# Data intelligens, DWH-systemer, Star-, snowflake-datamodeller

## Indhold

https://videos.ida.dk/media/Introduktion%2Btil%2BMicrosoft%2BPower%2BBI/1_jzewlzau?utm_source=netvaerk&utm_medium=email&utm_campaign=FTK_IDAMatematik_20240214 video-intro
Dashboard;Load Web-Data; ModelView; Shape,combine data fra flere tabeller
Model-arbejde; Measures=reatime-formler

Jeg har gennemgået flere web-tutorials med min installation af Microsofts PowerBI på pc'en.
I filen *Revenue Opportunities, en selvstændig analyse med PowerBI.pdf* viser jeg en lille analyse med PowerBI af salgsdata for en lille fiktiv IT-virksomhed, her har jeg få lavet simpel analyse, set at modellen der anvendes er baserset på velkendte starmodel.
I de andre filer i mappen har jeg generet store datasæt og undersøgt performance for forskellige typiske requests, med forskellige optimeringsmuligheder.

# Oracle Data Warehouse Aggregation, Roll-Ups and Star Schemas

***Oracle Data Warehouse Tips by Burleson Consulting***

## *Aggregation, Roll-Ups And Star Schemas*

We have now defined and populated a star schema that contains the total_sales for each order for each day. While it is now easy to see the total for each order, rarely do the users of a decision support require this level of detail. Most managers would be more interested in knowing the sum of sales or units sold--aggregated by month, quarter, region, and so on. Even with a star schema, these types of aggregation would be hard to compute at runtime with an acceptable response time. Essentially, we can either aggregate at runtime or pre-aggregate the data offline, making the totals available without real-time computation. One simple alternative to real-time aggregation is to write SQL to pre-aggregate the data according to the dimensions that the end-user may

want to see. In our example, let?s assume that management would want to aggregate monthly sales by region, state, item type, and salesperson. Since we have four possible dimensions, we can generate a list of the following six aggregate tables to precreate. Assume that all of the tables would have a month_year field as their primary key:

* region by state--This table would have region_name, state_name and monthly_sales as columns.

* Region by item_type--This table would have region_name, item_type and monthly_ sales as columns.

* Region by salesperson--This table would have region_name, salesperson_name and monthly_sales as columns.

* State by item_type--This table would have state_name, item_type and monthly_sales as columns.

* sState by salesperson--This table would have state_name, salesperson_name and monthly_sales as columns.

* item_type by salesperson--This table would have item_type, salesperson_name and monthly_sales as columns.

The SQL to produce these table can be easily run as a batch task at end-of-month processing. For example, the SQL to create the region_by_state table might look like this:

```
INSERT INTO region_item_type
VALUES
(SELECT ?3?, ?1996?, region_name, item_type, total_cost)
FROM fact_table_3_96
GROUP BY region_name, item_type
);

? erstatter en char
```

The sample region_item_type table might look like the one shown in Table 10.3.

Table 10.3 A sample region_item_type table, month=3, 96

| DATE | REGION | TYPE | MONTHLY_SALES |
|---|---|---|---|
| 3/96 | WEST | Clothes | $113,999 |
| 3/96 | WEST | Hardware | $ 56, 335 |
| 3/96 | WEST | Food | $ 23, 574 |
| 3/96 | EAST | Clothes | $ 45, 234 |
| 3/96 | EAST | Hardware | $ 66, 182 |

| | | | |
|---|---|---|---|
| 3/96 | EAST | Food | $835,342 |
| 3/96 | SOUTH | Clothes | $ 1, 223 |
| 3/96 | SOUTH | Hardware | $ 56, 392 |
| 3/96 | SOUTH | Food | $ 9, 281 |
| 3/96 | NORTH | Clothes | $826,463 |
| 3/96 | NORTH | Hardware | $ 77, 261 |
| 3/96 | NORTH | Food | $ 43, 383 |

These aggregate tables can be built in the middle of the night if need be--right after the master fact tables have been populated with the day?s sales. The next morning, the prior days sales will have been rolled-up into these summaries, giving management an accurate, fast and easy to use tool for decision support.

Of course, these tables are two dimensional, but they can easily be massaged by an application to provide a tabular representation of the variables. Table 10.4 presents this tabular form.

Table 10.4 REGION vs. TYPE.

| | Clothes | Food | Hardware |
|---|---|---|---|
| WEST | $113,999 | $ 23, 574 | $56,335 |
| EAST | $ 45,234 | $835,342 | $66,182 |
| NORTH | $826,463 | $ 43, 383 | $77,261 |
| SOUTH | $ 1,223 | $ 9,281 | $56,392 |

Note: This technique replicates the functionality of a multi-dimensional database where an end-user can specify the axis of interest and the multi-dimensional database (MDDB) will build a tabular representation of the data.

But what if management wants to look at quarterly summaries instead of monthly summaries? What about yearly summaries? Of course, this same technique can be used to roll up the monthly summary tables into quarterly summaries, yearly summaries, and so on, according to the demands of the end-user.

**History Of OLAP**

Dr. E. F. (Ted) Codd first coined the term OLAP in a 1993 report that was sponsored by Arbor software. In addition to coining the term, Codd also went on to create 12 rules for OLAP. Despite the claims of a new technology, some offerings such as IRI date to the early 1970s.

Note: The Internet offers a popular forum that discusses OLAP issues. It is called comp.database.olap.

**Simulation Of Cubic Databases (Dimensionality)**

For an illustrative example, consider the sample customer table in Figure 10.5. Now, let's take this data structure and assume that this tables is physically stored in data order. We can imagine how this data might look as a cubic table by reviewing Figure 10.8.

Table 10.5 A sample customer table.

| CUSTOMER-NAME | # Sales | YY-MM | City | State |
|---|---|---|---|---|
| Bob Papaj & Assoc. | 300 | 91-01 | NY | NY |
| Mark Reulbach Inc. | 400 | 91-01 | San Francisco | CA |
| Rick Willoughby Co. | 120 | 91-02 | NY | NY |
| Kelvin Connor Co. | 300 | 91-02 | San Francisco | CA |
| Jame Gaston Inc. | 145 | 91-03 | NY | NY |
| Linda O?Dell Assoc. | 337 | 91-03 | Fairport | NY |
| Rick Wahl & Assoc. | 134 | 91-03 | San Francisco | CA |

Figure 10.8 Cubic representation of relational data.

Of course, the cubic representation would require that the data be loaded into a multi-dimensional database or a spreadsheet that supported pivot tables. When considering an MDDB, two arguments emerge. The relational database vendors point out that MDDBs are proprietary--they feel that the more open relational databases should be used. The MDDB vendors point out some serious inadequacies with SQL that make it very difficult to use a relational database.

Keep in mind that dimensions may be hierarchical in nature, adding further confusion. A time dimension, for example, may be represented as a hierarchy with year, quarter, month, and day. Each of these 4 "levels" in the dimension hierarchy may have its own values. In other words, a cubic representation with time as a dimension may be viewed in two ways:

* A series of cubes--one for year, another for quarter, and another with full_date.

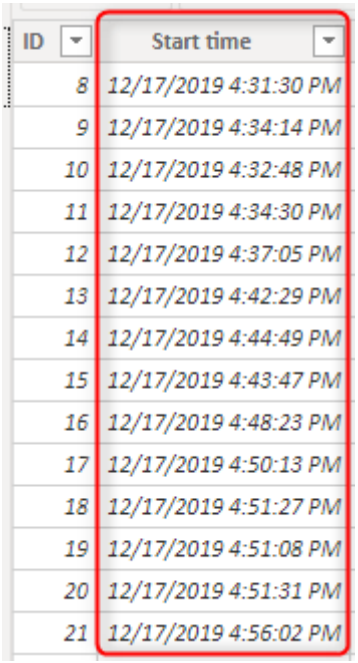
* A five-dimension table.

MDDBs are most commonly used with data that is a natural fit for pivot tables, and it should come as no surprise that most MDDB sites are used with finance and marketing applications. Unfortunately, most multi-dimensional databases do not scale up well for warehouse applications. For example, the largest supported database for Essbase is about 20 gigabytes, whereas data warehouses with sizes measured in terabytes are not uncommon.

It is important to note that defining aggregation of a multi-dimensional database is no different than defining aggregate tables to a relational database. At load time, the database will still need to compute the aggregate values. MDDBs also employ the concept of sparse data. Since data is aggregated and presliced, some cells on a cube may not contain data. For example, consider a cube that tracks sales of items across a large company. The cells representing sales of thermal underwear would be null for Hawaii, while the sales of surfboards in Wyoming would also be null. Nearly all of the product offerings are able to maintain a mechanism for compressing out these types of null values.

# Time and Date Tables in a Data Model

The time and the date tables should not be related to each other, their relationship should be made in the fact table. in below you can see that the fact table's data in a full date and time column (not separated);

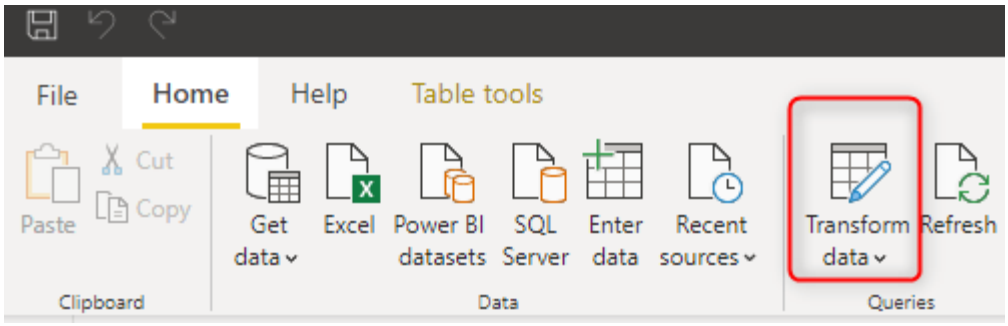


| ID | Start time |
|---|---|
| 8 | 12/17/2019 4:31:30 PM |
| 9 | 12/17/2019 4:34:14 PM |
| 10 | 12/17/2019 4:32:48 PM |
| 11 | 12/17/2019 4:34:30 PM |
| 12 | 12/17/2019 4:37:05 PM |
| 13 | 12/17/2019 4:42:29 PM |
| 14 | 12/17/2019 4:44:49 PM |
| 15 | 12/17/2019 4:43:47 PM |
| 16 | 12/17/2019 4:48:23 PM |
| 17 | 12/17/2019 4:50:13 PM |
| 18 | 12/17/2019 4:51:27 PM |
| 19 | 12/17/2019 4:51:08 PM |
| 20 | 12/17/2019 4:51:31 PM |
| 21 | 12/17/2019 4:56:02 PM |

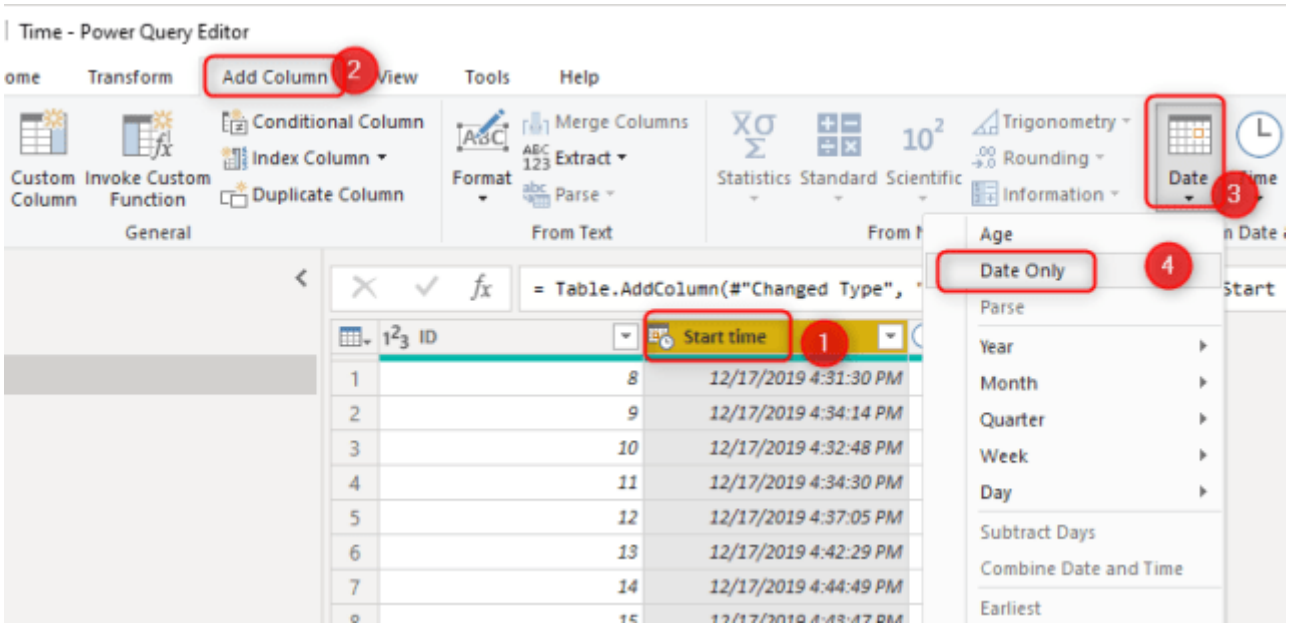
For connecting this table to the time and date tables, you have to separate date and time, which you can do that easier in the Power Query Editor.

