Junk Dimensions – A Real Estate Agent Case Study

An established real estate agent in Melbourne has started their business many years ago and has implemented a very simple database system. The simple database system consists of *one* large table with the following attributes as shown below.

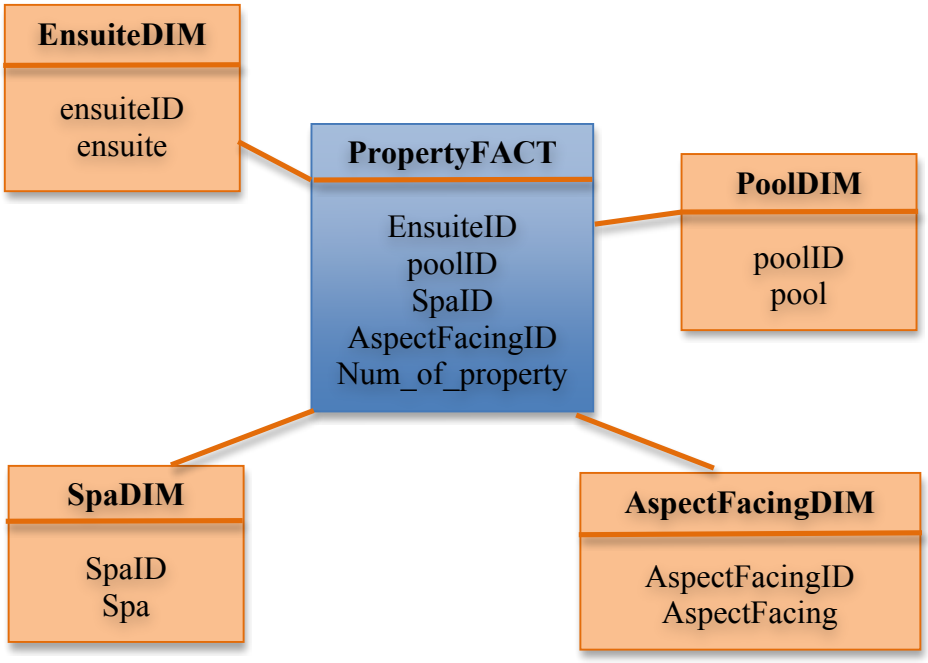
Table Name: PROPERTY	
Field Name	Description
Key	Unique key
Date_offered	Date property offered to the public
Summary	Short description of the property
Adtext	Longer description of the property
Url	The URL of the advertisement
Address	Property address
Suburb	Property suburb name
Postcode	Property postcode
Longitude	Longitude of address
Latitude	Latitude of address
Category	'Residential' or 'Commercial'
Zoning	Commercial Zoning Type
Property_type	Residential Property Type: 'House', 'apartment', or 'lot'
Houseprice	Price of property
Num_bedrooms	Number of bedrooms
Lot_size	Size of the lot
Heating	'ducted', 'gas', 'open fireplace' or 'wood'
Garage	Type of garage
Ensuite	'yes' or 'no'
Balcony	'yes' or 'no'
Pool	'yes', 'no'
Tennis_court	'yes', 'no'
Spa	'yes', 'no'
Aspect_facing	'north', 'south', 'east', or 'west'
School_distance	Distance to nearest school – in km
Shops_distance	Distance to nearest shops – in km
Train_distance	Distance to nearest train station – in km
Bus_distance	Distance to nearest bus stop – in km
Hospital_distance	Distance to nearest hospital – in km
Major_road_distance	Distance to nearest major road – in km

The manager of the real estate agent requires a data warehouse for analysis purposes. In particular, the manager would like to analyse number of properties using some variables, such as properties with pools or spa, or properties having a certain aspect facing (e.g. north facing), etc. Thus, a small data warehouse needs to be built.

There are two options for the desired star schema: Option 1 is to use **normal dimensions**, whereas option 2 is to use **a junk dimension**.

Option 1: Normal Dimensions (Non-Junk Dimensions)

For the sake of discussion of this topic, we call option 1, the **Non-Junk** Dimension Option. A non-junk dimension star schema could look like the following:



From the Property data, attributes such as ensuite, pool, spa, and aspect facing, have low cardinality. In a non-junk dimension schema, low-cardinality attributes are stored as individual dimension tables, such as pool dimension, spa dimension, etc.

