Advanced Data Warehousing

Topic: Surrogate Keys

1. Introduction

What is a Surrogate Key?
A surrogate key is a unique identification for each record in a dimension table. It is a primary key. Normally a surrogate key is a sequence number.

Why are Surrogate Keys needed in some cases?
In the real world, input operational databases are not only one system; it can be multiple operational databases. The same table may exist in various forms in these multiple operational databases.

For example, table Employee may exists in a Payroll database, as well as in an Academic database, and a Library database – all of which are different database systems. Another example is table Product, which may exist in an Inventory database, as well as in a Manufacturing database, a Sales database, etc.

A data warehouse is normally built from multiple operational databases, and when a dimension is created, ideally the primary key, which is the ID of the table is unique across all operational database systems. For example, if we are going to have EmployeeID as a PK in the EmployeeDIM in our data warehouse, we expect that EmployeeID is unique across all the input operational databases – which might likely be the case. Also, if we are going to have a ProductDIM with ProductID as a PK, we expect that ProductID is unique across all database systems.

However, in the real world, a unique identification, which is valid across all systems, does not always happen. Different databases may have different identification systems. For example, ProductID "AB335" in one system may be coded as "AB3351" in another system. Therefore, we cannot assume that ProductID is unique across all database systems. This is the reason by people in industry prefer to use surrogate keys in dimension tables.

So, instead of using the original ProductID (whether it be "AB335" or "AB3351"), we use a *surrogate key*, which is a new key – normally implemented as a sequence number. Hence, the new ProductID will be 1, 2, 3….

We have actually used a surrogate key in the Uselog case study, especially in the LabTimeDIM, where we have three records: morning, afternoon, night, with IDs 1, 2, and 3. These IDs are surrogate keys.

So far, in our case studies, because the operational database is simple, we do not enforce the use of surrogate key. For example, in the Uselog case study, the four dimensions are SemesterDIM (with SemID attribute having either "S1" or "S2"), LabTimeDIM (with LabTimeID attribute having either 1, 2, or 3), MajorDIM (with MajorCode from the original operational table), and CourseDIM (with CourseCode from the original operational table). In this case study, there is no need to create a sequence number-based surrogate key for

MajorDIM and CourseDIM, because both tables have unique PKs in the operational database anyway.

In the Clothing case study, we have a dimension called SourceDIM, which has one attribute called SourceID ("Fax", "Phone", and "Website"). These values are extracted from the OrderSource table from the operational database. In this case, because the case study is simple, we did not implement a surrogate key. However, we could have something like 1 for "Fax", 2 for "Phone", and 3 for "Website" – if we insist to have a surrogate key.

2. An Example

How do we implement Surrogate Keys in the star schema?
The concept of surrogate keys is very simple as explained above. It is basically a PK attribute of a dimension, and the contents are sequence numbers. The first record of a dimension table has surrogate key 1, the second record 2, the third record 3, etc. But implementing it can be challenging, in terms of coding it.

Suppose an operational database contains one table, called *Student*.

Table *Student*

StudentID	StudentName	Suburb	Postcode	Sex
21001	Adam	Caulfield	3162	M
21003	Ben	Caulfield	3162	M
21008	Christine	Chadstone	3148	F
21019	Daisy	Caulfield	3162	F
21033	Edward	Clayton	3168	M
21122	Fred	Caulfield	3162	M
21123	Greg	Chadstone	3148	M

Suppose the star schema contains only two dimensions: SuburbDIM and SexDIM. The fact table has one fact measure, called TotalStudents.

SuburbDIM table can be created using the following SQL command (assume that the operational database is located at `dtaniar` account):

```
CREATE TABLE SuburbDIM as
       SELECT distinct Suburb, Postcode
       FROM dtaniar.Student1;
```

Table *SuburbDIM*

Suburb	Postcode
Caulfield	3162
Chadstone	3148
Clayton	3168

SexDIM table can be created the same way:

```
CREATE TABLE SexDIM as
       SELECT distinct Sex
       FROM dtaniar.Student1;
```

Table *SexDIM*

Sex
M
F

And the Fact table can be created using the following SQL command:

```
CREATE TABLE Fact as
  SELECT Suburb, Sex, Count(*) as TotalStudents
  FROM dtaniar.Student1
  GROUP BY Suburb, Sex;
```

Table *Fact*

Suburb	Sex	TotalStudents
Caulfield	M	3
Caulfield	F	1
Chadstone	M	1
Chadstone	F	1
Clayton	M	1

In this case, we assume that the attribute Suburb in SuburbDIM table becomes the primary key. For the SexDIM, because it has only one attribute, namely Sex, this attribute automatically becomes the ID of this table.

Now, suppose we want to implement a surrogate key in the SuburbDIM, which we call SuburbID, for example.

Table *SuburbDIM*

SuburbID	Suburb	Postcode
1	Caulfield	3162
2	Chadstone	3148
3	Clayton	3168

The SQL to create SuburbDIM is as follows:

```
CREATE TABLE SuburbDIM as
       SELECT distinct Suburb, Postcode
       FROM dtaniar.Student1;

Alter Table SuburbDIM add (SuburbID number(2));
```

```
Drop Sequence Suburb_seq_ID;

Create Sequence Suburb_seq_ID
  start with 1
  increment by 1
  maxvalue 99999999
  minvalue 1
  nocycle;

Update SuburbDIM SET SuburbID = suburb_seq_ID.nextval;
```

The next question is how to create the fact (or tempfact) table?
First, we need to create a tempfact table that includes Suburb Name.

```
CREATE TABLE TempFact as
  SELECT Suburb, Sex FROM dtaniar.Student1;
```

The next step is to add a column, called SuburbID, in Tempfact table:

```
Alter Table TempFact
Add (SuburbID Number(2));
```

Then Tempfact's SuburbID attribute must be filled in with the correct values, which correspond to the values of Suburb and Postcode. There are particular TWO ways. The first way is to do an update one-by-one. In this case, we have only 3 suburb records, so we need 3 updates:

```
Update TempFact
Set SuburbID = 1
Where Suburb = 'Caulfield';

Update TempFact
Set SuburbID = 2
Where Suburb = 'Chadstone';

Update TempFact
Set SuburbID = 3
Where Suburb = 'Clayton';
```

However, if the number of suburb in the SuburbDIM is unknown or relatively large, it is undesirable to do one update for each suburb. A better solution is to have one update command as follows:

```
update TempFact tf
set tf.SuburbID = (select s.SuburbID
                   from SuburbDIM s
                   where s.Suburb = tf.Suburb);
```

The query statement to create the *fact table* is as followed:

```
Create Table Fact as
Select SuburbID, Sex, Count(*) as TotalStudents
From TempFact
GROUP BY SuburbID, Sex;
```

Table *Fact*

SuburbID	Sex	TotalStudents
1	M	3
1	F	1
2	M	1
2	F	1
3	M	1

Obviously, when you want to produce a report, you need to join between table Fact and table SuburbDIM, in order to get the real suburb/postcode, rather than just 1, 2, 3…

Homework: Create a surrogate key for the SexDIM, and modify the Fact table to include the SexID as the surrogate key.