# Separate Date and Time; 1 Date-dim,1  Time-dim-tabel i modellen

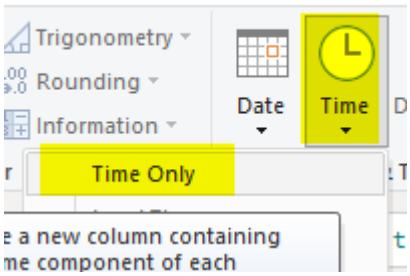
Go to transform data;




In the Power Query Editor, click on Add Column, and then Under Date, Select Date Only.



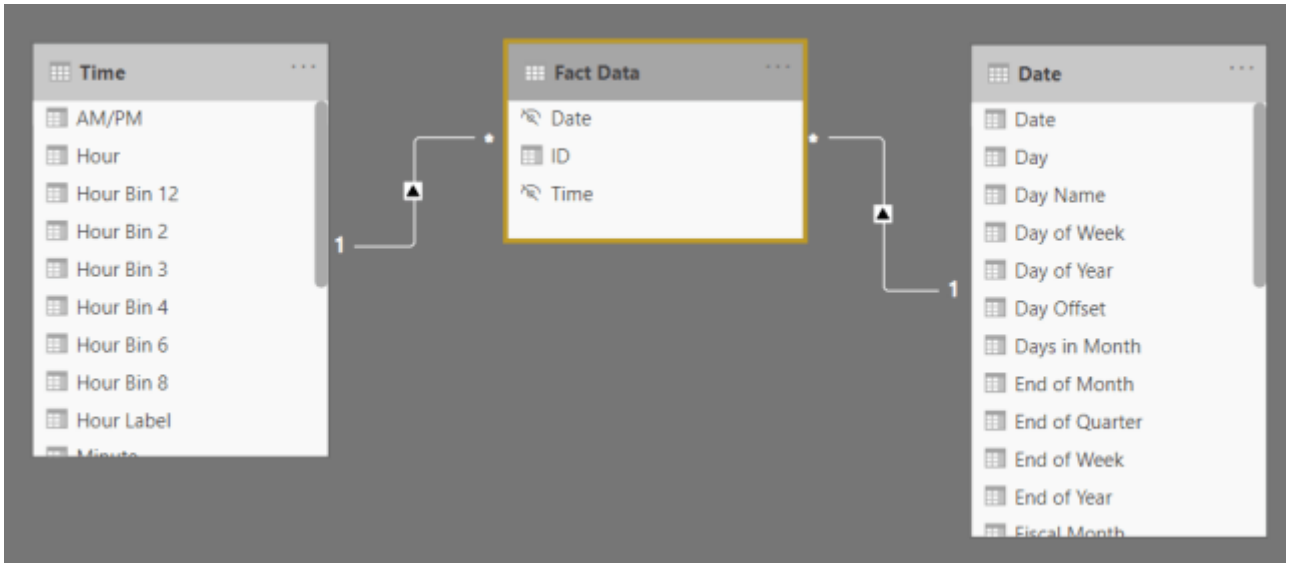

Do similar thing this time with Time, and Time only;




Now that you have the date and time columns separately, you can remove the main date and time column;

| ID | Time | Date |
|---|---|---|
| 8 | 4:31:30 PM | 12/17/2019 |
| 9 | 4:34:14 PM | 12/17/2019 |
| 10 | 4:32:48 PM | 12/17/2019 |
| 11 | 4:34:30 PM | 12/17/2019 |
| 12 | 4:37:05 PM | 12/17/2019 |
| 13 | 4:42:29 PM | 12/17/2019 |
| 14 | 4:44:49 PM | 12/17/2019 |
| 15 | 4:43:47 PM | 12/17/2019 |
| 16 | 4:48:23 PM | 12/17/2019 |
| 17 | 4:50:13 PM | 12/17/2019 |
| 18 | 4:51:27 PM | 12/17/2019 |
| 19 | 4:51:08 PM | 12/17/2019 |
| 20 | 4:51:31 PM | 12/17/2019 |
| 21 | 4:56:02 PM | 12/17/2019 |
| 22 | 4:58:28 PM | 12/17/2019 |

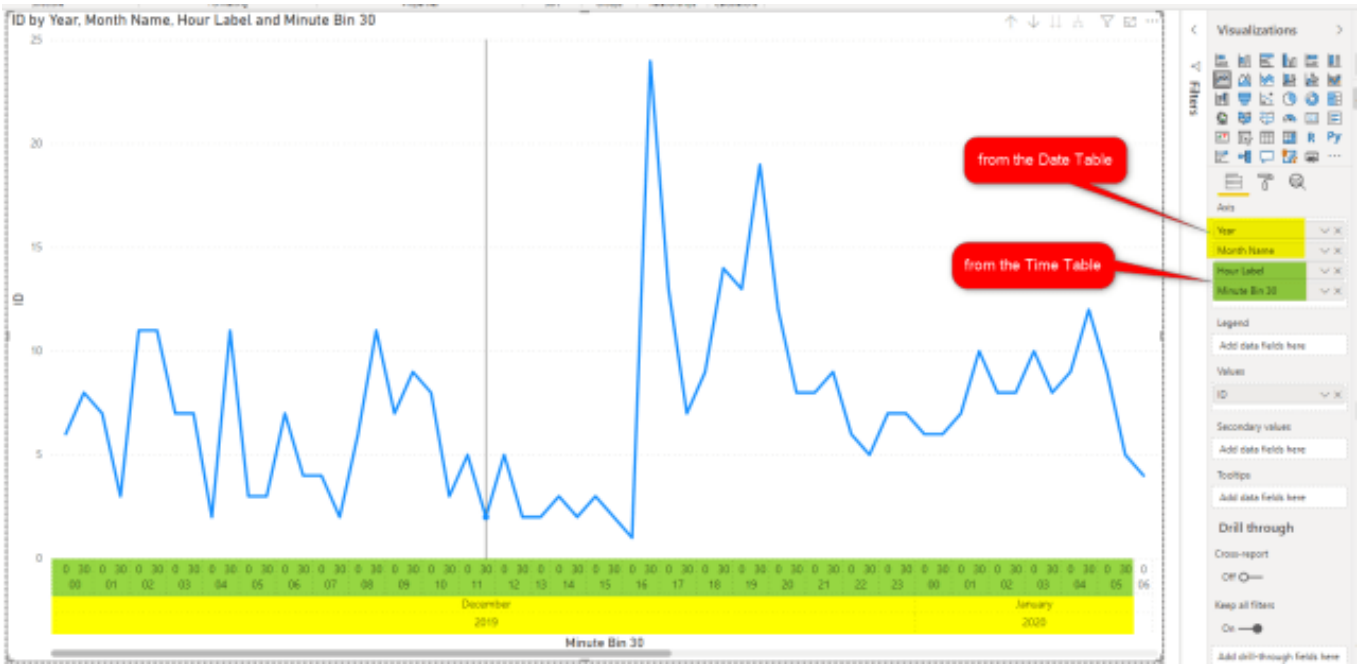
now you can close and apply the Power Query Editor window, and create the relationship between this table and the Date and Time dimensions;




## Having both date and time in one axis

Another question I got in the comments was that using this approach can we have the date and time in one axis? yes, you can. You just need to drag them into the axis one after each other, this forms a hierarchy which you can expand into.

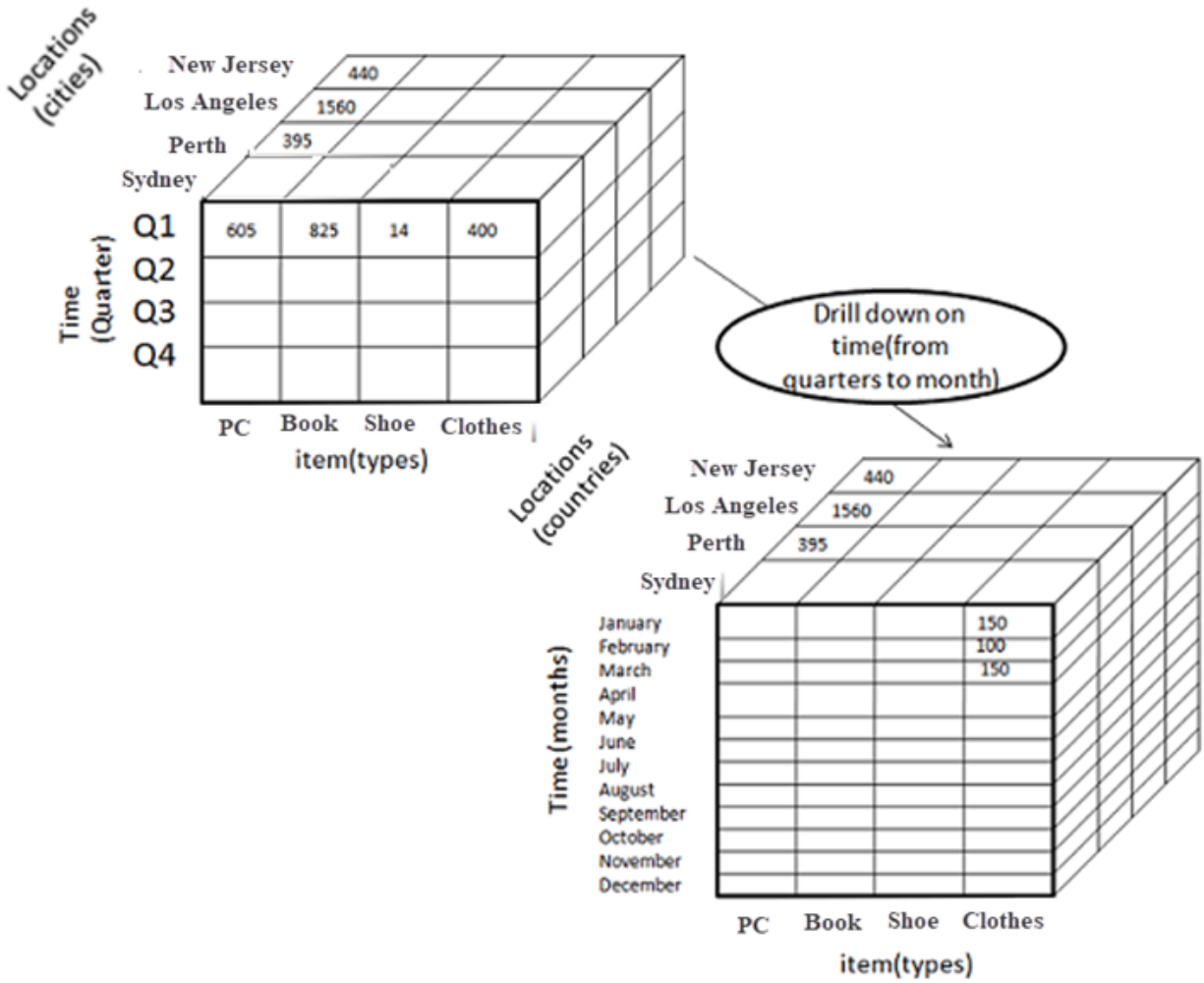
In the example below, I have both date and time from the model above demonstrated in a single axis;




I hope this post helps in your Power BI data model implementation. if you have any questions, please don't hesitate to contact me in the comments below.

# Roll up: aggregerer fx tid i kvartaler for fact; Drill down er den modsatte operation

Man kan gemme aggregeringerne i særskilte tabeller. Eksempel på data cube




med index for hver type dimension, time, item, location

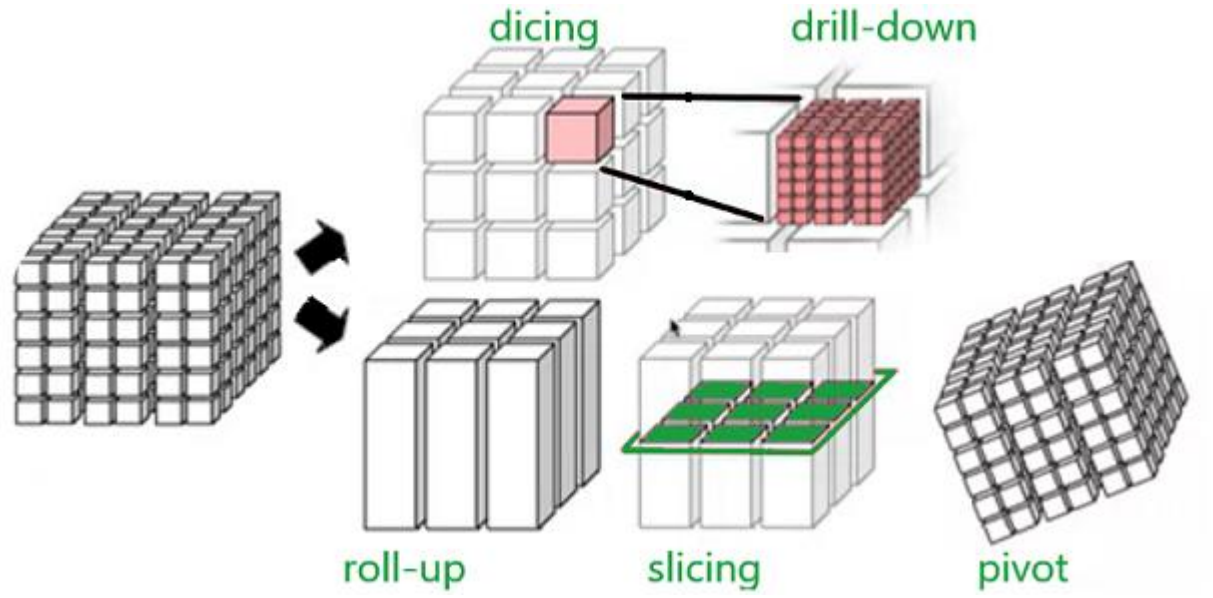
# Data cube classification:

The data cube can be classified into two categories:

**Multidimensional data cube:**
It basically helps in storing large amounts of data by making use of a multi-dimensional array. It increases its efficiency by keeping an index of each dimension. Thus, dimensional is able to retrieve data fast.

