OLAP Data Cube Research & Development Documentation

Andrés Montenegro
MLS Data Tools
andres.m@mlsdatatools.com

Table of Contents

Data Cube

In computer programming contexts, a data cube (or datacube) is a multi-dimensional ("n-D") array of values. Typically, the term datacube is applied in contexts where these arrays are massively larger than the hosting computer's main memory; examples include multi-terabyte/petabyte data warehouses and time series of image data.

The data cube is used to represent data (sometimes called facts) along some measure of interest. For example, in OLAP such measures could be the subsidiaries a company has, the products the company offers, and time; in this setup, a fact would be a sales event where a particular product has been sold in a particular subsidiary at a particular time. In satellite image timeseries measures would be Latitude and Longitude coordinates and time; a fact would be a pixel at a given space/time coordinate as taken by the satellite (following some processing that is not of concern here). Even though it is called a *cube* (and the examples provided above happen to be 3-dimensional for brevity), a data cube generally is a multi-dimensional concept which can be 1-dimensional, 2-dimensional, 3-dimensional, or higher-dimensional. In any case, every dimension represents a separate measure whereas the cells in the cube represent the facts of interest. Sometimes cubes hold only few values with the rest being *empty*, i.e.: undefined, sometimes most or all cube coordinates hold a cell value. In the first case such data are called *sparse*, in the second case they are called *dense*, although there is no hard delineation between both.

Standardization

In 2018, the ISO SQL database language is getting extended with datacube functionality as "SQL -- Part 15: Multi-dimensional arrays (SQL/MDA)".

Web Coverage Processing Service is a geo datacube analytics language issued by the Open Geospatial Consortium in 2008. In addition to the common datacube operations the language knows about the semantics of space and time and supports both regular and irregular grid datacubes, based on the concept of coverage data.

An industry standard for querying business datacubes, originally developed by Microsoft, is MultiDimensional eXpressions.

Implementation

Many high-level computer languages treat data cubes and other large arrays as single entities distinct from their contents. These languages, of which APL, IDL, NumPy, PDL, and S-Lang are examples, allow the programmer to manipulate complete film clips and other data en masse with simple expressions derived from linear algebra and vector mathematics. Some languages (such as PDL) distinguish between a list of images and a data cube, while many (such as IDL) do not.

Array DBMSs (Database Management Systems) offer a data model which generically supports definition, management, retrieval, and manipulation of n-dimensional datacubes. This database category has been pioneered by the rasdaman system since 1994.

Online Analytical Processing (OLAP)

Online analytical processing, or **OLAP**, is an approach to answer multi-dimensional analytical (MDA) queries swiftly in computing. OLAP is part of the broader category of business intelligence, which also encompasses relational databases, report writing and data mining. Typical applications of OLAP include business reporting for sales, marketing, management reporting, business process management (BPM), budgeting and forecasting, financial reporting and similar areas, with new applications emerging, such as agriculture. The term *OLAP* was created as a slight modification of the traditional database term online transaction processing (OLTP).

OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP consists of three basic analytical operations: consolidation (roll-up), drill-down, and slicing and dicing. Consolidation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For example, all sales offices are rolled up to the sales department or sales division to anticipate sales trends. By contrast, the drill-down is a technique that allows users to navigate through the details. For instance, users can view the sales by individual products that make up a region's sales. Slicing and dicing is a feature whereby users can take out (slicing) a specific set of data of the OLAP cube and view (dicing) the slices from different viewpoints. These viewpoints are sometimes called dimensions (such as looking at the same sales by salesperson, or by date, or by customer, or by product, or by region, etc.)

Databases configured for OLAP use a multidimensional data model, allowing for complex analytical and ad hoc queries with a rapid execution time. They borrow aspects of navigational databases, hierarchical databases and relational databases.

OLAP is typically contrasted to OLTP (online transaction processing), which is generally characterized by much less complex queries, in a larger volume, to process transactions rather than for the purpose of business intelligence or reporting. Whereas OLAP systems are mostly optimized for read, OLTP has to process all kinds of queries (read, insert, update and delete).

Overview

At the core of any OLAP system is an OLAP cube (also called a 'multidimensional cube' or a hypercube). It consists of numeric facts called *measures* that are categorized by *dimensions*. The measures are placed at the intersections of the hypercube, which is spanned by the dimensions as a vector space. The usual interface to manipulate an OLAP cube is a matrix interface, like Pivot tables in a spreadsheet program, which performs projection operations along the dimensions, such as aggregation or averaging.

The cube metadata is typically created from a star schema or snowflake schema or fact constellation of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

Each *measure* can be thought of as having a set of *labels*, or meta-data associated with it. A *dimension* is what describes these *labels*; it provides information about the *measure*.

A simple example would be a cube that contains a store's sales as a *measure*, and Date/Time as a *dimension*. Each Sale has a Date/Time *label* that describes more about that sale.

For example:

```
Sales Fact Table
+-------------+----------+
| sale_amount | time_id  |
+-------------+----------+            Time Dimension
|      2008.10|     1234 |----+     +---------+-------------------+
+-------------+----------+    |     | time_id | timestamp         |
                              |     +---------+-------------------+
                              +---->|   1234  | 20080902 12:35:43 |
                                    +---------+-------------------+
```

Multidimensional Databases

Multidimensional structure is defined as "a variation of the relational model that uses multidimensional structures to organize data and express the relationships between data". The structure is broken into cubes and the cubes are able to store and access data within the confines of each cube. "Each cell within a multidimensional structure contains aggregated data related to elements along each of its dimensions". Even when data is manipulated it remains easy to access and continues to constitute a compact database format. The data still remains interrelated. Multidimensional structure is quite popular for analytical databases that use online analytical processing (OLAP) applications. Analytical databases use these databases because of their ability to deliver answers to complex business queries swiftly. Data can be viewed from different angles, which gives a broader perspective of a problem unlike other models.

Aggregations

It has been claimed that for complex queries OLAP cubes can produce an answer in around 0.1% of the time required for the same query on OLTP relational data. The most important mechanism in OLAP which allows it to achieve such performance is the use of *aggregations*. Aggregations are built from the fact table by changing the granularity on specific dimensions and aggregating up data along these dimensions, using an aggregate function (or *aggregation function*). The number of possible aggregations is determined by every possible combination of dimension granularities.

The combination of all possible aggregations and the base data contains the answers to every query which can be answered from the data.

Because usually there are many aggregations that can be calculated, often only a predetermined number are fully calculated; the remainder are solved on demand. The problem of deciding which aggregations (views) to calculate is known as the view selection problem. View selection can be constrained by the total size of the selected set of aggregations, the time to update them from changes in the base data, or both. The objective of view selection is typically to minimize the average time to answer OLAP queries, although some studies also minimize the update time. View selection is NP-Complete. Many approaches to the problem have been explored, including greedy algorithms, randomized search, genetic algorithms and A* search algorithm.

Some aggregation functions can be computed for the entire OLAP cube by precomputing values for each cell, and then computing the aggregation for a roll-up of cells by aggregating these aggregates, applying a divide and conquer algorithm to the multidimensional problem to compute them efficiently. For example, the overall sum of a roll-up is just the sum of the sub-sums in each cell. Functions that can be decomposed in this way are called decomposable aggregation functions, and include COUNT, MAX, MIN, and SUM, which can be computed for each cell and then directly aggregated; these are known as self-decomposable aggregation functions. In other cases, the aggregate function can be computed by computing auxiliary numbers for cells, aggregating these auxiliary numbers, and finally computing the overall number at the end; examples include AVERAGE (tracking sum and count, dividing at the end) and RANGE (tracking max and min, subtracting at the end). In other cases, the aggregate function cannot be computed without analyzing the entire set at once, though in some cases approximations can be computed; examples include DISTINCT COUNT, MEDIAN, and MODE; for example, the median of a set is not the median of medians of subsets. These latter are difficult to implement efficiently in OLAP, as they require computing the aggregate function on the base data, either computing them online (slow) or precomputing them for possible rollouts (large space).

Types

OLAP systems have been traditionally categorized using the following taxonomy.

Multidimensional OLAP (MOLAP)

MOLAP (multi-dimensional online analytical processing) is the classic form of OLAP and is sometimes referred to as just OLAP. MOLAP stores this data in an optimized multi-dimensional array storage, rather than in a relational database.

Some MOLAP tools require the pre-computation and storage of derived data, such as consolidations – the operation known as processing. Such MOLAP tools generally utilize a pre-calculated data set referred to as a data cube. The data cube contains all the possible answers to a given range of questions. As a result, they have a very fast response to queries. On the other hand, updating can take a long time depending on the degree of pre-computation. Pre-computation can also lead to what is known as data explosion.

Other MOLAP tools, particularly those that implement the functional database model do not pre-compute derived data but make all calculations on demand other than those that were previously requested and stored in a cache.

Advantages of MOLAP

- Fast query performance due to optimized storage, multidimensional indexing and caching.
- Smaller on-disk size of data compared to data stored in relational database due to compression techniques.
- Automated computation of higher-level aggregates of the data.
- It is very compact for low dimension data sets.
- Array models provide natural indexing.
- Effective data extraction achieved through the pre-structuring of aggregated data.

Disadvantages of MOLAP

- Within some MOLAP systems the processing step (data load) can be quite lengthy, especially on large data volumes. This is usually remedied by doing only incremental processing, i.e., processing only the data which have changed (usually new data) instead of reprocessing the entire data set.
- Some MOLAP methodologies introduce data redundancy.

Products

Examples of commercial products that use MOLAP are Cognos Powerplay, Oracle Database OLAP Option, MicroStrategy, Microsoft Analysis Services, Essbase, TM1, Jedox and icCube.

Relational OLAP (ROLAP)

ROLAP works directly with relational databases and does not require pre-computation. The base data and the dimension tables are stored as relational tables and new tables are created to hold the aggregated information. It depends on a specialized schema design. This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement. ROLAP tools do not use pre-calculated data cubes but instead pose the query to the standard relational database and its tables in order to bring back the data required to answer the question. ROLAP tools feature the ability to ask any question because the methodology does not limit to the contents of a cube. ROLAP also has the ability to drill down to the lowest level of detail in the database.

While ROLAP uses a relational database source, generally the database must be carefully designed for ROLAP use. A database which was designed for OLTP will not function well as a ROLAP database. Therefore, ROLAP still involves creating an additional copy of the data. However, since it is a database, a variety of technologies can be used to populate the database.

Advantages of ROLAP

- ROLAP is considered to be more scalable in handling large data volumes, especially models with dimensions with very high cardinality (i.e., millions of members).

- With a variety of data loading tools available, and the ability to fine-tune the extract, transform, load (ETL) code to the particular data model, load times are generally much shorter than with the automated MOLAP loads.

- The data are stored in a standard relational database and can be accessed by any SQL reporting tool (the tool does not have to be an OLAP tool).

- ROLAP tools are better at handling *non-aggregatable facts* (e.g., textual descriptions). MOLAP tools tend to suffer from slow performance when querying these elements.

- By decoupling the data storage from the multi-dimensional model, it is possible to successfully model data that would not otherwise fit into a strict dimensional model.

- The ROLAP approach can leverage database authorization controls such as row-level security, whereby the query results are filtered depending on preset criteria applied, for example, to a given user or group of users (SQL WHERE clause).

Disadvantages of ROLAP

- There is a consensus in the industry that ROLAP tools have slower performance than MOLAP tools. However, see the discussion below about ROLAP performance.

- The loading of *aggregate tables* must be managed by custom ETL code. The ROLAP tools do not help with this task. This means additional development time and more code to support.

- When the step of creating aggregate tables is skipped, the query performance then suffers because the larger detailed tables must be queried. This can be partially remedied by adding additional aggregate tables, however it is still not practical to create aggregate tables for all combinations of dimensions/attributes.

- ROLAP relies on the general purpose database for querying and caching, and therefore several special techniques employed by MOLAP tools are not available (such as special hierarchical indexing). However, modern ROLAP tools take advantage of latest improvements in SQL language such as CUBE and ROLLUP operators, DB2 Cube Views, as well as other SQL OLAP extensions. These SQL improvements can mitigate the benefits of the MOLAP tools.

- Since ROLAP tools rely on SQL for all of the computations, they are not suitable when the model is heavy on calculations which don't translate well into SQL. Examples of such models include budgeting, allocations, financial reporting and other scenarios.

Performance of ROLAP

In the OLAP industry ROLAP is usually perceived as being able to scale for large data volumes, but suffering from slower query performance as opposed to MOLAP. The OLAP Survey, the largest independent survey across all major OLAP products, being conducted for 6 years (2001 to 2006) have consistently found that companies using ROLAP report slower performance than those using MOLAP even when data volumes were taken into consideration.

However, as with any survey there are a number of subtle issues that must be taken into account when interpreting the results.

- The survey shows that ROLAP tools have 7 times more users than MOLAP tools within each company. Systems with more users will tend to suffer more performance problems at peak usage times.

- There is also a question about complexity of the model, measured both in number of dimensions and richness of calculations. The survey does not offer a good way to control for these variations in the data being analyzed.

Downside of flexibility

Some companies select ROLAP because they intend to re-use existing relational database tables—these tables will frequently not be optimally designed for OLAP use. The superior flexibility of ROLAP tools allows this less than optimal design to work, but performance suffers. MOLAP tools in contrast would force the data to be re-loaded into an optimal OLAP design.

Hybrid OLAP (HOLAP)

The undesirable trade-off between additional ETL cost and slow query performance has ensured that most commercial OLAP tools now use a "Hybrid OLAP" (HOLAP) approach, which allows the model designer to decide which portion of the data will be stored in MOLAP and which portion in ROLAP.

There is no clear agreement across the industry as to what constitutes "Hybrid OLAP", except that a database will divide data between relational and specialized storage. For example, for some vendors, a HOLAP database will use relational tables to hold the larger quantities of detailed data, and use specialized storage for at least some aspects of the smaller quantities of more-aggregate or less-detailed data. HOLAP addresses the shortcomings of MOLAP and ROLAP by combining the capabilities of both approaches. HOLAP tools can utilize both pre-calculated cubes and relational data sources.

Vertical partitioning

In this mode HOLAP stores *aggregations* in MOLAP for fast query performance, and detailed data in ROLAP to optimize time of cube *processing*.

Horizontal partitioning

In this mode HOLAP stores some slice of data, usually the more recent one (i.e. sliced by Time dimension) in MOLAP for fast query performance, and older data in ROLAP. Moreover, we can store some dices in MOLAP and others in ROLAP, leveraging the fact that in a large cuboid, there will be dense and sparse subregions.

Products

The first product to provide HOLAP storage was Holos, but the technology also became available in other commercial products such as Microsoft Analysis Services, Oracle Database OLAP Option, MicroStrategy and SAP AG BI Accelerator. The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may store large volumes of detailed data in a relational database, while aggregations are kept in a separate MOLAP store. The Microsoft SQL Server 7.0 OLAP Services supports a hybrid OLAP server

Comparison

Each type has certain benefits, although there is disagreement about the specifics of the benefits between providers.

