Automated presentation of slowly changing dimensions

Christer Boedeker
Abstract

Automated presentation of slowly changing dimensions

Christer Boedeker

On the subject of data warehousing, a lot of material is available on what needs to be done to maintain a presentation area, but very little on how to do it. This paper presents a structure and process for automatically maintaining and updating a multidimensional presentation area from a fully normalized data staging area. The paper also discusses a theoretical algorithm for finding the best division of slowly changing dimension attributes into several groups based on how they change together.
## Contents

1. Abstract ........................................................................................................................................3

2. Introduction ..................................................................................................................................7

3. Background ...................................................................................................................................9

3.1. Terms ............................................................................................................................................9

3.2. Relational vs. Multidimensional design. ....................................................................................9

3.3. Slowly changing dimensions .....................................................................................................10

3.4. Dimension subdivision ..............................................................................................................11

3.5. Dimension presentation automation ........................................................................................14

4. Dimension subdivision ..................................................................................................................15

4.1. The number of rows when combining three or more attributes ..............................................17

4.2. Exhaustive search ........................................................................................................................18

5. Dimension presentation automation .............................................................................................21

5.1. Metadata repository for automated presentation dimension table creation ..............................21

5.2. Extending the metadata repository with the dimension partitioning ..........................................22

5.3. Automated sub dimension presentation table creation ..............................................................23

5.4. Structure of the staging area ......................................................................................................24

5.5. Natural key to surrogate key mapping .......................................................................................28

5.6. Incremental update of presentation area ....................................................................................30

5.7. Corrected, old dimension data ..................................................................................................33

5.8. Extending the dimension presentation tables ............................................................................36

5.9. Altering the sub dimension partitions .......................................................................................38
5.10. Surrogate key translation table ................................................................. 40

6. Insights and conclusions .................................................................................. 47

7. Bibliography ..................................................................................................... 49

8. APPENDIX ......................................................................................................... 50
   A. Surrogate key join vs. natural key join + date range limit ......................... 50
   B. Surrogate key translation table test trace files ........................................ 64
2. Introduction

This paper is a report on a thesis project in information technology at Uppsala University. The subject area is data warehousing which is a topic of computing science. The paper describes the structures and procedures for staging type 2 slowly changing dimensions and populating them in a presentation layer.

A data warehouse is a large collection of data from a business or comparable operation. Unlike an online transaction processing database, where focus is on single record read and writes, the data in a data warehouse is modeled to support retrieval of large portions of the database with good performance. In a multidimensional data warehouse we speak about dimensions and facts. Dimensions are the categories of data we store in the warehouse, such as customer information and product information. Facts are the measurements of the operation such as the sale of a product to a specific customer at a specific time. Dimensions are naturally more static than facts since it is more common to sell a product than to add a new product to the catalog. What is even less common is the change of an attribute in an already existing product, for example a change of product group or package size. Still this happens sometimes and then we talk about slowly changing dimensions.

The literature on data warehousing is rich with information about how slowly changing dimensions are constructed in the presentation layer of a multidimensional data warehouse. Detailed technical discussions about how the data structures for storing the data and the procedures for moving the data to the presentation layer are much harder to find. The aim of this project was to find and document working procedures for transforming raw attribute data into a dimensionally modeled data warehouse presentation layer.

The paper begins which a short introduction on the nature of a multidimensional data model as opposed to a normalized relational model and then specifically the techniques for presenting slowly
changing dimension attributes. The paper then contains a search for a method to find the best way to distribute attributes between sub dimensions and details data structures and procedures for storing the attributes and automatically copying them to the multidimensional presentation layer. The final outcome are the algorithms needed for planning the presentation of the the slowly changing dimensions and automating the maintenance and population of this presentation from a previously populated staging area.

The project was carried out in an oracle 10g rdbms environment where dummy dimension attribute data was generated. I chose to work in a lab environment with large amounts of dummy data since it is important to verify that the algorithms are practically useable and well performing. As part of the work I studied the available literature on slowly changing dimensions and dimensional modeling, although the algorithms had to be developed from scratch since the available literature doesn’t go into any details regarding implementation of procedures and staging structures.

The goal was to find the procedures necessary to transform the raw attribute data into presentation dimensions supporting flexible and highly performing queries along with the fact data. To achieve this, the necessary steps for the complete transformation were identified and for each step sql code was developed which was then run against the dummy attribute data. The result is a collection of sql statements which can be used to compile a complete package for scd maintenance. The project is focused on the Oracle rdbms, so much of the code is Oracle dependent and some of the reflections are based on the assumption that the underlying database engine is of the classical row based type, wide dimension tables taking a long time to read for instance. The newer more data warehouse centered column based architectures don’t have those read issues, but still most of the observations are true for that type of database engine as well.
3. **Background**

3.1. **Terms**

*Dimension* – A category of information such as person, product, time etc. A collection of dimension entities of the same type.

*Dimension entity* – the entity described in a dimension and identified by the natural key. May be a person, a product, a date etc.

*Dimension attribute* – An attribute of the dimension entity, such as phone number or product package size.

*Natural key* – a unique identifier of a dimension entity. Natural implies that the key originates from outside the database, such as a personal number, but in the context of this paper it may also be generated in the database.

*Surrogate key* – A generated key identifying a row in a slowly changing dimension table.

*SCD, slowly changing dimension* – A dimension whose attributes change only sporadically over time. (1)

<table>
<thead>
<tr>
<th>PK</th>
<th>ID</th>
<th>NUMBER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>123</td>
<td>ABC</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>234</td>
<td>ABC</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>345</td>
<td>CDE</td>
</tr>
</tbody>
</table>

Diagram 1: dimension table structure

3.2. **Relational vs. Multidimensional design.**
In the field of RDBMS data warehouses, two main models for database design have been proposed(2). The multidimensional model, also called the Kimball model after one of its strongest advocates, Ralph Kimball, and the relational model, referred to by some as the Inmon model, after William Inmon(3).

The multidimensional model is a highly denormalized model with large tables of measures or facts in the middle and smaller descriptive dimension tables surrounding the facts in a model resembling a star which gives name to the star schema concept.

The Relational model is the traditional normalized relational database model with normal forms 1, 2, 3 and higher, presenting a highly flexible, non-redundant way of storing data where data integrity can be easily enforced by the .

A multidimensional model is optimized for performance but is not as flexible as the relational model(3). The automation-part of this paper will describe a system where we stage dimensional data in a relational model and present it as needed in a multidimensional model. Staging the data in a relational structure gives the benefit of automatically enforcing integrity on the data and storing the data non-redundantly, whereas presenting the data in a star schema gives the benefits of access speed and a structure that’s easier to understand.

3.3. *Slowly changing dimensions*

A key concept in data warehousing is tracking changes over time. In a multidimensional design we choose which frequently occurring events we want to capture, and then we store information about these in fact tables. The facts, which might for instance be sales transactions or daily account balances, are stored with sparse ids and surrounded by dimension tables giving more verbose
descriptions of the products or customer accounts including inventory information, demographic information etc. This surrounding information can be seen as almost static compared to the facts, but still they do change and we need to capture this somehow. When a dimension’s attributes change only occasionally over time we say that the dimension is slowly changing, a slowly changing dimension or SCD(1).

**SCD types 1, 2 and 3**

Commonly, there are three ways of handling slowly changing dimensions, named SCD type 1, 2 and 3 (1) (4). Type 1 specifies simple overwriting of the changed attributes, making it useful only when historical attribute values are of no interest. In type 3, an additional column is created for storing the historical attribute value, which means that every time a change occurs, a new column is created, making the method most useful for singular attribute changes. The most quoted examples of uses for a type 3 scd are changes in sales territory assignments and changing the category of product(4).

The most commonly used change tracking method is type 2. In this type all dimension rows are marked with a time interval in which the attribute values are valid and for each change a new row is inserted in table. The type 2 SCD is the main focus of this paper.

**3.4. **Dimension subdivision

A dimension such as customer or product might contain hundreds of attributes. Applying a type 2 scd change tracking method to such a dimension means that whenever one attribute value changes a new complete row is inserted into the dimension table with all the other attribute values unchanged. The more attributes in a dimension the greater the chance of one of them changing value at a given point in time so growing a scd type 2 table horizontally, adding columns, also in effect means growing it vertically, increasing the rate at which rows are added.
A way to limit the size of this dimension, discussed among others by Joe Caserta (5), Margy Ross and Ralph Kimball(1), is to split up its attributes into sub-dimensions - groups of attributes whose values are most likely to change together. For a customer dimension, one such group might be the address where often street address, street number and zip code change at the same time when a person moves. The sub-dimensions might be tracked via a main dimension table (Figure 1), increasing the complexity of the design or directly in the fact table (Figure 2), increasing the size of that, already large, table.
A design where every dimension attribute resides in its own table would obviously never
duplicate unchanged attribute values but on the other hand it would create an enormously complex
design with lots of tables creating problems both for the sql optimizer and the client applications.

To find the best setup of sub dimensions the data has to be analyzed and the attributes that
typically change together must be identified. Here is described a theoretical algorithm which finds
the best attribute grouping under certain circumstances based on available data.
3.5. Dimension presentation automation

Since this is the most useful part, the main focus of this paper is on the automatic maintenance and population of the type 2 slowly changing dimension in the data warehouse presentation area. To accomplish this goal we first outline a structure for metadata repository and a dimension staging area which supports this automated handling. As the only components accessing the data in the staging layer is our well controlled procedures, we can utilize a normalized model to eliminate redundancy and better and easier enforce the rules of integrity on the data. Since the integrity has been confirmed in the staging layer and we have total control of the deployment to the presentation layer, we can use a less complex dimensional model here, introducing data redundancy, and still be confident that our data is robust.

We will describe the population of the presentation area from the staging data and we will consider the cases of new dimension entities, updated dimension entities, correction of old attribute values and new dimension attributes. We will use the concept of surrogate keys to map the dimensions to the fact data and compare this against the use of natural keys directly. Furthermore, we will describe a surrogate key translation table to improve performance of queries against the presentation.

The outcome of this treatment is algorithms and structures directly usable to build a working SCD manager package.
4. Dimension subdivision

When storing full history for several attributes in a dimension table, there is a potential for a very large number of rows. If we track changes to attributes on a daily basis, every day an attribute changes there will be a new row in the dimension table and with very wide rows this might result in a lot of data. Normally disk space is not expensive so the cost of the storage is not an issue. However the time required reading a large amount of data is greater than that required reading less data, so performance is negatively influenced by an increase in data.

By placing dimensions attributes into separate sub-dimensions the situation can be improved. Whenever an attribute changes in one sub-dimension there will be a new row in that table, but the other sub-dimension tables will remain unchanged so the amount of data is only increased by a fraction of a full dimension row.

Since every attribute change results in a new row, the number of rows in a dimension table is equal to the number of distinct attribute change dates. If we split up a dimension into sub dimensions the number of rows in each sub dimension is equal to the number of distinct attribute change dates in that sub dimension.

Given a master dimension, a set of type 2 attributes of that dimension and a number of desired sub dimensions, the goal of the algorithm is to find the distribution of attributes between the sub dimensions that give minimum number of total rows.

Assumptions: Attributes that tend to change at the same time should be put in the same sub dimensions. Attributes that tend not to change at all can be put in any sub dimension.

Combining two or more attributes in a sub dimension, the worst possible combination leads to a row count which is the total sum of the change counts for each individual attribute. The best possible
combination gives a row count which is equal to the change count for the attribute which changes the most.

Step 1. for each pair, A + B, of attributes of a dimension, values belonging to the same dimension value (same natural key), count how many times they change on the same day (CS), how many times A changes on a day B doesn’t change (CA) and how many times B changes on a day A doesn’t change (CB). Change here also includes first insert.

\[ CTA = CS + CA = \text{total change count for } A \]
\[ CTB = CS + CB = \text{total change count for } B \]

Pairing up A and B in a sub dimension, the number of type 2 rows of that table would be \( CS + CA + CB \). With \( CM = \max(CTA,CTB) \) and \( CT = CS + CA + CB \), the higher \( CQ = CM / CT \), the better this match is.