**Relational data cube, hver dimension i egen tabel, som i star-modellen**
It basically helps in storing large amounts of data by making use of relational tables**, star-model**. Each relational table displays the dimensions of the data cube. It is slower compared to a Multidimensional Data Cube.

Data cube operations:




**Slicing er filtrering,** kan fx være at udvælge bestemt tidsrum. Data cube operations are used to manipulate data to meet the needs of users. These operations help to select particular data for the analysis purpose. There are mainly 5 operations listed below-

- **Roll-up**: operation and aggregate certain similar data attributes having the same dimension together. For example, if the data cube displays the daily income of a customer, we can use a roll-up operation to find the monthly income of his salary.

- **Drill-down**: this operation is the reverse of the roll-up operation. It allows us to take particular information and then subdivide it further for coarser granularity analysis. It zooms into more detail. For example- if India is an attribute of a country column and we wish to see villages in India, then the drill-down operation splits India into states, districts, towns, cities, villages and then displays the required information.

**Slicing**: operation filters the unnecessary portions.

Suppose in a particular dimension, the user doesn't need everything for analysis, rather a particular attribute. For example, country="jamaica", this will display only about jamaica and only display other countries present on the country list.

- **Dicing**: this operation does a multidimensional cutting, that not only cuts only one dimension but also can go to another dimension and cut a certain range of it. As a result, it looks more like a subcube out of the whole cube(as depicted in the figure). For example- the user wants to see the annual salary of Jharkhand state employees.

- **Pivot**: this operation is very important from a viewing point of view. It basically transforms the data cube in terms of view. It doesn't change the data present in the data cube. For example, if the user is comparing year versus branch, using the pivot operation, the user can change the viewpoint and now compare branch versus item type.

Advantages of data cubes:

- **Multi-dimensional analysis**: Data cubes enable multi-dimensional analysis of business data, allowing users to view data from different perspectives and levels of detail.
- **Interactivity:** Data cubes provide interactive access to large amounts of data, allowing users to easily navigate and manipulate the data to support their analysis.
- **Speed and efficiency**: Data cubes are optimized for OLAP analysis, enabling fast and efficient querying and aggregation of data.
- **Data aggregation:** Data cubes support complex calculations and data aggregation, enabling users to quickly and easily summarize large amounts of data.
- **Improved decision-making:** Data cubes provide a clear and comprehensive view of business data, enabling improved decision-making and business intelligence.
- **Accessibility**: Data cubes can be accessed from a variety of devices and platforms, making it easy for users to access and analyze business data from anywhere.
- Helps in giving a summarised view of data.
- Data cubes store large data in a simple way.

- Data cube operation provides quick and better analysis,
- Improve performance of data.

### Disadvantages of data cube:

- **Complexity**: OLAP systems can be complex to set up and maintain, requiring specialized technical expertise.
- **Data size limitations**: OLAP systems can struggle with very large data sets and may require extensive data aggregation or summarization.
- **Performance issues**: OLAP systems can be slow when dealing with large amounts of data, especially when running complex queries or calculations.
- **Data integrity:** Inconsistent data definitions and data quality issues can affect the accuracy of OLAP analysis.
- **Cost:** OLAP technology can be expensive, especially for enterprise-level solutions, due to the need for specialized hardware and software.
- **Inflexibility:** OLAP systems may not easily accommodate changing business needs and may require significant effort to modify or extend.

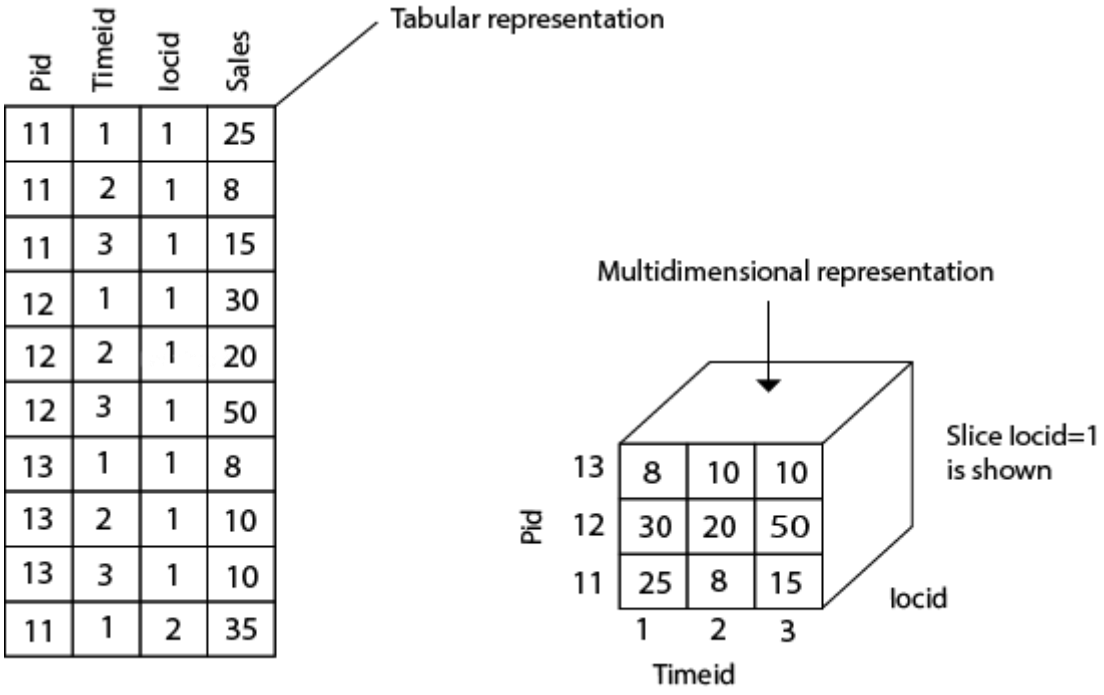
https://radacad.com/how-to-use-time-and-date-dimensions-in-a-power-bi-model

# What is Multi-Dimensional Data Model?

A multidimensional model views data in the form of a data-cube. A data cube enables data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

The dimensions are the perspectives or entities concerning which an organization keeps records. For example, a shop may create a sales data warehouse to keep records of the store's sales for the dimension time, item, and location. These dimensions allow the save to keep track of things, for example, monthly sales of items and the locations at which the items were sold. Each dimension has a table related to it, called a dimensional table, which describes the dimension further. For example, a dimensional table for an item may contain the attributes item_name, brand, and type.

A multidimensional data model is organized around a central theme, for example, sales. This theme is represented by a fact table. **Facts are numerical measures. The** fact table contains the names of the facts or measures of the related dimensional tables.




Consider the data of a shop for items sold per quarter in the city of Delhi. The data is shown in the table. In this 2D representation, the sales for Delhi are shown for the time dimension (organized in quarters) and the item dimension (classified according to the types of an item sold). The fact or measure displayed in rupee_sold (in thousands).

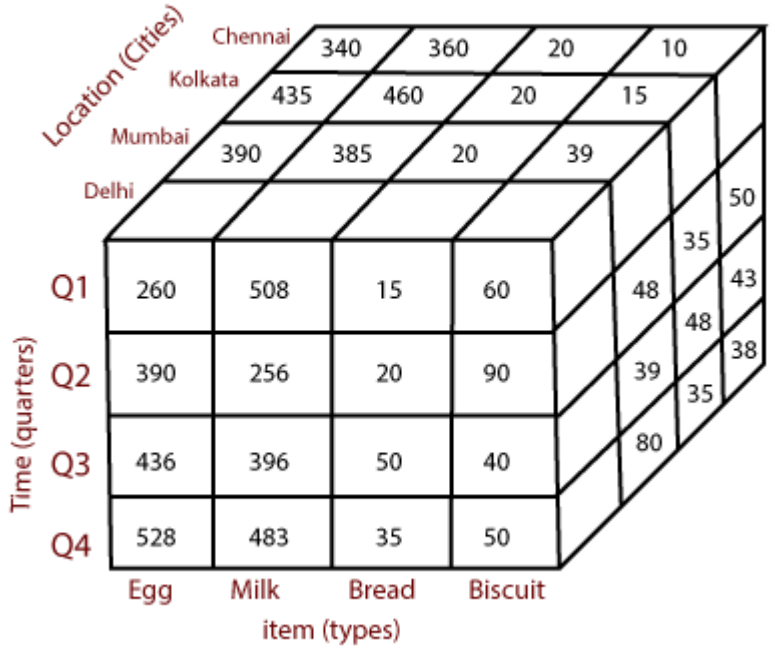


| Location="Delhi" | | | | |
|---|---|---|---|---|
| Time (quarter) | item (type) | | | |
| | Egg | Milk | Bread | Biscuit |
| Q1 | 260 | 508 | 15 | 60 |
| Q2 | 390 | 256 | 20 | 90 |
| Q3 | 436 | 396 | 50 | 40 |
| Q4 | 528 | 483 | 35 | 50 |

Now, if we want to view the sales data with a third dimension, For example, suppose the data according to time and item, as well as the location is considered for the cities Chennai, Kolkata, Mumbai, and Delhi. These 3D data are shown in the table. **The 3D data of the table are represented as a series of 2D tables.**

| | Location="Chennai" | | | | Location="Kolkata" | | | | Location="Mumbai" | | | | Location="Delhi" | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | item | | | | item | | | | item | | | | item | | | |
| Time | Egg | Milk | Bread | Biscuit | Egg | Milk | Bread | Biscuit | Egg | Milk | Bread | Biscuit | Egg | Milk | Bread | Biscuit |
| Q1 | 340 | 360 | 20 | 10 | 435 | 460 | 20 | 15 | 390 | 385 | 20 | 39 | 260 | 508 | 15 | 60 |
| Q2 | 490 | 490 | 16 | 50 | 389 | 385 | 45 | 35 | 463 | 366 | 25 | 48 | 390 | 256 | 20 | 90 |
| Q3 | 680 | 583 | 46 | 43 | 684 | 490 | 39 | 48 | 568 | 594 | 36 | 39 | 436 | 396 | 50 | 40 |
| Q4 | 535 | 694 | 39 | 38 | 335 | 365 | 83 | 35 | 338 | 484 | 48 | 80 | 528 | 483 | 35 | 50 |

Conceptually, it may also be represented by the same data in the form of a 3D data cube, as shown in fig:

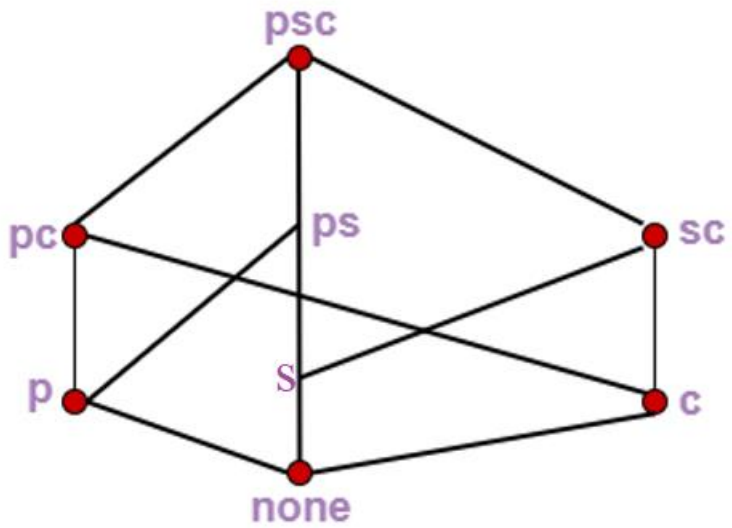



# What is Data Cube?

When data is grouped or combined in multidimensional matrices called Data Cubes. The data cube method has a few alternative names or a few variants, such as "Multidimensional databases," "materialized views," and "OLAP (On-Line Analytical Processing)."

The general idea of this approach is to materialize certain expensive computations that are frequently inquired.

**For example,** a relation with the schema sales (part, supplier, customer, and sale-price) can be materialized into a set of eight views as shown in fig, where **psc=** part, supplier, customer indicates a view consisting of aggregate function value (such as total-sales) computed by grouping three attributes part, supplier, and customer, **p** indicates a view composed of the corresponding aggregate function values calculated by grouping part alone, etc.




A data cube is created from a subset of attributes in the database. Specific attributes are chosen to be measure attributes, i.e., the attributes whose values are of interest. Another attributes are selected as dimensions or functional attributes. The measure attributes are aggregated according to the dimensions.

For example, XYZ may create a sales data warehouse to keep records of the store's sales for the dimensions time, item, branch, and location. These dimensions enable the store to keep track of things like monthly sales of items, and the branches and locations at which the items were sold. Each dimension may have a table identify with it, known as a dimensional table, which describes the dimensions. For example, a dimension table for items may contain the attributes item_name, brand, and type.

