

Data Warehouse Design

After the tools and team personnel selections are made, the data warehouse design can begin. The following are the typical steps involved in the data warehousing project cycle.

- Requirement Gathering
- Physical Environment Setup
- Data Modeling
- ETL
- OLAP Cube Design
- Front End Development
- Report Development
- Performance Tuning
- Query Optimization
- Quality Assurance
- Rolling out to Production
- Production Maintenance
- Incremental Enhancements

Requirement Gathering

- **Task Description**
- The first thing that the project team should engage in is gathering requirements from end users. Because end users are typically not familiar with the data warehousing process or concept, the help of the business sponsor is essential. Requirement gathering can happen as one-to-one meetings or as Joint Application Development (JAD) sessions, where multiple people are talking about the project scope in the same meeting.
- The primary goal of this phase is to identify what constitutes as a success for this particular phase of the data warehouse project. In particular, end user reporting / analysis requirements are identified, and the project team will spend the remaining period of time trying to satisfy these requirements.
- Associated with the identification of user requirements is a more concrete definition of other details such as hardware sizing information, training requirements, data source identification, and most importantly, a concrete project plan indicating the finishing date of the data warehousing project.
- Based on the information gathered above, a disaster recovery plan needs to be developed so that the data warehousing system can recover from accidents that disable the system. Without an effective backup and restore strategy, the system will only last until the first major disaster, and, as many data warehousing DBA's will attest, this can happen very quickly after the project goes live.

Deliverables

- A list of reports / cubes to be delivered to the end users by the end of this current phase.
- An updated project plan that clearly identifies resource loads and milestone delivery dates.

Physical Environment Setup

Task Description

Once the requirements are somewhat clear, it is necessary to set up the physical servers and databases. At a minimum, it is necessary to set up a development environment and a production environment. There are also many data warehousing projects where there are three environments: Development, Testing, and Production.

It is not enough to simply have different physical environments set up. The different processes (such as ETL, OLAP Cube, and reporting) also need to be set up properly for each environment.

It is best for the different environments to use distinct application and database servers. In other words, the development environment will have its own application server and database servers, and the production environment will have its own set of application and database servers.

Deliverables

- Hardware / Software setup document for all of the environments, including hardware specifications, and scripts / settings for the software.

Data Modeling

Task Description

This is a very important step in the data warehousing project. Indeed, it is fair to say that the foundation of the data warehousing system is the data model. A good data model will allow the data warehousing system to grow easily, as well as allowing for good performance.

In data warehousing project, the logical data model is built based on user requirements, and then it is translated into the physical data model. The detailed steps can be found in the [Conceptual, Logical, and Physical Data Modeling](#) section.

Part of the data modeling exercise is often the identification of data sources. Sometimes this step is deferred until the ETL step. However, my feeling is that it is better to find out where the data exists, or, better yet, whether they even exist anywhere in the enterprise at all. Should the data not be available, this is a good time to raise the alarm. If this was delayed until the ETL phase, rectifying it will becoming a much tougher and more complex process.

Deliverables

- Identification of data sources.
- Logical data model.
- Physical data model.

ETL

Task Description

The **ETL** (Extraction, Transformation, Loading) process typically takes the longest to develop, and this can easily take up to 50% of the data warehouse implementation cycle or longer. The reason for this is that it takes time to get the source data, understand the necessary columns, understand the business rules, and understand the logical and physical data models.

Deliverables

- Data Mapping Document
- ETL Script / ETL Package in the ETL tool

OLAP Cube Design

Task Description

Usually the design of the olap cube can be derived from the [Requirement Gathering](#) phase. More often than not, however, users have some idea on what they want, but it is difficult for them to specify the exact report / analysis they want to see. When this is the case, it is usually a good idea to include enough information so that they feel like they have gained something through the data warehouse, but not so much that it stretches the data warehouse scope by a mile. Remember that data warehousing is an iterative process - no one can ever meet all the requirements all at once.

Deliverables

- Documentation specifying the OLAP cube dimensions and measures.
- Actual OLAP cube / report.

Front End Development

Task Description

Regardless of the strength of the OLAP engine and the integrity of the data, if the users cannot visualize the reports, the data warehouse brings zero value to them. Hence front end development is an important part of a data warehousing initiative.

So what are the things to look out for in selecting a front-end deployment methodology? The most important thing is that the reports should need to be delivered over the web, so the only thing that the user needs is the standard browser. These days it is no longer desirable or feasible to have the IT department doing program installations on end users desktops just so that they can view reports. So, whatever strategy one pursues, make sure the ability to deliver over the web is a must.

