**Data Mart**

Data mart adalah suatu bagian pada data warehouse yang mendukung pembuatan laporan dan analisa data pada suatu unit, bagian atau operasi pada suatu perusahaan. Dalam beberapa implementasi data warehouse, data mart adalah miniature data warehouse. Data mart sering digunakan untuk memberikan informasi kepada segmen fungsional organisasi.  
Contoh umum data mart adalah untuk departemen penjualan, departemen persediaan dan pengiriman, departemen keuangan, manajemen tingkat atas, dan seterusnya. Data mart juga dapat digunakan untuk gudang data segmen data untuk mencerminkan bisnis secara geografis terletak di mana masing-masing daerah relatif otonom. Sebagai contoh, sebuah organisasi layanan yang besar mungkin memperlakukan pusat operasi regional sebagai unit usaha perorangan, masing-masing dengan data mart sendiri yang memberikan kontribusi untuk gudang data master.

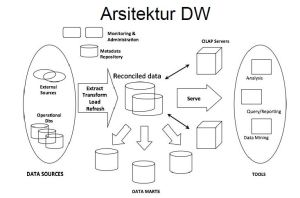
A **data mart** is a structure / access pattern specific to [data warehouse](https://en.wikipedia.org/wiki/Data_warehouse) environments, used to retrieve client-facing data. The data mart is a subset of the data warehouse and is usually oriented to a specific business line or team. Whereas data warehouses have an enterprise-wide depth, the information in data marts pertains to a single department. In some deployments, each department or business unit is considered the *owner* of its data mart including all the *hardware*, *software* and *data*.[[1]](https://en.wikipedia.org/wiki/Data_mart#cite_note-1) This enables each department to isolate the use, manipulation and development of their data. In other deployments where conformed dimensions are used, this business unit ownership will not hold true for shared dimensions like customer, product, etc.

A data mart is basically a condensed and more focused version of a data warehouse that reflects the regulations and process specifications of each business unit within an organization. Each data mart is dedicated to a specific business function or region. This subset of data may span across many or all of an enterprise’s functional subject areas. It is common for multiple data marts to be used in order to serve the needs of each individual business unit (different data marts can be used to obtain specific information for various enterprise departments, such as accounting, marketing, sales, etc.).

Data mart adalah struktur / pola akses khusus untuk lingkungan data warehouse, yang digunakan untuk mengambil data yang menghadap klien. Data mart adalah bagian dari gudang data dan biasanya berorientasi pada lini bisnis atau tim tertentu. Sedangkan gudang data memiliki kedalaman perusahaan yang luas, informasi dalam data mart berkaitan dengan satu departemen. Dalam beberapa penerapan, setiap departemen atau unit bisnis dianggap sebagai pemilik data mart termasuk semua perangkat keras, perangkat lunak, dan data. [1] Ini memungkinkan setiap departemen untuk mengisolasi penggunaan, manipulasi, dan pengembangan data mereka. Dalam penerapan lain di mana dimensi yang sesuai digunakan, kepemilikan unit bisnis ini tidak akan berlaku untuk dimensi yang dibagikan seperti pelanggan, produk, dll.

Data mart pada dasarnya adalah versi gudang data yang padat dan lebih terfokus yang mencerminkan peraturan dan spesifikasi proses dari setiap unit bisnis dalam suatu organisasi. Setiap data mart didedikasikan untuk fungsi atau wilayah bisnis tertentu. Subkumpulan data ini dapat menjangkau banyak atau semua bidang subjek fungsional perusahaan. Adalah hal yang umum untuk beberapa mart data yang akan digunakan untuk melayani kebutuhan masing-masing unit bisnis (berbagai data mart dapat digunakan untuk mendapatkan informasi spesifik untuk berbagai departemen perusahaan, seperti akuntansi, pemasaran, penjualan, dll.).

**Data Warehouse**

[](https://citaekayana.files.wordpress.com/2015/10/arsitektur-dw.jpg)

Arsitektur Data Warehouse

(Sumber:<http://1.bp.blogspot.com/-hllgOEv8yCI/UjcCMxoU2hI/AAAAAAAAAKg/vLyOd46lv7c/s1600/dw1.JPG>)

Data Warehouse adalah database yang didesain khusus untuk mengerjakan proses *query*, membuat laporan dan analisa. Data yang di simpan adalah data *business history* dari sebuah organisasi /perusahaan, dimana data tersebut tidak tersimpan secara rinci/detil. Sehingga data dapat bertahan lebih lama. Sumber data pada datawa rehouse berasal dari berbagai macam *format, software, platform* dan jaringan yang beda. Data tersebut adalah hasil dari proses transaksi perusahan / organisasi sehari-hari. Karena berasal dari sumber yang berbeda beda tadi, maka data pada data warehouse harus tersimpan dalam sebuah format yang baku.