Data cube method is an interesting technique with many applications. Data cubes could be sparse in many cases because not every cell in each dimension may have corresponding data in the database.

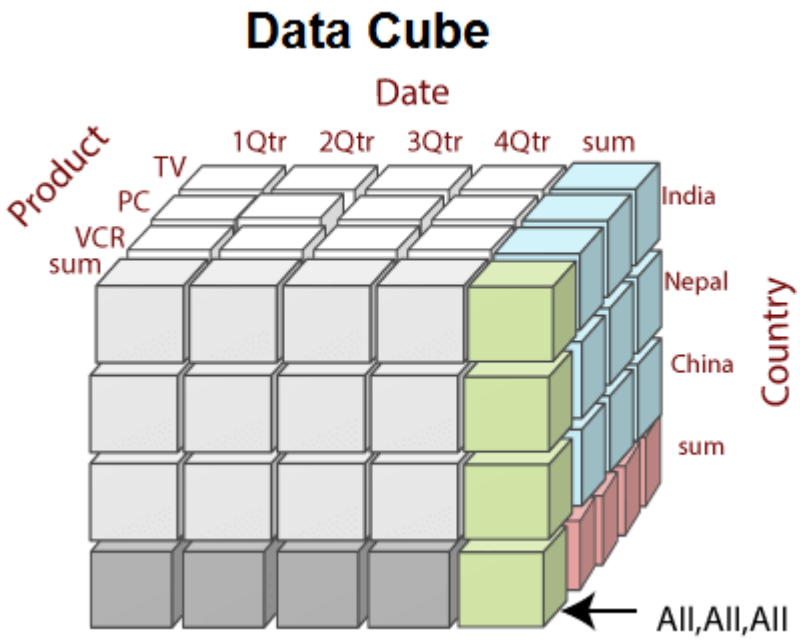
Techniques should be developed to handle sparse cubes efficiently.

If a query contains constants at even lower levels than those provided in a data cube, it is not clear how to make the best use of the precomputed results stored in the data cube.

The model view data in the form of a data cube. OLAP tools are based on the multidimensional data model. Data cubes usually model n-dimensional data.

A data cube enables data to be modeled and viewed in multiple dimensions. A multidimensional data model is organized around a central theme, like sales and transactions. A fact table represents this theme. Facts are numerical measures. Thus, the fact table contains measure (such as Rs_sold) and keys to each of the related dimensional tables.

Dimensions are a fact that defines a data cube. Facts are generally quantities, which are used for analyzing the relationship between dimensions.




nederste er summen af alle enheder

**Example:** In the **2-D representation**, we will look at the All Electronics sales data for **items sold per quarter** in the city of Vancouver. The measured display in dollars sold (in thousands).

## 2-D view of Sales Data

| location ="Vancouver" | | | | |
|---|---|---|---|---|
| | item (type) | | | |
| time (quarter) | home entertainment | computer | phone | security |
| Q1 | 605 | 825 | 14 | 400 |
| Q2 | 680 | 952 | 31 | 512 |
| Q3 | 812 | 1023 | 30 | 501 |
| Q3 | 927 | 1038 | 38 | 580 |

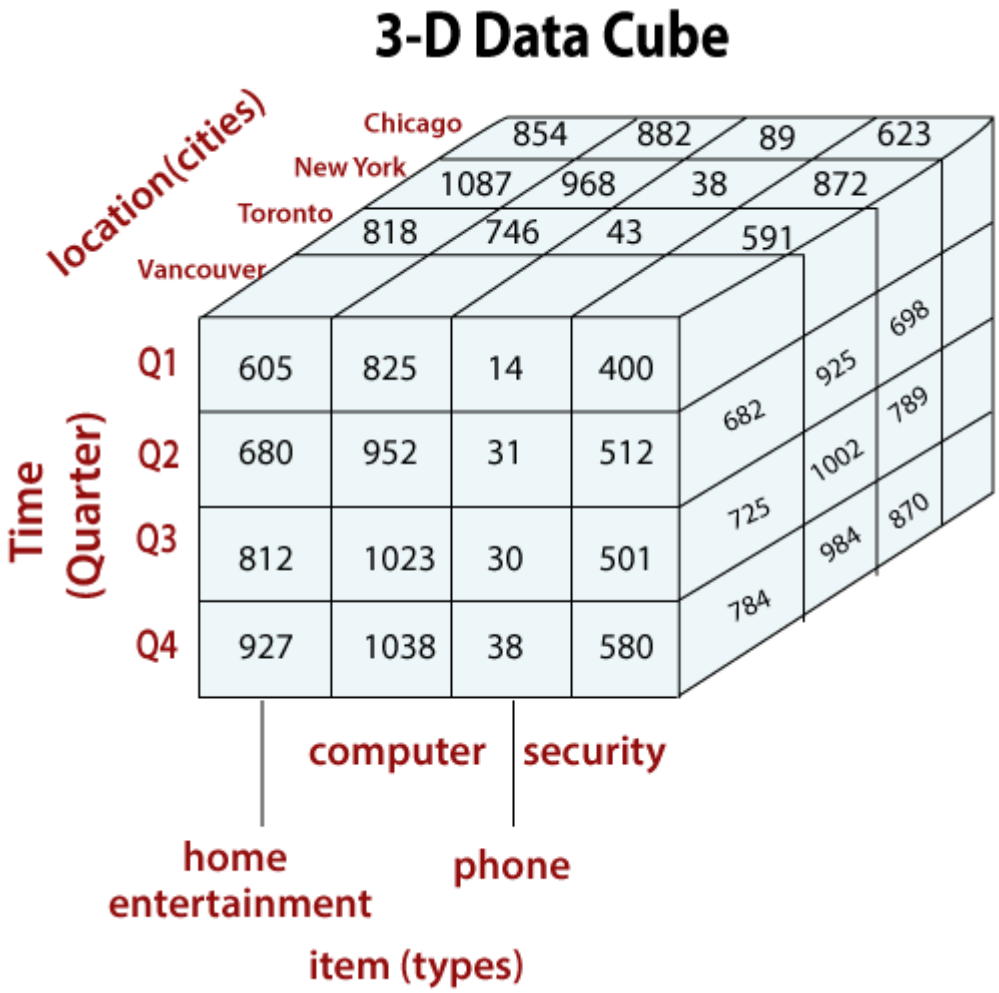
# 3-Dimensional Cuboids

Let suppose we would like to view the sales data with a third dimension. For example, suppose we would like to view the data according to time, item as well as the location for the cities Chicago, New York, Toronto, and Vancouver. The measured display in dollars sold (in thousands). These 3-D data are shown in the table. The 3-D data of the table are represented as a series of 2-D tables.

## 3-D view of Sales Data

| location ="Chicago" | | | | | location ="New York" | | | | location ="Toronto" | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| item | | | | | item | | | | item | | | |
| time | home ent. | comp. | phone | sec. | time | comp. | phone | sec. | home ent. | comp. | phone | sec. |
| Q1 | 854 | 882 | 89 | 623 | 1087 | 968 | 38 | 872 | 818 | 746 | 43 | 591 |
| Q2 | 943 | 890 | 64 | 698 | 1130 | 1024 | 41 | 925 | 894 | 769 | 52 | 682 |
| Q3 | 1032 | 924 | 59 | 789 | 1034 | 1048 | 45 | 1002 | 940 | 795 | 58 | 728 |
| Q4 | 1129 | 992 | 63 | 870 | 1142 | 1091 | 54 | 984 | 978 | 864 | 59 | 784 |

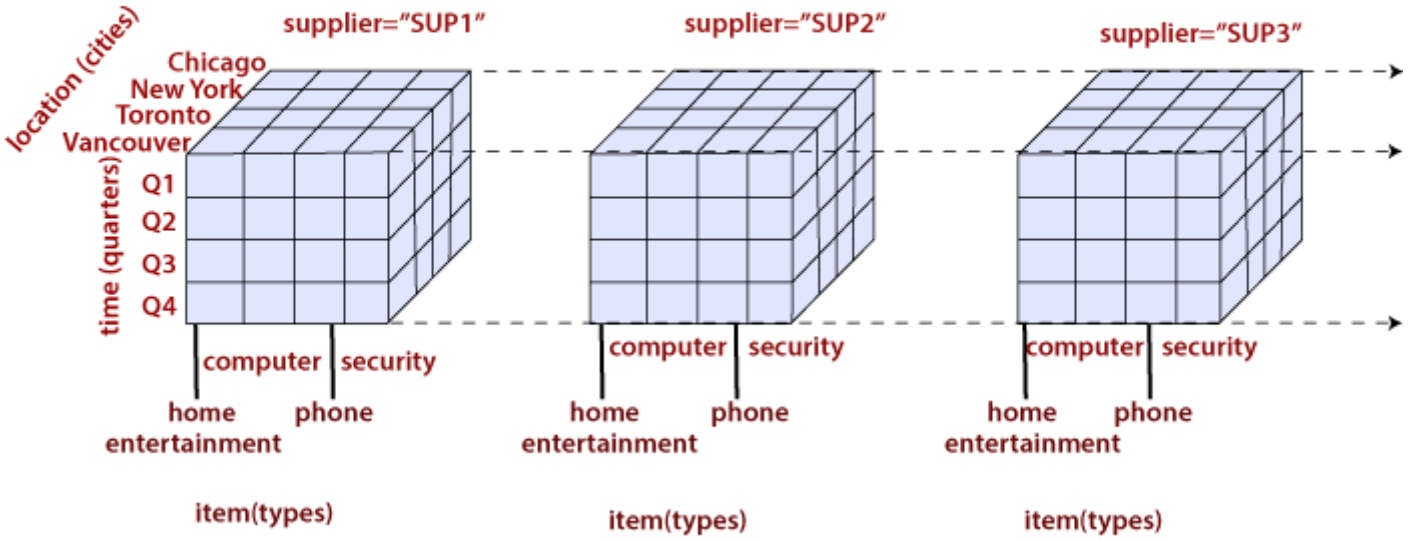
Conceptually, we may represent the same data in the form of 3-D data cubes, as shown in fig:




Let us suppose that we would like to view our sales data with an additional fourth dimension, such as a supplier.

In data warehousing, the data cubes are n-dimensional. The cuboid which holds the lowest level of summarization is called a **base cuboid**.

For example, the **4-D cuboid** in the figure is the base cuboid for the given time, item, location, and supplier dimensions.




Terningerne svarer til {item,time,location} -knuden i figur nedenfor, der skal udregnes sæt af knuder, en for hver supplierværdi + summen for alle supplier
Kan gemmes i én tabel med tid,loc,item , med krydsprodukt-kombinationer:

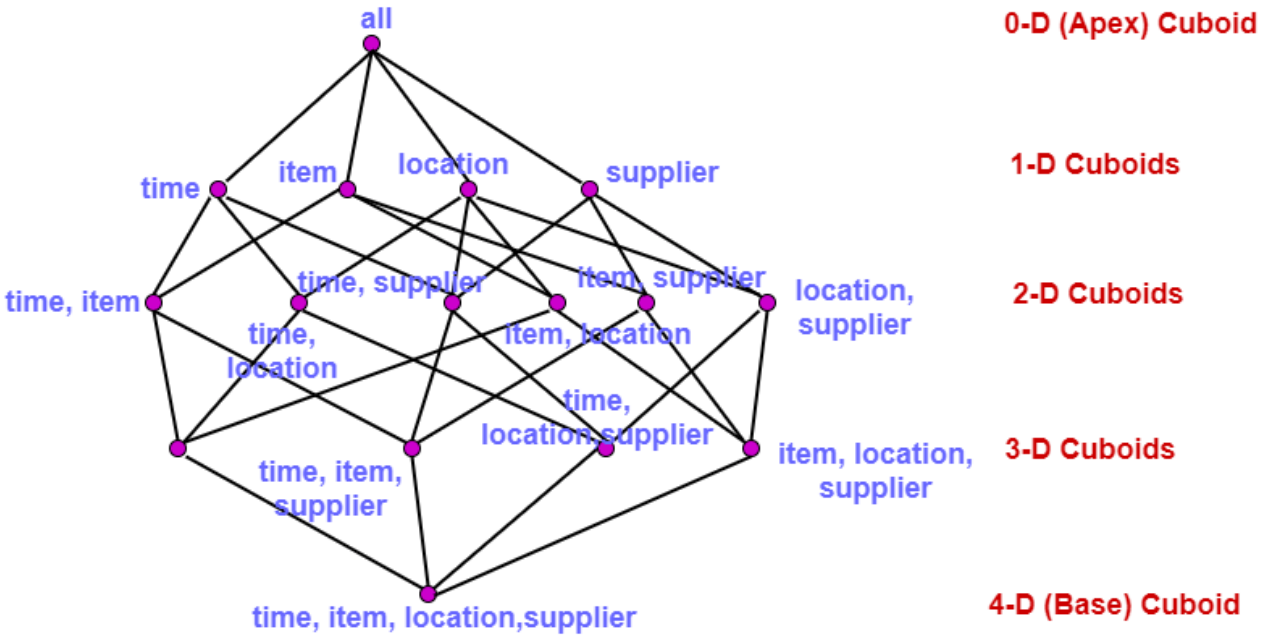
| sup | tid(Q | loc( | item |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 |
| 1 | 1 | 1 | 3 |
| 1 | 1 | 2 | 1 |
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 3 | 1 |
| 1 | 1 | 3 | 2 |
| 1 | 1 | 3 | 3 |
| 1 | 2 | 1 | 1 |
| 1 | 2 | 1 | 2 |
| 1 | 2 | 1 | 3 |
| 1 | 2 | 2 | 1 |
| 1 | 2 | 2 | 2 |
| 1 | 2 | 2 | 3 |
| 1 | 2 | 3 | 1 |
| 1 | 2 | 3 | 2 |
| 1 | 2 | 3 | 3 |
| .. | .. | .. | .. |

For supplier=sup1 har vi første cube, nok med ekstra sumfelt i hver af de 3 dim, T:{Q,Q-sum, felter: T=Qn,Q=Q-sum}, Itime: {item, item-sum}, location:{ locations,sum-locations}

Figure is shown a **4-D data cube** representation of sales data, according to the dimensions time, item, location, and supplier. The measure displayed is dollars sold (in thousands).

