**LECTURE 2**

**INTRODUCTION TO INFORMATION ARCHITECTURE AND DESIGN–PART III**

**Learning Goals**

It is a blend of many technologies, the basic concept being:

Take all data from different operational systems.
If necessary, add relevant data from industry.
Transform all data and bring into a uniform format.
Integrate all data as a single entity.
Store data in a format supporting easy access for decision support.
Create performance enhancing indices.
Implement performance enhancement joins.
Run ad-hoc queries with low selectivity.

**What is a Data Warehouse?**

A Data Warehouse is not something shrink-wrapped i.e. you take a set of CDs and install into a box and soon you have a Data Warehouse up and running. A Data Warehouse evolves over time, you don't buy it. Basically it is about taking/collecting data from different heterogeneous sources. Heterogeneous means not only the operating system is different but so is the underlying file format, different databases, and even with same database systems different representations for the same entity. This could be anything from different columns names to different data types for the same entity.

Companies collect and record their own operational data, but at the same time they also use reference data obtained from external sources such as codes, prices etc. This is not the only external data, but customer lists with their contact information are also obtained from external sources. Therefore, all this external data is also added to the data warehouse.

As mentioned earlier, even the data collected and obtained from within the company is not standard for a host of different reasons. For example, different operational systems being used in the company were developed by different vendors over a period of time, and there is no or minimal evenness in data representation etc. When that is the state of affairs (and is normal)

© St Paul's University

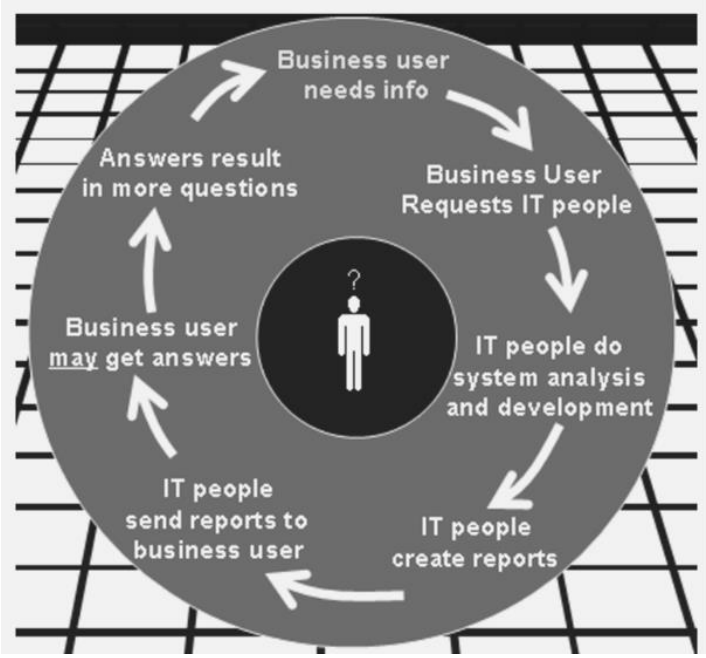
within a company, then there is no control on the quality of data obtained from external sources. Hence all the data has to be transformed into a uniform format, standardized and integrated before it can go into the data warehouse. In a decision support environment, the end user i.e. the decision maker is interested in the big picture. Typical DSS queries do not involve using a primary key or asking questions about a particular customer or account. DSS queries deal with number of variables spanning across number of tables (i.e. join operations) and looking at lots of historical data. As a result large numbers of records are processed and retrieved. For such a case, specialized or different database architectures/topologies are required, such as the star schema. We will cover this in detail in the relevant lecture. Recall that a B-Tree is a data structure that supports dictionary operations. In the context of a database, a B-Tree is used as an index that provides access to records without actually scanning the entire table. However, for very large databases the corresponding B- Trees becomes very large. Typically the node of a B-Tree is stored in a memory block, and traversing a B-Tree involves O(log n) page faults. This is highly undesirable, because by default the height of the B-Tree would be very large for very large data bases. Therefore, new and unique indexing techniques are required in the DWH or DSS environment, such as bitmapped indexes or cluster index etc. In some cases the designers want such powerful indexing techniques, that the queries are satisfied from the indexes without going to the fact tables. In typical OLTP environments, the size of tables are relatively small, and the rows of interest are also very small, as the queries are confined to specifics. Hence traditional joins such as nested-loop join of quadratic time complexity does not hurt the performance i.e. time to get the answer. However, for very large databases when the table sizes are in millions and the rows of interest are also in hundreds of thousands, nested-loop join becomes a bottle neck and is hardly used. Therefore, new and unique forms of joins are required such as sort-merge join or hash based join etc. Run Ad-Hoc queries with low Selectivity Have already explained what is meant by ad-hoc queries. A little bit about selectivity is in order. Selectivity is the ratio between the numbers of unique values in a column divided by the total number of values in that column. For example the selectivity of the gender column is 50%, assuming gender of all customers is known. If there are N records in a table, then the selectivity of the primary key column is 1/N. Note that a query consists of retrieving records based on a a combination of different columns, hence the choice of columns determine the selectivity of the query i.e. the number of records retrieved divided by the total number of records present in the table. In an OLTP (On-Line

