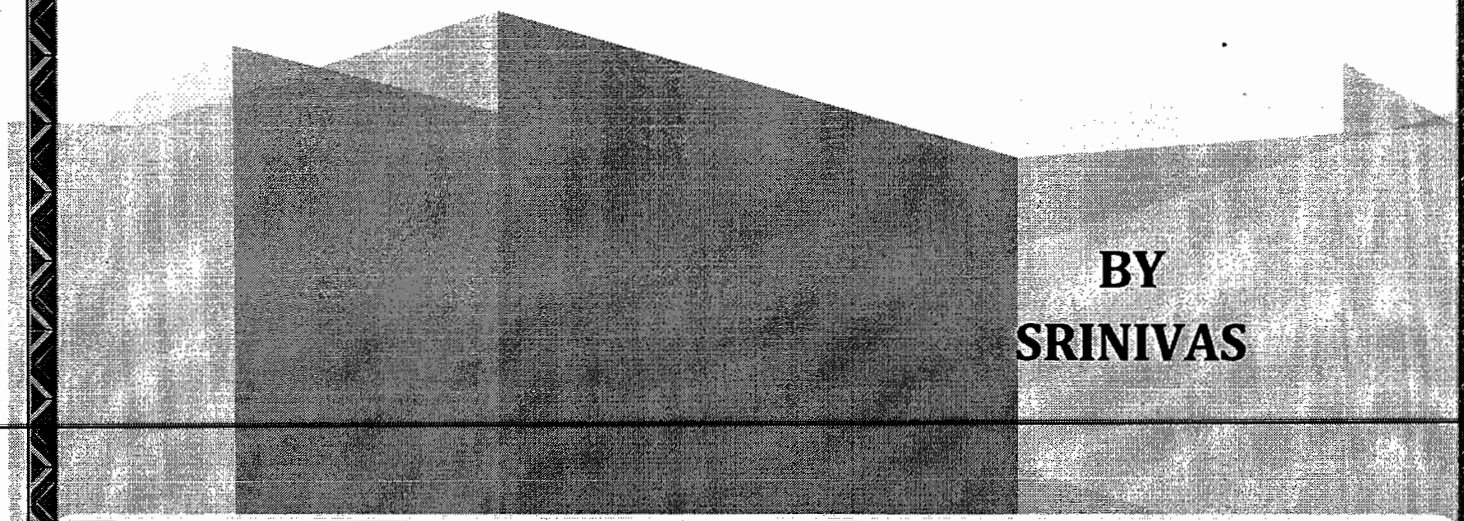
QUALITY THOUGHT
TECHNOLOGIES

ETL
TESTING

(Material)



BY
SRINIVAS

NEAR S.R.NAGAR BUS STOP, OPP. SINDHU TRAVELS, S.R.NAGAR

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

Data warehousing concepts

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

What is Data:

Data is collection of raw material in unorganized format, which refers an object.

Data:

- ✓ Items that are the most elementary descriptions of things, events, activities, and transactions
- ✓ May be internal or external

Information:

- ✓ Organized data that has meaning and value

Knowledge:

- ✓ Processed data or information that conveys understanding or learning applicable to a problem or activity

In Data Warehouse data is converted in to information format to get knowledge for making decisions.

Data---→Information------→Knowledge

Why DWH implemented:

Enterprise: Enterprise is integration of various departments which is working for business. Different departments will work on different transactions, we need to store all those transactions. Those all transactions are stored in a database called as Operational Data Store (ODS).

The main characteristic of ODS is data is volatile means data is changed randomly, to kept historical data is not possible in ODS. For that reason DWH is implemented.

DWH is also called as **Decision Support System** (DSS) because we use DWH to make decisions and also called as **Historical Database** because DWH stores historical data and **Read Only Database** because of we can just read and analyze the data.

Who required DWH:

Business Analyst: He is working for organization to make decisions. To make decision definitely he required historical data because while making decisions he should compare present data and historical data. But ODS is unable to store historical data, so a container is developed called as DWH. Main users for DWH are Business analysts,CEO's,High and Middle level management in the company.

Why we need DWH:

- ➢ To Integrate Data from multiple diverse sources
- ➢ Allows for analysis of data over time
- ➢ Provides ad-hoc querying, reporting and analysis capabilities to decisions makers

By using DWH users can resolve below questions and based on results he will take decisions.

- ➢ Which are our lowest/highest margin customers?
- ➢ What is the most effective distribution channel?
- ➢ Who are my customers and what products are they buying?

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

- What product promotions have the biggest impact on revenue?
- Which customers are most likely to go to the competition?

What is DWH:

- A DWH is integration of data from different sources, that data is used for analysis to make managerial decisions.
- A data warehouse is a relational database that is designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data.
- Data Warehouse is a subject-oriented, integrated, nonvolatile and time-variant collection of data in support of management's decisions.

Characteristics of DWH:

Subject-oriented — Data warehouse data are organized around major subject areas such as sales, claims, shipments, and enrollments. For example, a data warehouse for sales contains historical records of sales over specific time intervals.

Integrated — A data warehouse provides the facility for integration in a heterogeneous, fragmented environment of independent application systems, where the data is stored in multiple, incompatible formats. For example, a department store may have information about the same customers stored in several databases using different format representations. The data warehouse brings the data together into a single representation.

Time-variant — The data warehouse organizes and stores the data needed for informational and analytical processing over an extended historical time range. For example, a marketing analyst can analyze the sales history of five years from the information that was collected at the end of each year.

Non-volatile — Changes to the data warehouse environment occur in a controlled and scheduled manner, unlike the more volatile OLTP environment in which updates continually occur. A similar query run in five minute intervals in an OLTP environment may yield different results, while the same query run within the data warehouse should remain stable and consistent. For example, an airline may capture frequent flyer information in its data warehouse. During check-in for a flight, the additional mileage for a specific passenger is immediately updated in the OLTP system, but is not yet reflected in the data warehouse until its next scheduled load.

Difference between ODS & DWH:

OLTP (ODS)	OLAP (DWH)
Application Oriented	Subject Oriented
Used to run business	Used to analyze business
Detailed data	Summarized and refined
Current up to date	Snapshot data
Isolated Data	Integrated Data
Repetitive access	Ad-hoc access

Clerical User	Knowledge User (Manager)
Performance Sensitive	Performance relaxed
Few Records accessed at a time (tens)	Largevolumes accessed at a time(millions)
Read/Update Access	Mostly Read (Batch Update)
No data redundancy	Redundancy present
Database Size 100MB -100 GB	Database Size 100 GB - few terabytes

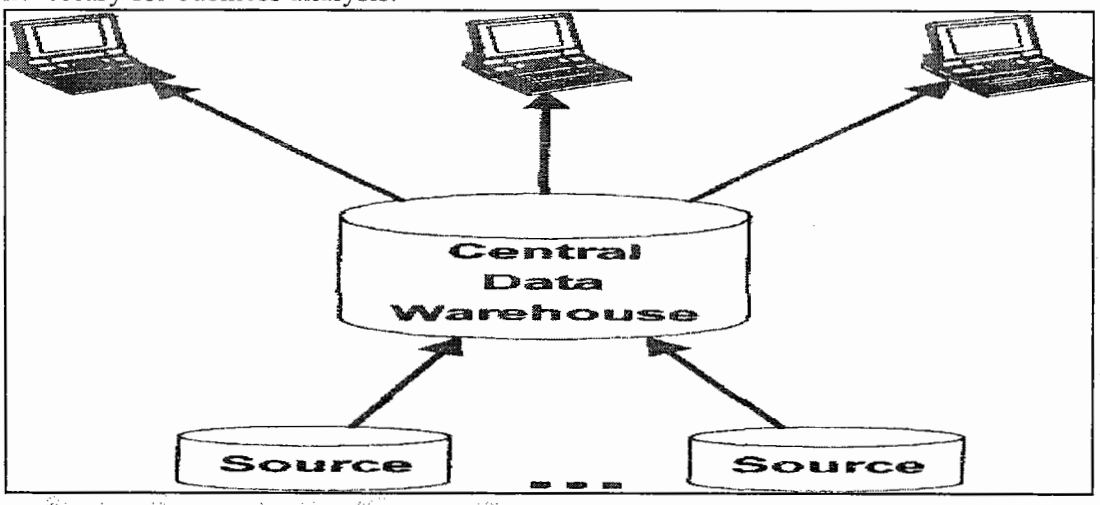
DWH architecture:

We are having 3 kinds of architecture for DWH.

1. Centralized architecture
2. Federated architecture
3. Tiered architecture

Centralized architecture:

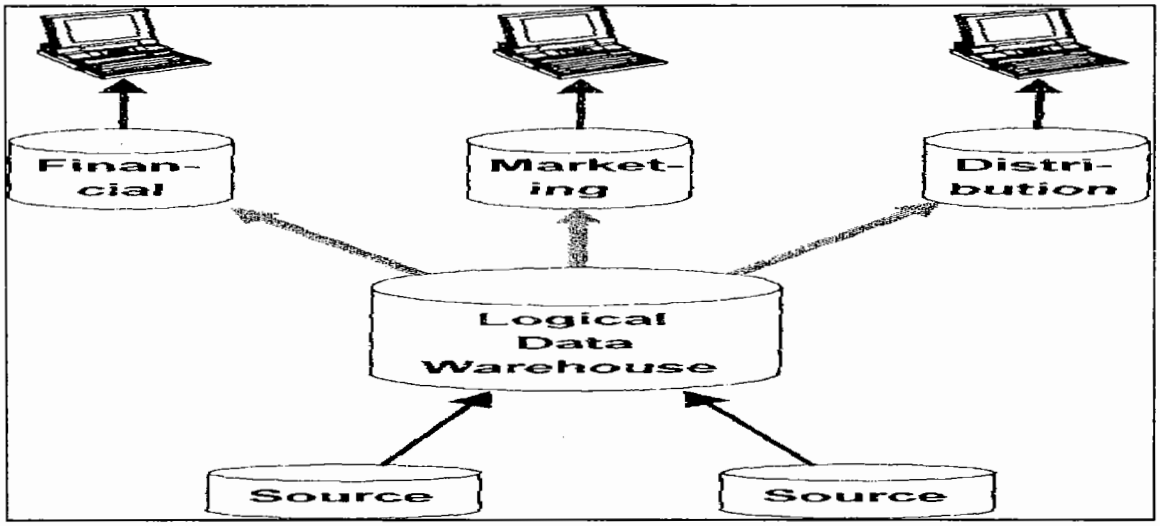
In a centralized architecture, there exists only one data warehouse which stores all data necessary for business analysis.



Federated architecture:

In a federated architecture the data is logically consolidated but stored in separate physical databases, at the same or at different physical sites. The local data marts store only the relevant information for a department.

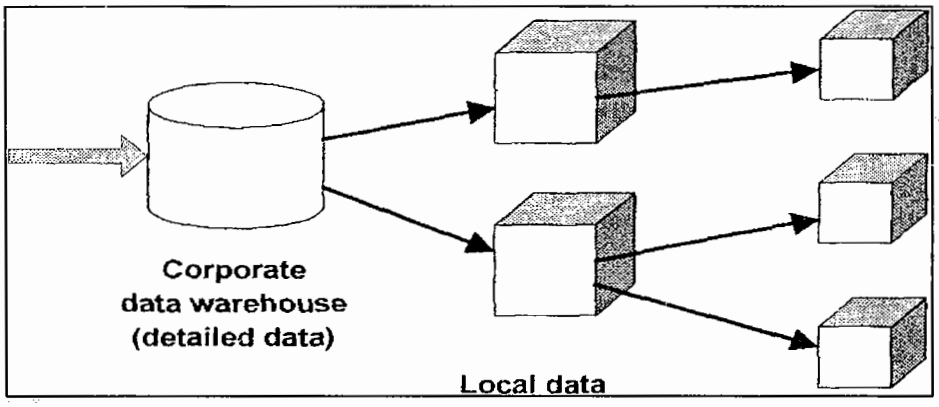
The amount of data is reduced in contrast to a central data warehouse. The level of detail is enhanced.



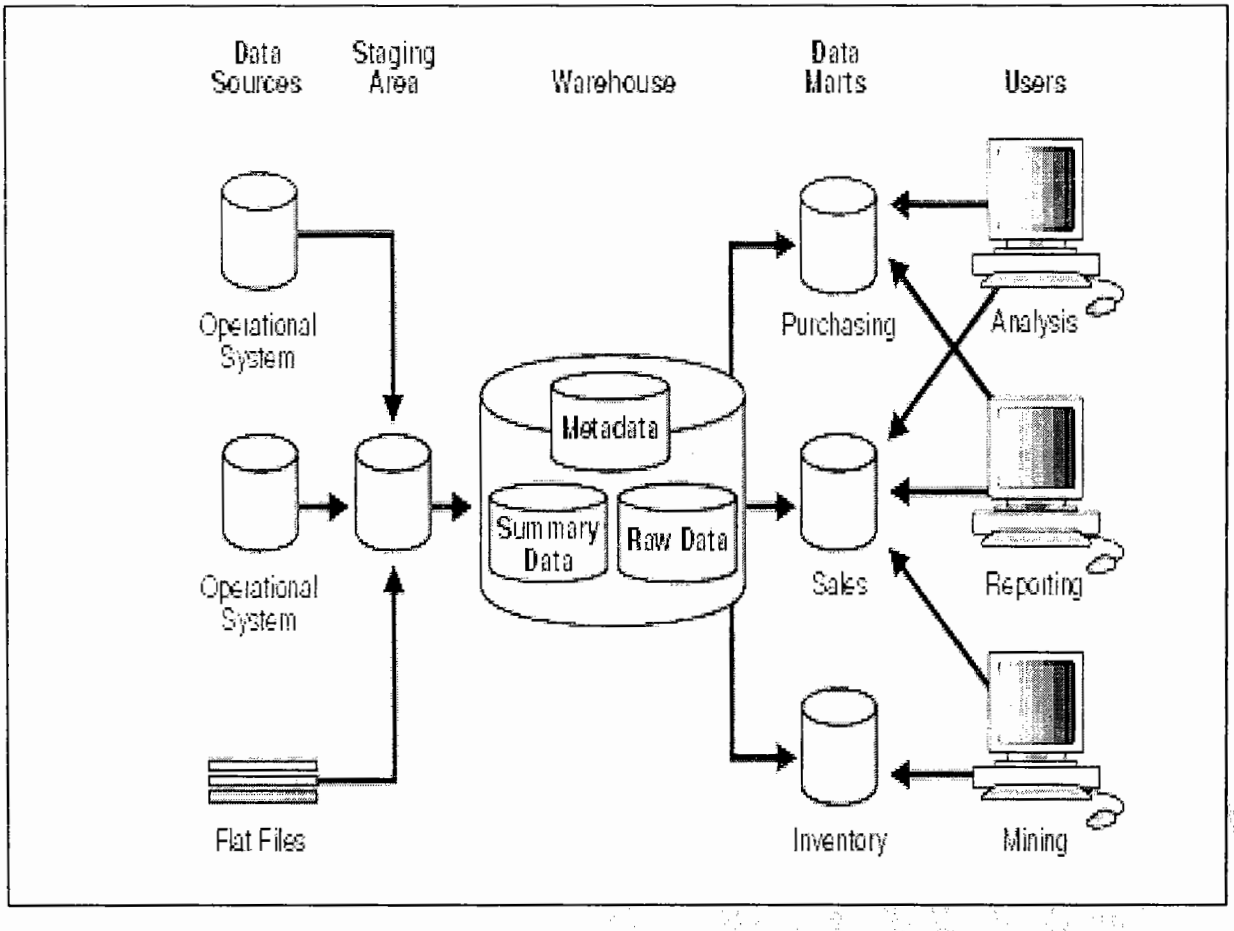
Tiered Architecture:

A tiered architecture is a distributed data approach. This process cannot be done in one step because many sources have to be integrated into the warehouse.

On a first level, the data of all branches in one region is collected, in the second level the data from the regions is integrated into one data warehouse.Eg: World→Countries→Cities→Regions



Generic Architecture for DWH:



As for above architecture data is integrated from different sources and that data stored in Staging DB. In staging Database all requirement related transformations performed. After that data will be loaded in to DWH form there data will be loaded in to different Data Marts for the purpose of analyzing data to make decisions by users.

- Staging Area - Staging area is a place where you hold temporary tables on data warehouse server. We basically need staging area to hold the data and perform data cleansing and merging before loading the data into warehouse.
- Data marts Vs Data Warehouse - Data mart is restricted to a single business process or single business group. Data Warehouse focuses on enterprise wide data across many or all subject areas .Data Marts are the subsets of a Data Warehouse.

Data Mart:

A data mart is a subject oriented data warehouse.

From the Data Warehouse, atomic data flows to various departments for their customized needs. If this data is periodically extracted from data warehouse and loaded into a local database, it becomes a data mart.

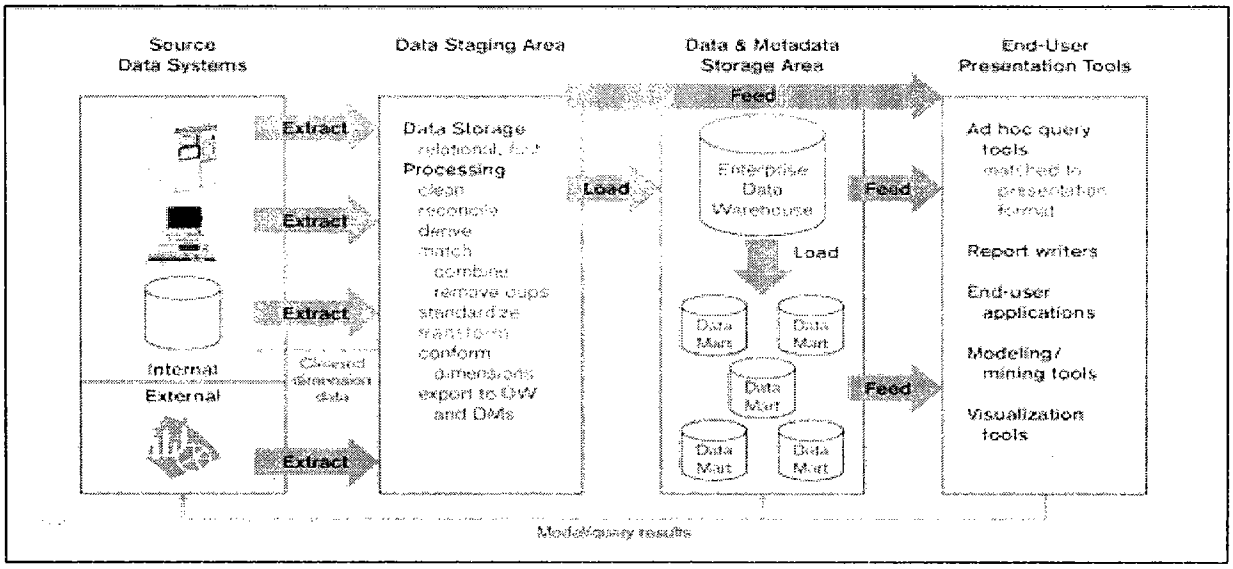
The data in Data Mart has a different level of granularity than that of Data Warehouse. Since the data in Data Marts is highly customized and lightly summarized, the departments can do whatever they want without worrying about resource utilization. Also the departments can use the analytical software they find convenient. The cost of processing becomes very low.

There are two types of Data Marts.

1. Dependent Data Mart
2. Independent Data Mart

Dependent Data Mart:

Here Data Marts are developed by using DWH.

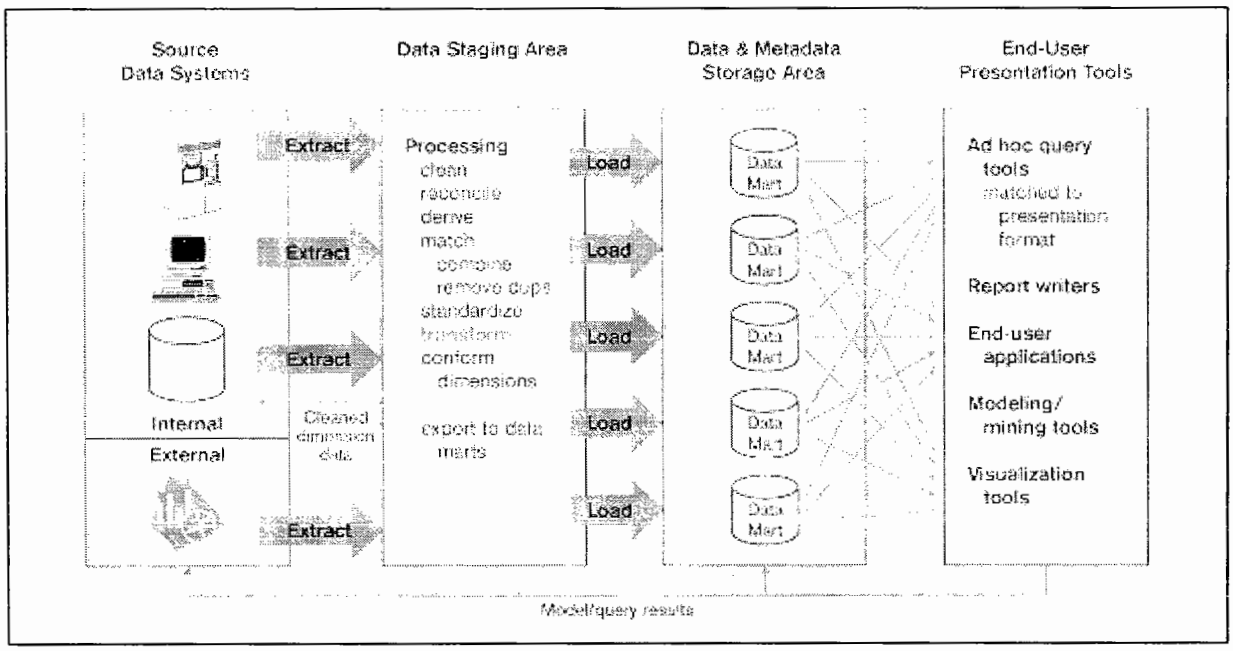


Top down Approach (<u>Inmon</u> Approach)

- The data flow begins with data extraction from the operational data sources. This data is loaded into the staging area and validated and consolidated for ensuring a level of accuracy and then transferred to the Data warehouse.
- A new set of transformations in done on the data in the data warehouse to help organize the data in particular structures required by data marts. Then the data marts can be loaded with the data and the OLAP environment becomes available to the users.

Independent DATAMART:

Here from Data Marts DWH will populate.



Bottom Up Approach (Kimball Approach)

- The bottom-up approach reverses the positions of the Data warehouse and the Data marts. Data marts are directly loaded with the data from the operational systems through the staging area.
- The data flow in the bottom up approach starts with extraction of data from operational databases into the staging area where it is processed and consolidated and then loaded into the Data mart.
- The data from the Data Mart, then is extracted to the staging area aggregated, summarized and so on and loaded into the Data Warehouse and made available to the end user for analysis.

Data Modeling:

Data Modeling is a technique aimed at optimizing the way that information is stored and used within an organization. It begins with the identification of the main data groups, for example the invoice, and continues by defining the detailed content of each of these groups.

Commonly Used Data Modeling Notations:

- **Entity** – Denotes the principal data object about which information is to be collected. E.g.: Student data, Employee data etc.
- **Attributes** – Attributes are characteristics of an Entity.
  E.g.: Student number, student name…e.t.c

- **Relationship** - A natural business association that exists between one or more entities. The relationship may represent an event that links the entities or merely a logical affinity that exists between the entities

Why should a Tester Know Data Modeling?

- Data Models provides the functional and technical aspects of the database design.
- Data Models help in ensuring that the design is complete for the defined business rules.
- Understanding the data models gives an understanding of the functionality to be tested.
- To carry out DB Testing like constraint testing, null value testing etc.
- In case multiple source systems, understanding the data model helps in validating the quality of data.

We need to understand the two important concepts that actually drive the design of these data models for OLTP and OLAP systems. They are,

1. ER Data Model
2. Dimensional Data Model

ER Data Model:

- Entity – Relationship Data Model is a data model that views the real world as entities and relationships. Entities are concepts, real or abstract about which information is collected. Entities are associated with each other by relationship and attributes are properties of entities. Business rules would determine the relationship between each of entities in a data model.
- The goal of OLTP data model is to normalize (avoid redundancy) data and to present it in a good normal form. While working with OLTP data modelling, a data modeller has to understand 1st normal form thru 5th normal form to design a good data model.
- The OLTP data model is popularly called as the OLTP Model or Relational Model

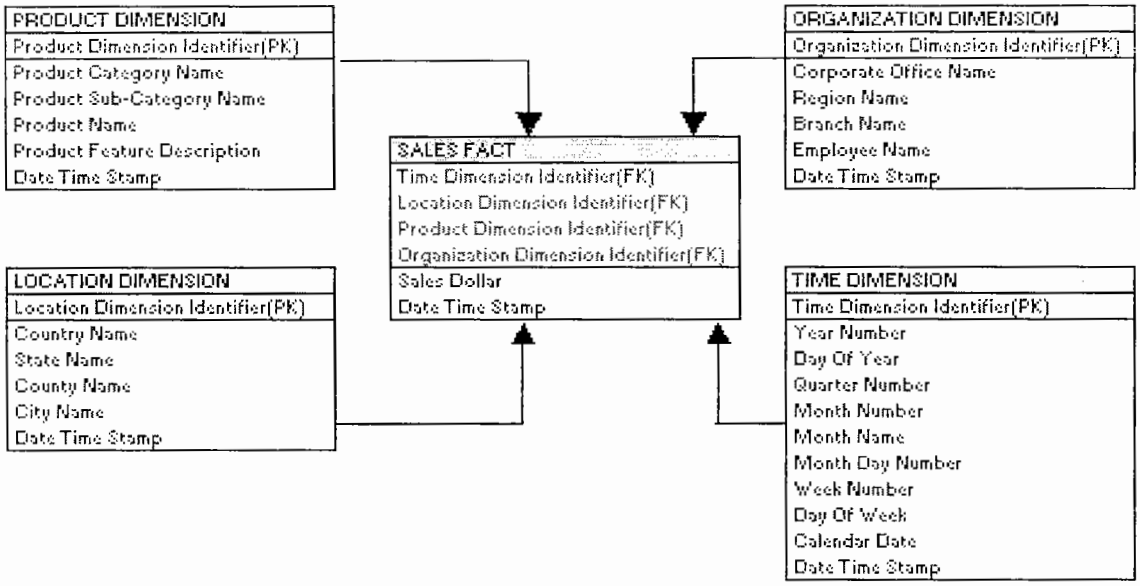
Dimensional Data Model

- Dimensional modelling is the design concept used by many data warehouse designers to build their data warehouse. It is a logical design technique that seeks to present the data in a standard, intuitive framework that allows for high-performance access.
- It adheres to a discipline that uses the relational model with some important restrictions.
    1. Every dimensional model is composed of one table with a multi-part key, called the fact table, and a set of smaller tables called dimension tables.
    2. Good examples of dimensions are location, product, time, promotion, organization etc. Dimension tables store records related to that particular dimension and no facts (measures) are stored in these tables.
    3. A fact (measure) table contains measures (sales gross value, total units sold) and dimension columns. These dimension columns are actually foreign keys from the respective dimension tables.
    4. Since here we look at faster query process, the data is de-normalized.

Dimensional Data Modeling – Example:

Sales fact table is connected to dimensions location, product, time and organization.

It shows that data can be sliced across all dimensions and again it is possible for the data to be aggregated across multiple dimensions. "Sales Dollar" in sales fact table can be calculated across all dimensions independently or in a combined manner.



Difference Between ER modeling & Dimensional Modeling:

ER Data Modeling	Dimensional Data Modeling
Data is stored in RDBMS	Data is stored in RDBMS or Multidimensional databases
Tables are units of storage	Cubes are units of storage
Data is normalized and used for OLTP. Optimized for OLTP processing	Data is demoralized and used in data warehouse and data mart. Optimized for OLAP
Several tables and chains of relationships among them	Few tables and fact tables are connected to dimensional tables
Volatile(several updates) and time variant	Non volatile and time invariant
SQL is used to manipulate data	ETL tools is used to manipulate data
Detailed level of transactional data	Summary of bulky transactional data(Aggregates and Measures) used in business decisions
Normal Reports	User friendly, interactive, drag and drop multidimensional OLAP Reports

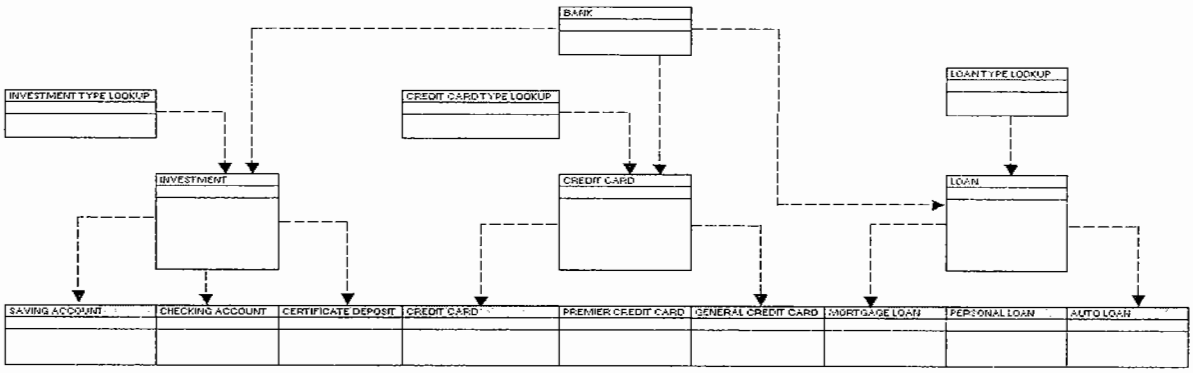
How to build Dimensional Modeling:

To build Dimensional modeling we need to follow five different phases.

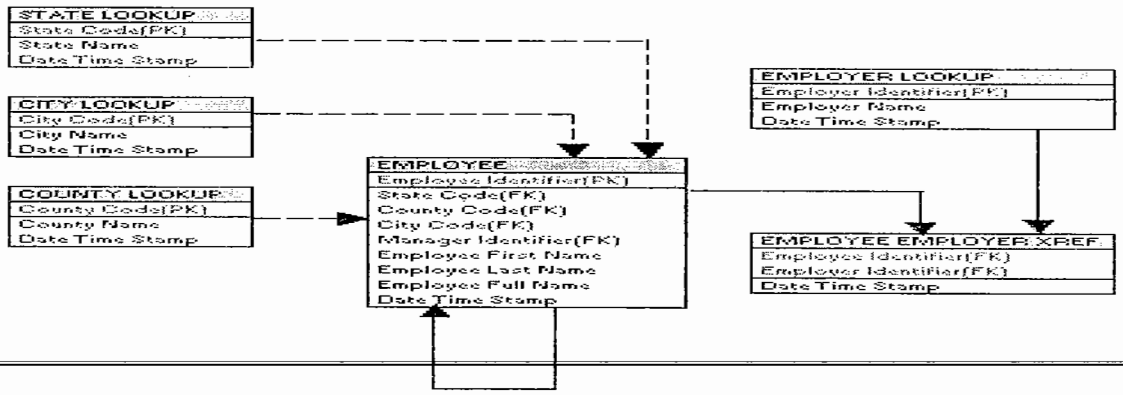
1. Gathering Business Requirements
2. Conceptual Data Modelling (CDM)
3. Logical Data Modelling (LDM)
4. Physical Data Modelling (PDM)
5. Generate Database

1. **Gathering Business Requirements - First Phase:** Data Modellers have to interact with business analysts to get the functional requirements and with end users to find out the reporting needs.

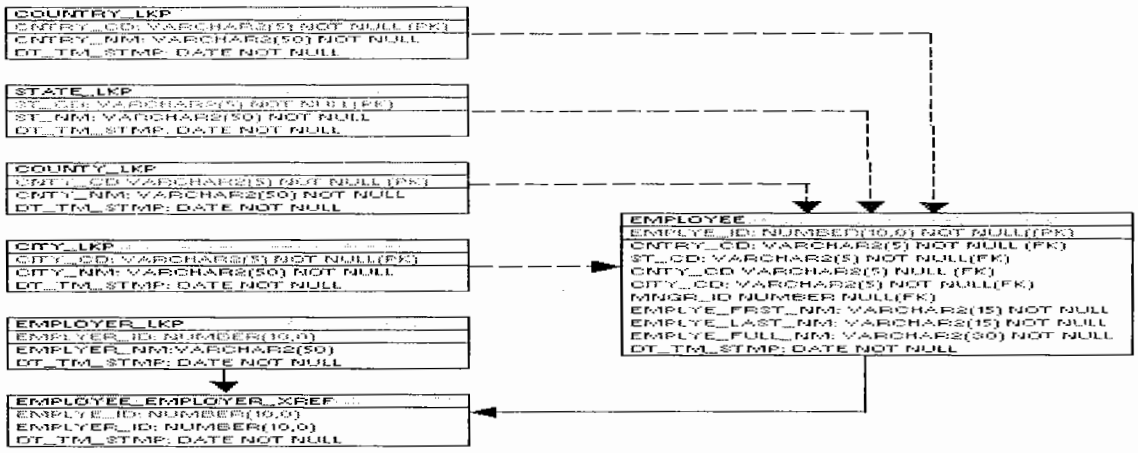
2. **Conceptual Data Modelling (CDM) - Second Phase:** This data model includes all major entities, relationships and it will not contain much detail about attributes and is often used in the INITIAL PLANNING PHASE.



3. **Logical Data Modelling (LDM) - Third Phase:** This is the actual implementation of a conceptual model in a logical data model. A logical data model is the version of the model that represents all of the business requirements of an organization.



4. **Physical Data Modelling (PDM) - Fourth Phase:** This is a complete model that includes all required tables, columns, relationship, database properties for the physical implementation of the database.



5. **Database - Fifth Phase:** DBA's instruct the data modelling tool to create SQL code from physical data model. Then the SQL code is executed in server to create databases.

Dimension table:

- Dimension table is one that describes the business entities of an enterprise, represented as hierarchical, categorical information such as time, departments, locations, and products. Dimension tables are sometimes called lookup or reference tables.
- Define business in terms already familiar to users
  - Wide rows with lots of descriptive text
- Small tables (about a million rows)
- Joined to fact table by a foreign key
- Heavily indexed
- Some Typical dimensions - time periods, geographic region (markets, cities), products, customers, salesperson, etc.

Fact Table:

- Represents a business process, i.e., models the business process as an artifact in the data model
- Contain the measurements or metrics or facts of business processes. Usually a numeric data.
- Some typical facts in a Fact Table are,
  - "monthly sales number" in the Sales business process
  - most are additive (sales this month), some are semi-additive (balance as of), some are not additive (unit price)

Fact tables contain foreign keys to the dimension tables

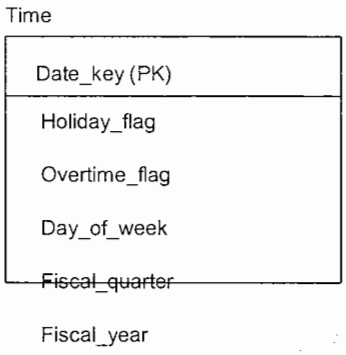
Measures: (Facts)

- A Fact table consists of measures.
- The measures are quantitative or factual data about the subject. The measures are generally numeric and correspond to the how much or how many aspects of a question.
- A measure can be based on a column in a table or it can be calculated.
- Examples - price, product sales, product inventory, revenue, sale amount, net profit margin and so forth.

Facts do not exist in a vacuum. They exist in the context of time, place, product, etc. For example, "units sold" means sales of a particular product, at a particular time, in a particular place. A fact stands at a cross-section of multiple dimensions. The combination of facts and the "who, what, where, and when" of these facts can be represented in a dimensional star schema.

Dimensions:

Dimension tables are a composite of one or more columns as primary key and other columns known as attributes. As an example, a time dimension is shown below.



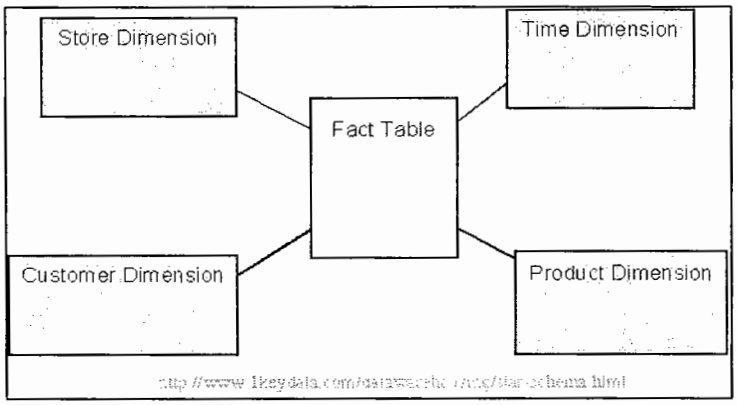
Dimensions typically contain hierarchies. Hierarchies show the parent-child relationships between elements of the dimension. Hierarchies are used to logically group and analyze information within one dimension. In other words, dimensions contain hierarchies, which are natural structures within business dimensions, as shown in Table 4 below:

Description	Hierarchy
Geography Dimension - table is at the store level but can be rolled up through to region	Store within Zip Code Zip Code within City City within Country Country within State State within Region
Product Dimension Hierarchy	Universal Product Code within Product Line Product Line within Brand
	Brand within Category Category within Department
Time Dimension Hierarchy	Day within Week

	Week within Month Month within Quarter Quarter within Year

Star Schema:

In the star schema design, a single object (the fact table) sits in the middle and is radially 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.



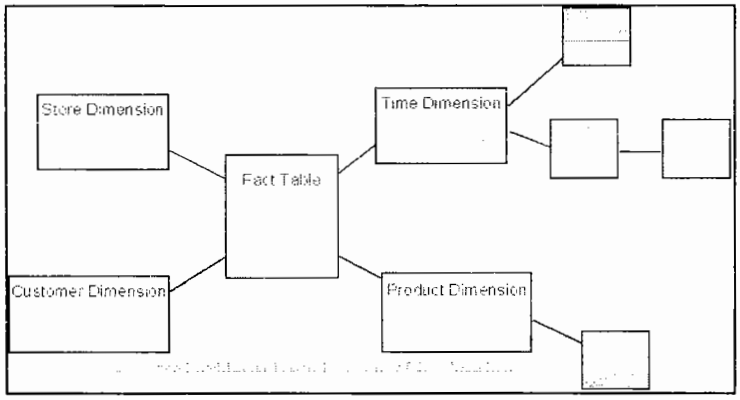
All measures in the fact table are related to all the dimensions that fact table is related to. In other words, they all have the same level of granularity.

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 repesents 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.

Snow flake 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.



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.

Types of Dimensions;

- Conformed Dimensions
- Degenerate Dimensions
- Junk Dimensions
- Slowly changing Dimensions

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.

The time dimension is a common conformed dimension in an organization. Usually the only rules to consider with the time dimension are 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.

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 those 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 x 2 x 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 offers better performance as well as being easier to manage.

Degenerate Dimensions:

A degenerate dimension is when the dimension attribute is stored as part of fact table, and not in a separate dimension table. These are essentially dimension keys for which there are no other attributes. In a data warehouse, these are often used as the result of a drill through query to analyze the source of an aggregated number in a report. You can use these values to trace back to transactions in the OLTP system.

Role Playing Dimensions:

A role-playing dimension is one where the same dimension key — along with its associated attributes — can be joined to more than one foreign key in the fact table. For example, a fact table may include foreign keys for both Ship Date and Delivery Date. But the same date dimension attributes apply to each foreign key, so you can join the same dimension table to both foreign keys. Here the date dimension is taking multiple roles to map ship date as well as delivery date, and hence the name of Role Playing dimension.

Slowly Changing Dimensions

Christina is a customer with ABC Inc. She first lived in Chicago, Illinois. So, the original entry in the customer lookup table has the following record

Customer Key	Name	State
1001	Christina	Illinois

At a later date, she moved to Los Angeles, California on January, 2003. How should ABC Inc. now modify its customer table to reflect this change? This is the "Slowly Changing Dimension"

Type 1: The new record replaces the original record. No trace of the old record exists.
Type 2: A new record is added into the customer dimension table. Therefore, the customer is treated essentially as two people.
Type 3: The original record is modified to reflect the change.

In Type 1 Slowly Changing Dimension, the new information simply overwrites the original information. In other words, no history is kept.
In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

After Christina moved from Illinois to California, the new information replaces the new record, and we have the following table:

Customer Key	Name	State
1001	Christina	California

- **Advantages**: - This is the easiest way to handle the Slowly Changing Dimension problem, since there is no need to keep track of the old information.
- **Disadvantages**: - All history is lost. By applying this methodology, it is not possible to trace back in history. For example, in this case, the company would not be able to know that Christina lived in Illinois before.
- **Usage**: About 50% of the time.
- **When to use Type** 1: Type 1 slowly changing dimension should be used when it is not necessary for the data warehouse to keep track of historical changes.

In Type 2 Slowly Changing Dimension, a new record is added to the table to represent the new information. Therefore, both the original and the new record will be present. The newe record gets its own primary key.
In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

After Christina moved from Illinois to California, we add the new information as a new row into the table:

Customer Key	Name	State
1001	Christina	Illinois
1005	Christina	California

- **Advantages**: - This allows us to accurately keep all historical information.
- **Disadvantages**: - This will cause the size of the table to grow fast. In cases where the number of rows for the table is very high to start with, storage and performance can become a concern. - This necessarily complicates the ETL process.
- **Usage**: About 50% of the time.

- **When to use Type 2**: Type 2 slowly changing dimension should be used when it is necessary for the data warehouse to track historical changes.

In Type 3 Slowly Changing Dimension, there will be two columns to indicate the particular attribute of interest, one indicating the original value, and one indicating the current value. There will also be a column that indicates when the current value becomes active.
In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

To accommodate Type 3 Slowly Changing Dimension, we will now have the following columns:

- Customer Key
- Name
- Original State
- Current State
- Effective Date

After Christina moved from Illinois to California, the original information gets updated, and we have the following table (assuming the effective date of change is January 15, 2003):

Customer Key	Name	Original State	Current State	Effective Date
1001	Christina	Illinois	California	15-JAN-2003

- **Advantages**: - This does not increase the size of the table, since new information is updated. This allows us to keep some part of history.
- **Disadvantages**: - Type 3 will not be able to keep all history where an attribute is changed more than once. For example, if Christina later moves to Texas on December 15, 2003, the California information will be lost.
- **Usage**: Type 3 is rarely used in actual practice.
- When to use Type 3: Type III slowly changing dimension should only be used when it is necessary for the data warehouse to track historical changes, and when such changes will only occur for a finite number of time.

1)SCD-1:

- HERE THE PREVIOUS DATA OVERWRITE BY CURRENT DATA
- MEANS HERE ONLY MAINTAIN CURRENT DATA.

2)SCD-2:

- HERE JUST ADD THE ADDITIONAL RECORDS
    - VERSIONING: MEANS HERE JUST SEND THE UPDATED RECORDS TO THE TARGET ALONG WITH VERSION NUMBER.NEW RECORDS WILL BE SENDING TO THE TARGET ALONG WITH PRIMARY KEY
    - FLAGVALUE:HERE UPDATED RECORDS SEND TO THE TARGET ALONG WITH 0 AND RECENT RECORDS SEND TO THE TARGET ALONG WITH 1
    - EFFECTIVE DATE RANGE:MEANS HERE TRACKS THE BOTH PREVIOUS AND CURRENT DATA

Method1 for Type 2: Versioning

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Version.
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

Method2: for Type To Add the Effective date Columns

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
124	ABC	Acme Supply Co	IL	22-Dec-2004	

The null End_Date in row two indicates the current tuple version. In some cases, a standardized surrogate high date (e.g. 9999-12-31) may be used as an end date, so that the field can be included in an index, and so that null-value substitution is not required when querying.

3)SCD-3:

- HERE MAINTAINS JUST PREVIOUS AND CURRENT DATA

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.

Example of Additive Fact

Date
Store
Product
Sales_Amount

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

Example of Semi-Additive or Non Additive Fact

Date
Account
Current_Balance
Profit_Margin

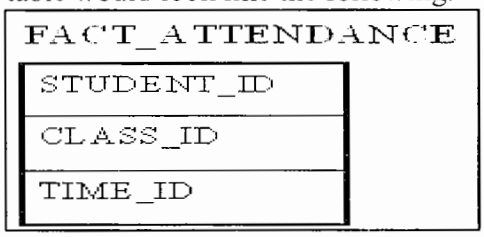
Current_Balance is a semi-additive fact, as it makes sense to add them up for all accounts (what's the total current balance for all accounts in the bank?), but it does not make sense to add them up through time

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.

Fact less Fact tables:

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 fact less fact table would look like the following:



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.

Fact less fact tables offers the most flexibility in data warehouse design. For example, one can easily answer the following questions with this fact less 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 fact less fact table, we will need two separate fact tables to answer the above two questions. With the above fact less fact table, it becomes the only fact table that's needed.

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

TERADATA BASICS

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

Teradata Overview

What is Teradata?

Teradata is a Relational Database Management System (RDBMS) that drives a company's data warehouse.

Teradata is an open system, compliant with industry ANSI standards

Teradata is a large database server that accommodates multiple client applications making inquiries against it concurrently. Various client platforms access the database through a TCP-IP connection or across an IBM mainframe channel connection.

The ability to manage terabytes of data is accomplished using the concept of parallelism, wherein many individual processors perform smaller tasks concurrently to accomplish an operation against a huge repository of data. To date, only parallel architectures can handle databases of this size.