Data Warehouse juga merupakan salah satu sistem pendukung keputusan, yaitu dengan menyimpan data dari berbagai sumber, mengorganisasikannya dan dianalisa oleh para pengambil kebijakan. Akan tetapi datawarehouse tidak dapat memberikan keputusan secara langsung. Namun ia dapat memberikan informasi yang dapat membuat user menjadi lebih paham dalam membuat kebijakan strategis.

Adapun karakteristik umum yang dimiliki datawarehouse adalah :

* Data terintegrasi dari berbagai sumber yang berasal dari proses transaksional (OLTP)
* Data dibuat konsisten
* Merupakan *aggregate*data/kesimpulan data, bukan data yang terperinci
* Data bertahan lebih lama
* Data tersimpan dalam format yang tepat sehinngga proses *query*dan analisa dapat dilakukan dengan cepat
* Data bersifat *read only*

**ETL (Extraction, Transformation, Loading)**

Tiga fungsi utama yang perlu dilakukan untuk membuat data siap digunakan pada datawarehouse adalah *extraction*, *transformation* dan *loading*. Ketiga fungsi ini terdapat pada *staging area*. Pada *data staging*ini, disediakan tempat dan area dengan beberapa fungsi seperti data *cleansing, change*, *conver*t, dan menyiapkan data untuk disimpan serta digunakan oleh datawarehouse.

[](https://citaekayana.files.wordpress.com/2015/10/etl.jpg)

Bagan ETL  
(Sumber:<http://www.dbbest.com/blog/wp-content/uploads/2012/12/ETL_input_output.jpg>)

* **Extraction**

*Data Extraction* adalah proses pengambilan data yang diperlukan dari sumber datawarehouse dan selanjutnya dimasukkan pada staging area untuk diproses pada tahap berikutnya [2]. Pada fungsi ini, kita akan banyak berhubungan dengan berbagai tipe sumberdata. Format data, mesin yang berbeda, software dan arsitektur yang tidak sama. Sehingga sebelum proses ini kita lakukan, sebaiknya perlu kita definisikan *requirement* terhadap sumber data yang akan kita butuhkan untuk lebih memudahkan pada *extraction*data ini.

* **Transformation**

Pada kenyataannya, pada proses transaksional data disimpan dalam berbagai format sehingga jarang kita temui data yang konsisten antara aplikasi-aplikasi yang ada. Transformasi data ditujukan untuk mengatasi masalah ini. Dengan proses transformasi data ini, kita melakukan standarisasi terhadap data pada satu format yang konsisten. Beberapa contoh ketidakkonsistenan data tersenut dapat diakibatkan oleh tipe data yang berbeda, data length dan lain sebagainya

* **Loading**

*Data loading* adalah memindahkan data ke datawarehouse. Ada dua *loading* data yang di lakukan pada datawarehouse. Pertama adalah *inisial load*, proses ini dilakukan pada saat telah selesai mendesign dan membangun data warhouse. Data yang masukkan tentunya akan sangat besar dan memakan waktu yang relatif lebih lama. Kedua *Incremental load*, dilakukan ketika data warehouse telah dioperasikan. Sehingga akan lebih mudah melakukan data extraction, transformation dan loading terhadap data tersebut.

**ELT (Extract Loading Transformation)**

ELT (Extract Loading Transformation) merupakan variasi dari ETL (Extract Transformation Loading). Bedanya dengan ETL adalah pada bagian keduanya, dimana pada ETL setelah data di extract lamu di transformasikan dan data di pindahkan ke data warehouse, sedangkan untuk ELT, pertama-tama data di extract lalu data mentah di muat secara langsung pada data warehouse dan akan di transformasi pada data warehouse. Ini berperan dalam melakukan analisa data yang sangat besar, dan lebih cepat dibandingkan dengan ETL.

**OLAP**

OLAP (Online Analytical Proccessing) merupakan suatu proses yang digunakan untuk melakukan permintaan terhadap data dalam bentuk yang kompleks dan menganalisa data yang bervolume besar. OLAP merupakan teknologi yang memproses data di dalam databasedalam struktur multidimensi, menyediakan jawaban yang cepat untuk query dan analisis yang kompleks. Data multidimensi adalah data yang dapat dimodelkan sebagai atribut dimensi dan atribut ukuran. Contoh atribut dimensi adalah nama barang dan warna barang, sedangkan contoh atribut ukuran adalah jumlah barang.

Dengan adanya analisis dari OLAP, maka pengguna akan dengan mudah dapat menyimpulkan data secara terstruktur. Misalnya di sebuah toko X menjual berbagai macam barang, dengan adanya OLAP maka pemilik toko dapat mengetahui barang mana yang paling banyak diminati pelanggan dan barang mana yang tidak laku/tidak terjual. Dengan adanya analisa seperti itu, pemilik toko akan dengan mudah mengeluarkan barang mana yang cepat terjual untuk di produksi lebih banyak lagi.