The SQL statement to create the *ensuite dimension* is as followed:

```
CREATE TABLE EnsuiteDIM1 as
       SELECT distinct Ensuite
       FROM dw.Property1;

ALTER TABLE EnsuiteDIM1
       ADD (EnsuiteID number(1));

UPDATE EnsuiteDIM1
       SET EnsuiteID = 1
       WHERE Ensuite = 'yes';

UPDATE EnsuiteDIM1
       SET EnsuiteID = 2
       WHERE Ensuite = 'no';

UPDATE EnsuiteDIM1
       SET EnsuiteID = 0
       WHERE Ensuite = 'null';

Note: there is string ‘null’ in dw.property1 table.
String ‘null’ is different from a NULL value.
```

Ensuite Dimension table:

EnsuiteID	Ensuite
1	yes
2	no
0	Null (string 'null', not a NULL value)

The SQL statement to create the *pool dimension* is as followed:

```
CREATE TABLE PoolDIM1 as
     SELECT distinct Pool
     FROM dw.Property1;

ALTER TABLE PoolDIM1
     ADD(PoolID number(1));

UPDATE PoolDIM1
     SET PoolID = 1
     WHERE Pool = 'yes';

UPDATE PoolDIM1
     SET PoolID = 2
     WHERE Pool = 'no';
```

Pool Dimension table:

PoolID	Pool
1	yes
2	no

The SQL statement to create the *aspect facing dimension* is as followed:

```
CREATE TABLE AspectFacingDIM1 as
     SELECT distinct Aspect_Facing
     FROM dw.Property1;

ALTER TABLE AspectFacingDIM1
     ADD(AspectFacingID number(1));


UPDATE AspectFacingDIM1
     SET AspectFacingID = 1
     WHERE Aspect_Facing = 'North';

UPDATE AspectFacingDIM1
     SET AspectFacingID = 2
     WHERE Aspect_Facing = 'South';
```

```
UPDATE AspectFacingDIM1
     SET AspectFacingID = 3
     WHERE Aspect_Facing = 'East';

UPDATE AspectFacingDIM1
     SET AspectFacingID = 4
     WHERE Aspect_Facing = 'West';
```

Aspect Facing Dimension table:

AspectFacingID	Aspect_Facing
1	North
2	South
3	East
4	West

The SQL statement to create the *spa dimension* is as followed:

```
CREATE TABLE SpaDIM1 as
     SELECT distinct Spa
     FROM dw.Property1;

ALTER TABLE SpaDIM1
     ADD(SpaID number(1));

UPDATE SpaDIM1
     SET SpaID = 1
     WHERE Spa = 'yes';

UPDATE SpaDIM1
     SET SpaID = 2
     WHERE Spa = 'no';
```

Spa Dimension table:

SpaID	Spa
1	yes
2	no

Table 1: Spa dimension table

The SQL statement to create the *Tempfact* table is as followed:

```
CREATE TABLE TempFact1 as
  SELECT Ensuite,
         Pool,
         Aspect_facing,
         Spa
  FROM dw.property1;
```

The SQL statement to alter the *Tempfact* table is as followed:

```
ALTER TABLE TempFact1
  ADD (EnsuiteID Number(1));

UPDATE TempFact1
  SET EnsuiteID = 1
  Where Ensuite = 'yes';

UPDATE TempFact1
  SET EnsuiteID = 2
  Where Ensuite = 'no';

UPDATE TempFact1
  SET EnsuiteID = 0
  Where Ensuite = 'null';


ALTER TABLE TempFact1
  ADD (poolID Number(1));

UPDATE TempFact1
  SET poolID = 1
  Where pool = 'yes';

UPDATE TempFact1
  SET poolID = 2
  Where pool = 'no';


ALTER TABLE TempFact1
  ADD (Aspect_FacingID Number(1));

UPDATE TempFact1
     SET Aspect_FacingID = 1
     WHERE Aspect_Facing = 'North';

UPDATE TempFact1
     SET Aspect_FacingID = 2
     WHERE Aspect_Facing = 'South';

UPDATE TempFact1
     SET Aspect_FacingID = 3
     WHERE Aspect_Facing = 'East';

UPDATE TempFact1
     SET Aspect_FacingID = 4
     WHERE Aspect_Facing = 'West';
```

```
ALTER TABLE TempFact1
  ADD (SpaID Number(1));

UPDATE TempFact1
     SET SpaID = 1
     WHERE spa = 'yes';

UPDATE TempFact1
     SET SpaID = 2
     WHERE spa = 'no';
```

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

```
Create table Propertyfact1 as
  SELECT EnsuiteID,
         poolID,
         Aspect_FacingID,
         SpaID,
         count(*) as Num_of_property
  FROM TempFact1
  GROUP BY EnsuiteID, poolID, Aspect_FacingID, SpaID;
```

The following shows the example of records stored in each tables according to the query statements above.

PropertyFact Table:

```
SQL> select *
  2       from propertyfact1
  3       order by ensuiteID, poolID, Aspect_FacingID, spaID;

 ENSUITEID     POOLID ASPECT_FACINGID      SPAID NUM_OF_PROPERTY
---------- ---------- --------------- ---------- ---------------
         0          2               1          2            1181
         0          2               2          2            4459
         0          2               3          2            2243
         0          2               4          2            2245
         1          1               1          1             145
         1          1               1          2             175
         1          1               2          1             716
         1          1               2          2             671
         1          1               3          1             387
         1          1               3          2             331
         1          1               4          1             334

 ENSUITEID     POOLID ASPECT_FACINGID      SPAID NUM_OF_PROPERTY
---------- ---------- --------------- ---------- ---------------
         1          1               4          2             334
         1          2               1          1             950
         1          2               1          2             940
         1          2               2          1            3741
         1          2               2          2            3730
         1          2               3          1            1953
         1          2               3          2            1900
         1          2               4          1            1854
         1          2               4          2            1805
         2          1               1          1             190
         2          1               1          2             176

 ENSUITEID     POOLID ASPECT_FACINGID      SPAID NUM_OF_PROPERTY
---------- ---------- --------------- ---------- ---------------
         2          1               2          1             764
         2          1               2          2             640
         2          1               3          1             375
         2          1               3          2             384
         2          1               4          1             331
         2          1               4          2             358
         2          2               1          1             863
         2          2               1          2             832
         2          2               2          1            3697
         2          2               2          2            3801
         2          2               3          1            1911

 ENSUITEID     POOLID ASPECT_FACINGID      SPAID NUM_OF_PROPERTY
---------- ---------- --------------- ---------- ---------------
         2          2               3          2            1824
         2          2               4          1            1903
         2          2               4          2            1857

36 rows selected.
```

Option 2: Junk Dimensions

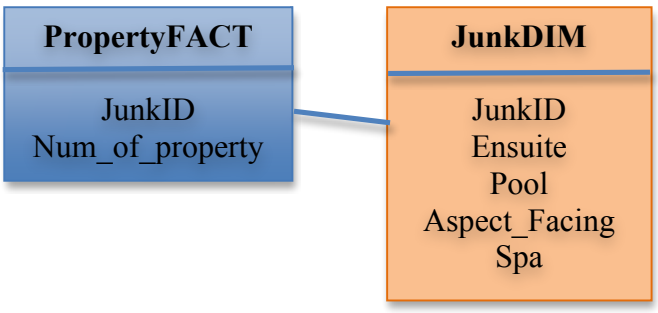
Junk dimension is a type of dimension that consolidates all the low-cardinality attributes or many small dimensions tables into a single dimension table (Low-cardinality attributes are attributes with small range of values, such as male/female, yes/no, 1/2/3, North/South/East/West and etc). Therefore, the content of a junk dimension is a Cartesian Product of the values of all its attributes. An example of a junk dimension is shown below:

Junk dimension table:

JUNKID	ENSUITE	POOL	ASPECT_FACING	SPA
1	yes	no	East	yes
2	yes	no	South	yes
3	no	no	North	yes
4	no	no	East	no
5	yes	no	North	no
6	yes	yes	South	yes
7	no	no	West	no
8	yes	yes	East	yes
9	null	no	North	no
10	no	yes	East	no
11	yes	no	South	no
12	yes	no	North	yes
13	yes	no	East	no
14	no	no	North	no
15	no	yes	South	yes
16	yes	yes	North	no
17	no	no	East	yes
18	yes	yes	South	no
19	no	no	South	yes
20	no	yes	West	no
21	yes	yes	North	yes
22	no	yes	North	yes
23	null	no	East	no
24	no	no	South	no
25	nul	no	West	no
26	no	yes	West	yes
27	yes	no	West	no
28	no	no	West	yes
29	yes	yes	West	no
30	no	yes	North	no
31	no	yes	East	yes
32	nul	no	South	no
33	yes	yes	West	yes
34	yes	no	West	yes
35	yes	yes	East	no
36	no	yes	South	no

It is clear that a junk table combines (through a Cartesian product) all the small dimensions into one junk dimension. In other words, a junk dimension can hold more than one low-cardinality attribute that has no correlation with one another.

As compared to the non-junk dimension schema as shown in Option#1 above, all of the low-cardinality attributes are stored into a dimension named, junkDim as shown in the schema below:



The SQL statement to create the *junk dimension* is as followed:

```
Create Table JunkDIM
as select distinct Ensuite, Pool, Aspect_Facing, Spa
from dw.Property1;

Alter Table JunkDim add (JunkID number(2));

Drop Sequence seq_ID;

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

Update JunkDim SET JunkID = seq_ID.nextval;
```

The SQL statement to create *TempFact* table is as followed:

```
Create Table TempFact2
 As SELECT Ensuite,
       Pool,
       Aspect_facing,
       Spa
 FROM dw.property1;
```

The next step is to add a column, called JunkID, in Tempfact2 table:

```
Alter Table TempFact2
Add (JunkID Number(2));
```

Then Tempfact2's JunkID attribute must be filled in with the correct values, which correspond to the values of Ensuite, Pool, Aspect_Facing, and Spa. There are particular TWO ways. The first way is to do an update one-by-one. Since there are 36 junk records, we need to do 36 updates:

```
Update TempFact2
Set JunkID = 1
Where Ensuite = 'yes'
And Pool = 'no'
And Aspect_Facing = 'East'
And Spa = 'yes';

Update TempFact2
Set JunkID = 2
Where Ensuite = 'yes'
And Pool = 'no'
And Aspect_Facing = 'South'
And Spa = 'yes';

…

Update TempFact2
Set JunkID = 36
Where Ensuite = 'no'
And Pool = 'yes'
And Aspect_Facing = 'South'
And Spa = 'no';
```

A better option is to put the Update statement in a loop.

```
Declare
   cursor junkcursor is
        select * from JunkDim;
begin
    for junkcursorrec in junkcursor LOOP
       update Tempfact2
       set JunkID = junkcursorrec.JunkID
       where Ensuite = junkcursorrec.Ensuite
       and Spa = junkcursorrec.Spa
       and Pool = junkcursorrec.Pool
       and Aspect_Facing = junkcursorrec.Aspect_Facing;
    end loop;
end;
/
```

An alternative solution without cursor is to have one update command as follows

```
update TempFact2 tf
set tf.junkID = (select jd.junkID
                 from JunkDIM jd
                 where jd.ensuite = tf.ENSUITE
                 and jd.pool = tf.POOL
                 and jd.ASPECT_FACING = tf.ASPECT_FACING
                 and jd.spa = tf.spa );
```

Query statement to create the *fact table* is as followed:

```
Create Table PropertyFact2 as
Select JunkID, count(*) as Num_of_property
From TempFact2
Group by JunkID;
```

From the junk dimension schema and query above, it shows that junk dimension has a simpler design. This helps to improve the performance and easier to maintain as compared to the non-junk dimension version. For example, non-junk dimension schema has *four* dimension tables that link to the fact table. In a junk dimension schema, it has only *one* dimension table that link to the fact table.

Fact Table:

```
SQL> select * from propertyfact2
     Order by junkid;

    JUNKID NUM_OF_PROPERTY
---------- ----------------
         1   1953
         2   3741
         3   863
         4   1824
         5   940
         6   716
         7   1857
         8   387
         9   1181
        10   384
        11   3730
        12   950
        13   1900
        14   832
        15   764
        16   175
        17   1911
        18   671
        19   3697
        20   358
        21   145
        22   190
```

```
        23   2243
        24   3801
        25   2245
        26   331
        27   1805
        28   1903
        29   334
        30   176
        31   375
        32   4459
        33   334
        34   1854
        35   331
        36   640

36 rows selected.
```