Why Teradata?

There are many reasons why customers like you choose Teradata as the preferred platform for enterprise data warehousing:

- Supports **more warehouse data** than all competitors combined. There are over 300 1 TB or larger warehouses in the field.
- Supports **easy scalability** from a small (10 GB) to a massive (100+TB) database.
- Provides a **parallel-aware Optimizer** that makes query tuning unnecessary to get a query to run.
- **Automatic and even data distribution** eliminates complex indexing schemes or time-consuming reorganizations.
- Designed and built with **parallelism** from day one (not a parallel retrofit).
- **Supports ad-hoc queries** using industry-standard SQL, and includes SQL-ready database management information (log files).
- Single operational view of the entire MPP system (AWS) and single point of control for the DBA (Teradata Manager).

Teradata has been doing data warehousing longer than any other vendor.Teradata was built for decision support from the beginning.

Scalability in a Production Environment

Teradata customers run the largest relational database systems in the world, but they often don't start out that way. Many times, a company starts out building a data warehouse in one area of the company. When they see the benefits realized from that one area, they envision the value of adding more areas to the data warehouse. As the value is realized, they add even more subject areas, and the system grows.

The Teradata system may be as small as 10 gigabytes. With its Parallelism and scalability, Teradata allows you to start small with a single node and grow large with many nodes through linear scalability. The largest Teradata system is now more than 330 nodes. Teradata accommodates such growth in many ways, including:

- Ability to handle many concurrent users.

- Ability to add more nodes to increase processing power – with no requirement to change applications or utilities, or replace the data model.
- Ability to add more disk capacity or processing power – with no requirement to offload/reload data or manually partition data.

Teradata Advantages

Teradata provides customers with unlimited, proven scalability. Teradata can scale from 10 gigabytes to over 100 terabytes of data on a single system. When we talk about scalability, it isn't just about being able to grow very large and handle enormous amounts of data. It's about growth without losing any performance capabilities. This scalability provides investment protection for customer's growth and application development.

As is proven in every benchmark we perform, Teradata can handle the most concurrent users, who are often running complex queries. Our competitors have difficulty when trying to run multiple queries.

The parallelism of Teradata is unlimited. The Teradata Database performs every task in parallel, including joins, sorts, and aggregations.

Our optimizer is the most robust in the industry, able to handle multiple,complex queries, joins per query, and unlimited ad-hoc processing.

Teradata provides the lowest total cost (TCO) of ownership due to:

- Ease of setup and maintenance
- No reorganization of data needed
- Most robust utilities in the industry
- Low cost of disk to data ratio
- Ease in expanding the system

High availability is also a major advantage because with the Teradata architecture, there is no single point of failure - fault tolerance is built-in to the system. Teradata is the only database that is truly scalable, and this extends to data loading with the use of our parallel utilities.

Teradata Advantages

- **Unlimited, Proven Scalability**
- **Most Concurrent Users -** Multiple complex queries
- **Unlimited Parallelism -** Parallel sorts/aggregations, temporary tables - "shared-nothing" architecture
- **Mature Optimizer** - Complex queries, joins per query, ad-hoc processing
- **Model the Business -** 3NF, robust view processing, star schema
- **Lowest TCO -** Ease of setup and maintenance, robust parallel utilities, no re-orgs, lowest disk to data ratio, robust expansion utility
- **High Availability** - No single point of failure, scalable data loading, parallel load utilities

Teradata —A Brief History

Teradata Corporation was founded in 1979 in Los Angeles, California. The corporate goal was the creation of a database computer that could handle billions of rows of data, up to and beyond a terabyte of data storage. In 1982, the YNET technology was patented for the

parallelism that was at the heart of the architecture. The YNET was the interconnect that allowed hundreds of individual processors to share the same bandwidth on the first Teradata systems.

After five years in development, a product was shipped to the first customer in 1984, based on the Teradata Operating System (TOS). In 1987, Teradata went public with its first stock offering. In 1989, Teradata partnered with NCR Corporation to build the next generation of database computers. Project 90 developed the advancements that made the 3600 system possible.

By 1990, the first terabyte-sized Teradata Database was in production.

NCR was purchased by AT&T Corporation in 1991 and Teradata was purchased and folded into the NCR structure in 1992.

In 1995, the Teradata Database was ported to the UNIX MP-RAS operating system.

In 1996, AT&T spun NCR off into a separate company and by 1997 NCR had become the world leader in scalable data warehouse solutions, due to the strength of the Teradata Database. Teradata is a consistent leader in all categories of multiple user TPC-D benchmarks, including 100 GB, 300GB, and 1TB.

Teradata continued on its path as an open system. In 1998 it was ported to Windows NT, then in the fall of 2000 it became available on Windows 2000.

By the end of 2000, NCR created the Teradata Division, which is committed to Teradata and to Data Warehousing through today and beyond.

By the end of 2002, Teradata Division released Teradata V2R5 which is a major release including features such as PPI, roles and profiles, multivalue compression, and more.

By the end of 2007: Teradata launches Teradata 12

By the end of 2009: Teradata launches Teradata 12

By the end of 2013 Teradata is named a leader in Gartner's Data Warehouse DBMS Magic Quadrant in February 2013

How large is a Trillion?

A trillion is ten to the twelfth power – that means 12 zeroes! Teradata was the first commercial database system to support a trillion bytes of data. Today, we have systems with more than 100 trillion bytes of data.

It is hard to imagine the size of a trillion. We think that because we have a word for it, we know how big it really is. Most people are comfortable with kilobytes, megabytes, and even gigabytes, but terabytes are another three orders of magnitude. A terabyte is 1,000 gigabytes. Your laptop might have a 9- or 10-GB hard drive -- a terabyte is equal to 56 18-GB drives.

How Large is a Trillion?

1 Kilobyte = 103 = 1000 bytes
1 Megabyte = 106 = 1,000,000 bytes
1 Gigabyte = 109 = 1,000,000,000 bytes
1 Terabyte = 1012 = 1,000,000,000,000 bytes
1 Petabyte = 1015 = 1,000,000,000,000,000 bytes
1 Exabyte = 1018 = 1,000,000,000,000,000,000 bytes
1 Zetabyte = 1021 = 1,000,000,000,000,000,000,000 bytes
1 Yottabyte = 1024 = 1,000,000,000,000,000,000,000,000 bytes
1 million seconds = 11.57 days

1 billion seconds = 31.6 years
1 trillion seconds = 31,688 years
1 million inches = 15.7 miles
1 trillion inches = 15,700,000 miles (30 round trips to the moon)
$1 million = < $ .01 for every person in U.S.
$1 billion = $ 3.64 for every person is U.S.
$1 trillion = $ 3,636 for every person in U.S.

Designed for Today's Business

Teradata was built from the start for decision support. The business needs that Teradata's founders worked to fulfill are even more pressing today: market fluctuations, global competition, high customer expectations, and the need to make faster decisions at all levels. Do these challenges sound familiar to you?

The key to these challenges is knowledge. Teradata helps you get that knowledge to meet the stringent requirements of today's business with:

- **Large capacity database machine:** Teradata handles the large data storage requirements to process the large amounts of detail data for decision support.
    - Billions of rows
    - Terabytes of data
- **Performance:** Early relational systems suffered seve re performance limitations as table size increased. Teradata addresses the performance issues of large databases.
- **Single data store for multiple clients:** Instead of replicating a database for different hosts, with Teradata you store it once and use it for all clients. This is what's known as "a single version of the truth".
- **Connectivity:** Teradata connects easily to network-attached host systems as well as mainframe hosts.
- **Standard access language (SQL):** SQL has been adapted as the industry standard for relational databases.
- **Manageable growth:** Teradata systems are linearly expandable to allow for growth without performance drop-off.
- **Fault tolerance:** Teradata can automatically detect and recover from one or more hardware failures.
- **Data integrity:** To guarantee the integrity of the data, transactions are either completed or, if a fault occurs, rolled back.

Throughout this course, you will see more about how Teradata meets the business challenges of today and tomorrow.

They know that if data doubles, the system can expand easily to accommodate it.

The workload for creating a table of 100 rows is the same as creating 1,000,000,000 rows!

Relational Database Advantages

Flexible

Flexibility provides substantial benefits. The user does not need to know the access path of the data; the RDBMS keeps track of where everything is. Relational databases use atomic data

—breaking data down into its smallest parts to allow maximum flexibility in selecting and using data.

Responds quickly

In a traditional database, adding a field means that all programs that use the database must be rewritten to become aware of the new data structure. In a relational database, programs do not need to be re-written when a field is added.

Data-driven

Relational databases are designed to represent a b usiness and its practices - not the application or the computer system.

Business-oriented

The two tables we have looked at, EMPLOYEE and DEPARTMENT, are organized to reflect the way the business really works.

Simple and easy to use and understand

Simplicity is useful not only to the people who ask the questions, but also to the people who have to figure out how to retrieve information from the database. Understanding how a RDBMS functions is not necessary.

Easier to build applications

Relational databases make the data do more work. Programs and transactions are simpler, which makes it easier to build applications.

Support the trend toward end-user computing

The trend is moving away from organizations funneling all data requests through a few people who know how the system works. As systems get easier to use, more people have access to them. This is called the "democratization of data."

Set Theory

Set theory is the mathematical science of the infinite. It studies properties of sets, abstract objects that pervade the whole of modern mathematics. The language of set theory, in its simplicity, is sufficiently universal to formalize all mathematical concepts and thus set theory, along with Predicate Calculus, con stitutes the true Foundations of Mathematics.

Components and Architecture

What is a Node?

Up to this point, we have discussed relational databases in terms of how the user sees them - as a collection of tables that relate to one another.Now we'll look at the physical components of the system.

A node is made up of various hardware a nd software components. All applications run under UNIX , or Windows, and all Teradata software runs under PDE. All share the resources of CPU and memory on the node.

AMPs and PEs are **virtual processors (vprocs)** running under control of the PDE. Their numbers are software configurable.

AMPs are associated with **virtual disks (vdisks),** which are configured as ranks of a disk array.

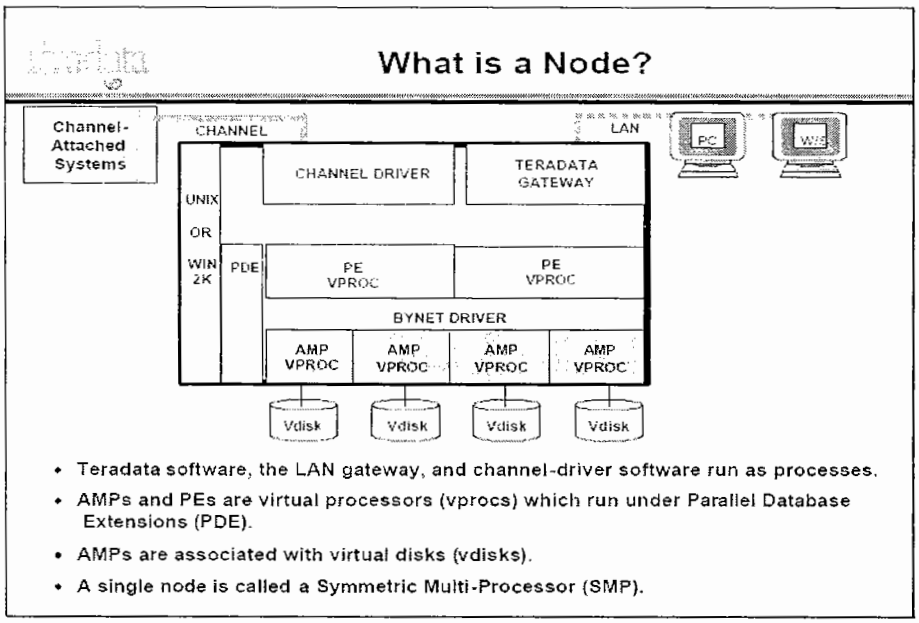
The "Shared Nothing" architecture of Teradata means that each vproc is responsible for its own portion of the database, and for all activities to be performed on that part of the system.

All AMPs and PEs communicate via the **BYNET**, which is a message passing system. In an SMP (Symmetric Multi-Processing) system, it is implemented as boardless BYNET, and in MPP (Massively Parallel Processing) systems, it is implemented as a software and hardware solution. The BYNET allo ws multiple vprocs on multiple nodes to communicate with each other. PDE actually controls message-passing activity, while the BYNET handles message queuing and flow control.

An application that runs under the control of PDE, such as Teradata, is considered a

Trusted Parallel Application (TPA).

Note: The versatility of the Teradata Database is based on virtual processors (vprocs) that eliminate dependency on specialized physical processors. These vprocs are a set of software processes that run on a node under the Teradata Parallel Database Extensions (PDE) and the multitasking environment of the operating system.



MPP System

When multiple SMP nodes are connected to form a larger configuration, we refer to this as a **Massively Parallel Processing (MPP)** system.

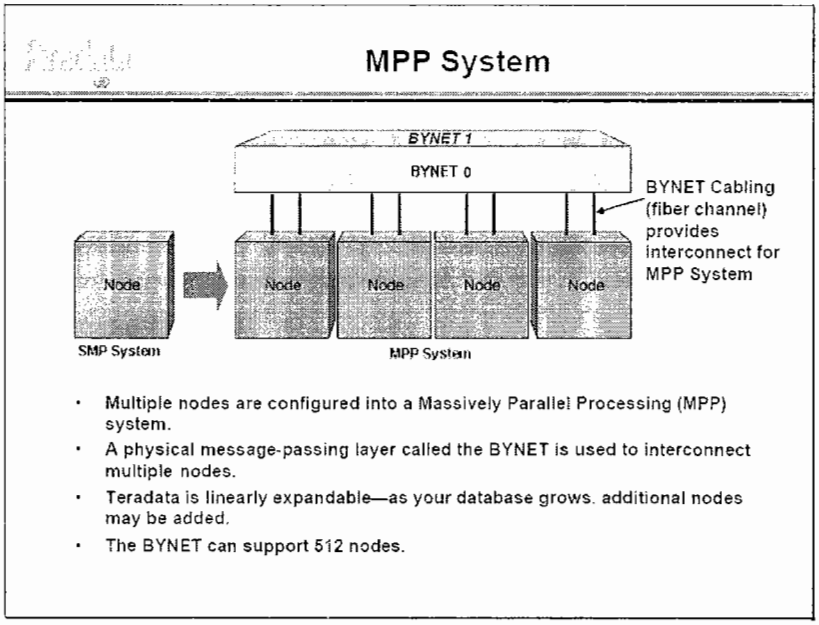
The **BYNET** connects multiple nodes together to create an MPP system.

Because Teradata is a **linearly expandable** database system, as additional nodes and vprocs are added to the system, the system capacity scales in a linear fashion.

The BYNET can currently support 512 nodes with the capacity to support 2048 nodes in the future.

An MPP system has two BYNET networks (BYNET 0 and BYNET 1). Because all networks in a system are active, the system benefits from having full use of the aggregate bandwidth of all the networks. Traffic is automatically and dynamically distributed between the BYNETs . Since the number of the networks can be scaled, the performance can also be scaled to meet the needs of especially demanding applications.

Each BYNET network has multiple connection paths. If the BYNET detects an unusable path in either network, it will automatically reconfigure that network so all messages avoid the unusable path. Additionally, in the rare case that BYNET 0 cannot be reconfigured, hardware on BYNET 0 is disabled and messages are re-routed to BYNET 1, and vice versa.



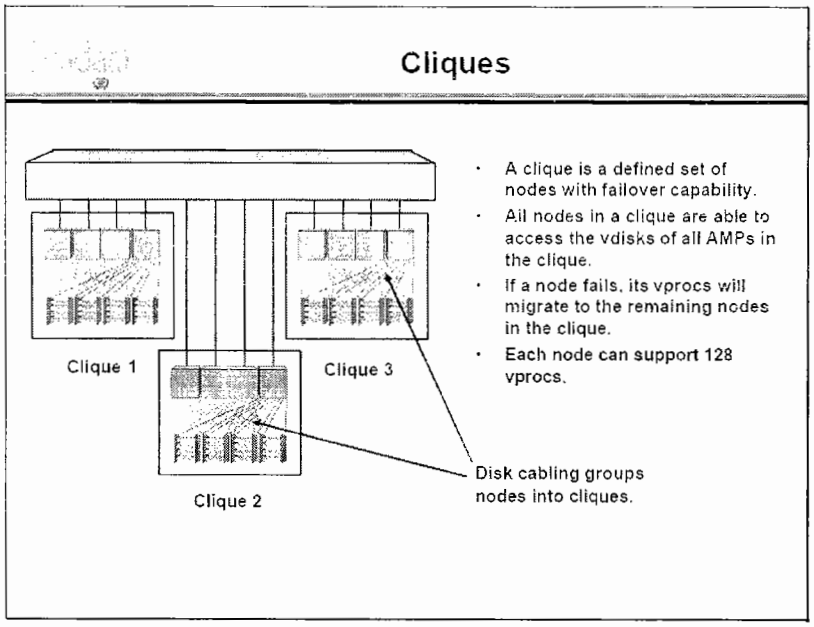
Cliques

A **clique** is a set of Teradata nodes that share a common set of disk arrays. Cabling a subset of nodes to the same disk arrays creates a clique.

In the event of node failure, virtual processors on the failed node can migrate to another available node in the clique to keep the system operational. Access to all data is maintained. In order for the vprocs to migrate, all nodes in a clique must have access to the same disk arrays.

The diagram on the facing page shows a twelve-node system consisting of three cliques, each containing four nodes. Because all disk arrays are available to all nodes in the clique, the AMP virtual processors will retain access to their responsible rows.

In the event of three out of four nodes failing, the remaining node would attempt to absorb all virtual processors from the failed nodes. Because each node can support a maximum of 128 virtual processors, the total number of virtual processors for the clique should not exceed 128.



Major Components of a Teradata System

We have looked at the overall node, and now we will describe the components that make up a node in detail.

Parsing Engine (PE)

The Parsing Engine (PE) is a component that interprets SQL requests, receives input records, and passes data. To do that it sends the messages through the BYNET to the AMPs.

BYNET

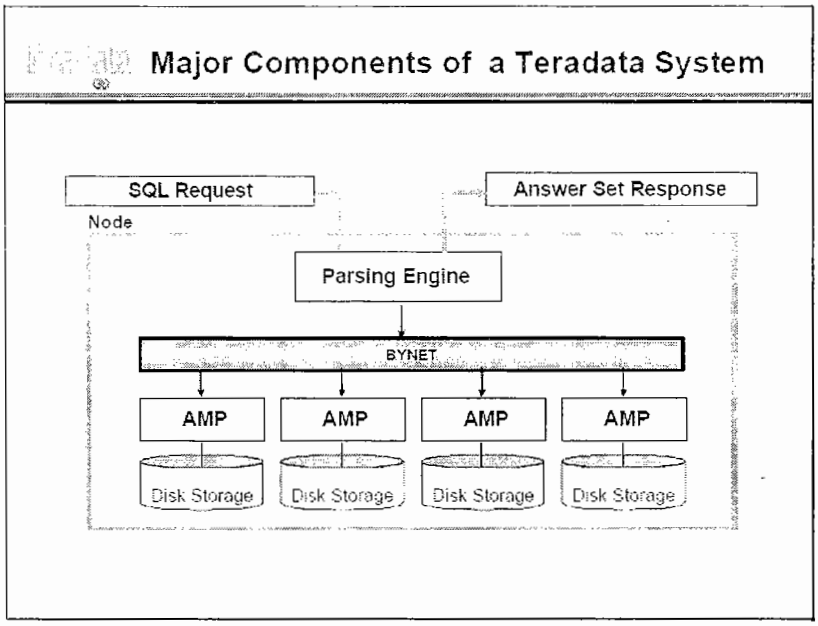
The BYNET is the message-passing layer. It determines which AMP(s) (Access Module Processor) should receive a message.

Access Module Processor (AMP)

The AMP is a virtual processor (vproc) designed for and dedicated to managing a portion of the entire database. It performs all database management functions such as sorting, aggregating, and formatting data. The AMP receives data from the PE, formats rows, and distributes them to the disk storage units it controls. The AMP also retrieves the rows requested by the Parsing Engine.

Disks

Disks are disk drives associated with an AMP that store the data rows. On current systems, they are implemented using a **disk array**.



The Parsing Engine

A **Parsing Engine (PE)** is a virtual processor (vproc). It is made up of the following software components: Session Control, the Parser, the Optimizer, and the Dispatcher.

Once a valid session has been established, the PE is the component that manages the dialogue between the client application and the RDBMS. Each PE can handle up to 120 sessions, and each session can handle multiple requests.

Session Control

The major functions performed by Session Control are logon and logoff.Logon takes a textual request for session authorization, verifies it, and returns a yes or no answer. Logoff terminates any ongoing activity and deletes the session's context.

Parser

The Parser interprets SQL statements, checks them for proper SQL syntax and evaluates them semantically. The PE also consults the Data Dictionary to ensure that all objects and columns exist and that the user has authority to access these objects.

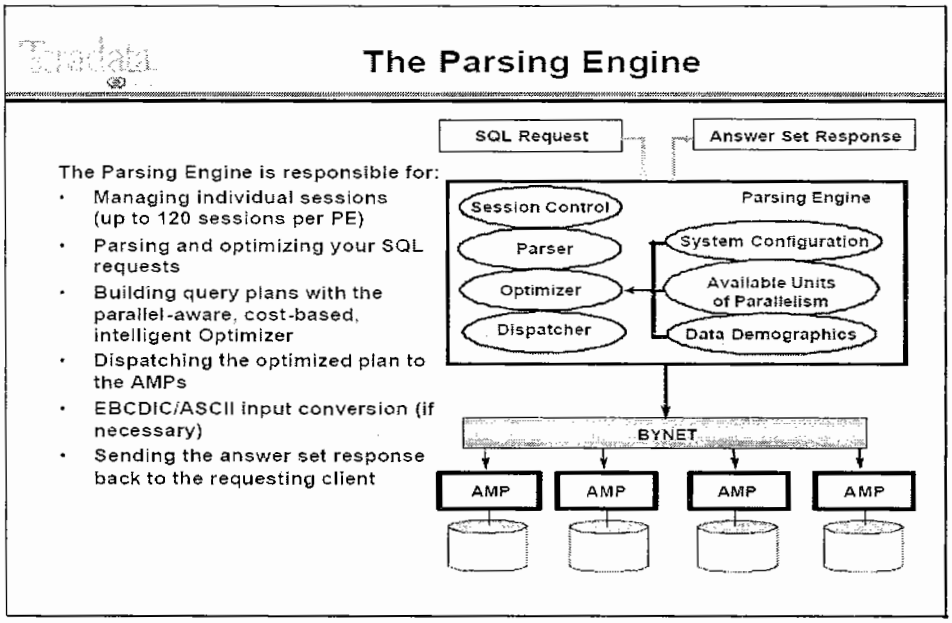
Optimizer

The **Optimizer** is responsible for developing the least expensive plan to return the requested response set. Processing alternatives are evaluated and the fastest plan is chosen. This plan is converted to executable steps, to be performed by the AMPs, which are then passed to the dispatcher. In order to maximize throughput and minimize resource contention, the optimizer must know about **system configuration**, available **units of parallelism** (AMPs and PE's), and **data demographics**. The Teradata Optimizer is robust and intelligent. The optimizer enables Teradata to handle multiple complex, ad hoc queries efficiently. It is **parallel-aware** and **cost-based** and uses full look-ahead capability.

Dispatcher

The Dispatcher controls the sequence in which the steps are executed and passes the steps on to the BYNET. It is composed of executioncontrol and response-control tasks. Execution control receives the step definitions from the Parser and transmits them to the appropriate AMP(s) for processing, receives status reports from the AMPs as they process the steps, and passes the results on to response control once the AMPs have completed processing. Response control returns the results to the user. The Dispatcher sees that all AMPs have finished a step before the next step is dispatched. Depending on the nature of the SQL request, a step will be sent to one AMP, or broadcast to all AMPs.

Components and Architecture Page 4-13



BYNET

The **BYNET** handles the internal communication of the Teradata Database. All communication between PEs and AMPs is done via the BYNET.

When the PE dispatches the steps for the AMPs to perform, they are dispatched onto the BYNET. The messages are routed to the appropriate AMP(s) where results sets and status information are generated. This response information is also routed back to the requesting PE via the BYNET.

Depending on the nature of the dispatch request, the communication between nodes may be a:

- **Broadcast**—message is routed to all nodes in the system.
- **Point-to-point**—message is routed to one specific node in the system.

Once the message is on a participating node, PDE directs the message to the appropriate AMPs on that node. All AMPs receive a broadcast message. With a point-to-point or multicast (multiple AMPs) message, the message is directed only to the appropriate AMP(s) on that node.

So, while a Teradata system does do multicast messaging, the BYNET hardware alone cannot do it - the BYNET can only do point-to-point and broadcast between nodes.

The BYNET has several unique features:

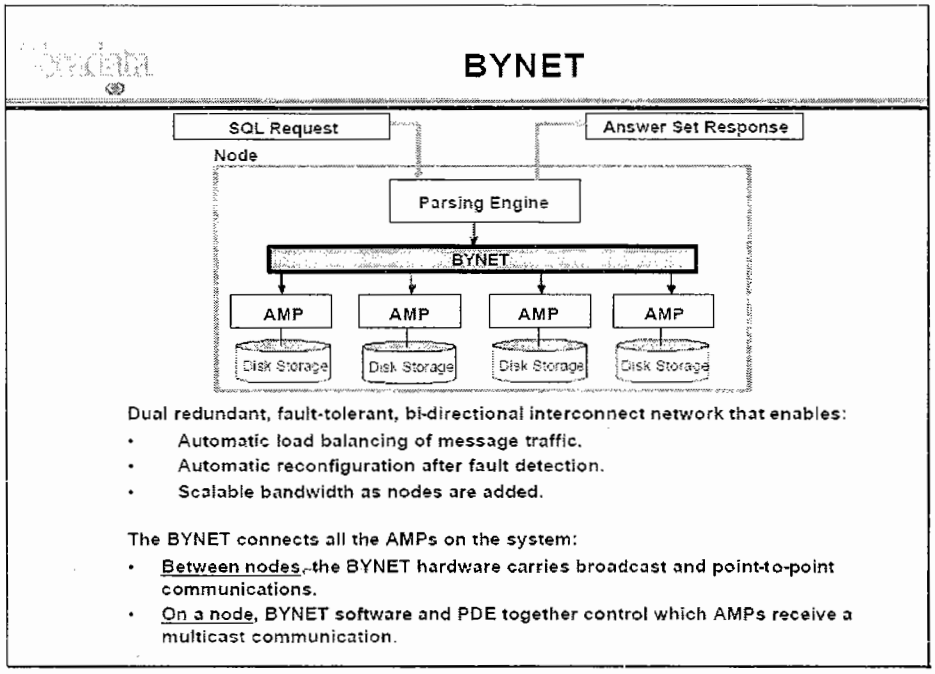
Fault tolerant: each network has multiple connection paths. If the BYNET detects an unusable path in either network, it will automatically reconfigure that network so all messages avoid the unusable path. Additionally, in the rare case that BYNET 0 cannot be reconfigured, hardware on BYNET 0 is disabled and messages are re-routed to BYNET 1, and vice versa.

Load balanced: traffic is automatically and dynamically distributed between both BYNETs.

Scalable: as you add nodes to the system, overall network bandwidth scales linearly - meaning an increase in system size without loss of performance.

High Performance: an MPP system has two BYNET networks. Because both networks are active, the system benefits from the full aggregate bandwidth.

The technology of the BYNET is what makes the T eradata parallelism possible.
Components and Architecture Page 4-15



The Access Module Processor (AMP)

The **Access Module Processor (AMP)** is the virtual processor (vproc) in Teradata's shared-nothing architecture that is responsible for managing a portion of the database. An AMP will control some portion of each table on the system. AMPs do the physical work associated with generating an answer set, including sorting, aggregating, formatting and converting. The AMPs perform all database management functions in the system.

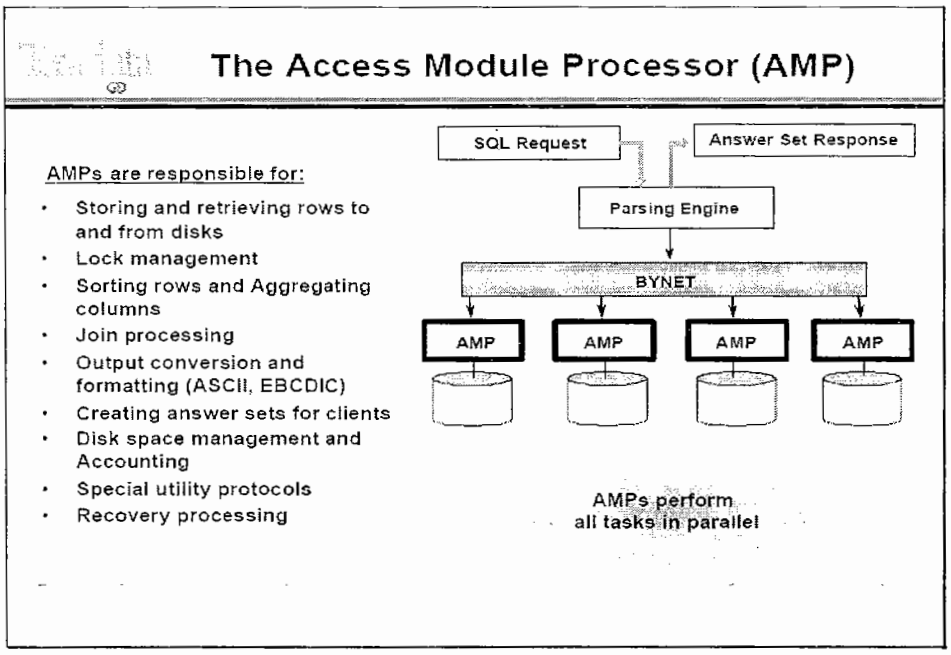
An AMP responds to Parser/Optimizer steps transmitted across the BYNET by selecting data from or storing data to its disks. For some requests, the AMPs may redistribute a copy of the data to other AMPs.

The **Database Manager** subsystem resides on each AMP. This subsystem will:

- **Lock** databases and tables.
- **Create, modify**, or **delete definitions** of tables.
- **Insert, delete,** or **modify rows** within the tables.
- **Retrieve information** from definitions and tables.
- Return responses to the Dispatcher.

Earlier in this course, we discussed the logical organization of data into tables. The database manager subsystem provides a bridge between that logical organization and the physical organization of the data on disks. The Database Manager performs a space-management function that controls the use and allocation of space.

Teradata performs all tasks in parallel, providing exceptional performance. The greater the number of tasks processed in parallel, the better the system performance. Many databases call themselves "parallel", but they can only perform some tasks in parallel.



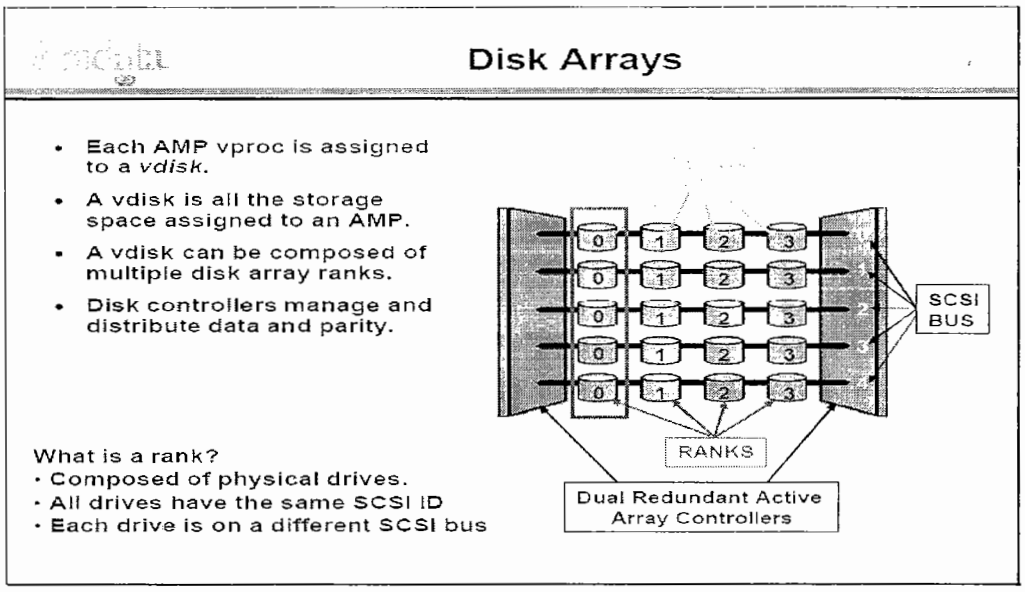
Disk Arrays

A **disk array** is a configuration of disk drives that utilizes specialized controllers to manage and distribute data and parity across the disks while providing fast access and data integrity.

The disk array controllers are referred to as **dual redundant active array controllers**, which means that both controllers are actively used, in addition to serving as backup for each other.

Each **AMP vproc** must have access to an array controller, which in turn accesses the physical disks. AMP vprocs are associated with one or more ranks (or mirrored pairs) of data.

The total disk space associated with an AMP is called a **vdisk**. A vdisk may have up to three ranks.



Teradata Client Software

The Teradata Database requires three distinct pieces of software: TPA, PDE, and OS.

An operating system (OS), UNIX or Windows 2000, and a Teradata software license are necessary for each node.

A Trusted Parallel Application (TPA) implements virtual processors and runs on the OS with PDE. The Teradata Databse is classified as a TPA.

The components of the Teradata Database software include:

- Channel Driver
- Teradata Gateway
- AMP
- PE

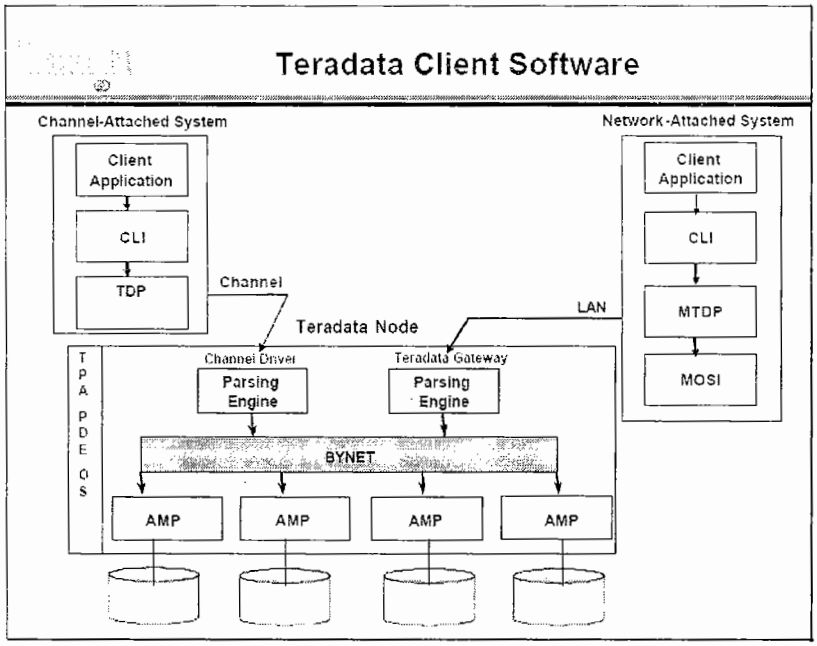
Parallel Database Extensions (PDE) were added to the operating system to support the parallel software environment.

On UNIX, a virtual processor (vproc) is a collection of software processes running under the multi-tasking environment of the UNIX operating system.

On Windows 2000, a vproc is a software process that runs under the multi-tasking environment of the Windows 2000 operating system.

The client may be a mainframe system, such as IBM or Amdahl, which is channel-attached to the Teradata Database, or it may be a PC or UNIX -based system that is LAN-attached.

The client application submits an SQL request to the RDBMS, receives the response, and submits the response to the user.



Linear Growth and Expandability

The Teradata Database is the first commercial database system to offer true parallelism and the performance increase that goes with it. Think back to the example we just discussed of how rows are divided up among AMPs. Assume that our three tables, EMPLOYEE, DEPARTMENT, and JOB total 100,000 rows and 50 users.

What happens if you **double the number of AMPs** and the number of users stays the same? Performance **doubles**. Each AMP processes half as many rows as it used to.

Now think of that system in a situation where both the number of **users** and the number of AMPs are **doubled**. We now have 100 users, but we also have twice as many AMPs. What happens to performance? It **stays the same**. There is no drop-off in the speed with which requests are executed because the system is **modular** and the workload is easily partitioned into independent pieces. In the last example, each AMP still does the same amount of work as when there were half the AMPs and half the users.

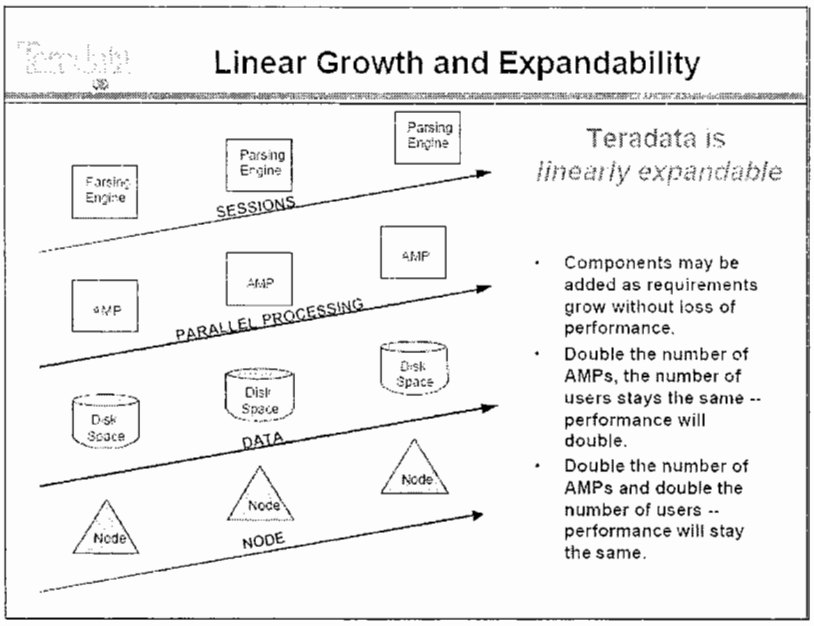
This feature—that the amount of time (or money) required to do a task is directly proportional to the size of the system—is unique to the Teradata Database. Traditional databases show a sharp drop in performance when the system approaches a critical size.

Look at the diagram on the facing page. As the number of Parsing Engines increases, the number of SQL requests that can be supported increases.

As you add **AMPs**, data is spread out more evenly as you add processing power (nodes) to handle it.

As you add **disks**, you add space for each AMP to store and process more information. All AMPs must have the same number of disks.

Note: Performance assumptions on this page are approximations, and assume appropriate hardware increases as well as added vproc software.



Accessing Teradata Objects

Teradata Objects

A **database** in the Teradata system is a collection of objects known as **tables, views, macros, triggers** and **stored procedures.** Databases provide a **logical grouping for information**. They are also the foundation for space allocation and access control.

Tables

A table is the logical structure of data in an RDBMS. It is a **twodimensional structure made up of columns and rows**. A user defines a table by giving it a table name that refers to the type of data that will be stored in the table (e.g., an Employee table stores data about employees.)

A **column** represents attributes of the table. Column names are given to each column of the table. All information in a column is of the same type. For example, a column named Date of Birth would only hold date of birth information. Each occurrence of an entity is stored in the table as a **row.** Entities are the people, places, things, or events that the table is describing. Tables require Permanent Space to store rows.

Views

A view is a **pre-defined subset of one or more tables or other views**. It does not exist as a real table, but serves as a reference to existing tables or views. One way to think of a view is as a virtual table. Views have definitions in the Data Dictionary (DD) but do not contain any physical

rows. Views can be used to control access to the underlying tables. Views can be used to hide columns from users, to insulate applications from database changes, and to standardize or simplify access techniques. Views do not require Permanent Space.

Macros

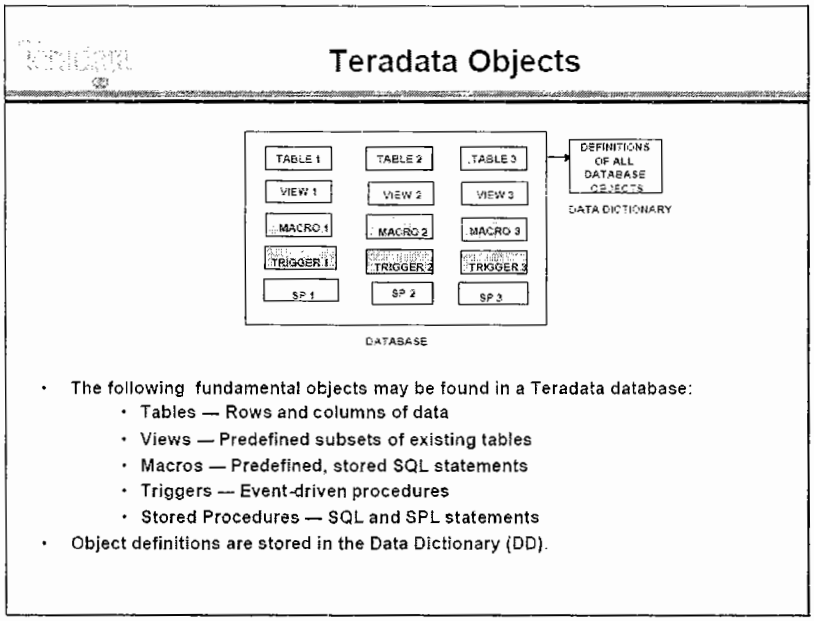
A macro is a predefined, stored set of one or more SQL commands and report-formatting commands. Macros are used to simplify the execution of frequently used SQL commands. Macros do not require Permanent Space.

Triggers

A trigger is an event-driven procedure attached to a table. A trigger defines events that happen when some other event, called a **triggering event**, occurs. A trigger consists of one or more SQL statements that are associated with a table and are executed when the trigger is fired. A trigger is created with the CREATE TRIGGER statement.

Stored Procedures

A stored procedure is a pre-defined set of statements invoked through a single SQL CALL statement. Stored procedures may contain both Teradata SQL statements and procedural statements (in Teradata, referred to as Stored Procedure Language, or SPL). Macros are similar to stored procedures but do not contain SPL.



The Data Dictionary (DD)

The Data Dictionary (DD) is an integrated set of system tables that:

- Stores database object definitions and accumulates information about **users, databases, resource usage, data demographics**, and **security rules**.
- Records specifications about **tables, views**, and **macros**.
- Contains information about **ownership, space allocation,accounting**, and **access rights (privileges)** for these objects.

Data Dictionary information is updated automatically during the processing of Teradata SQL **data definition language (DDL) statements**. It is used by the Parser to obtain information needed to process all Teradata SQL statements.

Users may access the Data Dictionary through Teradata-supplied views, if permitted by the System Administrator.

Examples of dictionary views:

- DBC.Tables—information about all tables, views, etc.
- DBC.Users—information about users
- DBC.AllRights—information about access rights
- DBC.DiskSpace—info about database space utilization

Temporary Tables:

Users can store the results of multiple queries of the Teradata RDBMS in tables, or they can load tables directly with data. Permanent storage of tables is necessary when different sessions and users must share table contents. However, users require some tables for only a single session. When a user requires a table for no more than one session, the creation of that table as temporary table can increase performance for the end user, because a temporary table requires no index maintenance. In addition, temporary tables require neither creation nor dropping. With temporary tables, a user can save query results for use in subsequent queries within the same session. Also, a user can break down complex queries into smaller queries by storing results in a temporary table for use during the same session. When the session ends, the system automatically drops the temporary table.

Temporary tables are very useful for solving problems that require "temporary" results or that require multiple SQL steps. They are particularly useful in de-normalizing tables to make query run faster, especially when multiple tables are used. Two of the many processes that can be addressed using temporary tables are:

- To create summary tables
- To translate a history table to a repeating group table

There are three types of temporary tables:

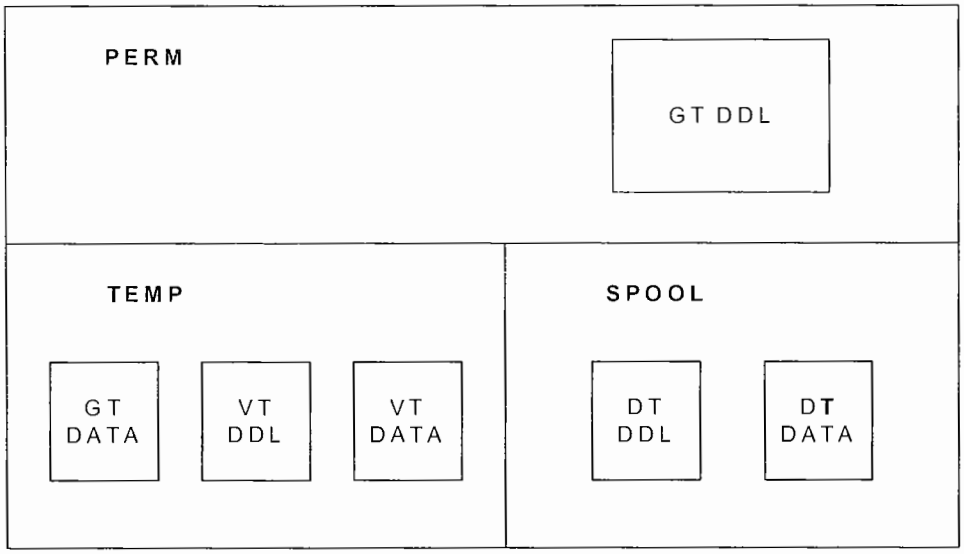
- Derived table (DT)
- Volatile table (VT)
- Global temporary table (GT)

Where is Temporary Table Created

In Teradata, there are three types of space:

- Permanent Space (PERM)
- Temporary Space (TEMP)
- Spool Space (SPOOL)

The following figure shows the space usage of each of the temporary tables:



As illustrated above, there are three types of temporary tables in Teradata, anyone of which will have advantages over traditional temporary table creation.