A **staging area**, or **landing zone**, is an intermediate storage area used for data processing during the [extract, transform and load (ETL)](https://en.wikipedia.org/wiki/Extract,_transform,_load) process. The data staging area sits between the data source(s) and the data target(s), which are often [data warehouses](https://en.wikipedia.org/wiki/Data_warehouse), [data marts](https://en.wikipedia.org/wiki/Data_mart), or other data repositories.[[1]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Oracle_Ref-1)

Area pementasan, atau zona pendaratan, adalah area penyimpanan menengah yang digunakan untuk pemrosesan data selama proses ekstrak, transformasi dan pemuatan (ETL). Area pementasan data berada di antara sumber data dan target data, yang Seringkali gudang data, mart data, atau repositori data lainnya. [1]

Data staging areas are often transient in nature, with their contents being erased prior to running an ETL process or immediately following successful completion of an ETL process. There are staging area architectures, however, which are designed to hold data for extended periods of time for archival or troubleshooting purposes.

Area pementasan data seringkali bersifat sementara, dengan isinya dihapus sebelum menjalankan proses ETL atau segera setelah berhasil menyelesaikan proses ETL. Namun demikian, ada arsitektur area pementasan, yang dirancang untuk menyimpan data untuk jangka waktu yang lama untuk tujuan pengarsipan atau pemecahan masalah.

Staging areas can be designed to provide many benefits, but the primary motivations for their use are to increase efficiency of ETL processes, ensure data integrity and support data quality operations. The functions of the staging area include the following:

**Consolidation**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=3)]

One of the primary functions performed by a staging area is consolidation of data from multiple source systems.[[2]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Data_Warehousing_Fundamentals_Ref-2) In performing this function the staging area acts as a large "bucket" in which data from multiple source systems can be temporarily placed for further processing. It is common to tag data in the staging area with additional metadata indicating the source of origin and timestamps indicating when the data was placed in the staging area.

**Alignment**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=4)]

Aligning data includes standardization of [reference data](https://en.wikipedia.org/wiki/Reference_data) across multiple source systems and validation of relationships between records and data elements from different sources.[[2]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Data_Warehousing_Fundamentals_Ref-2) Data alignment in the staging area is a function closely related to, and acting in support of, [master data management](https://en.wikipedia.org/wiki/Master_data_management) capabilities.[[5]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Master_Data_Management_Ref-5)

**Minimizing contention**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=5)]

The staging area and ETL processes it supports are often designed with a goal of minimizing contention within source systems. Copying required data from source systems to the staging area in one shot is often more efficient than retrieving individual records (or small sets of records) on a one-off basis. The former method takes advantage of technical efficiencies, such as data streaming technologies, reduced overhead through minimizing the need to break and re-establish connections to source systems and optimization of concurrency lock management on multi-user source systems. By copying the source data from the source systems and waiting to perform intensive processing and transformation in the staging area, the ETL process exercises a great degree of control over [concurrency issues](https://en.wikipedia.org/wiki/Concurrency_control) during processing.

**Independent scheduling/multiple targets**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=6)]

The staging area can support hosting of data to be processed on independent schedules, and data that is meant to be directed to multiple targets.[[2]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Data_Warehousing_Fundamentals_Ref-2) In some instances data might be pulled into the staging area at different times to be held and processed all at once. This situation might occur when enterprise processing is done across multiple time zones each night, for instance. In other cases data might be brought into the staging area to be processed at different times; or the staging area may be used to push data to multiple target systems. As an example, daily operational data might be pushed to an [operational data store (ODS)](https://en.wikipedia.org/wiki/Operational_data_store) while the same data may be sent in a monthly aggregated form to a data warehouse.

**Change detection**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=7)]

The staging area supports efficient change detection operations against target systems. This functionality is particularly useful when the source systems do not support reliable forms of change detection, such as system-enforced timestamping, change tracking or [change data capture (CDC)](https://en.wikipedia.org/wiki/Change_data_capture).

**Cleansing data**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=8)]

[Data cleansing](https://en.wikipedia.org/wiki/Data_cleansing) includes identification and removal (or update) of invalid data from the source systems. The ETL process utilizing the staging area can be used to implement business logic to identify and handle "invalid" data. Invalid data is often defined through a combination of business rules and technical limitations. Technical constraints may additionally be placed on staging area structures (such as table constraints in a relational database) to enforce data validity rules.[[2]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-Data_Warehousing_Fundamentals_Ref-2)