The front-end options ranges from an internal front-end development using scripting languages such as ASP, PHP, or Perl, to off-the-shelf products such as Seagate Crystal Reports, to the more higher-level products such as Actuate. In addition, many OLAP vendors offer a front-end on their own. When choosing vendor tools, make sure it can be easily customized to suit the enterprise, especially the possible changes to the reporting requirements of the enterprise. Possible changes include not just the difference in report layout and report content, but also include possible changes in the back-end structure. For example, if the enterprise decides to change from Solaris/Oracle to Microsoft 2000/SQL Server, will the front-end tool be flexible enough to adjust to the changes without much modification?

Another area to be concerned with is the complexity of the reporting tool. For example, do the reports need to be published on a regular interval? Are there very specific formatting requirements? Is there a need for a GUI interface so that each user can customize her reports?

Deliverables

Front End Deployment Documentation

Report Development

Task Description

Report specification typically comes directly from the requirements phase. To the end user, the only direct touchpoint he or she has with the data warehousing system is the reports they see. So, report development, although not as time consuming as some of the other steps such as [ETL](#) and [data modeling](#), nevertheless plays a very important role in determining the success of the data warehousing project.

One would think that report development is an easy task. How hard can it be to just follow instructions to build the report? Unfortunately, this is not true. There are several points the data warehousing team needs to pay attention to before releasing the report.

User customization: Do users need to be able to select their own metrics? And how do users need to be able to filter the information? The report development process needs to take those factors into consideration so that users can get the information they need in the shortest amount of time possible.

Report delivery: What report delivery methods are needed? In addition to delivering the report to the web front end, other possibilities include delivery via email, via text messaging, or in some form of spreadsheet. There are reporting solutions in the marketplace that support report delivery as a flash file. Such flash file essentially acts as a mini-cube, and would allow end users to slice and dice the data on the report without having to pull data from an external source.

Access privileges: Special attention needs to be paid to who has what access to what information. A sales report can show 8 metrics covering the entire company to the company CEO, while the same report may only show 5 of the metrics covering only a single district to a District Sales Director.

Report development does not happen only during the implementation phase. After the system goes into production, there will certainly be requests for additional reports. These types of requests generally fall into two broad categories:

1. Data is already available in the data warehouse. In this case, it should be fairly straightforward to develop the new report into the front end. There is no need to wait for a major production push before making new reports available.
2. Data is not yet available in the data warehouse. This means that the request needs to be prioritized and put into a future data warehousing development cycle.

Deliverables

- Report Specification Documentation.
- Reports set up in the front end / reports delivered to user's preferred channel.

Performance Tuning

Task Description

There are three major areas where a data warehousing system can use a little performance tuning:

- ETL - Given that the data load is usually a very time-consuming process (and hence they are typically relegated to a nightly load job) and that data warehousing-related batch jobs are typically of lower priority, which means that the window for data loading is not very long. A data warehousing system that has its ETL process finishing right on-time is going to have a lot of problems simply because often the jobs do not get started on-time due to factors that is beyond the control of the data warehousing team. As a result, it is always an excellent idea for the data warehousing group to tune the ETL process as much as possible.

- Query Processing - Sometimes, especially in a ROLAP environment or in a system where the reports are run directly against the relationship database, query performance can be an issue. A study has shown that users typically lose interest after 30 seconds of waiting for a report to return. My experience has been that ROLAP reports or reports that run directly against the RDBMS often exceed this time limit, and it is hence ideal for the data warehousing team to invest some time to tune the query, especially the most popularly ones. We present a number of [query optimization](#) ideas.
- Report Delivery - It is also possible that end users are experiencing significant delays in receiving their reports due to factors other than the query performance. For example, network traffic, server setup, and even the way that the front-end was built sometimes play significant roles. It is important for the data warehouse team to look into these areas for performance tuning.

Deliverables

- Performance tuning document - Goal and Result

Query Optimization

For any production database, SQL query performance becomes an issue sooner or later. Having long-running queries not only consumes system resources that makes the server and application run slowly, but also may lead to table locking and data corruption issues. So, query optimization becomes an important task.

First, we offer some guiding principles for query optimization:

1. Understand how your database is executing your query

Nowadays all databases have their own query optimizer, and offers a way for users to understand how a query is executed. For example, which index from which table is being used to execute the query? The first step to query optimization is understanding what the database is doing. Different databases have different commands for this. For example, in MySQL, one can use "EXPLAIN [SQL Query]" keyword to see the query plan. In Oracle, one can use "EXPLAIN PLAN FOR [SQL Query]" to see the query plan.

2. Retrieve as little data as possible

The more data returned from the query, the more resources the database needs to expand to process and store these data. So for example, if you only need to retrieve one column from a table, do not use 'SELECT *'.

3. Store intermediate results

Sometimes logic for a query can be quite complex. Often, it is possible to achieve the desired result through the use of subqueries, inline views, and UNION-type statements. For those cases, the intermediate results are not stored in the database, but are immediately used within the query. This can lead to performance issues, especially when the intermediate results have a large number of rows.