Derived Table

- Local to the query
- Incorporated into SQL query syntax
- Spool rows discarded when query finishes
- No system data dictionary involvement

Volatile Temporary Table

A volatile temporary table resides in memory but does not survive across a system restart. If a user needs a temporary table for a single use only, they should define a volatile temporary table. Using volatile temporary tables improves performance even more than using global temporary tables, since the system does not store the definitions of volatile temporary tables in the data dictionary. Moreover, users require no privilege to access volatile temporary tables.

- Local to a session
- Uses CREATE VOLATILE TABLE syntax (see below section)
- Discarded automatically at session end
- No system data dictionary involvement

Global Temporary Table

Global temporary tables are tables that exist only for the duration of the SQL session in which they are used. The contents of these tables are private to the session, and the system automatically drops the table at the end of that session. However, the system saves the global temporary table definition permanently in the data dictionary. In addition, global temporary tables allow the database administrator to define a template in the schema that a user can reference for their exclusive use during a session.

- Local to a session
- Uses CREATE GLOBAL TEMPORARY TABLE syntax (see below section)
- Materialized instance of table discarded at session end
- Creates and keeps table DDL in system data dictionary

Derived Table

Derived Table Example:

Derived table can be used to forces the Optimizer to get the rows it needs from the tables to be joined before the join processing, hence improves join processing by reducing the number of rows compared during the join. It is especially useful for aggregate processing which uses GROUP BY clause. OLAP functions are excellent candidates for the use of derived tables, in particular when the requirement is to perform a statistical function on an aggregation.

The following SQL is an example to find the top three selling items across all stores. It must first aggregate the sales by Product_ID using a derived table. Once the aggregation is done in spool, it then applies the RANK function to obtain the highest three:

```
SELECT prodid, sumsales, RANK(sumsales)
FROM (SELECT prodid, sum(sales) FROM salestbl GROUP BY 1) AS tmp(prodid, sumsales)
QUALIFY RANK(sumsales) <= 3;
```

Limitations of Derived Table:

- Derived table only exist for the duration of the query
- Can not create index
- Can not collect statistics

Volatile Temporary Table

Create Volatile Temporary Table Syntax:

```
CREATE VOLATILE TABLE vt_deptsal, NO FALLBACK
(
deptno SMALLINT,
avgsal DECIMAL(9,2),
maxsal DECIMAL(9,2),
minsal DECIMAL(9,2),
sumsal DECIMAL(9,2),
empcnt SMALLINT
)
```

PRIMARY INDEX (deptno)
ON COMMIT PRESERVE ROWS;

Volatile Table Usage Considerations:

- Materialized in temporary space, no perm space required
- No Data Dictionary access, DDL kept in cache
- Is local to the session, can be used by multiple queries in the session
- Each session has its own materialized instance, and may return different results
- Dropped automatically at session end or manually anytime
- Requires CREATE VOLATILE TABLE statement
- **PRESERVE ROWS** indicates keep table rows at transaction end

Volatile Table Limitations:

- Can not collect statistics
- Can not create secondary index
- Can not be ALTERed
- Can not be loaded by Fastload or Multiload
- Can not survive a system reset

Global Temporary Table

Create Global Temporary Table Syntax:

```
CREATE GLOBAL TEMPORARY TABLE gt_deptsal, NO FALLBACK
(
deptno SMALLINT,
avgsal DECIMAL(9,2),
maxsal DECIMAL(9,2),
minsal DECIMAL(9,2),
sumsal DECIMAL(9,2),
empcnt SMALLINT
)
PRIMARY INDEX (deptno)
```

Global Temporary Table Usage Considerations:

- DDL statement is required, is permanent and kept in system Data Dictionary
- Is materialized by the first DML statement accessing the table
- Each instance of a global temporary table is local to a session
- Materialized table is dropped automatically at session end
- Requires DML privileges necessary to materialize the table
- Space is charged against an allocation of temporary space
- Table can survive a system restart

- Has **ON COMMIT PRESERVE/DELETE** options
- Can be ALTERed
- **DROP TEMPORARY TABLE** drops local instance of table only
- Can create and drop secondary index
- Can collect and drop statistics

Global Temporary Table Limitations:

- Can materialize up to 32 global temporary tables per session

Materialized table contents aren't sharable with other sessions

Macros

You can create a **macro** that defines a sequence of Teradata SQL statements (and, optionally, Teradata report-formatting statements). A macro can contain one or more SQL statements. A macro may also contain comments.

When you execute the macro, the statements execute as a single transaction. Macros reduce the number of keystrokes needed to perform a complex task. This saves you time, reduces the chance of errors, reduces the communication volume to Teradata, and allows efficiencies internal to Teradata.

Macros

A macro is a predefined setof SQL statements.

May be created for frequently occurring queries or sets of operations.

Features and Benefits of Macros

- SQL that is stored in the Data Dictionary.
- Can be modified and executed at will.
- Optimized at execution time.
- Executed by a single EXECUTE command.
- Can accept user-provided parameter values.
- Simplify and control access to the system.
- Enhance system security.
- Reduce LAN/Channel traffic because they reduce the size of the query transmitted from the client application.
- Available to all connected hosts because they are stored in the Teradata DD.

Teradata Tools

Query Submitting Tools

Teradata provides **two front-end SQL** query products.

BTEQ

BTEQ stands for Basic Teradata Query utility. It is an SQL front-end that runs on all client platforms. It supports both interactive ad hoc and scripted batch queries and provides standard report writing and formatting functionality. It also provides a basic import/export capability. Other

utilities are available for these functions, particularly when larger tables are involved. When a user logs on with BTEQ, they can set the session number they need before logging on. Using multiple sessions may improve performance for certain operations.

Teradata SQL Assistant

Teradata SQL Assistant (formerly called **Queryman**) is an SQL front-end for Teradata, as well as any other ODBC compliant database. It offers a full menu of services including query history, timings, status, row counts, random sampling, and limited answer sets. It provides an import/export feature between database and PC and also allows export to Excel and Access.

Customer quote:

"The Teradata SQL Assistant product allows our development organization to use one product for SQL access to any of our different DBMS systems using each database's ODBC connectivity. Because Teradata SQL Assistant is installed on the PC, it is a very efficient way to provide access to our DBMS without establishing an ID on a host system."

Query Submitting Tools

- **BTEQ**
  - **Basic Teradata Query utility**
  - **SQL front-end**
  - **Report writing and formatting features**
  - **Interactive and batch queries**
  - **Import/Export across all platforms**

- **Teradata SQL Assistant (formerly Queryman)**
  - **SQL front-end for ODBC compliant databases**
  - **Historical record of queries including:**
    - **Timings**
    - **Status**
    - **Row counts**
  - **Random sampling feature**
  - **Limit amount of data returned**
  - **Import/Export between database and PC**
  - **Export to Excel or Access**

FastLoad Utility

Description

FastLoad is a command-driven utility you can use to quickly load large amounts of data in an empty table on a Teradata Database.

You can load data from:

- Disk or tape files on a channel-attached client system
- Input files on a network-attached workstation
- Special Access module (AXSMOD) or input module (INMOD) routines you write to select, validate, and preprocess input data
- Any other device providing properly formatted source data FastLoad uses multiple sessions to load data. It loads data into a single table on a Teradata system per job. If you

want to load data into more than one table in the system, you can submit multiple FastLoad jobs-one for each table.

How It Works

FastLoad processes a series of FastLoad commands and Teradata SQL statements you enter either interactively or in batch mode. You use the FastLoad commands for session control and data handling of the data transfers. The Teradata SQL statements create, maintain and drop tables on the Teradata Database.

During a load operation, FastLoad inserts the data from each record of your data source into one row of the table on a Teradata Database. The table on the Teradata system receiving the data must be empty and have no defined secondary indexes.

What It Does

When you invoke FastLoad, the utility executes the FastLoad commands and Teradata SQL statements in your FastLoad job script. These direct the FastLoad utility to:

1. Log you on to Teradata for a specified number of sessions, using your username, password and tdpid/acctid information.
2. Load the input data into the FastLoad table on the Teradata system.
3. Log you off from Teradata.

FastLoad Utility

- **Fast batch mode utility for loading empty (unpopulated) tables**
- **Automatic Restart capability**
- **Error Limits may be set and Error Tables may be accessed using SQL**
- **Restartable INMOD and Access Module routine capability**
- **Ability to load data in several stages**

MultiLoad Utility

Description

The MultiLoad utility gives you an efficient way to deal with batch maintenance of large databases. MultiLoad is a command -driven utility you can use to do fast, high-volume maintenance on multiple tables and views of a Teradata Database. Using a single MultiLoad job, you can do a number of different import and delete tasks on database tables and views:

- Each MultiLoad import task can do multiple data insert, update, and delete functions on up to five different tables or views.
- Each MultiLoad delete task can remove large numbers of rows from a single table. You can use MultiLoad to import data from:
- Disk or tape files on a channel-attached client system
- Input files on a network-attached workstation
- Special input module (INMOD) programs you write to select,validate, and preprocess input data
- Access modules
- Any device providing properly formatted source data The table or view in the database receiving the data can be any existing table or view for which you have access privileges for the maintenance tasks you want to do.

How it Works

MultiLoad processes a series of MultiLoad commands and Teradata SQL statements you enter usually as a batch mode job script. You use the MultiLoad commands for session control and data handling of the data transfers. The Teradata SQL statements do the actual maintenance functions on the database tables and views.

MultiLoad Utility

- Supports up to five populated tables.
- Performs block level operations against populated tables and is good for high percentage updates.
- Affected data blocks only written once.
- Multiple operations with one pass of input files.
- Uses conditional logic for applying changes.
- Supports INSERTs, UPDATEs, DELETEs and UPSERTs;typically with batch inputs from a host file.
- Supports restartable, parameterized INMODs and Access Modules.
- Errors reported and collected in error tables.
- Provides automatic Restart capability.

FastExport Utility

Description

FastExport is a command-driven utility that uses multiple sessions to quickly transfer large amounts of data from tables and views on the Teradata Database to a client-based application.

You can export data from any table or view to which you have the SELECT access privilege. The destination for the exported data can be:

- A file on your channel-attached or network-attached client system.
- An Output Modification (OUTMOD) routine you write to select, validate, and preprocess the exported data.

How it Works

FastExport processes a series of FastExport commands and Teradata SQL statements you enter, usually as a batch mode job script. The FastExport commands provide the session control and data handling specifications for the data transfer operations. The Teradata SQL statements perform the actual data export functions on the Teradata tables and views.

What it Does

When you invoke FastExport, the utility executes the FastExport commands and Teradata SQL statements in your FastExport job script.

These direct the FastExport utility to:

1. Log you on to Teradata for a specified number of sessions, using your username, password and tdpid/acctid information.

2. Retrieve the specified data from Teradata, in accordance with your format and selection specifications.

3. Export the data to the specified file or OUTMOD routine on your client system.

4. Log you off of Teradata.

FastExport Utility

- Exports large volumes of formatted data from Teradata to a host file or user-written application.
- Uses multiple sessions.
- Export from multiple tables.
- Fully automated restart.

TPump Utility

TPump (Teradata Parallel Data Pump) is a utility that provides high volume batch maintenance of large Teradata Databases. It enables acquisition of data from the client with low processor utilization.

The Support Environment enhances TPump's functionality. In addition to coordinating activities involved in TPump tasks, it provides facilities for managing file acquisition, conditional processing, and certain DML (Data Manipulation Language) and DDL (Data Definition Language) activities on the Teradata Database. The Support Environment enables an additional
level of user control over TPump.

TPump uses row-hash locks, making concurrent updates on the same table a possibility.

TPump has a built-in resource governing facility that allows the operator to specify how many updates occur (the statement rate) minute by minute, then change the statement rate while the job continues running. This utility can be used to increase the statement rate during windows when TPump is running by itself, then decrease the statement rate later on if
users log on for ad-hoc query access.

TPump can always be stopped and all work will be committed as run.

TPump does not utilize one of the loader slots used by FastLoad, MultiLoad, and FastExport, so many TPump jobs can run concurrently.

The slide on the opposite page identifies the principal features of the TPump utility.

TPump Utility

_ Allows near real-time updates from transactional systems into the warehouse.
_ Allows constant loading of data into a table.
_ Performs INSERT, UPDATE, DELETE, and ATOMIC UPSERT operations, or a combination, to more than 60 tables at a time.
_ High-volume SQL-based continuous update of multiple tables
_ Allows target tables to:

1) Have secondary indexes, referential integrity, constraints and enabled triggers.

2)Be MULTISET or SET.

3) Be populated or empty.

– Allows conditional processing.
– Supports automatic restarts.
– No session limit—use as many sessions as necessary.

– No limit to the number of concurrent instances.
– Uses row-hash locks, allowing concurrent updates on the same table.
– Can be stopped at any time with work committed with no ill effect.
– Designed for highest possible throughput.
– Gives users the control over the rate per minute (throttle) at which statements are sent to the database either dynamically or by script.

Creating a Teradata Database

A Teradata Database

A database provides a logical grouping of information (tables, views, and macros). A database could be considered a passive repository, because it is used solely to store other database objects. This is different from a user, which we will learn about on the next page.

Perm Space

All databases have a defined upper limit of **Permanent Space**. Permanent Space is used for storing the data rows of tables. Perm Space is not pre-allocated. It represents a maximum limit.

Spool Space

All databases also have an upper limit of **Spool Space**. If there is no limit defined for a particular database or user, limits are inherited from parents. Theoretically, a user could use all unallocated space in the system for their query. **Spool Space is Temporary Space used to hold intermediate query results or formatted answer sets to queries**. Once the query is complete, the Spool Space is released.

Example: You have a database with total disk space of 100GB. You have 10GB of user data and an additional 10GB of overhead. What is the maximum amount of Spool Space available for queries?

Answer: 80GB. All of the remaining space in the system is available for spool.

Temp Space

The third type of space is **Temporary Space**. Temp Space is used for global temporary tables, and these results remain available to the user until the session is terminated. Tables created in Temp Space will survive a restart. Temp Space is permanent space currently not used.

A Teradata Database

A Teradata Database is a defined, logical repository for:

- Tables (require Perm Space)
- Views (use no Perm Space)
- Macros (use no Perm Space)
- Triggers (use no Perm Space)
- Stored Procedures (require Perm Space)

Space limits may be specified for a database:

- **Perm Space**—max amount of space available for tables, not allocated
- **Spool Space**—max amount of work space available for requests, used to hold the intermediate result set
- **Temp Space** —used for global temporary tables
- A database with no Perm Space can hold views, macros, and triggers (which do not require Perm Space)
- A Teradata Database is created with the CREATE DATABASE command by a user with the appropriate privileges.

A Teradata User

A **user** can be thought of as a collection of tables, views, macros, triggers, and stored procedures.

A user is the same as a database except that a user can actually log on to the RDBMS. To logon, a user must have a **password**. A user may or may not have Perm Space. A user could be considered an active repository, because it is used to log on to the system as well as to store other database objects.

Users can access other databases depending on the privileges they have been granted.

Users are created with the SQL statement **CREATE USER.**

A Teradata User

- **A Teradata user is a database with an assigned password.**
- **A Teradata user may also own tables, views, macros, triggers, and stored procedures, but users with no Perm Space may not own tables or stored procedures.**
- **A user may logon to Teradata and access objects within:**
  - Itself
  - Other databases for which it has access rights
- **A user is created with the CREATE USER command.**

Example:

```
CREATE USER new_user FROM existing_user AS
PERMANENT = 10000000,PASSWORD = 'lucky_day',SPOOL = 20000000
,TEMPORARY = 20000000;
```

"New_user" is owned by "existing_user."

A user is *empty* until objects are created within it.

Teradata Space Management

Before defining application users and databases, the Database Administrator should first create a special administrative user and assign most of the space in the system to that user. This space comes from user DBC. User DBC becomes the owner of the administrative user.

There should be enough space left in user DBC to accommodate the growth of system tables and logs.

As the administrative user creates additional users and databases, space assigned to those objects will be subtracted from the administrative user's space.

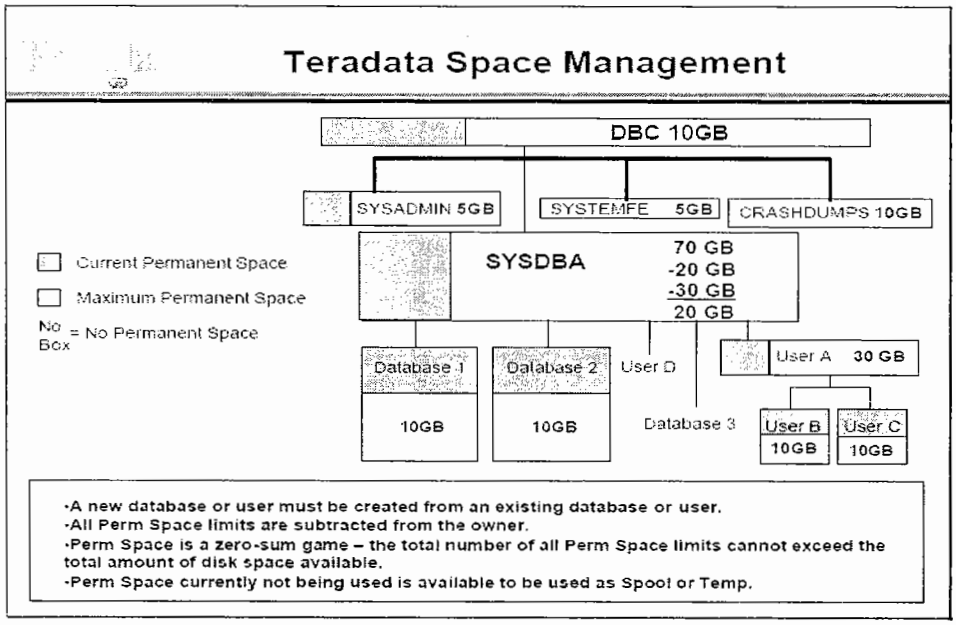
As these users and databases create subordinate objects, they will give up some of their space to these new users and databases.

The clear boxes represent all the space available in the system. The shaded boxes represent space that is being used in the system by tables. Space that is not currently being used by tables is available for use as spool or temporary space. If a table is dropped, that space becomes available to the system again.

Database 1 and Database 2 contain the production tables. Database 3 contains views and macros to control access to the data by end users and does not consume any database space. User D is an example of an end user who has no Perm Space but has been granted rights on certain views and macros to accomplish work. User A, User B, and User C are examples of application developers who need some Perm Space to hold sample data for program checkout.

Most database vendors do not handle space this way. Once space is allocated to a table, that space cannot be made available again without the Database Administrator having to do reorganizations and repartitioning of the data.

The way Teradata handles space management is different from other database implementations.



Storing and Accessing Data Rows

How Does Teradata Store Rows?

Each AMP is designed to hold a portion of the rows of each table. An AMP is responsible for the storage, maintenance and retrieval of the data under its control. Teradata uses hashing to randomly and evenly distribute data across all AMPs for balanced performance.

Teradata's automatic hash distribution eliminates costly data maintenance tasks. There is no specific order to the placement of the data. The benefits of having unordered data are that they don't need any maintenance to preserve order, and they are independent of any query being submitted. The DBA can spend more time on strategic development activities. As a result, strategic business data is more accessible to the users.

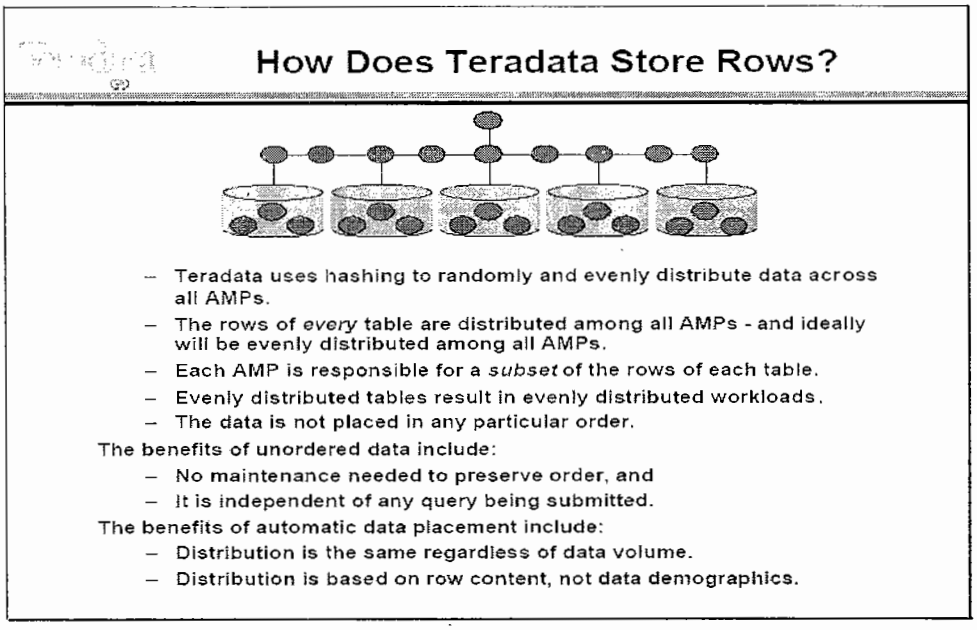
The benefits of automatic data placement include:

- Distribution is the same regardless of data volume.
- Distribution is based on row content, not data demographics.

Ideally, the rows of every table will be distributed among all of the AMPs. There may be some circumstances where this is not true. For example, what if there are fewer rows than AMPs? In this case, at least some AMPs will hold no rows from that table. (This example should be considered the exception, and not the rule.)

In an ideal situation, the rows of each table will be *evenly* distributed across all of the AMPs. Even distribution is desirable because in an operation involving all rows of the table (such as a full table scan), each AMP will have an equal portion of work to do. When workloads are not
evenly distributed, the desired response can only be as fast as the slowest AMP.

Remember: Teradata uses a Shared-Nothing Architecture - each AMP is responsible for its own portion of the database and does not share it with any other AMP.



Primary Indexes

On Teradata, the **Primary Index (PI)** is the physical mechanism for assigning a data row to an AMP and a location on the AMPs disks. **Indexes** are also used to access rows from a table without having to search the entire table.

Choosing a **Primary Index** for a table is perhaps the most critical decision a database designer makes. The choice will affect the distribution of the rows of the table and the performance of the table in a production environment. Althoug h many tables use combined columns as the Primary Index choice, the examples we use here are single-column indexes.

There are two types of primary indexes —**unique (UPI)** and **non-unique (NUPI)**.

A **Unique Primary Index (UPI)** is a column that has no duplicate values. UPIs are desirable because they guarantee uniform distribution of table rows. With a UPI, there is no duplicate row checking done during a load, which makes it a faster operation.

Because it is not always feasible to pick a Unique Primary Index, it is sometimes necessary to pick a column (or columns) which have nonunique, or duplicate values. This type of index is called a **Non-Unique Primary Index** or **NUPI**. While not a guarantor of uniform row distribution, the degree of uniqueness of the index will determine the degree of uniformity of the distribution. Because all rows with the same PI value end up on the same AMP, columns with a small number of distinct values that are repeated frequently do not make good PI candidates.

Accessing the row by its Primary Index value is the most efficient way to access a row and is always a one-AMP operation. Choosing a Primary Index is not an exact science. It requires analysis and thought for some tables and will be completely self-evident on others. Sometimes the obvious choice will be the Primary Key, which is known to be unique.Sometimes the choice of Primary Index may have to do with join performance and known access paths, and will therefore be a different choice than the Primary Key of the table.

Teradata is unique in hashing data directly to a physical address on disk, creating an even distribution of data. This allows a balanced application of parallelism, and also avoids imbalance due to data skew.

Note: While you cannot change the Primary Index itself, values in a Primary Index column may be changed. Teradata simply rehashes that row to its new location, based on the Primary Index value.

Primary Indexes

· The physical mechanism used to assign a row to an AMP
· A table must have a Primary Index
· The Primary Index cannot be changed

UPI · If the index choice of column(s) is unique, we call this a *UPI* (Unique Primary Index).

· A UPI choice will result in even distribution of the rows of the table across all AMPs. UPI's guarantee even data distribution and eliminate duplicate row checking.

NUPI · If the index choice of column(s) isn't unique, we call this a *NUPI* (Non-Unique Primary Index).

· A NUPI choice will result in even distribution of the rows of the table proportional to the degree of uniqueness of the index.

Why would you choose an Index that is different from the Primary Key?
• Join performance
• Known access paths

Creating a Primary Index

The Primary Index is always designated as part of the **CREATE TABLE** statement. When a table is created, it must have a Primary Index specified. The Primary Index may consist of a single column or a combination up to 64 columns.

If you do not specify a Primary Index in your CREATE TABLE statement, the system will use the Primary Key.

If you have not specified a Primary Key, the system will choose the first unique column.

If there are no unique columns, the system will use the first column in the table and designate it as a Non-Unique Primary Index.

Once you choose a Primary Index for a table, it cannot be changed to something else using an ALTER TABLE command. If an alternate choice of column(s) is desired for the PI, it is necessary to drop and recreate the table, then choose a new PI.

Creating a Primary Index

· A Primary Index is defined at table creation.
· It may consist of a single column or a combination of up to 64 columns.

UPI:

CREATE TABLE sample_1 (col_a INT,col_b INT,col_c INT)UNIQUE PRIMARY INDEX (col_b);

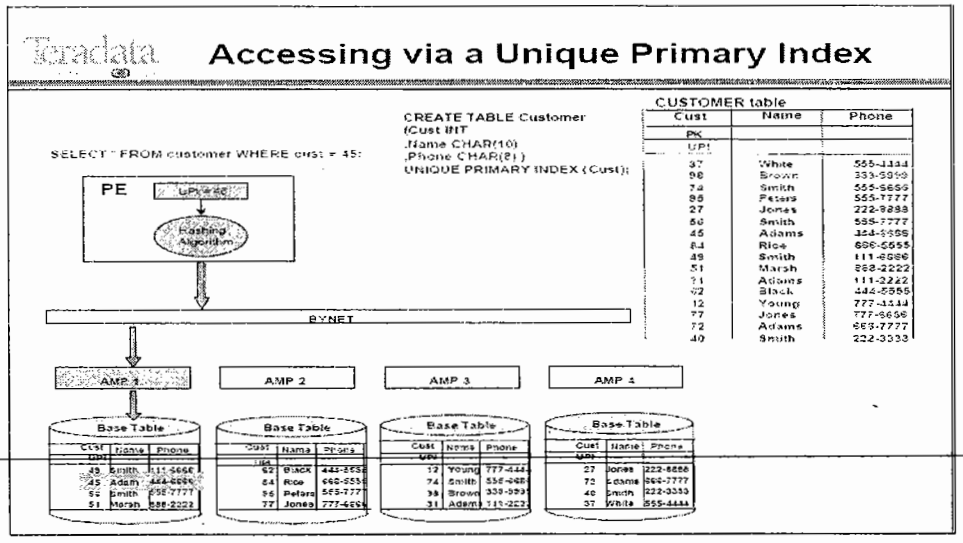
NUPI:

CREATE TABLE sample_2 (col_x INT,col_y INT,col_z INT)PRIMARY INDEX (col_x);
Note: Changing the Primary Index requires dropping and recreating the table.

Accessing via a Unique Primary Index

A **Primary Index operation** is always a one -AMP operation. In the case of a UPI, the one-AMP access can return, at most, one row. In the example on the facing page, we are looking for the row whose primary index value is 45. By specifying the PI value as part of our selection criteria, we are guaranteed that only the AMP containing the specified row will be searched.

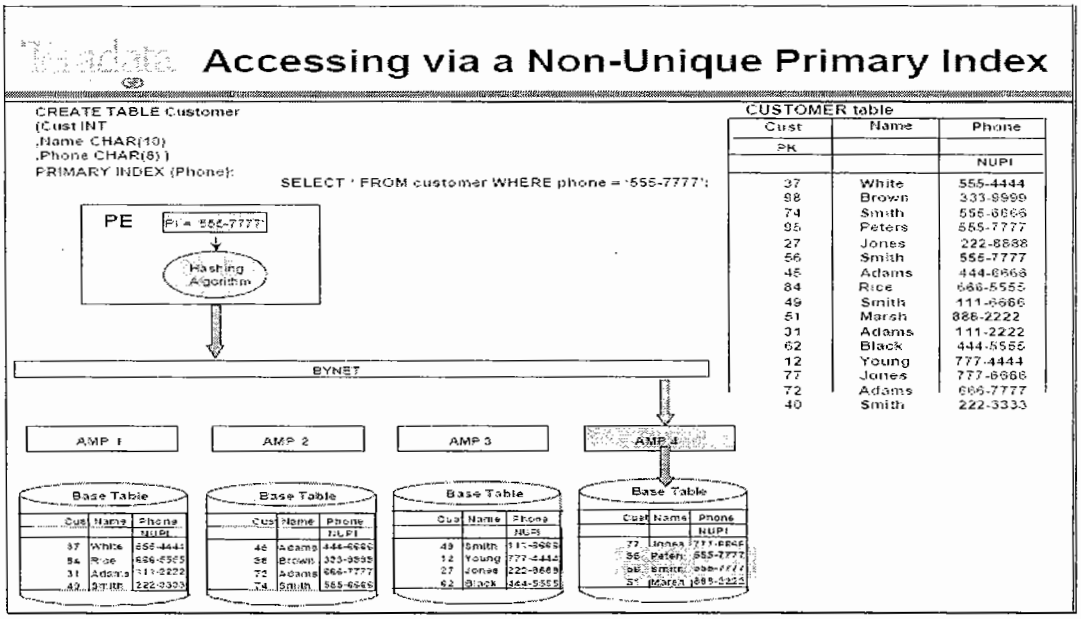
The correct AMP is located by taking the PI value and passing it through a hashing algorithm. Hashing takes place in the Parsing Engine. The output of the hashing algorithm contains information that will point to a specific AMP. Once it has isolated the appropriate AMP, finding the row is quick and efficient.



Accessing via a Non-Unique Primary Index

A **Non-Unique Primary Index operation (NUPI)** is also a one –AMP operation. In the case of a NUPI, the one-AMP access can return zero to many rows. In the example on the facing page, we are looking for the rows whose primary index value is 555-7777. By specifying the PI value as part of our selection criteria, we are once again guaranteeing that only the AMP containing the required rows will need to be searched.

The correct AMP is located by taking the PI value and passing it through a hashing algorithm executed in the Parsing Engine. The output of the hashing algorithm will once again point to a specific AMP. Once it has isolated the appropriate AMP, it must find all *rows* that have the specified value. In the example, the AMP returns two rows.



Primary Keys and Primary Indexes

While it is true that many tables use the same columns for both Primary Indexes and Primary Keys, **Indexes are conceptually different from Keys**. The table on the facing page summarizes those differences.

- A **Primary Key** is a relational data-modeling term. It defines, in the logical model, the columns that uniquely identify a row.
- A **Primary Index** is a physical database implementation term that defines the actual columns used to distribute and access rows in a table.

A significant percentage of the tables in any database will use the same column(s) for both the PI and the PK. However, one should expect that in any real scenario there might be some tables that will not conform to this rule. Only after a careful analysis of the type of processing that will take place can the tables be properly evaluated for PI candidates. Remember, changing your mind about the Primary Index means dropping and recreating the table.

Note: if you make Teradata aware of your Primary Key and you choose a different Primary Index, the Primary Key automatically becomes a Unique Secondary Index (USI).

Primary Keys and Primary Indexes

Indexes are conceptually different from *keys:*

- A *PK* is a relational modeling convention which uniquely identifies each row.
- A *PI* is a Teradata convention which determines how the rows are stored and accessed.

Primary Key	Primary Index
Logical concept of data modeling	Physical mechanism for access and storage
Teradata doesn't need to recognize	Each table must have exactly one
No limit on column numbers	64-column limit
Documented in data model (Optional in CREATE TABLE)	Defined in CREATE TABLE statement
Must be unique	May be unique or non-unique
Uniquely identifies each row	Used to place and locate each row on an AMP
Values should not change	Values may be changed (Del+ Ins)
May not be NULL—requires a value	May be NULL
Does not imply an access path	Defines most efficient access path
Chosen for logical correctness	Chosen for physical performance

- A significant percentage of tables may use the same columns for both the PK and the PI.
- A well-designed database will use a PI that is different from the PK for some tables.

Row Distribution Using a UPI

The Teradata hashing algorithm predictably distributes and retrieves rows across all AMPs. The same value stored in the same data type will always produce the same hash value. If the Primary Index is unique, Teradata can distribute the rows evenly. If the Primary Index is non-unique, but there are a fairly even number of rows per index value, the table will still distribute evenly. But if there are hundreds or thousands of rows for some index values, the distribution will probably be skewed.

In the example on the facing page, the Order_Number is used as a unique primary index. Since the primary index value for Order_Number is unique, the distribution of rows among AMPs is uniform. This assures maximum efficiency because each AMP does approximately the same amount of work. No AMPs sit idle waiting for another AMP to finish a task.

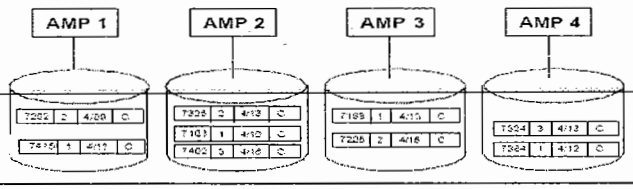
This way of storing the data provides for maximum efficiency and makes the best use of the parallel features of the Teradata system.

Row Distribution Using a UPI

Order

Order Number PK UPI	Customer Number	Order Date	Order Status
7325	2	4/13	O
7324	3	4/13	O
7415	1	4/13	O
7103	1	4/10	O
7225	2	4/15	O
7384	1	4/12	O
7402	3	4/16	O
7188	1	4/13	O
7202	2	4/09	O

- The PK column(s) will often be used as a UPI.
- PI values for Order_Number are known to be unique (it's a PK).
- Teradata will distribute different index values evenly across AMPs.
- Resulting row distribution among AMPs is uniform.



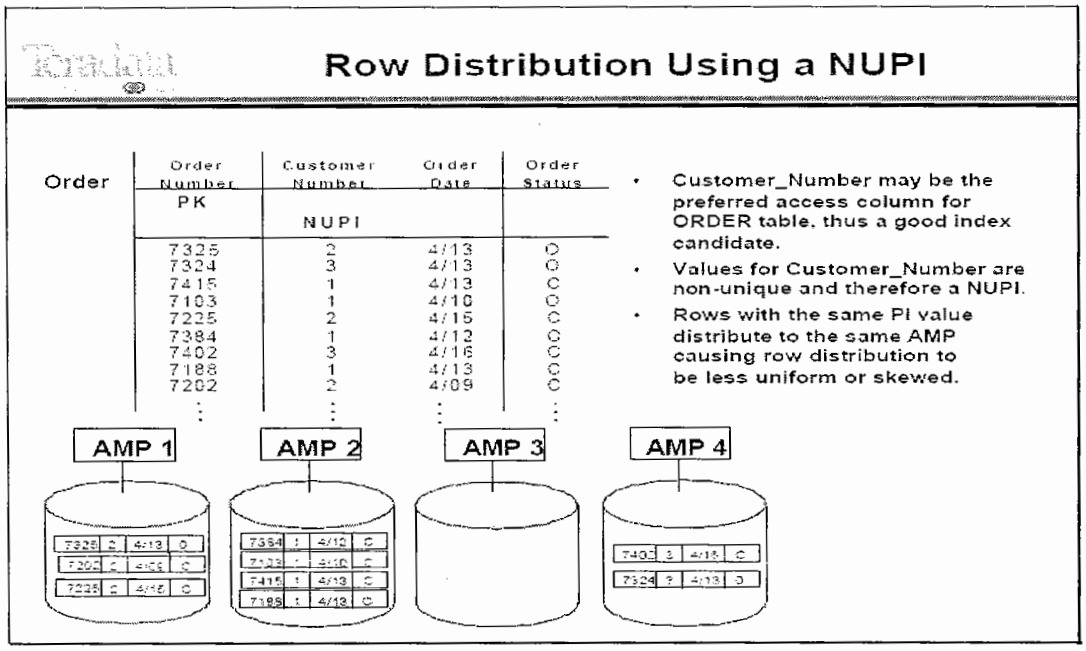
Row Distribution Using a NUPI

In the example on the facing page, **Customer_Number** has been used as a non-unique Primary Index (NUPI). Note that row distribution among AMPs is uneven. All rows with the same primary index value (with the same customer number) are stored on the same AMP.

Customer_Number has three possible values, so all the rows are hashed to three AMPs, leaving the fourth AMP without rows from this table. While this distribution will work, it is not as efficient as spreading all rows among all AMPs.

AMP 2 has a disproportionate number of rows and AMP 3 has none. In an all-AMP operation, AMP 2 will take longer than the other AMPs. The operation cannot complete until AMP 2 completes its tasks. Overall operation time is increased and some AMPs are under-utilized.

This illustrates how NUPIs can create irregular distributions, called skewed distributions. AMPs that have more than an average number of rows will take longer for full-table operations than other AMPs. Because an operation is not complete until all AMPs have finished, the operation where distribution is skewed will take longer than it would if all AMPs were utilized evenly.

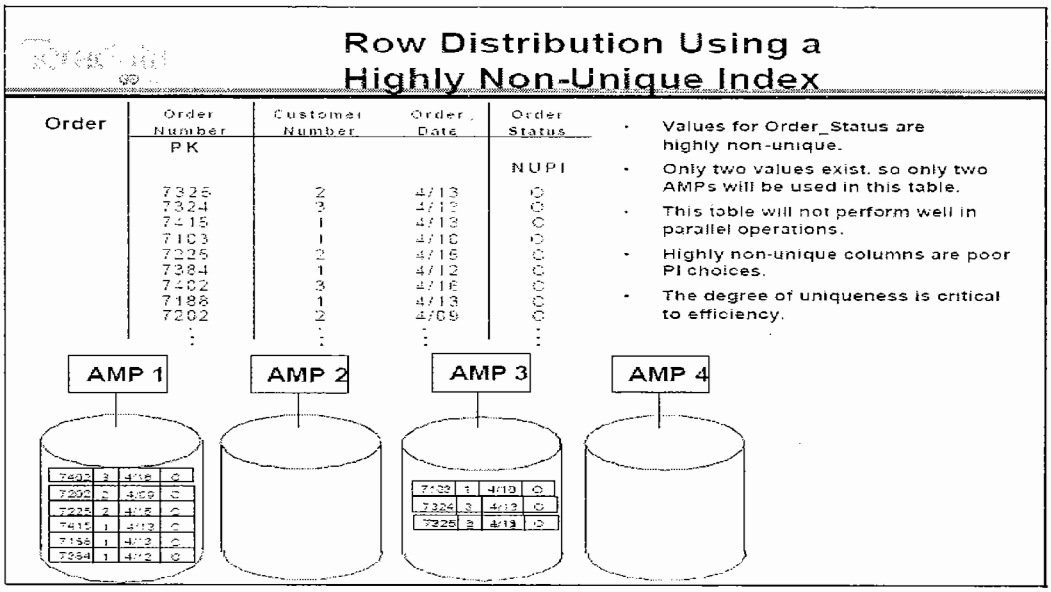


Row Distribution Using a Highly Non-Unique Index

The example on the facing page uses **Order_Status** as a **NUPI**.Order_Status is a poor choice, because it yields the most uneven distribution. Because there are only two possible values for Order_Status, all rows are placed on two AMPs.

STATUS is an example of a **highly non-unique Primary Index**.

When choosing a Primary Index, never pick a column with a severelylimited value set. The degree of uniqueness is critical to efficiency. Choose NUPIs that allow all AMPs to participate fairly equally.



Partitioned Primary Index

In Teradata Database V2R5 there is a new indexing mechanism called Partitioned Primary Index (PPI). PPI is used to improve performance for large tables when you submit queries that specify a range constraint. PPI allows you to reduce the number of rows to be processed by using a new technique called partition elimination. PPI will increase performance for incremental data loads, deletes, and data access when working with large tables with range constraints. PPI is useful for spontaneous dropping of old data and rapid addition of new data.

How Does PPI Work

Data distribution with PPI is still based on the **Primary Index**:

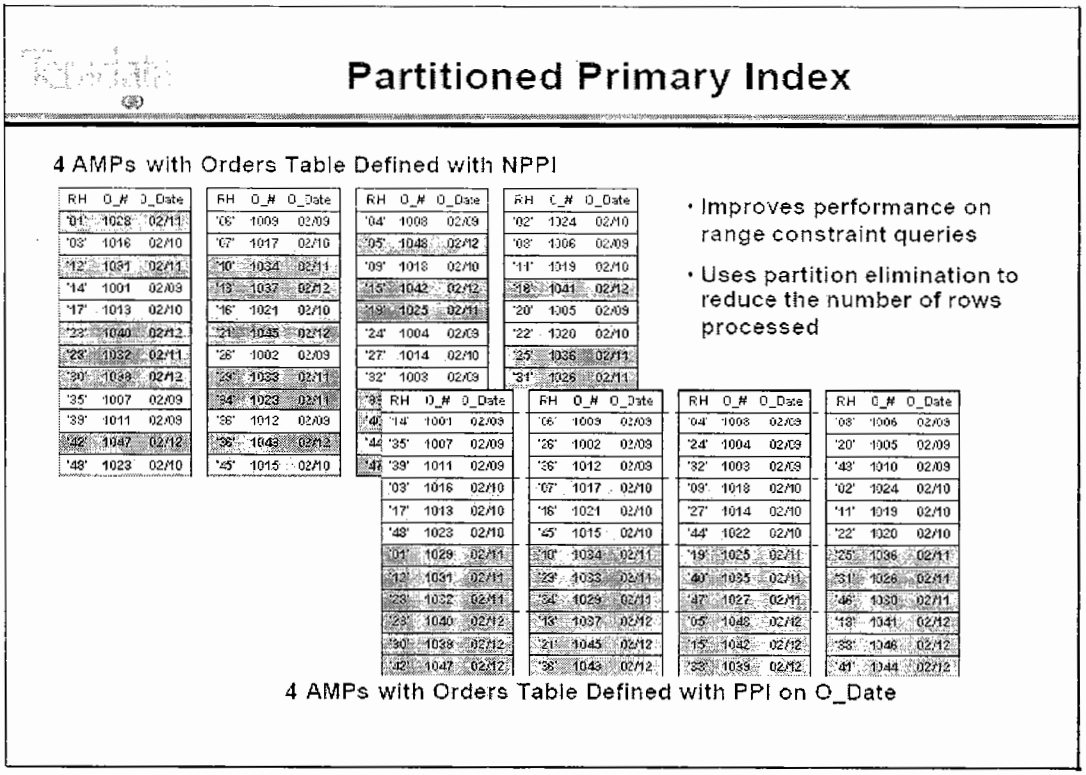
Primary Index >> Hash Value >> Determines which AMP gets the row

With PPI, the **ORDER** in which the rows are stored on the AMP is affected. Using the traditional method, No Partitioned Primary Index (NPPI), the rows are stored in row hash order.

Using PPI, the rows are stored first by partition and then by row hash. In our example, there are four partitions. Within the partitions, the rows are stored in row hash order.

Data Storage Using PPI

To store rows using PPI, specify Partitioning in the CREATE TABLE statement. The query will run through the hashing algorithm as normal, and come out with the Base Table ID, the Partition number(s), the Row Hash, and the Primary Index values.



Primary Index Mechanics

Hashing Primary Index Values

The diagram on the facing page gives an overview of Primary Index Hash Mapping, which is the process by which all data is distributed in the Teradata Database.

The Primary Index value is fed into the Hashing Algorithm, which produces the Row Hash. The row goes onto the Communications Layer. The Hash Maps, in combination with the Row Hash determines which AMP gets the row. The Hash Maps are a part of the Communications Layer interface.

Teradata's hash partitioning is the only approach which offers balanced processing and balanced data placement, while minimizing interconnect traffic and freeing the DBA from time-consuming reorganizations.

In this module, we will discuss how this entire process functions.

Hashing Down to the AMPs

Teradata is unique in hashing data directly to a physical address on disk, creating an even distribution of data. This allows a balanced application of parallelism, and also avoids imbalance due to data skew. Teradata's hashed data distribution scheme provides optimized performance with minimal tuning and no reorganizations, resulting in lower administration costs and reduced development time.

