**Pengertian Database dan Data warehouse**

Antara database vs data warehouse memiliki beberapa perbedaan dalam pengertian seperti berikut ini:

* Pengertian Database

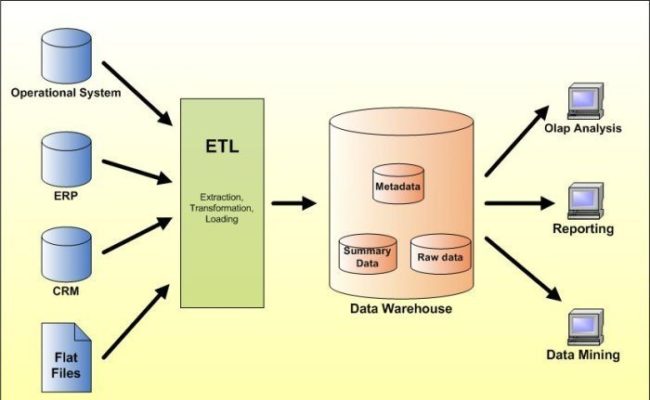
Database

Database merupakan kumpulan data berupa file, arsip, atau tabel yang tersusun sedemikian rupa menurut aturan tertentu, saling terhubung dan tersimpan dalam media elektronik yang mana pengguna mudah dalam mengelolanya serta mudah dalam mendapatkan informasi.

Database memberikan kecepatan dan kemudahan dalam menyeleksi data sehingga mendapatkan informasi yang diperlukan dalam waktu yang singkat. Namun kecepatan memperoleh informasi tergantung pada perancangan database.

Database bisa dipergunakan secara bersama-sama yang terpusat pada satu server, yang mana aplikasinya bisa dihubungkan ke beberapa komputer lainnya. Hal ini memudahkan kinerja di perusahaan/instansi, misalnya saat ingin mengontrol data atau keperluan update data. Keuntungan lain dari database yang terpusat adalah penghematan biaya karena perusahaan tidak perlu membeli beberapa database untuk beberapa komputer.

* Pengertian Data Warehouse

Data Warehouse

Data warehouse merupakan sekumpulan data yang berasal dari berbagai sumber yang tersimpan dalam gudang data dalam kapasitas besar yang dipergunakan dalam proses pengambilan keputusan .

Data tersimpan berdasarkan subyek. Contohnya adalah pada sebuah perusahaan manufaktur, maka subyek pentingnya antara lain penjualan barang, penyimpanan barang serta pengangkutan barang.

Data yang tersimpan meliputi berbagai sistem operasional. Maka dari itu bisa memungkinkan terjadi beberapa perbedaan, yakni dalam representasi kode, konvensi penamaan, atribut data serta pengukuran data. Perbedaan-perbedaan tersebut harus  disamakan dahulu menurut standar tertentu sehingga data dapat terintegrasi.

**Perbedaan Database dan Data warehouse**

Perbedaan database vs data warehouse secara detail dijelaskan berikut ini:

* Database dipergunakan dalam Transaksional Pengolahan Online (OLTP) dan dapat pula dipergunakan untuk kebutuhan data warehouse
* Data warehouse dipergunakan dalam Online Analytical Processing (OLAP) yang mampu membaca data historis bagi pengguna dalam proses pengambilan keputusan
* Tabel dan gabungan dalam database sangat kompleks sehingga dapat mengurangi data berlebihan serta menghemat ruang penyimpanan.
* Tabel dan gabungan dalam data warehouse tergolong sederhana untuk mengurangi waktu respon dan permintaan analitis.
* Database berorientasi pada pelanggan (customer oriented) dan dipergunakan dalam proses query serta transaksi, misalnya oleh kasir, pramuniaga, serta para profesional IT.
* Data warehouse berorientasi pada pasar (market oriented) dan dipergunakan dalam proses analisa data oleh orang-orang yang memiliki profesi knowlegde field, misalnya eksekutif, manager, serta para ahli analis data.
* Database mengatur dan mengolah data secara detil sehingga mudah dipergunakan dalam proses pengambilan keputusan.
* Data warehouse mengatur dan mengolah data-data historis dalam jumlah besar dan menyediakan beberapas fasilitas untuk keperluan menjumlah penjualan total serta menggabungkan penjualan pada tiap area, tiap bulan, dan seterusnya. Data warehouse menyimpan dan mengolah data menurut tingkat granularitas yang berbeda.
* Sistem database umumnya mengambil pemodelan data “entity relationship” serta desain yang berorientasi pada aplikasi.
* Sistem data warehouse umumnya mengambil pemodelan “snowflake” atau “star” serta desain yang berorientasi pada subyek.
* Database terfokus pada data real time dalam suatu organisasi atau instansi tanpa mengacu pada data-data historus maupun data-data dari organisasi lainnya.
* Data warehouse sering menjangkau berbagai tipe skema database dikarenakan proses evolusioner dari sebuah organisasi. Data warehouse juga berhubungan dengan berbagai informasi yang didapatkan dari beberapa organisasi lainnya, yang mana menggabungkan dari berbagai tempat penyimpanan data. Hal ini akan menyebabkan jumlah volume yang sangat besar sehingga data warehouse disimpan dalam beberapa tempat penyimpanan.
* Pola akses pada database terutama terdiri dari transaksi atomik yang pendek dan sistem seperti ini membutuhkan mekanisme “concurency control” serta “recovery”.
* Pola akses pada data warehouse sebagian besar adalah “read only”. Hal ini karena sebagian besar data yang tersimpan adalah data-data historis dan hanya sedikit data-data baru).

