SELECT p.piste_name, l.lift_name
8 FROM piste p
9 INNER JOIN serves s ON s.piste_name = p.piste_name
10 INNER JOIN lift l ON l.lift_name = s.lift_name
11 WHERE p.open = true AND l.operating = true;

```

## 6.6 Drop Tables

```

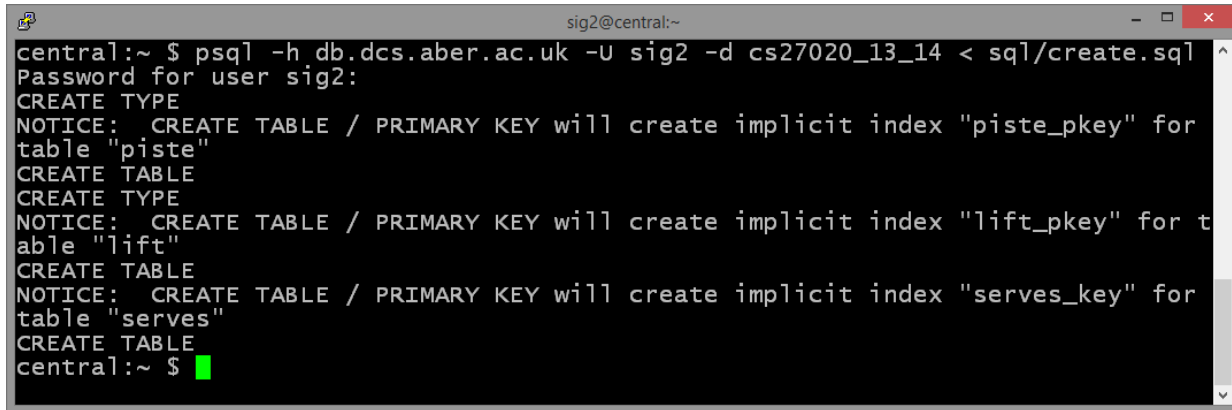
1 DROP TABLE IF EXISTS piste CASCADE;
2 DROP TABLE IF EXISTS lift CASCADE;
3 DROP TABLE IF EXISTS serves CASCADE;
4
5 DROP TYPE IF EXISTS difficulty CASCADE;
6 DROP TYPE IF EXISTS type_of_lift CASCADE;

```

## 6.7 Output

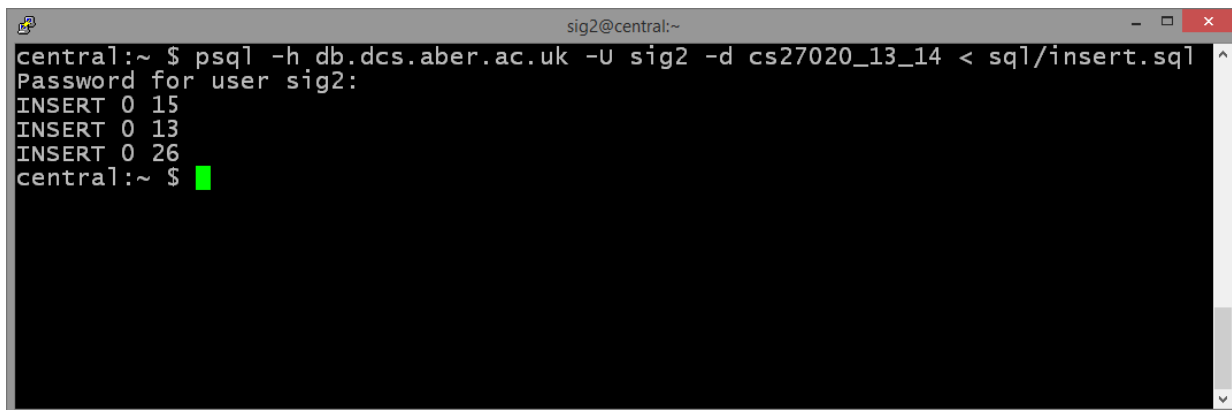
The following screenshots show the output of running the SQL scripts detailed above.

### 6.7.1 Create.sql



```
central:~ $ psql -h db.dcs.aber.ac.uk -U sig2 -d cs27020_13_14 < sql/create.sql
Password for user sig2:
CREATE TYPE
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "piste_pkey" for
table "piste"
CREATE TABLE
CREATE TYPE
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "lift_pkey" for t
able "lift"
CREATE TABLE
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "serves_key" for
table "serves"
CREATE TABLE
central:~ $
```

### 6.7.2 Insert.sql



```
central:~ $ psql -h db.dcs.aber.ac.uk -U sig2 -d cs27020_13_14 < sql/insert.sql
Password for user sig2:
INSERT 0 15
INSERT 0 13
INSERT 0 26
central:~ $
```

### 6.7.3 Query.sql

```
central:~ $ psql -h db.dcs.aber.ac.uk -U sig2 -d cs27020_13_14 < sql/query.sql
Password for user sig2:
lift_name
-----
Schoenjochbahn I
Waldlift
Rastlift
Schoenjochbahn II
Moeserlift
Wonnelifft
Plazoerlift
Schoengamplift
(8 rows)

lift_name
-----
Schoenjochbahn II
Plazoerlift
Almlift
(3 rows)

piste_name
-----
Schoenjochabfahrt
Plazoerabfahrt
Schoenjochpiste
Almabfahrt
(4 rows)

piste_name | lift_name
-----|-----
Zwischenholzabfahrt | Schoenjochbahn I
Schoenjochabfahrt | Schoenjochbahn II
Schoenjochabfahrt | Plazoerlift
Sattelkopf-Suedabfahrt | Waldlift
Moeserabfahrt | Moeserlift
Waldabfahrt | Waldlift
Verbindungsabfahrt | Schoenjochbahn I
Verbindungsabfahrt | Wonnelifft
Schoengampabfahrt | Schoengamplift
Schoenjochpiste | Schoenjochbahn II
Schoenjochpiste | Plazoerlift
(11 rows)

central:~ $ █
```

### 6.7.4 Drop.sql

```
central:~ $ psql -h db.dcs.aber.ac.uk -U sig2 -d cs27020_13_14 < sql/drop.sql
Password for user sig2:
NOTICE: drop cascades to constraint serves_piste_name_fkey on table serves
DROP TABLE
NOTICE: drop cascades to constraint serves_lift_name_fkey on table serves
DROP TABLE
DROP TABLE
DROP TYPE
DROP TYPE
central:~ $ █
```