The topmost **0-D cuboid**, which holds the highest level of summarization, is known as the apex cuboid. In this example, this is the total sales, or dollars sold, summarized over all four dimensions.

The lattice of cuboid forms a data cube. The figure shows the lattice of cuboids creating 4-D data cubes for the dimension time, item, location, and supplier. Each cuboid represents a different degree of summarization.




**1D time:** sum over alle(item,location,supplier), undtagen time, så for hver Quater Q har vi total salg, med alle (item,location,supplier), kan laves som gruppering over (item,location,supplier) for hvert Q , fordi kun time kan varieres i dette niveau. Men kan også findes som $sum(time,item)_{item}$ el. $sum(time,supplier)_{supplier}$ el. $sum(time,location)_{location}$
2D: time,item, hver hver Q og hver item findes sum af salg og location, en gruppering efter (time,item)
$Sum(time,item)_{item}$=time, for $Q_i$ er $Sum(Q_i,item)_{item}$=total salg i $Q_i$ osv.
All=sum(time)=sum(item)=sum(location)=sum(supplier) til tjek.
For item,location,supplier 3D-knuden: for hver tidsopløsning fx, Q, sum-Q, er lavet sum over Q for hver { item,location,supplier}-kombination

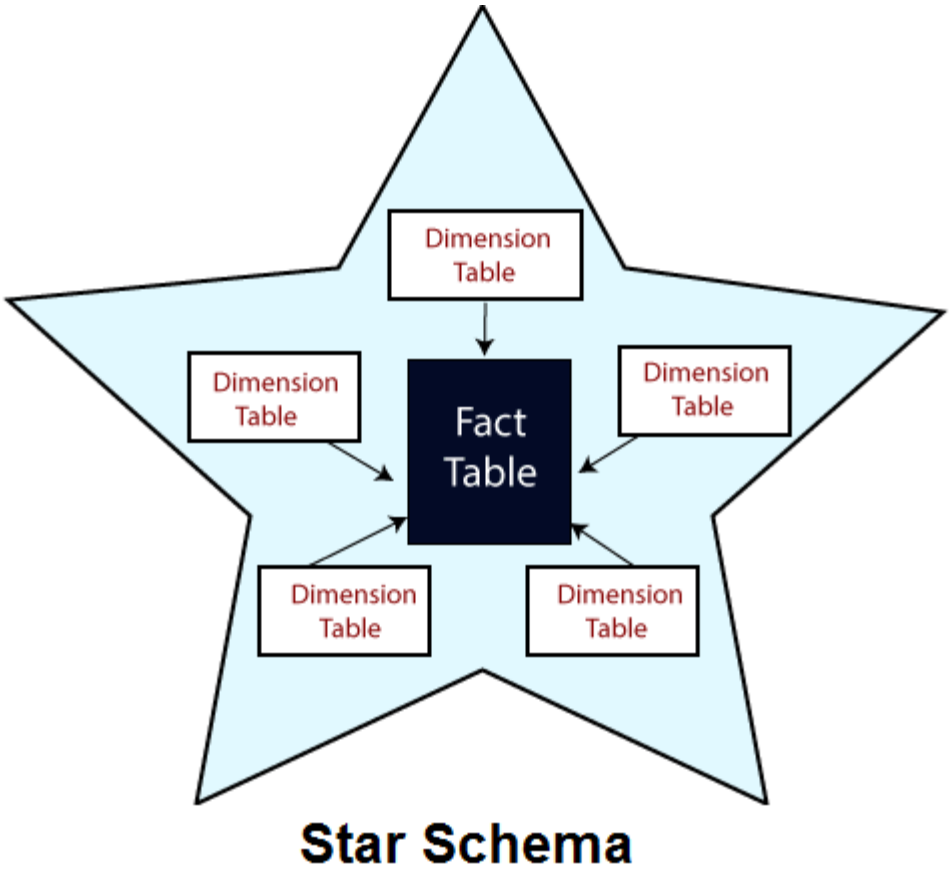
---

Next Topic What is Star Schema

# What is Star Schema?

A star schema is the elementary form of a dimensional model, in which data are organized into **facts** and **dimensions**. A fact is an event that is counted or measured, such as a sale or log in. A dimension includes reference data about the fact, such as date, item, or customer.

A star schema is a relational schema where a relational schema whose design represents a multidimensional data model. The star schema is the explicit data warehouse schema. It is known as **star schema** because the entity-relationship diagram of this schemas simulates a star, with points, diverge from a central table. The center of the schema consists of a large fact table, and the points of the star are the dimension tables.




Star Schema

## Fact Tables

A table in a star schema which contains facts and connected to dimensions. A fact table has two types of columns: those that include fact and those that are foreign keys to the dimension table. The primary key of the fact tables is generally a composite key that is made up of all of its foreign keys.

A fact table might involve either detail level fact or fact that have been aggregated (fact tables that include aggregated fact are often instead called summary tables). A fact table generally contains facts with the same level of aggregation.

# Dimension Tables

A dimension is an architecture usually composed of one or more hierarchies that categorize data. If a dimension has not got hierarchies and levels, it is called a **flat dimension** or **list**. The primary keys of each of the dimensions table are part of the composite primary keys of the fact table. Dimensional attributes help to define the dimensional value. They are generally descriptive, textual values. Dimensional tables are usually small in size than fact table.

Fact tables store data about sales while dimension tables data about the geographic region (markets, cities), clients, products, times, channels.

# Characteristics of Star Schema

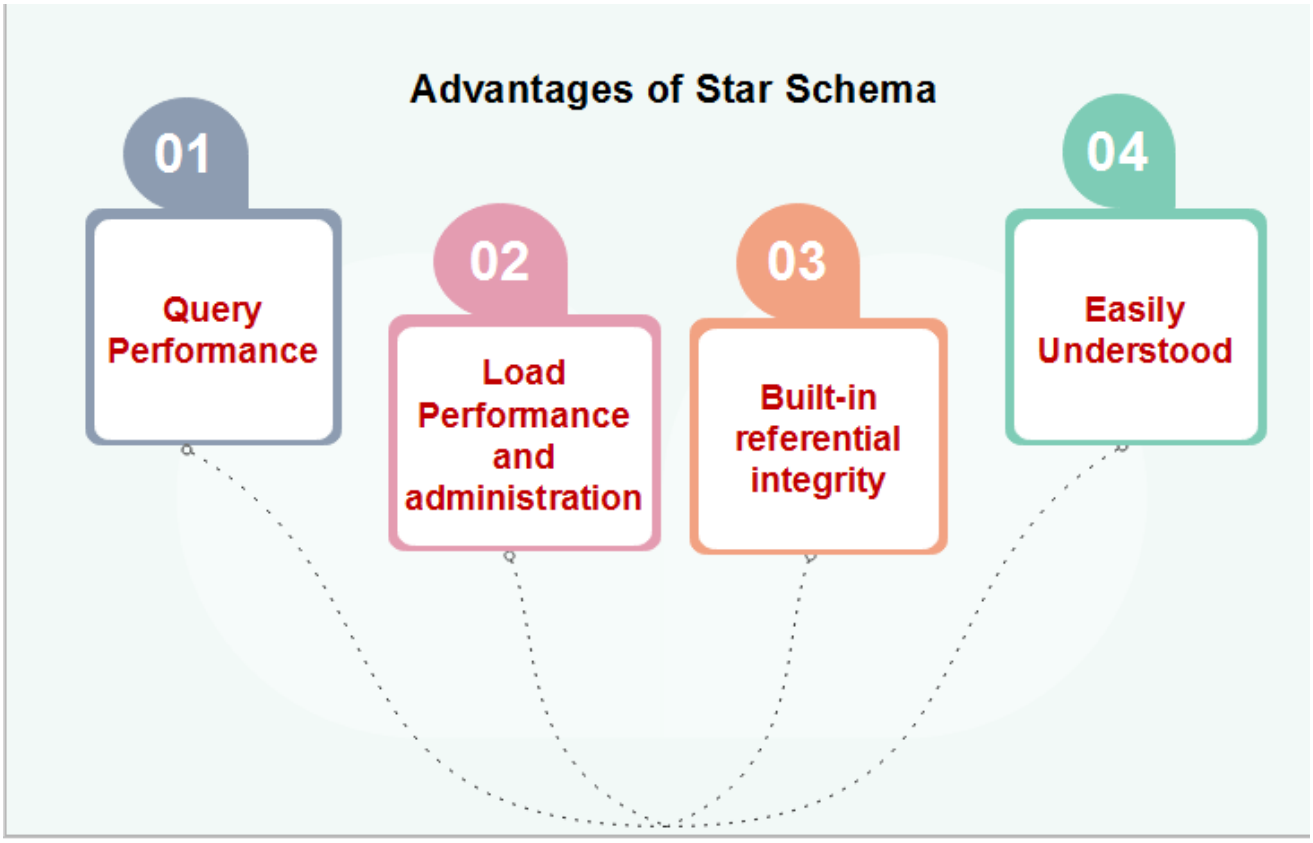
The star schema is intensely suitable for data warehouse database design because of the following features:

- It creates a DE-normalized database that can quickly provide query responses.
- It provides a flexible design that can be changed easily or added to throughout the development cycle, and as the database grows.
- It provides a parallel in design to how end-users typically think of and use the data.
- It reduces the complexity of metadata for both developers and end-users.

# Advantages of Star Schema

Star Schemas are easy for end-users and application to understand and navigate. With a well-designed schema, the customer can instantly analyze large, multidimensional data sets.

The main advantage of star schemas in a decision-support environment are:




## Query Performance

A star schema database has a limited number of table and clear join paths, the query run faster than they do against OLTP systems. Small single-table queries, frequently of a dimension table, are almost instantaneous. Large join queries that contain multiple tables takes only seconds or minutes to run.

In a star schema database design, the dimension is connected only through the central fact table. When the two-dimension table is used in a query, only one join path, intersecting the fact tables, exist between those two tables. This design feature enforces authentic and consistent query results.

## Load performance and administration

Structural simplicity also decreases the time required to load large batches of record into a star schema database. By describing facts and dimensions and separating them into the various table, the impact of a load structure is reduced. Dimension table can be populated once and occasionally refreshed, time dim er fastlagt. We can add new facts regularly and selectively by appending records to a fact table.

## Built-in referential integrity

A star schema has referential integrity built-in when information is loaded. Referential integrity is enforced because each data in dimensional tables has a unique primary key, and all keys in the fact table are legitimate foreign keys drawn from the dimension table. A record in the fact table which is not related correctly to a dimension cannot be given the correct key value to be retrieved.