**Aggregate precalculation**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=9)]

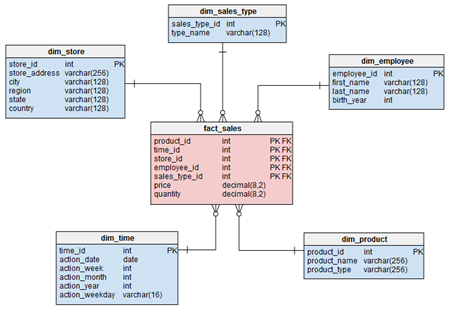
Precalculation of aggregates, complex calculations and application of complex business logic may be done in a staging area to support highly responsive service level agreements (SLAs) for summary reporting in target systems.[[3]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-TDWI_Ref-3)

**Data archiving and troubleshooting**[[edit](https://en.wikipedia.org/w/index.php?title=Staging_(data)&action=edit&section=10)]

Data archiving can be performed in, or supported by, a staging area. In this scenario the staging area can be used to maintain historical records during the load process, or it can be used to push data into a target archive structure. Additionally data may be maintained within the staging area for extended periods of time to support technical troubleshooting of the ETL process....[[3]](https://en.wikipedia.org/wiki/Staging_(data)#cite_note-TDWI_Ref-3)

Tempat penyimpanan data sementara, untuk melakukan proses Clean, Transform, Combine, Deduplicate, Household, archive, dan menyiapkan sumber data untuk digunakan Data Warehouse.

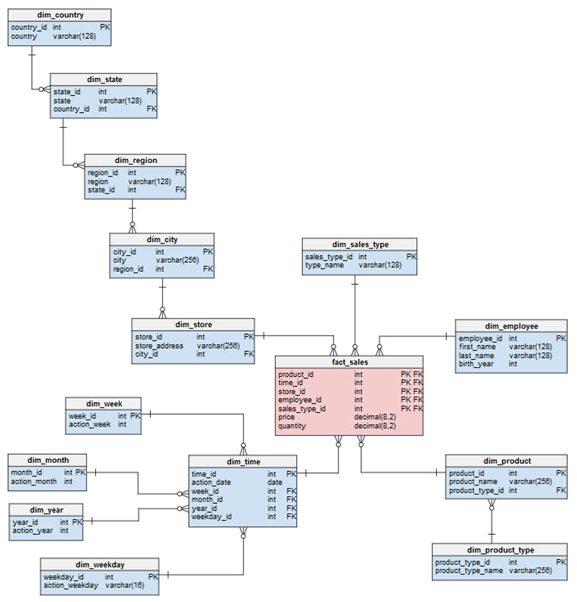
**Skema Bintang**

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

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

**SkemaSnowflake**

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

Gambar 2. Skema Snowflake

Skema snowflakejuga 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.

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:

1. Kelebihan Star Schema

a. Efisiensi, struktur database konsisten sehingga efisien dalam mengakses datadengan menggunakan tool untuk menampilkan data termasuk laporan tertulis dan quer.

b. Kemampuan untuk mengatasi perubahan kebutuhan, skema bintang dapatberadaptasi terhadap perubahan kebutuhan pengguna karena semua tabeldimensi memiliki kesamaan dalam hal menyediakan akses ke tabel fakta.

c. Extensibilit, model dimensional dapat dikembangkan. Seperti menambah tabelfakta selama data masih konsisten, menambah tabel dimensi selama ada nilaitunggal di table dimensi tersebut yang mendefinisikan setiap record tabel fakta yang ada, menambahkan attribute tabel imensi, dan memecah record tabeldimensi yang ada menjadi level yang lebih rendah daripada level sebelumnya.

d. Kemampuan untuk menggambarkan situasi bisnis pada umumnya, pendekatan standar untuk menangani situasi umum di dunia bisnis yang terus bertambah.

e. Proses query yang bisa diprediksi, aplikasi data warehouse yang mencari datadari level yang di bawahnya akan mudah menambahkan jumlah attribute padatabel dimensi dari sebuah skema bintang. Aplikasi yang mencari data dari levelyang setara akan menghubungkan tabel fakta yang terpisah melalui tabeldimensi yang dapat diakses bersama.

2. Kelebihan Snowflake Schema

a. Ukuran penyimpanan kecil didalam tempat penyimpanan.

b. Struktur yang normal lebih mudah untuk di-update dan di-maintenance.

3. Kelebihan Starflake Schema

a. Efisien dalam hal mengakses data.

b. Dapat beradaptasi terhadap kebutuhan-kebutuhan.

4. Kekurangan Star Schema

a. Ukuran penyimpanan relatif lebih besar. Karena ada data yang berulang sehingga disk space yang digunakan lebih banyak.

b. Maintenance dan update lebih sulit. Karena tabel yang tidak normal.

5. Kekurangan Snowflake Schema

a. Kurang jelas dan penggunaan akhir terhambat oleh kompleksitas.

b. Sulit untuk mencari isi, karena kompleks.