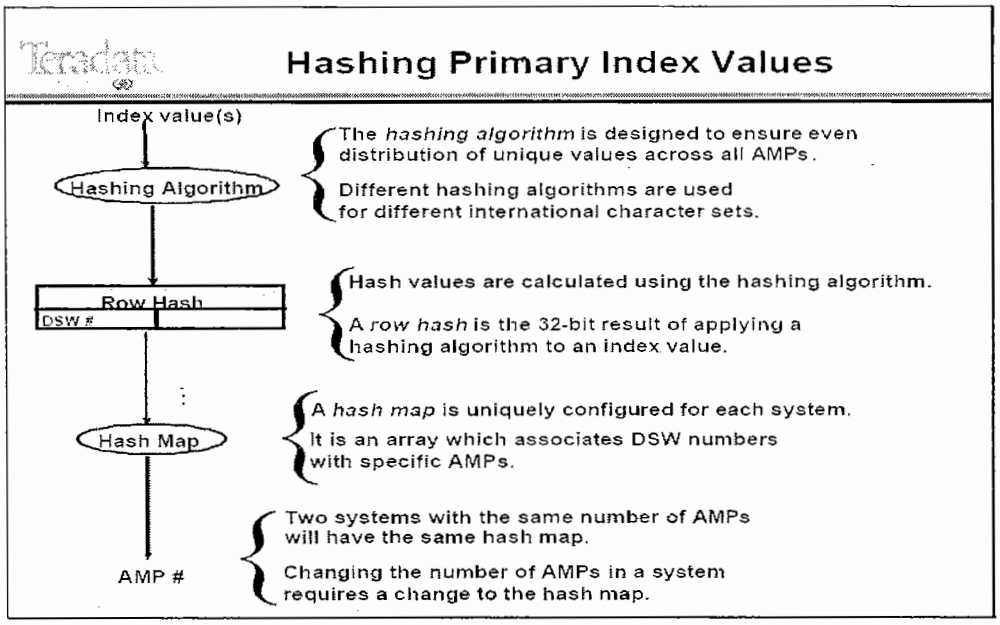
The rows of all tables are distributed across the AMPs according to their **primary index** value. The primary index value goes into the hashing algorithm. The output is a 32-bit **row hash**. The high-order 16 bits are referred to as the Destination Selection Word (DSW) and are used to identify a hash-map entry. This entry is used to identify the AMP to be targeted. The remaining 16 bits are not used to locate the AMP.

The entire 32-bit row hash will be used by the selected AMP to locate the row within its disk space.

Hash maps are uniquely configured for each **size** of system. A 50-AMP system will have a different hash map from a 30-AMP system.

A hash map is simply an array that associates bucket numbers (DSW #s) with specific AMPs.

When a system grows, new AMPs are typically added. Adding AMPs requires a change to the hash map to reflect the new total number of possible target AMPs.



Why is Automatic Distribution Good?

Hashing algorithms don't care about user-defined alphabetic order, don't use secondary structures that require reorganization, and there is never a need to sort the data before loading or inserting. There are several distinct advantages in decision support applications that flow from Teradata's use of hash partitioning. These include:

No Key Sequence

A side effect of using the hashing algorithm as an indexing mechanism is the absence of a user-defined order. This eliminates the problems encountered with the B-Tree indexing that many database vendors use.

Ease of Joining

Hash partitioning of primary index values allows rows from different tables with high affinities to be placed on the same node. By designating the columns that constitute the join

constraint between two tables as the primary index of both tables, associated rows -to-be-joined will reside on the same node. Since two rows can only be joined if they reside in the memory of one of the nodes, this co-location reduces the interconnect traffic that cross-node joins necessitate, improving query times, and freeing the BYNET for other work.

Simplicity of Set-Up

The only effort hashed data placement requires is the selection of the columns that comprise the primary index of the table. From that point on, the process of creating and maintaini ng partitions is completely automated. No files need to be allocated, sized or named. No DDL needs to be tediously created. No unload-reload activity is ever required.

Teradata Differentiator

Teradata's automatic hash distribution eliminates costly data maintenance tasks so the DBA can spend more time on strategic development activities. As a result, strategic business data is more accessible to the users.

Please see "Born to Be Parallel" in Appendix C of this course to read more about how other database vendors do data partitioning.

Why is Automatic Distribution Good?

No Key Sequence
No user defined order

Ease of Joining
Rows with high affinity are stored on same AMP
Reduces inter-connect traffic

Simplicity of Set-up

The only effort required for Teradata hashing is selection of the column(s) that comprise the Primary Index

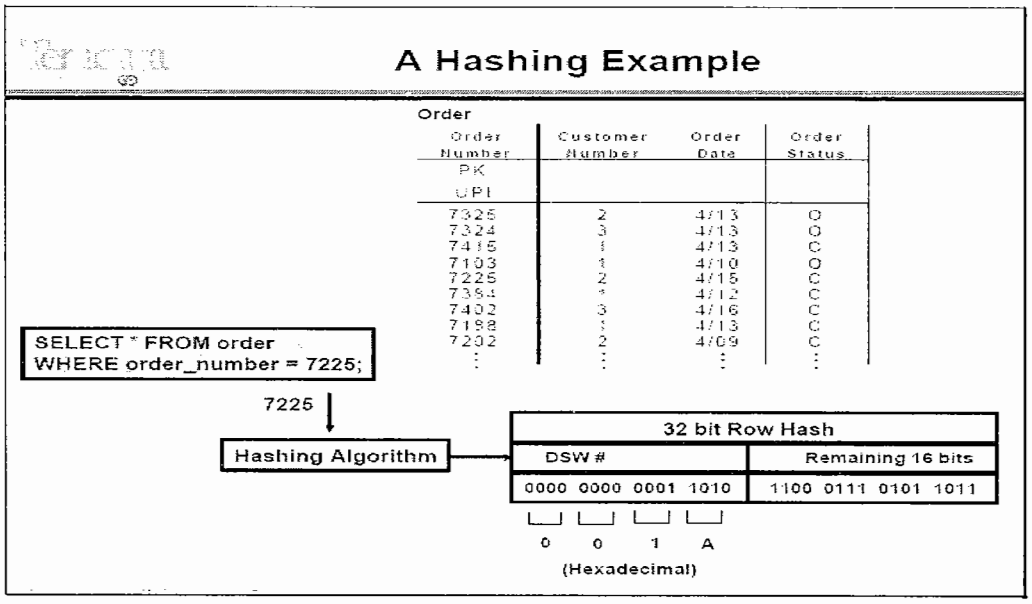
TERADATA'S AUTOMATIC HASH DISTRIBUTION ELIMINATES COSTLY DATA MAINTENANCE TASKS!

A Hashing Example

The facing page shows an example of how the hashing algorithm would produce a 32-bit row hash value on the primary index value, 7225.

The hash value is divided into two parts. The first 16 bits are the **Destination Selection Word (DSW)**.

The DSW points to a particular hash map entry, which in turns points to one AMP. The entire row hash along with the Table ID references a particular logical location on that AMP.



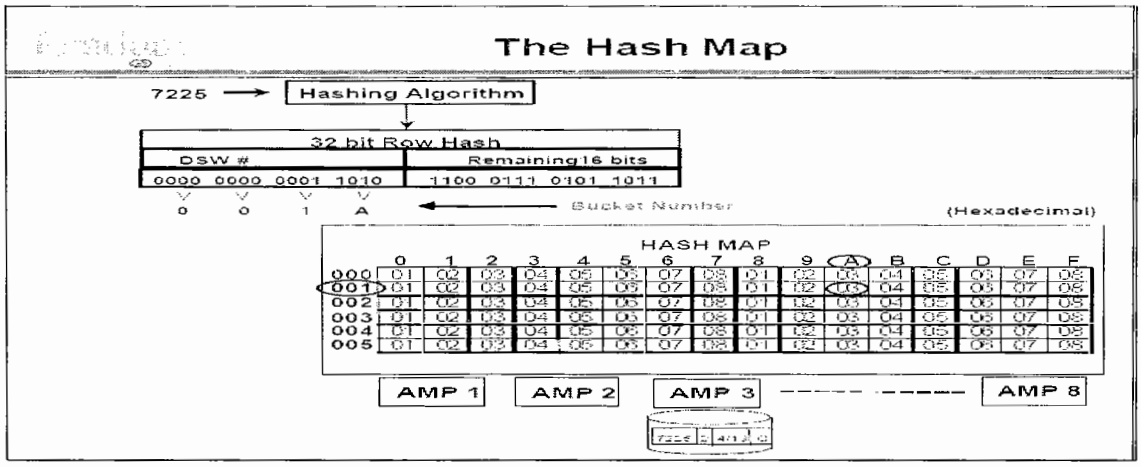
The Hash Map

A **hash map** is an array that associates DSW values with specific AMPs. The diagram on the facing page shows it folded after 16 entries and represents what a hash map might look like for an eight-AMP system.

To determine the destination AMP for a primary index operation, the hash map is checked by BYNET software using the row hash information. A message is placed on the BYNET to be sent to the target AMP using point-to-point communication.

In the example on the facing page, the bucket entry 001A (hexadecimal) contains an entry for AMP 3. AMP 3 will be the recipient of the message from the BYNET.

One major difference between Version 1 and Version 2 is the increase in the number of hash map entries from 3643 to 65,536. This increase results in better performance, and the ability to configure systems with thousands of AMPs rather than hundreds of AMPs. The increase in hash buckets also resulted in fewer hash collisions on the system.



Identifying Rows

Can two different Primary Index (PI) values come out of the hashing algorithm with the same row hash value? Yes. There are two ways this can happen:

- Two different primary index values may hash identically. This is called a **hash synonym** (or a hash collision).
- If a non-unique primary index is used, **duplicate NUPI values will produce the same row hash.**

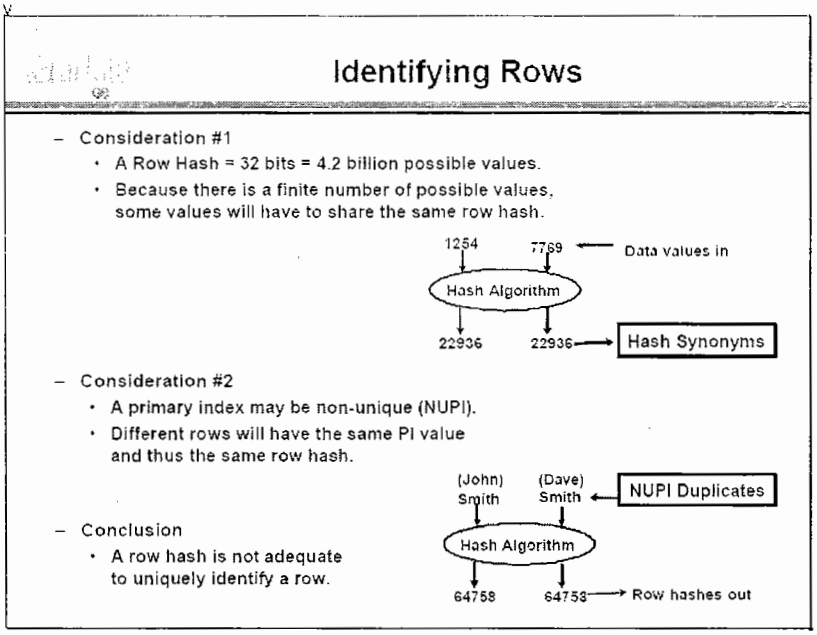
Physical Row Layout

The unique Row-ID is composed of the row hash and the uniqueness value. Remember that all rows have a unique Row-ID.

Within the User Data columns is the primary index value on which the row hash is based. The PI may represent one or several columns of data. Teradata determines the sequence of columns within the row to optimize performance.

The 2 bytes at the beginning of the row reflect the length of the row.

Normally, only the row hash portion of the Row-ID is used to locate the row. An exception occurs during Secondary Index access and is covered later in this course.



The Row-ID

To differentiate each row in a table, every row is assigned a unique **Row-ID.** The Row-ID is a combination of the row hash value plus a **uniqueness value**. The AMP appends the uniqueness value to the row hash when it is inserted. The uniqueness value is used to differentiate between PI values that generate identical row hashes.

The first row inserted with a particular row hash value is assigned a uniqueness value of 1. Each new row with the same row hash is assigned an integer value that is one greater than the current largest uniqueness value for this Row-ID.

If a row is deleted or the primary index is modified, the uniqueness value can be reused.

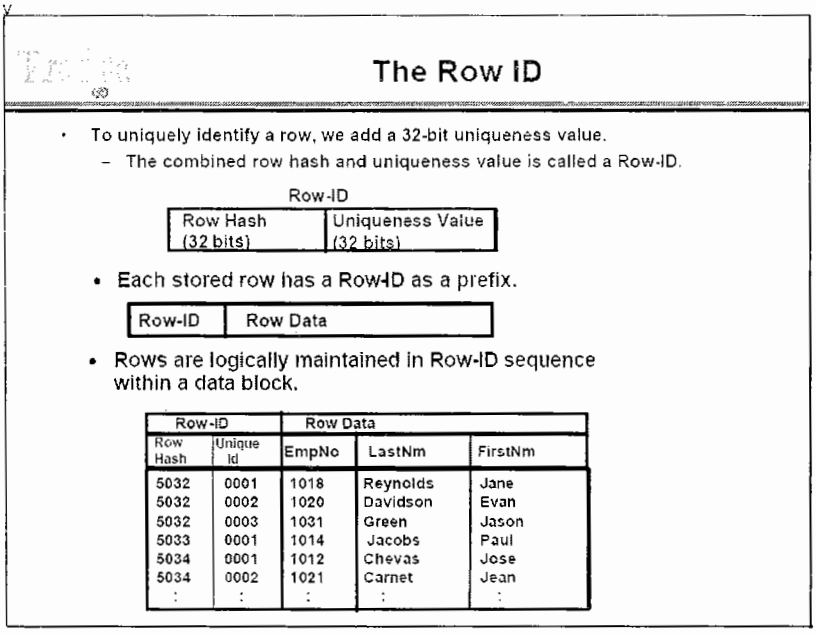
Only the row hash portion is used in primary index operations. The entire Row-ID is used for secondary index support and is discussed in a later chapter.

PPI tables have Row Keys:

Row Hash + Uniqueness Value + Partitioning Value

4 Bytes 4 Bytes 2 Bytes

Primary Index Mechanics Page 9-15



Secondary Indexes and Table Scans

Secondary Indexes

A **Secondary Index** is an alternate path to the data. Secondary indexes are used to improve performance b y allowing the user to avoid scanning the entire table during a query. A secondary index is like a primary index in that it allows the user to locate rows. Unlike a primary index, it has no influence on the way rows are distributed among AMPs. A database designer typically chooses a secondary index because it provides faster set selection.

Primary index requests require only one AMP to access rows, while secondary indexes require at least two and possibly all AMPs, depending on the index and the type o f operation. A secondary index search will typically be less expensive than a full-table scan.

Secondary indexes add overhead to the table, both in terms of disk space and maintenance, but they may be added or dropped when needed.

Secondary Indexes

Four basic ways to access a table:

- Primary index access (*one*-AMP access)
- Secondary index access (*two*-or *all*-AMP access)
- Full-Table Scan (*all*-AMP access)
- Partitioned primary index access (*all*-AMP access)

– A secondary index is an alternate path to the rows of a table.

– A table can have from 0 to 32 secondary indexes.

– Secondary indexes:

- Do not affect table distribution.
- Add overhead, both in terms of disk space and maintenance.
- May be added or dropped dynamically as needed.
- Are chosen to improve table performance.

Choosing a Secondary Index

As with primary indexes, there are two types of **secondary indexes—unique (USI)** and **non-unique (NUSI)**.

A secondary index may be specified at table creation or at any time during the life of the table. It may consist of up to 64 columns. To get the benefit of the index, the query has to specify a value for all columns in the secondary index.

A **unique secondary index (USI)** has two possible purposes:

· It can speed up access to a row which otherwise might require a full table scan without having to rely on the primary index.

· It can be used to enforce uniqueness on a column or set of columns. This is sometimes the case with a primary key that is not designated as the primary index. Making it a USI enforces the uniqueness of the PK.

A **non-unique secondary index (NUSI)** is usually specified in order to prevent full-table scans. NUSIs, however, do activate all AMPs because the value being sought might reside on many different AMPs (only primary indexes have same values on same AMPs). If the Optimizer decides that the cost of using the secondary index is greater than a table scan would be, it opts for the table scan.

As column values change, secondary indexes cause an AMP-local subtable to be built and maintained. Sccondary index **subtables** consist of rows which associate the secondary index value with one or more rows in the base table. When the index is dropped, the subtable is physically removed.

Choosing a Secondary Index

A secondary index may be defined:

· At table creation (CREATE TABLE)
· Following table creation (CREATE INDEX)
· Using up to 64 columns

USI:

· If the choice of column(s) is unique, it is called a USI (unique secondary index).
· Accessing a row via a USI typically requires 2 AMPs.

NUSI:

· If the choice of column(s) is non-unique, it is called a NUSI (non-unique secondary index).
· Accessing a row via a NUSI requires all AMPs.

USI:

CREATE UNIQUE INDEX (employee-number) on employee

NUSI:

CREATE INDEX (last-name) on employee
CREATE INDEX (last-name, first-name) on employee

NOTE:

· Secondary indexes cause an internal subtable to be built.
· Dropping the index causes the subtable to be deleted.

Unique Secondary Index (USI) Access

The facing page shows the one or two-AMP access necessary to retrieve a row via a **unique secondary index (USI)** access.

- After the row hash of the secondary index value is calculated, the hash map points us to AMP 2, containing the subtable row for this USI value.
- After locating the subtable row in AMP 2, we find the Row-ID of the base row. This base Row-ID (which includes the row hash) allows the hash map to point us to AMP 4 which contains the base row.

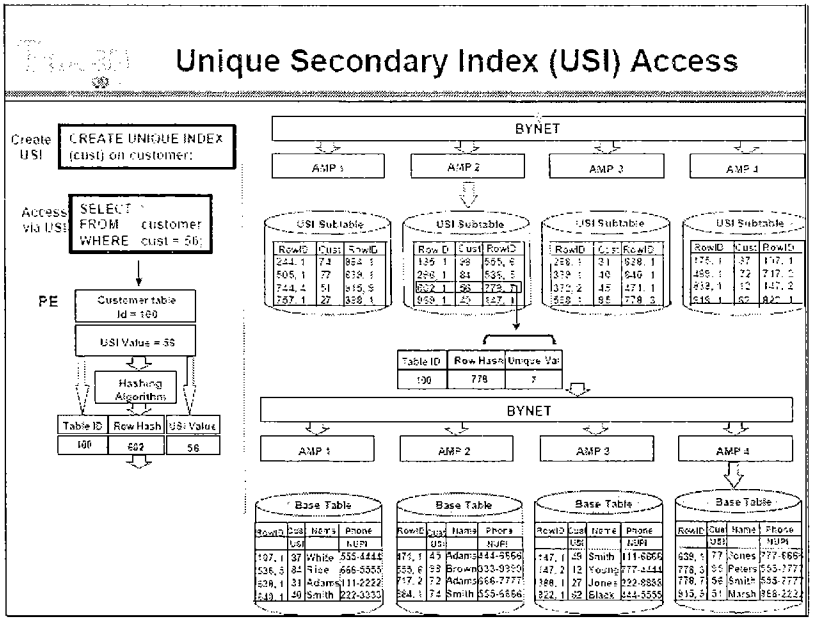
Note: It is possible that the base table row could be stored on the same AMP as the subtable, making this operation a one -AMP access.

Secondary index access uses the complete Row-ID to locate the row,unlike primary index access that uses only the row hash portion.

The Customer table below is used in the example on the facing page. It displays only a partial listing of the rows.

CUSTOMER table

Cust	Name	Phone
PK		
USI		NUPI
37	White	555-4444
98	Brown	333-9999
74	Smith	555-6666
95	Peters	555-7777
27	Jones	222-8888
56	Smith	555-7777
45	Adams	444-6666
84	Rice	666-5555
49	Smith	111-6666
51	Marsh	888-2222
31	Adams	111-2222
62	Black	444-5555
12	Young	777-4444
77	Jones	777-6666
72	Adams	666-7777
40	Smith	222-3333



Non-Unique Secondary Index (NUSI) Access

The facing page shows an all-AMP access necessary to retrieve a row via a **non-unique secondary index (NUSI)** access.

After the row hash of the secondary index value is calculated, the BYNET will automatically activate all AMPs per the Parsing Engine's instructions. Each AMP locates the subtable rows containing the qualifying value and row hash. These subtable rows contain the Row-ID(s) for the base rows, which are guaranteed to be on the same AMP as the subtable row. This reduces activity in the BYNET and essentially makes the query an AMPlocal operation.

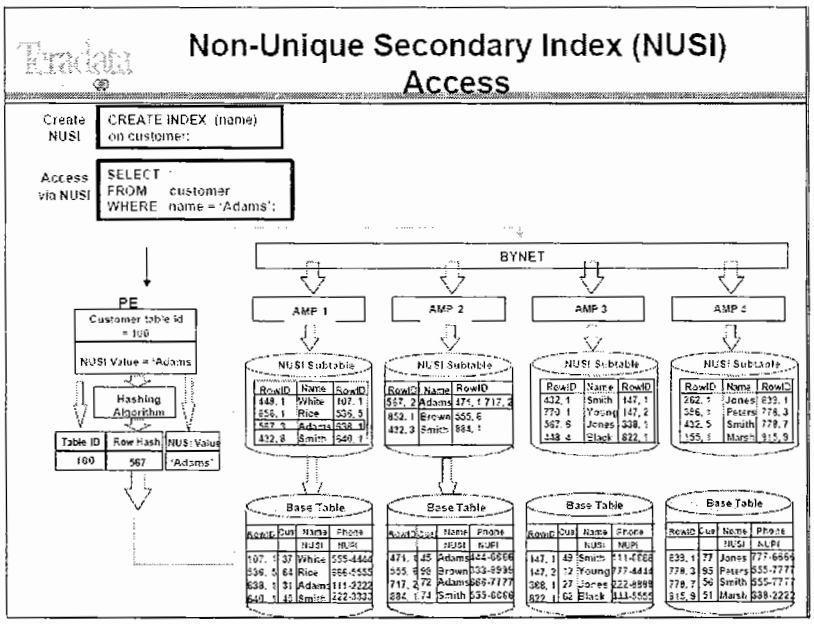
Because each AMP may have more than one qualifying row, it is possible for the subtable row to have more than one Row-ID for the base table rows.

In the Customer table example below, there is only a partial listing of the rows.

CUSTOMER table

Cust	Name	Phone
PK		
USI		**NUPI**
37	White	555-4444
98	Brown	333-9999
74	Smith	555-6666
95	Peters	555-7777
27	Jones	222-8888
56	Smith	555-7777
45	**Adams**	**444-6666**

84	Rice	666-5555
49	Smith	111-6666
51	Marsh	888-2222
31	**Adams**	**111-2222**
62	Black	444-5555
12	Young	777-4444
77	Jones	777-6666
72	**Adams**	**666-7777**
40	Smith	222-3333



Other Types of Secondary Indexes

Join Index

Join indexes have several uses:

- Define a pre-joined table on frequently joined columns (with optional aggregation) without demoralizing the database.
- Create a full or partial replication of a base table with a primary index on a foreign key column table to facilitate joins of very large tables by hashing their rows to the same AMP as the large table.
- Define a summary table without demoralizing the database.

You can define a join index on one or several tables. Single-table join index functionality is an extension of the original intent of join indexes, hence the confusing adjective "join" used to describe a single-table join index.

Sparse Index

Any join index, whether simple or aggregate, multi-table or single-table, can be sparse. A sparse join index uses a constant expression in the WHERE clause of its definition to narrowly filter its row population. This is known as a Sparse Join Index.

Hash Index

Hash indexes are used for the same purposes as single-table join indexes. Hash indexes create a full or partial replication of a base table with a primary index on a foreign key column table to facilitate joins of very large tables by hashing them to the same AMP.

You can define a hash index on one table only. Hash indexes are not indexes in the usual sense of the word. They are base tables that cannot be accessed directly by a query.

Value-Ordered NUSI

Value-ordered NUSIs are very efficient for range conditions and conditions with an inequality on the secondary index column set. Because the NUSI rows are sorted by data value, it is possible to search only a portion of the index subtable for a given range of key values. Thus the major advantage of a value-ordered NUSI is in the performance of range queries.

Value-ordered NUSIs have the following limitations.

- The sort key is limited to a single numeric column.
- The sort key column cannot exceed four bytes.
- They count as 2 indexes against the total of 32 non-primary indexes you can define on a base or join index table.

Comparison of Primary and Secondary Indexes

The table on the facing page compares **primary and secondary indexes**:

- Primary indexes are required; secondary indexes are optional. All tables must have a method of distributing rows among AMPs—the Primary Index.
- A table can have only one primary index, but up to 32 secondary indexes.
- Both primary and secondary indexes can have up to 64 columns. Secondary indexes, like primary indexes, can be either unique (USI) or non-unique (NUSI).
- The secondary index does not affect the distribution of rows. Rows are distributed according to the primary index values.
- Secondary indexes can be added and dropped dynamically as needed. In some cases, it is a good idea to wait and see how the database isused and then add secondary indexes to facilitate that usage.
- Both primary and secondary indexes affect system performance for different reasons. A poorly chosen PI results in skewed data distribution, which causes some AMPs to do more work than others, and slows the system.
- Secondary indexes affect performance because they require subtables.
- Both primary and secondary indexes allow rapid retrieval of specific rows.
- Both primary and secondary indexes can be created using multiple data types.
- Secondary indexes are stored in separate subtables; primary indexes are not.
- Because secondary indexes require separate subtables, extra processing overhead is needed to maintain those subtables.

Comparison of Primary and Secondary Indexes

Index Feature	Primary	Secondary
Required?	Yes	No
Number per Table	1	0-32
Max Number of Columns	64	64
Unique or Non-Unique?	Both	Both
Affects Row Distribution	Yes	No
Created/Dropped Dynamically	No	Yes
Improves Access	Yes	Yes
Separate Physical Structure	None	Subtable
Extra Processing Overhead	No	Yes

Full-Table Scans

A **full-table scan** is another way to access data without using any primary or secondary indexes.

In evaluating an SQL request, the Optimizer examines all possible access methods and chooses the one it believes to be the most efficient. The coding of the SQL request, along with the demographics of the table and the availability of indexes, all play a role in the decision of the Parser.

Some coding constructs, listed on the facing page, always cause a full table scan. A full-table scan typically occurs when the columns in an index are not referenced in the query, or a range of values is specified for the columns in a primary index. In other cases, a full-table scan might be chosen because it is the most efficient method. If the number of physical reads exceeds the number of data blocks, then the Optimizer may decide that a full-table scan is faster.

With a full-table scan, each data block is found using the Master and Cylinder Indexes and each data row is accessed only once.

As long as the choice of primary index has caused the table rows to distribute evenly across all of the AMPs, the parallel processing of the AMPs can do the full-table scan quickly. The file system stores each table on as few cylinders as possible to help reduce the cost of full table scans.

While full-table scans are impractical and even disallowed on some systems, Teradata routinely permits ad hoc queries with full-table scans.

Full-Table Scans

CUSTOMER

Cust_ID	Cust_Name	Cust_Phone
USI	NUSI	NUPI

- Every data block of the table must be read.
- All AMPs scan their portion of the table in parallel.
- Primary Index choice affects FTS performance.
- Full-table scans typically occur when either:
  - The index columns are not used in the query
  - An index is used in a non-equality test
  - A range of values is specified for the primary index

Examples of Full-Table Scans:

```
SELECT  *  FROM customer WHERE Cust_Phone LIKE '524-_ _ _ _';

SELECT  * FROM customer WHERE Cust_Name <> 'Davis';

SELECT  * FROM customer WHERE Cust_ID > 1000;
```

PPI Access

Partition Elimination is used to **access tables with a PPI**. The term "partition elimination" refers to an automatic optimization in which the optimizer determines, based on query conditions, that some partitions can't contain qualifying rows, and causes those partitions to be skipped. Partitions that are skipped for a particular q uery are called excluded partitions. Generally, the greatest benefit of a PPI table is obtained from partition elimination.

The facing page identifies the issues associated with accessing a table that has a defined PPI **and all of the primary index columns have values specified.**

- If the SELECT statement does not provide values for any of the partitioning columns, then all of the partitions may be probed to find row(s) with the hash value.
- If the SELECT statement provides values for some of the partitioning columns, then partition elimination may reduce the number of the partitions that will be probed to find row(s) with the hash value.

A common situation is with SQL specifying a range of values for partitioning columns. This allows some partitions to be excluded.

- If the SELECT statement provides values for all of the partitioning columns, then partition elimination will cause a single partition to be probed to find row(s) with the hash value.

If the partitioning expression references any columns that aren't part of the primary index, then PI access may be slower. When all partitioning expression columns are contained in the PI, then PI access is unchanged.

The worst case is when a query specifies the PI column values, but doesn't mention the partitioning column(s). In this situation, each partition must be probed for the appropriate PI value. In the worst case, the number of disk reads could increase by a factor equal to the number of partitions. While probing a partition is a fast operation, a table with thousands of partitions might not provide acceptable performance for PI accesses for some applications.

Data Protection

Locks

Locking prevents multiple users who are trying to change the same data at the same time from violating data integrity. This concurrency control is implemented by **locking** the desired data. Locks are automatically acquired during the processing of a request and released at the termination of the request. In addition, users can specify locks. There are four types of locks:

Exclusive Locks

Exclusive locks are applied only to databases or tables, never to rows. They are the most restrictive types of lock; all other users are locked out. Exclusive locks are typically used (only when structural changes are being made to the database or table), as in the case of a DDL statement.

Write Locks

Write locks enable users to modify data while locking out all other users except readers not concerned about data consistency (access lock readers). Until a write lock is released, no new read or write locks are allowed. When you update a table without a WHERE clause, the system places a write lock on the table.

Read Locks

Read locks are used to ensure consistency during read operations. Several users may hold concurrent read locks on the same data, during which no modification of the data is Permitted.

Access Locks

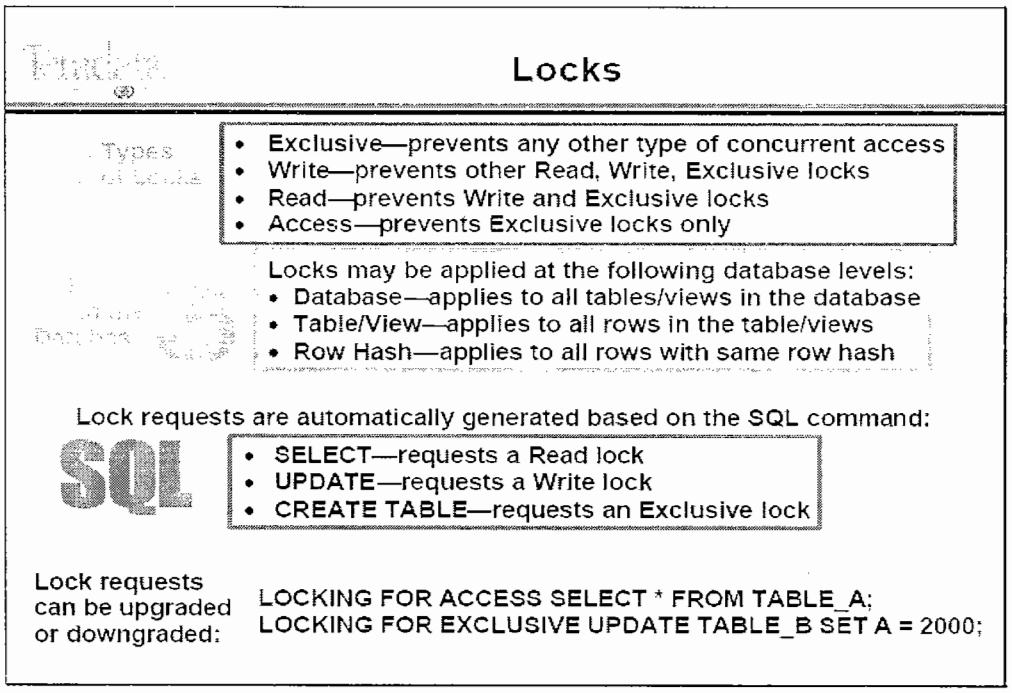
Users who are not concerned about data consistency can specify access locks. Using an access lock allows for reading data while modifications are in process. Access locks are designed for decision support on large tables that are updated only by small, single-row changes. Access locks are sometimes called stale read locks. You may get stale data that hasn't been updated.

Lock Levels

Three levels of database locking are provided:

Database	Locks all objects in the database.
Table	Locks all rows in the table or view.
Row Hash	Locks all rows with the same row hash (primary and Fallback rows, and Secondary Index subtable rows.

The type and level of locks are automatically chosen based on the type of SQL command issued as shown on the facing page. The user has, in some cases, the ability to upgrade or downgrade the lock.



Transient Journal

The **transient journal** permits the successful rollback of a failed transaction (TXN). Transactions are not committed to the database until the AMPs have received an End Transaction request, either implicitly or explicitly. There is always the possibility that the transaction may fail. If so, data is returned to its original state after transaction failure.

The transient journal maintains a copy on each AMP of before images of all rows affected by the transaction. In the event of transaction failure, the before images are reapplied to the affected tables, then deleted from the journal, and a rollback operation is completed. In the event of transaction success, the before images for the transaction are discarded from the journal at the point of transaction commit.

Transient Journal activities are automatic and transparent to the user.

Transient Journal

Transient Journal

- Consists of a journal of transaction before images.
- Provides rollback in the event of TXN failure.
- Is automatic and transparent.
- Before images are reapplied to table if TXN fails.
- Before images are discarded upon TXN completion.

Successful TXN

BEGIN TRANSACTION

UPDATE Row A — Before image Row A recorded

(Add $100 to checking)

UPDATE Row B — Before image Row B recorded

(Subtract $100 from savings)
END TRANSACTION — Discard before images
Failed TXN
BEGIN TRANSACTION
UPDATE Row A — Before image Row A recorded
UPDATE Row B — Before image Row B recorded
(Failure occurs)
(Rollback occurs) — Reapply before images
(Terminate TXN) — Discard before images

RAID Protection

There are many forms of disk array protection. Teradata supports the following protection schemes:

- RAID 1
- RAID 5
- RAID S

RAID 1 is a disk-mirroring technique. Each physical disk is mirrored elsewhere in the array. This requires the array controllers to write all data to two separate locations, which means data can be read from two locations as well. In the event of a disk failure, the mirror disk becomes the primary disk to the array controller and performance is unchanged.

RAID 5 is a parity-checking technique. For every three blocks of data (spread over three disks), there is a fourth block on a fourth disk that contains parity information. This allows any one of the four blocks to be reconstructed by using the information on the other three. If two of the disks fail, the rank becomes unavailable. A rank is defined as a group of disks that have the same SCSI ID (the disks in the rank are not on the same channel, but are on different channels). The array controller does the recalculation of the information for the missing block. Recalculation will have some impact on performance, but at a much lower cost in terms of disk space.

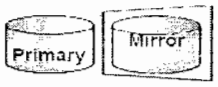
RAID S is a parity-checking technique similar to RAID 5 and is used with EMC disk arrays.

Note: In Teradata, RAID 1 and RAID 5 are commonly used. Unless there is a reason not to do so, NCR recommends that all our customers implement RAID 1.

RAID Protection

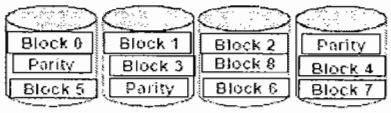
RAID 1 (Mirroring)

- Each physical disk in the array has an exact copy in the same array.
- The array controller can read from either disk and write to both.
- When one disk of the pair fails. there is no change in performance.
- Mirroring reduces available disk space by 50%.
- Array controller reconstructs failed disks quickly.



RAID 5 (Parity)

- Data and parity are striped across a rank of 4 disks.
- If a disk fails, any missing block may be reconstructed using the other three disks.
- Parity reduces available disk space by 25% in a 4-disk rank.
- Array controller reconstruction of failed disks takes longer than RAID 1.



RAID-1 - Good performance with disk failures
Higher cost in terms of disk space

RAID-5 - Reduced performance with disk failures
Lower cost in terms of disk space

Fallback

Fallback protects your data by storing a second copy of each row of a table on an alternate, Fallback AMP in the same cluster. If an AMP fails, the system accesses the Fallback rows to meet requests. Fallback provides AMP fault tolerance at the table level. With Fallback tables, if one AMP fails, all table data is still available. Users may continue to use Fallback tables without any loss of available data.

During table creation or after a table is created, you may specify whether or not the system should keep a Fallback copy of the table. If Fallback is specified, it is automatic and transparent.

Fallback guarantees that the two copies of a row will always be on different AMPs. If either AMP fails, the alternate row copy is still available on the other AMP.

There is a benefit to protecting your data, but there are costs associated with that benefit. With Fallback use, you need twice the disk space for storage and twice the I/O for INSERTs, UPDATEs, and DELETEs. (The Fallback option does not require any extra I/O for SELECT operations and the Fallback I/O will be performed in parallel with the primary I/O.)

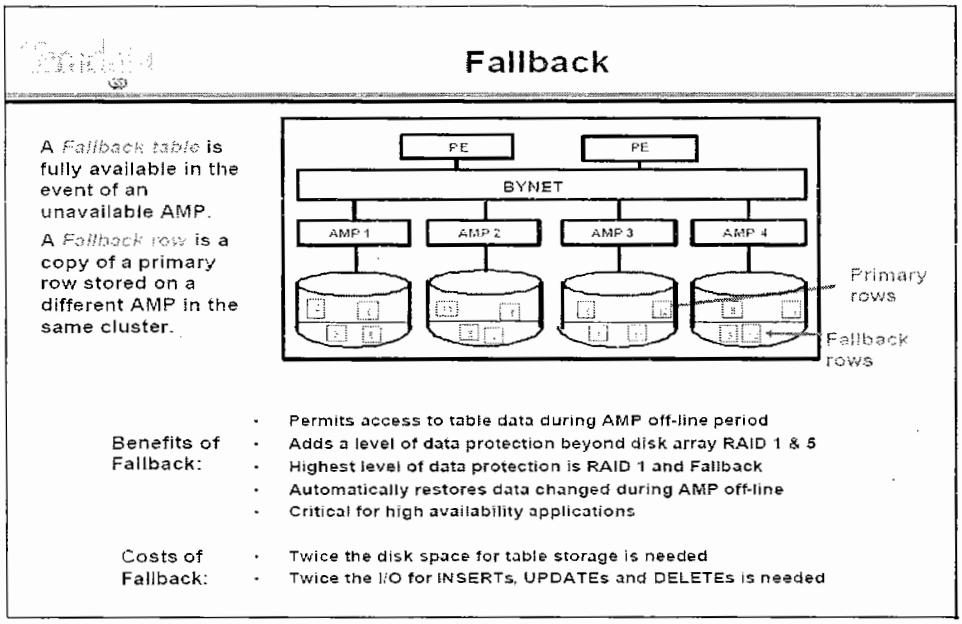
The benefits of Fallback include:

- Protects your data from hardware (disk) failure.
- Protects your data from software (node) failure.
- Automatically recovers with minimum recovery time, after repairs or fixes are complete.

A hardware (disk, cpu) or software (vproc) failure causes an AMP to be taken off-line until the problem is corrected.

During this period, fallback tables are fully available to users.

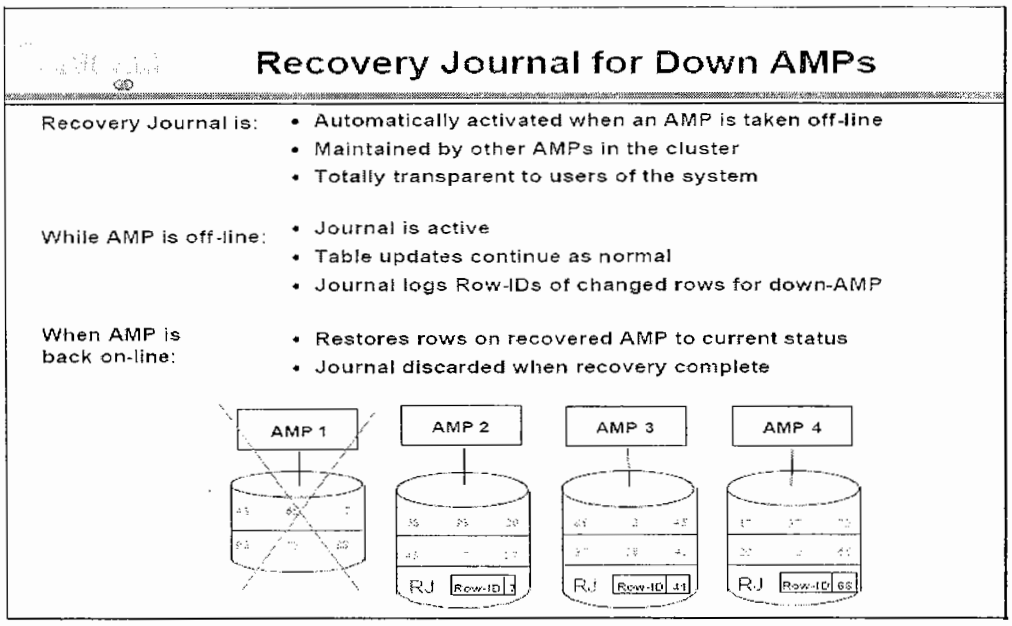
When the AMP is brought back on-line, the associated vdisk is refreshed to reflect any changes made during the off-line period.



Recovery Journal for Down AMPs

After the loss of any AMP, a **down-AMP recovery journal** is started automatically to log any changes to rows which reside on the down AMP.Any inserts, updates, or deletes affecting rows on the down AMP are applied to the Fallback copy within the cluster. The AMP that holds the Fallback copy logs the Row-ID in its recovery journal.

This process will continue until the down AMP is brought back on-line. As part of the restart activity, the recovery journal is read and changed rows are applied to the recovered AMP. When the journal has been exhausted, it is discarded and the AMP is brought on-line fully recovered.

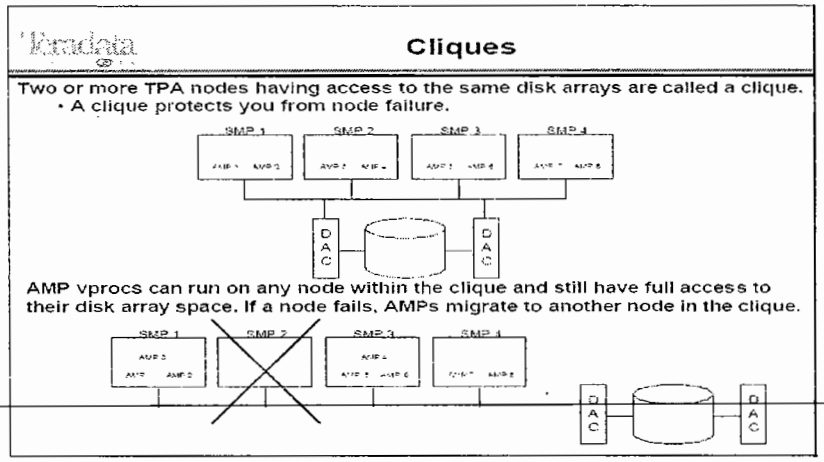


Cliques

In a clique, multiple nodes are accessible by any node in the same clique. The facing page shows a situation where a node fails. When the node fails, PDE resets and Teradata restarts and the AMP vprocs migrate to the other nodes in the clique.

A Massively Parallel Processing (MPP) system will continue to run if a node or disk fails in a clique, even if you do not select the Fallback option. In the situation shown, the TDP will restart the lost AMP vprocs on a different node within the clique. When the node is brought back into service, the TDP will move the AMPs back to their original node. For 7x24 systems, the Fallback option is recommended for minimizing the risks of system downtime.

Applications that recognize restarts and are coded to, will continue when the system comes back up. Otherwise, applications will need to be restarted.



Using Permanent Journals for Recovery

When recovery is initiated using the permanent journal, the question must be posed, recovery to what point in time? Assuming both before and after images have been captured, movement both forward and backward in time is possible.

A full backup was taken on Monday, and the change images in the permanent journal have been archived on Tuesday and Wednesday. When disaster hits on Thursday morning, a decision is made to restore the database up to the point of the disaster. The restore of Monday's dump is followed by a rollforward of the after images from Tuesday and Wednesday. Thursday's images remain in the journal and are restored directly without the need for external media.

If the failure on Thursday morning had been due to a corrupting application program, a recovery to Wednesday evening might have been chosen. In this case, a rollback operation is initiated, applying all before images captured since Wednesday night.

Permanent Journals

An optional, user-specified, system-maintained journal used for database recovery to a specified point in time.

- Used for recovery from unexpected hardware or software disasters.
- May be specified for:
  - One or more tables
  - One or more databases
- Permits capture of before images for database rollback.
- Permits capture of after images for database roll forward.
- Permits archiving change images during table maintenance.
- Reduces need for full-table backups.
- Provides a means of recovering NO FALLBACK tables.
- Requires additional disk space for change images.
- Requires user intervention for archive and recovery activity.

DATABASE CONCEPTS
(Terdata SQL)

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

What is an RDBMS?

Data is organized into tables in a relational database management system (RDBMS). Rows in the table represent instances of an entity, in this case an employee or a department. Columns represent the data fields which comprise the rows. Relations between tables occur when a column in one table also appears as a column in another.

EMPLOYEE (Partial Listing)

EMPLOYEE NUMBER	MANAGER EMPLOYEE NUMBER	DEPARTMENT NUMBER	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SALARY AMOUNT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1003	401	412101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

DEPARTMENT (Partial Listing)

DEPARTMENT NUMBER	DEPARTMENT NAME	BUDGET AMOUNT	MANAGER EMPLOYEE NUMBER
PK			FK
501	marketing sales	80050000	1017
301	research and development	46560000	1019
302	product planning	22600000	1016
403	education	93200000	1005
402	software support	30800000	1011
401	customer support	98230000	1003
201	technical operations	29380000	1025

PK = PRIMARY KEY

FK = FOREIGN KEY

Can you answer the following questions using the tables above?
What is the department name for employee 1004?
Answer: customer support
Who is the manager of employee 1004?
Answer: James Trader

Difference Between DBMS & RDBMS:

As mentioned before, DBMS provides a systematic and organized way of storing, managing and retrieving from a collection of logically related information. RDBMS also provides what DBMS provides, but above that, it provides relationship integrity. So in short, we can say:

RDBMS = DBMS + REFERENTIAL INTEGRITY

For example every person should have an Address. This is a referential integrity between Name and Address. If we break this referential integrity in DBMS and files, it will not complain, but RDBMS will not allow you to save this data if you have defined the relation integrity between person and addresses. These relations are defined by using "Foreign Keys" in any RDBMS.