## Easily Understood

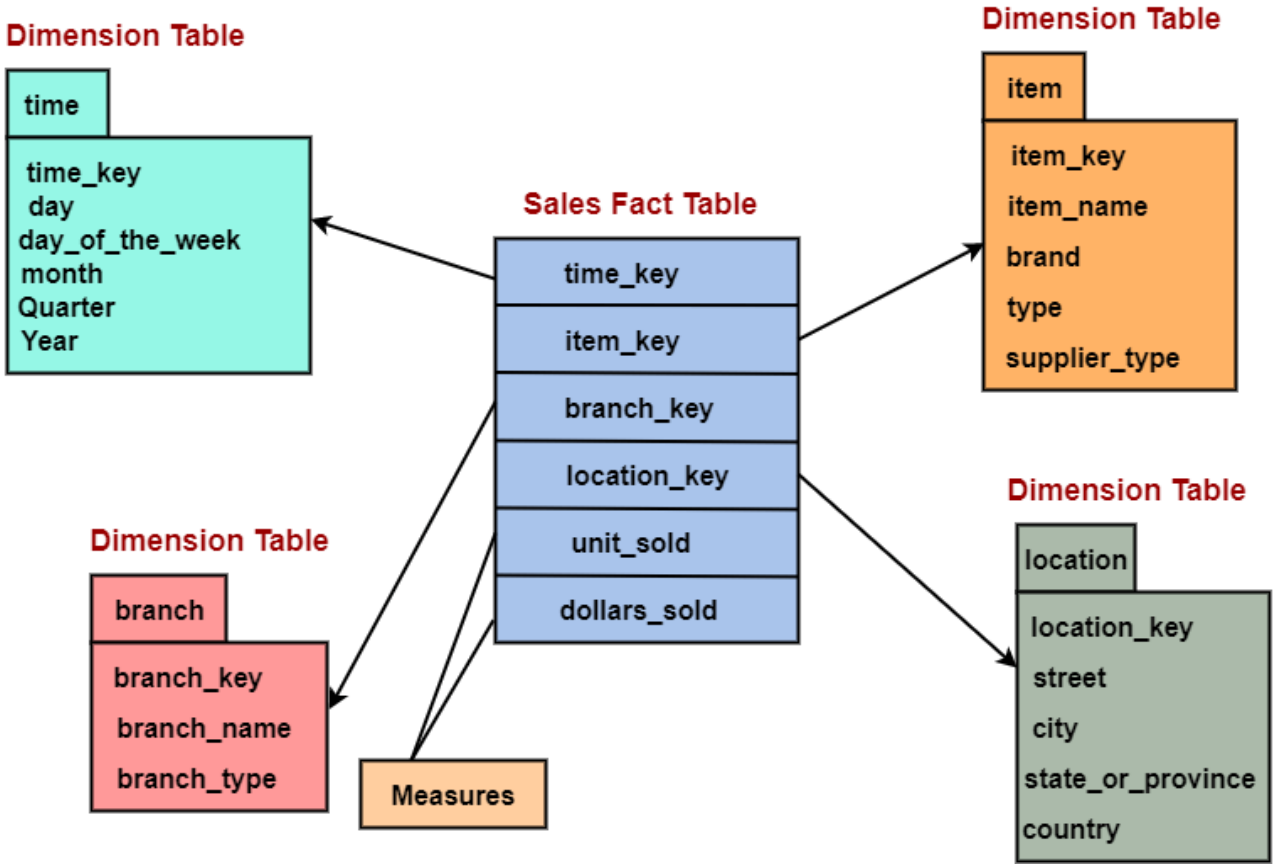
A star schema is simple to understand and navigate, with dimensions joined only through the fact table. These joins are more significant to the end-user because they represent the fundamental relationship between parts of the underlying business. Customer can also browse dimension table attributes before constructing a query.

# Disadvantage of Star Schema

There is some condition which cannot be meet by star schemas like the relationship between the user, and bank account cannot describe as star schema as the relationship between them is many to many.

**Example:** Suppose a star schema is composed of a fact table, SALES, and several dimension tables connected to it for time, branch, item, and geographic locations.

The TIME table has a column for each day, month, quarter, and year. The ITEM table has columns for each item_Key, item_name, brand, type, supplier_type. The BRANCH table has columns for each branch_key, branch_name, branch_type. The LOCATION table has columns of geographic data, including street, city, state, and country.




In this scenario, the SALES table contains only four columns with IDs from the dimension tables, TIME, ITEM, BRANCH, and LOCATION, instead of four columns for time data, four columns for ITEM data, three columns for BRANCH data, and four columns for LOCATION data. Thus, the size of the fact table is significantly reduced. When we need to change an item, we need only make a single change in the dimension table, instead of making many changes in the fact table.

We can create even more complex star schemas by normalizing a dimension table into several tables. The normalized dimension table is called a **Snowflake**. Fx med time, oprette tabeller med år, quarter, måneds, dag info

Next Topic What is Snowflake Schema

# What is Snowflake Schema? Dimensionstabeller opdelt i flere undertabeller,

godt for many-many relationer mellem Fact og dimension

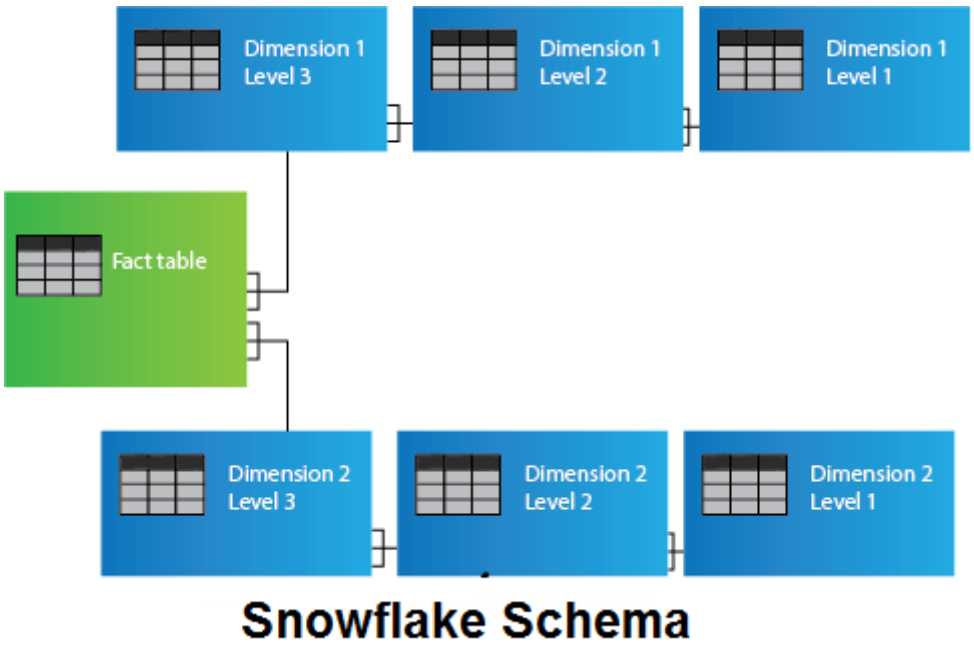
A snowflake schema is equivalent to the star schema. "A schema is known as a snowflake if one or more dimension tables do not connect directly to the fact table but must join through other dimension tables."

The snowflake schema is an expansion of the star schema where each point of the star explodes into more points. It is called snowflake schema because the diagram of snowflake schema resembles a snowflake. **Snowflaking** is a method of normalizing the dimension tables in a STAR schemas. When we normalize all the dimension tables entirely, the resultant structure resembles a snowflake with the fact table in the middle.

Snowflaking is used to develop the performance of specific queries. The schema is diagramed with each fact surrounded by its associated dimensions, and those dimensions are related to other dimensions, branching out into a snowflake pattern.

The snowflake schema consists of one fact table which is linked to many dimension tables, which can be linked to other dimension tables through a many-to-one relationship. Tables in a snowflake schema are generally normalized to the third normal form. Each dimension table performs exactly one level in a hierarchy.

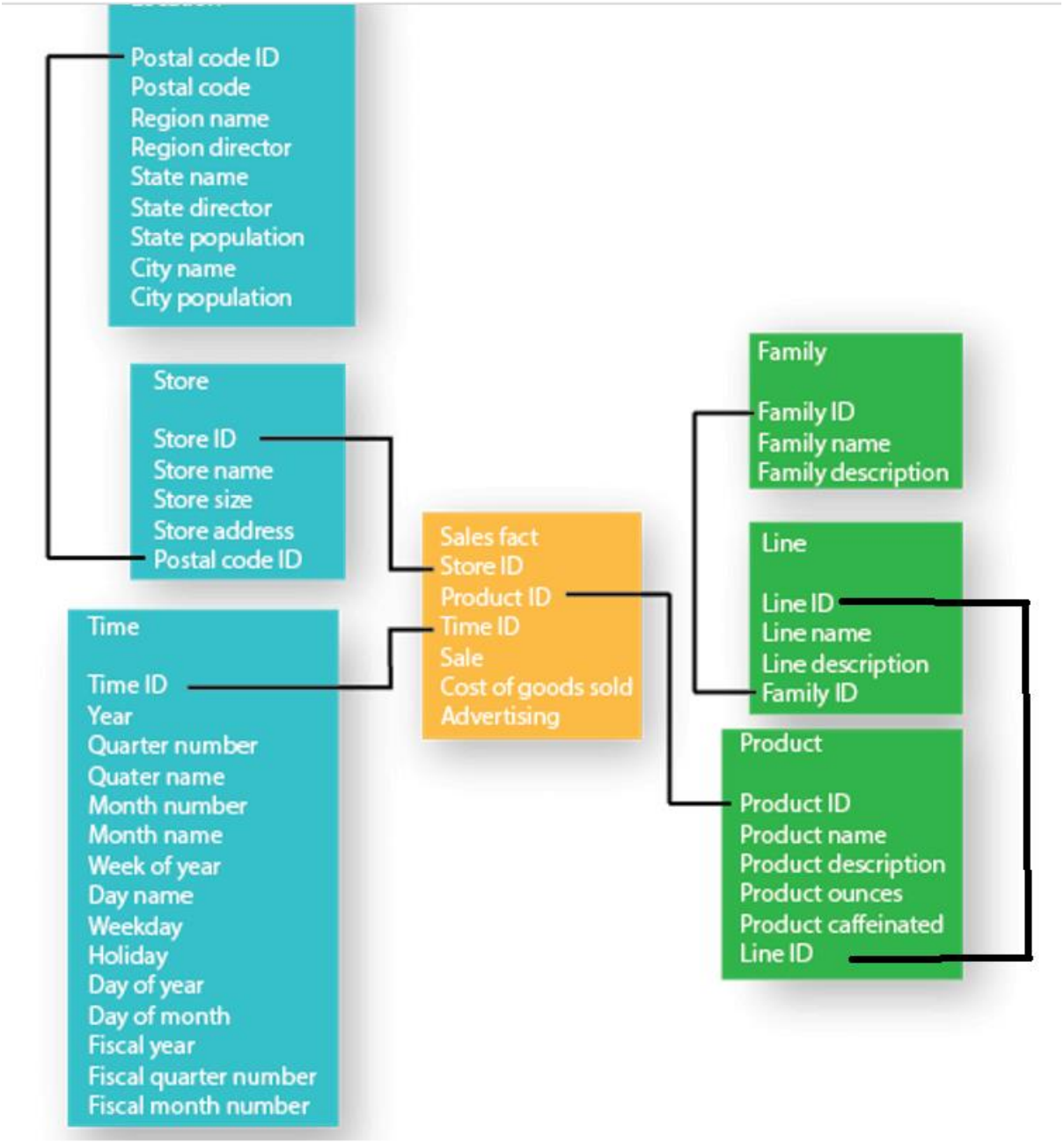
The following diagram shows a snowflake schema with two dimensions, each having three levels. A snowflake schemas can have any number of dimension, and each dimension can have any number of levels.



Snowflake Schema

**Example:**

Figure shows a snowflake schema with a Sales fact table, with Store, Location, Time, Product, Line, and Family dimension tables. The Market dimension has two dimension tables with Store as the primary dimension table, and Location as the outrigger dimension table. The product dimension has three dimension tables with Product as the primary dimension table, and the Line and Family table are the outrigger dimension tables.



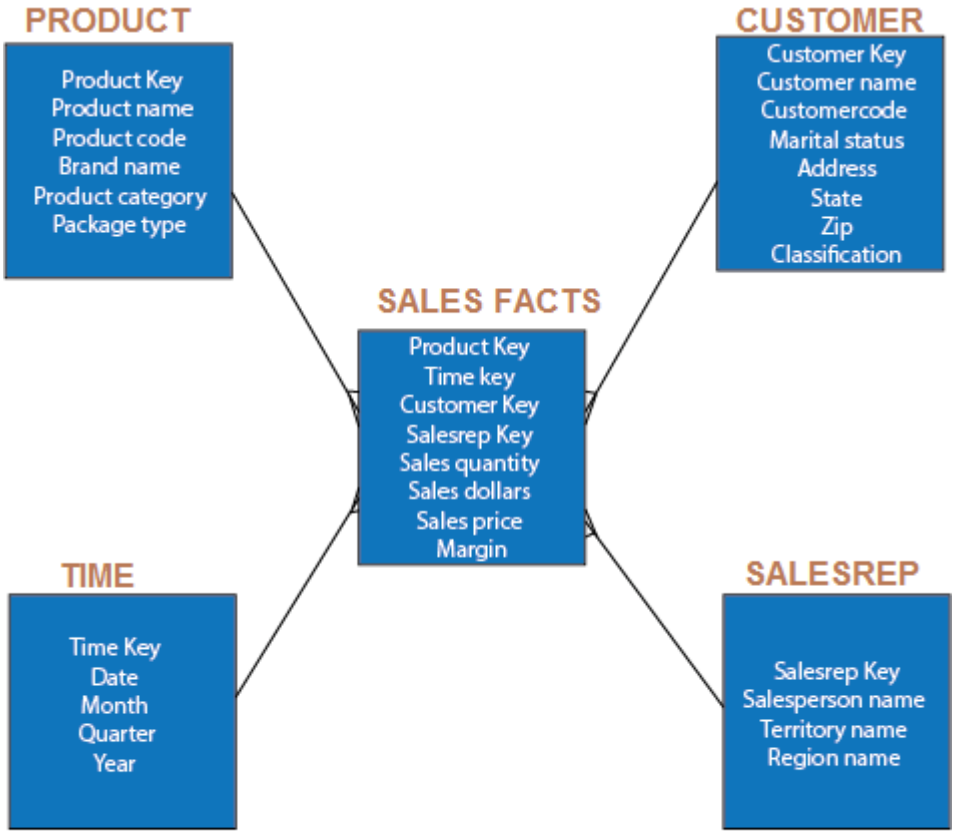

Product-linje angiver typen af produkt, kategor eller måske produkt-familier for fx Apple phone

A star schema store all attributes for a dimension into one denormalized table. This needed more disk space than a more normalized snowflake schema. Snowflaking normalizes the

dimension by moving attributes with low cardinality into separate dimension tables that relate to the core dimension table by using foreign keys. Snowflaking for the sole purpose of minimizing disk space is not recommended, because it can adversely impact query performance.