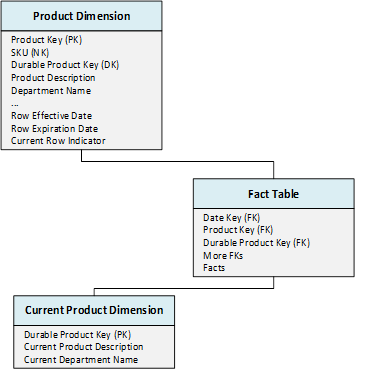
c. Performa query menurun karena adanya join table antar dimensi.

# 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)
* 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 |

Terdapat 8 tipe SCD :

Tipe 0

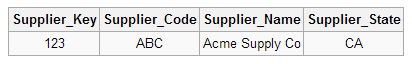
Tipe perubahan yang pasif. Metode ini akan terus menyimpan value data saat pertama kali insert. Tipe 0 biasa digunakan untuk data yang bersifat historikal dimana hanya perlu sedikit kontrol atau tidak sama sekali. Tipe jenis 1, 2, dan 3 lebih sering digunakan.

Tipe 1

Metode menggantikan data lama menjadi data baru. Metode ini tidak menyimpan historikal perubahan datanya.

Misal : Perubahan karena kesalahan pengetikan nama customer, Perubahan karena perpindahan alamat customer  dsb.

Contoh :

[](https://henryeka.files.wordpress.com/2013/02/1.png)

Dalam kasus ini sebuah supplier ingin melakukan perubahan lokasi pabrik dari CS (California) ke IL (Illinois).

Berubah menjadi :

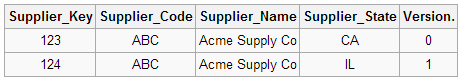
[2](https://henryeka.files.wordpress.com/2013/02/2.png)

Dapat kita perhatikan bawah tidak terjadi perubahan data apapun baik itu untuk natural key (Suplier\_code) maupun surrogate key (Suplier\_key). Sehingga untuk teknik ini sebenarnya surrogate key tidak essensial karena fungsinya dapat digantikan oleh natural key. Namun surrogate key juga diperlukan karena sistem akan lebih cepat melakukan mapping menggunakan integer dibanding dengan string.

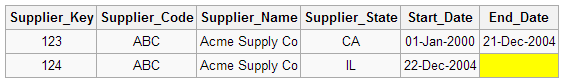
Tipe 2

Tipe ini dapat menyimpan histori perubahan data dengan menambahkan row baru dengan natural key yang sama namun surrogate key yang berbeda.

Contoh 1. Menggunakan field version yang akan bertambah secara urut:

[](https://henryeka.files.wordpress.com/2013/02/3.png)

Contoh 2. Menggunakan Start Date dan End Date :

[](https://henryeka.files.wordpress.com/2013/02/4.png)

Kolom Null berwarna kuning menandakan bawah data tersebut adalah data current yang dipakai. Tipe ini tidak cocok diterapkan untuk data yang berubah dengan sangat cepat karena akan menjadikan database tersebut sangat besar dan membebani kinerja proses.

Tipe 3

Tipe ini menyajikan data historikal dalam field baru dan kemampuan track historical yang terbatas.  Didalam contoh dapat kita lihat bawah perubahan lokasi supplier dimantain dalam suatu field baru yaitu current\_suplier\_state sedangkan data historinya berada di original\_suplier\_state.

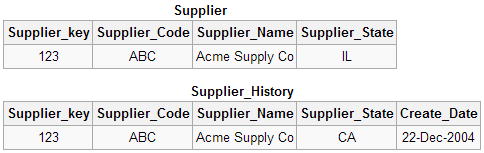
Contoh :

[5](https://henryeka.files.wordpress.com/2013/02/5.png)

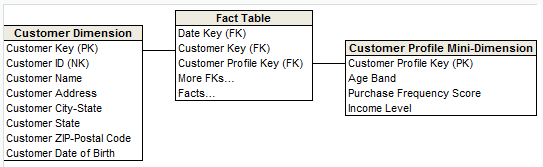
Tipe 4 / Mini Dimension

Tipe ini menyajikana data historikal dengan membentuk table baru. Seperti yang dapat kita lihat di dalam contoh bahwa data current disimpan didalam table supplier sedangkan data historikalnya disimpan dalam table supplier\_history.  Tipe ini juga disebut mini dimension. Cocok digunakan untuk data dengan perubahan yang cepat.

Contoh :

[](https://henryeka.files.wordpress.com/2013/02/6.png)

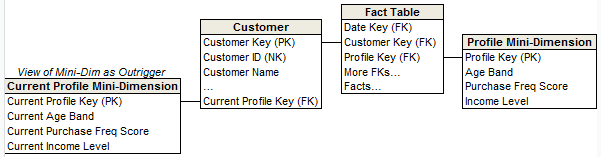
Contoh Arsitektur customer profile mini dimension :

[](https://henryeka.files.wordpress.com/2013/02/7.png)

Tipe 5

Tipe ini adalah gabungan antara tipe 4 dan tipe 1.  Intinya adalah penambahan table bertipe 1 gunanya adalah jika user ingin mengetahui data current dan data historikal secara bersamaan. Penambahan table bertipe satu tidak harus terhubung dengan fact.