- Some MOLAP implementations are prone to database explosion, a phenomenon causing vast amounts of storage space to be used by MOLAP databases when certain common conditions are met: high number of dimensions, pre-calculated results and sparse multidimensional data.

- MOLAP generally delivers better performance due to specialized indexing and storage optimizations. MOLAP also needs less storage space compared to ROLAP because the specialized storage typically includes compression techniques.

- ROLAP is generally more scalable. However, large volume pre-processing is difficult to implement efficiently so it is frequently skipped. ROLAP query performance can therefore suffer tremendously.

- Since ROLAP relies more on the database to perform calculations, it has more limitations in the specialized functions it can use.

- HOLAP attempts to mix the best of ROLAP and MOLAP. It can generally pre-process swiftly, scale well, and offer good function support.

Other types

The following acronyms are also sometimes used, although they are not as widespread as the ones above:

- **WOLAP** – Web-based OLAP

- **DOLAP** – Desktop OLAP

- **RTOLAP** – Real-Time OLAP

- **GOLAP** – Graph OLAP

- **CaseOLAP** – Context-aware Semantic OLAP, developed for biomedical applications. The CaseOLAP platform includes data preprocessing (e.g., downloading, extraction, and parsing text documents), indexing and searching with Elasticsearch, creating a functional document structure called Text-Cube, and quantifying user-defined phrase-category relationships using the core CaseOLAP algorithm.

APIs and Query Languages

Unlike relational databases, which had SQL as the standard query language, and widespread APIs such as ODBC, JDBC and OLEDB, there was no such unification in the OLAP world for a long time. The first real standard API was OLE DB for OLAP specification from Microsoft which appeared in 1997 and introduced the MDX query language. Several OLAP vendors – both server and client – adopted it. In 2001 Microsoft and Hyperion announced the XML for Analysis specification, which was endorsed by most of the OLAP vendors. Since this also used MDX as a query language, MDX became the de facto standard. Since September-2011 LINQ can be used to query SSAS OLAP cubes from Microsoft .NET.

Products

History

The first product that performed OLAP queries was *Express,* which was released in 1970 (and acquired by Oracle in 1995 from Information Resources). However, the term did not appear until 1993 when it was coined by Edgar F. Codd, who has been described as "the father of the relational database". Codd's paper resulted from a short consulting assignment which Codd undertook for former Arbor Software (later Hyperion Solutions, and in 2007 acquired by Oracle), as a sort of marketing coup. The company had released its own OLAP product, *Essbase*, a year earlier. As a result, Codd's "twelve laws of online analytical processing" were explicit in their reference to Essbase. There was some ensuing controversy and when Computerworld learned that Codd was paid by Arbor, it retracted the article. OLAP market experienced strong growth in late 1990s with dozens of commercial products going into market. In 1998, Microsoft released its first OLAP Server – Microsoft Analysis Services, which drove wide adoption of OLAP technology and moved it into mainstream.

Product comparison

OLAP clients

OLAP clients include many spreadsheet programs like Excel, web application, SQL, dashboard tools, etc. Many clients support interactive data exploration where users select dimensions and measures of interest. Some dimensions are used as filters (for slicing and dicing the data) while others are selected as the axes of a pivot table or pivot chart. Users can also vary aggregation level (for drilling-down or rolling-up) the displayed view. Clients can also offer a variety of graphical widgets such as sliders, geographic maps, heatmaps and more which can be grouped and coordinated as dashboards.

Open-source

- **Mondrian OLAP server** is an open-source OLAP server written in Java. It supports the MDX query language, the XML for Analysis and the olap4j interface specifications.

- **Druid (open-source data store)** is a popular open-source distributed data store for OLAP queries that is used at scale in production by various organizations.

- **Apache Kylin** is a distributed data store for OLAP queries originally developed by eBay.

- **Cubes (OLAP server)** is another light-weight open-source toolkit implementation of OLAP functionality in the Python programming language with built-in ROLAP.

- **Apache Pinot (incubating)** is used at LinkedIn, Uber, Slack and Microsoft to deliver scalable real time analytics with low latency. It can ingest data from offline data sources (such as Hadoop and flat files) as well as online sources (such as Kafka). Pinot is designed to scale horizontally.

OLAP Cube

An **OLAP cube** is a multi-dimensional array of data. Online Analytical Processing (OLAP) is a computer-based technique of analyzing data to look for insights. The term *cube* here refers to a multi-dimensional dataset, which is also sometimes called a hypercube if the number of dimensions is greater than 3.

Terminology

A cube can be considered a multi-dimensional generalization of a two- or three-dimensional spreadsheet. For example, a company might wish to summarize financial data by product, by time-period, and by city to compare actual and budget expenses. Product, time, city and scenario (actual and budget) are the data's dimensions.

Cube is a shorthand for *multidimensional dataset*, given that data can have an arbitrary number of *dimensions*. The term hypercube is sometimes used, especially for data with more than three dimensions. A cube is not a "cube" in the strict mathematical sense, as all the sides are not necessarily equal. But this term is used widely.

Slice is a term for a dimension which is held constant for all cells so that multi-dimensional information can be shown in a two-dimensional physical space of a spreadsheet or pivot table.

Each cell of the cube holds a number that represents some *measure* of the business, such as sales, profits, expenses, budget and forecast.

OLAP data is typically stored in a star schema or snowflake schema in a relational data warehouse or in a special-purpose data management system. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

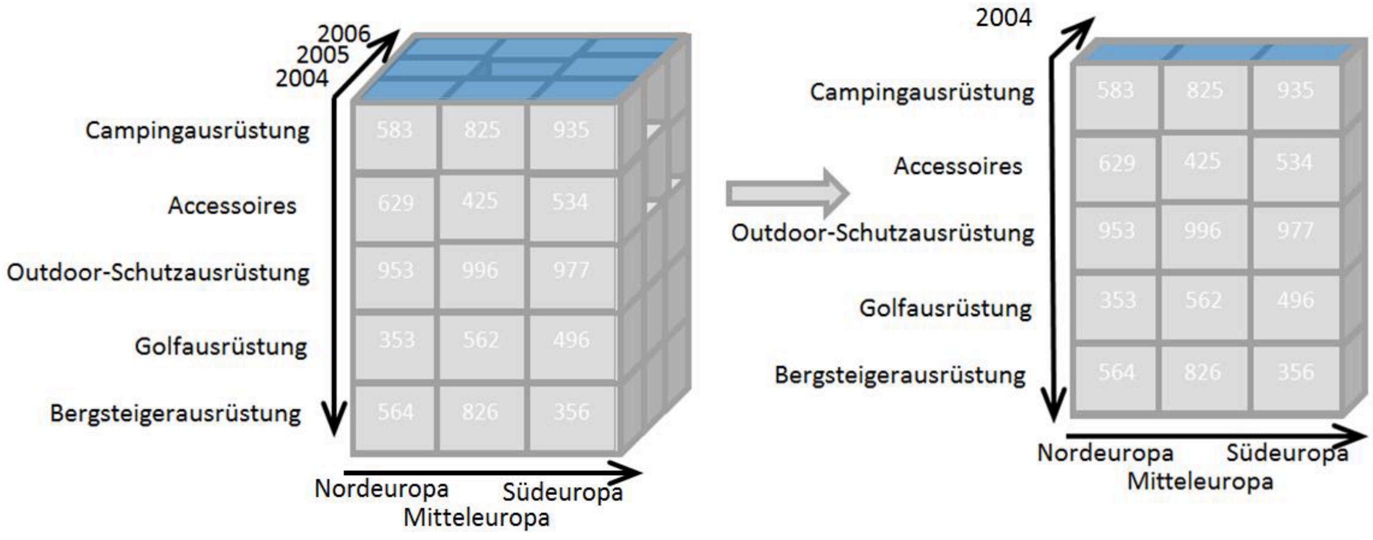
Hierarchy

The elements of a dimension can be organized as a hierarchy, a set of parent-child relationships, typically where a parent member summarizes its children. Parent elements can further be aggregated as the children of another parent.

For example, May 2005's parent is Second Quarter 2005 which is in turn the child of Year 2005. Similarly, cities are the children of regions; products roll into product groups and individual expense items into types of expenditure.

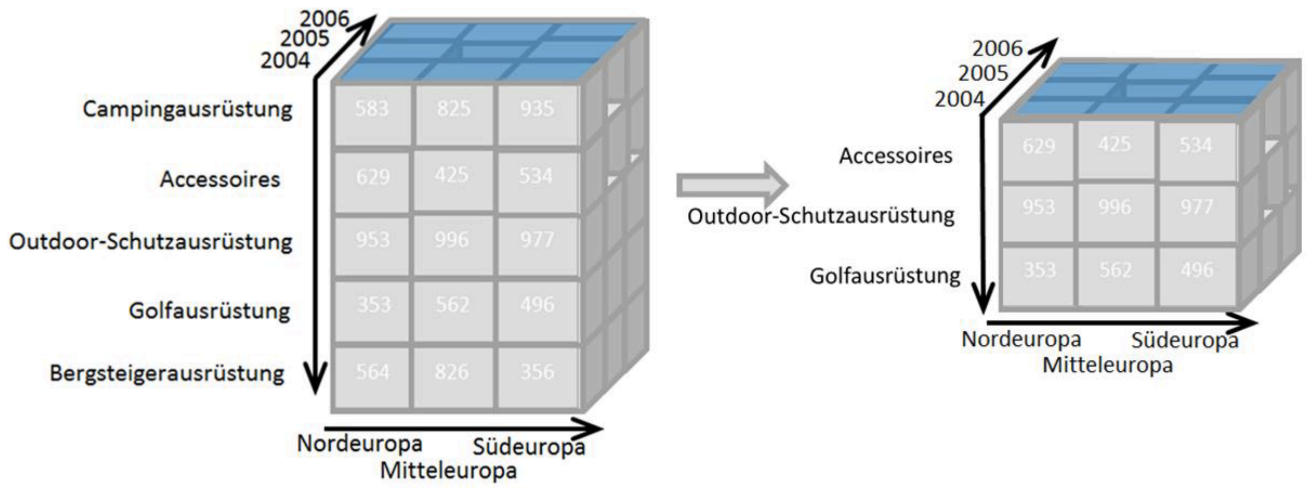
Operations

Conceiving data as a cube with hierarchical dimensions leads to conceptually straightforward operations to facilitate analysis. Aligning the data content with a familiar visualization enhances analyst learning and productivity. The user-initiated process of navigating by calling for page displays interactively, through the specification of slices via rotations and drill down/up is sometimes called "slice and dice". Common operations include slice and dice, drill down, roll up, and pivot.



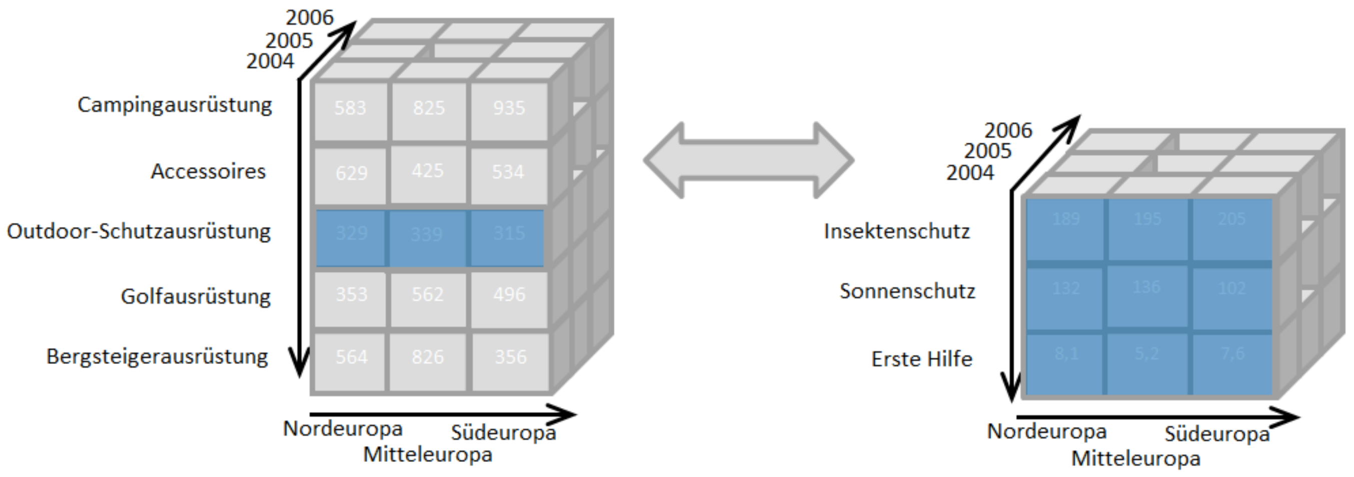
OLAP slicing

Slice is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension. The picture shows a slicing operation: The sales figures of all sales regions and all product categories of the company in the year 2005 and 2006 are "sliced" out of the data cube.



OLAP dicing

Dice: The dice operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions. The picture shows a dicing operation: The new cube shows the sales figures of a limited number of product categories, the time and region dimensions cover the same range as before.

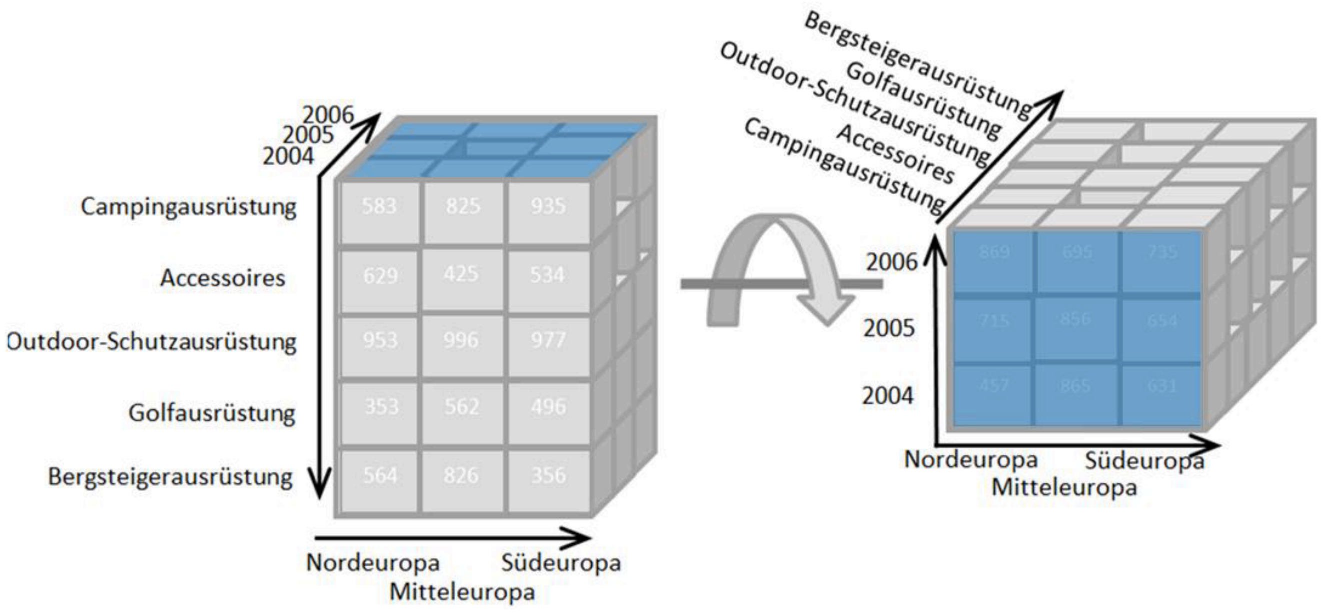


OLAP Drill-up and drill-down

Drill Down/Up allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down). The picture shows a drill-down operation: The analyst moves from the summary category "Outdoor-Schutzausrüstung" to see the sales figures for the individual products.