DBMS	RDBMS
Database Management System	DBMS with referential integrity between the tables
No normalization technique	Normalization technique adopted
Inclusion of flat file data in its system	Non-acceptance of flat file design
Used for simple business applications	Huge database applications
Foreign key support	Relationship established through foreign key only
Supported Languages: Programming Language, Data Manipulation Language, Data Definition Language, Schema Description Language, Sub-Schema Description Language	Only SQL (Sequential Query Language) supported
Structured into three major categories: network, hierarchical and relational	Follows relational model only
It requires low software & Hardware requirements	It requires high Software & Hardware requirements
It supports single User	It supports Multiple Users
No relation between tables	Relation between the tables can be established

Primary Key Rules

Rules governing how Primary Keys must be defined and how they function are:
Rule 1: A Primary Key is required.
Rule 2: A Primary Key value must be unique.
Rule 3: The Primary Key value cannot be NULL.
Rule 4: The Primary Key value should not be changed.
Rule 5: The Primary Key column should not be changed.
Rule 6: A Primary Key may be any number of columns.

Foreign Key Rules

Rules governing how Foreign Keys must be defined and how they operate are:
Rule 1: Foreign Keys are optional.
Rule 2: A Foreign Key value may be non-unique.
Rule 3: The Foreign Key value may be NULL.
Rule 4: The Foreign Key value may be changed.
Rule 5: A Foreign Key may be any number of columns.
Rule 6: Each Foreign Key must exist as a Primary Key in a related table.

Primary Key vs. Primary Index:

Primary Key (PK) - is defined as one or more columns used to uniquely identify each row in a table. PKs are used in conjunction with foreign keys to define the important column relationships in a database. PKs are always unique and cannot be null. PKs are not known to the Teradata RDBMS as such. **The Teradata RDBMS implements a primary key as a unique index**.
Primary Index - is defined as one or more columns used to distribute and locate rows in a table. Choice of primary index will affect distribution, access and performance. Oftentimes, but not always, the Primary Index and Primary Key are the same. Indexes (primary or secondary) may be used to enforce uniqueness (as in a PK) or to improve access. They may be unique or non-unique.

Indexes may be:

	Primary	Secondary
UNIQUE	UPI	USI
NON-UNIQUE	NUPI	NUSI

Every table must have exactly one primary index.

Examples to create Tables:

```
CREATE TABLE Employee
        (
         Employee_Number INTEGER NOT NULL,
         Manager_Employee_number INTEGER,
         Department_number INTEGER,
         Job_Code INTEGER,
         Last_name VARCHAR(20) ,
         First_Name VARCHAR(20) ,
         Hire_date Date,
         Birth_Date Date,
         Salary_Amount INTEGER
)
        UNIQUE PRIMARY INDEX Employee_UPI ( Employee_Number)

insert into Employee  Values(1006,1019,301,312101,'Stein','John',761015,531015,2945000);
insert into Employee  alues(1008,1019,301,312102,'Kanieski','Carol',770201,580517,2925000);
insert into Employee  Values(1005,0801,403,431100,'Ryan','Loretta',761015,550910,3120000);
insert into Employee  Values(1004,1003,401,412101,'Johnson','Darlene',761015,460423,3630000);
insert into Employee  Values(1007,1005,403,432101,'Villegas','Amando',770102,370131,4970000);
insert into Employee  Values(1003,0801,401,411100,'Trader','James',760731,470619,3785000)
```

```
CREATE TABLE Department
        (
         Department_number INTEGER NOT NULL,
         Department_name varchar(20),
         Budget_Amount Integer,
         Manager_Employee_Number Integer
)
        UNIQUE PRIMARY INDEX Department_UPI ( Department_number)

        insert into Department  Values(501,'marketing sales',80050000,1017);
        insert into Department  Values(301,'research and development',46560000,1019);
        insert into Department  Values(302,'product planning',22600000,1016);
        insert into Department  Values(403,'education',93200000,1005);
        insert into Department  Values(402,'software support',30800000,1011);
        insert into Department  Values(401,'customer support',98230000,1003);
        insert into Department  Values(201,'technical operations',29380000,1025)
```

What is SQL?

Structured Query Language (SQL) is the industry standard language for communicating with Relational Database Management Systems.

Structured Query Language is used to define the answer set that is returned from the RDBMS.

SQL is a non-procedural language, meaning it contains no procedural-type statements such as those listed here:

- GO TO
- PERFORM
- DO LOOP
- OPEN FILE
- CLOSE FILE
- END OF FILE

SQL Commands

Data Definition Language (DDL) Examples

SQL statement	Function
CREATE	Define a table, view, macro, index, trigger or stored procedure.
DROP	Remove a table, view, macro, index, trigger or stored procedure.
ALTER	Change table structure or protection definition.

Data Manipulation Language (DML)

SQL statement	Function
SELECT	Select data from one or more tables.
INSERT	Place a new row into a table.
UPDATE	Change data values in one or more existing rows.
DELETE	Remove one or more rows from a table.

Data Control Language (DCL)

SQL statement	Function
GRANT	Give user privileges.
REVOKE	Remove user privileges.
GIVE	Transfer database ownership.

DDL COMMANDS:

CREATE TABLE Elements

When executed, the CREATE TABLE statement creates and stores the table structure definition in the Teradata Data Dictionary.

The CREATE TABLE statement includes:
· Create Table options
· Column definitions
· Table-level constraints
· Index definitions

CREATE <SET/MULTISET> TABLE employee
<Create Table Options>
<Column Definitions>
<Table-level Constraints>
<Index Definitions>;

Create Table Options:
Fallback,Journlaing,Freespace,Datablock Size

- Duplicate row options:
  – SET—no duplicate rows allowed
  – MULTISET—duplicate rows allowed
- Table protection options:
  – FALLBACK or NO FALLBACK PROTECTION
  – BEFORE JOURNAL (NO, single or DUAL)
  – AFTER JOURNAL (NO, single (LOCAL or NOT LOCAL) or DUAL)
  – WITH JOURNAL TABLE (Table Name)
- Space Management options
  – FREESPACE—percentage(%) of cylinder freespace

– DATABLOCKSIZE—maximum block size for data blocks

Column Definitions

- Column name:
  – Name the column
- Data type:
  – Declare the column to be a character, byte, numeric, or graphic data type.
- Data type Attributes:
  – Specify DEFAULT, FORMAT, TITLE, NULL,CASESPECIFIC, UPPERCASE.
- Column Storage Attributes:
  – Compress NULL values or a specified value.
- Column-level Constraint Attributes:
  – Specify the single column as a primary key or foreign key.
  – Specify the single column as unique (must be NOT NULL).
  – Specify constraint conditions on the column.

Column-level Constraints

CONSTRAINT name—optional
PRIMARY KEY - No Nulls, No Dups
UNIQUE - No Nulls, No Dups
CHECK - Verify values or range
REFERENCES - Relates to other columns (foreignkey)

COLUMN Storage Attributes

Teradata Extensions
The COMPRESS phrase allows values in one or more columns of a permanent table to be compressed to zero space, thus reducing the physical storage space required for a table.
The COMPRESS phrase has three variations:

Variation:	What happens:
COMPRESS	Nulls are compressed.
COMPRESS NULL	Nulls are compressed.
COMPRESS <constant>	Nulls and the specified <constant> value are compressed.

NOTE: COMPRESS & COMPRESS NULL mean the same thing.

Table-level Constraints

- An alternate way to set column constraints
- Useful for multi-column constraints
- Uniqueness Constraint:
  [CONSTRAINT name] PRIMARY KEY (col_list)

[CONSTRAINT name] UNIQUE (col_list)

- Check Constraint:
  [CONSTRAINT name] CHECK boolean_condition)
- References Constraints
  [CONSTRAINT name] FOREIGN KEY (col_list)
  REFERENCES tablename [(col_list)]

DROP TABLE

To remove all data associated with a table, as well as the table structure definition from the Data Dictionary, use the DROP TABLE statement.

Example

Drop the employee data table created in the previous example.

```
DROP TABLE emp_data;
```

- Deletes all data in emp_data.
- Removes the emp_data definition from the Data Dictionary. You must recreate the table if you wish to use it again.
- Removes all explicit access rights on the table.

ALTER TABLE

Once a table has been created, certain characteristics are not alterable, such as the Primary Index choice. To change them you must CREATE a new table which includes the new characteristics, then populate that table.

Other characteristics are alterable. You can use the ALTER TABLE statement to modify these characteristics.

ALTER TABLE

1. ADDs and/or DROPs columns from an empty or populated table:

```
ALTER TABLE emp_data
  ADD educ_level CHAR(1), ADD insure_type SMALLINT
;
ALTER TABLE emp_data
  DROP educ_level, DROP insure_type
;
```

2. Changes the attribute options on existing columns:

```
ALTER TABLE emp_data
  ADD birthdate FORMAT 'mmmBdd,Byyy"
;
```

NOTE: In this example, the birthday column already exists. We are adding FORMAT to the birthday column

Data Manipulation

Data Manipulation consists of four commands:

- INSERT
  -Add a row to a table.
- INSERT SELECT
  -Add rows to a table from another table.
- UPDATE
  -Change column values in existing rows of a table.
- DELETE
  -Remove rows from a table.

INSERT

INSERT allows you to add a new row to a table.

Example

There are two types of insert:

1) Insert a new employee into the employee table:
   INSERT INTO employee
   VALUES (1210, NULL, 401, 412101, 'Smith', 'James', 890303, 460421, 41000);
2) Insert a new employee with only partial data:
   INSERT INTO employee
   (last_name, first_name, hire_date, birthday, salary_amount, employee_number)
   VALUES ('Garcia', 'Maria', 861027, 541110,
   76500.00, 1291);
3) Use INSERT SELECT to copy rows from one table to another.
   The syntax is as follows:

   INSERT INTO target_table SELECT * FROM source_table;

   The SELECT portion of the statement may be used to define a subset of rows and/or a subset of columns to be inserted to the target table.

UPDATE

UPDATE allows you to modify one or many columns of one or many rows in a single table.
The WHERE condition can include:

- Columns from the table being updated.
- Joins with columns from other tables.
- Subqueries.

Problem

Change employee 1010's department to 403, job code to 432101, and manager to 1005 in the employee table

```
UPDATE employee
SET department_number     =   403
  ,job_code               =   432101
```

```
   ,manager_employee_number    =    1005
WHERE employee_number          =    1010
;
```

UPDATE Using Subqueries or Joins

Problem

Update the employee table to give everyone in all support departments a 10% raise. Department numbers for all of the support departments are not known.

```
UPDATE      employee
SET         salary_amount=salary_amount * 1.10
WHERE       department_number IN
        (SELECT department_number
        FROM    department
        WHERE   department_name LIKE '%Support%')
;
```

Using Join:

```
UPDATE employee
SET     salary_amount=salary_amount * 1.10
WHERE   employee.department_number =
        department.department_number
AND     department_name LIKE '%Support%'
;
```

DELETE

DELETE allows you to delete rows from a single table. If no WHERE clause is specified, then all rows are deleted.

The WHERE condition can reference:

- Column values in the target table.
- Column values based on a subquery against another table.
- Column values based on a join with another table.

SELECT

The **SELECT** statement allows you to retrieve data from one or more tables.

SELECT * FROM employee;

This query would return all columns and all rows from the employee table.

The asterisk, "*", indicates that we wish to see all of the columns in the table.

Instead of using the asterisk symbol to specify all columns, we could name specific columns separated by commas:

SELECT employee_number

```
        ,hire_date
        ,last_name
        ,first_name
FROM    employee
WHERE   department_number = 401;
```

ORDER BY Clause

The ORDER BY clause specifies the column(s) to be used for sorting the result.

```
SELECT employee_number
        ,last_name
        ,first_name
        ,hire_date
        FROM employee
        WHERE department_number = 401
        ORDER BY hire_date
```

Sort Direction

In the example above, results will be returned in ascending order by hire date. Ascending order is the default sort sequence for an **ORDER BY** clause. To explicitly specify ascending or descending order, add **ASC** or **DESC**, to the end of the **ORDER BY** clause. The following is an example of a sort using descending sequence.

```
ORDER BY    hire_date DESC;   (descending sort)
```

Naming the Sort Column

You may indicate the sort column by naming it directly (e.g., hire_date) or by specifying its position within the **SELECT** statement. Since hire_date is the fourth column in the **SELECT** statement, the following **ORDER BY** clause is equivalent to saying **ORDER BY hire_date.**.

```
ORDER BY  4;
```

An **ORDER BY** clause may specify multiple columns. The order in which columns are listed in the ORDER BY clause is significant.

NOTE: Each column specified in the ORDER BY clause can have its own sort order, either ascending or descending.

```
SELECT      employee_number
            ,department_number
            ,job_code
FROM        employee
WHERE       department_number < 302
ORDER BY    department_number ASC
            ,job_code DESC;
```

DISTINCT

The **DISTINCT** operator will consolidate duplicate output rows to a single occurrence.

Example Without DISTINCT

```
SELECT  department_number
        ,job_code
FROM    employee
WHERE   department_number =
        501;
```

department_number	job_code
501	512101
501	512101
501	511100

NOTE: Two people in department 501 have the same job code (512101). If our purpose is simply to find out which job codes exist in department 501, we could use **DISTINCT** to avoid seeing duplicate rows.

Example With DISTINCT

```
SELECT      DISTINCT department_number
            ,job_code
FROM        employee
WHERE       department_number = 501;
```

department_number	job_code
501	511100
501	512101

Naming Database Objects

How database objects can be named are summarized in the following table:

Names are composed of:	a-z
	A-Z
	0-9
	_ (underscore)
	$
	#
Names must start with:	a-z
	A-Z
	_(underscore)
	#
	$

Naming Syntax
The syntax for fully qualifying a column name is:
databasename.tablename.columnname

Example:

NAME	(unqualified)
EMPLOYEE.NAME	(partially qualified)
PAYROLL.EMPLOYEE.NAME	(fully qualified)

Teradata SQL Extensions:

Several commands in our software are Teradata-specific. (ANSI standard SQL is covered in later sections of this course.) Here is a list of Teradata extensions covered within this course

ADD_MONTHS	BEGIN/ END TRANSACTION
COLLECT/ DROP STATISTICS	COMMENT ON
CONCATENATION	EXPLAIN
FALLBACK	FORMAT
HELP	INDEX
LOCKING	MACRO Facility
	• CREATE
	• REPLACE
	• DROP
	• EXECUTE
NAMED	NULLIFZERO/ZEROIFNULL
SHOW	SUBSTR
TITLE	TRIM
WITH	WITH . . . BY

HELP Commands: Database objects
The HELP Command is used to display information about database objects such as (but not limited to):

- Databases and Users
- Tables
- Views
- Macros

HELP retrieves information about these objects from the Data Dictionary. Below are the syntactical options for various forms of the HELP command:

HELP Command

HELP DATABASE	databasename;
HELP USER	username;
HELP TABLE	tablename;
HELP VIEW	viewname;
HELP MACRO	macroname;
HELP COLUMN	table or viewname.*; (all columns)
HELP COLUMN	table or viewname.colname . . ., colname;

The SHOW Command

The **SHOW** command displays the current Data Definition Language (DDL) of a database object (e.g., Table, View, Macro, Trigger, Join Index or Stored Procedure). The SHOW command is used primarily to see how an object was created.

Sample Show Commands

Command	Returns
SHOW TABLE tablename;	CREATE TABLE statement
SHOW VIEW viewname;	CREATE VIEW statement
SHOW MACRO macroname;	CREATE MACRO statement

The EXPLAIN Command

The EXPLAIN function looks at a SQL request and responds in English how the optimizer plans to execute it. It does not execute the statement and is a good way to see what database resources will be used in processing your request.

For instance, if you see that your request will force a full-table scan on a very large table or cause a Cartesian Product Join, you may decide to re-write a request so that it executes more efficiently.

EXPLAIN provides a wealth of information, including the following:

1.) Which indexes if any will be used in the query.
2.) Whether individual steps within the query may execute concurrently (i.e. parallel steps).
3.) An estimate of the number of rows which will be processed.
4.) An estimate of the cost of the query (in time increments).

EXPLAIN SELECT * FROM department;

Logical Operators

Types of operators in Logical Expressions:

Type of operator	Symbol	Meaning
Comparison operators	=	Equal
	<>	Not equal
	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less than or equal to
	BETWEEN <a> AND <b>	Inclusive range
[NOT] IN		<expression> is in a list or <expression> is not in a list
IS [NOT] NULL		<expression> is null or <expression> is not null
[NOT] EXISTS		Table contains at least 1 row or Table contains no rows
LIKE		Partial string operator

BETWEEN -- Numeric Range Testing

To locate rows for which a numeric column is within a range of values, use the BETWEEN <a> AND <b> operator. Specify the upper and lower range of values that qualify the row.

The BETWEEN operator looks for values between the given lower limit <a> and given upper limit <b> as well as any values that equal either <a> or <b> (BETWEEN is inclusive.)

Example

Select the name and the employee's manager number for all employees whose job codes are in the 430000 range.

```
SELECT    first_name
          ,last_name
          ,manager_employee_number
FROM      employee
WHERE     job_code BETWEEN 430000 AND 439999;
```

An alternative syntax is shown below:

```
SELECT    first_name
          ,last_name
          ,manager_employee_number
```

```
FROM        employee
WHERE       job_code >= 430000
AND         job_code <= 439999;
```

BETWEEN -- Character Range Testing

Use the BETWEEN <a> AND <b> operator to locate rows for which a character column is within a range of values. Specify the upper and lower range of values that qualify the row. BETWEEN will select those values which are greater than or equal to <a> and less or equal to <b>. (BETWEEN is inclusive.)

```
SELECT      last_name
FROM        employee
WHERE       last_name BETWEEN 'r' AND 's';
```

Set Operator IN

Use the IN operator as shorthand for when multiple values are to be tested. Select the name and department for all employees in either department 401 or 403. This query may also be written using the OR operator which we shall see shortly.

```
SELECT      first_name
            ,last_name
            ,department_number
FROM        employee
WHERE       department_number IN (401, 403);
```

Set Operator NOT IN

Use the NOT IN operator to locate rows for which a column does not match any of a set of values. Specify the set of values which disqualifies the row.

```
SELECT      first_name
            ,last_name
            ,department_number
FROM        employee
WHERE       department_number NOT IN (401, 403) ;
```

NOT IN vs. OR

NOT IN provides a shorthand version of a 'negative OR' request. The following is an example using the OR operator for the query example given above.
Select the name and the department for all employees who are NOT members of departments 401 and 403.

```
SELECT      first_name
            ,last_name
            ,department_number
FROM        employee
```

```
WHERE NOT     (department_number=401
              OR department_number=403);
```

NULL

NULL is used to show the absence of a data value. It is not the same as having a value of zero or spaces.
NULLS may be disallowed by defining a column with a NOT NULL attribute.
Null columns may be compressed to occupy zero row space. This is a space-saving feature.

Arithmetic and Comparison Operation on NULL

Col A	Operation	Col B	Results
10	+	NULL	NULL
10	-	NULL	NULL
10	*	NULL	NULL
10	/	NULL	NULL
10	>	NULL	UNKNOWN
10	<	NULL	UNKNOWN
10	>=	NULL	UNKNOWN
10	<=	NULL	UNKNOWN
10	=	NULL	UNKNOWN
10	<>	NULL	UNKNOWN
NULL	>	NULL	UNKNOWN
NULL	<	NULL	UNKNOWN
NULL	>=	NULL	UNKNOWN
NULL	<=	NULL	UNKNOWN
NULL	=	NULL	UNKNOWN
NULL	<>	NULL	UNKNOWN

Using NULL in a Select

Use NULL in a SELECT statement, to define that a range of values either IS NULL or IS NOT NULL.
To list employee numbers in this table with unknown extensions:

```
SELECT      employee_number
FROM        employee_phone
WHERE       extension IS NULL;
```

To list employee numbers of people with known extensions:

```
SELECT      employee_number
FROM        employee_phone
```

```
WHERE       extension IS NOT NULL;
```

LIKE Operator

The LIKE operator searches for patterns matching character data strings.

You must provide two parameters for the LIKE operator: a string expression to be searched and a string pattern for which to search.

The string can contain specific characters, as well as the following "wildcards":
% (indicates zero or more character positions)
_ (indicates a single character position)
Here are some examples using the LIKE operator:

String pattern example:	Meaning:
LIKE 'JO%'	begins with 'JO'
LIKE '%JO%'	contains 'JO' anywhere
LIKE '__HN'	contains 'HN' in 3rd and 4th position
LIKE '%H_'	contains 'H' in next to last position

Problem
To display the full name of employees whose last name contains the letter "R" followed by the letter "a".

Note: Default comparison is not case-specific.

Solution

```
SELECT      first_name
            ,last_name
FROM        employee
WHERE       last_name LIKE '%Ra%';
```

LIKE Operator -- Partial String

To display the full name of employees whose last name contains "Ra", use the CASESPECIFIC operator:

```
SELECT      first_name
            ,last_name
FROM        employee
WHERE       last_name (CASESPECIFIC) LIKE '%Ra%';
```

In the default, the comparison is not case-specific.
Use the Teradata extension (CASESPECIFIC) to force case-specific comparison.

Because we used the case-specific designator, we don't get James Trader in our answer set but only get answers that contain an uppercase "R". The name 'LaRaye' would have also appeared in this answer set. Use LIKE 'Ra%' to get only names that begin with "Ra".

LIKE Operator -- Using Quantifiers

To extend the pattern matching functions of the LIKE operator, use quantifiers.
There are three such quantifiers:

ANY — any single condition must be met (OR logic)
SOME — same as ANY
ALL — all conditions must be met (AND logic)

ANY and SOME are synonyms. Using LIKE ANY and LIKE SOME will give the same result.

Problem
To display the full name of all employees with both "E" and "S" in their last name.
Solution

```
SELECT      first_name
            ,last_name
FROM        employee
WHERE       last_name LIKE ALL ('%E%', '%S%');
```

Problem
To display the full name of all employees with either an "E" or "S" in their last name.
Solution

```
SELECT      first_name
            ,last_name
FROM        employee
WHERE       last_name LIKE ANY ('%E%', '%S%');
```

Logical Operator – AND

Logical operators AND, OR and NOT allow you to specify complex conditional expressions by combining one or more logical expressions.

Logical Operator AND

The logical operator AND combines two boolean expressions, both of which must be true in a given record for them to be included in the result set.

TRUE	AND	TRUE	TRUE
FALSE	AND	TRUE	FALSE
TRUE	AND	FALSE	FALSE
FALSE	AND	FALSE	FALSE

TRUE	OR	TRUE	TRUE
TRUE	OR	FALSE	TRUE

FALSE	OR	TRUE	TRUE
FALSE	OR	FALSE	FALSE

NOT	TRUE	FALSE
NOT	FALSE	TRUE

Problem

To display the name and employee number of employees in department 403 who earn less than $35,000 per year.

Solution

```
SELECT      first_name
            ,last_name
            ,employee_number
FROM        employee
WHERE       salary_amount < 35000.00
AND         department_number = 403 ;
```

Logical Operator -- OR

The logical operator OR combines two Boolean expressions. At least one must be true in a given record for it to be included in the result set.

Problem

To display the name and the employee number for employees who either earn less than $35,000 annually or work in department 403.

Solution

```
SELECT      first_name
            ,last_name
            ,employee_number
FROM        employee
WHERE       salary_amount < 35000.00
OR          department_number = 403;
```

Multiple AND . . . OR

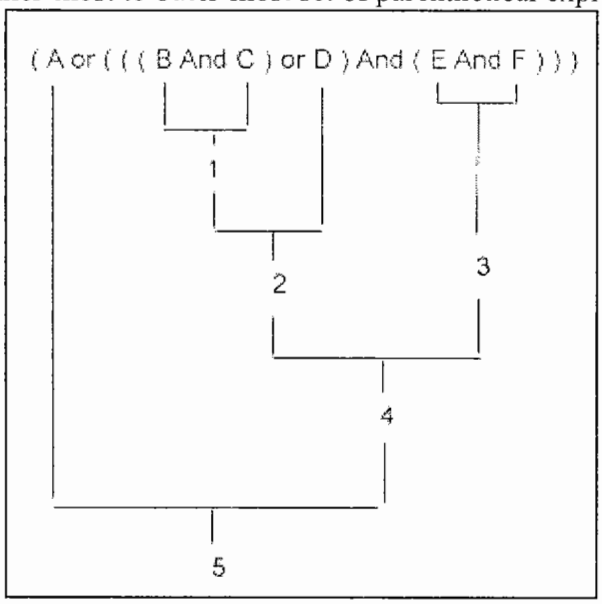
You want to find all employees in either department 401 or department 403 whose salaries are either under $35,000 or over $85,000.

```
SELECT      last_name
            ,salary_amount
            ,department_number
FROM        employee
WHERE       (salary_amount < 35000
OR          salary_amount > 85000)
AND         (department_number = 401
OR          department_number = 403) ;
```

Logical Operators – Combinations

Operator Procedures

Parentheses can be used to force an evaluation order. In the presence of parentheses, expressions are evaluated from the inner-most to outer-most set of parenthetical expressions.



Logical Operators -- Using Parentheses

Problem

Select the name, department number, and job code for employees in departments 401 or 403 who have job codes 412101 or 432101.

Solution

```
SELECT      last_name
            ,department_number
            ,job_code
FROM        employee
WHERE       (department_number = 401
OR          department_number = 403)
AND         (job_code = 412101
OR          job_code = 432101) ;
```

If we accidentally left the parentheses out of our statement. Would we get the same results?

```
SELECT      last_name
            ,department_number
            ,job_code
FROM        employee
WHERE       department_number = 401
OR          department_number = 403
AND         job_code = 412101
OR          job_code = 432101;
```

Logical NOT

Place the NOT operator in front of a conditional expression or in front of a comparison operator if you need to locate rows which do not meet specific criteria .

Problem
Select the name and employee number of employees NOT in department 301.

Solution for NOT Operator

```
SELECT      first_name
            ,last_name
            ,employee_number
FROM        employee
WHERE       department_number NOT = 301;
```

Solution for NOT Condition

```
SELECT      first_name
            ,last_name
            ,employee_number
FROM        employee
WHERE NOT   (department_number = 301);
```

SELECTing System Variables

SELECT can be used to query the system for current information, including current:

- System DATE.
- System TIME.
- Logged-on USER.
- Current default DATABASE.

Examples:

```
SELECT DATE;
SELECT TIME;
SELECT DATABASE;
SELECT USER;
```

Literal, Constant, and Calculator Features

Character Literals

You may add *character literals* to your SELECT statement:

```
SELECT  'Employee'
        ,last_name
        ,first_name
FROM    employee ;
```

Numeric Constants:

You may also add *numeric constants* to your SELECT statement:

```
SELECT  12345
```

```
        ,last_name
        ,first_name
FROM    employee ;
```

Calculator Mode:

You may use numeric expressions to do calculations:

Calculation	Result	Comment
SELECT 2*250;	500	Simple multiplication
SELECT 1.01 + 2.2;	3.21	Uses greatest possible precision
SELECT 10/3.000;	3.333	Rounds off
SELECT 10/6.000;	1.667	Rounds up

DATE Data Type

The Teradata DATE data type is used to store calendar dates representing year, month and day. It is stored internally as a four-byte integer.

The Teradata database performs basic DATE handling functions including:

- Month-to-month conversions.
- Year-to-year conversions.
- Leap-year conversions.
- Dates prior to 1900.
- Dates after 2000.

DATE Arithmetic

Find	Syntax
The date 30 days from today	SELECT DATE + 30;
The date 1 year from now	SELECT DATE + 365;

How would you find a person's age (in rounded years)?

```
SELECT      (DATE-birthdate)/365
FROM        employee
WHERE       last_name = 'Stein';
```

Problem

Find employees with more than 10 years of service.

```
SELECT      last_name
            ,hire_date
FROM        employee
WHERE       (DATE-hire_date)/365 > 10;
```

Date Function ADD_MONTHS

The ADD_MONTHS function allows the addition of a specified number of months to an existing date, resulting in a new date.
The format of the function is:
ADD_MONTHS (date, n)
where n is a number or an expression representing the number of months to be added to the date.
The following examples demonstrate its usage.

Query	Results
SELECT DATE; /* March 20, 2001 */	01/03/20
SELECT ADD_MONTHS (DATE, 2)	2001-05-20
SELECT ADD_MONTHS (DATE, 12*14)	2015-03-20
SELECT ADD_MONTHS (DATE, -3)	2000-12-20

Note: The results of the ADD_MONTH function are always displayed in YYYY-MM-DD format.
ADD_MONTHS may also be applied to a literal date, which must be specified in the YYYY-MM-DD format.

Query	Results
SELECT ADD_MONTHS ('1996-07-31', 2)	1996-09-30
SELECT ADD_MONTHS ('1995-12-31', 2)	1996-02-29
SELECT ADD_MONTHS ('1995-12-31', 14)	1997-02-28

ADD_MONTHS accounts for the Gregorian calendar and knows how many days are in each month, including leap years such as 1996.

Comparison of ADD_MONTHS and simple DATE arithmetic

Because of the variable number of days in a month, ADD_MONTH provides a much higher degree of accuracy when projecting dates based on month increments.

Problem

Show the date two month from today (March 20, 2001) using both ADD_MONTHS and simple arithmetic methods.

```
SELECT  DATE
    ,ADD_MONTHS(DATE, 2)
    ,DATE + 60;
```

Data Conversions Using CAST

The CAST function allows you to convert a value or expression from one data type to another. For example:

Numeric to Numeric Conversion
SELECT CAST (50500.75 AS INTEGER);
Result: 50500 (truncated),

SELECT CAST (50500.75 AS DEC (6,0));
Result: 50501. (rounded).

When you cast from a decimal to an integer, the system discards everything to the right of the decimal point. In our example, even though the decimal portion was .75, making the number closer to 50501 than to 50500, the result of the cast is still 50500.

Decimal to Decimal Conversions

Casting from a decimal at one precision level to a decimal at a lesser precision level causes the system to round the result to the nearest value displayed at the new precision level.
SELECT CAST(6.74 AS DEC(2,1));
Result: 6.7 (Drops precision)
SELECT CAST(6.75 AS DEC(2,1));
Result: 6.8 (Rounds up to even number)
SELECT CAST(6.85 AS DEC(2,1));
Result: 6.8 (Rounds down to even number)

Character to Character Conversions
SELECT CAST(last-name AS CHAR (5))
FROM employee
WHERE department_number = 401;

last_name
Johns
Trade
As you can see from the example above, CAST can take the CHAR(20) last_name data field from the employee table and convert it to a CHAR(5) data type. The values displayed as a result of this query are truncated to five characters or padded with trailing blanks up to five characters (if there are less than five characters to start with).

Attributes and Functions

Attributes are characteristics which may be defined for columns, such as titles and formats.

Functions are performed on columns to alter their contents in some way.

Expressions are columns and/or values combined with mathematical operators. (i.e. Col1 + Col2 + 3)

Attributes for columns and expressions include the following:

AS	Provides a new name for a column.	ANSI
TITLE	Provides a title for a column.	Teradata Extension
FORMAT	Provides formatting for a column.	Teradata Extension

Functions for columns and expressions include the following:

CHARACTERS	Count the number of characters in a column.	Teradata Extension
TRIM	Trim the trailing or leading blanks or binary zeroes from a column.	ANSI

AS: Naming a Column or Expression

The AS clause:

- Assigns a temporary name to a column or expression.
- Can be referenced elsewhere in the query.
- Will default column headings to the newly assigned name.
- Use of Keyword AS is optional and may be omitted.

```
SELECT      last_name
            ,first_name
            ,salary_amount / 12 AS monthly_salary
FROM        employee
WHERE       department_number = 401
ORDER BY    monthly_salary;
```

TITLE Attribute

The TITLE attribute allows you to rename columns for output with headings that may contain multiple lines and/or blanks. Column headings may stack up to three levels.

```
SELECT      last_name
            ,first_name
            ,salary_amount / 12 (TITLE 'MONTHLY // SALARY')
FROM        employee
WHERE       department_number = 401
```

```
ORDER BY        3;
```

CHARACTERS Function

The CHARACTERS function is a Teradata-specific function which counts the number of characters in a string. It is particularly useful for working with VARCHAR fields where the size of the string can vary from row to row. In the following example, first_name is a VARCHAR field. In order to determine which employees have more than five characters in their name, we apply the CHARACTERS function to the first_name column.

Problem

To find all employees who have more than five characters in their first name.

Solution

```
SELECT      first_name
FROM        employee
WHERE       CHARACTERS (first_name) > 5;
```

TRIM Function

Use the TRIM function to suppress leading and/or trailing blanks in a CHAR column or leading and/or trailing binary zeroes in a BYTE or VARBYTE column. TRIM is most useful when performing string concatenations.

There are several variations of the TRIM function:

```
TRIM ([expression])                  leading and trailing blanks/binary zeroes
TRIM (BOTH FROM [expression])    leading and trailing blanks/binary zeroes
TRIM (TRAILING FROM[expression]) trailing blanks/binary zeroes
TRIM (LEADING FROM[expression])  leading blanks/binary zeroes
```

Problem

List the employees who have exactly four characters in their last name. The data type of last_name is CHAR(20).

Solution 1

```
SELECT  first_name
        ,last_name (TITLE 'last')
FROM    employee
WHERE CHAR (TRIM (TRAILING FROM last_name)) = 4;
```

Solution 2

```
SELECT    first_name
      ,last_name(TITLE 'last')
FROM    employee
WHERE CHAR(TRIM(last_name))=4;
```

Both will return same result

Using TRIM with Concatenation Operator

The || (double pipe) symbol is the concatenation operator that creates a new string from the combination of the first string followed by the second.

Example 1:

Concatenating of literals **without** the TRIM function:

```
SELECT  '      Jones      '||    ','
        ||  '      Mary       '      AS Name;
Name
------------------------------
     Jones        ,        Mary
```

Example 2:

Concatenating of literals **with** the TRIM function:

```
SELECT    TRIM  (BOTH FROM '                Jones          ')  || ','
          ||  TRIM  (BOTH FROM '            Mary           ')  AS Name;
Name
--------------------
Jones,Mary
```

Using TRIM with Other Characters

TRIM may also be used to remove characters other than blanks and binary zeroes. Using the TRIM function format in the following examples, any defined character(s) may be trimmed from a character string.

Example 1:

```
SELECT TRIM(BOTH '?' FROM '??????PAUL??????') AS Trim_String;
Trim_String
---------------
PAUL
```

Example 2:

```
SELECT TRIM(LEADING '?' FROM '??????PAUL??????') AS Trim_String;
Trim_String
---------------
PAUL??????
```

Example 3:

```
SELECT TRIM(TRAILING '?' FROM '??????PAUL??????') AS Trim_String;
Trim_String
---------------
??????PAUL
```

Using the EXTRACT Function

Extracting From Current Date

The EXTRACT function allows for easy extraction of year, month and day from any DATE data type. The following examples demonstrate its usage.

Query	Result
SELECT DATE; /* March 20,2001 */	01/03/20 (Default format)
SELECT EXTRACT(YEAR FROM DATE);	2001
SELECT EXTRACT(MONTH FROM DATE);	03
SELECT EXTRACT(DAY FROM DATE);	20

Date arithmetic may be applied to the date prior to the extraction. Added values always represent days.

Query	Result
SELECT EXTRACT(YEAR FROM DATE + 365);	2002
SELECT EXTRACT(MONTH FROM DATE + 30);	04
SELECT EXTRACT(DAY FROM DATE + 12);	01

Note: CURRENT_DATE is the ANSI standard for today's date and may be used in place of DATE, which is Teradata specific.

Extracting From Current Time

The EXTRACT function may also be applied against the current time. It permits extraction of hours, minutes and seconds. The following examples demonstrate its usage.

Query	Result
SELECT TIME; /* 2:42 PM */	14:42:32 (Default format)
SELECT EXTRACT(HOUR FROM TIME);	14
SELECT EXTRACT(MINUTE FROM TIME);	42
SELECT EXTRACT(SECOND FROM TIME);	32

Time arithmetic may be applied prior to the extraction. Added values always represent seconds.

Query	Result
SELECT EXTRACT(HOUR FROM TIME + 20);	14
SELECT EXTRACT(MINUTE FROM TIME + 20);	42
SELECT EXTRACT(SECOND FROM TIME + 20);	52
SELECT EXTRACT(SECOND FROM TIME + 30);	Invalid Time

Attribute Functions

Attribute functions are Teradata extensions which return descriptive information about the operand, which may be either a column reference or general expression.
The functions available for attribute information are the following:

- TYPE
- TITLE
- FORMAT
- NAMED
- CHARACTERS
-

Here are some examples of how these functions work:

Query	Results
SELECT DISTINCT TYPE (job_code) FROM job;	INTEGER
SELECT DISTINCT TITLE (job_code)FROM job;	job_code
SELECT DISTINCT FORMAT (job_code)FROM job;	(10)9
SELECT DISTINCT NAMED (job_code)FROM job;	job_code
SELECT DISTINCT CHARACTERS(last_name) FROM employee;	20

Join

Join is a technique for accessing data from more than one table in an answer set. Tables are joined according to columns they have in common. A join between the employee table and the department table could be done according to department number. An example of an Inner Join will be seen on the next screen:

Joins may access data from tables, views or a combination of the two.

Types of Joins

Inner	Rows which match based on join criteria.
Outer*	Rows which match and those which do not.
Cross	Each row of one table matched with each row of another.
Self	Rows matching other rows within the same table.

Inner Join Problem

To get a report that includes employee number, last name, and department name, join the employee table and the department table.
Department number is the common column that determines the way data in these two tables will be joined. Most often joined columns are related to each other as primary and foreign keys.

Inner Join Solution

Suppose we need to display employee number, last name, and department name for all employees. The employee number and last name come from the employee table. The department name comes from the department table.

A join, by definition, is necessary whenever data is needed from more than one table or view, In order to perform a join, we need to find a column that both tables have in common. Fortunately, both tables have a department number column, which may be used to join the rows of both tables.

Solution

```
SELECT  employee.employee_number
        ,employee.last_name
        ,department.department_name
FROM    employee INNER JOIN
        department
ON      employee.department_number = department.department_number;
```

Defining and Using Alias Names

An alias is a temporary name for a TABLE or VIEW defined in the **FROM** clause. It can be useful for abbreviating long table names and is required to join a table to itself.
Once the alias name is defined, it must be used throughout the SQL statement.

```
SELECT      e.employee_number
        ,e.last_name
        ,d.department_name
FROM    employee  e  INNER JOIN
        department d
ON      e.department_number = d.department_number;
```

Cross Joins

A Cross Join is a join that requires no join condition (Cross Join syntax does not allow an ON clause).
Each participating row of one table is joined with each participating row of another table. The WHERE clause restricts which rows participate from either table.

```
SELECT      e.employee_number
        ,d.department_number
FROM    employee e CROSS JOIN
        department d
WHERE       e.employee_number = 1008;
```

Cartesian Product

A completely unconstrained Cross Join is called a Cartesian product. Each row of one table is joined to each row of another table. A Cartesian product results when a CROSS JOIN is issued without a WHERE clause.

```
SELECT employee.employee_number
       ,employee.department_number
FROM   employee CROSS JOIN
       department;
```

Each employee row (26) matched with each department row (9) yields 234 rows of output. An 8,000,000 row table and a 50,000 row table would yield a 400,000,000,000 row answer set. The output of a Cartesian product is often not meaningful however they do have useful application as we shall see.

Self Joins

A self join occurs when a table is joined to itself. Which employees share the same surname Brown and to whom do they report?

```
SELECT emp.first_name    (TITLE 'Emp//First Name')
       ,emp.last_name    (TITLE 'Emp//Last Name')
       ,mgr.first_name   (TITLE 'Mgr//First Name')
       ,mgr.last_name    (TITLE 'Mgr//Last Name')
FROM   employee          emp      INNER JOIN
       employee          mgr
ON              emp.manager_employee_number  =  mgr.employee_number
WHERE emp.last_name = 'Brown';
```

Aggregate Operators

Aggregate operators perform computations on values in a specified group. The five aggregate operators are:

ANSI Standard	Teradata Supported
COUNT	COUNT
SUM	SUM
AVG	AVERAGE, AVG
MAX	MAXIMUM, MAX
MIN	MINIMUM, MIN

AGGREGATE operations ignore NULLs and produce ONLY single-line answers.

Example

```
SELECT
       COUNT ( salary_amount )  (TITLE 'COUNT')
       ,SUM    ( salary_amount )  (TITLE 'SUM  SALARY')
       ,AVG    ( salary_amount )  (TITLE 'AVG  SALARY')
```

```
        ,MAX    ( salary_amount ) (TITLE 'MAX  SALARY')
        ,MIN    ( salary_amount ) (TITLE 'MIN  SALARY')
    FROM   employee ;
```

Result

COUNT	SUM SALARY	AVG SALARY	MAX SALARY	MIN SALARY
6	213750.00	35625.00	49700.00	29250.00

Aggregation using GROUP BY

Problem

To find the total amount of money spent by each department on employee salaries. Without the GROUP BY clause, we could attempt to get an answer by running a separate query against each department.

Solution

```
SELECT  SUM (salary_amount)
FROM    employee
WHERE  department_number = 401
;
Sum (salary_amount)
        74150.00
SELECT SUM (salary_amount)
FROM    employee
WHERE department_number = 403
;
Sum (salary_amount)
        80900.00
SELECT SUM (salary_amount)
FROM    employee
WHERE department_number = 301
;
Sum (salary_amount)
        58700.00
```

Question: What if there are 1000 departments?

GROUP BY provides the answer with a single query, regardless of how many departments there are.

```
SELECT department_number
,SUM (salary_amount)
FROM   employee
GROUP BY department_number
```

department_number	Sum(salary_amount)
401	74150.00
403	80900.00
301	58700.00

GROUP BY and the WHERE Clause

The WHERE clause eliminates some rows before GROUP BY puts them into desired groupings.

Problem

Compute total salaries by department for departments 401 and 403.

Solution

```
SELECT department_number
      ,SUM (salary_amount)
FROM   employee
WHERE department_number IN (401, 403)
GROUP BY   department_number
;
```

GROUP BY and ORDER BY

Example:

```
SELECT department_number
      ,SUM (salary_amount)
FROM   employee
WHERE department_number IN (401, 403)
GROUP BY   department_number
Order by 1
;
```

GROUP BY on Multiple Columns

It is possible and often desireable to GROUP BY more than one column. This permits the query to compute aggregates of groups within groups. In the following example, we are looking for salaries by job code (one group) within department (second group).

By job code, what are the total salaries for departments 401 and 403?

```
SELECT  department_number
              ,job_code
              ,SUM (salary_amount)
FROM   employee
WHERE department_number IN (401, 403)
GROUP BY       1, 2
ORDER BY       1, 2
```

GROUP BY and HAVING Condition

HAVING is just like WHERE , except that it applies to groups rather than rows. HAVING qualifies and selects only those groups that satisfy a conditional expression.

Problem

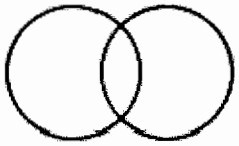
Determine which departments have average salaries of less than $36,000. Generate the report using the HAVING clause:

```
SELECT department_number          AS DEPT
      ,COUNT (*)                  AS #_EMPS
      ,CAST ( SUM (salary_amount) AS   FORMAT 'zz,zzz,zz9.99')
                                         AS TOTAL
```

```
        ,CAST ( MAX (salary_amount) AS   FORMAT 'zz,zzz,zz9.99')
                                          AS HIGHEST
        ,CAST ( MIN (salary_amount) AS   FORMAT 'zz,zzz,zz9.99')
                                          AS LOWEST
        ,CAST ( AVG (salary_amount) AS   FORMAT 'zz,zzz,zz9.99')
                                          AS MEAN
FROM    employee
GROUP BY department_number
HAVING  AVG (salary_amount) < 36000
;
```

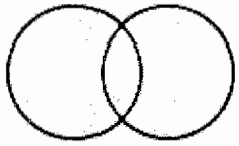
Set Operators

The following are graphic representations of the three set operators, INTERSECT, UNION and EXCEPT.



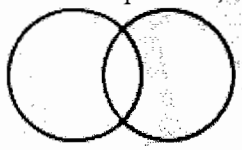
INTERSECT

The INTERSECT operator returns rows from multiple sets which share some criteria in common.



UNION

The UNION operator returns all rows from multiple sets, displaying duplicate rows only once.



EXCEPT

The EXCEPT operator subtracts the contents of one set from the contents of another.

NOTE: Using the Teradata keyword ALL in conjuction with the UNION operator allows duplicate rows to remain in the result set.

UNION:

Before the introduction of the OUTER JOIN feature the UNION operator was used to provide this functionality. UNION is still a viable feature. Here are some rules for its usage:

ALL SELECT clauses:

- Must have the same number of expressions.
- Corresponding expressions must have compatible data types.

The first SELECT statement:

- Determines output FORMAT.
- Determines output TITLE.

The last SELECT statement:

- Contains the ORDER BY clause for the entire result when applicable.
- Uses numeric designators for the ORDER BY columns.

Problem

Show manager 1019 identifying him as the manager, show his employees and identify each of them as an employee.