© St Paul's University

Transaction Processing) or MIS (Management Information System) environment, the queries are typically Primary Key (PK) based, hence the number of records retrieved is not more than a hundred rows. Hence the selectivity is very high. For a Data Warehouse (DWH) environment, we are interested in the big picture and have queries that are not very specific in nature and hardly use a PK. As a result hundreds of thousands of records (or rows) are retrieved from very large tables. Thus the ratio of records retrieved to the total number of records present is high, and hence the selectivity is low.

**How is it different?**

Decision making is Ad-Hoc



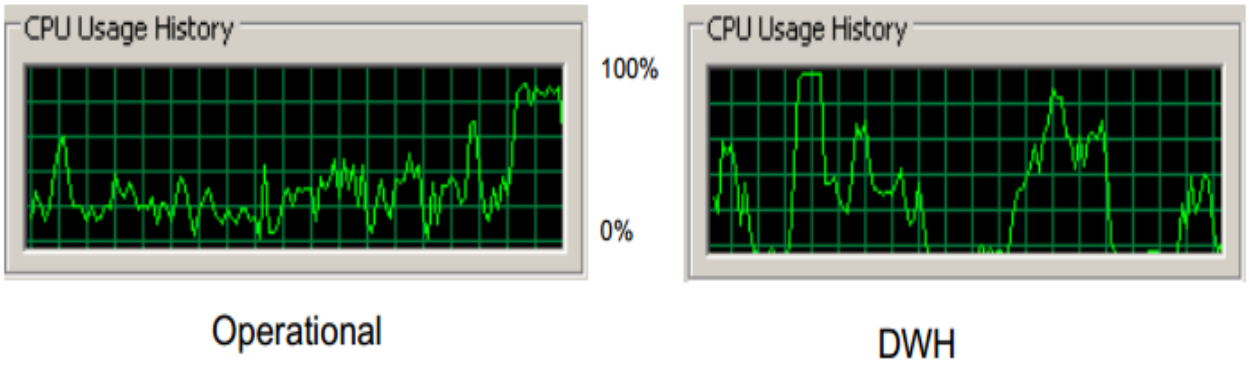

**Running in circles**

Consider a decision maker or a business user who wants some of his questions to be answered. He/she sets a meeting with the IT people, and explains the requirements. The IT people go over the cycle of system analysis and design, that takes anywhere from couple of weeks to couple of months and they finally design and develop the system. Happy and proud with their achievement the IT people go to the business user with the reporting system or MIS system. After a learning curve the business users spends some time with the brand new system, and may get some answers to the required questions. But then those answers results in more questions. The business

© St Paul's University

user has no choice to meet the IT people with a new set of requirements. The business user is frustrated that his questions are not getting answered, while the IT people are frustrated that the business user always changes the requirements. Both are correct in their frustration.