Contoh Arsitektur :

[](https://henryeka.files.wordpress.com/2013/02/8.png)

Tipe 6 / Hybrid

Tipe ini adalah gabungan tipe 1, 2 dan 3 ( 1+2+3 = 6 ). Surrogate key belum diletakan di table fact.

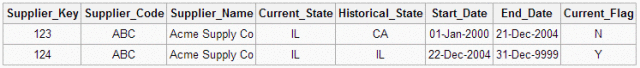
Contoh :

Data awal :

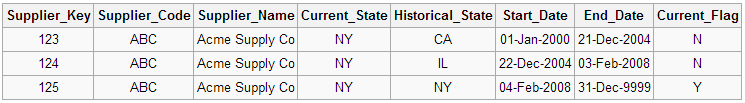
[9](https://henryeka.files.wordpress.com/2013/02/9.png)

Ketika perusahaan pindah ke illisois maka terjadi perubahan :

1. Perubahan Current\_state menjadi data baru yaitu IL (tipe 1).
2. Terbentuknya satu row baru dengan supplier key yang baru (tipe 2).
3. Lalu data state lama akan disimpan kedalam Historical\_state row lama (tipe 3)

[](https://henryeka.files.wordpress.com/2013/02/10.png)

Contoh ketika perusahaan tersebut pindah ke lokasi baru :

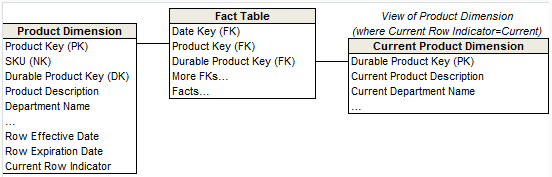
[](https://henryeka.files.wordpress.com/2013/02/11.png)

NB : Current\_flag menandakan bahwa data tersebut adalah data current.

Tipe 7

Tipe ini adalah pengembangan dari tipe 6 sebelumnya hanya saja surrogate key dan natural key semuanya diletakan pada fact table.

Contoh arsitekturnya:

[](https://henryeka.files.wordpress.com/2013/02/12.png)

A surrogate key on a table is a column with a unique identifier for each row. The key is not generated from the table data. Data modelers like to create surrogate keys on their tables when they design data warehouse models. You can use the IDENTITY property to achieve this goal simply and effectively without affecting load performance.

## **Creating a table with an IDENTITY column**

The IDENTITY property is designed to scale out across all the distributions in the data warehouse without affecting load performance. Therefore, the implementation of IDENTITY is oriented toward achieving these goals.

You can define a table as having the IDENTITY property when you first create the table by using syntax that is similar to the following statement:

CREATE TABLE dbo.T1

( C1 INT IDENTITY(1,1) NOT NULL

, C2 INT NULL

)

WITH

( DISTRIBUTION = HASH(C2)

, CLUSTERED COLUMNSTORE INDEX

)

;

You can then use INSERT..SELECT to populate the table.

This remainder of this section highlights the nuances of the implementation to help you understand them more fully.

### Allocation of values

The IDENTITY property doesn't guarantee the order in which the surrogate values are allocated, which reflects the behavior of SQL Server and Azure SQL Database. However, in Azure SQL Data Warehouse, the absence of a guarantee is more pronounced.

The following example is an illustration:

CREATE TABLE dbo.T1

( C1 INT IDENTITY(1,1) NOT NULL

, C2 VARCHAR(30) NULL

)

WITH

( DISTRIBUTION = HASH(C2)

, CLUSTERED COLUMNSTORE INDEX

)

;

INSERT INTO dbo.T1

VALUES (NULL);

INSERT INTO dbo.T1

VALUES (NULL);

SELECT \*

FROM dbo.T1;

DBCC PDW\_SHOWSPACEUSED('dbo.T1');

In the preceding example, two rows landed in distribution 1. The first row has the surrogate value of 1 in column C1, and the second row has the surrogate value of 61. Both of these values were generated by the IDENTITY property. However, the allocation of the values is not contiguous. This behavior is by design.

 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.

Granularity adalah tingkat kedetailan data dalam suatu data warehouse. Semakin detail data, maka tingkat granularity-nya akan semakin rendah juga. Jadi Level Low / yang paling terendah  adalah ketika tingkat kedetailan yang tinggi,misalnya pada data transaksi.

Titik awal untuk menentukan tingkat yang tepat dari granularity adalah dengan melakukan perkiraan baku jumlah baris data dan DASD (direct access storage device) atau akses langsung perangkat penyimpanan yang akan di masukkan dalam data warehouse.