OLAP Roll-up

Roll-up: A roll-up involves summarizing the data along a dimension. The summarization rule might be an aggregate function, such as computing totals along a hierarchy or applying a set of formulas such as "profit = sales - expenses". General aggregation functions may be costly to compute when rolling up: if they cannot be determined from the cells of the cube, they must be computed from the base data, either computing them online (slow) or precomputing them for possible rollouts (large space). Aggregation functions that can be determined from the cells are known as decomposable aggregation functions, and allow efficient computation. For example, it is easy to support COUNT, MAX, MIN, and SUM in OLAP, since these can be computed for each cell of the OLAP cube and then rolled up, since on overall sum (or count etc.) is the sum of sub-sums, but it is difficult to support MEDIAN, as that must be computed for every view separately: the median of a set is not the median of medians of subsets.



OLAP pivoting

Pivot allows an analyst to rotate the cube in space to see its various faces. For example, cities could be arranged vertically and products horizontally while viewing data for a particular quarter. Pivoting could replace products with time periods to see data across time for a single product.

The picture shows a pivoting operation: The whole cube is rotated, giving another perspective on the data.

Star Schema

In computing, the **star schema** is the simplest style of data mart schema and is the approach most widely used to develop data warehouses and dimensional data marts. The star schema consists of one or more fact tables referencing any number of dimension tables. The star schema is an important special case of the snowflake schema, and is more effective for handling simpler queries.

The star schema gets its name from the physical model's resemblance to a star shape with a fact table at its center and the dimension tables surrounding it representing the star's points.

Model

The star schema separates business process data into facts, which hold the measurable, quantitative data about a business, and dimensions which are descriptive attributes related to fact data. Examples of fact data include sales price, sale quantity, and time, distance, speed and weight measurements. Related dimension attribute examples include product models, product colors, product sizes, geographic locations, and salesperson names.

A star schema that has many dimensions is sometimes called a *centipede schema*. Having dimensions of only a few attributes, while simpler to maintain, results in queries with many table joins and makes the star schema less easy to use.

Fact Tables

Fact tables record measurements or metrics for a specific event. Fact tables generally consist of numeric values, and foreign keys to dimensional data where descriptive information is kept. Fact tables are designed to a low level of uniform detail (referred to as "granularity" or "grain"), meaning facts can record events at a very atomic level. This can result in the accumulation of a large number of records in a fact table over time. Fact tables are defined as one of three types:

- Transaction fact tables record facts about a specific event (e.g., sales events)
- Snapshot fact tables record facts at a given point in time (e.g., account details at month end)
- Accumulating snapshot tables record aggregate facts at a given point in time (e.g., total month-to-date sales for a product)

Fact tables are generally assigned a surrogate key to ensure each row can be uniquely identified. This key is a simple primary key.

Dimension Tables

Dimension tables usually have a relatively small number of records compared to fact tables, but each record may have a very large number of attributes to describe the fact data. Dimensions can define a wide variety of characteristics, but some of the most common attributes defined by dimension tables include:

- Time dimension tables describe time at the lowest level of time granularity for which events are recorded in the star schema

- Geography dimension tables describe location data, such as country, state, or city
- Product dimension tables describe products
- Employee dimension tables describe employees, such as sales people
- Range dimension tables describe ranges of time, dollar values or other measurable quantities to simplify reporting

Dimension tables are generally assigned a surrogate primary key, usually a single-column integer data type, mapped to the combination of dimension attributes that form the natural key.

Benefits

Star schemas are denormalized, meaning the normal rules of normalization applied to transactional relational databases are relaxed during star schema design and implementation. The benefits of star schema denormalization are:

- Simpler queries – star schema join logic is generally simpler than the join logic required to retrieve data from a highly normalized transactional schema.
- Simplified business reporting logic – when compared to highly normalized schemas, the star schema simplifies common business reporting logic, such as period-over-period and as-of reporting.

- Query performance gains – star schemas can provide performance enhancements for read-only reporting applications when compared to highly normalized schemas.

- Fast aggregations – the simpler queries against a star schema can result in improved performance for aggregation operations.

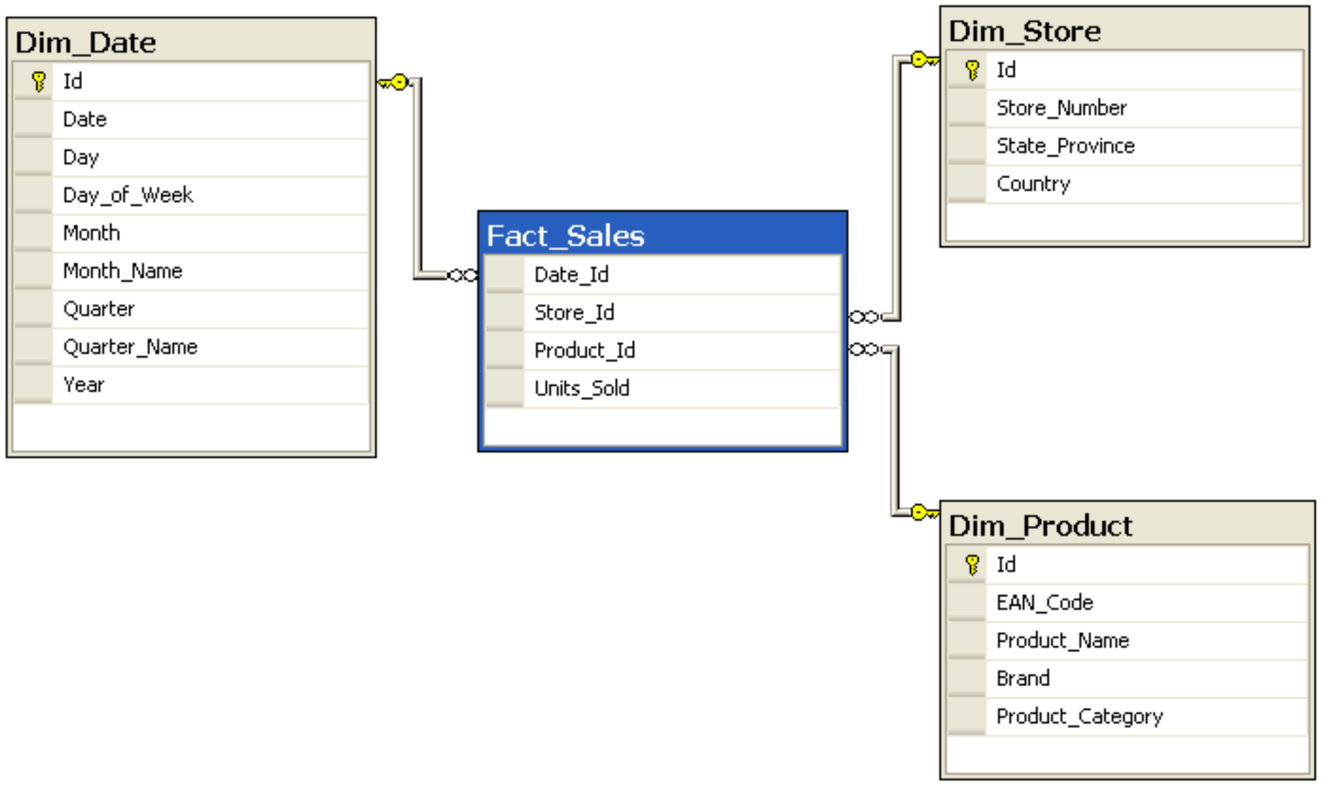
- Feeding cubes – star schemas are used by all OLAP systems to build proprietary OLAP cubes efficiently; in fact, most major OLAP systems provide a ROLAP mode of operation which can use a star schema directly as a source without building a proprietary cube structure.

Disadvantages

The main disadvantage of the star schema is that data integrity is not enforced well since it is in a highly de-normalized state. One-off inserts and updates can result in data anomalies which normalized schemas are designed to avoid. Generally speaking, star schemas are loaded in a highly controlled fashion via batch processing or near-real time "trickle feeds", to compensate for the lack of protection afforded by normalization.

Star schema is also not as flexible in terms of analytical needs as a normalized data model. Normalized models allow any kind of analytical queries to be executed as long as they follow the business logic defined in the model. Star schemas tend to be more purpose-built for a particular view of the data, thus not really allowing more complex analytics. Star schemas don't support many-to-many relationships between business entities - at least not very naturally. Typically, these relationships are simplified in star schema to conform to the simple dimensional model.

Example



Consider a database of sales, perhaps from a store chain, classified by date, store and product. The image of the schema below is a star schema version of the sample schema provided in the snowflake schema section.

Fact_Sales is the fact table and there are three-dimension tables **Dim_Date**, **Dim_Store** and **Dim_Product**.

Each dimension table has a primary key on its Id column, relating to one of the columns (viewed as rows in the example schema) of the **Fact_Sales** table's three-column (compound) primary key (**Date_Id**, **Store_Id**, **Product_Id**). The non-primary key **Units_Sold** column of the fact table in this example represents a measure or metric that can be used in calculations and analysis. The non-primary key columns of the dimension tables represent additional attributes of the dimensions (such as the Year of the **Dim_Datedimension**).

For example, the following query answers how many TV sets have been sold, for each brand and country, in 1997:

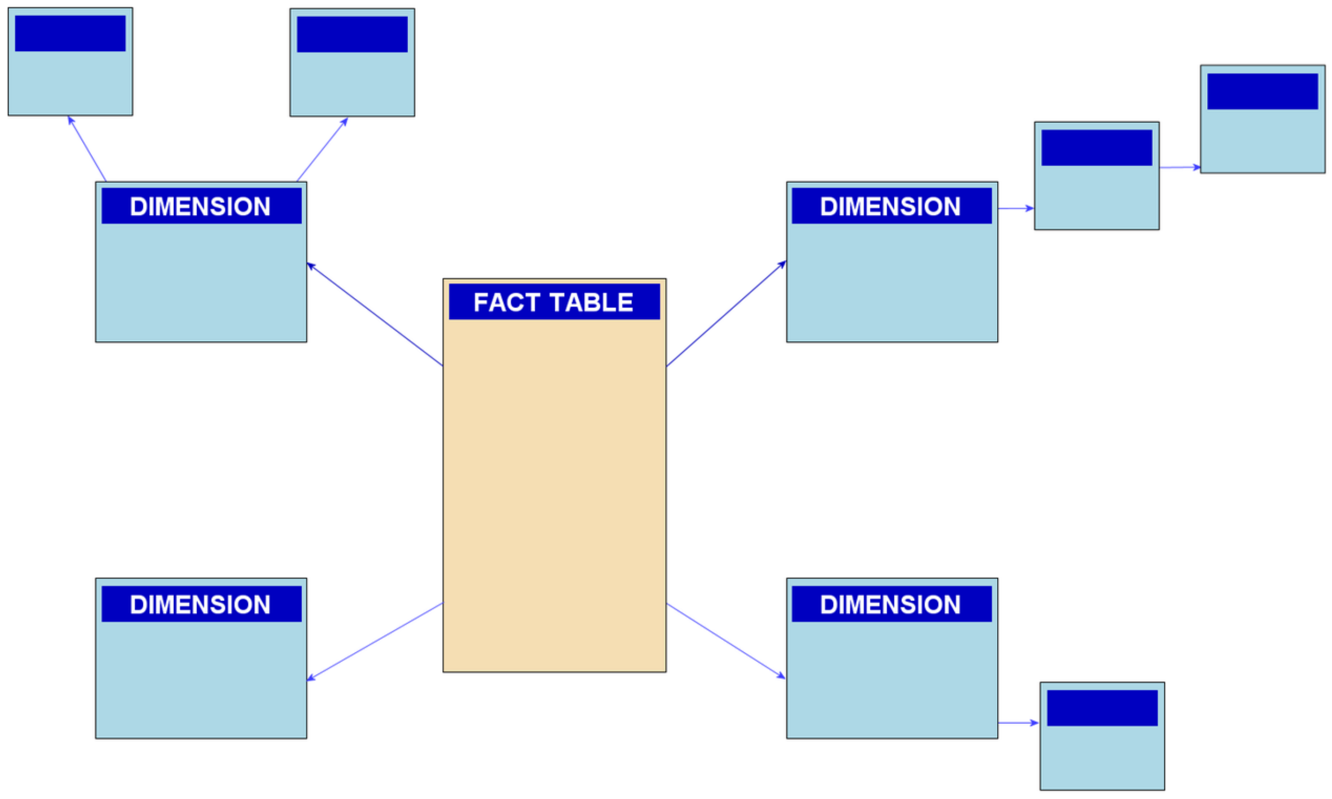
```
SELECT
      P.Brand,
      S.Country AS Countries,
      SUM(F.Units_Sold)

FROM Fact_Sales F
INNER JOIN Dim_Date D    ON (F.Date_Id = D.Id)
INNER JOIN Dim_Store S   ON (F.Store_Id = S.Id)
INNER JOIN Dim_Product P ON (F.Product_Id = P.Id)

WHERE D.Year = 1997 AND  P.Product_Category = 'tv'

GROUP BY
      P.Brand,
      S.Country
```

Snowflake Schema



In computing, a **snowflake schema** is a logical arrangement of tables in a multidimensional database such that the entity relationship diagram resembles a snowflake shape. The snowflake schema is represented by centralized fact tables which are connected to multiple dimensions. "Snowflaking" is a method of normalizing the dimension tables in a star schema. When it is completely normalized along all the dimension tables, the resultant structure resembles a snowflake with the fact table in the middle. The principle behind snowflaking is normalization of the dimension tables by removing low cardinality attributes and forming separate tables.

The snowflake schema is similar to the star schema. However, in the snowflake schema, dimensions are normalized into multiple related tables, whereas the star schema's dimensions are denormalized with each dimension represented by a single table. A complex snowflake shape emerges when the dimensions of a snowflake schema are elaborate, having multiple levels of relationships, and the child tables have multiple parent tables ("forks in the road").