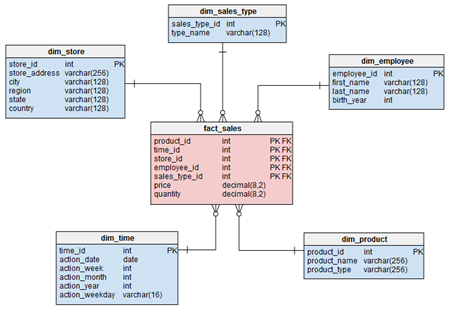
Demikian penjelasan mengenai database vs data warehouse dalam perbedaan-perbedaannya. Semoga bermanfaat.

Model yang sering digunakan di dalam data warehouse saat ini adalah skema bintang dan skema *snowflake*. Masing-masing model tentunya memiliki kelebihan dan kekurangannya masing-masing. Dalam artikel ini dijelaskan dengan detil mengenai perbedaan kedua skema tersebut. Selain itu dijelaskan pula kondisi-kondisi yang sesuai di dalam mengimplementasikan skema bintang maupun skema *snowflake*.

Skema bintang dan skema *snowflake* adalah sarana untuk mengorganisir data mart – data mart atau gudang-gudang data dengan menggunakan basis data relasional. Kedua skema tersebut menggunakan tabel-tabel dimensi untuk mendeskripsikan data-data yang terdapat di dalam tabel fakta.

Setiap perusahaan pada umumnya menjual produk, pengetahuan, maupun jasa. Sehingga sistem penjualan adalah sebuah sistem yang terdapat di sebagian besar perusahaan. Berikut ini dijelaskan mengenai model penjualan baik skema bintang maupun skema *snowflake*.

**Skema Bintang**

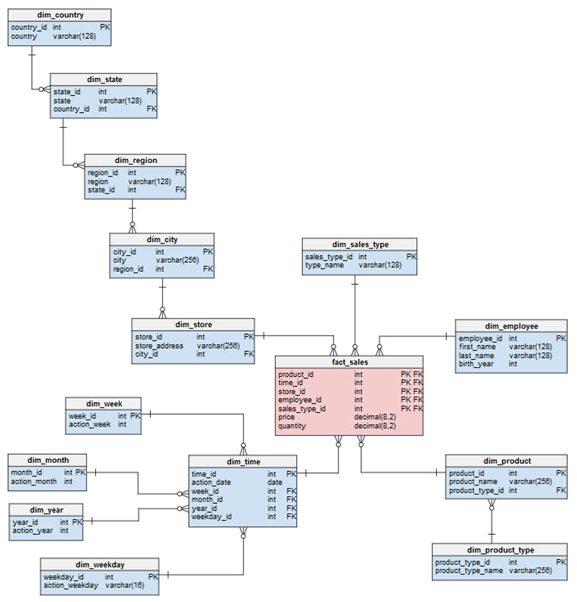
[](http://mti.binus.ac.id/files/2017/04/gambarrrrr.png)

Gambar 1. Skema Bintang

Karakteristik utama dari skema bintang adalah bahwa tabel dimensinya tidak dinormalisasi. Pada model di atas, tabel fakta fact\_sales (warna merah muda)berisi data-data yang diekstrakdari database operasional. Sedangkan tabel yang berwarna biru muda adalah tabel dimensi. Pada gambar di atas terdapat lima tabel dimensi yaitu dim\_sales\_type, dim\_store, dim\_employee, dim\_product, dan dim\_time.

Dari model ini, kita dapat dengan mudah melihat mengapa skema ini disebut ‘skema bintang’, karena model tersebut terlihat seperti bintang, dengan tabel dimensi yang mengelilingi tabel fakta

**Skema*Snowflake***

[](http://mti.binus.ac.id/files/2017/04/gambarrrrrrr.png)

Gambar 2. Skema *Snowflake*

Skema *snowflake*juga menyimpan data yang sama seperti pada skema bintang. Tabel fakta yang digunakan pada skema bintang maupun pada skema *snowflake* berisi *field-field* yang sama. Perbedaan utama antara skema bintang dan skema snowflake adalah semua tabel dimensi pada skema *snowflake* telah dinormalisasi. Proses normalisasi tabel-tabel dimensi pada skema *snowflake* ini disebut dengan proses *snowflaking*,sehingga tampilan tabel-tabel pada skema *snowflake* bentuknya menyerupai *snowflake*.