The way to increase query performance in those cases is to store the intermediate results in a temporary table, and break up the initial SQL statement into several SQL statements. In many cases, you can even build an index on the temporary table to speed up the query performance even more. Granted, this adds a little complexity in query management (i.e., the need to manage temporary tables), but the speedup in query performance is often worth the trouble.

Below are several specific query optimization strategies.

- **Use Index**
Using an index is the first strategy one should use to speed up a query. In fact, this strategy is so important that index optimization is also discussed.
- **Aggregate Table**
Pre-populating tables at higher levels so less amount of data need to be parsed.
- **Vertical Partitioning**
Partition the table by columns. This strategy decreases the amount of data a SQL query needs to process.

- **Horizontal Partitioning**
Partition the table by data value, most often time. This strategy decreases the amount of data a SQL query needs to process.
- **Denormalization**
The process of denormalization combines multiple tables into a single table. This speeds up query performance because fewer table joins are needed.
- **Server Tuning**
Each server has its own parameters, and often tuning server parameters so that it can fully take advantage of the hardware resources can significantly speed up query performance.

Quality Assurance

Task Description

Once the development team declares that everything is ready for further testing, the QA team takes over. The QA team is always from the client. Usually the QA team members will know little about data warehousing, and some of them may even resent the need to have to learn another tool or tools. This makes the QA process a tricky one.

Sometimes the QA process is overlooked. On my very first data warehousing project, the project team worked very hard to get everything ready for Phase 1, and everyone thought that we had met the deadline. There was one mistake, though, the project managers failed to recognize that it is necessary to go through the client QA process before the project can go into production. As a result, it took five extra months to bring the project to production (the original development time had been only 2 1/2 months).

Deliverables

- QA Test Plan
- QA verification that the data warehousing system is ready to go to production

Rollout To Production

Task Description

Once the QA team gives thumbs up, it is time for the data warehouse system to go live. Some may think this is as easy as flipping on a switch, but usually it is not true. Depending on the number of end users, it sometimes takes up to a full week to bring everyone online! Fortunately, nowadays most end users access the data warehouse over the web, making going production sometimes as easy as sending out an URL via email.

Deliverables

- Delivery of the data warehousing system to the end users.

Production Maintenance

Task Description

Once the data warehouse goes production, it needs to be maintained. Tasks as such regular backup and crisis management become important and should be planned out. In addition, it is very important to consistently monitor end user usage. This serves two purposes: 1. To capture any runaway requests so that they can be fixed before slowing the entire system down, and 2. To understand how much users are utilizing the data warehouse for return-on-investment calculations and future enhancement considerations.

Time Requirement

Ongoing.

Deliverables

Consistent availability of the data warehousing system to the end users.

Incremental Enhancements

Task Description

Once the data warehousing system goes live, there are often needs for incremental enhancements. I am not talking about a new data warehousing phases, but simply small changes that follow the business itself. For example, the original geographical designations may be different, the company may originally have 4 sales regions, but now because sales are going so well, now they have 10 sales regions.

Deliverables

- Change management documentation
- Actual change to the data warehousing system

Dimensional Data Model

Dimensional data model is most often used in data warehousing systems. This is different from the 3rd normal form, commonly used for transactional (OLTP) type systems. As you can imagine, the same data would then be stored differently in a dimensional model than in a 3rd normal form model.

To understand dimensional data modeling, let's define some of the terms commonly used in this type of modeling:

Dimension: A category of information. For example, the time dimension.

Attribute: A unique level within a dimension. For example, Month is an attribute in the Time Dimension.

Hierarchy: The specification of levels that represents relationship between different attributes within a dimension. For example, one possible hierarchy in the Time dimension is Year → Quarter → Month → Day.

Fact Table: A fact table is a table that contains the measures of interest. For example, sales amount would be such a measure. This measure is stored in the fact table with the appropriate granularity. For example, it can be sales amount by store by day. In this case, the fact table would contain three columns: A date column, a store column, and a sales amount column.

Lookup Table: The lookup table provides the detailed information about the attributes. For example, the lookup table for the Quarter attribute would include a list of all of the quarters available in the data warehouse. Each row (each quarter) may have several fields, one for the unique ID that identifies the quarter, and one or more additional fields that specifies how that particular quarter is represented on a report (for example, first quarter of 2001 may be represented as "Q1 2001" or "2001 Q1").

A dimensional model includes fact tables and lookup tables. Fact tables connect to one or more lookup tables, but fact tables do not have direct relationships to one another. Dimensions and hierarchies are represented by lookup tables. Attributes are the non-key columns in the lookup tables.

In designing data models for data warehouses / data marts, the most commonly used schema types are [Star Schema](#) and [Snowflake Schema](#).

Whether one uses a star or a snowflake largely depends on personal preference and business needs. Personally, I am partial to snowflakes, *when there is a business case to analyze the information at that particular level*.