Common Uses

Star and snowflake schemas are most commonly found in dimensional data warehouses and data marts where speed of data retrieval is more important than the efficiency of data manipulations. As such, the tables in these schemas are not normalized much, and are frequently designed at a level of normalization short of third normal form.

Data Normalization and Storage

Normalization splits up data to avoid redundancy (duplication) by moving commonly repeating groups of data into new tables. Normalization therefore tends to increase the number of tables that need to be joined in order to perform a given query, but reduces the space required to hold the data and the number of places where it needs to be updated if the data changes.

From a space storage point of view, dimensional tables are typically small compared to fact tables. This often negates the potential storage-space benefits of the star schema as compared to the snowflake schema. Example: One million sales transactions in 300 shops in 220 countries would result in 1,000,300 records in a star schema (1,000,000 records in the fact table and 300 records in the dimensional table where each country would be listed explicitly for each shop in that country). A more normalized snowflake schema with country keys referring to a country table would consist of the same 1,000,000 record fact table, a 300 record shop table with references to a country table with 220 records. In this case, the star schema, although further denormalized, would only reduce the number or records by a (negligible) factor of ~0.9998 (=[1,000,000+300] divided by [1,000,000+300+220])

Some database developers compromise by creating an underlying snowflake schema with views built on top of it that perform many of the necessary joins to simulate a star schema. This provides the storage benefits achieved through the normalization of dimensions with the ease of querying that the star schema provides. The tradeoff is that requiring the server to perform the underlying joins automatically can result in a performance hit when querying as well as extra joins to tables that may not be necessary to fulfill certain queries.

Benefits

The snowflake schema is in the same family as the star schema logical model. In fact, the star schema is considered a special case of the snowflake schema. The snowflake schema provides some advantages over the star schema in certain situations, including:

- Some OLAP multidimensional database modeling tools are optimized for snowflake schemas.

- Normalizing attributes results in storage savings, the tradeoff being additional complexity in source query joins.

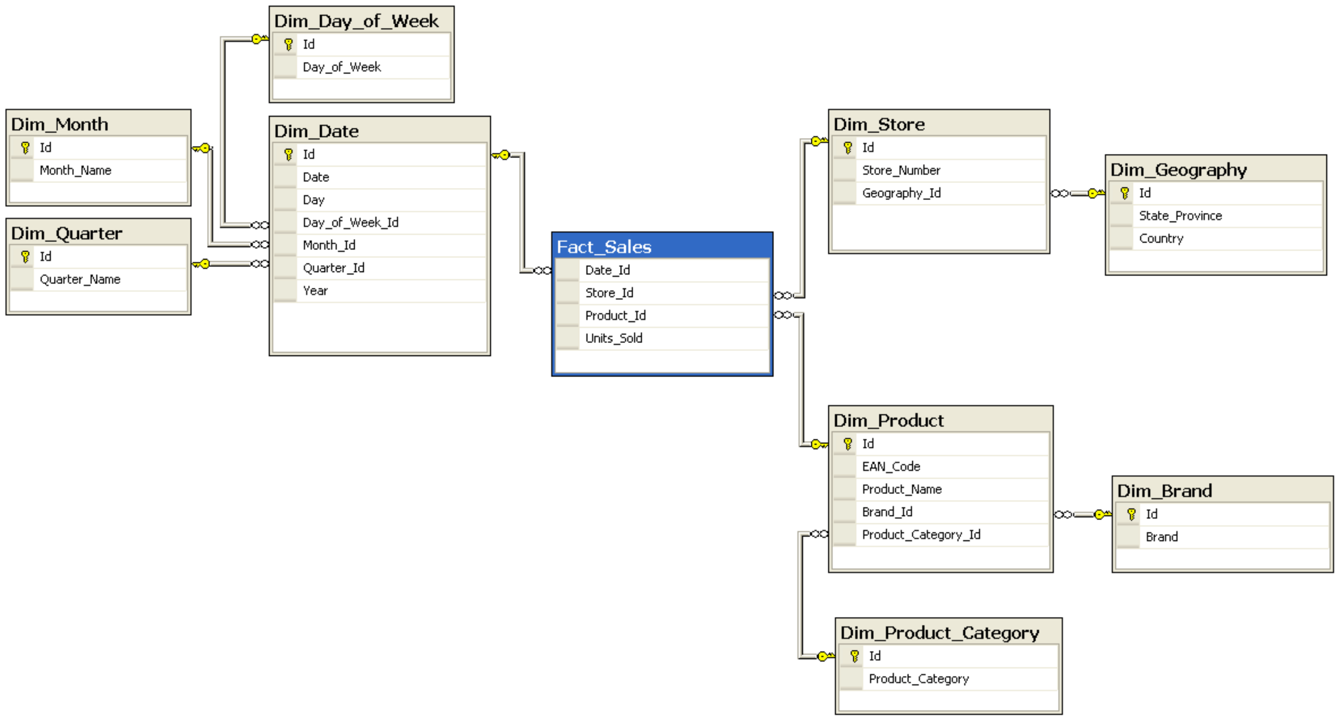
Disadvantages

The primary disadvantage of the snowflake schema is that the additional levels of attribute normalization adds complexity to source query joins, when compared to the star schema.

Snowflake schemas, in contrast to flat single table dimensions, have been heavily criticized. Their goal is assumed to be an efficient and compact storage of normalized data but this is at the significant cost of poor performance when browsing the joins required in this dimension. This disadvantage may have reduced in the years since it was first recognized, owing to better query performance within the browsing tools.

When compared to a highly normalized transactional schema, the snowflake schema's denormalization removes the data integrity assurances provided by normalized schemas. Data loads into the snowflake schema must be highly controlled and managed to avoid update and insert anomalies.

Example



The example schema shown below is a snowflaked version of the star schema example provided in the star schema section.

The following example query is the snowflake schema equivalent of the star schema example code which returns the total number of units sold by brand and by country for 1997. Notice that the snowflake schema query requires many more joins than the star schema version in order to fulfill even a simple query. The benefit of using the snowflake schema in this example is that the storage requirements are lower since the snowflake schema eliminates many duplicate values from the dimensions themselves.

```
SELECT
      B.Brand,
      G.Country,
      SUM(F.Units_Sold)
FROM Fact_Sales F
INNER JOIN Dim_Date D                ON F.Date_Id = D.Id
INNER JOIN Dim_Store S               ON F.Store_Id = S.Id
INNER JOIN Dim_Geography G           ON S.Geography_Id = G.Id
INNER JOIN Dim_Product P             ON F.Product_Id = P.Id
INNER JOIN Dim_Brand B               ON P.Brand_Id = B.Id
INNER JOIN Dim_Product_Category C ON P.Product_Category_Id
= C.Id
WHERE
      D.Year = 1997 AND
      C.Product_Category = 'tv'
GROUP BY
      B.Brand,
      G.Country
```

Fact Constellation

Fact constellation is a measure of online analytical processing, which is a collection of multiple fact tables sharing dimension tables, viewed as a collection of stars. It can be seen as an extension of the star schema.

A fact constellation schema has multiple fact tables. It is also known as galaxy schema. It is widely used schema and more complex than star schema and snowflake schema. It is possible to create fact constellation schema by splitting original star schema into more star schema. It has many fact tables and some common dimension table.

Dimension

A **dimension** is a structure that categorizes facts and measures in order to enable users to answer business questions. Commonly used dimensions are people, products, place and time (Note: People and time sometimes are not modeled as dimensions).

In a data warehouse, dimensions provide structured labeling information to otherwise unordered numeric measures. The dimension is a data set composed of individual, non-overlapping data elements. The primary functions of dimensions are threefold: to provide filtering, grouping and labelling.

These functions are often described as "slice and dice". A common data warehouse example involves sales as the measure, with customer and product as dimensions. In each sale a customer buys a product. The data can be sliced by removing all customers except for a group under study, and then diced by grouping by product.

A dimensional data element is similar to a categorical variable in statistics.

Typically dimensions in a data warehouse are organized internally into one or more hierarchies. "Date" is a common dimension, with several possible hierarchies:

- "Days (are grouped into) Months (which are grouped into) Years",
- "Days (are grouped into) Weeks (which are grouped into) Years"
- "Days (are grouped into) Months (which are grouped into) Quarters (which are grouped into) Years"

- Etc.

Types

Conformed Dimension

A conformed dimension is a set of data attributes that have been physically referenced in multiple database tables using the same key value to refer to the same structure, attributes, domain values, definitions and concepts. A conformed dimension cuts across many facts.

Dimensions are conformed when they are either exactly the same (including keys) or one is a perfect subset of the other. Most important, the row headers produced in two different answer sets from the same conformed dimension(s) must be able to match perfectly.'

Conformed dimensions are either identical or strict mathematical subsets of the most granular, detailed dimension. Dimension tables are not conformed if the attributes are labeled differently or contain different values. Conformed dimensions come in several different flavors. At the most basic level, conformed dimensions mean exactly the same thing with every possible fact table to which they are joined. The date dimension table connected to the sales facts is identical to the date dimension connected to the inventory facts.

Junk Dimension

A junk dimension is a convenient grouping of typically low-cardinality flags and indicators. By creating an abstract dimension, these flags and indicators are removed from the fact table while placing them into a useful dimensional framework. A Junk Dimension is a dimension table consisting of attributes that do not belong in the fact table or in any of the existing dimension tables. The nature of these attributes is usually text or various flags, e.g. non-generic comments or just simple yes/no or true/false indicators. These kinds of attributes are typically remaining when all the obvious dimensions in the business process have been identified and thus the designer is faced with the challenge of where to put these attributes that do not belong in the other dimensions.

One solution is to create a new dimension for each of the remaining attributes, but due to their nature, it could be necessary to create a vast number of new dimensions resulting in a fact table with a very large number of foreign keys. The designer could also decide to leave the remaining attributes in the fact table but this could make the row length of the table unnecessarily large if, for example, the attributes is a long text string.

The solution to this challenge is to identify all the attributes and then put them into one or several Junk Dimensions. One Junk Dimension can hold several true/false or yes/no indicators that have no correlation with each other, so it would be convenient to convert the indicators into a more describing attribute. An example would be an indicator about whether a package had arrived: instead of indicating this as "yes" or "no", it would be converted into "arrived" or "pending" in the junk dimension. The designer can choose to build the dimension table so it ends up holding all the indicators occurring with every other indicator so that all combinations are covered. This sets up a fixed size for the table itself which would be 2^x rows, where x is the number of indicators. This solution is appropriate in situations where the designer would expect to encounter a lot of different combinations and where the possible combinations are limited to an acceptable level. In a situation where the number of indicators is large, thus creating a very big table or where the designer only expect to encounter a few of the possible combinations, it would be more appropriate to build each row in the junk dimension as new combinations are encountered. To limit the size of the tables, multiple junk dimensions might be appropriate in other situations depending on the correlation between various indicators.

Junk dimensions are also appropriate for placing attributes like non-generic comments from the fact table. Such attributes might consist of data from an optional comment field when a customer places an order and as a result will probably be blank in many cases. Therefore, the junk dimension should contain a single row representing the blanks as a surrogate key that will be used in the fact table for every row returned with a blank comment field.

Degenerate Dimension

A degenerate dimension is a key, such as a transaction number, invoice number, ticket number, or bill-of-lading number, that has no attributes and hence does not join to an actual dimension table. Degenerate dimensions are very common when the grain of a fact table represents a single transaction item or line item because the degenerate dimension represents the unique identifier of the parent. Degenerate dimensions often play an integral role in the fact table's primary key.[6]

Role-Playing Dimension

Dimensions are often recycled for multiple applications within the same database. For instance, a "Date" dimension can be used for "Date of Sale", as well as "Date of Delivery", or "Date of Hire". This is often referred to as a "role-playing dimension".

Dimension Table

In data warehousing, a **dimension table** is one of the set of companion tables to a fact table.

The fact table contains business facts (or *measures*), and foreign keys which refer to candidate keys (normally primary keys) in the dimension tables.

Contrary to *fact* tables, *dimension* tables contain descriptive attributes (or fields) that are typically textual fields (or discrete numbers that behave like text). These attributes are designed to serve two critical purposes: query constraining and/or filtering, and query result set labeling.

Dimension attributes should be:

- Verbose (labels consisting of full words)
- Descriptive
- Complete (having no missing values)
- Discretely valued (having only one value per dimension table row)
- Quality assured (having no misspellings or impossible values)

Dimension table rows are uniquely identified by a single key field. It is recommended that the key field be a simple integer because a key value is meaningless, used only for joining fields between the fact and dimension tables. Dimension tables often use primary keys that are also surrogate keys. Surrogate keys are often auto-generated (e.g. a Sybase or SQL Server "identity column", a PostgreSQL or Informix serial, an Oracle SEQUENCE or a column defined with AUTO_INCREMENT in MySQL).

The use of surrogate dimension keys brings several advantages, including:

- Performance. Join processing is made much more efficient by using a single field (the surrogate key)
- Buffering from operational key management practices. This prevents situations where removed data rows might reappear when their natural keys get reused or reassigned after a long period of dormancy
- Mapping to integrate disparate sources
- Handling unknown or not-applicable connections
- Tracking changes in dimension attribute values

Although surrogate key use places a burden put on the ETL system, pipeline processing can be improved, and ETL tools have built-in improved surrogate key processing.

The goal of a dimension table is to create standardized, conformed dimensions that can be shared across the enterprise's data warehouse environment and enable joining to multiple fact tables representing various business processes.

Conformed dimensions are important to the enterprise nature of DW/BI systems because they promote:

- Consistency. Every fact table is filtered consistently, so that query answers are labeled consistently.
- Integration. Queries can drill into different process fact tables separately for each individual fact table, then join the results on common dimension attributes.
- Reduced development time to market. The common dimensions are available without recreating them.

Over time, the attributes of a given row in a dimension table may change. For example, the shipping address for a company may change. Kimball refers to this phenomenon as Slowly Changing Dimensions. Strategies for dealing with this kind of change are divided into three categories:

- Type One. Simply overwrite the old value(s).
- Type Two. Add a new row containing the new value(s) and distinguish between the rows using Tuple-versioning techniques.
- Type Three. Add a new attribute to the existing row.

Common Patterns

Date and Time

Since many fact tables in a data warehouse are time series of observations, one or more date dimensions are often needed. One of the reasons to have date dimensions is to place calendar knowledge in the data warehouse instead of hard-coded in an application. While a simple SQL date/timestamp is useful for providing accurate information about the time a fact was recorded, it can not give information about holidays, fiscal periods, etc. An SQL date/timestamp can still be useful to store in the fact table, as it allows for precise calculations.