Selain itu, perbedaan lainnya adalah dalam hal kompleksitas *query*-nya. Skema *snowflake* memiliki kompleksitas *query* yang lebih kompleks dibandingkan dengan skema bintang. Penjelasan mengenai kedua perbedaan tersebut adalah sebagai berikut:

1. **Normalisasi**

Seperti yang telah disebutkan di atas, normalisasi adalah perbedaan utama antara skema bintang dengan skema *snowflake*. Beberapa hal yang harus diperhatikan adalah sebagai berikut:

* Skema *snowflake* menggunakan ruang penyimpanan yang lebih kecil dibandingkan ruang penyimpanan pada skema bintang. Hal ini disebabkan karena tabel-tabel dimensi yang telah dinormalisasi memiliki record-record yang efisien karena tidak terjadi pengulangan data-data yang sama.
* Tabel dimensi yang tidak dinormalisasi dapat menyebabkan masalah integritas data. Karena data-data yang sama bisa muncul berulang-ulang, bahkan bisa juga terjadi kesalahan pengetikan pada data-data yang sama tersebut. Sehingga pada skema bintang harus dilakukan pengecekan dan *maintenance* secara berkala.
* Penyimpanan data pada skema *snowflake* lebih terorganisir dan lebih rapi dibandingkan dengan skema bintang.

1. **Kompleksitas *Query***

Sebelum masuk ke dalam penjelasan mengenai perbandingan kompleksitas *query* antara skema bintang dan skema *snowflake*, terlebih dahulu diberikan contoh perintah *query* yang digunakan untuk menghitung jumlah telepon yang terjual di toko-toko di kota Berlin sepanjang tahun 2016. Perintah *query*-nya adalah sebagai berikut:

* **Skema Bintang**

|  |
| --- |
| SELECT  dim\_store.store\_address,  SUM(fact\_sales.quantity) ASquantity\_sold    FROM  fact\_sales  INNERJOINdim\_product ONfact\_sales.product\_id = dim\_product.product\_id  INNERJOINdim\_time ONfact\_sales.time\_id = dim\_time.time\_id  INNERJOINdim\_store ONfact\_sales.store\_id = dim\_store.store\_id    WHERE  dim\_time.action\_year = 2016  ANDdim\_store.city = ‘Berlin’  ANDdim\_product.product\_type = ‘phone’    GROUPBY  dim\_store.store\_id,  dim\_store.store\_address |