**Different patterns of hardware utilization**




**Different patterns of CPU Usage**

Although there are peaks and valleys in the operational processing, but ultimately there is relatively static pattern of utilization. There is an essentially different pattern of hardware utilization in the data warehouse environment i.e. a binary pattern of utilization; either the hardware is utilized fully or not at all. Calculating a mean utilization for a DWH is not a meaningful activity. Therefore, trying to mix the two environments is a recipe for disaster. You can optimize the machine for the performance of one type of application, not for both.

**Bus vs. Train Analogy**

Consider the analogy of a bus and train. I believe you can find dozens of buses operating between Lahore and Rawalpindi almost every 30 minutes. As a consequence, literally there are buses moving between Lahore and Rawalpindi almost continuously throughout the day. But how many times a dedicated train moves between the two cities? Only twice a day and carries a bulk of passengers and cargo. Binary operation i.e. either traveling or not. The train can NOT be optimized for every 30-min travel; it will never fill to capacity and run into loss. A bus cannot be optimized to travel only twice, it will stand idle and passengers would take vans etc. Bottom line: Two modes of transportation cannot be interchanged.

© St Paul's University

**Combines historical & Operational Data**

- Don't do data entry into a DWH, OLTP or ERP are the source systems.
- OLTP systems don't keep history, can't get balance statement more than a year old.
- DWH keep historical data, even of bygone customers. Why?
- In the context of bank, want to know why the customer left?
- What were the events that led to his/her leaving? Why?
- Customer retention.

**Why keep historical data?**

The data warehouse is different because, again it's not a database you do data entry. You are actually collecting data from the operational systems and loading into the DWH. So the transactional processing systems like the ERP system are the source systems for the data warehouse. You feed the data into the data warehouse. And the data warehouse typically collects data over a period of time. So if you look at your transactional processing OLTP systems, normally such systems don't keep very much history. Normally if a customer leaves or expired, the OLTP systems typically purge the data associated with that customer and all the transactions off the database after some amount of time. So normally once a year most business will purge the database of all the old customers and old transactions. In the data warehouse we save the historical data. Because you don't need historical data to do business today, but you do need the historical data to understand patterns of business usage to do business tomorrow, such why a customer left?

How much History?

A. Depends on:

1. *f* Industry.
2. *f* Cost of storing historical data.
3. *f* Economic value of historical data.

B. Industries and history

1. *f* Telecomm calls are much more as compared to bank transactions- 18 months of historical data.
2. *f* Retailers interested in analyzing yearly seasonal patterns- 65 weeks of historical data.

© St Paul's University

3. *f* Insurance companies want to do actuary analysis, use the historical data in order to predict risk- 7 years of historical data.

**Hence, a DWH NOT a complete repository of data**

How back do you look historically? It really depends a lot on the industry. Typically it's an economic equation. How far back depends on how much does it cost to store that extra year's work of data and what is its economic value? So for example in financial organizations, they typically store at least 3 years of data going backward. Again it's typical. It's not a hard and fast rule. In a telecommunications company, for example, typically around 18 months of data is stored. Because there are a lot more call details records then there are deposits and withdrawals from a bank account so the storage period is less, as one cannot afford to store as much of it typically. Another important point is, the further back in history you store the data, the less value it has normally. Most of the times, most of the access into the data is within that last 3 months to 6 months. That's the most predictive data. In retail business, retailers typically store at least 65 weeks of data. Why do they do that? Because they want to be able to look at this season's selling history to last season's selling history. For example, if it is Eid buying season, I want to look at the transit-buying this Eid and compare it with the year ago. Which means I need 65 weeks in order to get year going back, actually more than a year? It's a year and a season. So 13 weeks are additionally added to do the analysis. So it really depends a lot on the industry. But normally you expect at least 13 months.