\( CQ = 1 \) means a perfect match since:

If both attributes always change at the same time we have:

\[ CA = 0 \]
\[ CB = 0 \]
\[ CS > 0 \]
\[ CTA = CS + 0 = CS \]
\[ CTB = CS + 0 = CS \]
\[ CM = \max(CS,CS) = CS \]
\[ CT = CS + 0 + 0 = CS \]
\[ CQ = CS / CS = 1 \]

If attribute A changes on its own, but B only changes at the same time as A we also have a perfect match:

\[ CA > 0 \]
\[ CS \geq 0 \]
\[ CB = 0 \]
\[ CTA = CS + CA \]
\[ CTB = CS + 0 = CS \]
\[ CM = \max(CS + CA, CS) = CS + CA \]
\[ CTA = CS + CA + 0 = CS + CA \]
\[ CQ = CS + CA / CS + CA = 1 \]

Equally many independent changes and no common changes would be the worst possible match:

\[ CS = 0 \]
\[ CTA = 0 + CA \]
\[ CTB = 0 + CB = 0 + CA = CA \]
\[ CT = 0 + CA + CA = 2 \times CA \]
\[ CM = \max(CA, CA) = CA \]
\[ CQ = CA / 2 \times CA = 1/2 \]

No changes at the same time but different amount of changes of A and B is not the worst possible match, and in fact goes towards a perfect match as one of the attributes goes toward 0 changes:

\[ CS = 0 \]
\[ CB > 0 \]
\[ CB < CA \]
\[ CB = CA - n, 0 < n < CA, n \text{ goes towards } CA \text{ as } CB \text{ goes towards } 0 \]
\[ CTA = 0 + CA \]
\[ CTB = 0 + CB = 0 + CA - n \]
\[ CT = 0 + CA + CA - n = 2 \times CA - n \]
\[ CM = \max(CA, CA - n) = CA \]
\[ CQ = \frac{CA}{2 \times CA - n} = \frac{1}{2 - \frac{n}{CA}} \]
\[ \frac{1}{2} < \frac{1}{2 - \frac{n}{CA}} < \frac{1}{2 - \frac{CA}{CA}} = \frac{1}{2 - 1} = 1 \]

4.1. The number of rows when combining three or more attributes
Given \( n \) attributes of a dimension we will have \( \frac{n(n-1)}{2} \) possible combinations of two attributes.

### 4.2. Exhaustive search

One way of finding the optimal distribution of attributes is to analyze all possible combinations and count the number of rows each combination results in.

First of all we need to find and enumerate all possible combinations of attributes. In one combination, each attribute might be included or it might be excluded. Let \( A \) be the set of all attributes: \( A = \{a_1...a_n\} \) and \( B \) be the set \( \{0,1\} \). If \( S \) is a subset of \( A \), the function \( f_B(i) \) maps to a member of \( B \):

\[
f_B(i) = \begin{cases} 
0 & \text{if } a_i \text{ is not a member of } S \\
1 & \text{if } a_i \text{ is a member of } S 
\end{cases}
\]

This function maps each set \( S \) to a word of length \( N \) consisting of \(|S|\) ones and \( N-|S| \) zeroes. Since this representation coincides with the standard binary numeral system there is a ready ordering of these strings. Efficient set operations are also already available in the form of the bitwise operations of computers: Let \( f_B(S) \) map a set \( S \) to its binary ordering number. If every attribute of \( A \) is included in one and only one set \( S_n \) i.e. \( \{S_1...S_n\} \) is a partition of \( A \) and \( S_1 \cup S_2 \cup ... \cup S_N = A \) and \( S_i \cap S_j = \emptyset \) for any \( i \) and \( j \) in \( 1...N \) we have \( f_B(S_1) OR f_B(S_2) OR ... OR f_B(S_N) = N - 1 \) and \( f_B(S_i) AND f_B(S_j) = 0 \) for any \( i \) and \( j \) in \( 1...N \)

Let \( f_{rows}(x) \) be a function mapping a set of attributes \( S \) to the number of rows a database table with attributes \( S \) and scd type 2 tracking would have. The problem we are trying to solve can now be expressed: Find the partition \( P = \{S_1...S_n\} \) of \( A \) which minimizes

\[
\sum_{i=0}^{n} f_{rows}(S_i) \times |S_i|
\]

This sum can be regarded as a “table area” and we denote it \( T_P \).
The next step is to find all $2^N$ values for $f_{\text{row}}$. With the dimension staging table `dim_stg` a query for finding the number of rows for a given combination is:

```sql
select natural_key, count(distinct lower_date_limit)
from dim_stg s
where s.attribute_id in (s1, s2, .., sn)
group by natural_key
```

This select has to be run $2^N$ times and the results saved. With a table containing all attribute combinations only one query is necessary:

```sql
select v_set_num, comb_num, sum(dates)
from (select comb_num, natural_key, count(distinct lower_date_limit) as dates
from comb_members m, dim_stg s
where s.attribute_id = m.mem_num
and set_num = v_set_num
group by comb_num, natural_key)
group by comb_num
```

Having collected the row numbers, the next step is to find the partition of attributes which minimizes $T_P$. Oracle SQL doesn’t supply a bitwise or function, but following the rules of AND, OR and binary addition:

\[
\begin{align*}
0 \land 0 & \rightarrow 0, \quad 1 \land 0 \rightarrow 0, \quad 0 \land 1 \rightarrow 0, \quad 1 \land 1 \rightarrow 1 \\
0 \lor 0 & \rightarrow 0, \quad 1 \lor 0 \rightarrow 1, \quad 0 \lor 1 \rightarrow 1, \quad 1 \lor 1 \rightarrow 1 \\
0 + 0 & \rightarrow 0, \quad 1 + 0 \rightarrow 1, \quad 0 + 1 \rightarrow 1, \quad 1 + 1 \rightarrow 0, \text{ carry 1}
\end{align*}
\]

We can see that the rules for OR and addition match except when $b_x \land b_y = 1$, so for integers $X$ and $Y$:

\[
X \land Y = 0 \rightarrow X \lor Y = X + Y
\]

Given 8 attributes, SQL can be used to find the best two-set partition with this select:
select t1.comb_num,
    t2.comb_num,
    t1.tot_rows * t1.num_attr + t2.tot_rows * t2.num_attr
from combinations t1, combinations t2
where t1.comb_num + t2.comb_num = 255
    and bitand(t1.comb_num, t2.comb_num) = 0
    and t1.comb_num < t2.comb_num
order by 3

And for a three-set partition:

select t1.comb_num,
    t2.comb_num,
    t3.comb_num,
    t1.tot_rows * t1.num_attr + t2.tot_rows * t2.num_attr + t3.tot_rows * t3.num_attr
from combinations t1, combinations t2, combinations t3
where t1.comb_num + t2.comb_num + t3.comb_num = 255
    and bitand(t1.comb_num, t2.comb_num) = 0
    and bitand(t1.comb_num, t3.comb_num) = 0
    and bitand(t2.comb_num, t3.comb_num) = 0
    and t1.comb_num < t2.comb_num and t2.comb_num < t3.comb_num
order by 4

Partitions with more sets are found analogously and with n attributes 255 is replaced by $2^n - 1$

The number of sums calculated for finding the number of rows when we have n attributes is $O(2^n)$ so with a realistic number of attributes this exhaustive search method cannot be used. To find a faster method for finding the optimal partition we need more advanced data mining techniques, which are beyond the scope of this paper.
5. Dimension presentation automation

The concept of automated maintenance of slowly changing dimensions, in the context of this paper, involves keeping the raw data in a normalized staging area and then to create or recreate and populate or repopulate the dimension tables in the presentation area as needed in periodic batch jobs. In this section we will present the methods needed for automating the maintenance of the data warehouse presentation area.

5.1. Metadata repository for automated presentation dimension table creation

In order to automatically construct the presentation area, information about dimension tables and attributes must be stored in a metadata repository. At this point we need basic information about the existing dimensions and their attributes. Since we don’t want to limit the possible number of attributes in a dimension, the relationship between dimension and attribute is 1-to-many so a normalized structure will contain two entities, a dimension entity and an attribute entity.

![Diagram 2, Basic dimension metadata]

In order to create the actual tables in the presentation area we need a procedure for transforming the metadata information into a table creation statement. The simplest form of this procedure needs to find the name of the dimension table and the names and data types of the attributes of that dimension and create a working sql statement.
declare
  vs_sql varchar2(10000);
  vs_dim_name varchar2(30) := '<Dimension name>'; begin
  vs_sql := 'create table ' || vs_dim_name || ' (surrogate_key integer,
    start_date date, end_date date';
  for i in (select ', ' || attribute_name || ' ' || attribute_type trow from
    master_dim_meta m, attr_meta a
    where m.master_dimension_id = a.master_dimension_id
    and m.master_dimension_name = vs_dim_name)
  loop
    vs_sql := vs_sql || i.trow;
  end loop;

  vs_sql := vs_sql || ');';

end;

query 1: procedural code for constructing a create table-statement from the metadata structure

Depending on the metadata, the code will produce a string something like this, which can be used by a maintenance procedure to create the presentation table:

create table TESTDIM (surrogate_key integer, start_date date, end_date date,
  ATTRIBUTE1 varchar2(32), ATTRIBUTE2 varchar2(32), ATTRIBUTE3 varchar2(32),
  ATTRIBUTE4 varchar2(32), ATTRIBUTE5 varchar2(32));

5.2. Extending the metadata repository with the dimension partitioning

For the wide dimensions with many change tracked attributes the algorithm of the previous section has been applied. Now the attributes belong in sub dimensions instead of directly in the master dimensions. A new structure for tracking this is needed. A sub dimension entity needs to be added with a many-to-1 relationship to the master dimension so that one master dimension can be divided into several sub dimensions. Each dimension attribute needs to be mapped to a sub dimension but an attribute naturally belongs to a master dimension so for integrity we need to retain the direct relationship between master dimension and attributes.
The sub dimension metadata table is added and a bridge table to track which attributes belong to which sub dimension. With this mapping structure we can easily move attributes between dimension tables and recreate the presentation area if the requirements change. Since the bridge table implies a many-to-many relationship between attributes and sub dimensions, we can also include an attribute in several sub dimensions. The master dimension foreign keys in the bridge table are added to avoid the possibility of adding an attribute of one master dimension to a sub dimension of another master dimension.

Additional columns are added to enable the automated creation of sub dimension presentation table with customizable names for all columns, making it easier to use the presentation area in, for instance, a graphical reporting tool. We also have additional columns for creating constraints and specifying additional storage parameters for the table. By adding more columns to sub_dim_meta is it also possibly to allow creation of indexes and foreign keys etc.

**5.3. Automated sub dimension presentation table creation**

For creating all sub dimensions of a given master dimension all sub dimension meta data, table_name, key names, date column names and any extra information for creation of the table, must be fetched from sub_dim_meta.

query 2: Get sub dimension table base meta data
select sdm.sub_dimension_name, sdm.primary_key_name, sdm.start_date_name, sdm.end_date_name, mdm.natural_key_name, sdm.constraints_ddl, sdm.storage_params
from sub_dim_meta sdm, master_dim_meta mdm
where sdm.master_dimension_id = mdm.master_dimension_id
and mdm.master_dimension_name = <master dimension name>

This query will return one row for each sub dimension of the master dimension of interest. With this query as a basis we can create a procedure which loops through the resulting sub dimensions and fetches its attributes. The attributes are fetched, based on sub dimension id, from the attribute metadata table, which contains the column name and data type, joined to the attribute to sub dimension bridge table, which identifies the sub dimension(s) for each attribute.

query 3: Get sub dimension attribute meta data

select am.attribute_name, am.attribute_type
from sub_dim_to_attr_meta sdam, attr_meta am
where sdam.attribute_id = am.attribute_id
and sdam.sub_dimension_id = vd_subdim_id

With this information we can automatically build full create table statements for the sub dimension tables, which can be executed in a maintenance procedure and place the table in the presentation area like in section 4.1.

5.4. Structure of the staging area

To be able to flexibly maintain the presentation area, the staging area needs to retain a complete history for all change tracked attributes. Most probably a surrogate key is used to identify the specific valid instance of a dimension entity in the fact table. Therefore we also need to keep track of the assigned surrogate keys to keep the relationship with the fact tables valid. We start with a design like this:

<table>
<thead>
<tr>
<th>DIM_STG</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
</tr>
<tr>
<td>PK</td>
</tr>
<tr>
<td>PK</td>
</tr>
</tbody>
</table>
| attribute id
| natural key
| lower date limit
| surrogate key
| value   |
Basic population of the presentation area type 2 scd from the staging table.

Every dimension row represents a dimension entity during a time interval. To populate a dimension table we first need to find all time intervals for each dimension entity. We observe that whenever on or more of the attributes of a dimension entity changes, a new row for that entity must be created in the dimension table since each attribute value in a row must be valid during the whole interval. Thus, we need a query to find all distinct change date values for every dimension entity.