* **Skema *Snowflake***

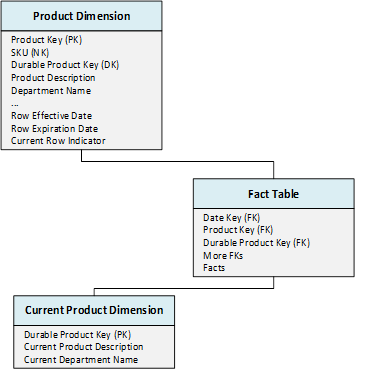
|  |
| --- |
| SELECT    dim\_store.store\_address,    SUM(fact\_sales.quantity) ASquantity\_sold  FROM    fact\_sales    INNERJOINdim\_product ONfact\_sales.product\_id = dim\_product.product\_id    INNERJOINdim\_product\_type ONdim\_product.product\_type\_id = dim\_product\_type.product\_type\_id    INNERJOINdim\_time ONfact\_sales.time\_id = dim\_time.time\_id    INNERJOINdim\_year ONdim\_time.year\_id = dim\_year.year\_id    INNERJOINdim\_store ONfact\_sales.store\_id = dim\_store.store\_id    INNERJOINdim\_city ONdim\_store.city\_id = dim\_city.city\_id  WHERE    dim\_year.action\_year = 2016    ANDdim\_city.city = ‘Berlin’    ANDdim\_product\_type.product\_type\_name = ‘phone’  GROUPBY    dim\_store.store\_id,    dim\_store.store\_address  Seperti terlihat dari dua perintah *query* di atas, query pada skema *snowflake* lebih kompleks dibandingkan dengan *query* yang digunakan pada skema bintang. Hal ini disebabkan karena pada skema *snowflake* melibatkan lebih banyak tabel sebagai akibat dari proses normalisasi, sehingga menggunakan perintah JOIN yang lebih banyak dibandingan dengan skema bintang.  Pada skema bintang,hanya ada satu perintah JOIN untuk setiap tabel dimensi. Namun jika tidak melibatkan tabel dimensi, maka perintah JOIN tidak perlu digunakan. Sedangkan pada skema *snowflake*, jumlah perintah JOIN yang digunakan bisa bervariasi tergantung kebutuhan.  Menggabungkan dua tabel memerlukan waktu karena Database Management System (DBMS) memerlukan waktu yang lebih lama untuk memproses permintaan query tersebut. Sebagai contoh tabel **dim\_store**dan **dim\_city**terlihat berdekatan pada gambar skema di atas, namun secara fisik kedua tabel tersebut mungkin tidak terletak berdekatan dalam media penyimpanan. Akan tetapi data-data yang terletak pada satu tabel, kemungkinan besar secara fisik data-data tersebut terletak berdekatan dalam media penyimpanan.  Umumnya, sebuah *query* yang berjalan dalam data mart dengan skema *snowflake* akan dieksekusi lebih lambat. Namun hal ini tidak menimbulkan masalah yang besar jika hasil yang ditampilkan hanya dalam waktu antara 1 mili detik sampai dengan 1 detik. Hal ini dapat menjadi masalah yang serius jika hasil yang ditampilkan memerlukan waktu 5 detik atau lebih.  **Cara-Cara Mempercepat Eksekusi *Query***  Hal-hal yang dapat dilakukan untuk mempercepat proses *query* dalam laporan adalah sebagai berikut:   * Menyimpan data ke dalam level yang kita perlukan dalam laporan. Hal ini dapat memampatkan ukuran data secara signifikan. Untuk itu diperlukan prosedur-prosedur ETL (Extract, Transform, Load) yang dapat mentransformasi data ke dalam struktur dengan tepat dan efisien. * Membuat sebuah tempat penyimpanan data yang terpusat untuk semua data perusahaan, tidak hanya untuk data-data bagian tertentu saja. * User hanya mengakses data-data yang diperlukan saja untuk analisa dan laporan, tidak perlu semua data yang dapat memperlambat proses eksekusi *query*.   **Skema Apa Yang Terbaik?**              Setiap skema baik skema bintang maupun skema *snowflake* memiliki kelebihan dan kekurangannya masing-masing. Untuk dapat menentukan skema apa yang sesuai dengan kebutuhan kita, berikut ini dijelaskan kelebihan dan kekurangan masing-masing skema.   1. **Berikut adalah kelebihan dan kekurangan model skema bintang.**   Kelebihan model skema bintang:   * Cenderung mudah dipahami karena modelnya yang lebih sederhana * Memudahkan mencari isi karena kesederhanaannya dengan cara melihat *step by step*dari masing-masing dimensinya * Proses *query* lebih cepat pada saat proses OLAP   Kekurangan model skema bintang:   * Ukuran data lebih besar karena ada data yang disimpan ulang * *Maintenance* dan *update* lebih sulit  1. **Berikut adalah kelebihan dan kekurangan model skema*snowflake*.**   Kelebihan model skema*snowflake*:   * Ukuran data lebih kecil di dalam tempat penyimpanan * Lebih mudah dilakukan *maintenance* dan *update* * Proses *query* lebih cepat pada saat proses ETL   Kekurangan model skema*snowflake*:   * Cenderung lebih sulit dipahami karena kompleksitasnya * Sulit mencari data yang dibutuhkan karena melihat strukturnya yang kompleks dan bercabang-cabang   **Skema Apa yang harus Dipilih?**              Berikut ini akan dijelaskan mengenai kondisi yang tepat di dalam menerapkan skema bintang maupun skema *snowflake*. Penjelasannya adalah sebagai berikut:   1. Penerapan skema bintang   Penerapan skema bintang adalah pada data mart suatu perusahaan. Data mart adalah bagian dari data warehouse. Data mart biasanya dibuat pada tiap departemen pada perusahaan tersebut. Tujuan dari pembuatan data mart adalah supaya karyawan atau staff yang terkait pada suatu departemen tertentu dapat mengakses data yang diperlukan dengan cepat tanpa harus melakukan query pada database utama sehingga dapat mengurangi beban server utama.   1. Penerapan skema *snowflake*  * **Dalam data warehouse**   Penerapan skema snowflake dalam data warehouse suatu perusahaan dapat menghemat ruang penyimpanan yang dibutuhkan.   * **Ketika tabel-tabel dimensi memerlukan ruang penyimpanan yang besar**   Dalam sebagian besar kasus, umumnya tabel fakta memerlukan ruang penyimpanan yang lebih besar daripada tabel dimensi. Tabel fakta juga berkembang lebih banyak dibanding tabel dimensi. Akan tetapi ada beberapa kondisi dimana ukuran tabel dimensi lebih besar dibandingkan dengan tabel fakta. Contohnya, tabel-tabel dimensi dapat berisi atribut-atribut redundan, namun diperlukan di dalam proses *query* laporan.  Dari kelebihan dan kekurangan dari masing-masing model yang telah dijelaskan sebelumnya, diharapkan dapat membantu *user* di dalam menentukan model skema yang tepat yang dapat digunakan di dalam membuat dan menerapkan data warehouse.  **Sumber:**   1. http://www.vertabelo.com/blog/technical-articles/data-warehouse-modeling-star-schema-vs-snowflake-schema 2. https://levinbosz.wordpress.com/2014/01/02/star-schema-dan-snowflake-schema-baru/ |

# SLOWLY CHANGING DIMENSION PADA DATA WAREHOUSE’

**Apakah pengertian dari ‘Slowly Changing Dimension’ ?**

Dimensi yang berubah perlahan dari waktu ke waktu, dimensi ini menyimpan nilai atau data lama dari atribut sebuah dimensi. Dimensi adalah struktur yang mengkategorikan kumpulan informasi sehingga. Dimensi dalam manajemen data dan data warehouse (gudang data) berisi data yang relatif statis. Namun data dari dimensi dapat berubah secara perlahan dari waktu ke waktu dan pada interval yang tidak dapat diprediksi. Jenis data dari dimensi dapat dinamakan dengan Slowly Changing Dimension (SCD). Nilai  yang  lama tersebut dapat disimpan untuk sebagai sebuah “sejarah perubahan nilai” dari atribut sebuah  dimensi  yang  berubah  tersebut.Penyimpanan  nilai  yang  lama  ini  dapat  dilakukan  dengan cara membuat sebuah kolom yang khusus untuk menyimpan nilai yang lama tersebut.