**Economic value of data Vs. Storage cost & Data Warehouse a complete repository of data?**

This raises an interesting question, do we decide about storage of historical data using only time, or consider space also, or both?

Usually (but not always) periodic or batch updates rather than real-time.

A. The boundary is blurring for active data warehousing.
B. For an ATM, if update not in real-time, then lot of real trouble.
C. DWH is for strategic decision making based on historical data. Won't hurt if transactions of last one hour/day are absent. Rate of update depends on:
    1. Volume of data,
    2. Nature of business,

© St Paul's University

3. Cost of keeping historical data,
4. Benefit of keeping historical data.

It's also true that in the traditional data warehouse the data acquisition is done on periodic or batch based, rather than in real time. So think again about ATM system, when I put my ATM card and make a withdrawal, the transactions are happening in real time, because if they don't the bank can get into trouble. Someone can withdraw more money than they had in their account! Obviously that is not acceptable. So in an online transaction processing (OLTP) system, the records are updated, deleted and inserted in real-time as the business events take place, as the data entry takes place, as the point of sales system at a super market captures the sales data and inserts into the database. In a traditional data warehouse that is not true, because the traditional data warehouse is for strategic decision-making not for running day to day business. And for strategic decision making, I don't need to know the last hour's worth of ATM deposits. Because strategic decisions take the long term perspective. For this reason and for efficiency reasons normally what happens is that in the data warehouse you update on some predefined schedule basis. May be its once a month, maybe it's once a week, maybe it's even once every night. It depends on the volume of data you are working with, and how important the timings of the data are and so on.

**Deviation from the PURIST approach**

Let me first explain what/who a purist is. A purist is an idealist or traditionalist who wants everything to be done by the book or the old arcane ways (only he/she knows), in short he/she is not pragmatic or realist. Because the purist wants everything perfect, so he/she has good excuses of doing nothing, as it is not a perfect world. When automobiles were first invented, it was the purists who said that the automobiles will fail, as they scare

The horses. As data warehouses become main-stream and the corresponding technology also becomes mainstream technology, some traditional attributes are being deviated in order to meet the increasing demands of the user's. We have already discussed and reconciled with the fact that a data warehouse is NOT the complete repository of data. The other most noticeable deviations being time variance and non-volatility. Deviation from Time Variance and No volatility

As the size of data warehouse grows over time (e.g., in terabytes), reloading and appending data

© St Paul's University

can become a very tedious and time consuming task. Furthermore, as business users get the "hang of it" they start demanding that more up-to-date data be available in the data warehouse. Therefore, instead of sticking to the traditional data warehouse characteristic of keeping the data nonvolatile and time variant, new data is being added to the data warehouse on a daily basis, if not on a real-time basis and at the same time historical data removed to make room for the "fresh" data. Thus, new approaches are being made to tackle this task. Two possible methods are as follows:

Perform hourly/daily batch updates from shadow tables or log files.
Transformation rules are executed during the loading process. Thus, when the data reaches the target data warehouse database, it is already transformed, cleansed and summarized.

Perform real-time updates from shadow tables or log files. Again, transformation rules are executed during the loading process. Instead of batch updates, this takes place on a per transaction basis that meets certain business selection criteria.

**Introduction to information architecture and design –Part IV**

**Leaning Goals**

1. Decision makers typically don't work 24 hrs a day and 7 days a week. An ATM system does.
2. Once decision makers' start using the DWH, and start reaping the benefits, they start liking it…
3. Starts using the DWH more often, till want it available 100% of the time.
4. For business across the globe, 50% of the world may be sleeping at any one time, but the businesses are up 100% of the time.
5. 100% availability not a trivial task, need to take into loading strategies, refresh rates etc.

Starts with a 6x12 availability requirement ... but 7x24 usually becomes the goal.