Having both the date and time of day in the same dimension, may easily result in a huge dimension with millions of rows. If a high amount of detail is needed it is usually a good idea to split date and time into two or more separate dimensions. A time dimension with a grain of seconds in a day will only have 86400 rows. A more or less detailed grain for date/time dimensions can be chosen depending on needs. As examples, date dimensions can be accurate to year, quarter, month or day and time dimensions can be accurate to hours, minutes or seconds.

As a rule of thumb, time of day dimension should only be created if hierarchical groupings are needed or if there are meaningful textual descriptions for periods of time within the day (ex. "evening rush" or "first shift").

If the rows in a fact table are coming from several timezones, it might be useful to store date and time in both local time and a standard time. This can be done by having two dimensions for each date/time dimension needed – one for local time, and one for standard time. Storing date/time in both local and standard time, will allow for analysis on when facts are created in a local setting and in a global setting as well. The standard time chosen can be a global standard time (ex. UTC), it can be the local time of the business' headquarter, or any other time zone that would make sense to use.

Facts

A fact is a value, or measurement, which represents a fact about the managed entity or system.

Facts, as reported by the reporting entity, are said to be at raw level; e.g., in a mobile telephone system, if a BTS (base transceiver station) receives 1,000 requests for traffic channel allocation, allocates for 820, and rejects the remaining, it would report three **facts** or measurements to a management system:

- tch_req_total = 1000
- tch_req_success = 820
- tch_req_fail = 180

Facts at the raw level are further aggregated to higher levels in various dimensions to extract more service or business-relevant information from it. These are called aggregates or summaries or aggregated facts.

For instance, if there are three BTS in a city, then the facts above can be aggregated from the BTS to the city level in the network dimension. For example:

- tch_req_success_city = tch_req_success_bts1 + tch_req_success_bts2 + tch_req_success_bts3
- avg_tch_req_success_city = (tch_req_success_bts1 + tch_req_success_bts2 + tch_req_success_bts3) / 3

Fact Table

In data warehousing, a **fact table** consists of the measurements, metrics or facts of a business process. It is located at the center of a star schema or a snowflake schema surrounded by dimension tables. Where multiple fact tables are used, these are arranged as a fact constellation schema. A fact table typically has two types of columns: those that contain facts and those that are a foreign key to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys. Fact tables contain the content of the data warehouse and store different types of measures like additive, non-additive, and semi additive measures.

Fact tables provide the (usually) additive values that act as independent variables by which dimensional attributes are analyzed. Fact tables are often defined by their *grain*. The grain of a fact table represents the most atomic level by which the facts may be defined. The grain of a sales fact table might be stated as "sales volume by day by product by store". Each record in this fact table is therefore uniquely defined by a day, product and store. Other dimensions might be members of this fact table (such as location/region) but these add nothing to the uniqueness of the fact records. These "affiliate dimensions" allow for additional slices of the independent facts but generally provide insights at a higher level of aggregation (a region contains many stores).

Example

If the business process is sales, then the corresponding fact table will typically contain columns representing both raw facts and aggregations in rows such as:

- *$12,000*, being "sales for New York store for 15-Jan-2005".
- *$34,000*, being "sales for Los Angeles store for 15-Jan-2005"
- *$22,000*, being "sales for New York store for 16-Jan-2005"
- *$21,000*, being "average daily sales for Los Angeles Store for Jan-2005"
- *$65,000*, being "average daily sales for Los Angeles Store for Feb-2005"
- *$33,000*, being "average daily sales for Los Angeles Store for year 2005"

"Average daily sales" is a measurement that is stored in the fact table. The fact table also contains foreign keys from the dimension tables, where time series (e.g. dates) and other dimensions(e.g. store location, salesperson, product) are stored.

All foreign keys between fact and dimension tables should be surrogate keys, not reused keys from operational data.

Measure Types

- Additive - measures that can be added across any dimension.
- Non-additive - measures that cannot be added across any dimension.
- Semi-additive - measures that can be added across some dimensions.

A fact table might contain either detail level facts or facts that have been aggregated (fact tables that contain aggregated facts are often instead called summary tables).

Special care must be taken when handling ratios and percentage. One good design rule is to never store percentages or ratios in fact tables but only calculate these in the data access tool. Thus, only store the numerator and denominator in the fact table, which then can be aggregated and the aggregated stored values can then be used for calculating the ratio or percentage in the data access tool.

In the real world, it is possible to have a fact table that contains no measures or facts. These tables are called "factless fact tables", or "junction tables".

The *factless fact tables* can for example be used for modellings many-to-many relationships or capture events.

Types of Fact Tables

There are four fundamental measurement events, which characterize all fact tables.

Transactional

A transactional table is the most basic and fundamental. The grain associated with a transactional fact table is usually specified as "one row per line in a transaction", e.g., every line on a receipt. Typically a transactional fact table holds data of the most detailed level, causing it to have a great number of dimensions associated with it.

Periodic snapshots

The periodic snapshot, as the name implies, takes a "picture of the moment", where the moment could be any defined period of time, e.g. a performance summary of a salesman over the previous month. A periodic snapshot table is dependent on the transactional table, as it needs the detailed data held in the transactional fact table in order to deliver the chosen performance output.

Accumulating snapshots

This type of fact table is used to show the activity of a process that has a well-defined beginning and end, e.g., the processing of an order. An order moves through specific steps until it is fully processed. As steps towards fulfilling the order are completed, the associated row in the fact table is updated. An accumulating snapshot table often has multiple date columns, each representing a milestone in the process. Therefore, it's important to have an entry in the associated date dimension that represents an unknown date, as many of the milestone dates are unknown at the time of the creation of the row.

Temporal snapshots

By applying temporal database theory and modeling techniques the *temporal snapshot fact table* allows to have the equivalent of daily snapshots without really having daily snapshots. It introduces the concept of time Intervals into a fact table, allowing to save a lot of space, optimizing performances while allowing the end user to have the logical equivalent of the "picture of the moment" they are interested in.

Steps in Designing a Fact Table

- Identify a business process for analysis (like sales).

- Identify measures of facts (sales dollar), by asking questions like 'what number of X are relevant for the business process?', replacing the X with various options that make sense within the context of the business.

- Identify dimensions for facts (product dimension, location dimension, time dimension, organization dimension), by asking questions that make sense within the context of the business, like 'analyze by X', where X is replaced with the subject to test.

- List the columns that describe each dimension (region name, branch name, business unit name).

- Determine the lowest level (granularity) of summary in a fact table (e.g. sales dollars).

An alternative approach is the four-step design process described in Kimball: select the business process, declare the grain, identify the dimensions, identify the facts.

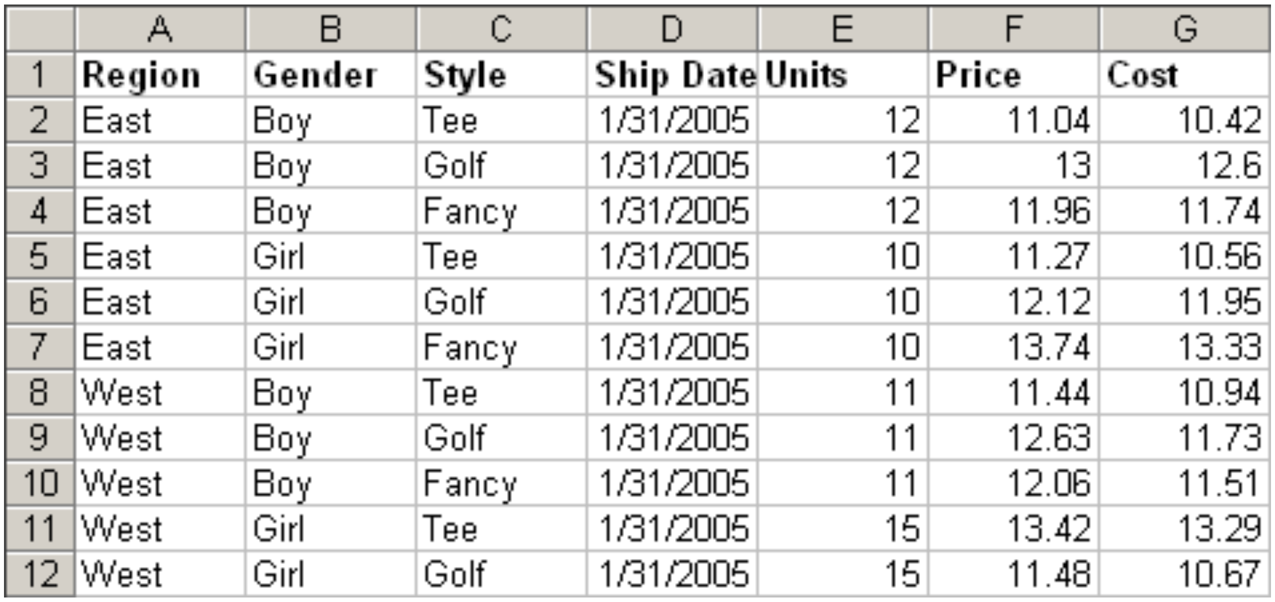
Pivot Table

A **pivot table** is a table of statistics that summarizes the data of a more extensive table (such as from a database, spreadsheet, or business intelligence program). This summary might include sums, averages, or other statistics, which the pivot table groups together in a meaningful way.

Pivot tables are a technique in data processing. They enable a person to arrange and rearrange (or "pivot") statistics in order to draw attention to useful information. Although *pivot table* is a generic term, Microsoft Corporation trademarked *PivotTable* in the United States in 1994.

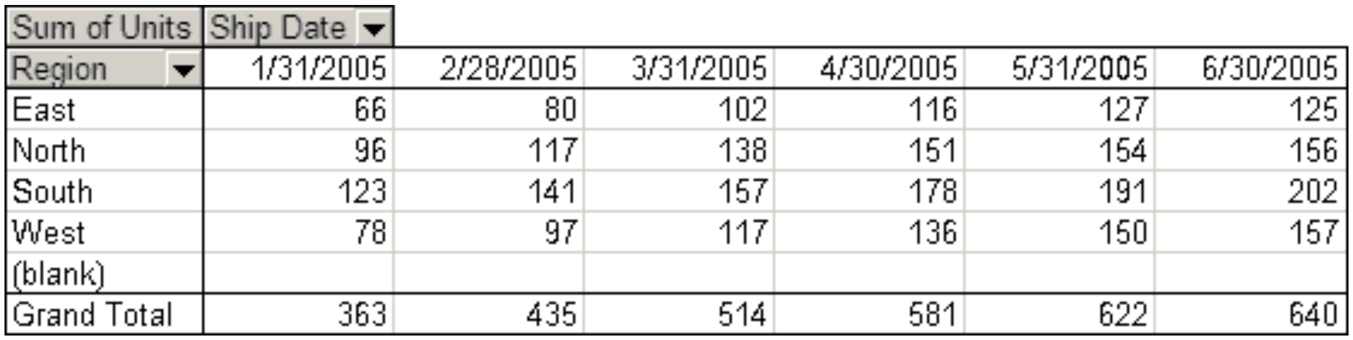
Mechanics

For typical data entry and storage, data usually appear in *flat* tables, meaning that they consist of only columns and rows, as in the following portion of a sample spreadsheet showing data on shirt types:



	A	B	C	D	E	F	G
1	**Region**	**Gender**	**Style**	**Ship Date**	**Units**	**Price**	**Cost**
2	East	Boy	Tee	1/31/2005	12	11.04	10.42
3	East	Boy	Golf	1/31/2005	12	13	12.6
4	East	Boy	Fancy	1/31/2005	12	11.96	11.74
5	East	Girl	Tee	1/31/2005	10	11.27	10.56
6	East	Girl	Golf	1/31/2005	10	12.12	11.95
7	East	Girl	Fancy	1/31/2005	10	13.74	13.33
8	West	Boy	Tee	1/31/2005	11	11.44	10.94
9	West	Boy	Golf	1/31/2005	11	12.63	11.73
10	West	Boy	Fancy	1/31/2005	11	12.06	11.51
11	West	Girl	Tee	1/31/2005	15	13.42	13.29
12	West	Girl	Golf	1/31/2005	15	11.48	10.67

While tables such as these can contain many data items, it can be difficult to get summarized information from them. A pivot table can help quickly summarize the data and highlight the desired information. The usage of a pivot table is extremely broad and depends on the situation. The first question to ask is, "What am I seeking?" In the example here, let us ask, "How many *Units* did we sell in each *Region* for every *Ship Date?*":



Sum of Units	Ship Date					
Region	1/31/2005	2/28/2005	3/31/2005	4/30/2005	5/31/2005	6/30/2005
East	66	80	102	116	127	125
North	96	117	138	151	154	156
South	123	141	157	178	191	202
West	78	97	117	136	150	157
(blank)						
Grand Total	363	435	514	581	622	640

A pivot table usually consists of *row*, *column* and *data* (or *fact*) fields. In this case, the column is *Ship Date*, the row is *Region* and the data we would like to see is (sum of) *Units*. These fields allow several kinds of aggregations, including: sum, average, standard deviation, count, etc. In this case, the total number of units shipped is displayed here using a *sum* aggregation.

Implementation

Using the example above, software will find all distinct values for *Region*. In this case, they are: *North*, *South*, *East*, *West*. Furthermore, it will find all distinct values for *Ship Date*. Based on the aggregation type, *sum*, it will summarize the fact, the quantities of *Unit*, and display them in a multidimensional chart. In the example above, the first datum is 66. This number was obtained by finding all records where both *Region* was *East* and *Ship Date* was *1/31/2005*, and adding the *Units* of that collection of records (*i.e.*, cells E2 to E7) together to get a final result.

Pivot tables are not created automatically. For example, in Microsoft Excel one must first select the entire data in the original table and then go to the Insert tab and select "Pivot Table" (or "Pivot Chart"). The user then has the option of either inserting the pivot table into an existing sheet or creating a new sheet to house the pivot table. A pivot table field list is provided to the user which lists all the column headers present in the data. For instance, if a table represents sales data of a company, it might include Date of sale, Sales person, Item sold, Color of item, Units sold, Per unit price, and Total price. This makes the data more readily accessible.

Date of sale	**Sales person**	**Item sold**	**Color of item**	**Units sold**	**Per unit price**	**Total price**
10/01/13	Jones	Notebook	Black	8	25000	200000
10/02/13	Prince	Laptop	Red	4	35000	140000
10/03/13	George	Mouse	Red	6	850	5100
10/04/13	Larry	Notebook	White	10	27000	270000

Date of sale	**Sales person**	**Item sold**	**Color of item**	**Units sold**	**Per unit price**	**Total price**
10/05/13	Jones	Mouse	Black	4	700	3200

The fields that would be created will be visible on the right-hand side of the worksheet. By default, the pivot table layout design will appear below this list.

Each of the fields from the list can be dragged on to this layout, which has four options:

1. Report filter
2. Column labels
3. Row labels
4. Summation values

Report Filter

Report filter is used to apply a filter to an entire table. For example, if the "Color of Item" field is dragged to this area, then the table constructed will have a report filter inserted above the table. This report filter will have drop-down options (Black, Red, and White in the example above). When an option is chosen from this drop-down list ("Black" in this example), then the table that would be visible will contain only the data from those rows that have the "Color of Item = Black".

Column Labels

Column labels are used to apply a filter to one or more columns that have to be shown in the pivot table. For instance, if the "Sales person" field is dragged to this area, then the table constructed will have values from the column "Sales Person", *i.e.*, one will have number of columns equal to the number of "Sales person". There will also be one added column of Total. In the example above, this instruction will create five columns in the table — one for each sales person, and Grand Total. There will be a filter above the data — column labels — from which one can select or deselect a particular sales person for the pivot table.

This table will not have any numerical values as no numerical field is selected but when it is selected, the values will automatically get updated in the column of "Grand total".

Row Labels

Row labels are used to apply a filter to one or more rows that have to be shown in the pivot table. For instance, if the "Sales person" field is dragged on this area then the other output table constructed will have values from the column "Sales person", *i.e.*, one will have number of rows equal to the number of "Sales Person". There will also be one added row of "Grand Total". In the example above, this instruction will create five rows in the table — one for each sales person, and Grand Total. There will be a filter above the data — row labels — from which one can select or deselect a particular sales person for the Pivot table.

This table will not have any numerical values, as no numerical field is selected, but when it is selected, the values will automatically get updated in the Row of "Grand Total".

Summation Values

This usually takes a field that has numerical values that can be used for different types of calculations. However, using text values would also not be wrong; instead of Sum it will give a count. So, in the example above, if the "Units sold" field is dragged to this area along with row label of "Sales person", then the instruction will add a new column, "Sum of units sold", which will have values against each sales person.

Row labels	Sum of units sold
Jones	12
Prince	4
George	6
Larry	10

Grand total	32

Application Support

Pivot tables or pivot functionality are an integral part of many spreadsheet applications and some database software, as well as being found in other data visualization tools and business intelligence packages.

Spreadsheets

- **Microsoft Excel** supports PivotTables, which can be visualized through PivotCharts.

- **LibreOffice Calc** supports pivot tables. Prior to version 3.4, this feature was named "DataPilot".

- **Google Sheets** natively supports pivot tables.

Database support

- **PostgreSQL**, an object-relational database management system, allows the creation of pivot tables using the *tablefunc* module.

- **MariaDB**, a MySQL fork, allows pivot tables using the CONNECT storage engine.

- **Microsoft Access** supports pivot queries under the name "crosstab" query.

- **Oracle Database** supports the PIVOT operation.

- Some popular databases that do not directly support pivot functionality, such as **Microsoft SQL Server** and **SQLite** can usually simulate pivot functionality using embedded functions, dynamic SQL or subqueries. The issue with pivoting in such cases is usually that the number of output columns must be known at the time the query starts to execute; for pivoting this is not possible as the number of columns is based on the data itself. Therefore, the names must be hard coded or the query to be executed must itself be created dynamically (meaning, prior to each use) based upon the data.

Web applications

- **ZK**, an Ajax framework, also allows the embedding of pivot tables in Web applications.

Programming languages and libraries

- Programming languages and libraries suited to work with tabular data contain functions that allow the creation and manipulation of pivot tables. **Python** data analysis toolkit **pandas** has the function *pivot_table*, and the *xs* method useful to obtain sections of pivot tables.

Online Analytical Processing (OLAP)

Excel pivot tables include the feature to directly query an online analytical processing (OLAP) server for retrieving data instead of getting the data from an Excel spreadsheet. On this configuration a pivot table is a simple client of an OLAP server. Excel's PivotTable not only allows for connecting to Microsoft's Analysis Service, but to any XML for Analysis (XMLA) OLAP standard-compliant server.

Power Pivot

Power Pivot is a feature of Microsoft Excel. It is available as an add-in in Excel 2010, 2013 in separate downloads, and as an add-in included with the Excel 2016 program. Power Pivot extends a local instance of Microsoft Analysis Services Tabular that is embedded directly into an Excel Workbook. This allows a user to build a ROLAP model in Power Pivot, and use pivot tables to explore the model once it is built. This allows Excel to act as a Self-Service BI platform, implementing professional expression languages to query the model and calculate advanced measures.

Power Pivot primarily uses DAX (Data Analysis Expressions) as its expression language, although the model can be queried via MDX in a row set expression. DAX expressions allow a user to create measures based on the data model, which can summarize and aggregate millions of rows of table data in seconds. DAX expressions resolve to T-SQL queries in the Formula and Storage Engines that drive the data model, abstracting the more verbose and tedious work of writing formal queries to excel-like formula expressions.

Power Pivot uses the SSAS Vertipaq compression engine to hold the data model in memory on the client computer. Practically, this means that Power Pivot is acting as an Analysis Services Server instance on the local workstation. As a result, larger data models may not be compatible with the 32-bit version of Excel.

Prior to the release of Power Pivot, Microsoft relied heavily on SQL Server Analysis Services as the engine for its Business Intelligence suite. Power Pivot complements the SQL Server core BI components under the vision of one Business Intelligence Semantic Model (BISM), which aims to integrate on-disk multidimensional analytics previously known as Unified Dimensional Model (UDM), with a more flexible, in-memory "tabular" model.

As a self-service BI product, Power Pivot is intended to allow users with no specialized BI or analytics training to develop data models and calculations, sharing them either directly or through SharePoint document libraries.

M Formula Language

A feature in Power Pivot's Get & Transform (formally known as Power Query) includes a new formula language called M. It is a mashup query language designed to build queries that mashup data. It is similar to F-Sharp. According to Microsoft, it "is a mostly pure, higher-order, dynamically typed, partially lazy, functional language."

Multidimensional Analysis (MDA)

In statistics, econometrics, and related fields, **multidimensional analysis** (**MDA**) is a data analysis process that groups data into two categories: data dimensions and measurements. For example, a data set consisting of the number of wins for a single football team at each of several years is a single-dimensional (in this case, longitudinal) data set. A data set consisting of the number of wins for several football teams in a single year is also a single-dimensional (in this case, cross-sectional) data set. A data set consisting of the number of wins for several football teams over several years is a two-dimensional data set.

Higher Dimensions

In many disciplines, two-dimensional data sets are also called panel data. While, strictly speaking, two- and higher-dimensional data sets are "multi-dimensional", the term "multidimensional" tends to be applied only to data sets with three or more dimensions. For example, some forecast data sets provide forecasts for multiple target periods, conducted by multiple forecasters, and made at multiple horizons. The three dimensions provide more information than can be gleaned from two-dimensional panel data sets.

Software

Computer systems for MDA include Online analytical processing (OLAP) for data in relational databases, and pivot tables, for data in spreadsheets.

MultiDimensional eXpressions (MDX)

Multidimensional Expressions (MDX) is a query language for online analytical processing (OLAP) using a database management system. Much like SQL, it is a query language for OLAP cubes. It is also a calculation language, with syntax similar to spreadsheet formulas.

Background

The MultiDimensional eXpressions (MDX) language provides a specialized syntax for querying and manipulating the multidimensional data stored in OLAP cubes. While it is possible to translate some of these into traditional SQL, it would frequently require the synthesis of clumsy SQL expressions even for very simple MDX expressions. MDX has been embraced by a wide majority of OLAP vendors and has become the standard for OLAP systems.

MDX Data Types

There are six primary data types in MDX

- **Scalar**. Scalar is either a number or a string. It can be specified as a literal, e.g. number 5 or string "OLAP" or it can be returned by an MDX function, e.g. Aggregate (number), UniqueName (string), Value (number or string) etc.
- **Dimension/Hierarchy**. Dimension is a dimension of a cube. A dimension is a primary organizer of measure and attribute information in a cube. MDX does not know of, nor does it assume any, dependencies between dimensions - they are assumed to be mutually independent. A dimension will contain some members (see below) organized in some hierarchy or

hierarchies containing levels. It can be specified by its unique name, e.g. [Time] or it can be returned by an MDX function, e.g. Dimension. Hierarchy is a dimension hierarchy of a cube. It can be specified by its unique name, e.g. [Time].[Fiscal] or it can be returned by an MDX function, e.g. .Hierarchy. Hierarchies are contained within dimensions. (*OLEDB for OLAP MDX specification does not distinguish between dimension and hierarchy data types. Some implementations, such as Microsoft Analysis Services, treat them differently.*)

- **Level**. Level is a level in a dimension hierarchy. It can be specified by its unique name, e.g. [Time].[Fiscal].[Month] or it can be returned by an MDX function, e.g. .Level.
- **Member**. Member is a member in a dimension hierarchy. It can be specified by its unique name, e.g. [Time].[Fiscal].[Month].[August 2006], by qualified name, e.g. [Time].[Fiscal].[2006].[Q3].[August 2006] or returned by an MDX function, e.g. .PrevMember, .Parent, .FirstChild etc. Note that all members are specific to a hierarchy. If the self-same product is a member of two different hierarchies ([Product].[ByManufacturer] and [Product].[ByCategory]), there will be two different members visible that may need to be coordinated in sets and tuples (see below).
- **Tuple**. Tuple is an ordered collection of one or more members from different dimensions. Tuples can be specified by enumerating the members, e.g. ([Time].[Fiscal].[Month].[August], [Customer].[By Geography].[All Customers].[USA], [Measures].[Sales]) or returned by an MDX function, e.g. .Item.
- **Set**. Set is an ordered collection of tuples with the same dimensionality, or hierarchality in the case of Microsoft's implementation. It can be specified

enumerating the tuples, e.g. {([Measures].[Sales], [Time].[Fiscal].[2006]), ([Measures].[Sales], [Time].[Fiscal].[2007])} or returned by MDX function or operator, e.g. Crossjoin, Filter, Order, Descendants etc.

- Other data types. Member properties are equivalent to *attributes* in the data warehouse sense. They can be retrieved by name in a query through an axis PROPERTIES clause of a query. The scalar data value of a member property for some member can be accessed in an expression through MDX, either by naming the property (for example, [Product].CurrentMember.[Sales Price]) or by using a special access function (for example, [Product].CurrentMember.Properties("Sales Price")). In limited contexts, MDX allows other data types as well - for example Array can be used inside the SetToArray function to specify an array that is not processed by MDX but passed to a user-defined function in an ActiveX library. Objects of other data types are represented as scalar strings indicating the object names, such as measure group name in Microsoft's MeasureGroupMeasures function or KPI name in for example Microsoft's KPIValue or KPIGoal functions.

Example Query

The following example, adapted from the SQL Server 2000 Books Online, shows a basic MDX query that uses the SELECT statement. This query returns a result set that contains the 2002 and 2003 store sales amounts for stores in the state of California.

```
SELECT
    { [Measures].[Store Sales] } ON COLUMNS,
    { [Date].[2002], [Date].[2003] } ON ROWS
 FROM Sales
 WHERE ( [Store].[USA].[CA] )
```

In this example, the query defines the following result set information

- The SELECT clause sets the query axes as the Store Sales member of the Measures dimension, and the 2002 and 2003 members of the Date dimension.
- The FROM clause indicates that the data source is the Sales cube.
- The WHERE clause defines the "slicer axis" as the California member of the Store dimension.

Note: You can specify up to 128 query axes in an MDX query.

If you create two axes, one must be the column axis and one must be the row axis, although it doesn't matter in which order they appear within the query. If you create a query that has only one axis, it must be the column axis. The square brackets around the particular object identifier are optional as long as the object identifier is not one of the reserved words and does not otherwise contain any characters other than letters, numbers or underscores.

```
SELECT
     [Measures].[Store Sales] ON COLUMNS,
     [Date].Members ON ROWS
 FROM Sales
 WHERE ( [Store].[USA].[CA] )
```

Data Analysis Expressions (DAX)

Data Analysis Expressions (**DAX**) is the native formula and query language for Microsoft PowerPivot, Power BI Desktop and SQL Server Analysis Services (SSAS) Tabular models. DAX includes some of the functions that are used in Excel formulas with additional functions that are designed to work with relational data and perform dynamic aggregation. It is, in part, an evolution of the Multidimensional Expression (MDX) language developed by Microsoft for Analysis Services multidimensional models (often called cubes) combined with Excel formula functions. It is designed to be simple and easy to learn, while exposing the power and flexibility of PowerPivot and SSAS tabular models.

Background

The Data Analysis expressions (DAX) language provides a specialized syntax for querying Analysis Services tabular model. DAX is not a programming language. DAX is primarily a formula language and is also a query language. You can use DAX to define custom calculations for Calculated Columns and for Calculated Fields (measures) in Analysis Services Tabular Model.

DAX Data Types

DAX can compute values for seven data types:

- Integer
- Real
- Currency
- Date (datetime)

- TRUE/FALSE (Boolean)
- String
- BLOB (binary large object)

DAX has a powerful type-handling system so that you do not have to worry much about data types. When you write a DAX expression, the resulting type is based on the type of the terms used in the expression and on the operator used. Type conversion happens automatically during the expression evaluation.

XML for Analysis (XMLA)

XML for Analysis (abbreviated as **XMLA**) is an industry standard for data access in analytical systems, such as OLAP and data mining. XMLA is based on other industry standards such as XML, SOAP and HTTP. XMLA is maintained by **XMLA Council** with Microsoft, Hyperion and SAS being the official XMLA Council founder members.

API

XMLA consists of only two SOAP methods. It was designed in such a way to preserve simplicity.

- Execute
- Discover

Execute

Execute method has two parameters:

- Command - command to be executed. It can be MDX, DMX or SQL.
- Properties - XML list of command properties such as Timeout, Catalog name, etc.

The result of Execute command could be *Multidimensional Dataset* or *Tabular Rowset*.

Discover

Discover method was designed to model all the discovery methods possible in OLEDB including various schema rowset, properties, keywords, etc. Discover method allows users to specify both what needs to be discovered and the possible restrictions or properties. The result of Discover method is a rowset.

Query Language

XMLA specifies **MDXML** as the query language. In the XMLA 1.1 version, the only construct in MDXML is an MDX statement enclosed in the <Statement> tag.

Example

Below is an example of XMLA Execute request with MDX query in command.

```
<soap:Envelope>
 <soap:Body>
  <Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
   <Command>
    <Statement>SELECT Measures.MEMBERS ON COLUMNS FROM
Sales</Statement>
   </Command>
   <Properties>
    <PropertyList>
     <DataSourceInfo/>
     <Catalog>FoodMart</Catalog>
     <Format>Multidimensional</Format>
     <AxisFormat>TupleFormat</AxisFormat>
    </PropertyList>
   </Properties>
  </Execute>
 </soap:Body>
```

```
</soap:Envelope>
```

Session Management

XMLA has a notion of *session state*. It is maintained through predefined *SOAP headers*

- BeginSession - to begin a new session
- EndSession - to end existing session
- UseSession - to use existing session. SessionId attribute previously returned for BeginSession should be used.

OLE DB for OLAP

OLE DB for OLAP (Object Linking and Embedding Database for Online Analytical Processing abbreviated **ODBO**) is a Microsoft published specification and an industry standard for multi-dimensional data processing. ODBO is the standard application programming interface (API) for exchanging metadata and data between an OLAP server and a client on a Windows platform. ODBO extends the ability of OLE DB to access multi-dimensional (OLAP) data stores.

Description

ODBO is the most widely supported, multi-dimensional API to date. Platform-specific to Microsoft Windows, ODBO was specifically designed for Online Analytical Processing (OLAP) systems by Microsoft as an extension to Object Linking and Embedding Database (OLE DB). ODBO uses Microsoft's Component Object Model.

ODBO permits independent software vendors (ISVs) and corporate developers to create a single set of standard interfaces that allow OLAP clients to access multi-dimensional data, regardless of vendor or data source. ODBO is currently supported by a wide spectrum of server and client tools.

When exposing the ODBO interface, the underlying multi-dimensional database must also support the MDX Query Language. XML for Analysis is a newer interface to MDX Data Sources that is often supported in parallel with ODBO.

Microsoft Analysis Services

Microsoft SQL Server Analysis Services, SSAS, is an online analytical processing (OLAP) and data mining tool in Microsoft SQL Server. SSAS is used as a tool by organizations to analyze and make sense of information possibly spread out across multiple databases, or in disparate tables or files. Microsoft has included a number of services in SQL Server related to business intelligence and data warehousing. These services include Integration Services, Reporting Services and Analysis Services. Analysis Services includes a group of OLAP and data mining capabilities and comes in two flavors - Multidimensional and Tabular.

Multidimensional Storage Modes

Microsoft Analysis Services takes a neutral position in the MOLAP vs. ROLAP arguments among OLAP products. It allows all the flavors of MOLAP, ROLAP and HOLAP to be used within the same model.

Partition Storage Modes

MOLAP - Multidimensional OLAP - Both fact data and aggregations are processed, stored, and indexed using a special format optimized for multidimensional data.

- ROLAP - Relational OLAP - Both fact data and aggregations remain in the relational data source, eliminating the need for special processing.

- HOLAP - Hybrid OLAP - This mode uses the relational data source to store the fact data, but pre-processes aggregations and indexes, storing these in a special format, optimized for multidimensional data.

Dimension Storage Modes

- MOLAP - dimension attributes and hierarchies are processed and stored in the special format
- ROLAP - dimension attributes are not processed and remain in the relational data source.

APIs and Object Models

Microsoft Analysis Services supports different sets of APIs and object models for different operations and in different programming environments.

Querying

- **XML for Analysis** - The lowest level API. It can be used from any platform and in any language that supports HTTP and XML
- **OLE DB for OLAP** - Extension of OLEDB. COM based and suitable for C/C++ programs on Windows platform.
- **ADOMD** - Extension of ADO. COM Automation based and suitable for VB programs on Windows platform.
- **ADOMD.NET** - Extension of ADO.NET. .NET based and suitable for managed code programs on CLR platforms.
- **ADO.NET Entity Framework** - Entity Framework and LINQ can be used on top of ADOMD.NET (SSAS Entity Framework Provider is required)

Administration and management

- **DSO** - For AS 2000. COM Automation based and suitable for VB programs on Windows platform.

- **AMO** - For AS 2005 and later versions. .NET based and suitable for managed code programs on CLR platforms.

Query Languages

Microsoft Analysis Services supports the following query languages

Data definition language (DDL)

DDL in Analysis Services is XML based and supports commands such as <Create>, <Alter>, <Delete>, and <Process>. For data mining models import and export, it also supports PMML.

Data manipulation language (DML)

- **MDX** - for querying OLAP cubes

- **LINQ** - for querying OLAP cubes from .NET using ADO.NET Entity Framework and Language INtegrated Query (SSAS Entity Framework Provider is required)

- **SQL** - small subset of SQL (in form of management views also called as DMV's) for querying OLAP cubes and dimensions as if they were tables

- **DMX** - for querying Data Mining models

- **DAX** - for querying Tabular models

Mondrian OLAP Server

Mondrian is an open source OLAP (online analytical processing) server, written in Java. It supports the MDX (multidimensional expressions) query language and the XML for Analysis and olap4j interface specifications. It reads from SQL and other data sources and aggregates data in a memory cache.

Mondrian is used for:

- High performance, interactive analysis of large or small volumes of information
- Dimensional exploration of data, for example analyzing sales by product line, by region, by time period
- Parsing the MDX language into Structured Query Language (SQL) to retrieve answers to dimensional queries
- High-speed queries through the use of aggregate tables in the RDBMS
- Advanced calculations using the calculation expressions of the MDX language

Mondrian Frontends

- JPivot
- Saiku
- Pentaho Analyzer
- Pivot4J
- Pivot4j Drill
- EazyBI

Cubes (OLAP Server)

Cubes is a light-weight open source multidimensional modelling and OLAP toolkit for development reporting applications and browsing of aggregated data written in Python programming language released under the MIT License.

Cubes provides to an analyst or any application end-user "understandable and natural way of reporting using concept of Data Cubes – multidimensional data objects".

Cubes was first publicly released in March 2011. The project was originally developed for Public Procurements of Slovakia. Cubes 1.0 was released in September 2014 and presented on the PyData Conference in New York

Features

- OLAP and aggregated browsing (default is ROLAP)
- Logical model of OLAP cubes in JSON or provided from external sources
- Hierarchical dimensions (attributes that have hierarchical dependencies, such as category-subcategory or country-region)
- Multiple hierarchies in a dimension
- Arithmetic expressions for computing derived measures and aggregates
- Localizable metadata and data

Model

The logical conceptual model in Cubes is described using JSON and can be provided either in a form of a file, directory bundle or from an external model provider (for example a database). The basic model objects are: cubes and their measures and aggregates, dimensions and their attributes, hierarchies. Logical model also contains mapping from logical attributes to their physical location in a database (or other data source).

Example model:

```
{
    "cubes": [
        {
            "name": "sales",
            "label": "Our Sales",
            "dimensions": [ "date", "customer", "location",
"product" ],
            "measures": [ "amount" ]
        }
    ]
    "dimensions": [
        {
            "name": "product",
            "label": "Product",
            "levels": [
                {
                    "name":"category",
                    "label":"Category",
                    "attributes": [ "category_id",
"category_label" ],
                },
                {
                    "name":"product",
                    "label":"Product",
                    "attributes": [ "product_id",
"product_label" ],
                }
            ]
        },
        ...
    ]
}
```

Operations

Cubes provides basic set of operations such as Data drilling and filtering (slicing and dicing). The operations can be accessed either through Python interface or through a light web server called Slicer.

Example of the python interface:

```
import cubes

workspace = Workspace("slicer.ini")
browser = workspace.browser("sales")

result = browser.aggregate()

print(result.summary)
```

Server

The Cubes provides a non-traditional OLAP server with HTTP queries and JSON response API. Example query to get "total amount of all contracts between January 2012 and June 2016 by month":

```
http://localhost:5000/cube/contracts/aggregate?drilldown=da
te&drilldown=criteria&cut=date:2012,1-
2012,6&order=date.month:desc
```

The response looks like:

```
{
    "summary": {
        "contract_amount_sum": 10000000.0
    },
    "remainder": {},
    "cells": [
        {
            "date.year": 2012,
            "criteria.code": "ekonaj",
            "contract_amount_sum": 12345.0,
            "criteria.description": "economically best
offer",
            "criteria.sdesc": "best offer",
            "criteria.id": 3
        },
        {
            "date.year": 2012,
            "criteria.code": "cena",
            "contract_amount_sum": 23456.0,
            "criteria.description": "lowest price",
            "criteria.sdesc": "lowest price",
            "criteria.id": 4
        },
...
    "total_cell_count": 6,
    "aggregates": [
        "contract_amount_sum"
    ],
    "cell": [
        {
            "type": "range",
            "dimension": "date",
            "hierarchy": "default",
            "level_depth": 2,
            "invert": false,
            "hidden": false,
            "from": ["2012", "1" ],
            "to": ["2015", "6" ]
        }
    ],
    "levels": {
        "criteria": [ "criteria" ],
        "date": [ "year" ]
    }
}
```

The simple HTTP/JSON interface makes it very easy to integrate OLAP reports in web applications written in pure HTML and JavaScript.

The Slicer server contains endpoints describing the cube metadata which helps to create generic reporting applications that don't have to know the database model structure and conceptual hierarchies up-in-front.

The Slicer server is written using the Flask (web framework).

ROLAP and SQL

The built-in SQL backend of the framework provides ROLAP functionality on top a relational database. Cubes contains a SQL query generator that translates the reporting queries into SQL statements. The query generator takes into account topology of the star or snowflake schema and executes only joins that are necessary to retrieve attributes required by the data analyst.

The SQL backend uses SQLAlchemy Python toolkit to construct the queries.

icCube

icCube is a company founded in Switzerland that provides business intelligence software of the same name. The solution can be fully embedded as an integrated solution, can be hosted in a managed environment or installed locally, on premises.

The BI tool allows end-users to create or edit dashboards themselves and is capable of processing data from multiple sources in real-time. The solution distinguishes itself by making the dashboards, the dashboard builder, the schema/cube builder and the server monitoring application accessible from a browser only. No software has to be installed at the device of the end-user.

Next to the browser-based dashboard builder, data can be accessed by running queries directly on the OLAP cube using MDX, SQL or R.

Architecture

icCube is implemented in Java and follows J2EE standards. For the latter, it embeds both an HTTP server (Jetty) and a servlet container to handle all the communication tasks.

Being an in-memory OLAP server, the icCube server does not need to source its data from a RDBMS; any data source that exposes its data in a tabular form can be used; several plugins exists for accessing files, HTTP stream, etc. Accessing datasource that expose JSON objects is also supported (e.g., MongoDB). icCube is then taking care of possibly complex relations (e.g., many-2-many) implied by the JSON structure.

Accessing icCube (cube modeling, server monitoring, MDX queries, Web reporting and dashboards) is performed through a unique Web interface and a JSON Rest API.

The icCube OLAP server does not use any caching or pre-aggregation mechanism.

Interfaces

icCube uses Multidimensional Expressions (MDX) as its query language and several extensions to the original language : function declarations, vector (even at measures level), matrix, objects, Java and R interactions. icCube patented an MDX debugger. icCube supports a standard interface and a proprietary one. The XML for Analysis (XMLA protocol can connect to any XMLA compatible reporting tool.

icCube supports its own proprietary protocol called GVI. HTTP based, it can be extended. This protocol leverages the Google Visualization wire protocol. JavaScript is the primary implementation language and a Java mapping library is also available.

Since icCube 6.8.6, the icCube server supports a JSON Rest API for a programmatic access.

Apache Kylin

Apache Kylin is an open source distributed analytics engine designed to provide a SQL interface and multi-dimensional analysis (OLAP) on Hadoop and Alluxio supporting extremely large datasets.

It was originally developed by eBay and is now a project of the Apache Software Foundation.

Architecture

Apache Kylin is built on top of Apache Hadoop, Apache Hive, Apache HBase, Apache Calcite, Apache Spark and other technologies. As these technologies are very powerful and matured, making Kylin can support massive data load, and easy to scale.

Kylin has the following core components:

- REST Server: Receive and response user or API requests
- Metadata: Persistent and manage system, especially the cube metadata;
- Query Engine: Parse SQL queries to execution plan, and then talk with storage engine;
- Storage Engine: Pushdown and scan underlying cube storage (default in HBase);

- Job Engine: Generate and execute MapReduce or Spark job to build source data into cube;

Users

Apache Kylin has been adopted by many companies as their OLAP platform in production. Typical users include eBay, Meituan, XiaoMi, NetEase, Beike, Yahoo! Japan.

Roadmap

Apache Kylin roadmap (from Kylin website):

- Hadoop 3.0 support (Erasure Coding)
- Fully on Spark Cube engine
- Connect more data sources (MySQL, Oracle, SparkSQL, etc)
- Ad-hoc queries without Cubing
- Better storage (Druid, Kudu, etc)
- Real-time analytics with Lambda Architecture

Apache Druid

Druid is a column-oriented, open-source, distributed data store written in Java. Druid is designed to quickly ingest massive quantities of event data and provide low-latency queries on top of the data. The name Druid comes from the shapeshifting Druid class in many role-playing games, to reflect the fact that the architecture of the system can shift to solve different types of data problems.

Druid is commonly used in business intelligence/OLAP applications to analyze high volumes of real-time and historical data. Druid is used in production by technology companies such as Alibaba, Airbnb, Cisco, eBay, Netflix, PayPal, Yahoo. and Wikimedia Foundation

Architecture

Fully deployed, Druid runs as a cluster of specialized processes (called nodes in Druid) to support a fault-tolerant architecture where data is stored redundantly, and there is no single point of failure. The cluster includes external dependencies for coordination (Apache ZooKeeper), metadata storage (e.g. MySQL, PostgreSQL, or Derby), and a deep storage facility (e.g. HDFS, or Amazon S3) for permanent data backup.

Query management

Client queries first hit broker nodes, which forward them to the appropriate data nodes (either historical or real-time). Since Druid segments may be partitioned, an incoming query can require data from multiple segments and partitions (or shards) stored on different nodes in the cluster. Brokers are able to learn which nodes have the required data, and also merge partial results before returning the aggregated result.

Cluster management

Operations relating to data management in historical nodes are overseen by coordinator nodes. Apache ZooKeeper is used to register all nodes, manage certain aspects of internode communications, and provide for leader elections.

Features

- Low latency (streaming) data ingestion
- Arbitrary slice and dice data exploration
- Sub-second analytic queries
- Approximate and exact computations

Comparison of OLAP Servers

The following tables compare general and technical information for a number of online analytical processing (OLAP) servers.

General Information

OLAP Server	Company	Latest stable version	Software license
Apache Kylin	Apache Software Foundation	2.5.1	Apache 2.0
Druid	Open source community	0.11.0	Apache 2.0
Apache Pinot	Apache Software Foundation	0.1.0	Apache 2.0
Essbase	Oracle	11.1.2.4	Proprietary

OLAP Server	Company	Latest stable version	Software license
IBM Cognos TM1	IBM	10.2.2 FP7	Proprietary
icCube	icCube	6.8.9	Proprietary
Infor BI OLAP Server	Infor	10.6.0	Proprietary
Jedox OLAP Server	Jedox	2019.2	GPL v2 or EULA, Proprietary
Kyligence	Kyligence	3.2.1	Proprietary
Kyvos BI on Big Data	Kyvos Insights	4.0	Proprietary

OLAP Server	Company	Latest stable version	Software license
Microsoft Analysis Services	Microsoft	2016	Proprietary
MicroStrategy Intelligence Server	MicroStrategy	10.11	Proprietary
Mondrian OLAP Server	Pentaho	3.7	EPL
Oracle Database OLAP Option	Oracle	11g R2	Proprietary
SAP NetWeaver BW	SAP	7.30	Proprietary
SAS OLAP Server	SAS Institute	9.4	Proprietary

Data Storage Modes

OLAP Server	MOLAP	ROLAP	HOLAP	Offline
Apache Kylin	Yes	No	No	Yes
Druid	Yes	Yes	Yes	Yes
Apache Pinot	Yes	Yes	Yes	Yes
Essbase	Yes	No	No	
IBM Cognos BI	Yes	Yes	Yes	
IBM Cognos TM1	Yes	No	No	Cognos Insight Distributed mode
icCube	Yes	No	No	Offline Cubes

OLAP Server	MOLAP	ROLAP	HOLAP	Offline
Infor BI OLAP Server	Yes	No	No	Local cubes
Jedox OLAP Server	Yes	No	No	No
Kyligence	Yes	Yes	Yes	Yes
Kyvos BI on Big Data	Yes	Yes	No	Yes
Microsoft Analysis Services	Yes	Yes	Yes	Local cubes, PowerPivot for Excel, Power BI Desktop
MicroStrategy Intelligence Server	Yes	Yes	Yes	MicroStrategy Office, Dynamic Dashboards

OLAP Server	MOLAP	ROLAP	HOLAP	Offline
Mondrian OLAP Server	No	Yes	No	
Oracle Database OLAP Option	No	Yes	No	
SAP NetWeaver BW	Yes	Yes	No	
SAS OLAP Server	Yes	Yes	Yes	

APIs and Query Languages

APIs and query languages OLAP servers support.

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
Apache Kylin	No	No	No	No	Yes	Yes	Superset, Zeppelin, Tableau, Qlik, Redash	Yes	Yes
Druid	No	No	No	No	Yes	Druid SQL	Superset, **Pivot**	Yes	Yes

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
Apache Pinot	No	No	No	No	Yes	Pinot SQL	Superset	Yes	Yes
Essbase	Yes	Yes	Yes	Yes	Yes	No	SmartView (Excel-AddIn), WebAnalysis, Financial Reports	?	?
IBM Cognos TM1	Yes	Yes	Yes	Yes	Yes	No	TM1 Web/TM1 Contributor, IBM	Yes	Yes

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
	🟩	🟩	🟩	🟩	🟩	🟥	Cognos Insight, IBM Performance Modeler, IBM Cognos Cafe for Excel, Cognos BI, TM1 Perspectives for Excel	🟩	🟩

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
icCube	Yes	Yes	Yes	Java, R	Yes	In the reporting	icCube reporting and all XMLA compliant visualization tools like Excel, etc	Yes	Yes
Infor BI OLAP Server	Yes	Yes	Yes	OLAP Rules, Push Rules, Applicati	Yes	Yes	Application Studio	?	?

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
				on Engine					
Jedox OLAP Server	Yes	Yes	Yes	Cube Rules, SVS Triggers	Yes	No	Microsoft Excel, Qlik, Tableau, Jedox Web, Power BI	No	Yes
Kyligence	No	No	Yes	No	Yes	Yes	IBM Cognos, Business Objects, OBIEE,	Yes	Yes

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
	🟥	🟥	🟩	🟥	🟩	🟩	MicroStrategy, Microsoft Excel, Microsoft Power BI, Qlik, Superset, Tableau	🟩	🟩
Kyvos BI on Big Data	Yes	Yes	Yes	No	Yes	Yes	Kyvos Insights, Microsoft Excel, Qlik, Tableau, Power BI,	??	Yes

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
	🟩	🟩	🟩	🟥	🟩	🟩	MicroStrategy, IBM Cognos, Business Objects		🟩
Microsoft Analysis Services	Yes	Yes	Yes	.NET	Yes	Yes	Microsoft Excel, SharePoint, Microsoft Power BI, and 70+ other visualization tools	No	No

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
MicroStrategy Intelligence Server	Yes	No	Yes	Yes	Yes	Yes	?	?	?
Mondrian OLAP Server	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	?
Oracle Database OLAP Option	No	Yes	Yes	Java, PL/SQL, OLAP DML	Yes	Yes	?	?	?

OLAP Server	XMLA	OLE DB for OLAP	MDX	Stored Procedures	Custom Functions	SQL	Visualization	JSON	REST API
SAP NetWeaver BW	Yes	Yes	Yes	No	Yes	No	?	?	?
SAS OLAP Server	Yes	Yes	Yes	No	No	No	Web Report Studio	?	?

OLAP distinctive features

A list of OLAP features that are not supported by all vendors. All vendors support features such as parent-child, multilevel hierarchy, drilldown.

Data processing, management and performance related features:

OLAP Server	Real Time	Write-back	Partitioning	Usage Based Optimizations	Load Balancing and Clustering
Apache Kylin	Developing	No	Yes	Yes	Yes
Druid	Yes	?	Yes	Yes	Yes
Apache Pinot	Yes	?	Yes	Yes	Yes
Essbase	Yes	Yes	Yes	Yes	Yes

OLAP Server	Real Time	Write-back	Partitioning	Usage Based Optimizations	Load Balancing and Clustering
IBM Cognos BI	Yes	No	Yes	Yes	?
IBM Cognos TM1	Yes	Yes	Yes	?	?
icCube	Yes	Yes	Yes	?	?
Infor BI OLAP Server	Yes	Yes	Yes	?	?
Jedox OLAP Server	Yes	Yes	Yes	?	?

OLAP Server	Real Time	Write-back	Partitioning	Usage Based Optimizations	Load Balancing and Clustering
Kyligence	No	No	Yes	Yes	Yes
Kyvos BI on Big Data	Yes	Yes	Yes	Yes	Yes
Microsoft Analysis Services	Yes	Yes	Yes	Yes	Yes
MicroStrategy Intelligence Server	?	Yes	Yes	?	?
Mondrian OLAP server	Yes	Planned	Yes	?	?

OLAP Server	Real Time	Write-back	Partitioning	Usage Based Optimizations	Load Balancing and Clustering
Oracle Database OLAP Option	?	Yes	Yes	No	?
SAP NetWeaver BW	?	Yes	Yes	?	?
SAS OLAP Server	?	Yes	Yes	?	?

Data modeling features:

OLAP Server	Semi-additive measures	Many-to-Many	Multi-Cube Model	Perspectives	KPI	Multilingual	Named Sets	Multi-attribute Hierarchies	Actions
Apache Kylin	No	No	Yes	No	Yes	Yes	No	Yes	Yes
Druid	Yes	Yes	Yes	?	No	Yes	?	Yes	Yes
Apache Pinot	Yes	Yes	Yes	?	?	No	No	No	No
Essbase	Yes	?	?	?	Yes	Yes	Yes	Yes	?

OLAP Server	**Semi-additive measures**	**Many-to-Many**	**Multi-Cube Model**	**Perspectives**	**KPI**	**Multilingual**	**Named Sets**	**Multi-attribute Hierarchies**	**Actions**
IBM Cognos BI	Yes	Yes	?	?	?	?	Yes	Yes	?
IBM Cognos TM1	Yes	Yes	Yes	?	?	?	?	?	?
icCube	Yes	Yes	Yes	Yes	?	Yes	Yes	?	?
Infor BI OLAP Server	Yes	?	Yes	?	Yes	?	?	?	?

OLAP Server	Semi-additive measures	Many-to-Many	Multi-Cube Model	Perspectives	KPI	Multilingual	Named Sets	Multi-attribute Hierarchies	Actions
Jedox OLAP Server	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?
Kyligence	Yes	No	Yes	No	Yes	Yes	?	Yes	?
Kyvos BI on Big Data	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No
Microsoft	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

OLAP Server	Semi-additive measures	Many-to-Many	Multi-Cube Model	Perspectives	KPI	Multilingual	Named Sets	Multi-attribute Hierarchies	Actions
Analysis Services	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
MicroStrategy Intelligence Server	Yes	?	?	?	?	?	?	?	?
Mondrian OLAP server	Yes	?	?	?	?	?	?	?	?

OLAP Server	Semi-additive measures	Many-to-Many	Multi-Cube Model	Perspectives	KPI	Multilingual	Named Sets	Multi-attribute Hierarchies	Actions
Oracle Database OLAP Option	Yes	?	?	?	?	?	?	?	?
SAP NetWeaver BW	Yes	?	?	?	?	?	?	?	?
SAS OLAP Server	Yes	?	?	?	?	?	?	?	?

System limits

OLAP Server	# cubes	# measures	# dimensions	# dimensions in cube	# hierarchies in dimension	# levels in hierarchy	# dimension members
Apache Kylin	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted
Druid	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted
Apache Pinot	Unrestricted	Unrestricted	Unrestricted	Unrestricted	-	-	Unrestricted
Essbase	?	?	?	255	255	?	20,000,000 (ASO), 1,000,0

OLAP Server	# cubes	# measures	# dimensions	# dimensions in cube	# hierarchies in dimension	# levels in hierarchy	# dimension members
							00 (BSO)
IBM Cognos TM1	Unrestricted	Unrestricted	Unrestricted	256	Unrestricted	Unrestricted	Unrestricted
icCube	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647
Infor BI OLAP Server	?	10,000,000	?	30	?	?	10,000,000

OLAP Server	# cubes	# measures	# dimensions	# dimensions in cube	# hierarchies in dimension	# levels in hierarchy	# dimension members
Jedox OLAP Server	(32 bits)		(32 bits)	250			
Kyligence	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted	Unrestricted
Kyvos BI on Big Data	Unrestricted	Unrestricted	Unrestricted	Unrestricted	**1**	Unrestricted	Unrestricted
Microsoft Analysis	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647 (max. number	2,147,483,647	2,147,483,647	2,147,483,647 (xOLAP)

OLAP Server	# cubes	# measures	# dimensions	# dimensions in cube	# hierarchies in dimension	# levels in hierarchy	# dimension members
Services				of dimensions in a database)			Unrestricted (In-memory)
MicroStrategy Intelligence Server	Unrestricted	Unrestricted	Unrestricted	?	Unrestricted	Unrestricted	Unrestricted
SAS OLAP Server	Unrestricted	1024	128	?	128	19	4,294,967,296

Security

OLAP Server	Authentication	Network encryption	On-the-Fly	Data access		
				Cell security	Dimension security	Visual totals
Apache Kylin	LDAP, SAML, Kerboros, Microsoft Active Directory	SSL	Yes	No	No	?
Druid	Druid Database authentication	SSL	Yes	No	Yes	No
Apache Pinot	Plugin based	Coming soon	Plugin based	?	?	?
Essbase	Essbase authentication, LDAP auth	SSL	Yes	Yes	Yes	No

OLAP Server	Authentication	Network encryption	On-the-Fly	Data access		
				Cell security	Dimension security	Visual totals
	entication, Microsoft Active Directory					
IBM Cognos TM1	Built-in, LDAP, Microsoft Active Directory, NTLM, IBM Cognos BI authentication	SSL	Yes	Yes	Yes	Yes
icCube	HTTP Basic/Form Authentication, Windows SSO (NTLM, Kerberos), Plugin Based for Embedded Usage	SSL	Yes	Yes	Yes	Yes

OLAP Server	Authentication	Network encryption	On-the-Fly	Data access		
				Cell security	Dimension security	Visual totals
Infor BI OLAP Server	OLAP authentication, Infor Federation Services, LDAP, Microsoft Active Directory	Yes	Yes	Yes	Yes	?
Jedox OLAP Server	Jedox authentication, LDAP, Microsoft Active Directory	SSL	Yes	Yes	Yes	?
Kyligence	LDAP, SAML, Kerboros, Microsoft Active Directory, Built-in	SSL	?	Yes	Yes	?

OLAP Server	Authentication	Network encryption	On-the-Fly	Data access		
				Cell security	Dimension security	Visual totals
Kyvos BI on Big Data	Built-in, LDAP, Microsoft Active Directory, Windows SSO (NTLM, Kerberos)	SSL	Yes	Yes	Yes	?
Microsoft Analysis Services	NTLM, Kerberos	SSL and SSPI	Yes	Yes	Yes	Yes
MicroStrategy Intelligence Server	Host authentication, database authentication, LDAP, Microsoft Active Directory, NTLM, SiteMinder, Tivoli, SAP, Kerberos	SSL, AES	?	Yes	Yes	Yes

OLAP Server	Authentication	Network encryption	On-the-Fly	Data access		
				Cell security	Dimension security	Visual totals
Oracle Database OLAP Option	Oracle Database authentication	SSL	?	Yes	Yes	No
SAS OLAP Server	Host authentication, SAS token authentication, LDAP, Microsoft Active Directory	Yes	?	Yes	Yes	Yes

Operating systems

The OLAP servers can run on the following operating systems:

OLAP Server	Windows	Linux	UNIX	z/OS	AIX
Apache Kylin	No	Yes	Yes	No	No
Druid	No	Yes	Yes		
Apache Pinot	No	Yes	Yes		
Essbase	Yes	Yes	Yes	No	
IBM Cognos TM1	Yes	Yes	Yes	No	Yes
icCube	Yes	Yes	Yes	Yes	Yes
Infor BI OLAP Server	Yes	No	No	No	

OLAP Server	Windows	Linux	UNIX	z/OS	AIX
Jedox OLAP Server	Yes	Yes	Yes	No	
Kyligence	No	Yes	Yes	No	No
Kyvos BI on Big Data	No	Yes	Yes	No	No
Microsoft Analysis Services	Yes	No	No	No	
MicroStrategy Intelligence Server	Yes	Yes	Yes	No	
Mondrian OLAP server	Yes	Yes	Yes	Yes	
Oracle Database OLAP Option	Yes	Yes	Yes	Yes	
SAP NetWeaver BW	Yes	Yes	Yes	Yes	

OLAP Server	Windows	Linux	UNIX	z/OS	AIX
SAS OLAP Server	Yes	Yes	Yes	Yes	

Support information

OLAP Server	Issue Tracking System	Source code
Apache Kylin	Jira	Open
Druid	Druid – Github Issues	Open
Apache Pinot	Jira	Open

OLAP Server	Issue Tracking System	Source code
Essbase	myOracle Support	Closed
IBM Cognos TM1	IBM Service Request	Closed
icCube	YouTrack	Closed
Infor BI OLAP Server	Infor Xtreme	Closed
Jedox OLAP Server	Mantis	Open
Kyligence	Zendesk	Closed

OLAP Server	Issue Tracking System	Source code
Kyvos BI on Big Data	Zendesk	Open
Microsoft Analysis Services	Connect	Closed
MicroStrategy Intelligence Server	MicroStrategy Resource Center	Closed
Mondrian OLAP server	Jira	Open
Oracle Database OLAP Option	myOracle Support	Closed

OLAP Server	Issue Tracking System	Source code
SAP NetWeaver BW	OSS	Closed
SAS OLAP Server	Support	Closed