Dalam data warehouse memiliki tabel dimensi yang merupakan kunci utama secara unik untuk mengidentifikasikan setiap catatan (record) dan bagian informasi lainnya yang dikenal sebagai dimensi.  Kunci utama (primary key) akan terhubung denga tabel fakta, oleh karena itu, data dari primary key menjadi penting untuk dapat diperbaharui sesuai dengan kebutuhan proses transaksi data saat ini dan data historis yang telah disimpan seperti pada gambar 1. Relasi Tabel Fakta dan Tabel Dimensi.

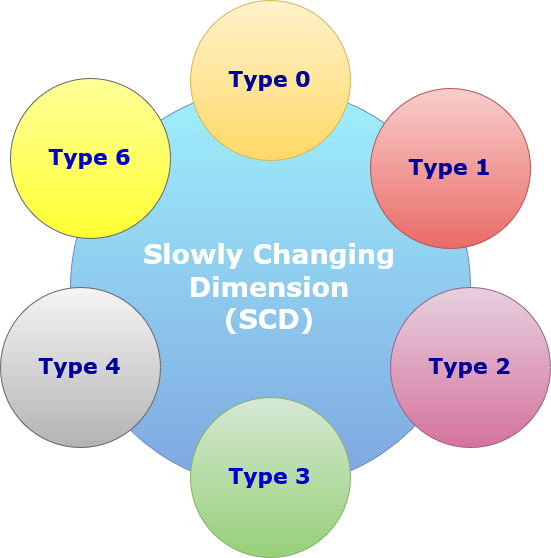
[](http://mti.binus.ac.id/files/2018/05/1111km1.png)

Gambar 1.  Relasi Tabel Fakta dan Tabel Dimensi

**Apa saja tipe-tipe dari Slowly Changing Dimension ?**

Terdapat banyak pendeketan mengenai tipe-tipe dari SCD, Namun yang paling populer pendeketan dari SCD adalah :

* Tipe 0 : Metode pasif.
* Tipe 1 : Mengganti nilai lama.
* Tipe 2 : Membuat catatan tambahan baru.
* Tipe 3 : Manmbahkan kolom baru.
* Tipe 4 : Menggunakan tabel historis.
* Tipe 6 : Menggabungkan pendekatan tipe 1, 2, 3 (1+2+3 = 6)

[](http://mti.binus.ac.id/files/2018/05/2222222222222km2.png)

Gambar 2. Tipe SCD

* Tipe 0 – Metode pasif. Metode ini tidak ada tindakan khusus yang dilakukan jika terjadi perubahan dalam suatu dimensi. Beberapa data dari dimensi menjadi tetap sama seperti pertama kali dimasukkan, data yang lain mungkin akan ditimpa dengan data yang baru.
* Tipe 1 – Mengganti nilai lama. Dalam metode ini tidak ada riwayat dari perubahan dimensi yang disimpan dalam database. Nilai dimensi dari data yang lama hanya diganti dengan yang baru. Jenis metode ini mudah dipelihara (maintain) dan sering digunakan untuk data yang perubahannya disebabkan oleh proses dari koreksi data, misalnya seperti penghapusan karakter khusus atau memperbaiki ejaan data.

Contoh dari metode ini ditunjukkan pada tabel 1. Terdapat tabel yang memiliki data Corporate dari kolom CustType, kemudia terdapat perubahan pada tabel  2. Kolom CustType menjadi Retail. Perubahan tersebut dapat berubaha sesuai kedaan kondisi data yang dibutuhkan saat ini.

Sebelum perubahan :

Tabel 1. Sebelum Perubahan CustType

|  |  |  |
| --- | --- | --- |
| **CustD** | **CustName** | **CustType** |
| **1** | Cust\_1 | Corporate |

Setalah perubahan :

Tabel 2. Setelah Perubahan CustType

|  |  |  |
| --- | --- | --- |
| **CustD** | **CustName** | **CustType** |
| **1** | Cust\_1 | Retail |

* Tipe 2 – Membuat catatan tambahan baru. Dalam metodologi ini semua perubahan sejarah dimensi disimpan dalam database. Melakukan perubahan atribut dengan menambahkan baris baru, menggunakan primay key yang baru ke dalam tabel dimensi selain itu terdapat penambahan kolom ‘Tanggal Efektif’ dan ‘Indikator saat ini’ dalam metode ini. Penambahan kolom ‘Indikator saat ini’ atau ‘Current Indicator’ ditulis dengan ‘Y’ dan untuk kolom ‘Tanggal Efektik’ atau ‘Effective Date’ ditulis dengan start\_date dan end\_date. Memperkenalkan tipe 2 dari SCD ini menjadi suatu bentuk metode operasi data yang membutuhkan biaya mahal sehingga belum sepenuhnya disarankan untuk menggunakannya dalam dimensi dimana tribut baru dapat ditambahkan dimasa yang akan datang.SCD tipe 2 jauh lebih fleksibel untuk menyimpan sejarah perubahan data.Karena dapat menyimpan perubahan data sebanyak apapun tanpa harus mengubah struktur tabel itu sendiri.