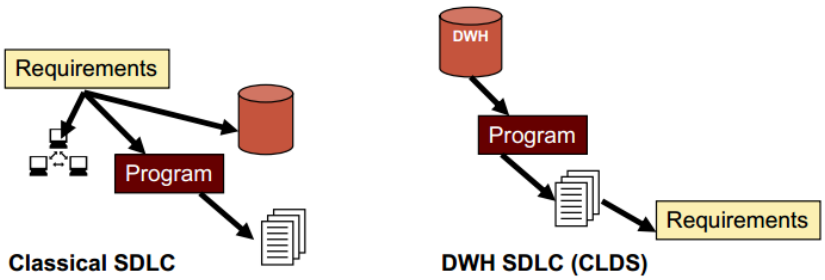
If we look at the availability requirements for a data warehouse, normally you start out with 6 days a week, 12 hours a day. The decision makers are not going to come during the middle of a weekend to ask marketing questions. They don't normally do that. They don't come in midnight to ask these questions either. However, a transaction processing system, like the ATM systems,

© St Paul's University

should be available all the time. I can go to an ATM at midnight if I want to, and withdraw money. That's not usually the requirement in the beginning of the data warehouse project, but as the data warehouse matures, availability becomes much more important. Because the decision makers start accessing and using the data more often. They start doing maybe interactive kind of analysis in the day and data mining kind of things over night. We will talk more about data mining later. So you can start out with

6 *12 availability but will usually evolve in to 7*24 over a period of time. So one of the things we are going to talk about, although there will not be a major focus is about the tradeoffs that you make in availability. How do I design my data warehouse for 100% availability? So that it is always available for quires. And it turns out that there are very subtle interactions between availability and data loading. How do I make sure the data is available for querying while uploading the data? These issues turn out to have some tricky subtleties in that kind of environment.

**Does not follows the traditional development model**




**Classical SDLC**

- Requirements gathering
- Analysis
- Design
- Programming
- Testing
- Integration
- Implementation

**DWH SDLC (CLDS)**

- Implement warehouse
- Integrate data
- Test for biasness
- Program w.r.t data
- Design DSS system
- Analyze results
- Understand requirement

**Comparison of SDLC & CLDS**

Operational environment is created using the classical systems development life cycle. The DWH on the other hand operates under a very different life cycle, sometimes called CLDS i.e. the reverse of SDLC. The classical SDLC is requirement driven. In order to build the system, you

© St Paul's University

must first understand the end user or business user requirements. Then you go into the stages of design and development typically taught in software engineering. The CLDS is almost exactly the reverse of SDLC. The CLDS starts with the data. Once the data is in hand, it is integrated, and then tested to see what bias there is, if any. Programs are then written against the data. The results are analyzed, and finally the requirements of the system are understood. Some Data Warehouse developers/vendors overuse this "natural" approach and add extensive charges for adding features or enhancing features already present once the end users get the "hang" of the system. The CLDS is a classical data driven development life cycle. Trying to apply inappropriate tools and techniques of development will only result in waste of effort/resources and confusion

**Typical Queries**

**OLTP (On Line Transaction Processing) specific query**

Select tx_date, balance from tx_table

Where account_ID = 23876;

DWH specific query

Select balance, age, sal, gender from customer_table and tx_table

Where age between (30 and 40) and

Education = 'graduate' and CustID.customer_table = Customer_ID.tx_table;

Let's take a brief look at the two queries, the results of comparing them are summarized in table 4.1 below:

© St Paul's University

| OLTP | DWH |
|---|---|
| Primary key used | Primary key NOT used |
| No concept of Primary Index | Primary index used |
| May use a single table | Uses multiple tables |
| Few rows returned | Many rows returned |
| High selectivity of query | Low selectivity of query |
| Indexing on primary key (unique) | Indexing on primary index (non-unique) |

**Table-4.1: Comparison of OLTP and DWH for given queries**

| | Data Warehouse | OLTP |
|---|---|---|
| **Scope** | * Application –Neutral<br>* Single source of "truth"<br>* Evolves over time<br>* How to improve business | * Application specific<br>* Multiple databases with repetition<br>* Off the shelf application<br>* Runs the business |
| **Data Perspective** | * Historical, detailed data<br>* Some summary<br>* Lightly denormalized | * Operational data<br>* No summary<br>* Fully normalized |
| **Queries** | * Hardly uses PK<br>* Number of results returned in thousands | * Based on PK<br>* Number of results returned in hundreds |
| **Time factor** | * Minutes to hours<br>* Typical availability 6x12 | * Sub seconds to seconds<br>* Typical availability 24x7 |

**Detailed comparison of OLTP and DWH**

**Comparisons of response times**

1. On-line analytical processing (OLAP) queries must be executed in a small number of seconds.
2. Often requires demoralizations and/or sampling. Complex query scripts and large list selections can generally be executed in a small number of minutes.
3. Sophisticated clustering algorithms (e.g., data mining) can generally be executed in a small number of hours (even for hundreds of thousands of customers).

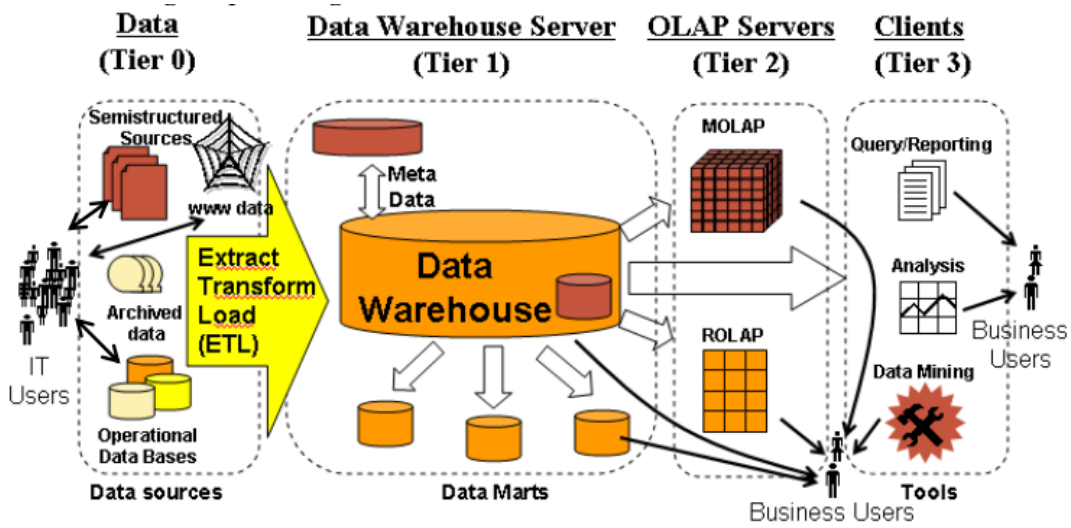
So there is one class of decision support environment called OLAP (Online Analytical Processing). The idea is that I am doing iterative decision making, using point and clicking, drilling down into data, and refining my decision model and I should be able to "execute" queries in an OLAP environment in a small number of seconds. How it is possible to execute queries in small numbers of seconds? And potentially working with billions of rows of data? I have to be a bit clever here. I have to consider special techniques like sampling, like de-normalization, special indexing techniques and other smart kinds of techniques. We will talk about those techniques when we go to the relevant parts. And how do I design a database which has these

© St Paul's University

characteristics?

If I have got complex combination of tables and a large list selection and if it takes small number of minutes that's acceptable. If I got a lot of iterative analysis to do by pulling a list of names and customers and some information about them to send a direct mail to those customers, then it is acceptable to assume that that such queries don't have to be executed in few seconds. Of course as long as I am going to find them, it's all right.

**Putting the pieces together**




**Putting the pieces together**

Figure 4.2 gives a complete picture of how different "pieces" of a data warehouse fit together. We start with extracting data from different sources, transform and clean that data and then load it into the data warehouse. The data warehouse itself has to have specialized schemas to give quick answers to typical macroscopic queries. Data marts are created out of the data warehouse to service the needs of different departments such as marketing, sales etc such that they don't have to work with the heavy load of the complete data warehouse. Once the data warehouse is in place (as briefly discussed before) data cubes are created for OLAP based drill-down "all" possible queries. And we move further and use smart tools such as data mining clustering and classification techniques.

© St Paul's University

**Why this is hard?**

1. Data sources are unstructured & heterogeneous.
2. Requirements are always changing.
3. Most computer scientist trained on OTLP systems, those concepts not valid for VLDB & DSS.
4. The scale factor in VLDB implementations is difficult to comprehend.
5. Performance impacts are often non-linear O(n) Vs. O(nlog_n) e.g. scanning vs. indexing
6. Complex computer/database architectures.
7. Rapidly changing product characteristics.
8. And so on...

In early days of data warehousing you would never even conceive giving business end user access to the data. They would always have to go through a data center, some kind of reporting or MIS system, that's really not true anymore. The hard part of the problem now is architecting, designing and building these systems. And it is particularly hard at the high end. Because in a data warehouse environment, there are no stable requirements- change is constant. This is one of the main reasons why SDLC system does not work. By the time you collect all the requirements following the SDLC approach and start designing the system the requirements may have completely changed. They are completely meaningless. Because any question that was interesting 6 months ago is definitely not interesting today.

The other issue is with very large databases scale dose strange things in the environment. And this is very important for computer scientist to understand because at small scale (i.e. hundreds of rows) an O (n log n) algorithm and an O(n2 ) algorithms are not much different in terms of performance. It does not matter if you are using bubble sort or a heap sort for small amount of data; nobody cares as the difference in performance is not noticeable. However, when you get millions of records or a billion records, then it starts to hurt. So when you get very large data bases, and manipulating large amount of data because we are retaining history, this becomes a big problem and a proper design is required to be in place. Else the scale will kill you. The difference between O (n log n) and an O (n2) is huge when you get to billions of records.

© St Paul's University

**High level implementation steps**

**Phase-I**

1. Determine Users' Needs
2. Determine DBMS Server Platform
3. Determine Hardware Platform
4. Information & Data Modeling
5. Construct Metadata Repository

**Phase-II**

1. Data Acquisition & Cleansing
2. Data Transform, Transport & Populate
3. Determine Middleware Connectivity
4. Prototyping, Querying & Reporting
5. Data Mining
6. On Line Analytical Processing (OLAP)

**Phase-III**

1. Deployment & System Management

If you look at the high level implementations steps for a data warehouse it is important that you are driving the design of the data warehouse in the context of the data warehouse by the business requirements, and not driven by the technology.

So you start with the business requirements and say ok what problems I am trying to solve here. Am I going to do fraud detection or do customer retention analysis? What are the specifications of the problems that we have discussed? And identify the key source of these problems so you can understand what is going to be the cost and time required to fix them. But make sure this is done in the context of a logical data model that expresses the underlying business relationship of the data.

The data warehouse should be able to support multiple applications, multiple workloads against a single source of truth for the organization. So by identifying the business opportunity, you identify the information required, and then in the next stage, how to schedule those deliverables that are most important to the business. Ask yourself, what can I do in a 90 days' time period to

© St Paul's University

deliver values to the business? And then based on what I have decided here do the software and hardware selection. Remember that software is hardware, and hardware is easy ware. Because the software is much more complex to scale (algorithm complexity) then the hardware. Hardware is getting cheaper over time. So you drive typically from the software not the hardware

© St Paul's University