Fact Table Granularity

Granularity

The first step in designing a fact table is to determine the **granularity** of the fact table. By **granularity**, we mean the lowest level of information that will be stored in the fact table. This constitutes two steps:

1. Determine which dimensions will be included.
2. Determine where along the hierarchy of each dimension the information will be kept.

The determining factors usually goes back to the requirements.

Which Dimensions To Include

Determining which dimensions to include is usually a straightforward process, because business processes will often dictate clearly what are the relevant dimensions.

For example, in an off-line retail world, the dimensions for a sales fact table are usually time, geography, and product. This list, however, is by no means a complete list for all off-line retailers. A supermarket with a Rewards

Card program, where customers provide some personal information in exchange for a rewards card, and the supermarket would offer lower prices for certain items for customers who present a rewards card at checkout, will also have the ability to track the customer dimension. Whether the data warehousing system includes the customer dimension will then be a decision that needs to be made.

What Level Within Each Dimension To Include

Determining which part of hierarchy the information is stored along each dimension is not an exact science. This is where user requirement (both stated and possibly future) plays a major role.

In the above example, will the supermarket wanting to do analysis along at the hourly level? (*i.e.*, looking at how certain products may sell by different hours of the day.) If so, it makes sense to use 'hour' as the lowest level of granularity in the time dimension. If daily analysis is sufficient, then 'day' can be used as the lowest level of granularity. Since the lower the level of detail, the larger the data amount in the fact table, the granularity exercise is in essence figuring out the sweet spot in the tradeoff between detailed level of analysis and data storage.

Note that sometimes the users will not specify certain requirements, but based on the industry knowledge, the data warehousing team may foresee that certain requirements will be forthcoming that may result in the need of additional details. In such cases, it is prudent for the data warehousing team to design the fact table such that lower-level information is included. This will avoid possibly needing to re-design the fact table in the future. On the other hand, trying to anticipate **all** future requirements is an impossible and hence futile exercise, and the data warehousing team needs to fight the urge of the "dumping the lowest level of detail into the data warehouse" symptom, and only includes what is practically needed. Sometimes this can be more of an art than science, and prior experience will become invaluable here.

Fact And Fact Table Types

Types of Facts

There are three types of facts:

- **Additive:** Additive facts are facts that can be summed up through all of the dimensions in the fact table.
- **Semi-Additive:** Semi-additive facts are facts that can be summed up for some of the dimensions in the fact table, but not the others.
- **Non-Additive:** Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table.

Let us use examples to illustrate each of the three types of facts. The first example assumes that we are a retailer, and we have a fact table with the following columns:

Date
Store
Product
Sales_Amount

The purpose of this table is to record the sales amount for each product in each store on a daily basis. **Sales_Amount** is the fact. In this case, **Sales_Amount** is an additive fact, because you can sum up this fact along any of the three dimensions present in the fact table -- date, store, and product. For example, the sum of **Sales_Amount** for all 7 days in a week represents the total sales amount for that week.

Say we are a bank with the following fact table:

Date

Account
Current_Balance
Profit_Margin

The purpose of this table is to record the current balance for each account at the end of each day, as well as the profit margin for each account for each day. **Current_Balance** and **Profit_Margin** are the facts.

Current_Balance is a semi-additive fact, as it makes sense to add them up for all accounts (what's the total current balance for all accounts in the bank?), but it does not make sense to add them up through time (adding up all current balances for a given account for each day of the month does not give us any useful information).

Profit_Margin is a non-additive fact, for it does not make sense to add them up for the account level or the day level.

Types of Fact Tables

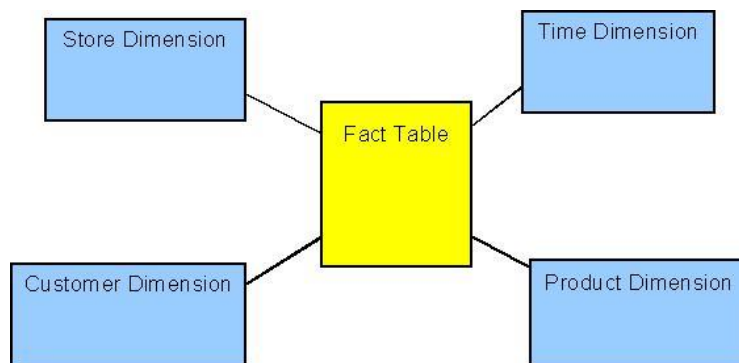
Based on the above classifications, there are two types of fact tables:

- **Cumulative:** This type of fact table describes what has happened over a period of time. For example, this fact table may describe the total sales by product by store by day. The facts for this type of fact tables are mostly additive facts. The first example presented here is a cumulative fact table.
- **Snapshot:** This type of fact table describes the state of things in a particular instance of time, and usually includes more semi-additive and non-additive facts. The second example presented here is a snapshot fact table.