Jalur algoritma untuk menghitung ruang yang ditempati oleh data warehouse adalah :

– Identifikasi semua tabel yang akan dibangun

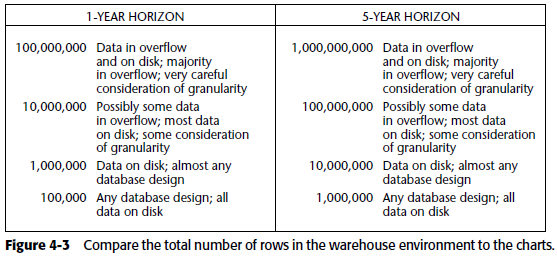
– Perkirakan ukuran baris di setiap tabel

– Pada cakrawala satu tahun, memperkirakan jumlah maksimum dan minimum baris dalam tabel

– Hitung proyeksi ruang data indeks

1. **Data Overflow**

Setelah perkiraan baku untuk ukuran data warehouse dibuat, langkah selanjutnya adalah membandingkan jumlah baris dalam lingkungan gudang ke grafik.

[](http://sis.binus.ac.id/files/2016/12/granularity1.png)

Teorinya adalah bahwa setelah lima tahun, faktor-faktor ini akan di tempat:

– Akan ada lebih banyak keahlian yang tersedia dalam mengelola volume data pada data warehouse.

– Biaya Hardware (perangkat keras) akan turun sampai batas tertentu.

– Software (perangkat lunak) yang lebih kuat akan tersedia.

– End user atau pengguna akhir akan lebih canggih.

1. **Tingkat Granularity**

Setelah analisis sederhana berapa banyak data yang akan ada di gudang data dilakukan (dan, sebenarnya, banyak perusahaan menemukan bahwa mereka perlu menempatkan setidaknya beberapa data ke dalam penyimpanan overflow), langkah berikutnya adalah untuk menentukan tingkat granularity untuk data yang berada pada disk penyimpanan.

Satu-satunya cara nyata untuk menentukan tingkat yang tepat dari granularity untuk data ringan diringkas adalah untuk menempatkan data di depan end user atau pengguna akhir. Pertimbangan kedua dalam menentukan tingkat granularity untuk mengantisipasi kebutuhan entitas arsitektur yang berbeda yang akan di isi berdasarkan dari data warehouse.

1. **Feedback loop techniques (teknik umpan balik)**

            Berikut ini adalah teknik untuk membuat feedback loop techniques atau teknik umpan balik yang harmonis:

– Membangun bagian pertama dari data warehouse di sangat kecil, langkah sangat cepat, dan hati-hati mendengarkan komentar pengguna akhir pada akhir setiap langkah pembangunan.

– Jika tersedia, gunakan prototyping dan memungkinkan umpan balik untuk fungsi menggunakan pengamatan yang dikumpulkan dari prototipe.

– Lihatlah bagaimana orang lain telah membangun tingkat granularity mereka dan belajar dari pengalaman mereka.

– Pergi melalui proses umpan balik dengan pengguna berpengalaman yang menyadari proses yang terjadi.

– Lihatlah apa saja organisasi yang kini muncul dan terlihat berjalan, dan menggunakan persyaratan fungsional tersebut sebagai panduan.

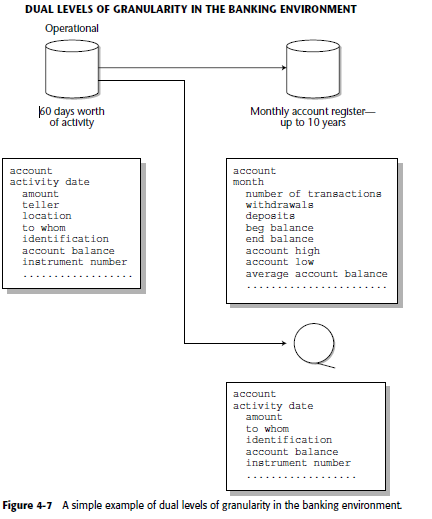
– Jalankan sesi Joint Application Design (JAD) atau aplikasi desain bersama dan mensimulasikan output untuk mencapai umpan balik yang diinginkan.

1. **Tabel fakta Granularity**

            Langkah pertama dalam merancang sebuah tabel fakta adalah untuk menentukan granularity dari tabel fakta. Dengan granularitas, berarti tingkat terendah informasi yang akan disimpan dalam tabel fakta. Ini merupakan dua langkah:

1. Tentukan dimensi mana yang akan dimasukkan
2. Tentukan sepanjang apa hirarki masing-masing dimensi informasi akan disimpan.

Faktor-faktor yang menentukan biasanya kembali kepada persyaratannya.

[](http://sis.binus.ac.id/files/2016/12/granularity2.png)

## CONTOH KASUS, DAN CARA PEMBUATAN STAR SCHEMA

Sebahai contoh kasus, kita coba membuat suatu star schema dari data sbb,

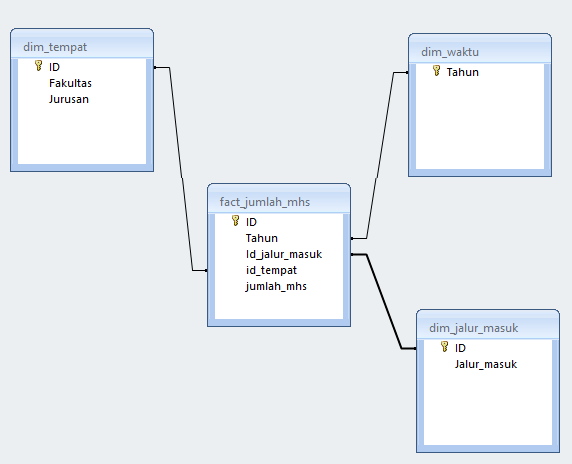
Data tersebut merupakan data fiktif

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tahun | Fakultas | | Jurusan | | Jalur Masuk | | | |
| Kode | Nama | Kode | Nama | Smart U | Gel. I | Gel.II | Gel.III |
| 2014 | 1 | Teknik & Ilmu Komputer | 01 | Teknik Informatika | 50 | 150 | 360 | 267 |
|  | 1 | Teknik & Ilmu Komputer | 05 | Sistem Informasi | 70 | 200 | 100 | 100 |
|  | 2 | Ekonomi | 01 | Akuntansi | 65 | 200 | 120 | 201 |
|  | 6 | Sastra | 01 | Sastra Inggris | 90 | 109 | 211 | 209 |
|  | 6 | Sastra | 02 | Sastra Jepang | 60 | 190 | 220 | 100 |

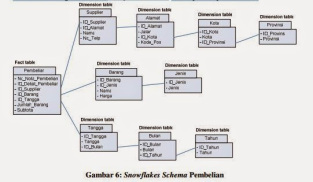
Jika dilihat dari data di atas, maka hanya ada satu measure, yaitu jumlah mahasiswa yang bisa di lihat dari tabel jalur masuk. Dan terdapat paling tidak 3 dimensi.

1. DImensi waktu (Tahun mahasiswa masuk)
2. Dimensi tempat (Fakultas, dan Jurusan)
3. Dimensi jenis jalur masuk (Smart U, Gel I, Gel II, Gel III)

Dengan dimensi dan measure di atas, maka star schema yang dibuat menjadi sbb



# SNOW FLAKE SCHEMA

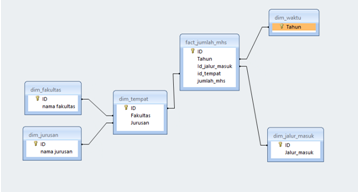


Menurut Connolly dan Begg (2010:1229), Snowflake adalah jenis dari star schema dimana tabel dimensinya tidak mengandung denormalisasi.

Sehingga dapat disimpulkan bahwa Snowflake Schema adalah suatu logical arrangement dari tabel di multidimensional database, sama seperti pada star schema, snowflake schema juga mempunyai table fact dan table dimensi, hanya saja pada snowflake schema, satu table fact terhubung dengan sati table dimensi utama, dan tabel dimensi lainya terhubung dengan tabel dimensi utama.

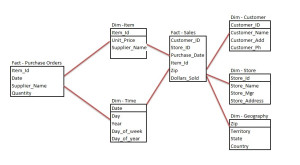
## CONTOH KASUS, DAN CARA PEMBUATAN SNOW FLAKE SCHEMA

Dari star schema di atas, maka bias dibuat snowflake schema pada gambar dibawah ini



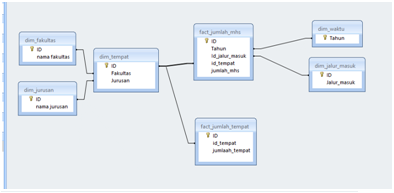
# FACT CONSTELLATION

Pada skema ini terdapat beberapa tabel fakta yang menggunakan satu atau  beberapa tabel dimensi secara bersama-sama sehingga jika digambarkan akan terlihat seperti sekumpulan bintang. Skema ini juga dikenal dengan galaxy  schema.



## CONTOH KASUS PEMBUATAN SKEMA FACT CONSTELLATION

Dari snowflake schema di atas, maka bias dibuat fact constellation schema pada gambar dibawah ini



conformed dimension

Dalam data warehousing, dimensi yang sesuai adalah dimensi yang memiliki arti yang sama untuk setiap fakta yang berhubungan. Dimensi yang sesuai memungkinkan fakta dan langkah-langkah dikategorikan dan dijelaskan dengan cara yang sama di beberapa fakta dan / atau data mart, memastikan pelaporan yang konsisten di seluruh perusahaan.