This SQL query will find all change dates for a dimension by asking for every distinct lower date limit for each natural key:

query 4: find change dates

```sql
select distinct natural_key,
               lower_date_limit
from dim_stg s, master_dim_meta m, attr_meta a
where m.master_dimension_id = a.master_dimension_id
  and a.attribute_id = s.attribute_id
  and m.master_dimension_name = '<DIMENSION NAME>'
```

Using this query as a subquery we can find both start time and end time of the intervals by using the start time from the following row as the end time of the current with Oracle analytic functions, creating a half open interval. The last row will have no such value so the resulting null is replaced by 12/31/9999 using the coalesce function:

query 5: find intervals

```sql
select natural_key,
       lower_date_limit,
       coalesce(lead(lower_date_limit, 1) over(partition by natural_key order by lower_date_limit), to_date('99991231', 'YYYYMMDD')) as upper_date_limit
from (select distinct natural_key,
lower_date_limit
from dim_stg s, master_dim_meta m, attr_meta a
where m.master_dimension_id = a.master_dimension_id
and a.attribute_id = s.attribute_id
and m.master_dimension_name = '<DIMENSION NAME>')

Since query 5 maps all possible intervals for a dimension entity, what remains is to place the correct attribute values in the correct intervals. We need to find the valid time intervals for all attribute values. Each of these values will map to one or more consecutive intervals from query 5. So, for each combination of natural key and attribute id, find and order all lower date limits into an interval map. Again analytic functions are used to take the start date of the next row as the end date of the current.

query 6: attribute intervals

select s.attribute_id,
       s.natural_key,
       s.value,
       s.lower_date_limit,
       coalesce(lead(lower_date_limit, 1) over(partition by natural_key, s.attribute_id order by lower_date_limit), to_date('99991231', 'YYYYMMDD'))
       as upper_date_limit
from dim_stg s, attr_meta a, master_dim_meta m
where m.master_dimension_id = a.master_dimension_id
    and a.attribute_id = s.attribute_id
    and m.master_dimension_name = '<DIMENSION NAME>'

By combining query 5 and query 6 we can now put all attribute values into their correct dimension entity time intervals. The join between the two queries will give a result set with one attribute value per row. To be able to fill a dimension presentation table, which has one row per dimension entity and time interval, we need to pivot the result to put all coinciding attribute values on the same row. To do this we need to know which attribute id:s maps to which dimension table columns. In practice this information will be gathered from the metadata tables to dynamically construct the query, but, for the sake of simplicity, here we will assume that attribute id:s are 1,2,3 etc. The pivoting is done by grouping on natural key and time interval. We use the case function to
extract the correct values: for the first value column, when attribute id is one we return the attribute value else we return null. Since database constraints assert that we have only one value for attribute 1 per entity in any given time interval, we will have a list of nulls and a single non-null value. Now, on this list, we need to apply an aggregate function which will return the only non-null value. There are many such functions, but here we will choose the max function.

query 7: basic presentation construction

```sql
select v.natural_key, d.lower_date_limit, d.upper_date_limit,
   max(case v.attribute_id when 1 then v.value else null end),
   max(case v.attribute_id when 2 then v.value else null end),
...
from
(select s.attribute_id,
   s.natural_key,
   s.value,
   s.lower_date_limit,
   coalesce(lead(lower_date_limit, 1) over(partition by natural_key, s.attribute_id order by lower_date_limit), to_date('99991231', 'YYYYMMDD'))
as upper_date_limit
from dim_stg s, attr_meta a, master_dim_meta m
   where m.master_dimension_id = a.master_dimension_id
       and a.attribute_id = s.attribute_id
       and m.master_dimension_name = 'DIM_D') v,
(select natural_key,
   lower_date_limit,
   coalesce(lead(lower_date_limit, 1) over(partition by natural_key order by lower_date_limit), to_date('99991231', 'YYYYMMDD')) as upper_date_limit
from { select distinct natural_key,
   lower_date_limit
   from dim_stg s, master_dim_meta m, attr_meta a
   where m.master_dimension_id = a.master_dimension_id
       and a.attribute_id = s.attribute_id
       and m.master_dimension_name = 'DIM_D'}) d
where v.natural_key = d.natural_key
and v.lower_date_limit <= d.lower_date_limit
and v.upper_date_limit > d.lower_date_limit
group by v.natural_key, d.lower_date_limit, d.upper_date_limit
```
Combining the data from the dimension metadata with this structure, this would be the type of query created by the procedural code in the batch process for updating the presentation area.

5.5. **Natural key to surrogate key mapping**

The surrogate keys could be assigned to the attributes in the dim_stg table but the real relationship is one surrogate key for each combination of natural key, sub dimension and time interval. Since we strive for a normalized structure in the staging area to make it easier to maintain data integrity, a surrogate key mapping table is needed.

![Diagram showing the relationship between SUB_DIM_META and DIM_NK_TO_SK tables]

Now there exists an implicit integrity constraint between attributes and surrogate keys which can’t be automatically enforced in the presentation table. Care must be taken to ensure that the attributes are presented with the corresponding correct surrogate key.

First we need a way to populate the mapping table based on the attribute values we have in the attribute staging table. What we need is one surrogate key for each interval of each sub dimension entity. We populate the table with this query:

**query 8: create new surrogate key values**

```sql
select sub_dimension_id, natural_key, lower_date_limit, surrogate_key_seq.nextval from (  
  select distinct dam.sub_dimension_id, s.natural_key, s.lower_date_limit  
  from dim_stg s, sub_dim_to_attr_meta dam  
  where s.attribute_id = dam.attribute_id)
```
We can now change query 4 to select from this new table instead since here we have a 1-to-1 relationship to the dimension presentation table: for every sub dimension table we have one row per natural key and date value.

**Incremental update of mapping table**

When new date values arrive in the dimension staging table the surrogate key table needs to be updated to include the new dates. We need a way to find all new intervals, and those are the ones that are present in the staging table but not in the surrogate key table. A simple solution is to add this row to query 8:

```
and (dam.sub_dimension_id, s.natural_key, s.lower_date_limit) not in
(select sub_dimension_id, natural_key, lower_date_limit from dim_nk_to_sk
dns)
```

This will exclude values for rows already present in the table. For performance reasons this query might need to be improved when these tables grow large, since a join between two large tables is potentially very slow(6). The concept of batch id introduced in the section on incremental update of the presentation area is also applicable here.

**Issues with new data and sub dimension restructuring**

With this mapping table structure it is necessary that for every combination of natural_key and lower_date_limit in dim_stg there is a surrogate key in dim_nk_to_sk for the attribute’s corresponding sub_dimension_id. If an attribute is moved between sub dimensions and introduces new date intervals in the new sub dimension, the surrogate keys need to be regenerated. This leads to a need to change the keys in the fact tables that store the surrogate keys directly. To be able to move attributes between sub dimensions easily we need a solution to this problem. A possible solution will be suggested further on.

**Time interval in data staging tables**

Both the dim_stg table and the new dim_nk_to_sk table store only the start_date for their validity intervals. The implied rule is that each row is valid until the next value for that attribute or
sub dimension appears. The reason for this is that it becomes much easier to keep the data valid since it’s not possible to create overlapping intervals or “holes”. However, it becomes a bit trickier to tell the end of the interval for a specific row. We solve this by creating views.

We use the same principle as in query 5 and query 6 and find the end date for the interval with an analytic function. The lead function finds the start date from the following row of the same attribute with the same natural key and uses this value as end date for the current row. The end date for each attribute value can be added to the data from the dim_stg table by using a view based on this query. The view will be called dim_stg_view:

query 9: Add upper_date_limit to dim_stg data

```
select ds.*,
    coalesce(lead(lower_date_limit, 1)
    over(partition by natural_key, attribute_id
        order by lower_date_limit),
    to_date('99991231', 'YYYYMMDD')) upper_date_limit
from dim_stg ds
```

Similarly the end date for the validity of each surrogate key can be added to the mapping table using this view, dim_nk_to_sk_view:

query 10: Add upper_date_limit to surrogate key table data

```
select dns.*,
    coalesce(lead(lower_date_limit, 1)
    over(partition by natural_key, sub_dimension_id
        order by lower_date_limit),
    to_date('99991231', 'YYYYMMDD')) upper_date_limit
from dim_nk_to_sk dns
```

### 5.6. Incremental update of presentation area.

Normally the dimension presentation is correct up to the last update and at regular intervals new values for attributes that have changed since the last presentation update arrive in the staging area.
These new values have to be appended to the dimension presentation and thus need to replace the last “current” values.

**Verify that the new values are indeed “last” values**

For the normal incremental update to succeed all new values must be guaranteed to have a valid date later than any attribute value of the same (sub)dimension already in the warehouse. This can be achieved by controlling the procedures inserting the data to the staging area but it’s also a good idea to provide a method to check the data for this consistency. By checking that any new rows that would be put into the dim_nk_to_sk table have value dates higher than any existing rows for that combination of sub dimension and natural key we can guarantee that all new values are the most current ones. Finding out which rows are new could be achieved by checking the dimension staging rows against the rows in dim_nk_to_sk, although an easier way would be to mark the rows in the dimension staging table with an id identifying the inserting batch. Knowing which batch number was last sent to the presentation area we see that all rows with higher batch numbers are new. For this purpose we add the batch_id column to dim_stg and dim_nk_to_sk and a new table to keep track of batches:

![Diagram 3: batch_id in dimension staging](image)
The etl batch process set batch status depending on where the child values are in the process, whether they have been assigned surrogate keys or if they have been transferred to the presentation layer.

By joining all combinations of sub dimension id:s and natural keys from the new batch of staging values to the rows in dim_nk_to_sk and see if any rows already exist with a date equal to or later than the new values we will find any values that break the rule. This query will return the offending rows from the staging table and their assigned sub dimension:

**query 11: Find date-offending new attribute values**

```sql
select distinct s.*, dam.sub_dimension_id
from dim_nk_to_sk dns, dim_stg s, sub_dim_to_attr_meta dam
where s.attribute_id = dam.attribute_id
and s.batch_id = <new batch id>
and dam.sub_dimension_id = dns.sub_dimension_id
and s.natural_key = dns.natural_key
and s.lower_date_limit <= dns.lower_date_limit
```

When the new batch has been cleared, surrogate keys for the new values can be created as usual. After this, when updating the presentation area, the rows that are succeeded by new rows have to have their end dates updated since they have 9999-12-31 as end date at this point. Since dim_nk_to_sk is now updated with the new sub dimension, natural key, lower date limit rows we can use those new rows to find the presentation rows with the same sub dimension and natural key and an end date of 9999-12-31.

**query 12: Find old "current" rows in presentation**

```sql
select sd.* from <sub_dim table> sd, dim_nk_to_sk dns where dns.batch_id = <batch id> and sub_dimension_id = <sub dim>
and sd.dim_1_nk = dns.natural_key
and sd.dim_1_b_end = to_date('99991231', 'YYYYMMDD')
```

Again, this query has to be automatically constructed since it accesses a dynamically created presentation table. By including the rowid from the presentation table and storing the result in an
array the rows can be easily updated in pl/sql. Also we have to allow for multiple new values for the same natural key in the same sub dimension so the old “current” row has to end at the first new row which we solve by grouping by dimension entity and selecting the minimum new lower date limit:

**query 13: update old "current" rows in presentation**

```sql
declare
    rids rowid_tab;
    dats date_tab;
begin
    select min(dns.lower_date_limit), sd.rowid
    bulk collect into dats, rids
    from <sub dim table> sd, dim_nk_to_sk dns
    where dns.batch_id = <batch id> and sub_dimension_id = <sub dim>
    and sd.dim_1_nk = dns.natural_key
    and sd.dim_1_b_end = to_date('99991231', 'YYYYMMDD')
    group by sd.rowid;

    forall i in 1..rids.count
        update <sub dim table> sd set sd.dim_1_b_end = dats(i) where sd.rowid = rids(i);
end;
```

The select in the procedure also exploits the fact that there is a one-to-one relationship between natural key and rowid from the sub dim table – i.e. there is only one row in the sub dim table with end date 99991231 for each natural key – and groups by the rowid instead of the natural key. The rowid is stored and used in the subsequent update to quickly find the rows for update.

The final stage is sending the new rows to the presentation layer with last end date 99991231.

Following this procedure for data that can’t be appended at the end of the presentation table will result in corrupted data so we need another way of handling changes to data in the middle of the table.

**5.7. Corrected, old dimension data.**
Sometimes new information is discovered which warrants a historical change of the dimension attribute values. For instance, there might have been an error somewhere which is discovered and data needs to be corrected. If only the values change there is no real problem, but if the correction is to the time intervals when the values are valid the correction mechanism is more complicated since fact rows have already been assigned surrogate keys based on the time periods and new surrogate keys might have to be created and updated in the fact tables.

However, we start by leaving the fact tables aside and only concerning ourselves with the dimension tables. By controlling the dml towards the dim_stg table we can separate updates from inserts. An updated value in dim_stg requires no new surrogate key, only a direct update of the corresponding row(s) in the presentation table. Inserts however, might mean values for a new natural key, new values for an existing natural key on the same date as other attribute values or new date values for an existing natural key and these cases must be treated differently.

i. A new natural key means new surrogate key values but no conflicts with existing data since the whole dimension entity is necessarily new and therefore there can’t exist any later “last” row. These rows require no special consideration.

ii. New values on the same date as other attribute values don’t require any new surrogate keys since the surrogate keys were already generated when the other values arrived. However, the corresponding presentation rows must be updated. For each attribute value several presentation rows might have to be updated since the new value is valid until the start date of the next value for this attribute. The procedure for this case is similar to updates of the dim_stg table.

iii. New date values for an existing natural key will require new surrogate keys and also require that the presentation layer be updated for the rows preceding the new rows. Like case ii, several presentation rows succeeding the new value might have to be updated in addition to the new row that has to be inserted.
The first step for case iii is to create new surrogate keys for those new values that don’t have any matching row in the surrogate key mapping table. This query fetches the values from a new batch and, by use of the inner query, excludes those where the natural key and time interval elicit a match in the surrogate key table:

**query 14: Create surrogate keys for staged batch**