Star Schema

In the star schema design, a single object (the [fact table](#)) sits in the middle and is radically connected to other surrounding objects (dimension lookup tables) like a star. Each dimension is represented as a single table. The primary key in each dimension table is related to a foreign key in the fact table.

Sample star schema



<http://www.1keydata.com/datawarehousing/star-schema.html>

All measures in the fact table are related to all the dimensions that fact table is related to. In other words, they all have the same level of granularity.

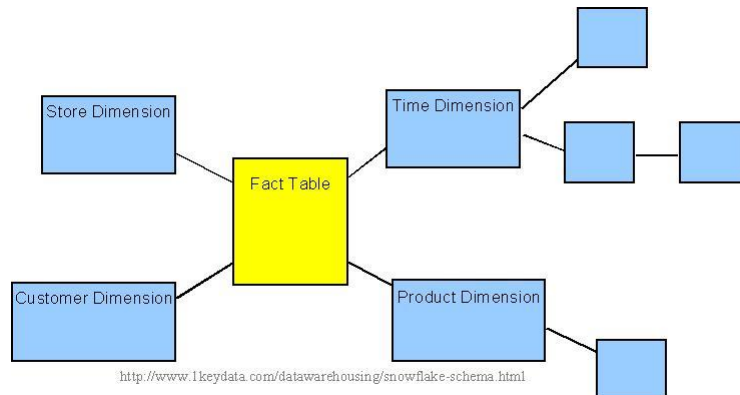
A star schema can be simple or complex. A simple star consists of one fact table; a complex star can have more than one fact table.

Let's look at an example: Assume our data warehouse keeps store sales data, and the different dimensions are time, store, product, and customer. In this case, the figure on the left represents our star schema. The lines

between two tables indicate that there is a primary key / foreign key relationship between the two tables. Note that different dimensions are not related to one another.

Snowflake Schema

The snowflake schema is an extension of the [star schema](#), where each point of the star explodes into more points. In a star schema, each dimension is represented by a single dimensional table, whereas in a snowflake schema, that dimensional table is normalized into multiple lookup tables, each representing a level in the dimensional hierarchy.



Sample snowflake schema

For example, the Time Dimension that consists of 2 different hierarchies:

1. Year → Month → Day
2. Week → Day

We will have 4 lookup tables in a snowflake schema: A lookup table for year, a lookup table for month, a lookup table for week, and a lookup table for day. Year is connected to Month, which is then connected to Day. Week is only connected to Day. A sample snowflake schema illustrating the above relationships in the Time Dimension is shown to the right.

The main advantage of the snowflake schema is the improvement in query performance due to minimized disk storage requirements and joining smaller lookup tables. The main disadvantage of the snowflake schema is the additional maintenance efforts needed due to the increase number of lookup tables.

Data Modeling - Conceptual, Logical, And Physical Data Models

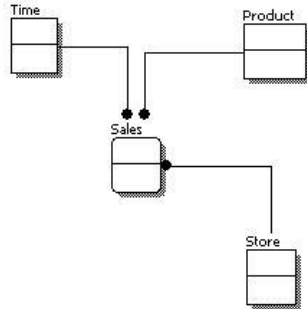
The three levels of data modeling, [conceptual data model](#), [logical data model](#), and [physical data model](#), were discussed in prior sections. Here we compare these three types of data models. The table below compares the different features:

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓

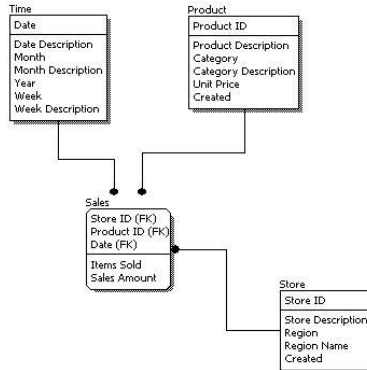
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Below we show the conceptual, logical, and physical versions of a single data model.

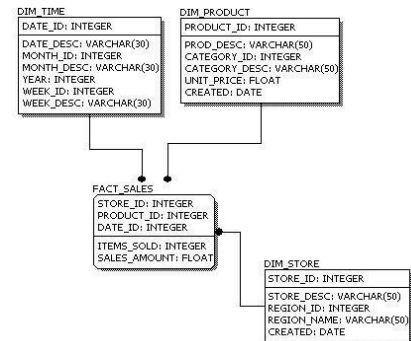
Conceptual Model Design



Logical Model Design



Physical Model Design



We can see that the complexity increases from conceptual to logical to physical. This is why we always first start with the conceptual data model (so we understand at high level what are the different entities in our data and how they relate to one another), then move on to the logical data model (so we understand the details of our data without worrying about how they will actually implemented), and finally the physical data model (so we know exactly how to implement our data model in the database of choice). In a data warehousing project, sometimes the conceptual data model and the logical data model are considered as a single deliverable.