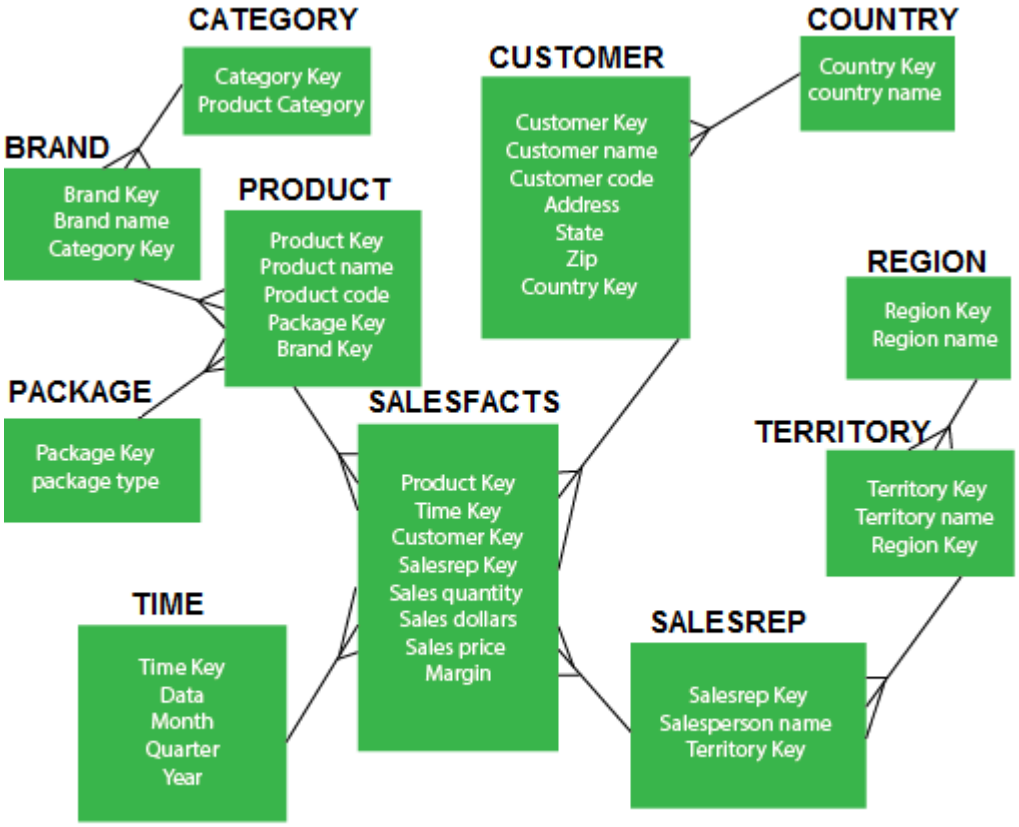
In snowflake, schema tables are normalized to delete redundancy. In snowflake dimension tables are damaged into multiple dimension tables.

Figure shows a simple STAR schema for sales in a manufacturing company. The sales fact table include quantity, price, and other relevant metrics. SALESREP, CUSTOMER, PRODUCT, and TIME are the dimension tables.




**STAR Schema**

The STAR schema for sales, as shown above, contains only five tables, whereas the normalized version now extends to eleven tables. We will notice that in the snowflake schema, the attributes with low cardinality in each original dimension tables are removed to form separate tables. These new tables are connected back to the original dimension table through artificial keys.




Time kunne også være opdelt i år,k kvarter, monåed, dag tabeller ;efter min mening noget rod at Category er koblet til Brand, bør være koblet direkete på Product
Nemt at søge over customers indenfor givet country, 'USA' : CountryKey hurtigt at finde i meget lille Country tabel -> int-value, må være indeks i customer, nem at opslå; alternativt skal man søge efter string Country i Customer tabel

A snowflake schema is designed for flexible querying across more complex dimensions and relationship. It is suitable for many to many and one to many relationships between dimension levels.

# Advantage of Snowflake Schema

1. The primary advantage of the snowflake schema is the development in query performance due to minimized disk storage requirements and joining smaller lookup tables,
2. It provides greater scalability in the interrelationship between dimension levels and components.
3. No redundancy, so it is easier to maintain, -med redundancy står informationen jo flere steder, der så skal rettes korrekt alle steder

## Disadvantage of Snowflake Schema

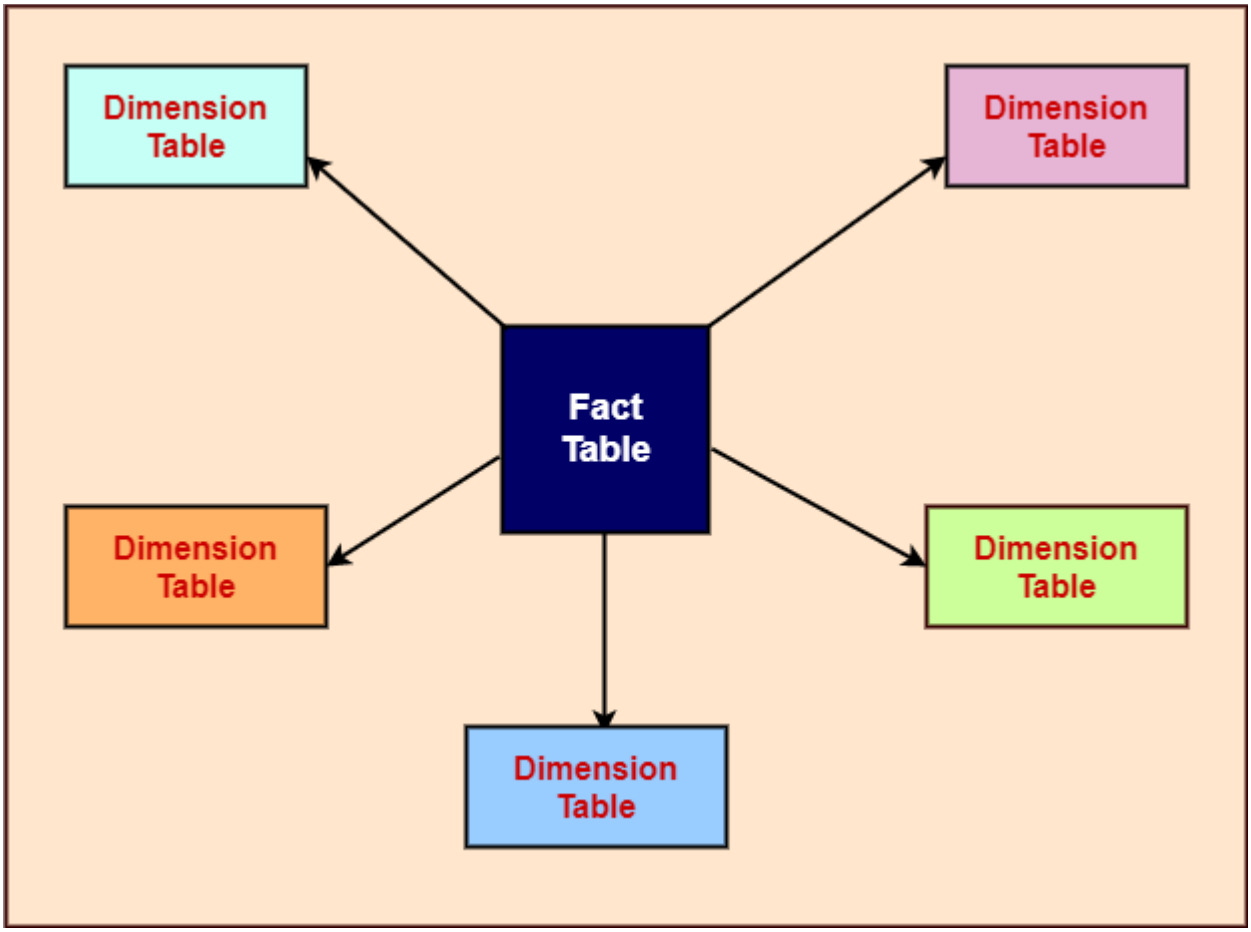
1. The primary disadvantage of the snowflake schema is the additional maintenance efforts required due to the increasing number of lookup tables. It is also known as a multi fact star schema.
2. There are more complex queries and hence, difficult to understand.
3. More tables more join so more query execution time.

---

Next Topic Star Schema vs Snowflake Schema

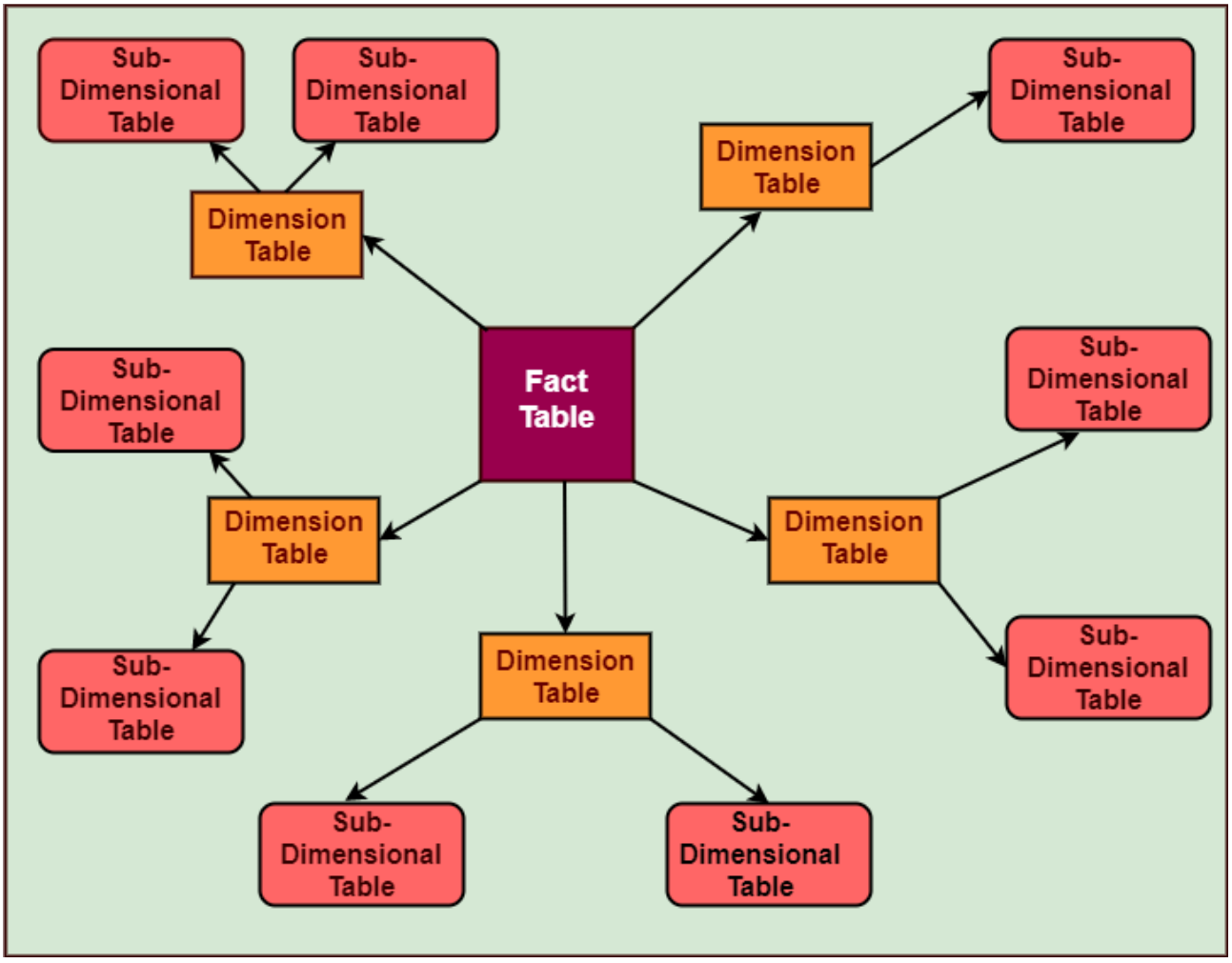
# Difference between Star and Snowflake Schemas

- In a star schema, the fact table will be at the center and is connected to the dimension tables.
- The tables are completely in a denormalized structure.
- SQL queries performance is good as there is less number of joins involved.
- Data redundancy is high and occupies more disk space.

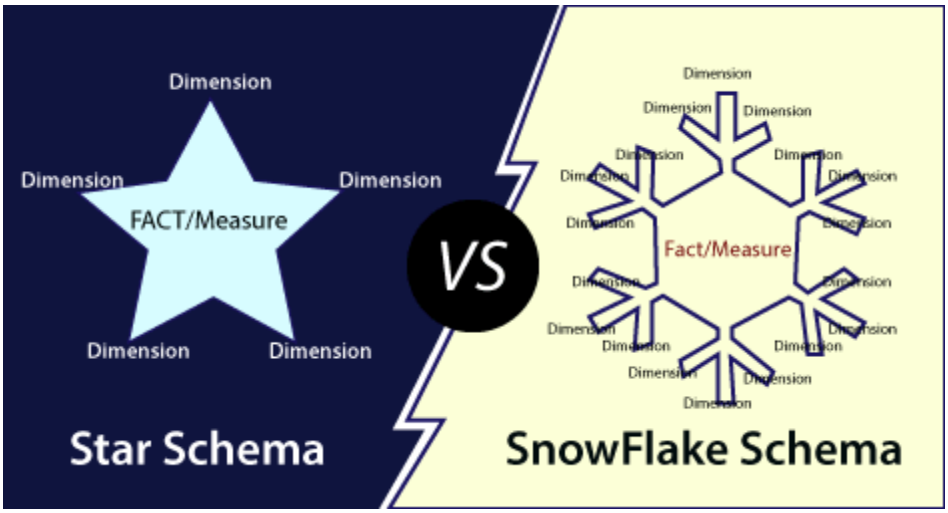



## Snowflake Schema

- A snowflake schema is an extension of star schema where the dimension tables are connected one or more dimensions.
- The tables are partially denormalized in structure.
- The performance of SQL queries is a bit less when compared to star schema as more numbe joins are involved.
- Data redundancy is low and occupies less disk space when compared to star schema.




**Let's see the differentiate between Star and Snowflake Schema.**




| Basis for Comparison | Star Schema | Snowflake Schema |
|---|---|---|

| | | |
|---|---|---|
| Ease of Maintenance/change | It has redundant data and hence less easy to maintain/change | No redundancy and therefore more easy maintain and change |
| Ease of Use | Less complex queries and simple to understand | More complex queries and therefore less easy understand |
| Parent table | In a star schema, a dimension table will not have any parent table | In a snowflake schema, a dimension table have one or more parent tables |
| Query Performance | Less number of foreign keys and hence lesser query execution time | More foreign keys and thus more qu execution time |
| Normalization | It has De-normalized tables | It has normalized tables, in purest form |
| Type of Data Warehouse | Good for data marts with simple relationships (one to one or one to many) | Good to use for data warehouse core to simp complex relationships (many to many) |
| Joins | Fewer joins | Higher number of joins |
| Dimension Table | It contains only a single dimension table for each dimension | It may have more than one dimension table each dimension |
| Hierarchies | Hierarchies for the dimension are stored in the dimensional table itself in a star schema | Hierarchies are broken into separate tables i snowflake schema. These hierarchies help to d down the information from topmost hierarch to the lowermost hierarchies. |
| When to use | When the dimensional table contains less number of rows, we can go for Star schema. | When dimensional table store a huge number rows with redundancy information and space such an issue, we can choose snowflake sche to store space. |
| Data Warehouse system | Work best in any data warehouse/ data mart | Better for small data warehouse/data mart. |

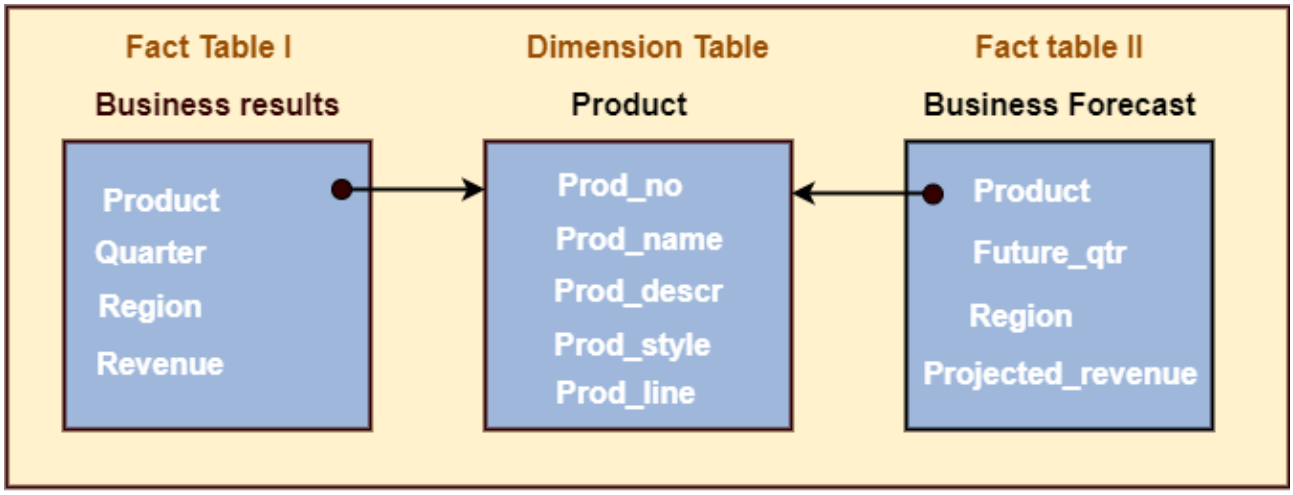
Next Topic What is Fact Constellation Schema

← PrevNext →

# Fact Constellation Schema, når flere Fact-tabeller deles om dimensions-tabel

A Fact constellation means two or more fact tables sharing one or more dimensions. It is also called **Galaxy schema**.

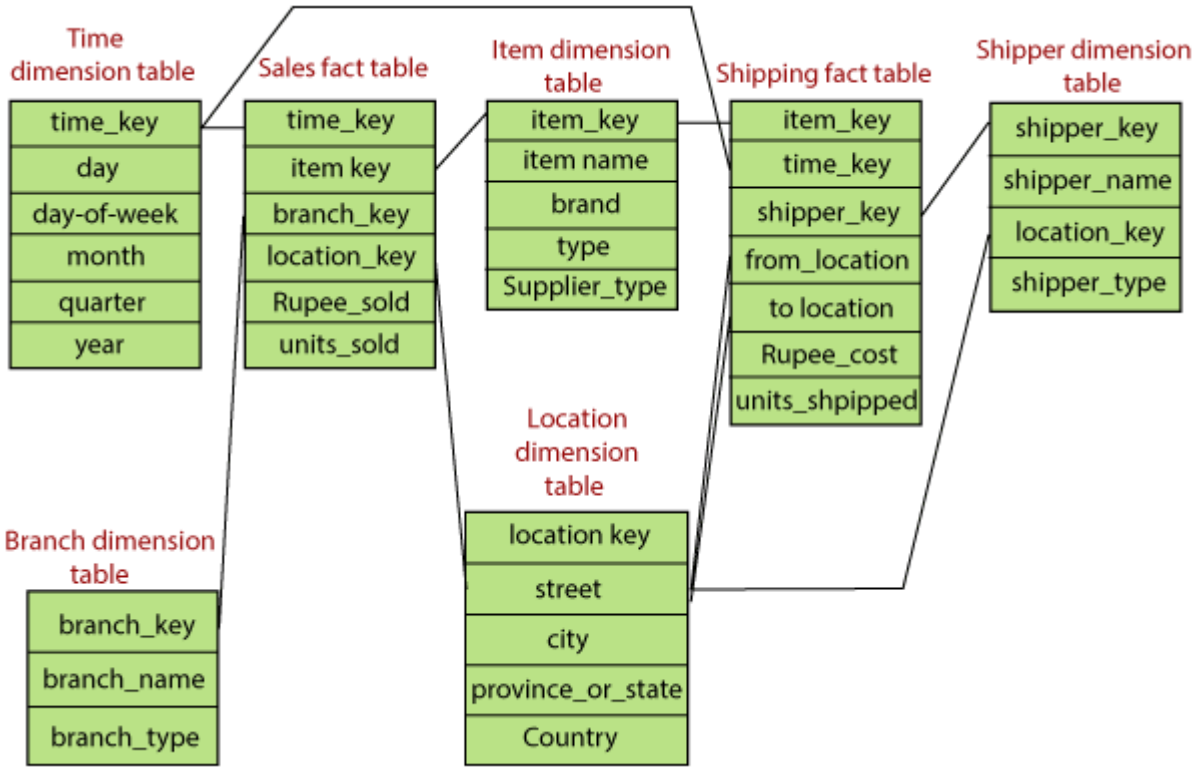
Fact Constellation Schema describes a logical structure of data warehouse or data mart. Fact Constellation Schema can design with a collection of de-normalized FACT, Shared, and Conformed Dimension tables.




FACT Constellation Schema

Fact Constellation Schema is a sophisticated database design that is difficult to summarize information. Fact Constellation Schema can implement between aggregate Fact tables or decompose a complex Fact table into independent simplex Fact tables.

**Example:** A fact constellation schema is shown in the figure below.

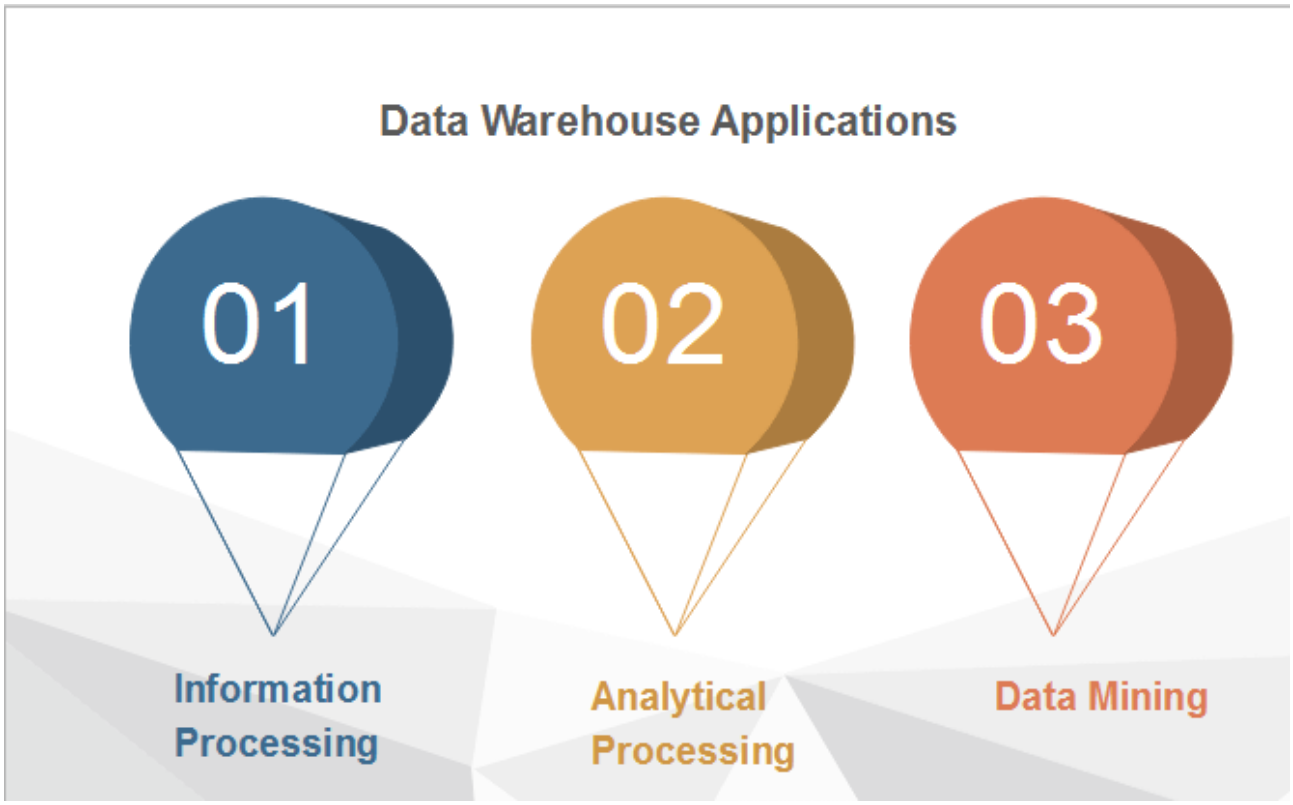



This schema defines two fact tables, sales, and shipping. Sales are treated along four dimensions, namely, time, item, branch, and location. The schema contains a fact table for sales that includes keys to each of the four dimensions, along with two measures: Rupee_sold and units_sold. The shipping table has five dimensions, or keys: item_key, time_key, shipper_key, from_location, and to_location, and two measures: Rupee_cost and units_shipped.

The primary disadvantage of the fact constellation schema is that it is a more challenging design because many variants for specific kinds of aggregation must be considered and selected.

## Data Warehouse Applications

The application areas of the data warehouse are:




**Information Processing**

It deals with querying, statistical analysis, and reporting via tables, charts, or graphs. Nowadays, information processing of data warehouse is to construct a low cost, web-based accessing tools typically integrated with web browsers.

**Analytical Processing**

It supports various online analytical processing such as drill-down, roll-up, and pivoting. The historical data is being processed in both summarized and detailed format.

OLAP is implemented on data warehouses or data marts. The primary objective of OLAP is to support ad-hoc querying needed for support DSS. The multidimensional view of data is fundamental to the OLAP application. OLAP is an operational view, not a data structure or schema. The complex nature of OLAP applications requires a multidimensional view of the data.

**Data Mining**

It helps in the analysis of hidden design and association, constructing scientific models, operating classification and prediction, and performing the mining results using visualization tools.

Data mining is the technique of designing/find out essential new correlations, patterns, and trends by changing through high amounts of a record save in repositories, using pattern recognition technologies as well as statistical and mathematical techniques.

It is the phase of selection, exploration, and modeling of huge quantities of information to determine regularities or relations that are at first unknown to access precise and useful results for the owner of the database.

It is the process of inspection and analysis, by automatic or semi-automatic means, of large quantities of records to discover meaningful patterns and rules.

---

Next Topic Data Warehouse Process Architecture

← PrevNext →