```sql
select sdam.sub_dimension_id, ds.natural_key, ds.lower_date_limit,
surrogate_key_seq.nextval, ds.batch_id
from dim_stg ds, sub_dim_to_attr_meta sdam where ds.batch_id = <batch id>
and ds.attribute_id = sdam.attribute_id and not exists
  (select null from dim_nk_to_sk_view dns
   where dns.sub_dimension_id = sdam.sub_dimension_id
   and dns.natural_key = ds.natural_key
   and dns.lower_date_limit = ds.lower_date_limit)
```

New surrogate keys mean new rows in the presentation table and an update of the end date value of the rows preceding the new ones. By observing that the rows in the surrogate key mapping table followed by rows from the new batch correspond to the ones that should be updated in the presentation we can construct the following query. In the inline view, for each row, the lead function fetches the batch_id from the next row entry of the same sub_dimension entity. We then use filter on this result to find the rows that are followed by an entry of the new batch. We also filter out those rows that are themselves part of the new batch:

**query 15: Find end dates to update**

```sql
select surrogate_key, upper_date_limit from (  select dns.*, lead(batch_id, 1) over (partition by natural_key,
  sub_dimension_id order by lower_date_limit) lead_batch
from dim_nk_to_sk_view dns)
where lead_batch = <batch id> and (batch_id != <batch id> or batch_id is null)
```

The analytic function lead is used to find the batch_id of the consecutive row. Since two consecutive rows might be inserted in the latest batch, rows followed by a row from the last batch might themselves belong to this batch so we must also filter out these. In case some procedure is
allowed to insert rows in the surrogate key mapping table with a null batch id we must explicitly allow for these as well, since they would otherwise be filtered out by the preceding batch id check.

For both case i and iii the new rows to be inserted can be found by filtering on the new batch id in dim_nk_to_sk. When a new attribute value is valid over several consecutive presentation table rows, the keys are found using this query which finds all existing intervals in which the new value is valid, by looking for all rows in dim_nk_to_sk_view where the start date is within the valid date range of the new values:

```sql
select dns.sub_dimension_id, dns.surrogate_key, ds.value
from dim_stg_view ds, sub_dim_to_attr_meta sdam, dim_nk_to_sk_view dns
where ds.batch_id = <batch id>
and dns.batch_id != <batch id>
and ds.attribute_id = sdam.attribute_id
and dns.sub_dimension_id = sdam.sub_dimension_id
and dns.natural_key = ds.natural_key
and dns.lower_date_limit >= ds.lower_date_limit
and dns.lower_date_limit < ds.upper_date_limit;
```

For inserts the first row where dns.lower_date_limit = ds.lower_date_limit can be omitted since its newly inserted with the correct values anyway.

**Deleting rows from the dim_stg table**

Even though deleting values might remove the need for a specific surrogate key value, it’s not strictly necessary to remove the key since two consecutive rows in the presentation area with all values identical is not a problem. Leaving the rows also removes the issue of having to update the foreign keys in the fact tables, so we will choose not to delete any rows in the presentation. What happens when a value is deleted is that all rows displaying that value are updated to display the previous value instead. The methods for accomplishing this have already been described above.

**5.8. Extending the dimension presentation tables**

Whenever new attributes are added to a dimension and assigned to a sub dimension we need to have a way of extending the presentation table of that sub dimension. To do this automatically we
need to look at the presentation table in the Oracle data dictionary and compare it to the table definition we get from our own metadata repository. We find the natural key column name in master_dim_meta, the natural key name and the date range names in sub_dim_meta and all attribute column names in attr_meta. After we have decided which sub dimension to update, the query to find the columns to be added to the presentation table compares our metadata to the oracle supplied user_tab_columns table, which contains the name of all table columns in the database, finding the rows missing from the presentation table. We do this by creating a list of what we should have, according to the metadata repository. From this list, we subtract what we actually have in the database according to the Oracle data dictionary:

query 16: Find missing columns in presentation table

```
select upper(sdm.primary_key_name) from sub_dim_meta sdm where sdm.sub_dimension_id = <sub dim>
union
select upper(sdm.start_date_name) from sub_dim_meta sdm where sdm.sub_dimension_id = <sub dim>
union
select upper(sdm.end_date_name) from sub_dim_meta sdm where sdm.sub_dimension_id = <sub dim>
union
select upper(mdm.natural_key_name) from sub_dim_meta sdm, master_dim_meta mdm where sdm.sub_dimension_id = <sub dim> and mdm.master_dimension_id = sdm.master_dimension_id
union
select upper(am.attribute_name) from attr_meta am, sub_dim_to_attr_meta sdam where am.attribute_id = sdam.attribute_id and sdam.sub_dimension_id = <sub dim>
minus
select column_name from user_tab_columns t, sub_dim_meta sdm where t.table_name = upper(sdm.sub_dimension_name) and sdm.sub_dimension_id = <sub dim>;
```

With this information we create an alter table statement to add the new columns and following this, given that values for the new attributes have been staged, we use the procedures described in the chapter on merging old data into the presentation area to populate the new columns.
5.9. **Altering the sub dimension partitions**

Moving attributes between sub dimensions means dropping the columns in one sub dimension and adding them in another. This change might be done when attribute changes have started occurring differently and an attribute now fits better in another subdimension. The adding of a column to a table has been previously discussed and also deleting rows which would happen for change dates unique to the moved attribute. However, to make sense of the attribute move, the attribute-unique change dates must in this case be removed from the original sub dimension to minimize the storage space requirements.

We have now discussed several situations where the surrogate keys in the fact tables need to be updated because of data changes in the dimension tables. Because of the possible immensity of the fact tables this might become an insurmountable task. Here, we will look for a solution which will allow the fact tables to remain unchanged yet still allow mutating sub dimension tables.

**Join on natural key and date range instead of surrogate keys**

Instead of joining the fact tables to the dimensions using the surrogate keys, it is possible to store the natural keys in the fact tables instead and do a join including the fact table date stamp, thus finding the correct match between fact rows and dimension rows:

**query 17: Join fact to dimension on natural key**

```sql
select ... 
from fact_table f, sub_dim d 
where f.natural_key = d.natural_key 
and d.start_date <= f.date 
and d.end_date > f.date 
```

Theoretically this would be a perfectly good solution but since this is not a simple equijoin (a join using only the equality operator in the predicate), it’s very hard for the query optimizer to generate an efficient execution plan for this query. One of the greatest advantages of a dimensional data warehouse model is the star join between a foreign key of the fact table and the primary key of the
dimension table. This join is very easy to optimize and allows an efficient join between a fact table and several dimension tables.

Setting up a fact table and a dimension table for test we can demonstrate the differences in execution with the different join principles.

3-way star schema access:

![Diagram 4: 3-way star schema](image)

After creating data for these tables, 130000 rows in each dimension table and 19200000 rows in the fact table, joins according to the two different principles were made between the fact table and all three dimension tables, filtering on the int_property columns of the dimensions.

The first set of queries limited the number of returned rows to 10748 rows. In this case the natural key join took about 25 times as long as the primary key join. From the oracle optimizer execution plans we can see that the primary key join let the oracle optimizer do an efficient star join where only bitmap indexes where accessed until the resulting 10748 rows where found, and only
then were they accessed from the fact table. With the natural key join the optimizer had trouble estimating the expected rows returned and made a much more costly plan which resulted in hash joins and full scans of all the involved tables, specifically reading through all 19200000 rows of the fact table. (~65000 disk read operation versus none for the primary key join).

Changing the predicates to be a little more selective so the query fetches 1466 rows we still have the same star query plan for the primary key joins. In the natural key join, the method is still hash joins and full table scans, albeit with a different order between the table reads. The number of disk reads in the slow query is about the same as in the first example but the pk join again gets a 100% cache hit rate and thus doesn’t read from disk at all. The elapsed time ratio in this example is even greater, with the nk query taking more than 30 times as long as the pk query.

Complete code and results of these examples can be found in the appendix.

DBMS improvement suggestion

Looking at the logical schema design, it is clear that all necessary information for the join is available in the combination of the natural key and the date key. This information, however, can’t currently help the optimizer in generating an efficient query plan. Oracle could choose to develop the star join further and add some possibility of making this information available to the optimizer. There would be a need for a primary key which implies a date range: declaring a date column date range in the key would say that, not only does the value have to be unique, but we would also have the added meaning that the row is valid until the next higher date value. An index lookup for this would only need to find the leaf node with the greatest value smaller than or equal to the lookup value, very close in performance potential to an equijoin and certainly very much faster than the range comparison you have now where no uniqueness can be applied so basically all smaller values have to be checked.

5.10. Surrogate key translation table
To make the attributes more independent from the fact tables we introduce the surrogate key translation table, mapping a single surrogate key in the fact table to multiple sub dimension tables.

Diagram 5: Surrogate key translation table

This lookup table maps a key to a set of sub dimension rows that are valid at the same time. Each time a sub dimension attribute changes and a new row is created in the sub dimension table a new row will be created in the key lookup table as well. This way, for each legal combination of sub dimension rows, there will be one row in the lookup table.
The new key lookup table will contain more rows than any single one of the sub dimension tables by themselves, but much fewer rows than the fact table. The rows will also be much narrower than the sub dimensions. Since we replace a set of surrogate keys in the potentially huge fact table with a single key this design will also most probably save space, which in itself is a potential performance gain.

With a join between a fact table and one dimension divided into three sub dimensions we replace a simple 3-way star join with a 3-way star and an equijoin between a small set and a large table, thus adding some complexity to the query. However, since the joins are simple primary key joins the sql optimizer will still be able to build an efficient query plan. Additionally, the star join in this case will produce only possible combinations of sub dimension rows with respect to the valid time intervals. This means that we will have a smaller set of sub dimension combinations in the join to the fact table than in the original star query.

Running some tests on this new structure reveals the performance to be slightly better with the lookup table in the middle in some cases. More IO operations are needed in the original query since the three larger bitmap indexes on the surrogate key columns in the large fact table have to be scanned. With the lookup table, only the lookup key index needs to be read in the fact table resulting in the lower total IO figure of the new structure query when the predicates on the dimension tables are fairly selective. With less selective attributes, reading a total of about 100000 rows from the fact
table, the traditional star join is more efficient, doing the job in about 30 seconds versus a full minute for the lookup join.

Running tests with a more realistic setup with 3 dimensions split into 3 sub dimensions each – see diagram 7- there is a clear benefit of the three lookup tables.

The star join method is based on first retrieving the rows from each sub dimension fulfilling its respective predicates and doing a cartesian join (A join producing every possible combination of rows from table A and table B) between them, this set then driving the join to the fact table. Since the number of rows in a cartesian join between sets s₁...sₙ with row counts r₁...rₙ is \( \prod r \) there is a potential for a huge set. Since the number of rows returned from each dimension in the selective test query is around 800-900 the total set for the fact table access would be greater than 800⁹, far from a practical size. The oracle solution is to access one dimension at a time and comparing the result set to a bitmap index built on the fact table key column to produce a set of matching rows, then merging these results together(7).

With the lookup tables the optimizer finds all lookup keys through star joins, and then uses two of the three key sets in a new star join against the fact table and the does a hash join against the third set. The sub dimension to lookup table stars are highly selective since, even though each sub dimension fulfills the predicate 800-900 times, they only do it about 35 times at the same time.

With predicates returning about 250000 rows, the star join takes about three times as much time to run as the lookup table query although the star uses only 50% more IO operations. Making it more selective, with predicates returning about 35000 rows the time ratio is about the same but now the star join uses a lot more IO operations, 3 times that of the lookup table query. An even more selective query, returning only 500 rows which takes a full minute with the star runs in half a second with the lookup table since almost no disk access is required. Obviously, this fast performance
depends on the right data being in the cache, but less IO always improves the chances for a fast query execution.

Joining to the time dimension as well, limiting the result set to rows within one month is more beneficial to the lookup table join than to the star join method giving greatly reduced disk reads. The non-selective query was 20 times as fast with the lookup table and the selective query almost 35 times as fast.
Managing historically changing dimensions with the surrogate key translation table.

If we know from the beginning that we are dealing with a dimension that is very likely to have its historical data updated, with date ranges appearing and disappearing we can use the translation table principle to avoid going through and updating the fact table join keys. We replace the start date - end date interval columns in the lookup table with a single valid date. For every natural key, a new
row is created in this table each day, regardless of whether any of the sub dimensions have been changed or not. Now, when dimension data changes historically we can update the surrogate keys on the corresponding rows of this table and the fact table never need to be updated since rows are never added or removed from the middle of this table. The size of this table will be greater than the date range variant, but the dimension tables, which are much wider than this table will remain as small as before.

<table>
<thead>
<tr>
<th>PK</th>
<th>lookup_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid_date</td>
<td>natural_key</td>
</tr>
<tr>
<td></td>
<td>surrogate_key_1</td>
</tr>
<tr>
<td></td>
<td>surrogate_key_3</td>
</tr>
<tr>
<td></td>
<td>surrogate_key_2</td>
</tr>
</tbody>
</table>

Diagram 8: Lookup table for historically unstable dimension
6. Insights and conclusions

The paper has presented a basis for a theoretical approach to optimal dimension subdivision, although, during my work, I have failed to find an efficient algorithm for completing the subdivision. Further work in this area would be to look for an algorithm that could be run in a realistic amount of time on a larger set of dimension attribute. Although on the practical usefulness of such an algorithm I have found that it is almost always possible to do without a mathematical analysis of the dimension data for the subdivision of dimensions into sub dimensions, since the structure of the data is usually quite well known to the data modeler. We know that street address and postal code tend to change together for instance. For some types of data though, the algorithm would be useful but the treatment in this paper is too limited to be of much use there.

The algorithms for automation of the dimension presentation however, should be very useful in a real life situation of developing an etl process. The collection of algorithms presented is enough to build a complete basic etl process for type 2 slowly changing dimensions. Further work on this topic could include, for instance, algorithms for integrity checking and investigations into performance tuning.

Also, the paper clearly demonstrates the benefit of the dimension lookup tables for performance improvements by earlier filtering of impossible dimension row combinations. This technique should also be quite possible to use with totally uncorrelated dimensions, since the number of fact entries is usually magnitudes larger than the number of dimension entities. Consider for instance the combination of customers and products in a retail business. The number of customers might be large and there might be many products, but a typical customer would only ever buy a few different products, so a table mapping customer to the products they have bought at some point would be much smaller than the number of customers times the number of products.
For the future it would be very interesting to continue the investigation into the date range data type and index. Since Oracle provides the possibility of creating custom index types, that could be a way to solve the problem.
7. Bibliography


8. APPENDIX

A. Surrogate key join vs. natural key join + date range limit.

create table randfact (measure integer,
surrogate_key integer,
natural_key integer,
date_key date);

create bitmap index randfact_skix on randfact(surrogate_key);
create bitmap index randfact_nkix on randfact(natural_key);
create bitmap index randfact_dtix on randfact(date_key);

create table key_dim (natural_key integer, surrogate_key integer, start_date date, end_date date, text_val varchar2(128), int_property integer);

create table months (month_dat date);

insert into months values (to_date('20070101', 'YYYYMMDD'));
insert into months values (to_date('20070201', 'YYYYMMDD'));
insert into months values (to_date('20070301', 'YYYYMMDD'));
insert into months values (to_date('20070401', 'YYYYMMDD'));
insert into months values (to_date('20070501', 'YYYYMMDD'));
insert into months values (to_date('20070601', 'YYYYMMDD'));
insert into months values (to_date('20070701', 'YYYYMMDD'));
insert into months values (to_date('20070801', 'YYYYMMDD'));
insert into months values (to_date('20070901', 'YYYYMMDD'));
insert into months values (to_date('20071001', 'YYYYMMDD'));
insert into months values (to_date('20071101', 'YYYYMMDD'));
insert into months values (to_date('20071201', 'YYYYMMDD'));

insert into key_dim (natural_key, surrogate_key, start_date, text_val, int_property)
select nat_key, nat_key + mon_rnum, month_dat + trunc(dbms_random.value*28),
object_name, int_prop
from (select rownum mon_rnum, month_dat from months),
(select object_id*100 nat_key, object_name, trunc (object_id / 100) int_prop
from all_objects where rownum <= 10000)
create index key_dim_nk_ix on key_dim (natural_key);

update key_dim k set end_date = (select min(start_date) from key_dim where start_date > k.start_date and natural_key = k.natural_key);

update key_dim k set end_date = to_date('20080101', 'YYYYMMDD') where end_date is null;
**Star join example**

The table structure has one fact table and three dimension tables:

```
drop table randfact_3;
drop table key_dim_1;
drop table key_dim_2;
drop table key_dim_3;
drop table Key_Dim_Bridge;

create table randfact_3 (measure integer,
surrogate_key_1 integer,
surrogate_key_2 integer,
surrogate_key_3 integer,
natural_key integer,
date_key date
);

create table key_dim_1 (natural_key integer, surrogate_key_1 integer,
start_date_1 date, end_date_1 date, text_val_1 varchar2(128), int_property_1 integer);
create table key_dim_2 (natural_key integer, surrogate_key_2 integer,
start_date_2 date, end_date_2 date, text_val_2 varchar2(128), int_property_2 integer);
create table key_dim_3 (natural_key integer, surrogate_key_3 integer,
start_date_3 date, end_date_3 date, text_val_3 varchar2(128), int_property_3 integer);

alter table key_dim_1 add constraint key_dim_1_pk primary key (surrogate_key_1);
alter table key_dim_2 add constraint key_dim_2_pk primary key (surrogate_key_2);
alter table key_dim_3 add constraint key_dim_3_pk primary key (surrogate_key_3);

create bitmap index randfact_3_sk1ix on randfact_3(surrogate_key_1);
create bitmap index randfact_3_sk2ix on randfact_3(surrogate_key_2);
create bitmap index randfact_3_sk3ix on randfact_3(surrogate_key_3);
create bitmap index randfact_3_nkix on randfact_3(natural_key);
create bitmap index randfact_3_dtix on randfact_3(date_key);

alter table randfact_3 add constraint rfact_sk1_fk foreign key (surrogate_key_1) references key_dim_1(surrogate_key_1);
```
alter table randfact_3 add constraint rfact_sk2_fk foreign key 
(surrogate_key_2) references key_dim_2(surrogate_key_2);
alter table randfact_3 add constraint rfact_sk3_fk foreign key 
(surrogate_key_3) references key_dim_3(surrogate_key_3);

Create dimension data:

insert into key_dim_1 (natural_key, surrogate_key_1, start_date_1, 
text_val_1, int_property_1) 
select nat_key, nat_key, to_date('20061231', 'YYYYMMDD'), object_name, 
int_prop
from (select object_id*100 nat_key, object_name, trunc(dbms_random.value * 
100) int_prop --trunc (object_id / 50) int_prop
from all_objects where rownum <= 10000);

insert into key_dim_2 (natural_key, surrogate_key_2, start_date_2, 
text_val_2, int_property_2) 
select nat_key, nat_key, to_date('20061231', 'YYYYMMDD'), object_name, 
int_prop
from (select object_id*100 nat_key, object_name, trunc(dbms_random.value * 
100) int_prop --trunc (object_id / 50) int_prop
from all_objects where rownum <= 10000);

insert into key_dim_3 (natural_key, surrogate_key_3, start_date_3, 
text_val_3, int_property_3) 
select nat_key, nat_key, to_date('20061231', 'YYYYMMDD'), object_name, 
int_prop
from (select object_id*100 nat_key, object_name, trunc(dbms_random.value * 
100) int_prop --trunc (object_id / 50) int_prop
from all_objects where rownum <= 10000);

insert into key_dim_1 (natural_key, surrogate_key_1, start_date_1, 
text_val_1, int_property_1) 
select nat_key, nat_key + mon_rnum, month_dat + trunc(dbms_random.value*28), 
object_name, int_prop
from (select rownum mon_rnum, month_dat from months),
(select object_id*100 nat_key, object_name, trunc(dbms_random.value * 100) 
int_prop --trunc (object_id / 50) int_prop
from all_objects where rownum <= 10000);

insert into key_dim_2 (natural_key, surrogate_key_2, start_date_2, 
text_val_2, int_property_2) 
select nat_key, nat_key + mon_rnum, month_dat + trunc(dbms_random.value*28), 
object_name, int_prop
from (select rownum mon_rnum, month_dat from months),
(select object_id*100 nat_key, object_name, trunc(dbms_random.value * 100) 
int_prop --trunc (object_id / 50) int_prop
from all_objects where rownum <= 10000);
(select object_id*100 nat_key, object_name, trunc(dbms_random.value * 100)
  int_prop = trunc (object_id / 100) int_prop
from all_objects where rownum <= 10000);

insert into key_dim_3 (natural_key, surrogate_key_3, start_date_3,
text_val_3, int_property_3)
select nat_key, nat_key + mon_rnum, month_dat + trunc(dbms_random.value*28),
object_name, int_prop
from (select rownum mon_rnum, month_dat from months),
(select object_id*100 nat_key, object_name, trunc(dbms_random.value * 100)
  int_prop = trunc (object_id / 150) int_prop
from all_objects where rownum <= 10000);

cREATE INDEX key_dim_3_nk_ix ON key_dim_3 (natural_key);

cREATE INDEX key_dim_2_nk_ix ON key_dim_2 (natural_key);

CREATE INDEX key_dim_1_nk_ix ON key_dim_1 (natural_key);

UPDATE key_dim_1 k set end_date_1 = (select min(start_date_1) from key_dim_1
  where start_date_1 > k.start_date_1 and natural_key = k.natural_key);

commit;

UPDATE key_dim_2 k set end_date_2 = (select min(start_date_2) from key_dim_2
  where start_date_2 > k.start_date_2 and natural_key = k.natural_key);

commit;

UPDATE key_dim_3 k set end_date_3 = (select min(start_date_3) from key_dim_3
  where start_date_3 > k.start_date_3 and natural_key = k.natural_key);

commit;

UPDATE key_dim_1 k set end_date_1 = to_date('99991231', 'YYYYMMDD') where
  end_date_1 is null;

commit;

UPDATE key_dim_2 k set end_date_2 = to_date('99991231', 'YYYYMMDD') where
  end_date_2 is null;

commit;

UPDATE key_dim_3 k set end_date_3 = to_date('99991231', 'YYYYMMDD') where
  end_date_3 is null;

commit;

cREATE TABLE key_dim_bridge (
  natural_key,
  start_date,
  end_date,
  surrogate_key_1,
  surrogate_key_2,
  surrogate_key_3 )
AS SELECT dv.natural_key, dv.start_date, dv.end_date, k1.surrogate_key_1,
k2.surrogate_key_2, k3.surrogate_key_3 FROM
(select natural_key, start_date, coalesce(lead(start_date, 1) over (partition by natural_key order by start_date), to_date('99991231', 'YYYYMMDD')) end_date from ( select natural_key, start_date_1 start_date from key_dim_1 union select natural_key, start_date_2 from key_dim_2 union select natural_key, start_date_3 from key_dim_3 ) dv, key_dim_1 k1, key_dim_2 k2, key_dim_3 k3 where dv.natural_key = k1.natural_key and dv.natural_key = k2.natural_key and dv.natural_key = k3.natural_key and k1.start_date_1 <= dv.start_date and k1.end_date_1 >= dv.end_date and k2.start_date_2 <= dv.start_date and k2.end_date_2 >= dv.end_date and k3.start_date_3 <= dv.start_date and k3.end_date_3 >= dv.end_date; commit;

Create 19200000 random fact table rows:

declare
    vh_nat_key int_tab;
    vh_sur_key_1 int_tab;
    vh_sur_key_2 int_tab;
    vh_sur_key_3 int_tab;

    vd_start date := to_date('20070101', 'YYYYMMDD');
    vd_end date := to_date('20080101', 'YYYYMMDD');

    vd_num integer;
    vd_row integer;

    measvals int_tab := int_tab();
    nat int_tab := int_tab();
    sur1 int_tab := int_tab();
    sur2 int_tab := int_tab();
    sur3 int_tab := int_tab();

begin

    measvals.extend(200);
    nat.extend(200);
    sur1.extend(200);
    sur2.extend(200);
    sur3.extend(200);
while vd_start < vd_end loop

    select kdb.natural_key, kdb.surrogate_key_1, kdb.surrogate_key_2, kdb.surrogate_key_3 bulk collect
        into vh_nat_key, vh_sur_key_1, vh_sur_key_2, vh_sur_key_3
        from key_dim_bridge kdb
        where kdb.start_date <= vd_start
            and kdb.end_date > vd_start;

    vd_num := vh_nat_key.last;

    for vd_i in 1..200 loop
        for vd_j in 1..200 loop
            vd_row := trunc(dbms_random.value*vd_num)+1;

            measvals(vd_j) := trunc(dbms_Random.value*1000);
            nat(vd_j) := vh_nat_key(vd_row);
            sur1(vd_j) := vh_sur_key_1(vd_row);
            sur2(vd_j) := vh_sur_key_2(vd_row);
            sur3(vd_j) := vh_sur_key_3(vd_row);
        end loop;

        forall vd_x in 1..:iter2
            insert into randfact_3(measure, surrogate_key_1, surrogate_key_2, surrogate_key_3, natural_key, date_key)
                values (measvals(vd_x), sur1(vd_x), sur2(vd_x), sur3(vd_x), nat(vd_x), vd_start);

        end loop;

        commit;

    vd_start := vd_start + 1;
    end loop;
end;

Gather statistics for the oracle optimizer
begin
    dbms_stats.gather_table_stats(ownname => 'DWCONC', tabname => 'RANDFACT_3', method_opt => 'FOR ALL COLUMNS SIZE AUTO', cascade => true);
dbms_stats.gather_table_stats(ownname => 'DWCONC', tabname => 'KEY_DIM_1', method_opt => 'FOR ALL COLUMNS SIZE AUTO', cascade => true);

dbms_stats.gather_table_stats(ownname => 'DWCONC', tabname => 'KEY_DIM_2', method_opt => 'FOR ALL COLUMNS SIZE AUTO', cascade => true);

dbms_stats.gather_table_stats(ownname => 'DWCONC', tabname => 'KEY_DIM_3', method_opt => 'FOR ALL COLUMNS SIZE AUTO', cascade => true);

end;

Run a test script

-- first run both queries to avoid issues with unequal caching benefits

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.natural_key = d1.natural_key
and f.natural_key = d2.natural_key
and f.natural_key = d3.natural_key
and f.date_key >= d1.start_date_1 and f.date_key < d1.end_date_1
and f.date_key >= d2.start_date_2 and f.date_key < d2.end_date_2
and f.date_key >= d3.start_date_3 and f.date_key < d3.end_date_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.surrogate_key_1 = d1.surrogate_key_1
and f.surrogate_key_2 = d2.surrogate_key_2
and f.surrogate_key_3 = d3.surrogate_key_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;

-- trace session
alter session set events '10046 trace name context forever, level 12';

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.natural_key = d1.natural_key
and f.natural_key = d2.natural_key
and f.natural_key = d3.natural_key
and f.date_key >= d1.start_date_1 and f.date_key < d1.end_date_1
and f.date_key >= d2.start_date_2 and f.date_key < d2.end_date_2
and f.date_key >= d3.start_date_3 and f.date_key < d3.end_date_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;
and f.date_key >= d3.start_date_3 and f.date_key < d3.end_date_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.surrogate_key_1 = d1.surrogate_key_1
and f.surrogate_key_2 = d2.surrogate_key_2
and f.surrogate_key_3 = d3.surrogate_key_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;

alter session set events '10046 trace name context off';

---

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.natural_key = d1.natural_key
and f.natural_key = d2.natural_key
and f.natural_key = d3.natural_key
and f.date_key >= d1.start_date_1 and f.date_key < d1.end_date_1
and f.date_key >= d2.start_date_2 and f.date_key < d2.end_date_2
and f.date_key >= d3.start_date_3 and f.date_key < d3.end_date_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90;

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

---

1 SORT AGGREGATE (cr=79809 pr=64966 pw=0 time=28866552 us)
10748 HASH JOIN (cr=79809 pr=64966 pw=0 time=28880124 us)
select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.surrogate_key_1 = d1.surrogate_key_1
and f.surrogate_key_2 = d2.surrogate_key_2
and f.surrogate_key_3 = d3.surrogate_key_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows | Row Source Operation
------ | -------------------
1 | SORT AGGREGATE (cr=95745 pr=0 pw=0 time=1157811 us)
10748 | TABLE ACCESS BY INDEX ROWID RANDFACT_3 (cr=95745 pr=0 pw=0 time=885484 us)
10748 | BITMAP CONVERSION TO ROWIDS (cr=85740 pr=0 pw=0 time=831722 us)
4 | BITMAP AND (cr=85740 pr=0 pw=0 time=1285310 us)
97 | BITMAP MERGE (cr=28310 pr=0 pw=0 time=540586 us)
12621 | BITMAP KEY ITERATION (cr=28310 pr=0 pw=0 time=214623 us)
12655 | TABLE ACCESS FULL KEY_DIM_1 (cr=1005 pr=0 pw=0 time=25156 us)
12621 | BITMAP INDEX RANGE SCAN RANDFACT_3_SK1IX (cr=27305 pr=0 pw=0 time=149071 us)(object id 58291)
<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:
Test 2 with int_property values more selective instead. Tracefile:

select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.natural_key = d1.natural_key
and f.natural_key = d2.natural_key
and f.natural_key = d3.natural_key
and f.date_key >= d1.start_date_1 and f.date_key < d1.end_date_1
and f.date_key >= d2.start_date_2 and f.date_key < d2.end_date_2
and f.date_key >= d3.start_date_3 and f.date_key < d3.end_date_3
and d1.int_property_1 >= 94
and d2.int_property_2 >= 95
and d3.int_property_3 >= 95

call count cpu elapsed disk query current rows
------- ------ -------- --------- ---------- ---------- ---------- ----------
Parse   1  0.01  0.02   0   0   3   0
Execute 1  0.00  0.00   0   0   0   0
Fetch   1  10.07 22.27  65080   79809   3   0
------- ------ -------- --------- ---------- ---------- ---------- ----------
total   3  10.09 22.29  65080   79809   3   1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows Row Source Operation
------- ---------------------------------------------------
1 SORT AGGREGATE (cr=79809 pr=65080 pw=0 time=22271775 us)
1466 HASH JOIN (cr=79809 pr=65080 pw=0 time=22256369 us)
29596 HASH JOIN (cr=78804 pr=65080 pw=0 time=22024499 us)
6101 TABLE ACCESS FULL KEY_DIM_2 (cr=1005 pr=0 pw=0 time=18365 us)
752166 HASH JOIN (cr=77799 pr=65080 pw=0 time=19574849 us)
6690 TABLE ACCESS FULL KEY_DIM_3 (cr=1005 pr=0 pw=0 time=20105 us)
14600000 TABLE ACCESS FULL RANDFACT_3 (cr=76794 pr=65080 pw=0 time=43800036 us)
7590 TABLE ACCESS FULL KEY_DIM_1 (cr=1005 pr=0 pw=0 time=22832 us)

Elapsed times include waiting on following events:
Event waited on Times Max. Wait Total Waited
------------------------------------------ ----------- -----------
SQL*Net message to client 2 0.00 0.00
log file sync 1 0.00 0.00
SQL*Net message from client 2 0.00 0.00
db file scattered read 7103 0.04 11.64
db file sequential read 1023 0.02 0.65
latch: shared pool 6 0.02 0.03

*******************************************************************************
```sql
select count(*), sum(f.measure) from
randfact_3 f,
key_dim_1 d1, key_dim_2 d2, key_dim_3 d3
where f.surrogate_key_1 = d1.surrogate_key_1
and f.surrogate_key_2 = d2.surrogate_key_2
and f.surrogate_key_3 = d3.surrogate_key_3
and d1.int_property_1 >= 94
and d2.int_property_2 >= 95
and d3.int_property_3 >= 95
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>0.64</td>
<td>0.65</td>
<td>0</td>
<td>48323</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| total | 3 | 0.64 | 0.66 | 0 | 48323 | 0 | 1 |

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows | Row Source Operation
----- | ---------------------------------------------
1 | SORT AGGREGATE (cr=48323 pr=0 pw=0 time=656373 us)
1466 | TABLE ACCESS BY INDEX ROWID RANDFACT_3 (cr=48323 pr=0 pw=0 time=662641 us)
1466 | BITMAP CONVERSION TO ROWIDS (cr=46868 pr=0 pw=0 time=653823 us)
65 | BITMAP AND (cr=46868 pr=0 pw=0 time=652345 us)
6672 | BITMAP INDEX RANGE SCAN RANDFACT_3_SK3IX (cr=14376 pr=0 pw=0 time=81936 us)(object id 58293)
6690 | TABLE ACCESS FULL KEY_DIM_3 (cr=1005 pr=0 pw=0 time=20117 us)
6672 | BITMAP KEY ITERATION (cr=15381 pr=0 pw=0 time=20117 us)
60 | BITMAP MERGE (cr=14102 pr=0 pw=0 time=212480 us)
6082 | BITMAP KEY ITERATION (cr=14102 pr=0 pw=0 time=103469 us)
6101 | TABLE ACCESS FULL KEY_DIM_2 (cr=1005 pr=0 pw=0 time=18358 us)
6082 | BITMAP INDEX RANGE SCAN RANDFACT_3_SK2IX (cr=13097 pr=0 pw=0 time=68208 us)(object id 58292)
71 | BITMAP MERGE (cr=17385 pr=0 pw=0 time=288486 us)
7570 | BITMAP KEY ITERATION (cr=17385 pr=0 pw=0 time=128744 us)
7590 | TABLE ACCESS FULL KEY_DIM_1 (cr=1005 pr=0 pw=0 time=22811 us)
7570 | BITMAP INDEX RANGE SCAN RANDFACT_3_SK1IX (cr=16380 pr=0 pw=0 time=86965 us)(object id 58291)

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

*******************************************************************************
62
B. Surrogate key translation table test trace files

3-way selective test

*******************************************************************************

select count(*), sum(f.measure)
from randfact_l f, key_dim_lookup_a l, key_dim_la_1 d1, key_dim_la_2 d2, key_dim_la_3 d3
where f.lookup_key = l.lookup_key_a
and d1.surrogate_key_1 = l.surrogate_key_1
and d2.surrogate_key_2 = l.surrogate_key_2
and d3.surrogate_key_3 = l.surrogate_key_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90

call count cpu elapsed disk query current rows
------- ------  ------- ---------- ---------- ---------- ----------  ----------
Parse     1     0.01       0.01          0          0          0           0          0
Execute   1     0.00       0.00          0          0          0           0          0
Fetch     1     0.73      10.74       9986      41534          0           1
------- ------  ------- ---------- ---------- ---------- ----------  ----------
total     3     0.75      10.75       9986      41534          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows Row Source Operation
------- ---------------------------------------------------------------
1  SORT AGGREGATE (cr=41534 pr=9986 pw=0 time=10745808 us)
42009 TABLE ACCESS BY INDEX ROWID RANDFACT_L (cr=41534 pr=9986 pw=0 time=9265039 us)
42109 NESTED LOOPS (cr=4888 pr=23 pw=0 time=12717353 us)
99 TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_A (cr=4677 pr=22 pw=0 time=99592 us)
99 BITMAP CONVERSION TO ROWIDS (cr=4644 pr=8 pw=0 time=98989 us)
1 BITMAP AND (cr=4644 pr=8 pw=0 time=98784 us)
1 BITMAP MERGE (cr=1442 pr=0 pw=0 time=18838 us)
1333 BITMAP KEY ITERATION (cr=1442 pr=0 pw=0 time=21417 us)
1333 TABLE ACCESS FULL KEY_DIM_LA_2 (cr=92 pr=0 pw=0 time=2737 us)
1333 BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK2IX (cr=1350 pr=0 pw=0 time=11723 us) (object id 58587)
1 BITMAP MERGE (cr=1517 pr=5 pw=0 time=43478 us)
1403 BITMAP KEY ITERATION (cr=1517 pr=5 pw=0 time=22482 us)
1403 TABLE ACCESS FULL KEY_DIM_LA_3 (cr=92 pr=0 pw=0 time=2834 us)
select count(*), sum(f.measure)
from randfact_3n f, key_dim_la_1 d1, key_dim_la_2 d2, key_dim_la_3 d3
where d1.surrogate_key_1 = f.surrogate_key_1
and d2.surrogate_key_2 = f.surrogate_key_2
and d3.surrogate_key_3 = f.surrogate_key_3
and d1.int_property_1 >= 90
and d2.int_property_2 >= 90
and d3.int_property_3 >= 90

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56
<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>31767</td>
<td>0.02</td>
<td>11.50</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>31767</td>
<td>0.02</td>
<td>11.50</td>
</tr>
</tbody>
</table>
### 3-way non-selective test

```sql
select count(*), sum(f.measure)
from randfact_l f, key_dim_lookup_a l, key_dim_la_1 d1, key_dim_la_2 d2, key_dim_la_3 d3
where f.lookup_key = l.lookup_key_a
and d1.surrogate_key_1 = l.surrogate_key_1
and d2.surrogate_key_2 = l.surrogate_key_2
and d3.surrogate_key_3 = l.surrogate_key_3
and d1.int_property_1 >= 85
and d2.int_property_2 >= 85
and d3.int_property_3 >= 85
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.01</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>1.90</td>
<td>59.40</td>
<td>25775</td>
<td>89010</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Total 3 calls: 1.93 seconds elapsed, 59.41 seconds total.

Misses in library cache during parse: 1

Optimizer mode: ALL_ROWS

Parsing user id: 56

Rows | Row Source Operation
--- | ---------------------------------------------------
1 | SORT AGGREGATE (cr=89010 pr=25775 pw=0 time=59404725 us)
100856 | TABLE ACCESS BY INDEX ROWID RANDFACT_L (cr=89010 pr=25775 pw=0 time=56313946 us)
101098 | NESTED LOOPS (cr=1029 pr=276 pw=0 time=16480898 us)
241 | HASH JOIN (cr=503 pr=205 pw=0 time=127792 us)
992 | HASH JOIN (cr=411 pr=205 pw=0 time=53011 us)
2167 | TABLE ACCESS FULL KEY_DIM_LA_3 (cr=92 pr=0 pw=0 time=4516 us)
5865 | HASH JOIN (cr=319 pr=205 pw=0 time=79103 us)
2140 | TABLE ACCESS FULL KEY_DIM_LA_2 (cr=92 pr=0 pw=0 time=4319 us)
35727 | TABLE ACCESS FULL KEY_DIM_LOOKUP_A (cr=227 pr=205 pw=0 time=272458 us)
2317 | TABLE ACCESS FULL KEY_DIM_LA_1 (cr=92 pr=0 pw=0 time=6997 us)
100856 | BITMAP CONVERSION TO ROWIDS (cr=526 pr=71 pw=0 time=655486 us)
243 | BITMAP INDEX SINGLE VALUE RANDFACT_L_LRI (cr=526 pr=71 pw=0 time=617250 us)(object id 58607)

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>25575</td>
<td>0.12</td>
<td>58.06</td>
</tr>
</tbody>
</table>
select count(*), sum(f.measure)
from randfact_3n f, key_dim_la_1 d1, key_dim_la_2 d2, key_dim_la_3 d3
where d1.surrogate_key_1 = f.surrogate_key_1
and d2.surrogate_key_2 = f.surrogate_key_2
and d3.surrogate_key_3 = f.surrogate_key_3
and d1.int_property_1 >= 85
and d2.int_property_2 >= 85
and d3.int_property_3 >= 85
<table>
<thead>
<tr>
<th>Event</th>
<th>Waited</th>
<th>-------</th>
<th>-------</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>57692</td>
<td>0.17</td>
<td>32.05</td>
</tr>
</tbody>
</table>

*******************************************************************************

69
**9-way non selective**

```
select count(*), sum(f.measure) 
from randfact_l3 f, 
key_dim_lookup_a la, key_dim_la_1 da1, key_dim_la_2 da2, key_dim_la_3 da3, 
key_dim_lookup_b lb, key_dim_lb_1 db1, key_dim_lb_2 db2, key_dim_lb_3 db3, 
key_dim_lookup_c lc, key_dim_lc_1 dc1, key_dim_lc_2 dc2, key_dim_lc_3 dc3, 
date_dim dd 
where dd.date_key = f.date_key 
  and dd.nyear = 2007 
  and dd.nmonth = 6 
  and f.lookup_key_a = la.lookup_key_a 
  and f.lookup_key_b = lb.lookup_key_b 
  and f.lookup_key_c = lc.lookup_key_c 
  and da1.surrogate_key_1 = la.surrogate_key_1 
  and da2.surrogate_key_2 = la.surrogate_key_2 
  and da3.surrogate_key_3 = la.surrogate_key_3 
  and db1.surrogate_key_1 = lb.surrogate_key_1 
  and db2.surrogate_key_2 = lb.surrogate_key_2 
  and db3.surrogate_key_3 = lb.surrogate_key_3 
  and dc1.surrogate_key_1 = lc.surrogate_key_1 
  and dc2.surrogate_key_2 = lc.surrogate_key_2 
  and dc3.surrogate_key_3 = lc.surrogate_key_3 
  and da1.int_property_1 >= 85 
  and da2.int_property_2 >= 85 
  and da3.int_property_3 >= 85 
  and db1.int_property_1 >= 85 
  and db2.int_property_2 >= 85 
  and db3.int_property_3 >= 85 
  and dc1.int_property_1 >= 85 
  and dc2.int_property_2 >= 85 
  and dc3.int_property_3 >= 85
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.17</td>
<td>0.19</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.07</td>
<td>0.23</td>
<td>72</td>
<td>7350</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>0.75</td>
<td>4.18</td>
<td>8160</td>
<td>25900</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Total:** 3 | 1.00 | 4.60 | 8232 | 33250 | 9 | 1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

**Rows**  
---

1 TEMP TABLE TRANSFORMATION (cr=33250 pr=8232 pw=1 time=4412979 us)
1 LOAD AS SELECT (cr=7350 pr=72 pw=1 time=229679 us)
HASH JOIN (cr=7350 pr=72 pw=0 time=173106 us)
HASH JOIN (cr=7255 pr=72 pw=0 time=169683 us)
HASH JOIN (cr=7160 pr=72 pw=0 time=170400 us)
TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_B (cr=7065 pr=72 pw=0 time=134346 us)

BITMAP CONVERSION TO ROWIDS (cr=6988 pr=8 pw=0 time=128907 us)
BITMAP AND (cr=6988 pr=8 pw=0 time=128417 us)
BITMAP MERGE (cr=2446 pr=3 pw=0 time=49783 us)
BITMAP KEY ITERATION (cr=2446 pr=3 pw=0 time=39717 us)
TABLE ACCESS FULL KEY_DIM_LB_1 (cr=95 pr=0 pw=0 time=4938 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK1IX (cr=2351 pr=3 pw=0 time=37748 us)

BITMAP MERGE (cr=2246 pr=1 pw=0 time=39876 us)
BITMAP KEY ITERATION (cr=2246 pr=1 pw=0 time=34279 us)
TABLE ACCESS FULL KEY_DIM_LB_2 (cr=95 pr=0 pw=0 time=4312 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK2IX (cr=2151 pr=1 pw=0 time=27647 us)

BITMAP MERGE (cr=2296 pr=4 pw=0 time=38578 us)
BITMAP KEY ITERATION (cr=2296 pr=4 pw=0 time=58537 us)
TABLE ACCESS FULL KEY_DIM_LB_3 (cr=95 pr=0 pw=0 time=4366 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK3IX (cr=2201 pr=4 pw=0 time=28427 us)

TABLE ACCESS FULL KEY_DIM_LB_1 (cr=95 pr=0 pw=0 time=4675 us)
TABLE ACCESS FULL KEY_DIM_LB_2 (cr=95 pr=0 pw=0 time=4315 us)
TABLE ACCESS FULL KEY_DIM_LB_3 (cr=95 pr=0 pw=0 time=4368 us)
SORT AGGREGATE (cr=25900 pr=8160 pw=0 time=418294 us)

HASH JOIN (cr=25900 pr=8160 pw=0 time=4207343 us)
HASH JOIN (cr=25805 pr=8160 pw=0 time=4160964 us)
HASH JOIN (cr=25710 pr=8160 pw=0 time=4117418 us)
HASH JOIN (cr=25615 pr=8160 pw=0 time=4063800 us)
HASH JOIN (cr=18559 pr=8091 pw=0 time=3888560 us)
HASH JOIN (cr=11492 pr=8019 pw=0 time=4163813 us)

TABLE ACCESS FULL SYS_TEMP_OFD9D6633_110985A (cr=5 pr=1 pw=0 time=1171 us)

TABLE ACCESS BY INDEX ROWID RANDFACT_L3 (cr=11487 pr=8018 pw=0 time=4015049 us)

BITMAP CONVERSION TO ROWIDS (cr=984 pr=153 pw=0 time=736119 us)
BITMAP AND (cr=984 pr=153 pw=0 time=713614 us)
BITMAP MERGE (cr=794 pr=153 pw=0 time=688491 us)
BITMAP KEY ITERATION (cr=794 pr=153 pw=0 time=16919 us)
TABLE ACCESS FULL SYS_TEMP_OFD9D6633_110985A (cr=3 pr=0 pw=0 time=778 us)

BITMAP INDEX RANGE SCAN RANDFACT_L3_LUBIX (cr=791 pr=153 pw=0 time=664901 us)

BITMAP MERGE (cr=190 pr=0 pw=0 time=9777 us)
BITMAP KEY ITERATION (cr=190 pr=0 pw=0 time=1341 us)
TABLE ACCESS FULL DATE_DIM (cr=23 pr=0 pw=0 time=165 us)
BITMAP INDEX RANGE SCAN RANDFACT_L3_DTIX (cr=167 pr=0 pw=0 time=1085 us)

TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_C (cr=7067 pr=72 pw=0 time=131216 us)
BITMAP CONVERSION TO ROWIDS (cr=6990 pr=8 pw=0 time=120619 us)
BITMAP AND (cr=6990 pr=8 pw=0 time=120109 us)
BITMAP MERGE (cr=2439 pr=3 pw=0 time=45053 us)
BITMAP KEY ITERATION (cr=2439 pr=3 pw=0 time=37214 us)
TABLE ACCESS FULL KEY_DIM_LC_1 (cr=95 pr=0 pw=0 time=4755 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK1IX (cr=2344 pr=3 pw=0 time=33205 us)(object id 58623)
BITMAP MERGE (cr=2255 pr=1 pw=0 time=38988 us)
BITMAP KEY ITERATION (cr=2255 pr=1 pw=0 time=34287 us)
TABLE ACCESS FULL KEY_DIM_LC_2 (cr=95 pr=0 pw=0 time=4320 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK2IX (cr=2160 pr=1 pw=0 time=28630 us)(object id 58624)
BITMAP MERGE (cr=2296 pr=4 pw=0 time=35886 us)
BITMAP KEY ITERATION (cr=2296 pr=4 pw=0 time=36878 us)
TABLE ACCESS FULL KEY_DIM_LC_3 (cr=95 pr=0 pw=0 time=4367 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK3IX (cr=2201 pr=4 pw=0 time=26094 us)(object id 58625)
TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_A (cr=7056 pr=69 pw=0 time=118386 us)
BITMAP CONVERSION TO ROWIDS (cr=6979 pr=5 pw=0 time=114397 us)
BITMAP AND (cr=6979 pr=5 pw=0 time=113906 us)
BITMAP MERGE (cr=2251 pr=1 pw=0 time=39042 us)
BITMAP KEY ITERATION (cr=2251 pr=1 pw=0 time=36468 us)
TABLE ACCESS FULL KEY_DIM_LA_2 (cr=92 pr=0 pw=0 time=4347 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK2IX (cr=2159 pr=1 pw=0 time=27157 us)(object id 58587)
BITMAP MERGE (cr=2293 pr=1 pw=0 time=38084 us)
BITMAP KEY ITERATION (cr=2293 pr=1 pw=0 time=34718 us)
TABLE ACCESS FULL KEY_DIM_LA_3 (cr=92 pr=0 pw=0 time=4372 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK3IX (cr=2201 pr=1 pw=0 time=28215 us)(object id 58588)
BITMAP MERGE (cr=2435 pr=3 pw=0 time=36601 us)
BITMAP KEY ITERATION (cr=2435 pr=3 pw=0 time=62584 us)
TABLE ACCESS FULL KEY_DIM_LA_1 (cr=92 pr=0 pw=0 time=4664 us)
BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK1IX (cr=2343 pr=3 pw=0 time=25075 us)(object id 58586)
TABLE ACCESS FULL KEY_DIM_LC_1 (cr=95 pr=0 pw=0 time=4710 us)
TABLE ACCESS FULL KEY_DIM_LC_2 (cr=95 pr=0 pw=0 time=4361 us)
TABLE ACCESS FULL KEY_DIM_LC_3 (cr=95 pr=0 pw=0 time=4437 us)

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>control file sequential read</td>
<td>5</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>8232</td>
<td>0.04</td>
<td>3.55</td>
</tr>
<tr>
<td>direct path write temp</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>log file sync</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
select count(*), sum(f.measure)
from randfact_9 f,
key_dim_la_1 da1, key_dim_la_2 da2, key_dim_la_3 da3,
key_dim_lb_1 db1, key_dim_lb_2 db2, key_dim_lb_3 db3,
key_dim_lc_1 dc1, key_dim_lc_2 dc2, key_dim_lc_3 dc3,
date_dim dd
where dd.date_key = f.date_key
  and dd.nyear = 2007
  and dd.nmonth = 6
and da1.surrogate_key_1 = f.surrogate_key_a1
and da2.surrogate_key_2 = f.surrogate_key_a2
and da3.surrogate_key_3 = f.surrogate_key_a3
and db1.surrogate_key_1 = f.surrogate_key_b1
and db2.surrogate_key_2 = f.surrogate_key_b2
and db3.surrogate_key_3 = f.surrogate_key_b3
and dc1.surrogate_key_1 = f.surrogate_key_c1
and dc2.surrogate_key_2 = f.surrogate_key_c2
and dc3.surrogate_key_3 = f.surrogate_key_c3
and da1.int_property_1 >= 85
and da2.int_property_2 >= 85
and da3.int_property_3 >= 85
and db1.int_property_1 >= 85
and db2.int_property_2 >= 85
and db3.int_property_3 >= 85
and dc1.int_property_1 >= 85
and dc2.int_property_2 >= 85
and dc3.int_property_3 >= 85