Data Integrity

Data integrity refers to the validity of data, meaning data is consistent and correct. In the data warehousing field, we frequently hear the term, "Garbage In, Garbage Out." If there is no data integrity in the data warehouse, any resulting report and analysis will not be useful.

In a data warehouse or a data mart, there are three areas of where data integrity needs to be enforced:

Database level

We can enforce data integrity at the database level. Common ways of enforcing data integrity include:

Referential integrity

The relationship between the primary key of one table and the foreign key of another table must always be maintained. For example, a primary key cannot be deleted if there is still a foreign key that refers to this primary key.

Primary key / Unique constraint

Primary keys and the UNIQUE constraint are used to make sure every row in a table can be uniquely identified.

Not NULL vs. NULL-able

For columns identified as NOT NULL, they may not have a NULL value.

Valid Values

Only allowed values are permitted in the database. For example, if a column can only have positive integers, a value of '-1' cannot be allowed.

ETL process

For each step of the ETL process, data integrity checks should be put in place to ensure that source data is the same as the data in the destination. Most common checks include record counts or record sums.

Access level

We need to ensure that data is not altered by any unauthorized means either during the ETL process or in the data warehouse. To do this, there needs to be safeguards against unauthorized access to data (including physical access to the servers), as well as logging of all data access history. Data integrity can only be ensured if there is no unauthorized access to the data.

What Is OLAP

OLAP stands for On-Line Analytical Processing. The first attempt to provide a definition to OLAP was by Dr. Codd, who proposed 12 rules for OLAP. Later, it was discovered that this particular white paper was sponsored by one of the OLAP tool vendors, thus causing it to lose objectivity. The OLAP Report has proposed the FASMI test, **F**ast **A**nalysis of **S**hared **M**ultidimensional **I**nformation. For a more detailed description of both Dr. Codd's rules and the FASMI test, please visit [The OLAP Report](#).

For people on the business side, the key feature out of the above list is "Multidimensional." In other words, the ability to analyze metrics in different dimensions such as time, geography, gender, product, etc. For example, sales for the company are up. What region is most responsible for this increase? Which store in this region is most responsible for the increase? What particular product category or categories contributed the most to the increase? Answering these types of questions in order means that you are performing an OLAP analysis.

Depending on the underlying technology used, OLAP can be broadly divided into two different camps: MOLAP and ROLAP. A discussion of the different OLAP types can be found in the [MOLAP, ROLAP, and HOLAP](#) section.

MOLAP, ROLAP, And HOLAP

In the OLAP world, there are mainly two different types: Multidimensional OLAP (MOLAP) and Relational OLAP (ROLAP). Hybrid OLAP (HOLAP) refers to technologies that combine MOLAP and ROLAP.

MOLAP

This is the more traditional way of OLAP analysis. In MOLAP, data is stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats.

Advantages:

- Excellent performance: MOLAP cubes are built for fast data retrieval, and are optimal for slicing and dicing operations.
- Can perform complex calculations: All calculations have been pre-generated when the cube is created. Hence, complex calculations are not only doable, but they return quickly.

Disadvantages:

- Limited in the amount of data it can handle: Because all calculations are performed when the cube is built, it is not possible to include a large amount of data in the cube itself. This is not to say that the data in the cube cannot be derived from a large amount of data. Indeed, this is possible. But in this case, only summary-level information will be included in the cube itself.
- Requires additional investment: Cube technology are often proprietary and do not already exist in the organization. Therefore, to adopt MOLAP technology, chances are additional investments in human and capital resources are needed.

ROLAP

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.

Advantages:

- Can handle large amounts of data: The data size limitation of ROLAP technology is the limitation on data size of the underlying relational database. In other words, ROLAP itself places no limitation on data amount.
- Can leverage functionalities inherent in the relational database: Often, relational database already comes with a host of functionalities. ROLAP technologies, since they sit on top of the relational database, can therefore leverage these functionalities.

Disadvantages:

- Performance can be slow: Because each ROLAP report is essentially a SQL query (or multiple SQL queries) in the relational database, the query time can be long if the underlying data size is large.
- Limited by SQL functionalities: Because ROLAP technology mainly relies on generating SQL statements to query the relational database, and SQL statements do not fit all needs (for example, it is difficult to perform complex calculations using SQL), ROLAP technologies are therefore traditionally limited by what SQL can do. ROLAP vendors have mitigated this risk by building into the tool out-of-the-box complex functions as well as the ability to allow users to define their own functions.

HOLAP

HOLAP technologies attempt to combine the advantages of MOLAP and ROLAP. For summary-type information, HOLAP leverages cube technology for faster performance. When detail information is needed, HOLAP can "drill through" from the cube into the underlying relational data.

Bill Inmon vs. Ralph Kimball

In the data warehousing field, we often hear about discussions on where a person / organization's philosophy falls into Bill Inmon's camp or into Ralph Kimball's camp. We describe below the difference between the two.