Solution

```
SELECT          first_name
                ,last_name
                ,'employee' (TITLE 'employee//type')
FROM   employee
WHERE manager_employee_number = 1019
UNION
SELECT          first_name
                ,last_name
                ,' manager '
FROM   employee
WHERE employee_number = 1019
ORDER BY        2
```

INTERSECT

Problem

Create a list all department managers who are assigned subordinate employees. Note that not all department managers have subordinates and not all managers with subordinates are department managers.

Solution

```
SELECT manager_employee_number
FROM   employee
INTERSECT
SELECT manager_employee_number
FROM   department
ORDER BY 1
;
```

EXCEPT

This flagged message refers to "MINUS" because the Teradata-compatible synonym for EXCEPT is MINUS. Use the EXCEPT operator rather than MINUS because although both provide the same functionality, EXCEPT is ANSI compliant.

Problem

To list all department managers who do not have subordinates.

Solution

```
SELECT manager_employee_number
FROM   department
EXCEPT
SELECT manager_employee_number
FROM   employee
```

```
ORDER BY        1
;
```

SET Operators -- Additional Rules

Set operators may be used in most SQL constructs but cannot contain a WITH or WITH...BY clause. Set operators are evaluated in order of precedence, as follows:

- INTERSECT
- UNION
- EXCEPT from left to right

Evaluation order may be manipulated with parentheses.

- Each SELECT statement must have a FROM table_name.
- The GROUP BY clause does not apply to the result set as a whole.

Duplicates are eliminated unless the ALL option is used.

What is a View?

A view is like a 'window' into a table. It provides customized access to base tables by:

- Restricting which columns are visible from the base table.
- Restricting which rows are visible from the base table.
- Combining columns and rows from several base tables.

Restrict Columns from the Base Table

Restrict columns from the base table(s) by explicitly listing the desired column names from the base table(s).

Restrict Rows from the Base Table

Restrict rows to be accessed by using the WHERE clause. This will limit the rows returned from a SELECT statement which references the view.

Creating and Using Views

Problem

Create a view of the employees in department 403 to use for both read and update. Limit the view to an employee's number, last name, and salary.

Solution

```
CREATE VIEW emp_403 AS
SELECT employee_number
    ,last_name
    ,salary_amount
FROM employee
WHERE department_number = 403;
```

To Read From This View

```
SELECT * FROM emp_403;
```

To Update Through This View

```
UPDATE emp_403 SET salary_amount = 100000
WHERE last_name = 'Villegas' ;
```

What is a Join View?

A Join View consists of columns from more than one table.

Example
Create a Join View of the employee and call_employee tables for call dispatch use.

```
CREATE VIEW  employee_call   AS
SELECT  employee.employee_number
      ,last_name
      ,first_name
      ,call_number
      ,call_status_code
      ,assigned_date
      ,assigned_time
      ,finished_date
      ,finished_time
FROM    employee INNER JOIN call_employee
ON call_employee.employee_number = employee.employee_number;
```

Using this view permits SELECT of information from both tables. Note that column names which appear in both tables (i.e. employee_number) must be qualified.

Renaming Columns

You may create a view that renames columns. This is often used to shorten long column names, making queries easier to create.

Example

```
CREATE VIEW  shortcut (emp, dept, last, first, sal) AS
SELECT  employee_number
      ,department_number
      ,last_name
      ,first_name
      ,salary_amount
FROM    employee
WHERE department_number = 201;
```

Now you may SELECT rows using this view.

```
SELECT    last
        ,sal
FROM      shortcut
ORDER BY  last DESC;
```

Aggregate Views

You may create a view that summarizes information by using aggregates.
For instance, let's create a view that summarizes salary information by department.

```
CREATE VIEW deptsals
AS      SELECT department_number        AS department
              ,SUM    (salary_amount)AS salary_total
              ,AVG    (salary_amount)AS salary_average
              ,MAX    (salary_amount)AS salary_max
              ,MIN    (salary_amount)   AS salary_min
        FROM  employee
        GROUP BY      department_number;
```

NOTE: Aggregate and derived columns must be assigned names.

Aggregate View Using HAVING

The HAVING clause may be used within a view definition to restrict which groups will participate in the view.
We may modify the deptsals view to include only departments with average salaries of less than $36,000.

```
REPLACE VIEW deptsals AS
SELECT department_number  AS department
    ,SUM(salary_amount) AS salary_total
    ,AVG(salary_amount) AS salary_average
    ,MAX(salary_amount) AS salary_max
    ,MIN(salary_amount) AS salary_min
FROM  employee
GROUP BY department_number
HAVING AVG(salary_amount) < 36000;
```

To SELECT all departments with average salaries less than $36,000, use the view:

```
SELECT  department
    ,salary_average
FROM   deptsals;
```

Restrictions and Advantages With Views

Restrictions When Using Views

- An index cannot be created on a view.
- A view cannot contain an ORDER BY clause.
- The WHERE clause of a SELECT against a view can reference all aggregated columns of that view.
- Derived and aggregated columns must be assigned a name using one of the following techniques:
    - NAMED clause.

- AS clause (ANSI standard).
- Creating a list of assigned column names in the CREATE VIEW definition.

- A view cannot be used to UPDATE if it contains:
  - Data from more than one table (Join View).
  - The same column twice.
  - Derived columns.
  - A DISTINCT clause.
  - A GROUP BY clause.

Advantages When Using Views

- An additional level of security.
- Help in controlling read and update privileges.
- Simplify end-user's access to data.
- Are unaffected if a column is added to a table; and
- Are unaffected if a column is dropped, unless the dropped column is referenced by the view.

Suggestions For Using Views

- Create at least one view for each base table to restrict user's access to data they don't need to see.
- Create query views as you need them for ad hoc situations.
- Use Access Locks when creating views to maximize data availability to users.

Arithmetic Operators

Operator	Meaning
( )	Evaluated first
*	Multiply
/	Divide
+	Add (positive value)
-	Subtract (negative value)
**	Exponentiation
MOD	Modulo (remainder)

Arithmetic Functions

ABS—absolute value
EXP—raises e to the power of <arg>
LOG—base 10 logarithm
LN —natural logarithm
SQRT—square root

Subqueries:

Suppose you need to find all the employees who are department managers.
You could manually look up all the managers from the department table, and then hard code them into this request:

SELECT ,last_name FROM employee
WHERE employee_number IN (1017,1005,1003);

SELECT Subquery

Subqueries are nested SELECT statements. They can be used to ask a series of questions in order to arrive at a single answer. Subqueries can be nested to any number of levels.

You need to find all the employees who are department managers. You do not want to look them up manually. Use a SELECT subquery.

SELECT last_name FROM employee WHERE employee_number IN
(SELECT manager_employee_number FROM department);

SELECT Subquery with AND
Who are the department managers whose salaries are less than $35,000 and whose budget amounts are greater than $900,000?

SELECT last_name,first_name FROM employeeWHERE salary_amount<35000
AND (SELECT manager_employee_number FROM department WHERE budget_amount >900000)

Rules for Subqueries

- Must be enclosed in parentheses
- Can be the object of an IN or NOT IN clause
- Can be the object of EXISTS or NOT EXISTS clause
- Support quantifiers ALL, ANY, SOME
- Support LIKE or NOT LIKE used with a quantifier
- Can specify more than one column to match
- Generate a DISTINCT list of values
- Cannot use ORDER BY (within the subquery statement).
- Keep in mind that a maximum of 64 tables/views can be specified in an SQL statement.

Determining Whether to Use Subquery or Join

- A *subquery* qualifies which rows SELECTed in the main query will be in the answer set.
- Data SELECTed in the subquery will not be included in the answer set.

- A *join* qualifies which rows from two or more tables will be matched to create rows of an answer set.
- The answer set can include data from one or more of the joined tables.

Outer Joins

An outer join returns qualifying rows and nonmatching rows.

INNER

If you wish to use a SELECT statement with the keywords 'INNER JOIN', you specify a join operation in which qualifying rows from one table are combined with qualifying rows from another table according to the defined join condition. This could be done without these keywords, as in the example
shown previously.

LEFT OUTER

This option indicates which columns should be extended with NULL. Using the rows from table to the left of the keywords "LEFT" or "LEFT OUTER", the rows from the other table will be returned with null or the designation for null (? or other character) for the column. Therefore, if the FROM clause was:
FROM TableA LET JOIN TableB ON TableA.col = TableB.col
TableA.col values would be used to qualify rows for the join operation.
TableB rows would be returned as null for any non-matching rows.

RIGHT OUTER

This option specifies that the table to the right of the keyword "JOIN" should be used to qualify the result. The table to the left of the keywords "RIGHT OUTER JOIN" would be extended to null when non-matching rows are returned. Non-matching rows are any rows that do not result from the inner
join.

FULL OUTER

This option returns both matching and non-matching rows from both tables and extends the non-matching rows with null values.

CROSS

If you use this option, you indicate that you wish to have an unconstrained or
Cartesian Product join. This returns all rows from all tables specified in the FROM clause.

SELECT Statement Join Syntax

SELECT cname [, cname , ...]
FROM tname [aname] { [INNER]
LEFT [OUTER]
RIGHT [OUTER]
FULL [OUTER]

CROSS
}
JOIN tname [aname] ON condition ;
INNER JOIN - All matching rows
LEFT OUTER JOIN - Table to the left is used to qualify, table on the right has nulls when rows do not match.
RIGHT OUTER JOIN - Table to the right is used to qualify, table on the left has nulls when rows do not match.
FULL OUTER JOIN - Both Tables are used to qualify and extended with nulls.
CROSS JOIN - Cartesian product

LEFT Outer Join Example

The example, the Employee and Department tables are used to perform a LEFT OUTER JOIN. In this example, the results will be:
· Employee information for employees with valid department numbers
· Employee information for employees with invalid or null department numbers

```
SELECT E.Department_Number
,Department_Name
,Last_Name
FROM Employee E LEFT OUTER JOIN
Department D
ON E.Department_Number=
D.Department_Number
```

RIGHT Outer Join Example
The example, the Employee and Department tables are now used to perform a RIGHT OUTER JOIN. In this example, the results will be:

· Department and employee information for departments that have been assigned to employees (matching rows)
· Department information where no employees have those department numbers
The Department table is used to qualify the Employee table rows in this example.

```
SELECT D.Department_Number
,Department_Name
,Last_Name
FROM Employee E RIGHT OUTER JOIN
Department D
ON E.Department_Number=
D.Department_Number
```

ON Clause Placement
The placement of the ON clause in the SELECT is important. The rules are:
· The first ON clause (from left to right) is evaluated first.

· An ON clause applies to the immediately preceding join operation.

CASE Expression

The CASE expression is used to return alternate values based on search conditions. There are two forms of the CASE expression:

- Valued
- Searched

- CASE allows for conditional processing of returned rows.
- CASE returns a single result for each row processed.
- Each row is evaluated against each WHEN clause.
- First match returns a result for that row.
- If no match, ELSE result is produced for that row.

Valued CASE format:

Syntax:

```
CASE value-expr WHEN expr1 THEN result1
WHEN expr2 THEN result2
:
ELSE resultn END
```

Calculate the fraction of the total salary of all employees represented by the salaries of Dept. 401.

```
SELECT SUM(
CASE department_number
WHEN 401 THEN salary_amount
ELSE 0
END) / SUM(salary_amount)
FROM employee;
```

Valued CASE Statement

In a Valued CASE statement, you must:

- Specify a single expression to test.
- List the possible values for the test expression that return different results.

Get the ratio of Dept 401 salaries to all employees.

```
SELECT CAST (SUM(
CASE department_number
WHEN 401 THEN salary_amount
ELSE 0
END) / SUM(salary_amount) AS
```

```
numeric (2,2))
AS sal_ratio
FROM employee;
```

Get the total salaries for departments 401 and 501.

```
SELECT CAST (SUM(
CASE department_number
WHEN 401 THEN salary_amount
WHEN 501 THEN salary_amount
ELSE 0
END) / AS NUMERIC (9,2))
AS total_sals_401_501
FROM EMPLOYEE;
```

Searched CASE Statement

In a Searched CASE statement..

- you do not specify an expression to test.
- You specify multiple, arbitrary, search conditions that can return different results.

```
CASE WHEN condition1 THEN value-expr1
WHEN condition2 THEN value-expr2
:
ELSE value-expr END
```

Calculate the fraction of the total salaries represented by departments 401 and 501. Allow for a 10% salary increase for the employees in department 501.

```
SELECT SUM (
CASE
WHEN department_number = 401 THEN salary_amount
WHEN department_number = 501 THEN salary_amount * 1.1
ELSE 0
END) / SUM (CASE WHEN department_number = 501
THEN salary_amount * 1.1 ELSE salary_amount
END ) (FORMAT 'Z.99') AS sal_ratio FROM employee;
```

NULLIF Expression

NULLIF is considered an abbreviated CASE statement.

NULLIF returns the following:

- NULL if the two expressions are equal.
- The first expression if the two expressions are not equal.

NULLIF (<expression1> , <expression2>) is equivalent to:

```
CASE
WHEN <expression1> = <expression2> THEN NULL
```

ELSE <expression1>
END

COALESCE Expression

COALESCE is considered an abbreviated CASE statement which returns the first non-null value in an expression list.

COALESCE (<expression1> , <expression2> [… , <expressionX>])

Returns the first expression, which is not null. If all expressions are null, then returns null.

The following statement uses syntax with CASE as an alternative to the example on following one:

```
SELECT name
,CASE
WHEN office_phone IS NOT NULL THEN office_phone
ELSE home_phone
END
FROM phone_table;
```

Show office phone number if present, else show home phone. Use COALESCE:

```
SELECT name
,COALESCE (office_phone, home_phone)
FROM phone_table;
```

String Functions

String Operator:

Concatenation (||)	-	Putting character strings together

String Functions:

SUBSTRING	-	Obtaining a section of a character string
POSITION	-	Locating a character position in a string
TRIM*	-	Trims blanks from a string
UPPER*	-	Converts a string to uppercase

SUBSTRING Function

SUBSTRING('catalog' FROM 5 FOR 4)	-	'log'
SUBSTRING('catalog' FROM 0 FOR 3)	-	'ca'
SUBSTRING('catalog' FROM -1 FOR 3)	-	'c'
SUBSTRING('catalog' FROM 8 FOR 3)	-	0 length string
SUBSTRING('catalog' FROM 1 FOR 0)	-	0 length string
SUBSTRING('catalog' FROM 5 FOR -2)	-	error

Using SUBSTRING in a List

Display the first initial and last name of all employees in department 403, sorted by last name.

```
SELECT SUBSTRING (first_name FROM 1 FOR 1)
(TITLE 'FI')
,last_name
FROM employee
WHERE department_number = 403
ORDER BY last_name
```

Using SUBSTRING in a WHERE Clause

Find all the customers in the location table whose zip codes end with four zeros.

```
SELECT customer_number
,zip_code (FORMAT '9(9)')
FROM location
WHERE SUBSTRING (zip_code FROM 8 FOR 4)
= '0000'
```

Concatanation:

Use concatenation to display the first and last names for employees in department 403:

```
SELECT first_name || ' ' || last_name
( TITLE 'EMPLOYEE')
FROM employee
WHERE department_number = 403
```

String Concatenation with SUBSTRING and TRIM

Use SUBSTRING with concatenation to display the first initial and last name of employees in department 403.

```
SELECT SUBSTRING (first_name FROM 1 FOR 1 ) || '. ' ||
last_name ( TITLE 'EMPLOYEE') FROM employee
WHERE department_number = 403;
```

POSITION—Character String Position

This is used to Find out position of a letter in column value.

```
SELECT POSITION ('b' IN 'abc',);        -    2
SELECT POSITION ('ab' IN 'abc',);       -    1
SELECT POSITION ('d' IN 'abc',); 0
```

Display a list of departments where the word "SUPPORT" appears in the department name, and list the starting position of the word.

```
SELECT department_name
,POSITION('SUPPORT' IN department_name,)
FROM department
WHERE POSITION('SUPPORT' IN department_name,) >0
ORDER BY department_number
```

Using SUBSTRING and POSITION

Display the first name and last name for each person in the contact table. Contact_name is a VARCHAR column.

```
SELECT SUBSTRING (contact_name FROM POSITION (', ' IN contact_name) +2)
11''11 SUBSTRING (contact_name FROM 1 FOR POSITION (', ' IN contact_name) -1)
(TITLE 'Contact Names') FROM contact;
```

On-Line Analytical Functions

OLAP stands for On-Line Analytical Processing.
These functions include:

- RANK - (Rankings)
- QUANTILE - (Quantiles)
- CSUM - (Cumulation)
- MAVG - (Moving Averages)
- MSUM - (Moving Sums)
- MDIFF - (Moving Differences)
- MLINREG - (Moving Linear Regression)

OLAP functions are similar to aggregate functions in that they:

- Operate on groups of rows (like the GROUP BY clause)
- Can filter groups using QUALIFY (like the HAVING clause)

OLAP functions are unlike aggregate functions because they:

- Return a data value for each qualifying row - not group
- May not be performed within subqueries

Example table to describe OLAP Functions

```
CREATE TABLE salestbl
   (storeid        INTEGER,
    prodid         CHAR(1),
    sales DECIMAL(9,2));

insert into salestb1 values(1001,'A',100000);
insert into salestb1 values(1001,'C',60000);
insert into salestb1 values(1001,D,35000);
insert into salestb1 values(1001,F,150000);
insert into salestb1 values(1002,'A',40000);
```

```
insert into salestb1 values(1002,'C',35000);
insert into salestb1 values((1002,'D',25000);
insert into salestb1 values(1003,'A',30000);
insert into salestb1 values(1003,'B',65000);
insert into salestb1 values(1003,'C',20000);
insert into salestb1 values(1003,'D',50000)
```

Simple Ranking

The Ranking function permits a column to be ranked, either based on high or low order, against other rows in the answer set. By default, the output will be sorted in descending sequence of the ranking column, which usually correlates to ascending rank.

The syntax for the RANK function is:

RANK(colname)

where colname represents the column to be ranked and the descending sort key of the result.

Problem

Show the ranking of product sales for store 1001.

Solution

```
SELECT storeid, prodid, sales, RANK(sales)
FROM salestbl
WHERE storeid = 1001;
```

storeid	prodid	sales	Rank
1001	F	150000.00	1
1001	A	100000.00	2
1001	C	60000.00	3
1001	D	35000.00	4

Ranking With Qualification

QUALIFY performs like the HAVING clause by requesting a specific range in the output.

Problem

To get the top three selling products by store.

Solution

```
SELECT storeid, prodid, sales, RANK(sales)
FROM salestbl
GROUP BY storeid
QUALIFY rank(sales) <= 3;
```

Ranking with order by

```
SELECT storeid, prodid, sales, RANK(sales)
FROM salestbl
GROUP BY storeid
```

Order by 1,2,3
QUALIFY rank(sales) <= 3;

Rank() OVER

```
SELECT      Product_ID
            ,Sale_Date
            ,Daily_Sales
            ,RANK()  OVER (PARTITION BY Product_ID
                           ORDER BY Daily_Sales DESC )  AS Ranked
FROM   Sales_table
QUALIFY Ranked <= 2 ;
```

*The next SELECT uses **MAX** to produce a report for the dates in September:*

```
SELECT      Product_ID
            ,Sale_Date
            ,Daily_Sales  (Format '$$$,$$$.99')
            ,max(daily_sales) OVER ( partition by product_id
                          ORDER BY product_id, daily_sales desc) AS "Max
Over"
FROM   Sales_table
WHERE EXTRACT(MONTH FROM Sale_Date) = 9
;
```

The next SELECT uses **COUNT** to produce a report for the dates in September:

```
SELECT      Product_ID
            ,Sale_Date
            ,Daily_Sales  (Format '$$$,$$$.99')
            ,COUNT(*) OVER ( ORDER BY product_id, daily_sales desc)
                          AS "Count Over"
FROM   Sales_table
WHERE EXTRACT(MONTH FROM Sale_Date) = 9
;
```

The next SELECT uses **ROW_NUMBER** to produce a report for the dates in September:

```
SELECT      Product_ID
            ,Sale_Date
            ,Daily_Sales  (Format '$$$,$$$.99')
            ,ROW_NUMBER() OVER ( ORDER BY product_id, daily_sales
desc)
                          AS "Row Number"
FROM   Sales_table
WHERE EXTRACT(MONTH FROM Sale_Date) = 9 ;
```

Sampling

To display sample records we can use Sample.
Syntax: Sample n;

Select * from Employee sample 10;

Teradata SQL Performance Tuning Checklist

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

Explain

EXPLAIN may be used on any SQL statement, except EXPLAIN itself. Use EXPLAIN:

- To get costs of different approaches.
- To find unexpected PRODUCT joins.
- To find the best exclusion joins (EXCEPT, NOT IN, NOT EXISTS). Even though Teradata recommends to use NOT IN vs. NOT EXISTS, there are some cases when NOT EXITS has better performance. Running EXPLAINS for both approaches will help to find more cost effective plan for a particular query.
- To choose between subquery or join. Sometimes simple joins are more cost efficient than subqueries. Therefore, make sure to compare both approaches.
- To insure that sync scanning is enabled.

When using EXPLAIN, look for key words and phrases:

- Execution cost and row count estimates depend on:
  - STATISTICS
- Actual execution time depends on:
  - Other request being processed by the DBS
  - Channel or network usage
- How the data is relocated in preparation for a JOIN: REDISTRIBUTED by hash code to all AMPS; DUPLICATED on all AMPs. DUPLICATED SPOOLs or repeated access to the same data can be resolved with derived tables, CASE statements, correlated subqueries, etc.
- How much spool space is used for a query. "…LAST USE…" Means spool file is not longer needed and will be released when this step completes.
- If you have partitioned tables used in a query, look for how many partitions are accessed : "…A SINGLE PARTITION OF…" or "…N PARTITIONS OF…".
- Row and time estimates are based on the CONFIDENCE.
  - "…WITH HIGH CONFIDENCE…" – statistics available on an index or column.
  - "…JOIN INDEX CONFIDENCE…" – a join condition via a primary index.
  - "…WITH LOW CONFIDENCE…" – random sampling of the index(es); statistics available, but are "AND-ed"/"OR-ed" together with conditions on non-indexed columns.
  - "…WITH NO CONFIDENCE…" – random sampling based on AMP row count; one of the joined tables has no confidence; statistics do not exist for either join.
- Check if existing indexes are used by query optimizer.

Teradata PMON utility allows to look at the actual SQL and EXPLAIN that is currently being executed within each session. From either the Session Detail screen or the Session Skew screen you may drill down to see the EXPLAIN that shows a list of all the steps required to execute the current SQL. The steps that are currently executing are highlighted. The online EXPLAIN can help to identify the possible query bottlenecks.

Statistics

To find out whether statistics has been collected on column/index/table used in a query and how current this statistics is, use the following statement:

HELP STATISTICS <database-name>.<table-name>;

Several ways to influence the optimizer by means of the statistics:

- Collect statistics on non-unique indexes
- Collect statistics on non-index join columns
- Define statistics on other columns involved in WHERE conditions and join conditions
- Collect statistics on small tables

The efficacy of collected statistics varies with the types of access used on a table. If performance does not seem to be improved by the statistics you have collected, then the Optimizer is probably not using the column to access or join the table. When you observe this behavior and EXPLAINs do not indicate that the Optimizer is using the column as you thought it might, use the DROP STATISTICS statement to remove the statistics for that column from the data dictionary.

Query Redesign Techniques:

Case Statement Optimization

OPTION 1:

```
CASE <column/value>
        WHEN <value/column> THEN result-1
        WHEN <value/column> THEN result-2
                .....
        ELSE result-n  END
```

OPTION 2:

```
CASE
        WHEN <condition> THEN result-1
        WHEN <condition> THEN result-2
                .....
        ELSE result-n  END
```

Where <condition> can be: any valid comparison operator, BETWEEN, IN, NULLIF, or COALESE

Possible CASE optimization techniques include:

CASE Too Many Comparisons:

```
UPDATE Department
SET budget_amount = CASE WHEN department_nbr <300 THEN budget_amount
WHEN department_nbr >500 THEN budget_amount
WHEN department_nbr <402 THEN budget_amount*1.05
WHEN department_nbr >402 THEN budget_amount*1.10
WHEN department_nbr =402 THEN budget_amount*1.15
END
```

Better CASE for Performance:

```
UPDATE Department
SET budget_amount = CASE WHEN department_nbr <402 THEN budget_amount*1.05
WHEN department_nbr >402 THEN budget_amount*1.10
```

```
ELSE budget_amount*1.15
END
WHERE department_nbr
BETWEEN 300 AND 500;
```

Horizontal Reporting:

```
SELECT
SUM(CASE WHEN department_nbr = 402 THEN budget_amount END) AS DEPT_402
,SUM(CASE WHEN department_nbr <> 402 THEN budget_amount END) AS
DEPT_NOT_402
,SUM(budget_amount) AS DEPT_ALL FROM Department;
```

Better CASE for Performance:

```
SELECT
SUM(CASE WHEN department_nbr = 402 THEN budget_amount END) AS DEPT_402
,DEPT_ALL - DEPT_402  AS DEPT_NOT_402 ,SUM(budget_amount) AS DEPT_ALL
FROM Department;
```

Compound Comparison:

```
CASE
WHEN department_nbr = 401and budget_amt <90000 THEN budget_amt *1.15
WHEN department_nbr = 401 and budget_amt >=90000 THEN budget_amt
WHEN department_nbr = 402 and budget_amt <90000 THEN budget_amt*1.25
WHEN department_nbr = 402 and budget-amt >=90000 THEN budget_amt*1.20
ELSE budget_amt
END
```

Better CASE for Performance:

```
CASE WHEN department_nbr = 401and budget_amt <90000 THEN budget_amt *1.15
WHEN department_nbr = 402 and budget_amt <90000 THEN budget_amt*1.25
WHEN department_nbr = 402 and budget-amt >=90000 THEN budget_amt*1.20
ELSE budget_amt
END WHERE department_nbr IN (401, 402);
```

Nested CASE for even better performance:

```
CASE department_nbr          WHEN 401 THEN
(CASE WHEN budget_amt <90000 THEN budget_amt*1.15
ELSE budget_amt END) WHEN 402 THEN
(CASE WHEN budget_amt <90000 THEN budget_amt*1.25
ELSE budget_amt*1.20 END)
END WHERE department_nbr IN (401, 402);
```

Use of a Temporary Table

The example below shows how temporary tables can significantly improve a query performance with aggregations and joins. Sometimes, it is better to aggregate before you join, in other words to perform an "early group by" to reduce the number of rows as early as possible.

Assume the sales table has 2 billion rows and 1500 locations

```
SELECT  si.region, sum(sl.sales) as total_sales
FROM  sales sl, store_info si
WHERE  sl.location = si.location
GROUP BY si.region;
```

Use of TEMP1 temporary table will replace 2 billion row join with a 1500 row join.

```
TEMP1     =  SELECT sl.location, sum(sl.sales) as sales
             FROM sales sl
             GROUP BY sl.location;
SELECT si.region, SUM(temp1.sales)
FROM temp1, store_info si
WHERE TEMP1.location=si.location
GROUP BY si.region;
```

Temporary table vs. a Secondary Index

Under certain circumstances it may be faster and consume far fewer resources to:

- Copy a table (or even better filter it with a WHERE clause) into a temp table with a different PI
- Use the temp table to do a PI join
- Drop the temp table

Use of a Derived Table

Using derived tables is another efficient way of optimizing SQL queries. These tables are materialized in SPOOL, populated with SELECT statement, used for the main SELECT and then released immediately upon completion. Privileges are not needed.

Assume that Discount table has 5 billion rows and 2 thousand unique (sls_key,item_id)

```
SELECT
sl.sls_key
, sl.item_id
, SUM(dcnt.dcnt_qty * dcnt.dcnt_unit_amt)
FROM  Sales sl INNER JOIN Dicounts dcnt
ON sl.sls_key=dcnt.sls_key AND sl.item_id=dcnt.item_id
INNER  JOIN Vendors vndr
ON vndr.vendor_id=sl.vendor_id
WHERE sl.bus_date BETWEEN CAST ('11/01/2003' AS DATE FORMAT 'MM/DD/YYYY')
AND
CAST('11/15/2003' AS DATE FORMAT 'MM/DD/YY')
AND dcnt.dcnt_typ='DISC'
GROUP BY sl.sls_key, sl.item_id;
```

Efficient Use of a Derived Table: 5 billion row join will be replaced with 2 thousand row join

```
SELECT sl.sls_key
,sl.item_id
,dcnt_derived.total_dcnt
```

```
FROM Sales sl INNER JOIN
        (SELECT dcnt.sls_key, dcnt.item_id, (dcnt.dcnt_qty * dcnt.dcnt_unit_amt) AS total_dcnt
        FROM Dicounts dcnt
        WHERE dcnt.dcnt_typ='DISC'
        GROUP BY dcnt.sls_key, dcnt.item_id) dcnt_derived
ON sl.sls_key=dcnt_derived.sls_key AND sl.item_id=dcnt_derived.item_id
INNER  JOIN Vendors vndr
ON vndr.vendor_id=sl.vendor_id
WHERE sl.bus_date BETWEEN CAST ('11/01/2003' AS DATE FORMAT 'MM/DD/YYYY')
AND
CAST('11/15/2003' AS DATE FORMAT 'MM/DD/YY');
```

Inefficient Use of a Derived Table

Assume that it is necessary to change the table layout from horizontal to vertical. This kind of transformation is common when working with multidimensional data models.

```
SELECT *
FROM ( SELECT Class_id, Year, Month, bdgt_amt_1 as bdg_amt, act_amt_1 as act_amt, '001'
as sub_type
FROM Class_table1
WHERE Class_id < 400) DT_Class1
UNION
SELECT *
FROM( SELECT Class_id, Year, Month, bdgt_amt_2 as bdg_amt, act_amt_2 as act_amt, '002'
as sub_type
FROM Class_table1
WHERE Class_id < 400) DT_Class2
UNION
SELECT *
FROM ( SELECT Class_id, Year, Month, bdgt_amt_3 as bdg_amt, act_amt_3 as act_amt, '003'
as sub_type
FROM Class_table1
WHERE Class_id < 400) DT_Class3;
```

Since the same table is used three times, a derived table may not be the best solution because it increases the workload and SPOOL utilization.

A Temporary Table is more efficient than a Derived Table for this query:

```
CREATE VOLATILE TABLE TEMP_Class
(Class_id       SMALLINT
,Month          SMALLINT
,Bdgt_amt       DECIMAL(9,2)
,Act_amt        DECIMAL(9,2)
,Sub_type       VARCHAR(5))
UNIQUE PRIMARY INDEX (Class_id)
ON COMMIT PRESERVE ROWS;
INSERT INTO TEMP_Class
```

```
SELECT Class_id, Year, Month, bdgt_amt_1, act_amt_1, bdgt_amt_2, act_amt_2, bdgt_amt_3,
act_amt_3
FROM Class_table1
WHERE Class_id < 400;
```

Now, the table is created one time in SPOOL and then populated for multiple accesses.

```
SELECT Class_id, Year, Month, bdgt_amt_1 as bdg_amt, act_amt_1 as act_amt, '001' as
sub_type
FROM TEMP_Class
UNION
SELECT Class_id, Year, Month, bdgt_amt_2 as bdg_amt, act_amt_2 as act_amt, '002' as
sub_type
FROM TEMP_Class
UNION
SELECT Class_id, Year, Month, bdgt_amt_3 as bdg_amt, act_amt_3 as act_amt, '003' as
sub_type
FROM TEMP_Class;
```

Now, the same table is used three times but only populated once, so it is approximately three times as efficient as the previous approach. This Volatile Table is still available for other requests by the same user. Additionally, UNION eliminates duplicate values and UNION ALL does not. Therefore, if no duplicates are expected, UNION ALL is faster.

Moving Logic from WHERE to SELECT

Rearranging two queries into one by moving logic from WHERE clause to SELECT also solves the problem of duplicated spool files or repeated access to the same data.

```
SELECT t1.column1, count('1') as Dec
FROM Table1  t1
JOIN Table2 t2 ON t1.ID = t2.ID
JOIN table3 t3 ON t1.ID = t3.ID
JOIN table4 t4 ON t1.date = t4.date
WHERE t4.monthID = 1234
GROUP BY 1;

SELECT column1, count('1') as Nov
FROM Table1  t1
JOIN Table2 t2 ON t1.ID = t2.ID
JOIN table3 t3 ON t1.ID = t3.ID
JOIN table4 t4 ON t1.date = t4.date
WHERE t4.monthID = 1233
GROUP BY 1;
```

Efficient Use of a CASE statement eliminates repeated access to the same data

```
SELECT t1.column1,
SUM(CASE WHEN t4.monthID = 1234 THEN 1 ELSE 0 END) as Dec,
SUM(CASE WHEN t4.monthID = 1233 THEN 1 ELSE 0 END) as Nov
FROM Table1  t1
JOIN Table2 t2 ON t1.ID = t2.ID
```

```
JOIN table3 t3 ON t1.ID = t3.ID
JOIN table4 t4 ON t1.date = t4.date
GROUP BY 1;
```

"Magic Sets"

This type of query is common for DSS complex request that are generated by BI tools. Typically these SQL request have several levels of correlated subqueries. Look for common qualifications that can reduce the number of rows and spool size of the data sets used in subqueries. The example below shows how to redesign typical DSS query using the "Magic Set" approach.

```
SELECT  P.PARTKEY, PS.AVAILQTY, PS.SUPPKEY
FROM    PARTTBL P, PARTSUPP PS
WHERE  P.PARTKEY = PS.PARTKEY
      AND P.SIZE = 15
      AND P.TYPE LIKE '%EXAMPLE'
      AND PS.SUPPLYCOST = (
           SELECT  MIN(PS.SUPPLYCOST)
           FROM  PARTSUPP
           WHERE P.PARTKEY = PS.PARTKEY ) ;
```

Efficient Use of a "Magic Set"

```
SELECT     P.PARTKEY, PS.AVAILQTY, PS.SUPPKEY
FROM              PARTTBL P, PARTSUPP PS
WHERE      P.PARTKEY = PS.PARTKEY
AND        P.SIZE = 15
AND        P.TYPE LIKE '%EXAMPLE'
AND         (P.PARTKEY, PS.SUPPLYCOST) IN (
              SELECT    PS.PARTKEY, MIN(PS.SUPPLYCOST)
              FROM              PARTSUPP P,
                        (SELECT     P.PARTKEY FROM PARTTBL P
                         WHERE  P.SIZE = 15
                         AND       P.TYPE LIKE '%EXAMPLE') AS MAGIC
                       WHERE   P.PARTKEY = PS.PARTKEY
                       GROUP BY P.PARTKEY) ;
```

Group by vs. Distinct

Group By in Teradata sorts locally on the AMP, Distinct performs the sort after data is redistributed to all AMPs. Therefore, GROUP BY is more efficient in Teradata than DISTINCT. However, there is an exclusion of this rule: Use DISTINCT with a VERY FEW Duplicates.

```
SELECT DISTINCT Dept_No
FROM EMPLOYEE;
SELECT Dept_No
FROM EMPLOYEE
GROUP BY Dept_No;
```

IN vs. EXISTS

IN and NOT IN are more efficient operations in Teradata than EXISTS and NOT EXISTS. It usually takes less spool space, however always compare the EXPLAINs, since under certain circumstances NOT EXISTS performs better.

```
SELECT dpt.*  FROM Department AS dpt
WHERE NOT EXISTS
(SELECT * FROM Employee emp WHERE dept.dept_nbr  = emp.dept_nbr);
SELECT  dpt.* FROM Department AS dpt WHERE dpt.dept_nbr NOT IN
(SELECT emp.dept_nbr FROM Employee emp);
```

Multi Statement Requests

Put the SELECT on the same line as the ";"

```
INSERT INTO A SELECT A1 FROM AAA;
 INSERT INTO A SELECT A1 FROM AAA;
```

Multi statement Request for Better Performance:

```
INSERT INTO A SELECT A1 FROM AAA
;INSERT INTO A SELECT A1 FROM AAA;
```

When you putt the ";" at the beginning of the next statement, Teradata views these three INSERTs as one process. It spools up the data and does one INSERT process.

Date Usage

In Teradata, DATE is stored as an integer using the following format:

YYYYMM → YYY = offset from 1900

So, today 2004-08-09 is stored as:

(09 * 1) +(08*100) +((2004-1900) *10000) = 1040809

Using Teradata date manipulations at the integer level is more efficient as opposed to using date functions that have conversion overheads.

```
SELECT Sales_date, Sales_Amt, CSUM(Sales_Amt, Sales_date) as "CSUM"
FROM Sales
GROUP BY Extract(YEAR from sale_date), Extract(MONTH from sale_date);
```

Better performance:

This OLAP query will run faster if instead of EXTRACT (YEAR..and MOTH) it has the following substitution:

```
SELECT Sales_date, Sales_Amt, CSUM(Sales_Amt, Sales_date) as "CSUM"
FROM Sales
GROUP BY sale_date/100;  -- leaves → YYMM
```

It is also efficient to use intervals with date manipulations:

```
SELECT current_date, current_date + interval '1' day, current_date – interval '2' month;
```

Update vs. Insert

If possible, try to use INSERT instead of UPDATE. Temporary tables can be used to rewrite UPDATE queries to INSERT ones.

Use of Excessive DDL
When use permanent temporary tables, do "Delete all" if possible rather than drop table and recreate.

Table Alias Names
It is strongly recommended using table alias when writing query. Table alias in the query should be consistent and be applied throughout the query.

Other SQL tips

- Use consistent, standard, trick-free SQL syntax.
- When constructing queries that contain literals built from user supplied input, make certain that you preprocess the user input to avoid syntactically illegal queries
- Do not issue SELECT * FROM … queries; instead, always spell out all the members of the SELECT list.
- Use ANSI SQL syntax instead of Teradata specific syntax.
- Avoid unnecessary conditions in WHERE clauses.
- In SQL queries, always perform your own constant folding and propagation, if possible.
- Be familiar with the tricky aspects of three-valued logic when NULLs can be involved.
- Case insensitive comparisons should be done by using the function LOWER, not UPPER.
- The left side of a search condition should be a simple column name; the right side should be an easy-to-look-up value.
- Apply the distributive law: Write *A* AND (*B* OR *C*) instead of (*A* AND *B*) OR (*A* AND *C*).
- Transform logical expressions involving NOT into something more readable. Use De Morgan's theorem:
    - NOT (*A* AND *B*) = (NOT *A*) OR (NOT *B*)
    - NOT (*A* OR *B*) = (NOT *A*) AND (NOT *B*)
- Depending on available indexes, a logical expression in DNF (disjunctive normal form) may compile into better code than the equivalent expression in CNF (conjunctive normal form), or vice versa.
- A LIKE pattern that starts with a literal character will allow the use of an index (if one is available). If the pattern starts with a wildcard character, no index can be used.
- Keep the number of grouping columns as small as possible.
- Reduce before you expand: GROUP BY (usually with aggregation involving COUNT, SUM, etc.) tends to reduce row counts, whereas JOIN tends to expand row counts.
- Use the new ANSI style join syntax with ON conditions.
- For complex (inner and outer) joins, first determine the expected cardinality of the join result.
- For OUTER JOIN expressions, make certain that you have put the search restrictions in the correct place.
- Issue SQL statements that change data (INSERT, UPDATE, DELETE) in batches, if possible.
- Update multiple columns with the same UPDATE … SET statement, rather than with multiple UPDATE statements.

- Delete multiple rows with the same DELETE statement, rather than with multiple DELETE statements.
- Do data types in ON conditions match correctly to avoid type conversion

Top 10 Teradata Performance Tips

All users should apply these techniques when submit query to Teradata. These tips will help your query run quicker and save system resources

1. Explain query before submit

What to look for:

1. Confidence -- each step in the Explain text will have a confidence and row count associated with it
2. Timing -- each step in the explain will have a timing associated with it
3. Spool -- Look to see how the spool files are created. They will be LOCAL, REDISTRIBUTED, and DUPLICATED
4. Joins -- MERGE, NESTED, and PRODUCT. Look to see if you are doing product joins and if they are constrained or unconstrained

2. Apply Table Alias Name Throughout the Query

It is strongly recommended using table alias when writing query. Table alias in the query should be consistent and be applied throughout the query

3. Poor Primary Index Choice

Always specify Primary Index when create table (table DDL). Choose the most distinctive column(s) when possible for better data distribution, hence better query performance

4. Missing Join Conditions in the Where Clause

Make sure all tables involved in the query are qualified in the WHERE clause. A product join is guaranteed if no join conditions specified

5. No Statistics on Join Column and Index

Collect statistics for all column(s) and index(es) involved in the WHERE clause. Statistics on small table is equally as important as on large table. Avoid expression of column in WHERE clause, because no statistics will be used on substr or column concatenation

6. Avoid Long "In-List"

Limit In-List qualification to less than 70 values in WHERE clause. Large In-list causes excessive CPU usage, forces all-rows scan, and runs many times longer. Use sub-select in place of large In-List.

7. Group By vs. Distinct

Group By sorts locally on the AMP, Distinct performs the sort after data is redistributed to all AMPs. Therefore less data is redistributed

8. Do Not Run Test Job in Production

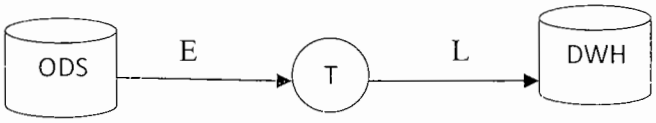
QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

ETL CONCEPTS
(INFORMATICA)

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

ETL System Design:

ODS: It contains transactions related data for business.



From ODS we will extract data then transform that data based on business logics then load data in to DWH.

In real time we cannot directly extract the data from ODS because clients are not provide access to their ODS.Reason is ODS is main system which contains all the transactions which are running daily based of the client.

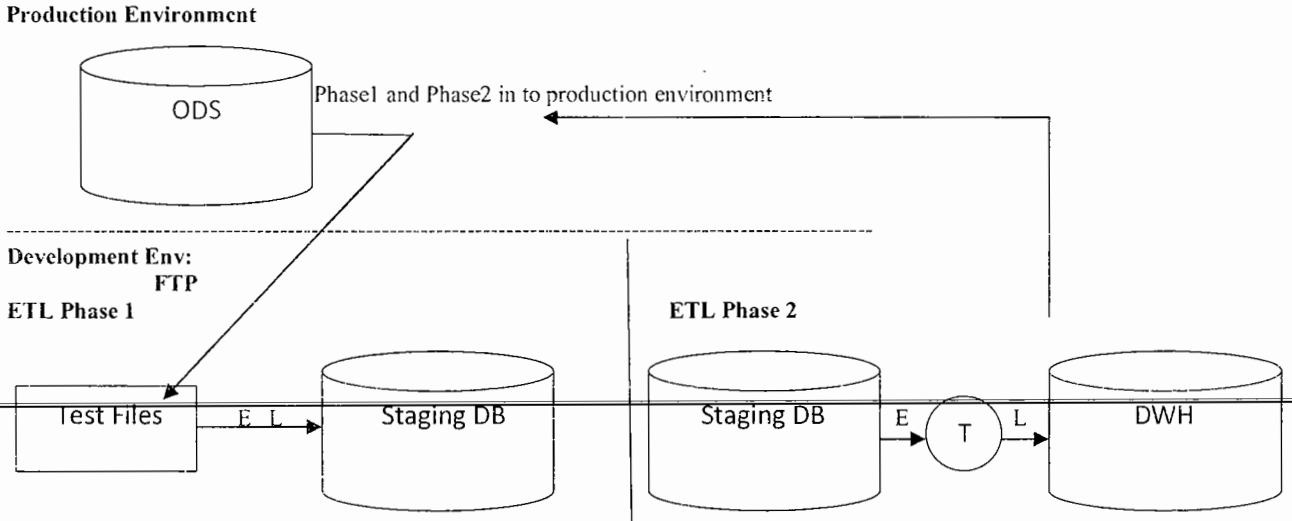
The client will provide access to ODS to their internal employees not provide access to external employees (like the people who are developing DWH).Because there is possibility to insert, update, delete operations can perform in ODS data.

Client: Who need DWH Eg: Vodafone
Vendor: Who develop the DWH Eg: Wipro

If client will not provide access to ODS how to implement DWH?
The client will provide the test data to implement the DWH. The test data is in the form of test files.

ETL system design means representing how data is moved from source to DWH.
For this there are two phases.

1. ETL Phase 1
2. ETL Phase 2



In ETL system design we are going to design a temporary data base which is staging database. This is same as operational database; this staging database is developed at vendor locations.

Instead of directly using ODS why we are creating staging DB?

The application which are developed using informatica in such a way that extract data from DB to load into DWH. Here clients are providing test files instead of access to DB, so we are creating Staging area.

ETL Phase1: The following activities will take place in ETL Phase1.

1. Define the source as file system
2. Create a staging database (Dummy operational Database or Test DB)
3. Define simple pass applications to move data from files to staging database.
4. Perform data profiling(Study and analyze)

ETL Phase2: The following activities take place in ETL Phase 2.

1. Define the staging DB as source
2. Design Data warehouse.
3. Design ETL applications.
4. Perform the following steps.
   a. ETL Unit testing
   b. Data validation testing
   c. Performance testing
   d. ETL UAT(User Acceptance Testing)

Like this DWH applications are developed in vendor locations, which will deploy in production environment by using operational source and load data in to DWH. (Deploy Phase 2 in to production environment)

Data acquisition Process:

Data Extractions: It is the process if reading the data from various operation systems like SAP, People Soft, Cobol Files, Flat files, and XML files …e.t.c

Data Transformation: It is the process if cleansing the data and converting the data in to the required business format .The following data processing activities takes place in staging area.

1. Data Merging
2. Data Cleansing
3. Data Scrubbing
4. Data Aggregation

Data Loading: It is the process of inserting the data in to the target system.