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56
Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>187335</td>
<td>0.13</td>
<td>118.61</td>
</tr>
</tbody>
</table>

*******************************************************************************
9-way selective

*******************************************************************************
select count(*), sum(f.measure)
from randfact_l3 f,
key_dim_lookup_a la, key_dim_la_1 da1, key_dim_la_2 da2, key_dim_la_3 da3,
key_dim_lookup_b lb, key_dim_lb_1 db1, key_dim_lb_2 db2, key_dim_lb_3 db3,
key_dim_lookup_c lc, key_dim_lc_1 dc1, key_dim_lc_2 dc2, key_dim_lc_3 dc3,
date_dim dd
where dd.date_key = f.date_key
    and dd.nyear = 2007
    and dd.month = 6
    and f.lookup_key_a = la.lookup_key_a
    and f.lookup_key_b = lb.lookup_key_b
    and f.lookup_key_c = lc.lookup_key_c
    and da1.surrogate_key_1 = la.surrogate_key_1
    and da2.surrogate_key_2 = la.surrogate_key_2
    and da3.surrogate_key_3 = la.surrogate_key_3
    and db1.surrogate_key_1 = lb.surrogate_key_1
    and db2.surrogate_key_2 = lb.surrogate_key_2
    and db3.surrogate_key_3 = lb.surrogate_key_3
    and dc1.surrogate_key_1 = lc.surrogate_key_1
    and dc2.surrogate_key_2 = lc.surrogate_key_2
    and dc3.surrogate_key_3 = lc.surrogate_key_3
    and da1.int_property_1 >= 94
    and da2.int_property_2 >= 94
    and da3.int_property_3 >= 94
    and db1.int_property_1 >= 94
    and db2.int_property_2 >= 94
    and db3.int_property_3 >= 94
    and dc1.int_property_1 >= 94
    and dc2.int_property_2 >= 94
    and dc3.int_property_3 >= 94

call  count  cpu  elapsed  disk  query  current  rows
------- ------ ------ ------- ------ ------- ------- ------
Parse   1 0.18   0.18      0       0     1       0
Execute 1 0.06 0.13 3329     7      0
Fetch   1 0.14 0.14 9321     1      1

--- total --- ------- ------- ------- ------- ------- ------
    3 0.39 0.46 12650 9      1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows  Row Source Operation
------- -----------------------------------------------
1 TEMP TABLE TRANSFORMATION  (cr=12650 pr=1 pw=1 time=278304 us)
1 LOAD AS SELECT  (cr=3329 pr=0 pw=1 time=137140 us)
HASH JOIN (cr=3329 pr=0 pw=0 time=45204 us)
HASH JOIN (cr=3234 pr=0 pw=0 time=47285 us)
HASH JOIN (cr=3139 pr=0 pw=0 time=39810 us)
TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_B (cr=3044 pr=0 pw=0 time=38662 us)
  BITMAP CONVERSION TO ROWIDS (cr=3031 pr=0 pw=0 time=36512 us)
  BITMAP AND (cr=3031 pr=0 pw=0 time=36437 us)
  BITMAP MERGE (cr=1097 pr=0 pw=0 time=13574 us)
  BITMAP KEY ITERATION (cr=1097 pr=0 pw=0 time=15803 us)
  TABLE ACCESS FULL KEY_DIM_LB_1 (cr=95 pr=0 pw=0 time=2047 us)
  BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK1IX (cr=1002 pr=0 pw=0 time=8225 us)(object id 58619)
  BITMAP MERGE (cr=981 pr=0 pw=0 time=12308 us)
  BITMAP KEY ITERATION (cr=981 pr=0 pw=0 time=14959 us)
  TABLE ACCESS FULL KEY_DIM_LB_2 (cr=95 pr=0 pw=0 time=2661 us)
  BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK2IX (cr=886 pr=0 pw=0 time=7424 us)(object id 58620)
  BITMAP MERGE (cr=953 pr=0 pw=0 time=12446 us)
  BITMAP KEY ITERATION (cr=953 pr=0 pw=0 time=15099 us)
  TABLE ACCESS FULL KEY_DIM_LB_3 (cr=95 pr=0 pw=0 time=3375 us)
  BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_B_SK3IX (cr=858 pr=0 pw=0 time=8088 us)(object id 58621)
  TABLE ACCESS FULL KEY_DIM_LB_1 (cr=95 pr=0 pw=0 time=2006 us)
  TABLE ACCESS FULL KEY_DIM_LB_2 (cr=95 pr=0 pw=0 time=1794 us)
  TABLE ACCESS FULL KEY_DIM_LB_3 (cr=95 pr=0 pw=0 time=2547 us)
  SORT AGGREGATE (cr=9321 pr=1 pw=0 time=140470 us)
  HASH JOIN (cr=9321 pr=1 pw=0 time=143453 us)
  HASH JOIN (cr=9226 pr=1 pw=0 time=136218 us)
  HASH JOIN (cr=9131 pr=1 pw=0 time=128914 us)
  HASH JOIN (cr=9036 pr=1 pw=0 time=123655 us)
  HASH JOIN (cr=5998 pr=1 pw=0 time=80417 us)
  HASH JOIN (cr=2948 pr=1 pw=0 time=58530 us)
  TABLE ACCESS FULL SYS_TEMP_0FD9D662D_110985A (cr=5 pr=1 pw=0 time=685 us)
  TABLE ACCESS BY INDEX ROWID RANDFACT_L3 (cr=2943 pr=0 pw=0 time=38865 us)
  BITMAP CONVERSION TO ROWIDS (cr=305 pr=0 pw=0 time=23995 us)
  BITMAP AND (cr=305 pr=0 pw=0 time=21011 us)
  BITMAP MERGE (cr=115 pr=0 pw=0 time=10270 us)
  BITMAP KEY ITERATION (cr=115 pr=0 pw=0 time=1536 us)
  TABLE ACCESS FULL SYS_TEMP_0FD9D662D_110985A (cr=3 pr=0 pw=0 time=155 us)
  BITMAP INDEX RANGE SCAN RANDFACT_L3_LUBIX (cr=112 pr=0 pw=0 time=538 us)(object id 58635)
  BITMAP MERGE (cr=190 pr=0 pw=0 time=9204 us)
  BITMAP KEY ITERATION (cr=190 pr=0 pw=0 time=1470 us)
  TABLE ACCESS FULL DATE_DIM (cr=23 pr=0 pw=0 time=189 us)
  BITMAP INDEX RANGE SCAN RANDFACT_L3_DTIX (cr=167 pr=0 pw=0 time=1063 us)(object id 58735)
  TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_C (cr=3050 pr=0 pw=0 time=39792 us)
34        BITMAP CONVERSION TO ROWIDS (cr=3037 pr=0 pw=0 time=39679 us)
1          BITMAP AND (cr=3037 pr=0 pw=0 time=39602 us)
1          BITMAP MERGE (cr=1094 pr=0 pw=0 time=13483 us)
981         BITMAP KEY ITERATION (cr=1094 pr=0 pw=0 time=16759 us)
981         TABLE ACCESS FULL KEY_DIM_LC_1 (cr=95 pr=0 pw=0 time=2026 us)
981         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK1IX (cr=999 pr=0 pw=0
time=8390 us) (object id 58623)
1          BITMAP MERGE (cr=990 pr=0 pw=0 time=13674 us)
878         BITMAP KEY ITERATION (cr=990 pr=0 pw=0 time=14960 us)
878         TABLE ACCESS FULL KEY_DIM_LC_2 (cr=95 pr=0 pw=0 time=2661 us)
878         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK2IX (cr=999 pr=0 pw=0
time=8390 us) (object id 58624)
837         BITMAP KEY ITERATION (cr=953 pr=0 pw=0 time=15099 us)
837         TABLE ACCESS FULL KEY_DIM_LC_3 (cr=92 pr=0 pw=0 time=2540 us)
837         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_C_SK3IX (cr=858 pr=0 pw=0
time=7541 us) (object id 58625)
34        TABLE ACCESS BY INDEX ROWID KEY_DIM_LOOKUP_A (cr=3038 pr=0 pw=0
time=39792 us)
34        BITMAP CONVERSION TO ROWIDS (cr=3025 pr=0 pw=0 time=39677 us)
1          BITMAP AND (cr=3025 pr=0 pw=0 time=39603 us)
1          BITMAP MERGE (cr=950 pr=0 pw=0 time=11580 us)
837         BITMAP KEY ITERATION (cr=950 pr=0 pw=0 time=15119 us)
837         TABLE ACCESS FULL KEY_DIM_LA_3 (cr=92 pr=0 pw=0 time=2550 us)
837         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK3IX (cr=858 pr=0 pw=0
time=7080 us) (object id 58588)
1          BITMAP MERGE (cr=987 pr=0 pw=0 time=12712 us)
878         BITMAP KEY ITERATION (cr=987 pr=0 pw=0 time=14961 us)
878         TABLE ACCESS FULL KEY_DIM_LA_2 (cr=92 pr=0 pw=0 time=2663 us)
878         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK2IX (cr=895 pr=0 pw=0
time=7510 us) (object id 58587)
1          BITMAP MERGE (cr=1088 pr=0 pw=0 time=15208 us)
980         BITMAP KEY ITERATION (cr=1088 pr=0 pw=0 time=13769 us)
980         TABLE ACCESS FULL KEY_DIM_LA_1 (cr=92 pr=0 pw=0 time=1990 us)
980         BITMAP INDEX RANGE SCAN KEY_DIM_LOOKUP_A_SK1IX (cr=996 pr=0 pw=0
time=10002 us) (object id 58586)
981         TABLE ACCESS FULL KEY_DIM_LC_1 (cr=95 pr=0 pw=0 time=2004 us)
878         TABLE ACCESS FULL KEY_DIM_LC_2 (cr=95 pr=0 pw=0 time=2668 us)
837         TABLE ACCESS FULL KEY_DIM_LC_3 (cr=95 pr=0 pw=0 time=2545 us)

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>control file sequential read</td>
<td>5</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>direct path write temp</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>log file sync</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

*******************************************************************************

78
select count(*), sum(f.measure)
from randfact_9 f,
key_dim_la_1 da1, key_dim_la_2 da2, key_dim_la_3 da3,
key_dim_lb_1 db1, key_dim_lb_2 db2, key_dim_lb_3 db3,
key_dim_lc_1 dc1, key_dim_lc_2 dc2, key_dim_lc_3 dc3,
date_dim dd
where dd.date_key = f.date_key
  and dd.nyear = 2007
  and dd.nmonth = 6
and da1.surrogate_key_1 = f.surrogate_key_a1
and da2.surrogate_key_2 = f.surrogate_key_a2
and da3.surrogate_key_3 = f.surrogate_key_a3
and db1.surrogate_key_1 = f.surrogate_key_b1
and db2.surrogate_key_2 = f.surrogate_key_b2
and db3.surrogate_key_3 = f.surrogate_key_b3
and dc1.surrogate_key_1 = f.surrogate_key_c1
and dc2.surrogate_key_2 = f.surrogate_key_c2
and dc3.surrogate_key_3 = f.surrogate_key_c3
and da1.int_property_1 >= 94
and da2.int_property_2 >= 94
and da3.int_property_3 >= 94
and db1.int_property_1 >= 94
and db2.int_property_2 >= 94
and db3.int_property_3 >= 94
and dc1.int_property_1 >= 94
and dc2.int_property_2 >= 94
and dc3.int_property_3 >= 94

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.06</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>3.06</td>
<td>64.70</td>
<td>38778</td>
<td>54951</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>3</td>
<td>3.12</td>
<td>64.76</td>
<td>38778</td>
<td>54951</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 56

Rows Row Source Operation
------- ---------------------------------------------------
1 SORT AGGREGATE (cr=54951 pr=38778 pw=0 time=64700038 us)
2971 HASH JOIN (cr=54951 pr=38778 pw=0 time=64700750 us)
2971 HASH JOIN (cr=54859 pr=38778 pw=0 time=64696543 us)
2971 HASH JOIN (cr=54767 pr=38778 pw=0 time=64692767 us)
2971 HASH JOIN (cr=54675 pr=38778 pw=0 time=64691001 us)
2971 HASH JOIN (cr=54580 pr=38778 pw=0 time=64686892 us)
2971 HASH JOIN (cr=54485 pr=38778 pw=0 time=64679700 us)
<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message to client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net message from client</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>38778</td>
<td>0.05</td>
<td>62.20</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
</tr>
</tbody>
</table>

************************************************************************************