Sebelum perubahan :

Tabel 3. Sebelum Perubahan CustType Tipe 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustD** | **CustName** | **CustType** | **StarDate** | **EndDate** | **CurrFlag** |
| **1** | Cust\_1 | Corporate | 22-05-2018 | 31-12-9999 | Y |

Setalah perubahan :

Tabel 4. Setelah Perubahan CustType Tipe 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustD** | **CustName** | **CustType** | **StarDate** | **EndDate** | **CurrFlag** |
| **1** | Cust\_1 | Corporate | 22-05-2018 | 24-08-2018 | N |
| **2** | Cust\_1 | Retail | 25-08-2018 | 31-12-9999 | Y |

* Tipe 3 – Menambahkan kolom baru. Dalam tipe ini biasanya hanya nilai dimensi saat ini dan sebelumnya disimpan di database. Nilai baru dimuat dalam kolom ‘Sekarang/Baru’ (Current Type) dan kolom lama dimuat dalam kolom lama yang diberi nama ‘Lama/Sebelumnya’ (Previous Type). SCD tipe 3  menggunakan  kolom  untuk  menyimpan  datalama  setelah  perubahan  terjadi,  yang  menyebabkannya tidak  terlalu  fleksibel  untuk  Metode  ini  ideal  digunakan  pada  situasi  dimana  tidak  terlalu banyak  data  peruabahan  yang  ingin  disimpan,  atau  jumlah  perubahan  yang  ingindisimpan diketahui secara pasti.Metode ini juga sesuai apabila perubahan data pada atribut tersebut akan  mempengaruhi banyakbaris data lainnya, atau nilai atribut ini diubah oleh banyak baris data lain secara  bersamaan.

Sebelum perubahan :

Tabel 5. Sebelum Perubahan Kolom PerviousType

|  |  |  |  |
| --- | --- | --- | --- |
| **CustD** | **CustName** | **CurrentType** | **PreviousType** |
| **1** | Cust\_1 | Corporate | Corporate |

Setalah perubahan :

Tabel 6. Setelah Perubahan Kolom PerviousType

|  |  |  |  |
| --- | --- | --- | --- |
| **CustD** | **CustName** | **CurrentType** | **PreviousType** |
| **1** | Cust\_1 | Retail | Corporate |

* Tipe 4 – Menggunakan tabel historis. Dalam metode ini , tabel historis terpisah digunakan untuk mencari semua perubahan historis pada atribut data untuk masing-masing dimensi. Tabel dimensi utama hanya menyimpan data saat ini. Misalnya tabel customer dan customer\_history.

Sebelum perubahan :

Tabel 7. Sebelum Perubahan Data Corporate

|  |  |  |
| --- | --- | --- |
| **CustD** | **CustName** | **CustType** |
| **1** | Cust\_1 | Corporate |

Setalah perubahan :

Tabel 8. Penambahan StarDate dan EndDate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustD** | **CustName** | **CustType** | **StartDate** | **EndDate** |
| **1** | Cust\_1 | Retail | 01-01-2018 | 21-07-2018 |
| **1** | Cust\_1 | Other | 22-07-2018 | 17-05-2018 |
| **1** | Cust\_1 | Corporate | 18-05-2018 | 31-12-9999 |

* Tipe 6 – Menggabungkan pendekatan tipe 1, 2, 3 (1+2+3 = 6). Pada tipe ini memiliki dimensi dari kolom tambahan seperti :
* current\_type untuk menyimpan nilai atribut saat ini. Semua catatan sejarah untuk item atribut yang diberikan memiliki nilai saat ini yang sama.
* historical\_type untuk menyimpan nilai historis atribut. Semua catatan sejarah untuk item atribut tertentu dapat memiliki nilai yang berbeda.
* start\_date – untuk menjaga tanggal mulai dari ‘tanggal efektif’ dari sejarah atribut.
* end\_date – untuk menjaga tanggal akhir dari ‘tanggal efektif’ dari sejarah atribut.
* current\_flag – untuk menyimpan informasi tentang catatan terbaru.

Didalam metode ini menangkap perubahan dari nilai atribut dengan cara menambahkan catatan baru seperti tipe 2. Informasi dari kolom current\_type diganti dengan yang baru seperti pada tipe 1. Kemudian menyimpan sejarah histori dari perubahan data seperti tipe 3.

Tabel 9. Menggabungkan Tipe 1 – 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustD** | **CustName** | **CustType** | **HistType** | **StartDate** | **EndDate** | **CurrFlag** |
| **1** | Cust\_1 | Corporate | Retil | 01-01-2018 | 21-07-2018 | N |
| **1** | Cust\_1 | Corporate | Other | 22-07-2018 | 17-05-2018 | N |
| **1** | Cust\_1 | Corporate | Corporate | 18-05-2018 | 31-12-9999 | Y |

[**Dimensions**](https://en.wikipedia.org/wiki/Dimension_(data_warehouse)) in [data management](https://en.wikipedia.org/wiki/Data_management) and [data warehousing](https://en.wikipedia.org/wiki/Data_warehousing) contain relatively static [data](https://en.wikipedia.org/wiki/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]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-1)

Some scenarios can cause [Referential integrity](https://en.wikipedia.org/wiki/Referential_integrity) problems.

For example, a [database](https://en.wikipedia.org/wiki/Database) may contain a [fact table](https://en.wikipedia.org/wiki/Fact_table) that stores sales records. This fact table would be linked to dimensions by means of [foreign keys](https://en.wikipedia.org/wiki/Foreign_key). 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

* [1Type 0: retain original](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_0:_retain_original)
* [2Type 1: overwrite](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_1:_overwrite)
* [3Type 2: add new row](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_2:_add_new_row)
* [4Type 3: add new attribute](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_3:_add_new_attribute)
* [5Type 4: add history table](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_4:_add_history_table)
* [6Type 5](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_5)
* [7Type 6: Combined Approach](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_6:_Combined_Approach)
* [8Type 2 / type 6 fact implementation](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_2_/_type_6_fact_implementation)
  + [8.1Type 2 surrogate key with type 3 attribute](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_2_surrogate_key_with_type_3_attribute)
  + [8.2Pure type 6 implementation](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Pure_type_6_implementation)
  + [8.3Both surrogate and natural key](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Both_surrogate_and_natural_key)
* [9Combining types](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Combining_types)
* [10See also](https://en.wikipedia.org/wiki/Slowly_changing_dimension#See_also)
* [11Notes](https://en.wikipedia.org/wiki/Slowly_changing_dimension#Notes)
* [12References](https://en.wikipedia.org/wiki/Slowly_changing_dimension#References)

## Type 0: retain original[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=1)]

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]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-2)

## Type 1: overwrite[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=2)]

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](https://en.wikipedia.org/wiki/Natural_key) and Supplier\_Key is a [surrogate key](https://en.wikipedia.org/wiki/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]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-1)

## Type 2: add new row[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=3)]

This method tracks historical data by creating multiple records for a given [natural key](https://en.wikipedia.org/wiki/Natural_key) in the dimensional tables with separate [surrogate keys](https://en.wikipedia.org/wiki/Surrogate_key) 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](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-1)

## Type 3: add new attribute[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=4)]

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]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-1)

## Type 4: add history table[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=5)]

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](https://en.wikipedia.org/wiki/Change_data_capture) techniques function.

## Type 5[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=6)]

Reserved for future expansion.

## Type 6: Combined Approach[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=7)]

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](https://en.wikipedia.org/wiki/Ralph_Kimball) during a conversation with Stephen Pace from Kalido[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]. [Ralph Kimball](https://en.wikipedia.org/wiki/Ralph_Kimball) calls this method "Unpredictable Changes with Single-Version Overlay" in *The Data Warehouse Toolkit*.[[1]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-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[[edit](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=8)]

### Type 2 surrogate key with type 3 attribute**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=9)**]**

In many Type 2 and Type 6 SCD implementations, the [surrogate key](https://en.wikipedia.org/wiki/Surrogate_key) from the dimension is put into the fact table in place of the [natural key](https://en.wikipedia.org/wiki/Natural_key) when the fact data is loaded into the data repository.[[1]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-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**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=10)**]**

Having a Type 2 surrogate key for each time slice can cause problems if the dimension is subject to change.[[1]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-KimballToolkit-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](https://en.wikipedia.org/wiki/Referential_integrity) by DBMS is now possible, but one cannot use Supplier\_Code as [foreign key](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Temporal_database#Bitemporal_Relations) 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**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Slowly_changing_dimension&action=edit&section=11)**]**

An alternative implementation is to place *both* the [surrogate key](https://en.wikipedia.org/wiki/Surrogate_key) and the [natural key](https://en.wikipedia.org/wiki/Natural_key) into the fact table.[[3]](https://en.wikipedia.org/wiki/Slowly_changing_dimension#cite_note-SCDnot123-3) 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

### "KUNCI PENGGANTI" (SURROGATE KEY) PEMROGRAMAN (WAJIB DI BACA BAGI MAHASISWA TI)

Posted by Jonathan Eka , at [12:09 PM](http://technaccountingshare.blogspot.com/2013/12/kunci-pengganti-surrogate-key.html)

Dalam tulisan kali ini saya mengawali dengan "Primary key" atau "Foreign key" yang di ajarkan di pemrograman oleh para dosen, atau pun anda yang belajar dari buku pemrograman dan sebagainya.

# PRIMARY KEY

biasa di pakai dalam database adalah kode yang di inputkan oleh user.

Contoh:

* untuk master Barang, yang di gunakan sebagai primary key biasanya adalah kode barang.
* untuk master pelanggan, yang di gunakan biasanya juga kode customer.
* Master supplier dan sebagainya biasanya juga menggunakan kode yang di inputkan oleh user.

Cara seperti ini sudah sering di pakai untuk pemrograman di awal kita belajar bahasa pemrograman. Tetapi, cara seperti ini banyak mengandung kelemahan.

1. Kode yang sudah di inputkan oleh user, tidak akan bisa di edit, karena kode tersebut di gunakan untuk primary key, yang menjadi foreign key dari table lain. Jika kode tersebut di ganti, maka hubungan antar table akan menjadi rancu, dan beberapa form tidak akan jalan dengan baik karena primary key / foreign key yang di maksud telah di rubah.
2. Berhubungan dengan poin pertama, karena kode tidak bisa di ubah, saat user **salah** memasukan kode, maka kode itu tidak bisa di edit kembali. Kalau pun bisa di ubah, itu hanya di perbolehkan saat master tersebut belum di pakai dalam table transaksi mana pun.
3. Kalau pun berhasil menemukan algorithm agar kode bisa di rubah, hal itu menghabiskan waktu, karena harus membuat trigger dan sebagainya untuk mengganti semua foreign key yang berhubungan dengan kode table master yang di ganti.

Dalam artikel kali ini, saya ingin menjelaskan tentang:

# *SURROGATE KEY.*

*Surrogate (1) – Hall, Owlett and Codd (1976)*

*A surrogate represents an entity in the outside world. The surrogate is internally generated by the system but is nevertheless visible to the user or application.*

*Surrogate (2) – Wieringa and De Jonge (1991)*

*A surrogate represents an object in the database itself. The surrogate is internally generated by the system and is invisible to the user or application.*

[](http://2.bp.blogspot.com/-Pz-v7oPi1Nc/UrVNNWyR1PI/AAAAAAAAAcg/Bvvumpr0fms/s1600/surrogate+key.jpg)

 Jika di jelaskan dengan bahasa sehari-hari, jadi surrogate key itu adalah key yang di generate dari program. Biasanya menggunakan auto index number. Di kolom pertama dalam database, di gunakan field yang menghasilkan angka secara berurutan. Key ini tidak di tampilkan dalam aplikasi ataupun ke user.

Contoh konkritnya:

sebuah tabel barang (item), biasanya menggunakan kode barang untuk key (primary key). Dibandingkan dengan kode barang, surrogate key menggunakan nomor generate di kolom depan kode, dengan urutan sesuai dengan jumlah record yang di inputkan.

contoh tabel:  
**TABLE ITEM**

|  |  |  |
| --- | --- | --- |
| **ITEM\_ID (surrogate key)** | **ITEM\_CODE (unique key)** | **ITEM\_DESCRIPTION** |
| 1 | HVS70GR | Kertas HVS 70gr |
| 2 | Q80GR | Kertas Quarto 80gr |
| 3 | A470GR | Kertas A4 70gr |

Dari tabel di atas, dapat di ketahui, primary key yang di pakai adalah **ITEM\_ID**,sedangkan kode barang hanya di gunakan sebagai index *unique.***ITEM\_ID**tidak akan pernah muncul di interface, di layar yang di gunakan user, dan user tidak dapat merubah nilainya.  
  
**ITEM\_ID** yang ada di master barang, menjadi foreign key untuk tabel transaksi lainnya. Contoh tabel transaksi detail penjualan:  
  
**TABLE JUAL**

|  |  |  |
| --- | --- | --- |
| **JUAL\_DETAIL\_ID** | **ITEM\_ID** | **QTY** |
| 1 | 2 | 5 |
| 2 | 3 | 3 |
| 3 | 3 | 4 |

Dari contoh tabel di atas dapat di lihat, **JUAL\_DETAIL\_ID**menjadi primary key (surrogate key) dari tabel JUAL\_DETAIL, sedangkan **ITEM\_ID**menjadi foreign key (surrogate key).  
Jadi dalam satu transaksi, terdapat penjualan quarto 80gr sebanyak 5 unit, A4 70gr sebanyak 3 unit, dan A4 70gr sebanyak 4 unit.   
  
Kedua key tersebut tidak akan pernah di tampilkan dalam interface.

### KEUNTUNGAN MENGGUNAKAN *SURROGATE KEY:*

#### SURROGATE KEY TIDAK AKAN BERUBAH SELAMA BARIS ADA.

### **INI MEMILIKI KEUNTUNGAN SEBAGAI BERIKUT:**

* Aplikasi tidak bisa kehilangan referensi mereka (karena identifier tidak pernah berubah).
* Data primary key atau foreign key selalu dapat dimodifikasi (kode supplier, barang, dsb).

#### PERFORMANCE:

untuk performa, karena surrogate key hanya menggunakan angka generate dari system, maka field ini bisa menggunakan integer 4 digit saja. Jadi saat melakukan query, proses akan lebih cepat, karena index hanya terdapat di satu kolom saja.

### KERUGIAN:

* Nilai yang di keluarkan oleh surrogate key tidak memiliki arti sesungguhnya, karena hanya berupa angka generate. Jadi ketika kita mencari referensi dalam tabel yang berbeda, kita perlu men-join-kan beberapa tabel agar mengerti key apa yang sedang kita pakai.
* Karena key yang sebenarnya kita gantikan dengan surrogate key, sedangkan yang di namakan key itu harus unique, maka kita perlu memberi index (unique) pada kolom yang sebenarnya adalah key. jadi database akan membaca keseluruhan tabel jika di perlukan.

Jadi, di bandingkan dengan penggunakan primary key dengan menggunakan kode, saya pribadi lebih menyukai menggunakan *surrogate key.*