There are two types of data loads.

1. Initial Load or Full load
2. Incremental Load or Delta load.

Data Merging: It is as process of combining the data from multiple inputs in to a single output pile line. There are two types of data warehousing operations.

1. Data Join
2. Union

If we want to combine data definitely we require two sources. If these two sources are having

Different data like one source contains Employee data and other source contains Department data then we are extract data in to staging area from different sources and integrate data using JOIN.

If these two sources having same data like one source contains Sales data and other source contains Sales Info data, then we are extract data in to staging area these different sources and integrate using UNION.

Data Cleansing: It is the process of changing the inconsistencies and inaccuracies.
The following activities will take place under data cleansing.

1. Removing unwanted data.(Data filtering)
   Eg: In source I am having null records but while loading data in to target I will remove all NULL records.
2. Transforming inconsistent data in to consistent format.
   Eg: In source data like Hyd,Hyderabad,Mub,Bhuvaneswar..but while loading data in to target it should be in consistent format like Hyd,Mumb,Bhuv or Hyderbad,Mumbai,Bhuvaneswar..

Data Scrubbing: It is as process of deriving new data definitions using existing source data definition.

Data definition is nothing but METADATA.(Data about data)

Read metadata definition from both source and target. In target there is a column like sales_Amount and in source there are no columns like that. So we need to define that target column in staging area.

Using existing source definition we are deriving new definitions depends on warehouse requirements. It is called scrubbing.

Data Aggregation: It is the process of calculating the summaries using following aggregate functions Sum(),Avg(),Mac(),Min()….e.t.c
From the source we are gathering detail data in to staging area here we perform the aggregations to get summary data.

Initial Load or Full Load: It's the process of first time load in to DWH.At first load all the required data moves to target system.

Incremental Load or Delta Load: It is the process if loading only new records in to the target system. Incremental load takes places after initial load.

Informatica 8.6

Informatica – Company Information

- Founded in 1993
- Leader in enterprise solution products
- Headquarters in Redwood City, CA
- Public company since April 1999 (INFA)
- 2000+ customers, including over 80% of Fortune 100
- Strategic partnerships with IBM Global Services, HP, Accenture, SAP, and many others
- Technology partnership with Composite Software for Enterprise Information Integration (EII) – real-time federated views and reporting across multiple data sources
- Worldwide distribution

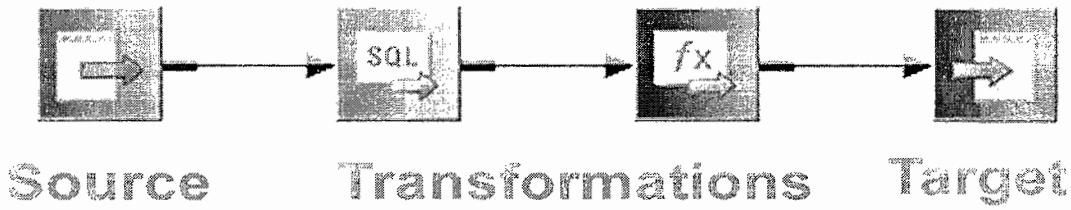
An informatica is as Client-Server technology and an integrated tool set used for designing, running, monitoring and administrating the data acquisition application known as Mapping.

A mapping is a structural representation of data flow from source to target.

An informatica is a single, unified, enterprise data integration platform which allows you to access the data from various business systems,integreate the data in any uniformat and delivers the data throughout the enterprise at any speed.

Data acquisition is made up of three components.
Source: This defines data extraction
Transformation rule: This defined data processing
Target: This defines the dataloading.

Before going to design and develop the DWH first we need to design plan for data acquisition. This plan is called as **mapping** in informatica.



To prepare the plan for data acquisition we need a client GUI based technology we need to execute that plan on Server.

In Client GUI environment we deal with data definition or Metadata while deploying mapping
in sever we deal with Actual Data.

So server is the main component to extract, transform and loading of the plan prepared from client.

Repository: A repository is a central metadata storage location which contains all the metadata which is required to implement ETL process.

Repository is created by administrator at the time of installation.

We are having two types of ETL tools.

1. Code Based ETL Tools : Eg : SAS,Teradata Utilities.....e.t.c
2. GUI Based ETL Tools: Eg: Informatica,ABINITIO,DATASTAGE...e.t.c

Infomratica software releases in different flavors.

Informatica Power Centre: This flavor has a powerful ETL engine to process the massive amount of source data. This flavor supports extracting the data from ERI packages like SAP,People Soft...e.t.c

Informatica power Mart: This flavor of informatica does not support ERP applications to read the data. The small and middle scale enterprises can buy the license of power mart to handle low volumes of data to be processed.

Mapping

- Logically Defines the Data Integration Process:
- Reads data from sources
- Applies transformation logic to data
- Writes transformed data to targets

Source Transformations Target

Transformations

- Generate, modify, or pass data
- Data passes into and out of transformations through ports that you link in a mapping

Task (Session)

- An executable set of action , functions or commands
- Session task runs a mappin
- Command task runs a shell script

Workflow

- A collection of ordered tasks
- Tasks can be linked sequentially, concurrently or both

- Links can depend on the successful completion of previous tasks

Metadata

- Defines data and processes
- Examples:
  - Source and target definitions
    - Type (flat file, database table, XML file, etc)
    - Datatype (character string, integer, decimal, etc)
    - Other attributes (length, precision, etc.)
  - Mapping logic
  - Workflow logic
- Stored in a metadata repository

Power Centre Components:

When we install the powercentre software the following main components get installed.

1. Power centre Repository
2. Power Centre Clients
3. Repository Service
4. Integration Service

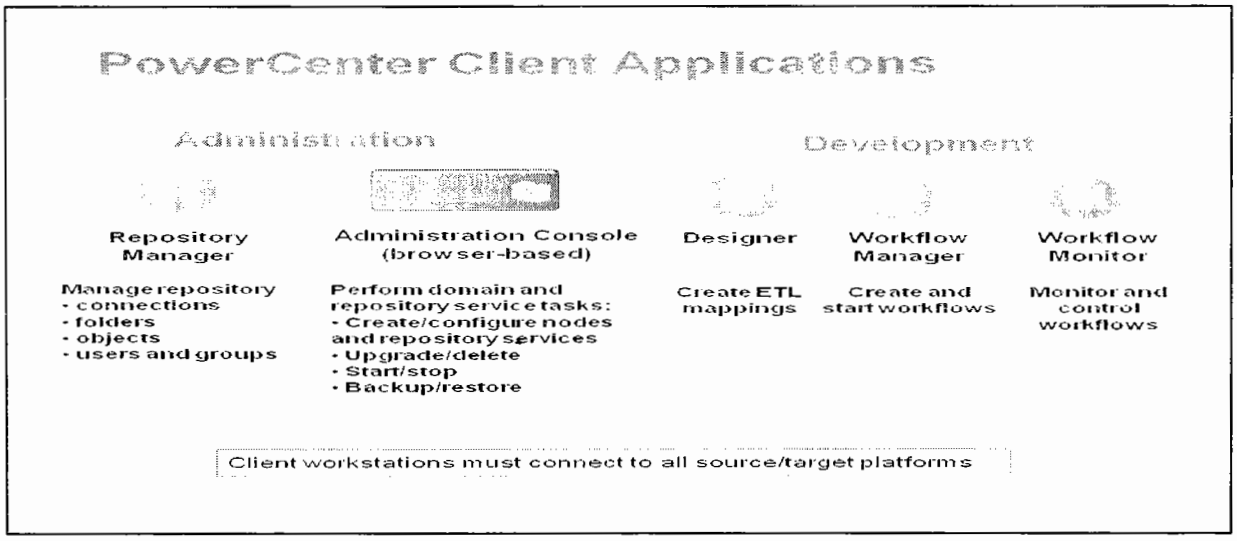
Power Centre Repository:

Informatica powercnetre supports a relational repository to store the metadata.
The repository database tables contains metadata required to Extract, Transform and load data
The powercentre applications access the repository database tables through the Repository Service.

POWERCENTER CLIENTS:

The Power Center Client consists of the following applications that we use to manage the repository, design mappings, mapplets, and create sessions to load the data:

1. Designer
2. Repository Manager
3. Workflow Manager
4. Workflow Monitor



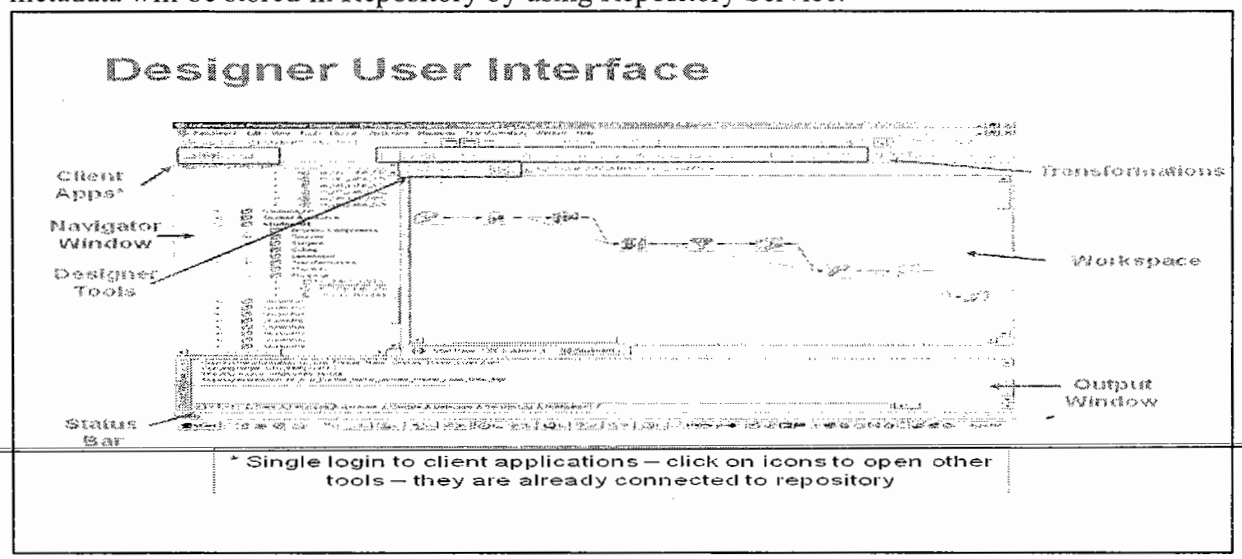
1. Designer:

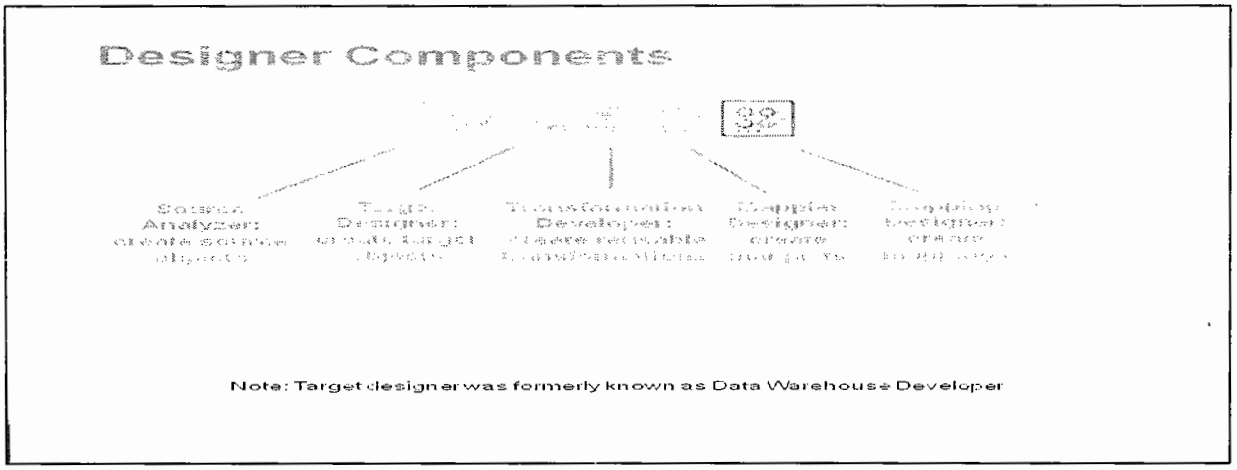
Use the Designer to create mappings that contain transformation instructions for the Integration Service.

The Designer has the following tools that you use to analyze sources, design target Schemas, and build source-to-target mappings:

- **Source Analyzer**: Import or create source definitions.
- **Target Designer:** Import or create target definitions.
- **Transformation Developer:** Develop transformations to use in mappings.
  You can also develop user-defined functions to use in expressions.
- **Mapplet Designer**: Create sets of transformations to use in mappings.
- **Mapping Designer:** Create mappings that the Integration Service uses to Extract, transform, and load data.

By using Designer we can prepare source definition, target definition and mapping. All this metadata will be stored in Repository by using Repository Service.



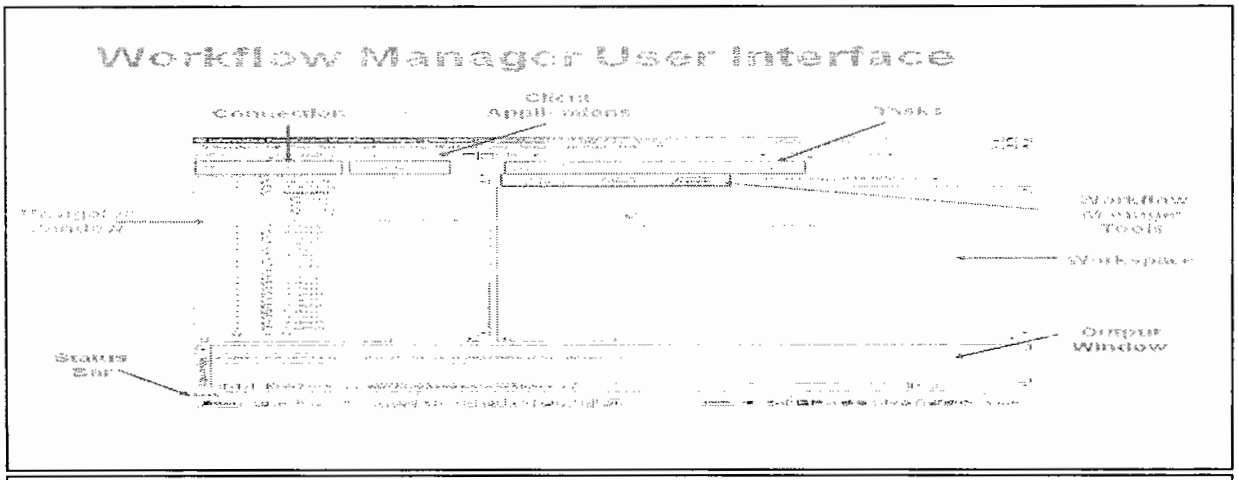


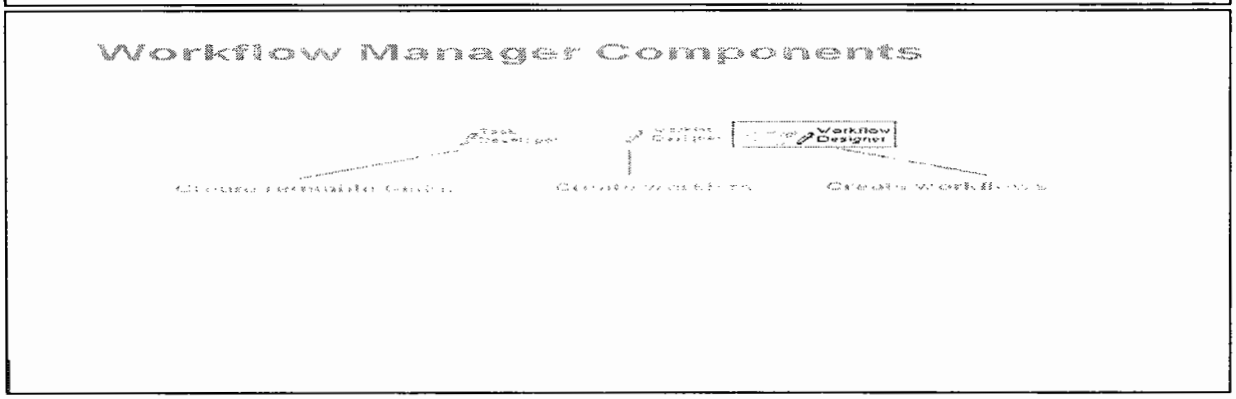
2.Workflow Manager :

Use the Workflow Manager to create, schedule, and run workflows. A workflow is a set of instructions that describes how and when to run tasks related to extracting, transforming, and loading data.
The Workflow Manager has the following tools to help us develop a workflow:

- **Task Developer:** Create tasks we want to accomplish in the workflow.
- **Work let Designer**: Create a worklet in the Worklet Designer. A worklet is an object that groups a set of tasks. A worklet is similar to a workflow, but without scheduling information. We can nest worklets inside a workflow.
- **Workflow Designer:** Create a workflow by connecting tasks with links in the Workflow Designer. You can also create tasks in the Workflow Designer as you develop the workflow.
  When we create a workflow in the Workflow Designer, we add tasks to the workflow. The Workflow Manager includes tasks, such as the Session task, the Command task, and the Email task so you can design a workflow. The Session task is based on a mapping we build in the Designer.

We then connect tasks with links to specify the order of execution for the tasks we created. Use conditional links and workflow variables to create branches in the workflow.
By using workflow manager we can create Sessions and Workflow. This information will be stored in to repository by using repository service.
When we run workflow then the request will go to integration service, integration service will connect to repository service to collect all metadata information from repository and convert that data in to actual data.





3. Workflow Monitor

Use the Workflow Monitor to monitor scheduled and running workflows for each Integration Service. We can view details about a workflow or task in Gantt chart view or Task view. We Can run, stop, abort, and resume workflows from the Workflow Monitor. We can view Sessions and workflow log events in the Workflow Monitor Log Viewer.

The Workflow Monitor displays workflows that have run at least once. The Workflow Monitor continuously receives information from the Integration Service and Repository Service. It also fetches information from the repository to display historic Information.

Work flow Monitor will produce Session Log and Status information. This information will be stored in to repository by using integration service.

4. Repository Manager

Use the Repository Manager to administer repositories. You can navigate through multiple folders and repositories, and complete the following tasks:

- **Manage users and groups:** Create, edit, and delete repository users and User groups. We can assign and revoke repository privileges and folder Permissions.
- **Perform folder functions: Create**, edit, copy, and delete folders. Work we perform in the Designer and Workflow Manager is stored in folders. If we want to share metadata, you can configure a folder to be shared.

- **View metadata:** Analyze sources, targets, mappings, and shortcut dependencies, search by keyword, and view the properties of repository Objects. We create repository objects using the Designer and Workflow Manager Client tools.
  We can view the following objects in the Navigator window of the Repository Manager:
- **Source definitions:** Definitions of database objects (tables, views, synonyms) or Files that provide source data.
- **Target definitions:** Definitions of database objects or files that contain the target data.
- **Mappings:** A set of source and target definitions along with transformations containing business logic that you build into the transformation. These are the instructions that the Integration Service uses to transform and move data.
- **Reusable transformations:** Transformations that we use in multiple mappings.
- **Mapplets:** A set of transformations that you use in multiple mappings.
- **Sessions and workflows:** Sessions and workflows store information about how and When the Integration Service moves data. A workflow is a set of instructions that Describes how and when to run tasks related to extracting, transforming, and loading Data. A session is a type of task that you can put in a workflow. Each session Corresponds to a single mapping.

Application Services:

Application Services: The application services is a group of services that provide powercenter server based functionality.

1. Repository service
2. Integration Service
3. Web services hub

Repository Service:

1. The repository service manages connection to powercenter repository from client applications.
2. The repository service is a multi threaded process that retrieves and inserts and updates metadata in to repository.
3. The repository service accepts the connection request from following powercentre applications.

Powecentre Client: 1) Use designer to store the mappings
2) Use workflow to store sessions and workflows
3) Use repository manager to organize and secure metadata by creating folders, users and groups.
4) Use workflow monitor to retrieve session logs created by integration service.
Integration Service: When you run the workflow the integration service retrieves mapping metadata from repository through repository service.
Web services hub: When you start web service hub it connects to the repository to access web enable workflows through repository service.

Integration Service:

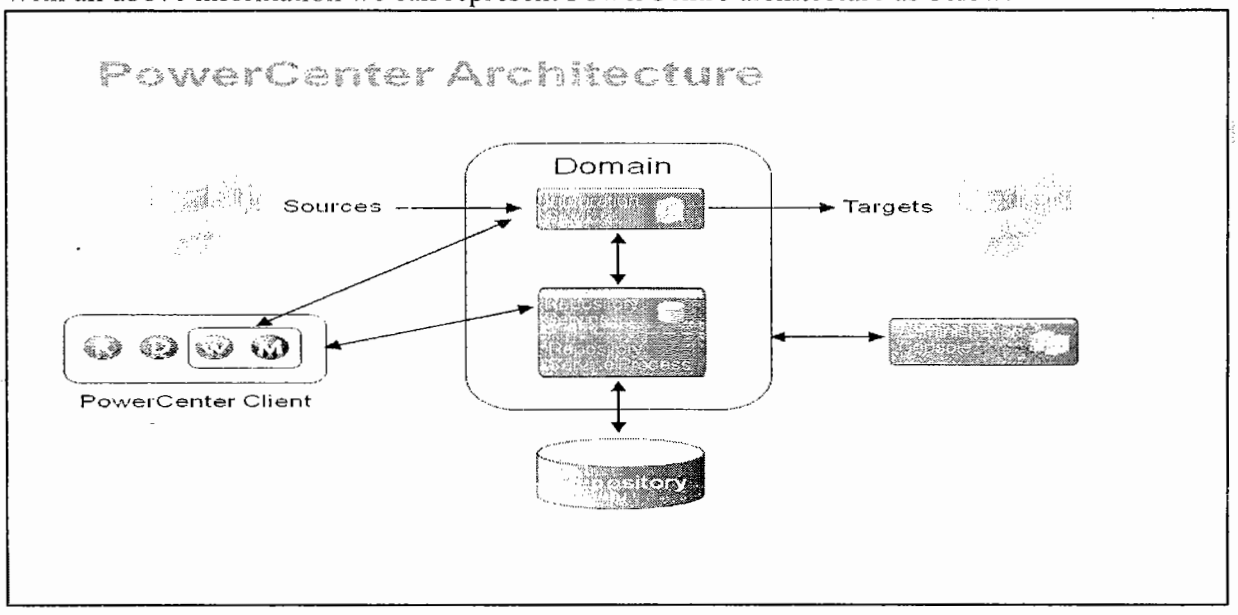
1) The integration service reads mapping and session information from repository related workflow.
2) The integration service extract data from mapping sources and store the data in temporary memory. Where it applies the transformation rules to process the data
3) The integration service loads the transformed data into mapping targets
4) The integration service connects to repository through repository service
5) The integration service write session log and workflow status to the repository.

Web service hub:

1. The web service hub is a web services gateway to external clients.
2. The web service client's access the repository service and integrated service through web service hub.
3. When you start the web services hub it connects the repository to access the web enabled workflows.

PowerCenter Architecture:

With all above information we can represent PowerCentre architecture as below.



Kinds of Metadata Stored in Repository:

1. *Source Definition*
2. *Target Definition*
3. *Mapping*
4. *Session*
5. *Workflows*
6. *Tasks*
7. *Session Log*
8. *Worklets*
9. *Mapplets*
10. *Users*

Steps to create Sample mapping:

1. *Create Folders*
2. *Create Source Definition (Create ODBC connection)*
3. *Create Target Definition (Create Target ODBC connection)*
4. *Create Mapping*
5. *Create Session*
6. *Create Relational Connections*
7. *Create Workflow*
8. *Run Work flow*

1. Create Folder:

1. *Start→All Programs→Informatica 8.6→Clients→Repository manager*
2. *Connect to repository by passing Username and Password.*
3. *Go to Folder menu click on Create and pass folder name.*
4. *Same folder name will be display in Designer, Workflow Manager, and Workflow Monitor.*

2. Create Source Definition:

1. *Start→All Programs→Informatica 8.6→Clients→Designer*
2. *Connect to repository by passing Username and Password.*
3. *Select your folder name and click on tools menu, select source analyzer.*
4. *Click on Sources→Import from database→Create User using ODBC connections→Select ODBC and pass username and password→Click on Connect→Select database→Select table→Click on Ok*
5. *Source definition will appear on screen.*

3. Create Target Definition:

1. *In Designer select Target analyzer.*

2. *Click on Targets→Import from database→Create User using ODBC connections→Select ODBC and pass username and password→click on Connect→Select database→Select table→Click on Ok*
3. *Target definition will appear on screen.*

4. Create Mapping:

1. *In designer select Mapping Designer*
2. *From navigator window drag and drop Sources and Targets.*
3. *Connect ports from Source to Target→Save mapping*

5. Create Session:

1. *Start→All Programs→Informatica 8.6→Clients→Workflow Manager*
2. *Connect to repository by passing Username and Password.*
3. *Select your folder name and click on tolls→Select Task developer*
4. *Select tasks→Click on create→Select Session→mention name of mapping→Click on ok*
5. *Select for which mapping we are refereeing this session.*
6. *In workflow manager click on connections→Click on relational→Create connections based on Teradta/Oracle→For source we need create one connection and for target we need to create one connection.*
7. *Right click on session→Click on Edit→Select mapping tab→For source pass source connection→For target pass target Connection.*
8. *Save it.*

6. Create Workflow:

1. *In work flow manager select workflow Designer*
2. *Click on Worflows menu→Click on Workflow→pass workflow name*
3. *From navigator window drag and drop Sessions.*
4. *Connect Start button and Session. (From Tasks select link task)*
5. *Save it.*

7. Run Workflow:

1. *In work flow manager→select workflows tab→Click on Start workflow*
2. *Then Workflow Monitor will open if not open Workflow monitor.*
3. *In work flow monitor you can check status of Workflow.*

Informatica Transformations

A transformation is a repository object that generates, modifies, or passes data. The Designer provides a set of transformations that perform specific functions. For example, an Aggregator transformation performs calculations on groups of data.
Transformations can be of two types:

Active Transformation

An active transformation can change the number of rows that pass through the transformation, change the transaction boundary, and can change the row type. For example, Filter, Transaction Control and Update Strategy are active transformations.

The key point is to note that Designer does not allow you to connect multiple active transformations or an active and a passive transformation to the same downstream transformation or transformation input group because the Integration Service may not be able to concatenate the rows passed by active transformations However, Sequence Generator transformation(SGT) is an exception to this rule. A SGT does not receive data. It generates unique numeric values. As a result, the Integration Service does not encounter problems concatenating rows passed by a SGT and an active transformation.

Passive Transformation.

A passive transformation does not change the number of rows that pass through it, maintains the transaction boundary, and maintains the row type.

The key point is to note that Designer allows you to connect multiple transformations to the same downstream transformation or transformation input group only if all transformations in the upstream branches are passive. The transformation that originates the branch can be active or passive.

Transformations can be **Connected or UnConnected** to the data flow.
ConnectedTransformation

Connected transformation is connected to other transformations or directly to target table in the mapping
Eg; All transformations

Unconnected Transformation

An unconnected transformation is not connected to other transformations in the mapping. It is called within another transformation, and returns a value to that transformation. Eg: Lookup, Stored Procedure transformations.

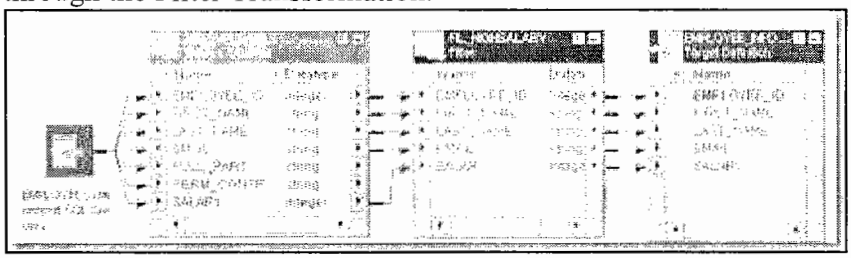
- Expression Transformation
- Filter Transformation
- Rank Transformation
- Router Transformation
- Aggregator Transformation
- Joiner Transformation
- Lookup Transformation

- Normalizer Transformation
- Sequence Generator Transformation
- Sorter Transformation
- Source Qualifier Transformation
- Stored Procedure Transformation
- Transaction Control Transaction
- Union Transformation
- Update Strategy Transformation
- XML Source Qualifier Transformation

FILTER Transformation

- Active and connected transformation.

We can filter rows in a mapping with the Filter transformation. We pass all the rows from a source transformation through the Filter transformation, and then enter a Filter condition for the transformation. All ports in a Filter transformation are input/output and only rows that meet the condition pass through the Filter Transformation.



Example: to filter records where SAL>2000

- Import the source table EMP in Shared folder. If it is already there, then don't Import.
- In shared folder, create the target table Filter_Example. Keep all fields as in EMP table.
- Create the necessary shortcuts in the folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping -> Create -> Give mapping name. Ex: m_filter_example
4. Drag EMP from source in mapping.
5. Click Transformation -> Create -> Select Filter from list. Give name and click Create. Now click done.
6. Pass ports from SQ_EMP to Filter Transformation.
7. Edit Filter Transformation. Go to Properties Tab
8. Click the Value section of the Filter condition, and then click the Open button.
9. The Expression Editor appears.
10. Enter the filter condition you want to apply.
11. Click Validate to check the syntax of the conditions you entered.
12. Click OK -> Click Apply -> Click Ok.
13. Now connect the ports from Filter to target table.
14. Click Mapping -> Validate
15. Repository -> Save

Create Session and Workflow as described earlier. Run the workflow and see the data in target table.

How to filter out rows with null values?

To filter out rows containing null values or spaces, use the ISNULL and IS_SPACES Functions to test the value of the port. For example, if we want to filter out rows that Contain NULLs in the FIRST_NAME port, use the following condition:

IIF (ISNULL (FIRST_NAME), FALSE, TRUE)

This condition states that if the FIRST_NAME port is NULL, the return value is FALSE and the row should be discarded. Otherwise, the row passes through to the next Transformation.

Performance tuning:

Filter transformation is used to filter off unwanted fields based on conditions we Specify.

1. Use filter transformation as close to source as possible so that unwanted data gets Eliminated sooner.
2. If elimination of unwanted data can be done by source qualifier instead of filter,Then eliminate them at Source Qualifier itself.
3. Use conditional filters and keep the filter condition simple, involving TRUE/FALSE or 1/0

Expression Transformation

- Passive and connected transformation.

Use the Expression transformation to calculate values in a single row before we write to the target. For example, we might need to adjust employee salaries, concatenate first and last names, or convert strings to numbers.

Use the Expression transformation to perform any non-aggregate calculations.

Example: Addition, Subtraction, Multiplication, Division, Concat, Uppercase conversion, lowercase conversion etc.

We can also use the Expression transformation to test conditional statements before we output the results to target tables or other transformations. Example: IF, Then, Decode

There are 3 types of ports in Expression Transformation:

- Input
- Output
- Variable: Used to store any temporary calculation.

Calculating Values :

To use the Expression transformation to calculate values for a single row, we must include the following ports:

- Input or input/output ports for each value used in the calculation: For example: To calculate Total Salary, we need salary and commission.
- **Output port for the expression:** We enter one expression for each output port. The return value for the output port needs to match the return value of the expression.

We can enter multiple expressions in a single Expression transformation. We can create any number of output ports in the transformation.

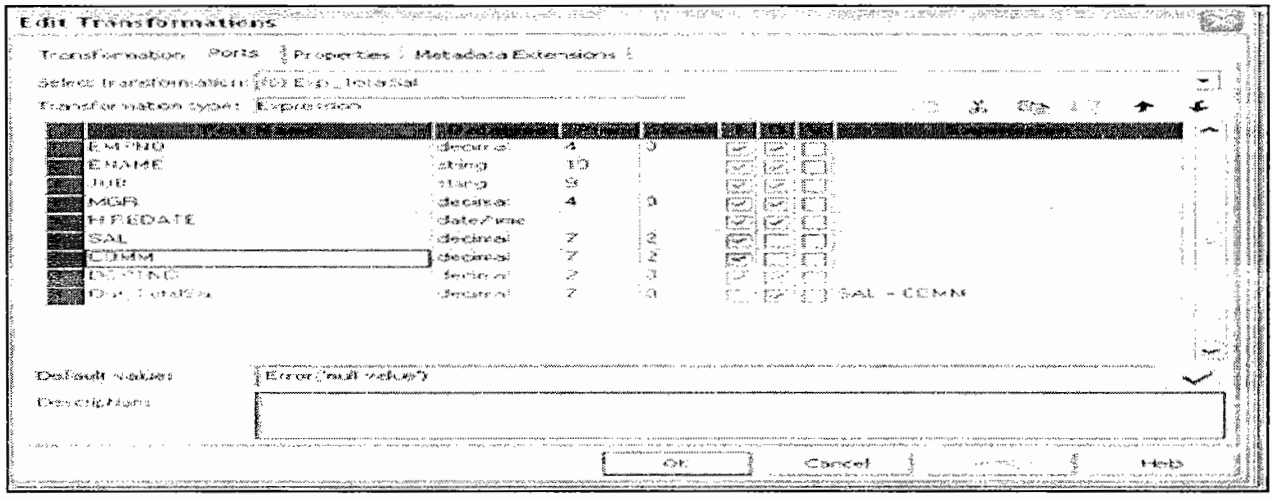
Example: Calculating Total Salary of an Employee

- Import the source table EMP in Shared folder. If it is already there, then don't import.
- In shared folder, create the target table Emp_Total_SAL. Keep all ports as in EMP table except Sal and Comm in target table. Add Total_SAL port to store the calculation.
- Create the necessary shortcuts in the folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping -> Create -> Give mapping name. Ex: m_totalsal
4. Drag EMP from source in mapping.
5. Click Transformation -> Create -> Select Expression from list. Give name and click Create. Now click done.
6. Link ports from SQ_EMP to Expression Transformation.
7. Edit Expression Transformation. As we do not want Sal and Comm in target, remove check from output port for both columns.
8. Now create a new port out_Total_SAL. Make it as output port only.
9. Click the small button that appears in the Expression section of the dialog box and enter the expression in the Expression Editor.
10. Enter expression SAL + COMM. You can select SAL and COMM from Ports tab in expression editor.
11. Check the expression syntax by clicking Validate.
12. Click OK -> Click Apply -> Click Ok.
13. Now connect the ports from Expression to target table.
14. Click Mapping -> Validate
15. Repository -> Save

Create Session and Workflow as described earlier. Run the workflow and see the data in target table.



As COMM is null, Total_SAL will be null in most cases. Now open your mapping and expression transformation. Select COMM port, In Default Value give 0. Now apply changes. Validate Mapping and Save.

Refresh the session and validate workflow again. Run the workflow and see the result again.

Performance tuning :

Expression transformation is used to perform simple calculations and also to do Source lookups.

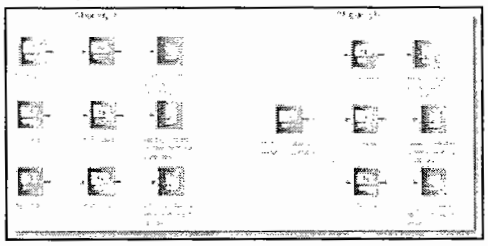
1. Use operators instead of functions.
2. Minimize the usage of string functions.
3. If we use a complex expression multiple times in the expression transformer, then Make that expression as a variable. Then we need to use only this variable for all computations.

ROUTER TRANSFORMATION

- Active and connected transformation.

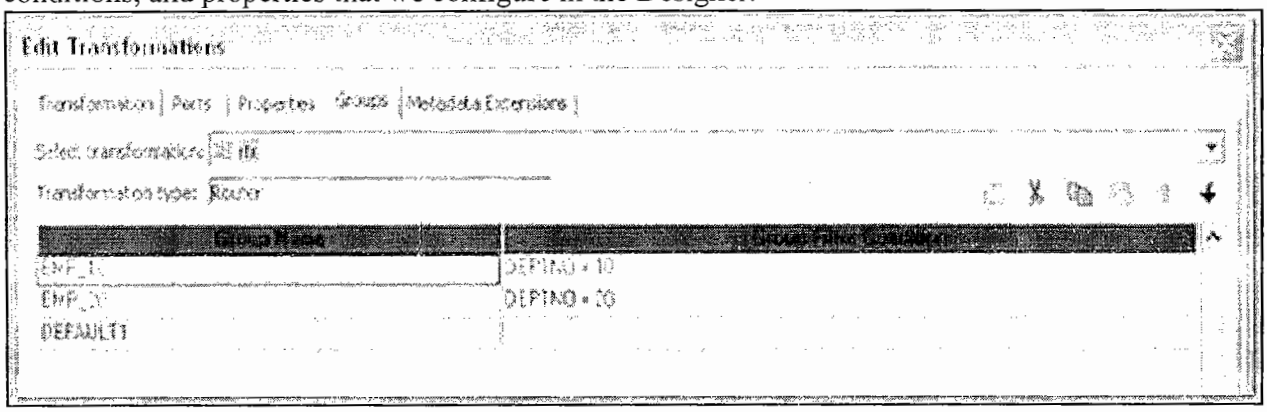
A Router transformation is similar to a Filter transformation because both transformations allow you to use a condition to test data. A Filter transformation tests data for one condition and drops the rows of data that do not meet the Condition. However, a Router transformation tests data for one or more conditions And gives you the option to route rows of data that do not meet any of the conditions to a default output group.

Example: If we want to keep employees of France, India, US in 3 different tables, then we can use 3 Filter transformations or 1 Router transformation.



Mapping A uses three Filter transformations while Mapping B produces the same result with one Router transformation.

A Router transformation consists of input and output groups, input and output ports, group filter conditions, and properties that we configure in the Designer.



Working with Groups

A Router transformation has the following types of groups:

- **Input:** The Group that gets the input ports.
- **Output:** User Defined Groups and Default Group. We cannot modify or delete Output ports or their properties.

User-Defined Groups: We create a user-defined group to test a condition based on incoming data. A user-defined group consists of output ports and a group filter Condition. We can create and edit user-defined groups on the Groups tab with the Designer. Create one user-defined group for each condition that we want to specify.

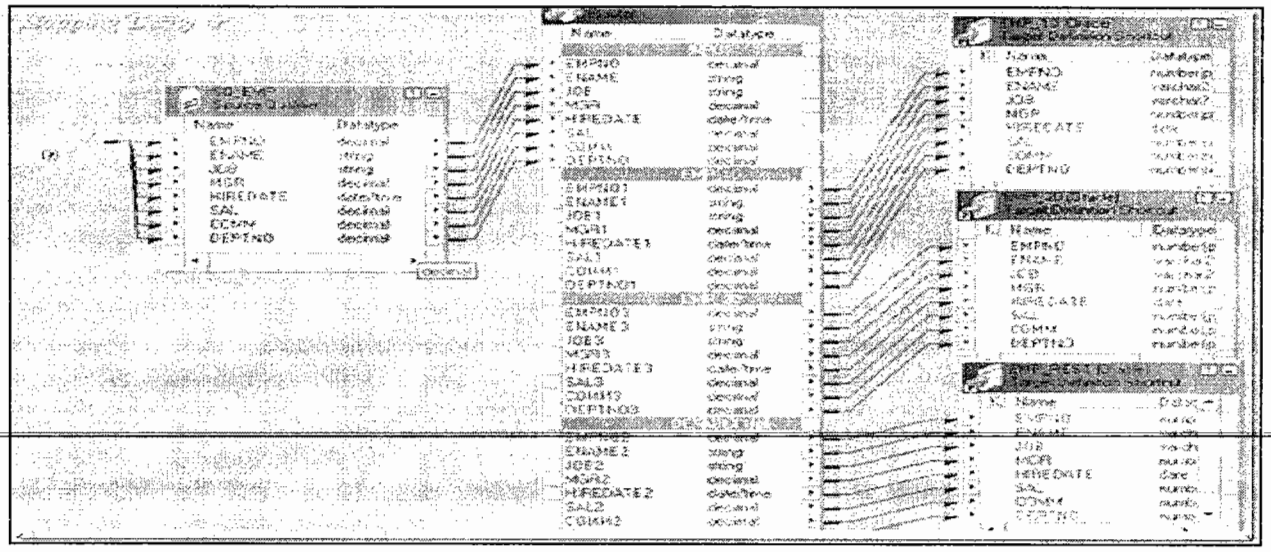
The Default Group: The Designer creates the default group after we create one new user-defined group. The Designer does not allow us to edit or delete the default group. This group does not have a group filter condition associated with it. If all of the conditions evaluate to FALSE, the IS passes the row to the default group.

Example: Filtering employees of Department 10 to EMP_10, Department 20 to EMP_20 and rest to EMP_REST

- Source is EMP Table.
- Create 3 target tables EMP_10, EMP_20 and EMP_REST in shared folder. Structure should be same as EMP table.
- Create the shortcuts in your folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_router_example
4. Drag EMP from source in mapping.
5. Click Transformation -> Create -> Select Router from list. Give name and Click Create. Now click done.
6. Pass ports from SQ_EMP to Router Transformation.
7. Edit Router Transformation. Go to Groups Tab
8. Click the Groups tab, and then click the Add button to create a user-defined Group. The default group is created automatically..
9. Click the Group Filter Condition field to open the Expression Editor.
10. Enter a group filter condition. Ex: DEPTNO=10
11. Click Validate to check the syntax of the conditions you entered.



12. Create another group for EMP_20. Condition: DEPTNO=20
13. The rest of the records not matching the above two conditions will be passed to DEFAULT group. See sample mapping
14. Click OK -> Click Apply -> Click Ok.
15. Now connect the ports from router to target tables.
16. Click Mapping -> Validate
17. Repository -> Save

- Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.
- Make sure to give connection information for all 3 target tables.

Difference between Router and Filter :

We cannot pass rejected data forward in filter but we can pass it in router. Rejected data is in Default Group of router.

SORTER TRANSFORMATION

- Connected and Active Transformation
- The Sorter transformation allows us to sort data.
- We can sort data in ascending or descending order according to a specified sort key.
- We can also configure the Sorter transformation for case-sensitive sorting, and specify whether the output rows should be distinct.

When we create a Sorter transformation in a mapping, we specify one or more ports as a sort key and configure each sort key port to sort in ascending or descending order. We also configure sort criteria the Power Center Server applies to all sort key ports and the system resources it allocates to perform the sort operation.

The Sorter transformation contains only input/output ports. All data passing through the Sorter transformation is sorted according to a sort key. The sort key is one or more ports that we want to use as the sort criteria.

Sorter Transformation Properties

1. Sorter Cache Size:

The Power Center Server uses the Sorter Cache Size property to determine the maximum amount of memory it can allocate to perform the sort operation. The Power Center Server passes all incoming data into the Sorter transformation Before it performs the sort operation.

- We can specify any amount between 1 MB and 4 GB for the Sorter cache size.
- If it cannot allocate enough memory, the Power Center Server fails the Session.
- For best performance, configure Sorter cache size with a value less than or equal to the amount of available physical RAM on the Power Center Server machine.
- Informatica recommends allocating at least 8 MB of physical memory to sort data using the Sorter transformation.

2. Case Sensitive:

The Case Sensitive property determines whether the Power Center Server considers case when sorting data. When we enable the Case Sensitive property, the Power Center Server sorts uppercase characters higher than lowercase characters.

3. Work Directory

Directory Power Center Server uses to create temporary files while it sorts data.

4. Distinct:
Check this option if we want to remove duplicates. Sorter will sort data according to all the ports when it is selected.

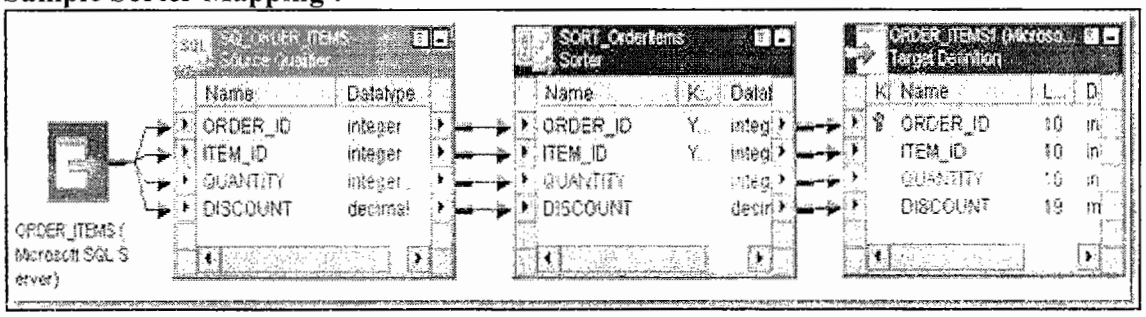
Example: Sorting data of EMP by ENAME

- Source is EMP table.
- Create a target table EMP_SORTER_EXAMPLE in target designer. Structure same as EMP table.
- Create the shortcuts in your folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_sorter_example
4. Drag EMP from source in mapping.
5. Click Transformation -> Create -> Select Sorter from list. Give name and click Create. Now click done.
6. Pass ports from SQ_EMP to Sorter Transformation.
7. Edit Sorter Transformation. Go to Ports Tab
8. Select ENAME as sort key. CHECK mark on KEY in front of ENAME.
9. Click Properties Tab and Select Properties as needed.
10. Click Apply -> Ok.
11. Drag target table now.
12. Connect the output ports from Sorter to target table.
13. Click Mapping -> Validate
14. Repository -> Save
    - Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.

Sample Sorter Mapping :



Performance Tuning:

Sorter transformation is used to sort the input data.

1. While using the sorter transformation, configure sorter cache size to be larger than the input data size.
2. Configure the sorter cache size setting to be larger than the input data size while Using sorter transformation.
3. At the sorter transformation, use hash auto keys partitioning or hash user keys Partitioning.

RANK TRANSFROAMTION

- Active and connected transformation

The Rank transformation allows us to select only the top or bottom rank of data. It Allows us to select a group of top or bottom values, not just one value.
During the session, the Power Center Server caches input data until it can perform The rank calculations.

Rank Transformation Properties :

- Cache Directory where cache will be made.
- Top/Bottom Rank as per need
- Number of Ranks Ex: 1, 2 or any number
- Case Sensitive Comparison can be checked if needed
- Rank Data Cache Size can be set
- Rank Index Cache Size can be set
-

Ports in a Rank Transformation :

Ports	Number Required	Description
I	1 Minimum	Port to receive data from another transformation.
O	1 Minimum	Port we want to pass to other transformation.
V	not needed	can use to store values or calculations to use in an expression.
R	Only 1	Rank port. Rank is calculated according to it. The Rank port is an input/output port. We must link the Rank port to another transformation. Example: Total Salary

Rank Index

The Designer automatically creates a RANKINDEX port for each Rank transformation. The Power Center Server uses the Rank Index port to store the ranking position for Each row in a group.

For example, if we create a Rank transformation that ranks the top five salaried employees, the rank index numbers the employees from 1 to 5.

- The RANKINDEX is an output port only.
- We can pass the rank index to another transformation in the mapping or directly to a target.
- We cannot delete or edit it.

Defining Groups

Rank transformation allows us to group information. For example: If we want to select the top 3 salaried employees of each Department, we can define a group for Department.

- By defining groups, we create one set of ranked rows for each group.
- We define a group in Ports tab. Click the Group By for needed port.
- We cannot Group By on port which is also Rank Port.

1) **Example: Finding Top 5 Salaried Employees**

- EMP will be source table.
- Create a target table EMP_RANK_EXAMPLE in target designer. Structure should be same as EMP table. Just add one more port Rank_Index to store RANK INDEX.
- Create the shortcuts in your folder.

Creating Mapping:

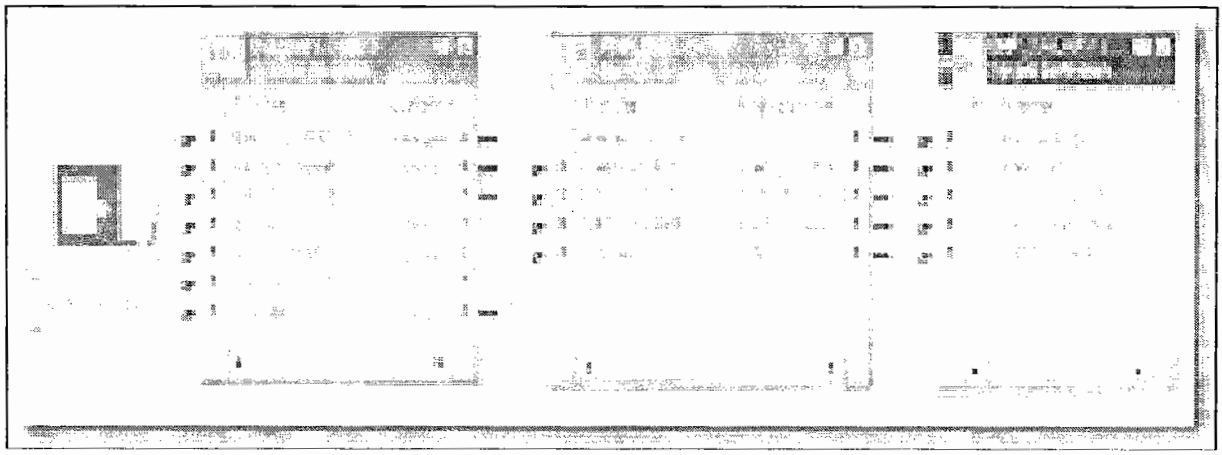
1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_rank_example
4. Drag EMP from source in mapping.
5. Create an EXPRESSION transformation to calculate TOTAL_SAL.
6. Click Transformation -> Create -> Select RANK from list. Give name and click Create. Now click done.
7. Pass ports from Expression to Rank Transformation.
8. Edit Rank Transformation. Go to Ports Tab
9. Select TOTAL_SAL as rank port. Check R type in front of TOTAL_SAL.
10. Click Properties Tab and Select Properties as needed.
11. Top in Top/Bottom and Number of Ranks as 5.
12. Click Apply -> Ok.
13. Drag target table now.
14. Connect the output ports from Rank to target table.
15. Click Mapping -> Validate
16. Repository -> Save

- Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.
- Make sure to give connection information for all tables.

2) Example: Finding Top 2 Salaried Employees for every DEPARTMENT

- Open the mapping made above. Edit Rank Transformation.
- Go to Ports Tab. Select Group By for DEPTNO.
- Go to Properties tab. Set Number of Ranks as 2.
- Click Apply -> Ok.
- Mapping -> Validate and Repository Save.

Refresh the session by double clicking. Save the changed and run workflow to see the new result.



RANK CACHE

Sample Rank Mapping

When the Power Center Server runs a session with a Rank transformation, it compares an input row with rows in the data cache. If the input row out-ranks a Stored row, the Power Center Server replaces the stored row with the input row.

Example: Power Center caches the first 5 rows if we are finding top 5 salaried Employees. When 6th row is read, it compares it with 5 rows in cache and places it in Cache is needed.

1) **RANK INDEX CACHE:**

The index cache holds group information from the group by ports. If we are Using Group By on DEPTNO, then this cache stores values 10, 20, 30 etc.

- All Group By Columns are in RANK INDEX CACHE. Ex. DEPTNO

2) **RANK DATA CACHE:**

It holds row data until the Power Center Server completes the ranking and is Generally larger than the index cache. To reduce the data cache size, connect Only the necessary input/output ports to subsequent transformations.

- All Variable ports if there, Rank Port, All ports going out from RANK Transformations are stored in RANK DATA CACHE.
- **Example:** All ports except DEPTNO In our mapping example.

TRANSACTION CONTROL TRANSFORAMTION

Power Center lets you control commit and roll back transactions based on a set of rows that pass through a Transaction Control transformation. A transaction is the set of rows bound by commit or roll back rows. You can define a transaction based on a varying number of input rows. You might want to define transactions based on a group of rows ordered on a common key, such as employee ID or order entry date.

In Power Center, you define transaction control at the following levels:

- **Within a mapping.** Within a mapping, you use the Transaction Control transformation to define a transaction. You define transactions using an expression in a Transaction Control transformation. Based on the return value of the expression, you can choose to commit, roll back, or continue without any transaction changes.

- **Within a session.** When you configure a session, you configure it for user-defined commit. You can choose to commit or roll back a transaction if the Integration Service fails to transform or write any row to the target.

When you run the session, the Integration Service evaluates the expression for each row that enters the transformation. When it evaluates a commit row, it commits all rows in the transaction to the target or targets. When the Integration Service evaluates a roll back row, it rolls back all rows in the transaction from the target or targets. If the mapping has a flat file target you can generate an output file each time the Integration Service starts a new transaction. You can dynamically name each target flat file.

Properties Tab

On the Properties tab, you can configure the following properties:

- Transaction control expression
- Tracing level

Enter the transaction control expression in the Transaction Control Condition field. The transaction control expression uses the IIF function to test each row against the condition. Use the following syntax for the expression:

IIF (condition, value1, value2)

The expression contains values that represent actions the Integration Service performs based on the return value of the condition. The Integration Service evaluates the condition on a row-by-row basis. The return value determines whether the Integration Service commits, rolls back, or makes no transaction changes to the row.

When the Integration Service issues a commit or roll back based on the return value of the expression, it begins a new transaction. Use the following built-in variables in the Expression Editor when you create a transaction control expression:

- **TC_CONTINUE_TRANSACTION.** The Integration Service does not perform any transaction change for this row. This is the default value of the expression.
- **TC_COMMIT_BEFORE.** The Integration Service commits the transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
- **TC_COMMIT_AFTER.** The Integration Service writes the current row to the target, commits the transaction, and begins a new transaction. The current row is in the committed transaction.
- **TC_ROLLBACK_BEFORE.** The Integration Service rolls back the current transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
- **TC_ROLLBACK_AFTER.** The Integration Service writes the current row to the target, rolls back the transaction, and begins a new transaction. The current row is in the rolled back transaction.

If the transaction control expression evaluates to a value other than commit, roll back, or continue, the Integration Service fails the session.

Mapping Guidelines and Validation

Use the following rules and guidelines when you create a mapping with a **Transaction Control transformation:**

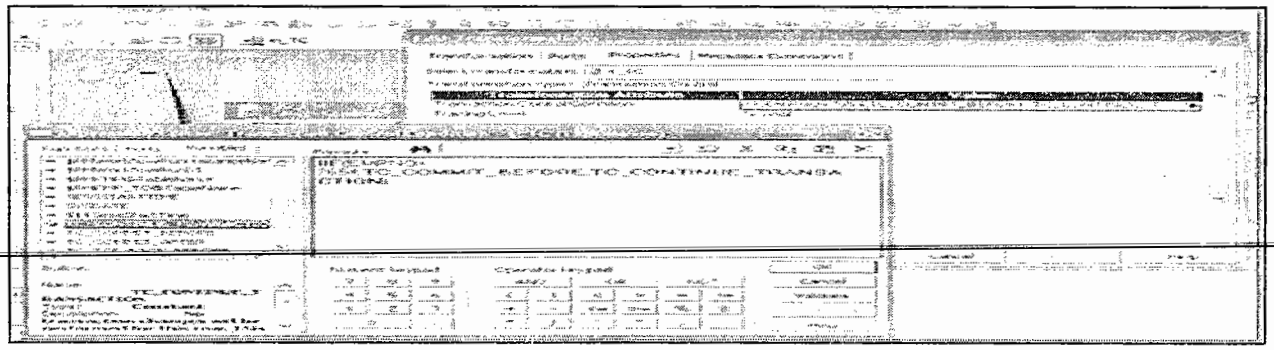
- If the mapping includes an XML target, and you choose to append or create a new document on commit, the input groups must receive data from the same transaction control point.
- Transaction Control transformations connected to any target other than relational, XML, or dynamic MQSeries targets are ineffective for those targets.
- You must connect each target instance to a Transaction Control transformation.
- You can connect multiple targets to a single Transaction Control transformation.
- You can connect only one effective Transaction Control transformation to a target.
- You cannot place a Transaction Control transformation in a pipeline branch that starts with a Sequence Generator transformation.
- If you use a dynamic Lookup transformation and a Transaction Control transformation in the same mapping, a rolled-back transaction might result in unsynchronized target data.
- A Transaction Control transformation may be effective for one target and ineffective for another target. If each target is connected to an effective Transaction Control transformation, the mapping is valid.
- Either all targets or none of the targets in the mapping should be connected to an effective Transaction Control transformation.

Example to Transaction Control:

Step 1: Creating a Transaction Control Transformation.

- In the Mapping Designer, click Transformation > Create. Select the Transaction Control transformation.
- Enter a name for the transformation.[The naming convention for Transaction Control transformations is TC_TransformationName].
- Enter a description for the transformation.
- Click Create.
- Click Done.
- Drag the ports into the transformation.
- Open the Edit Transformations dialog box, and select the Ports tab.

Select the Properties tab. Enter the transaction control expression that defines the commit and roll back behavior.



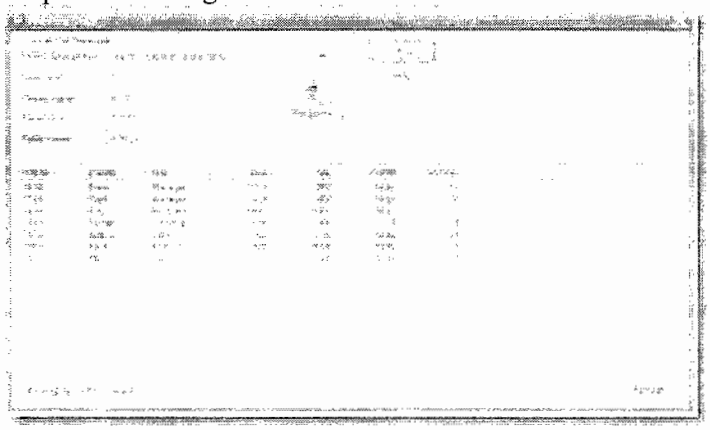
Go to the Properties tab and click on the down arrow to get in to the expression editor window. Later go to the Variables tab and Type IIF(EMpno=7654,) select the below things from the built in functions.

IIF (EMPNO=7654,TC_COMMIT_BEFORE,TC_CONTINUE_TRANSACTION)

- Connect all the columns from the transformation to the target table and save the mapping.
- Select the Metadata Extensions tab. Create or edit metadata extensions for the Transaction Control transformation.
- Click OK.

Step 2: Create the task and the work flow.

Step 3: Preview the output in the target table.



SOURCE QUALIFIER TRANSFORMATION

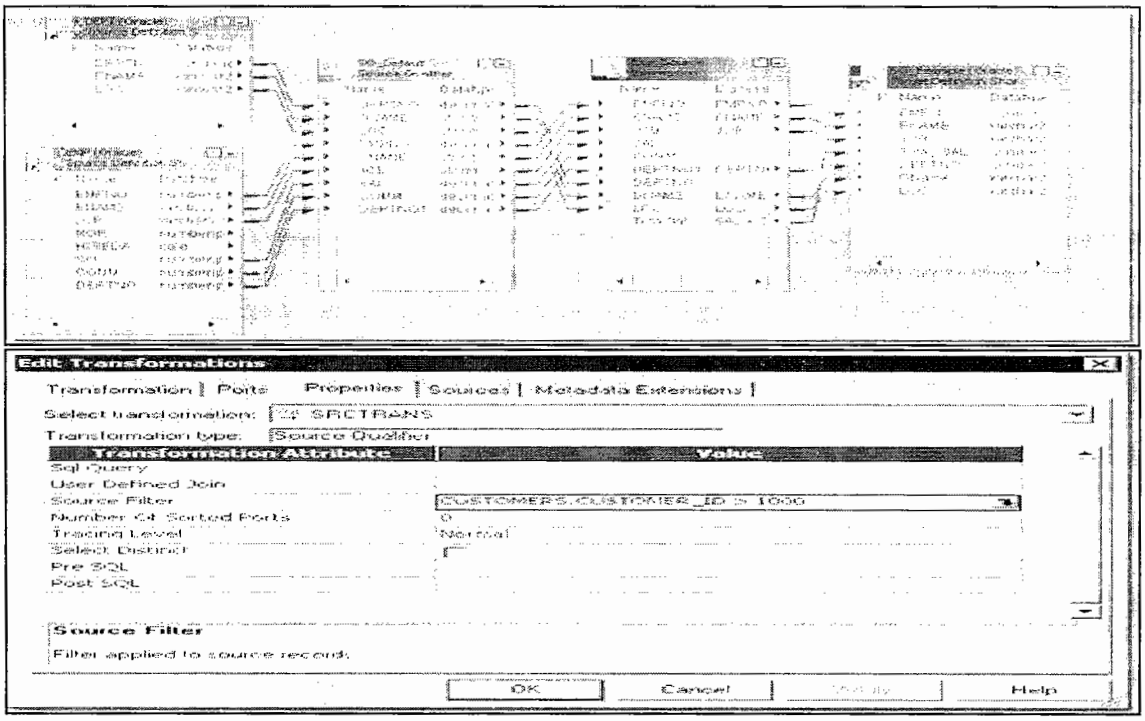
- Active and Connected Transformation.
- The Source Qualifier transformation represents the rows that the Power Center Server reads when it runs a session.
- It is only transformation that is not reusable.
- Default transformation except in case of XML or COBOL files.

Tasks performed by Source Qualifier:

- Join data originating from the same source database: We can join two or more tables with primary key-foreign key relationships by linking the sources to one Source Qualifier transformation.
- Filter rows when the Power Center Server reads source data: If we Include a filter condition, the Power Center Server adds a WHERE clause to the Default query.
- Specify an outer join rather than the default inner join: If we include a User-defined join, the Power Center Server replaces the join information Specified by the metadata in the SQL query.
- Specify sorted ports: If we specify a number for sorted ports, the
- Power Center Server adds an ORDER BY clause to the default SQL query.
- Select only distinct values from the source: If we choose Select Distinct,the Power Center Server adds a SELECT DISTINCT statement to the default SQL query.

- Create a custom query to issue a special SELECT statement for the Power Center Server to read source data: For example, you might use a Custom query to perform aggregate calculations. The entire above are possible in Properties Tab of Source Qualifier t/f.

SAMPLE MAPPING TO BE MADE:



- Source will be EMP and DEPT tables.
- Create target table as showed in Picture above.
- Create shortcuts in your folder as needed.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_SQ_example
4. Drag EMP, DEPT, Target.
5. Right Click SQ_EMP and Select Delete from the mapping.
6. Right Click SQ_DEPT and Select Delete from the mapping.
7. Click Transformation -> Create -> Select Source Qualifier from List -> Give Name -> Click Create
8. Select EMP and DEPT both. Click OK.
9. Link all as shown in above picture.
10. Edit SQ -> Properties Tab -> Open User defined Join -> Give Join condition EMP.DEPTNO=DEPT.DEPTNO. Click Apply -> OK
11. Mapping -> Validate
12. Repository -> Save

- Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.
- Make sure to give connection information for all tables.

SQ PROPERTIES TAB

1) SOURCE FILTER:

We can enter a source filter to reduce the number of rows the Power Center Server queries.

Note: When we enter a source filter in the session properties, we override the customized SQL query in the Source Qualifier transformation.

Steps:

1. In the Mapping Designer, open a Source Qualifier transformation.
2. Select the Properties tab.
3. Click the Open button in the Source Filter field.
4. In the SQL Editor Dialog box, enter the filter. Example: EMP.SAL)2000
5. Click OK.

Validate the mapping. Save it. Now refresh session and save the changes. Now run the workflow and see output.

2) NUMBER OF SORTED PORTS:

When we use sorted ports, the Power Center Server adds the ports to the ORDER BY clause in the default query.

By default it is 0. If we change it to 1, then the data will be sorted by column that is at the top in SQ. Example: DEPTNO in above figure.

- If we want to sort as per ENAME, move ENAME to top.
- If we change it to 2, then data will be sorted by top two columns.

Steps:

1. In the Mapping Designer, open a Source Qualifier transformation.
2. Select the Properties tab.
3. Enter any number instead of zero for Number of Sorted ports.
4. Click Apply -> Click OK.

Validate the mapping. Save it. Now refresh session and save the changes. Now run the workflow and see output.

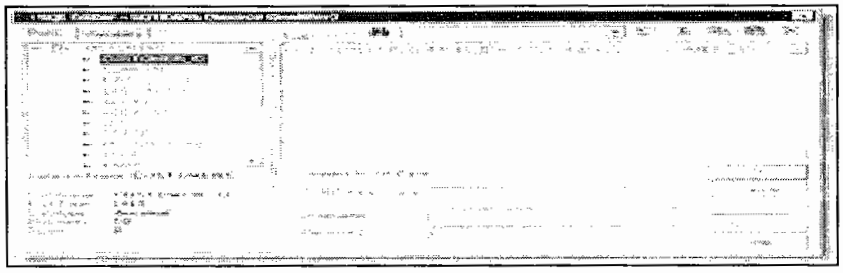
3) SELECT DISTINCT:

If we want the Power Center Server to select unique values from a source, we can use the Select Distinct option.

- Just check the option in Properties tab to enable it.

4) PRE and POST SQL Commands

- The Power Center Server runs pre-session SQL commands against the source database before it reads the source.
- It runs post-session SQL commands against the source database after it writes to the target.
- Use a semi-colon (;) to separate multiple statements.



5) USER DEFINED JOINS

Entering a user-defined join is similar to entering a custom SQL query. However, we only enter the contents of the WHERE clause, not the entire query.

- We can specify equi join, left outer join and right outer join only. We Cannot specify full outer join. To use full outer join, we need to write SQL Query.

Steps:

1. Open the Source Qualifier transformation, and click the Properties tab.
2. Click the Open button in the User Defined Join field. The SQL Editor Dialog Box appears.
3. Enter the syntax for the join.
4. Click OK -> Again Ok.

Validate the mapping. Save it. Now refresh session and save the changes. Now run the workflow and see output.

Join Type	Syntax
Equi Join	DEPT.DEPTNO=EMP.DEPTNO
Left Outer Join	{EMP LEFT OUTER JOIN DEPT ON DEPT.DEPTNO=EMP.DEPTNO}
Right Outer Join	{EMP RIGHT OUTER JOIN DEPT ON DEPT.DEPTNO=EMP.DEPTNO}

6) SQL QUERY

For relational sources, the Power Center Server generates a query for each Source Qualifier transformation when it runs a session. The default query is a SELECT statement for each source column used in the mapping. In other words, the Power Center Server reads only the columns that are connected to another Transformation.

In mapping above, we are passing only SAL and DEPTNO from SQ_EMP to Aggregator transformation. Default query generated will be:

- SELECT EMP.SAL, EMP.DEPTNO FROM EMP

Viewing the Default Query

1. Open the Source Qualifier transformation, and click the Properties tab.
2. Open SQL Query. The SQL Editor displays.
3. Click Generate SQL.
4. The SQL Editor displays the default query the Power Center Server uses to Select source data.

5. Click Cancel to exit.

Note: If we do not cancel the SQL query, the Power Center Server overrides the default query with the custom SQL query.
We can enter an SQL statement supported by our source database. Before entering the query, connect all the input and output ports we want to use in the mapping.
Example: As in our case, we can't use full outer join in user defined join, **we can write SQL query for FULL**

OUTER JOIN:
SELECT DEPT.DEPTNO, DEPT.DNAME, DEPT.LOC, EMP.EMPNO, EMP.ENAME, EMP.JOB, EMP.SAL, EMP.COMM, EMP.DEPTNO FROM EMP FULL OUTER JOIN DEPT ON DEPT.DEPTNO=EMP.DEPTNO WHERE SAL>2000

- We also added WHERE clause. We can enter more conditions and write More complex SQL.

We can write any query. We can join as many tables in one query as Required if all are in same database. It is very handy and used in most of the projects.

Important Points:

- When creating a custom SQL query, the SELECT statement must list the port names in the order in which they appear in the transformation.

Example: DEPTNO is top column; DNAME is second in our SQ mapping.
So when we write SQL Query, SELECT statement have name DNAME first, DNAME second and so on. SELECT DEPT.DEPTNO, DEPT.DNAME

- Once we have written a custom query like above, then this query will Always be used to fetch data from database. In our example, we used WHERE SAL>2000. Now if we use Source Filter and give condition SAL) 1000 or any other, then it will not work. Informatica will always use the custom query only.
- Make sure to test the query in database first before using it in SQL Query. If query is not running in database, then it won't work in Informatica too.
- Also always connect to the database and validate the SQL in SQL query editor.

STORED PROCEDUTER TARNSFORMATION

- Passive Transformation
- Connected and Unconnected Transformation
- Stored procedures are stored and run within the database.

A Stored Procedure transformation is an important tool for populating and Maintaining databases. Database administrators create stored procedures to Automate tasks that are too complicated for standard SQL statements.

Use of Stored Procedure in mapping:

- Check the status of a target database before loading data into it.
- Determine if enough space exists in a database.
- Perform a specialized calculation.
- Drop and recreate indexes. Mostly used for this in projects.

Data Passes Between IS and Stored Procedure One of the most useful features of stored procedures is the ability to send data to the stored procedure, and receive data from the stored procedure. There are three types of data that pass between the Integration Service and the stored procedure:

Input/output parameters: Parameters we give as input and the parameters returned from Stored Procedure.
Return values: Value returned by Stored Procedure if any.
Status codes: Status codes provide error handling for the IS during a workflow. The stored procedure issues a status code that notifies whether or not the stored procedure completed successfully. We cannot see this value. The IS uses it to determine whether to continue running the session or stop. Specifying when the Stored Procedure Runs
Normal: The stored procedure runs where the transformation exists in the mapping on a row-by-row basis. We pass some input to procedure and it returns some calculated values. Connected stored procedures run only in normal mode.
Pre-load of the Source: Before the session retrieves data from the source, the stored procedure runs. This is useful for verifying the existence of tables or performing joins of data in a temporary table.
Post-load of the Source: After the session retrieves data from the source, the stored procedure runs. This is useful for removing temporary tables.
Pre-load of the Target: Before the session sends data to the target, the stored procedure runs. This is useful for dropping indexes or disabling constraints.
Post-load of the Target: After the session sends data to the target, the stored procedure runs. This is useful for re-creating indexes on the database.

Using a Stored Procedure in a Mapping :

1. Create the stored procedure in the database.
2. Import or create the Stored Procedure transformation.
3. Determine whether to use the transformation as connected or unconnected.
4. If connected, map the appropriate input and output ports.
5. If unconnected, either configure the stored procedure to run pre- or post-session, or configure it to run from an expression in another transformation.
6. Configure the session.

Stored Procedures:
Connect to Source database and create the stored procedures given below:

```
CREATE OR REPLACE procedure sp_agg (in_deptno in number, max_sal out number,
min_sal out number, avg_sal out number, sum_sal out number)
As
Begin
select max(Sal),min(sal),avg(sal),sum(sal) into max_sal,min_sal,avg_sal,sum_sal
from emp where deptno=in_deptno group by deptno;
End;
/
```

```
CREATE OR REPLACE procedure sp_unconn_l_value(in_deptno in number, max_sal out
number)
As
Begin
Select max(Sal) into max_sal from EMP where deptno=in_deptno;
End;
/
```

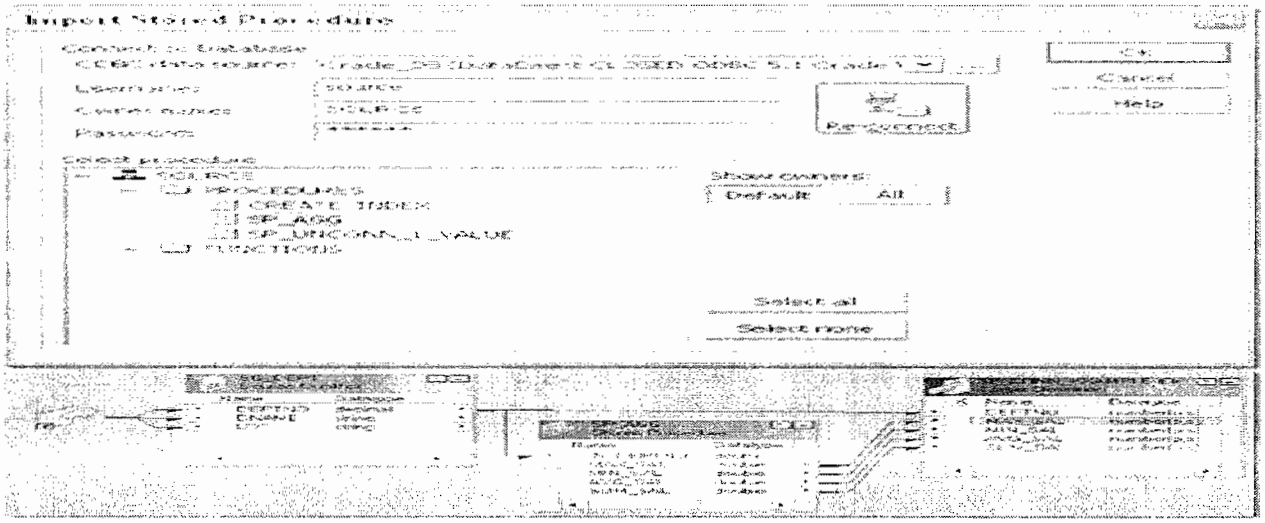
1. Connected Stored Procedure T/F

Example: To give input as DEPTNO from DEPT table and find the MAX, MIN, AVG and SUM of SAL from EMP table.

- DEPT will be source table. Create a target table SP_CONN_EXAMPLE with fields DEPTNO, MAX_SAL, MIN_SAL, AVG_SAL & SUM_SAL.
- Write Stored Procedure in Database first and Create shortcuts as needed.

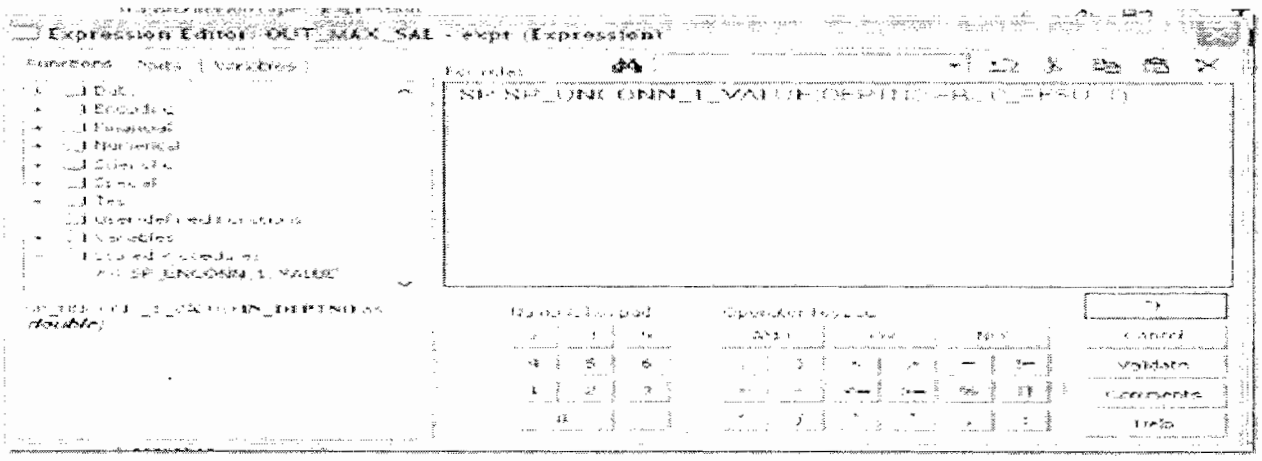
Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_SP_CONN_EXAMPLE
4. Drag DEPT and Target table.
5. Transformation -> Import Stored Procedure -> Give Database Connection -> Connect -> Select the procedure sp_agg from the list.



6. Drag DEPTNO from SQ_DEPT to the stored procedure input port and also to DEPTNO port of target.
7. Connect the ports from procedure to target as shown below:
8. Mapping -> Validate
9. Repository -> Save

- Create Session and then workflow.
- Give connection information for all tables.
- Give connection information for Stored Procedure also.
- Run workflow and see the result in table.



NORMALIZER TANSFORMATION

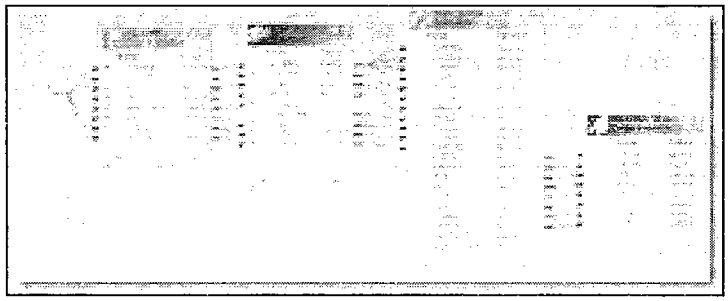
- Active and Connected Transformation.
- The Normalizer transformation normalizes records from COBOL and relational sources, allowing us to organize the data.
- Use a Normalizer transformation instead of the Source Qualifier transformation when we normalize a COBOL source.
- We can also use the Normalizer transformation with relational sources to create multiple rows from a single row of data.

Example 1: To create 4 records of every employee in EMP table.

- EMP will be source table.
- Create target table Normalizer_Multiple_Records. Structure same as EMP and datatype of HIREDATE as VARCHAR2.
- Create shortcuts as necessary.

Creating Mapping :

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_ Normalizer_Multiple_Records
4. Drag EMP and Target table.
5. Transformation->Create->Select Expression-> Give name, Click create, done.
6. Pass all ports from SQ_EMP to Expression transformation.
7. Transformation-> Create-> Select Normalizer-> Give name, create & done.
8. Try dragging ports from Expression to Normalizer. Not Possible.
9. Edit Normalizer and Normalizer Tab. Add columns. Columns equal to columns in EMP table and datatype also same.
10. Normalizer doesn't have DATETIME datatype. So convert HIREDATE to char in expression t/f. Create output port out_hdate and do the conversion.
11. Connect ports from Expression to Normalizer.
12. Edit Normalizer and Normalizer Tab. As EMPNO identifies source records and we want 4 records of every employee, give OCCUR for EMPNO as 4.



13. Click Apply and then OK.
14. Add link as shown in mapping below:
15. Mapping -> Validate
16. Repository -> Save

- Make session and workflow.
- Give connection information for source and target table.
- Run workflow and see result.

Example 2: To break rows into columns
Source:

Roll_Number	Name	ENG	HINDI	MATHS
100	Amit	78	76	90
101	Rahul	76	78	87
102	Jessie	65	98	79

Target :

Roll_Number	Name	Marks
100	Amit	78
100	Amit	76
100	Amit	90
101	Rahul	76
101	Rahul	78
101	Rahul	87
102	Jessie	65
102	Jessie	98
102	Jessie	79

- Make source as a flat file. Import it and create target table.
- Create Mapping as before. In Normalizer tab, create only 3 ports Roll_Number, Name and Marks as there are 3 columns in target table.
- Also as we have 3 marks in source, give Occurs as 3 for Marks in Normalizer tab.
- Connect accordingly and connect to target.
- Validate and Save

- Make Session and workflow and Run it. Give Source File Directory and Source File name for source flat file in source properties in mapping tab of session.
- See the result.

SEQUENCE GENERATOR TRANSFORMATION

- Passive and Connected Transformation.
- The Sequence Generator transformation generates numeric values.
- Use the Sequence Generator to create unique primary key values, replace missing primary keys, or cycle through a sequential range of numbers.

We use it to generate Surrogate Key in DWH environment mostly. When we want to Maintain history, then we need a key other than Primary Key to uniquely identify the record. So we create a Sequence 1,2,3,4 and so on. We use this sequence as the key. Example: If EMPNO is the key, we can keep only one record in target and can't maintain history. So we use Surrogate key as Primary key and not EMPNO.

Sequence Generator Ports :

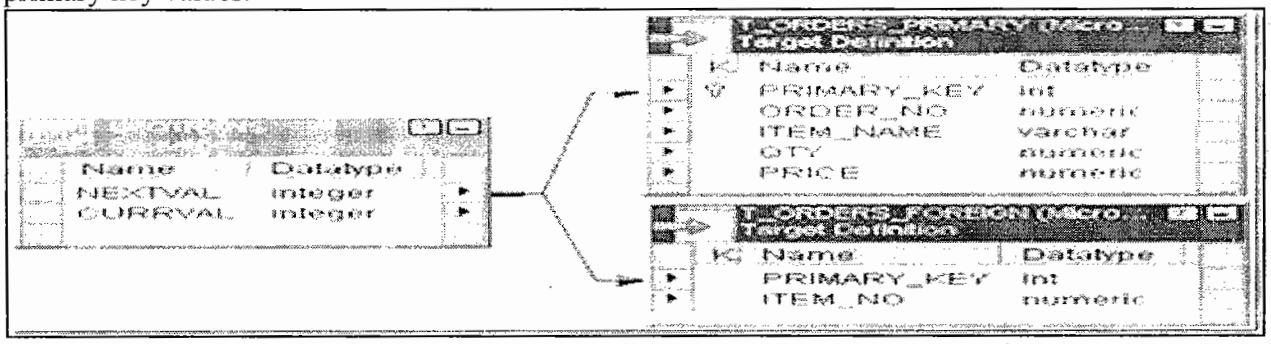
The Sequence Generator transformation provides two output ports: NEXTVAL and CURRVAL.

- We cannot edit or delete these ports.
- Likewise, we cannot add ports to the transformation.

NEXTVAL:

Use the NEXTVAL port to generate sequence numbers by connecting it to a Transformation or target.

For example, we might connect NEXTVAL to two target tables in a mapping to generate unique primary key values.



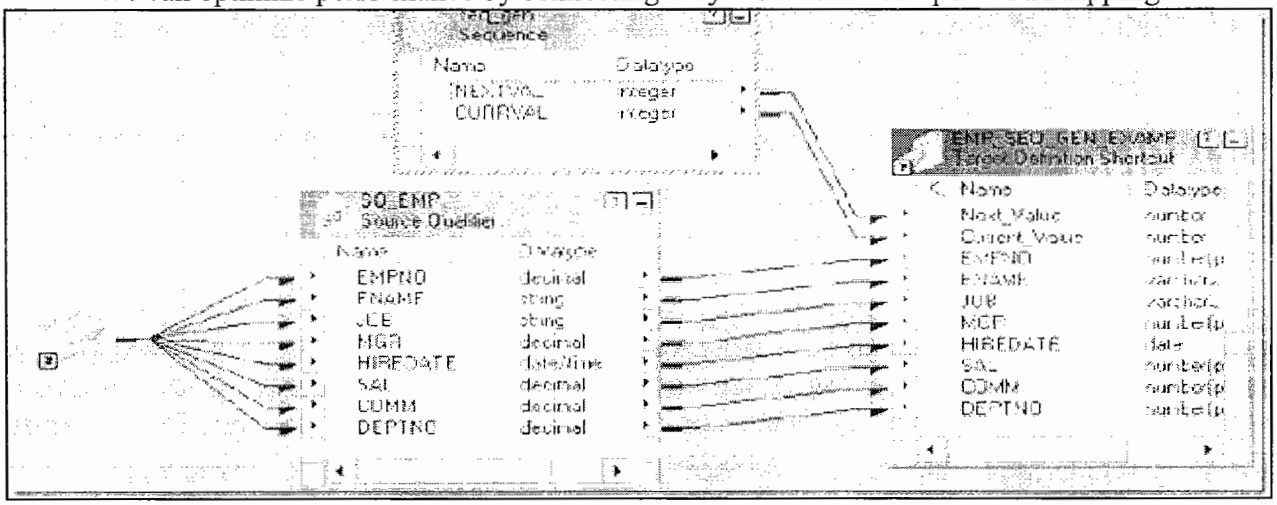
Sequence in Table 1 will be generated first. When table 1 has been loaded, only then Sequence for table 2 will be generated.

CURRVAL:

CURRVAL is NEXTVAL plus the Increment By value.

- We typically only connect the CURRVAL port when the NEXTVAL port is Already connected to a downstream transformation.
- If we connect the CURRVAL port without connecting the NEXTVAL port, the Integration Service passes a constant value for each row.
- when we connect the CURRVAL port in a Sequence Generator Transformation, the Integration Service processes one row in each block.

- We can optimize performance by connecting only the NEXTVAL port in a Mapping.



Example: To use Sequence Generator transformation

- EMP will be source.
- Create a target EMP_SEQ_GEN_EXAMPLE in shared folder. Structure same as EMP. Add two more ports NEXT_VALUE and CURR_VALUE to the target table.
- Create shortcuts as needed.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_seq_gen_example
4. Drag EMP and Target table.
5. Connect all ports from SQ_EMP to target table.
6. Transformation -> Create -> Select Sequence Generator for list -> Create -> Done
7. Connect NEXT_VAL and CURR_VAL from Sequence Generator to target.
8. Validate Mapping
9. Repository -> Save
   - Create Session and then workflow.
   - Give connection information for all tables.
   - Run workflow and see the result in table.

Sequence Generator Properties:

Setting	Required/Optional	Description
Start Value	Required	Start value of the generated sequence that we want IS to use if we use Cycle option. Default is 0.
Increment By	Required	Difference between two consecutive values from the NEXTVAL port.
End Value	Optional	Maximum value the Integration Service generates.

Current Value	Optional	First value in the sequence.If cycle option used, the value must be greater than or equal to the start value and less the end value.
Cycle	Optional	If selected, the Integration Service cycles through the sequence range. Ex: Start Value:1 End Value 10 Sequence will be from 1-10 and again start from 1.
Reset	Optional	By default, last value of sequence during session is saved to repository. Next time the sequence is started from the valued saved. If selected, the Integration Service generates values based on the original current value for each session.

Points to Ponder:

- If Current value is 1 and end value 10, no cycle option. There are 17 records in source. In this case session will fail.
- If we connect just CURR_VAL only, the value will be same for all records.
- If Current value is 1 and end value 10, cycle option there. Start value is 0.
- There are 17 records in source. Sequence: 1 2 – 10. 0 1 2 3 –
- To make above sequence as 1-10 1-20, give Start Value as 1. Start value is used along with Cycle option only.
- If Current value is 1 and end value 10, cycle option there. Start value is 1.
- There are 17 records in source. Session runs. 1-10 1-7. 7 will be saved in repository. If we run session again, sequence will start from 8.
- Use reset option if you want to start sequence from CURR_VAL every time.

AGGREGATOR TRANSFORMATION

- Connected and Active Transformation
- The Aggregator transformation allows us to perform aggregate calculations, such as averages and sums.
- Aggregator transformation allows us to perform calculations on groups.

Components of the Aggregator Transformation

1. Aggregate expression
2. Group by port
3. Sorted Input
4. Aggregate cache

1) Aggregate Expressions

- Entered in an output port.
- Can include non-aggregate expressions and conditional clauses.

The transformation language includes the following aggregate functions:

- AVG, COUNT, MAX, MIN, SUM
- FIRST, LAST
- MEDIAN, PERCENTILE, STDDEV, VARIANCE

Single Level Aggregate Function: MAX(SAL)

Nested Aggregate Function: MAX(COUNT(ITEM))

Nested Aggregate Functions

- In Aggregator transformation, there can be multiple single level functions or multiple nested functions.
- An Aggregator transformation cannot have both types of functions together.
- MAX(COUNT(ITEM)) is correct.
- MIN(MAX(COUNT(ITEM))) is not correct. It can also include one aggregate function nested within another aggregate function

Conditional Clauses

We can use conditional clauses in the aggregate expression to reduce the number of rows used in the aggregation. The conditional clause can be any clause that evaluates to TRUE or FALSE.

- SUM(COMMISSION, COMMISSION > QUOTA)

Non-Aggregate Functions

We can also use non-aggregate functions in the aggregate expression.

- IIF(MAX(QUANTITY) > 0, MAX(QUANTITY), 0))

2) Group By Ports

- Indicates how to create groups.
- When grouping data, the Aggregator transformation outputs the last row of each group unless otherwise specified.

  The Aggregator transformation allows us to define groups for aggregations, rather than performing the aggregation across all input data.

  For example, we can find Maximum Salary for every Department.
- In Aggregator Transformation, Open Ports tab and select Group By as needed.

3) Using Sorted Input

- Use to improve session performance.
- To use sorted input, we must pass data to the Aggregator transformation sorted by group by port, in ascending or descending order.
- When we use this option, we tell Aggregator that data coming to it is already sorted.
- We check the Sorted Input Option in Properties Tab of the transformation.
- If the option is checked but we are not passing sorted data to the transformation, then the session fails.

4) Aggregator Caches

- The Power Center Server stores data in the aggregate cache until it completes Aggregate calculations.
- It stores group values in an index cache and row data in the data cache. If the Power Center Server requires more space, it stores overflow values in cache files.

Note: The Power Center Server uses memory to process an Aggregator transformation with sorted ports. It does not use cache memory. We do not need to configure cache memory for Aggregator transformations that use sorted ports.

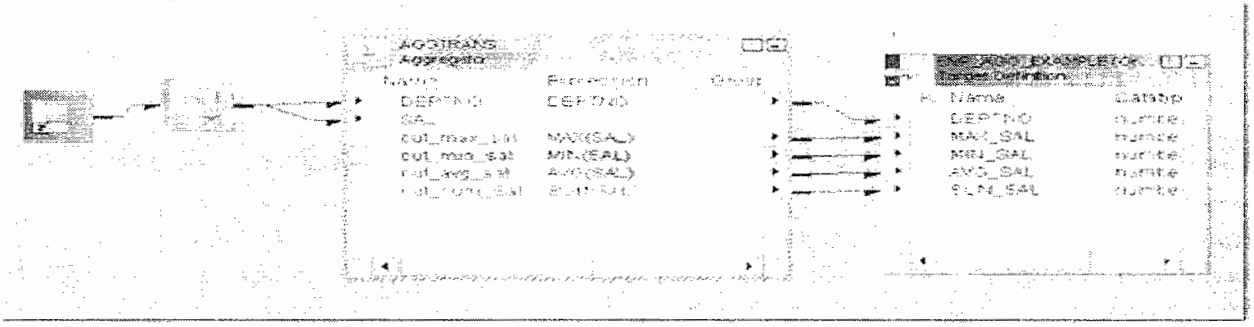
1) Aggregator Index Cache:
The index cache holds group information from the group by ports. If we are using Group By on DEPTNO, then this cache stores values 10, 20, 30 etc.

- All Group By Columns are in AGGREGATOR INDEX CACHE. Ex. DEPTNO

2) Aggregator Data Cache:
DATA CACHE is generally larger than the AGGREGATOR INDEX CACHE.

Columns in Data Cache:

- Variable ports if any
- Non group by input/output ports.
- Non group by input ports used in non-aggregate output expression.
- Port containing aggregate function



1) **Example: To calculate MAX, MIN, AVG and SUM of salary of EMP table.**

- EMP will be source table.
- Create a target table EMP_AGG_EXAMPLE in target designer. Table should contain DEPTNO, MAX_SAL, MIN_SAL, AVG_SAL and SUM_SAL
- