WIKIPEDIA

Slowly changing dimension

Dimensions in data management and data warehousing contain relatively static data about such entities as geographical locations, customers, or products. Data captured by **Slowly Changing Dimensions (SCDs)** change slowly but unpredictably, rather than according to a regular schedule.[1]

Some scenarios can cause Referential integrity problems.

For example, a database may contain a fact table that stores sales records. This fact table would be linked to dimensions by means of foreign keys. One of these dimensions may contain data about the company's salespeople: e.g., the regional offices in which they work. However, the salespeople are sometimes transferred from one regional office to another. For historical sales reporting purposes it may be necessary to keep a record of the fact that a particular sales person had been assigned to a particular regional office at an earlier date, whereas that sales person is now assigned to a different regional office.

Dealing with these issues involves SCD management methodologies referred to as Type 0 through 6. Type 6 SCDs are also sometimes called Hybrid SCDs.

Contents

Type 0: retain original

The **Type 0** dimension attributes never change and are assigned to attributes that have durable values or are described as 'Original'. Examples: *Date of Birth*, *Original Credit Score*. Type 0 applies to most Date Dimension attributes[2]

Type 1: overwrite

This methodology overwrites old with new data, and therefore does not track historical data.

Example of a supplier table:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

In the above example, Supplier_Code is the natural key and Supplier_Key is a surrogate key. Technically, the surrogate key is not necessary, since the row will be unique by the natural key (Supplier_Code). However, to optimize performance on joins use integer rather than character keys (unless the number of bytes in the character key is less than the number of bytes in the integer key).

If the supplier relocates the headquarters to Illinois the record would be overwritten:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

The disadvantage of the Type 1 method is that there is no history in the data warehouse. It has the advantage however that it's easy to maintain.

If one has calculated an aggregate table summarizing facts by state, it will need to be recalculated when the Supplier_State is changed.[1]

Type 2: add new row

This method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys and/or different version numbers. Unlimited history is preserved for each insert.

For example, if the supplier relocates to Illinois the version numbers will be incremented sequentially:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Version.
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

Another method is to add 'effective date' columns.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
124	ABC	Acme Supply Co	IL	22-Dec-2004	NULL

The null End_Date in row two indicates the current tuple version. In some cases, a standardized surrogate high date (e.g. 9999-12-31) may be used as an end date, so that the field can be included in an index, and so that null-value substitution is not required when querying.

And a third method uses an effective date and a current flag.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Effective_Date	Current_Flag
123	ABC	Acme Supply Co	CA	01-Jan-2000	N
124	ABC	Acme Supply Co	IL	22-Dec-2004	Y

The Current_Flag value of 'Y' indicates the current tuple version.

Transactions that reference a particular surrogate key (Supplier_Key) are then permanently bound to the time slices defined by that row of the slowly changing dimension table. An aggregate table summarizing facts by state continues to reflect the historical state, i.e. the state the supplier was in at the time of the transaction; no update is needed. To reference the entity via the natural key, it is necessary to remove the unique constraint making Referential integrity by DBMS impossible.

If there are retroactive changes made to the contents of the dimension, or if new attributes are added to the dimension (for example a Sales_Rep column) which have different effective dates from those already defined, then this can result in the existing transactions needing to be updated to reflect the new situation. This can be an expensive database operation, so Type 2 SCDs are not a good choice if the dimensional model is subject to change.[1]

Type 3: add new attribute

This method tracks changes using separate columns and preserves limited history. The Type 3 preserves limited history as it is limited to the number of columns designated for storing historical data. The original table structure in Type 1 and Type 2 is the same but Type 3 adds additional columns. In the following example, an additional column has been added to the table to record the supplier's original state - only the previous history is stored.

Supplier_Key	Supplier_Code	Supplier_Name	Original_Supplier_State	Effective_Date	Current_Supplier_State
123	ABC	Acme Supply Co	CA	22-Dec-2004	IL

This record contains a column for the original state and current state—cannot track the changes if the supplier relocates a second time.

One variation of this is to create the field Previous_Supplier_State instead of Original_Supplier_State which would track only the most recent historical change.[1]

Type 4: add history table

The **Type 4** method is usually referred to as using "history tables", where one table keeps the current data, and an additional table is used to keep a record of some or all changes. Both the surrogate keys are referenced in the Fact table to enhance query performance.

For the above example the original table name is **Supplier** and the history table is **Supplier_History.**

Supplier

Supplier_key	Supplier_Code	Supplier_Name	Supplier_State
124	ABC	Acme & Johnson Supply Co	IL

Supplier_History

Supplier_key	Supplier_Code	Supplier_Name	Supplier_State	Create_Date
123	ABC	Acme Supply Co	CA	14-June-2003
124	ABC	Acme & Johnson Supply Co	IL	22-Dec-2004

This method resembles how database audit tables and change data capture techniques function.

Type 6

The **Type 6** method combines the approaches of types 1, 2 and 3 (1 + 2 + 3 = 6). One possible explanation of the origin of the term was that it was coined by Ralph Kimball during a conversation with Stephen Pace from Kalido. Ralph Kimball calls this method "Unpredictable Changes with Single-Version Overlay" in *The Data Warehouse Toolkit*.[1]

The Supplier table starts out with one record for our example supplier:

Supplier_Key	Row_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	1	ABC	Acme Supply Co	CA	CA	01-Jan-2000	31-Dec-9999	Y

The Current_State and the Historical_State are the same. The optional Current_Flag attribute indicates that this is the current or most recent record for this supplier.

When Acme Supply Company moves to Illinois, we add a new record, as in Type 2 processing, however a row key is included to ensure we have a unique key for each row:

Supplier_Key	Row_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	1	ABC	Acme Supply Co	CA	CA	01-Jan-2000	21-Dec-2004	N
123	2	ABC	Acme Supply Co	IL	CA	22-Dec-2004	31-Dec-9999	Y

We overwrite the Current_Flag information in the first record (Row_Key = 1) with the new information, as in Type 1 processing. We create a new record to track the changes, as in Type 2 processing. And we store the history in a second State column (Historical_State), which incorporates Type 3 processing.

For example, if the supplier were to relocate again, we would add another record to the Supplier dimension, and we would overwrite the contents of the Current_State column:

Supplier_Key	Row_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	1	ABC	Acme Supply Co	CA	CA	01-Jan-2000	21-Dec-2004	N
123	2	ABC	Acme Supply Co	IL	CA	22-Dec-2004	03-Feb-2008	N
123	3	ABC	Acme Supply Co	NY	IL	04-Feb-2008	31-Dec-9999	Y

Type 2 / type 6 fact implementation

Type 2 surrogate key with type 3 attribute

In many Type 2 and Type 6 SCD implementations, the surrogate key from the dimension is put into the fact table in place of the natural key when the fact data is loaded into the data repository.[1] The surrogate key is selected for a given fact record based on its effective date and the Start_Date and End_Date from the dimension table. This allows the fact data to be easily joined to the correct dimension data for the corresponding effective date.

Here is the Supplier table as we created it above using Type 6 Hybrid methodology:

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	IL	CA	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	IL	04-Feb-2008	31-Dec-9999	Y

Once the Delivery table contains the correct Supplier_Key, it can easily be joined to the Supplier table using that key. The following SQL retrieves, for each fact record, the current supplier state and the state the supplier was located in at the time of the delivery:

```
SELECT
  delivery.delivery_cost,
  supplier.supplier_name,
  supplier.historical_state,
  supplier.current_state
FROM delivery
INNER JOIN supplier
  ON delivery.supplier_key = supplier.supplier_key
```

Pure type 6 implementation

Having a Type 2 surrogate key for each time slice can cause problems if the dimension is subject to change.[1]

A pure Type 6 implementation does not use this, but uses a Surrogate Key for each master data item (e.g. each unique supplier has a single surrogate key).

This avoids any changes in the master data having an impact on the existing transaction data.

It also allows more options when querying the transactions.

Here is the Supplier table using the pure Type 6 methodology:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
456	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
456	ABC	Acme Supply Co	IL	22-Dec-2004	03-Feb-2008
456	ABC	Acme Supply Co	NY	04-Feb-2008	31-Dec-9999

The following example shows how the query must be extended to ensure a single supplier record is retrieved for each transaction.

```
SELECT
  supplier.supplier_code,
  supplier.supplier_state
FROM supplier
INNER JOIN delivery
  ON supplier.supplier_key = delivery.supplier_key
AND delivery.delivery_date BETWEEN supplier.start_date AND supplier.end_date
```

A fact record with an effective date (Delivery_Date) of August 9, 2001 will be linked to Supplier_Code of ABC, with a Supplier_State of 'CA'. A fact record with an effective date of October 11, 2007 will also be linked to the same Supplier_Code ABC, but with a Supplier_State of 'IL'.

While more complex, there are a number of advantages of this approach, including:

1. Referential integrity by DBMS is now possible, but one cannot use Supplier_Code as foreign key on Product table and using Supplier_Key as foreign key each product is tied on specific time slice.
2. If there is more than one date on the fact (e.g. Order Date, Delivery Date, Invoice Payment Date) one can choose which date to use for a query.
3. You can do "as at now", "as at transaction time" or "as at a point in time" queries by changing the date filter logic.
4. You don't need to reprocess the Fact table if there is a change in the dimension table (e.g. adding additional fields retrospectively which change the time slices, or if one makes a mistake in the dates on the dimension table one can correct them easily).
5. You can introduce bi-temporal dates in the dimension table.
6. You can join the fact to the multiple versions of the dimension table to allow reporting of the same information with different effective dates, in the same query.

The following example shows how a specific date such as '2012-01-01 00:00:00' (which could be the current datetime) can be used.

```
SELECT
  supplier.supplier_code,
  supplier.supplier_state
FROM supplier
INNER JOIN delivery
  ON supplier.supplier_key = delivery.supplier_key
 AND '2012-01-01 00:00:00' BETWEEN supplier.start_date AND supplier.end_date
```

Both surrogate and natural key

An alternative implementation is to place *both* the surrogate key and the natural key into the fact table.[3] This allows the user to select the appropriate dimension records based on:

- the primary effective date on the fact record (above),
- the most recent or current information,
- any other date associated with the fact record.

This method allows more flexible links to the dimension, even if one has used the Type 2 approach instead of Type 6.

Here is the Supplier table as we might have created it using Type 2 methodology:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	IL	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	04-Feb-2008	31-Dec-9999	Y

The following SQL retrieves the most current Supplier_Name and Supplier_State for each fact record:

```
SELECT
  delivery.delivery_cost,
```

```
  supplier.supplier_name,
  supplier.supplier_state
FROM delivery
INNER JOIN supplier
  ON delivery.supplier_code = supplier.supplier_code
WHERE supplier.current_flag = 'Y'
```

If there are multiple dates on the fact record, the fact can be joined to the dimension using another date instead of the primary effective date. For instance, the Delivery table might have a primary effective date of Delivery_Date, but might also have an Order_Date associated with each record.

The following SQL retrieves the correct Supplier_Name and Supplier_State for each fact record based on the Order_Date:

```
SELECT
  delivery.delivery_cost,
  supplier.supplier_name,
  supplier.supplier_state
FROM delivery
INNER JOIN supplier
  ON delivery.supplier_code = supplier.supplier_code
 AND delivery.order_date BETWEEN supplier.start_date AND supplier.end_date
```

Some cautions:

- Referential integrity by DBMS is not possible since there is not a unique key to create the relationship.
- If relationship is made with surrogate to solve problem above then one ends with entity tied to a specific time slice.
- If the join query is not written correctly, it may return duplicate rows and/or give incorrect answers.
- The date comparison might not perform well.
- Some Business Intelligence tools do not handle generating complex joins well.
- The ETL processes needed to create the dimension table needs to be carefully designed to ensure that there are no overlaps in the time periods for each distinct item of reference data.

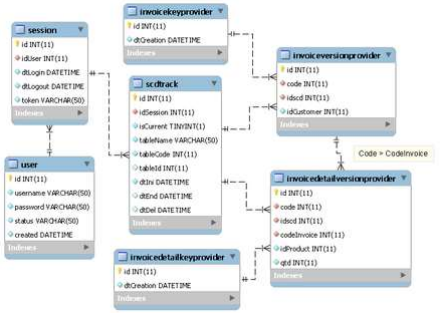
Combining types

Different SCD Types can be applied to different columns of a table. For example, we can apply Type 1 to the Supplier_Name column and Type 2 to the Supplier_State column of the same table.

See also

- Change data capture
- Temporal database
- Log trigger
- Entity–attribute–value model - Vertical

- Multitenancy



Scd model example

Notes

1. Kimball, Ralph; Ross, Margy. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*.
2. http://www.kimballgroup.com/2013/02/design-tip-152-slowly-changing-dimension-types-0-4-5-6-7/
3. Ross, Margy; Kimball, Ralph (March 1, 2005). "Slowly Changing Dimensions Are Not Always as Easy as 1, 2, 3" (http://intelligent-enterprise.informationweek.com/showArticle.jhtml;jsessionid=VQABJR4PDJSW1QE1GHPSKH4ATMY32JVN?articleID=59301280). *Intelligent Enterprise*.

References

- Bruce Ottmann, Chris Angus: *Data processing system*, US Patent Office, Patent Number 7,003,504 (http://patft.uspto.gov/netacgi/nph-Parser?u=%2Fnetahtml%2Fsrchnum.htm&Sect1=PTO1&Sect2=HITOFF&p=1&r=1&l=50&f=G&d=PALL&s1=7003504.PN.&OS=PN/7003504&RS=PN/7003504). February 21, 2006
- Ralph Kimball:*Kimball University: Handling Arbitrary Restatements of History* [1] (http://www.kimballgroup.com/2005/03/slowly-changing-dimensions-are-not-always-as-easy-as-1-2-3/). December 9, 2007