Bill Inmon's paradigm: Data warehouse is one part of the overall business intelligence system. An enterprise has one data warehouse, and data marts source their information from the data warehouse. In the data warehouse, information is stored in 3rd normal form.

Ralph Kimball's paradigm: Data warehouse is the conglomerate of all data marts within the enterprise. Information is always stored in the dimensional model.

There is no right or wrong between these two ideas, as they represent different data warehousing philosophies. In reality, the data warehouse systems in most enterprises are closer to Ralph Kimball's idea. This is because most data warehouses started out as a departmental effort, and hence they originated as a data mart. Only when more data marts are built later do they evolve into a data warehouse.

Factless Fact Table

A factless fact table is a fact table that does not have any measures. It is essentially an intersection of dimensions. On the surface, a factless fact table does not make sense, since a fact table is, after all, about facts. However, there are situations where having this kind of relationship makes sense in data warehousing.

For example, think about a record of student attendance in classes. In this case, the fact table would consist of 3 dimensions: the student dimension, the time dimension, and the class dimension. This factless fact table would look like the following:

FACT_ATTENDANCE

STUDENT_ID
CLASS_ID
TIME_ID

The only measure that you can possibly attach to each combination is "1" to show the presence of that particular combination. However, adding a fact that always shows 1 is redundant because we can simply use the COUNT function in SQL to answer the same questions.

Factless fact tables offer the most flexibility in data warehouse design. For example, one can easily answer the following questions with this factless fact table:

- How many students attended a particular class on a particular day?
- How many classes on average does a student attend on a given day?

Without using a factless fact table, we will need two separate fact tables to answer the above two questions. With the above factless fact table, it becomes the only fact table that's needed.

Junk Dimension

In data warehouse design, frequently we run into a situation where there are yes/no indicator fields in the source system. Through business analysis, we know it is necessary to keep such information in the fact table. However, if keep all those indicator fields in the fact table, not only do we need to build many small dimension tables, but the amount of information stored in the fact table also increases tremendously, leading to possible performance and management issues.

Junk dimension is the way to solve this problem. In a junk dimension, we combine these indicator fields into a single dimension. This way, we'll only need to build a single dimension table, and the number of fields in the fact

table, as well as the size of the fact table, can be decreased. The content in the junk dimension table is the combination of all possible values of the individual indicator fields.

Let's look at an example. Assuming that we have the following fact table:

FACT_TABLE

CUSTOMER_ID
PRODUCT_CD
TXN_ID
STORE_ID
TXN_CODE
COUPON_IND
PREPAY_IND
TXN_AMT

In this example, the last 3 fields are all indicator fields. In this existing format, each one of them is a dimension. Using the junk dimension principle, we can combine them into a single junk dimension, resulting in the following fact table:

FACT_TABLE

CUSTOMER_ID
PRODUCT_CD
TXN_ID
STORE_ID
JUNK_ID
TXN_AMT

Note that now the number of dimensions in the fact table went from 7 to 5.

The content of the junk dimension table would look like the following:

DIM_JUNK

JUNK_ID	TXN_CODE	COUPON_IND	PREPAY_IND
1	1	Y	Y
2	2	Y	Y
3	3	Y	Y
4	1	Y	N
5	2	Y	N
6	3	Y	N
7	1	N	Y
8	2	N	Y
9	3	N	Y
10	1	N	N
11	2	N	N
12	3	N	N

In this case, we have 3 possible values for the TXN_CODE field, 2 possible values for the COUPON_IND field, and 2 possible values for the PREPAY_IND field. This results in a total of $3 \times 2 \times 2 = 12$ rows for the junk dimension table.

By using a junk dimension to replace the 3 indicator fields, we have decreased the number of dimensions by 2 and also decreased the number of fields in the fact table by 2. This will result in a data warehousing environment that offer better performance as well as being easier to manage.

Conformed Dimension

A conformed dimension is a dimension that has exactly the same meaning and content when being referred from different fact tables. A conformed dimension can refer to multiple tables in multiple data marts within the same organization. For two dimension tables to be considered as conformed, they must either be identical or one must be a subset of another. There cannot be any other type of difference between the two tables. For example, two dimension tables that are exactly the same except for the primary key are not considered conformed dimensions.

Why is conformed dimension important? This goes back to the [definition of data warehouse](#) being "integrated." Integrated means that even if a particular entity had different meanings and different attributes in the source systems, there must be a single version of this entity once the data flows into the data warehouse.

The time dimension is a common conformed dimension in an organization. Usually the only rule to consider with the time dimension is whether there is a fiscal year in addition to the calendar year and the definition of a week. Fortunately, both are relatively easy to resolve. In the case of fiscal vs. calendar year, one may go with either fiscal or calendar, or an alternative is to have two separate conformed dimensions, one for fiscal year and one for calendar year. The definition of a week is also something that can be different in large organizations: Finance may use Saturday to Friday, while marketing may use Sunday to Saturday. In this case, we should decide on a definition and move on. The nice thing about the time dimension is once these rules are set, the values in the dimension table will never change. For example, October 16th will never become the 15th day in October.

Not all conformed dimensions are as easy to produce as the time dimension. An example is the customer dimension. In any organization with some history, there is a high likelihood that different customer databases exist in different parts of the organization. To achieve a conformed customer dimension means those data must be compared against each other, rules must be set, and data must be cleansed. In addition, when we are doing incremental data loads into the data warehouse, we'll need to apply the same rules to the new values to make sure we are only adding truly new customers to the customer dimension.

Building a conformed dimension also part of the process in [master data management](#), or MDM. In MDM, one must not only make sure the master data dimensions are conformed, but that conformity needs to be brought back to the source systems.

Logical Data Model

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

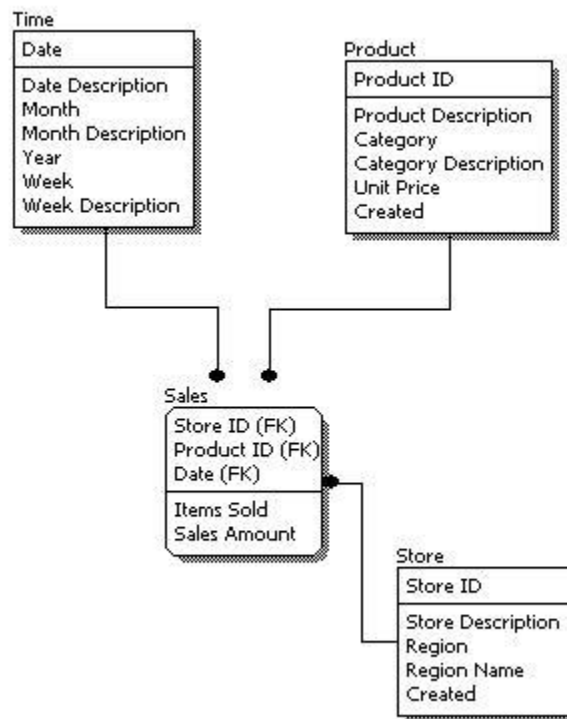
- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

The figure below is an example of a logical data model.

Logical Data Model



Comparing the logical data model shown above with the [conceptual data model](#) diagram, we see the main differences between the two:

- In a logical data model, primary keys are present, whereas in a conceptual data model, no primary key is present.

- In a logical data model, all attributes are specified within an entity. No attributes are specified in a conceptual data model.
- Relationships between entities are specified using primary keys and foreign keys in a logical data model. In a conceptual data model, the relationships are simply stated, not specified, so we simply know that two entities are related, but we do not specify what attributes are used for this relationship.

Physical Data Model

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

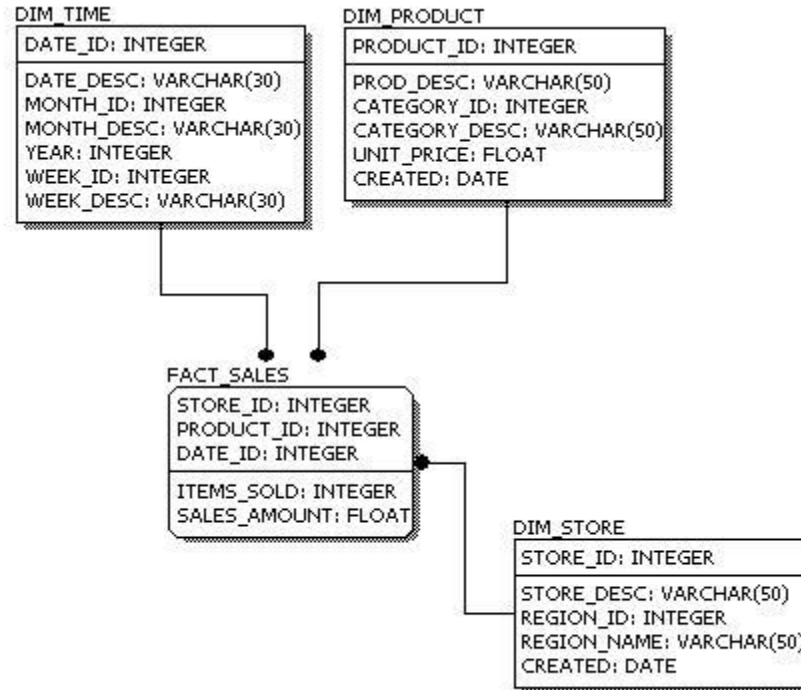
- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

The steps for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

The figure below is an example of a physical data model.

Physical Data Model



Comparing the logical data model shown above with the [logical data model](#) diagram, we see the main differences between the two:

- Entity names are now table names.
- Attributes are now column names.
- Data type for each column is specified. Data types can be different depending on the actual database being used.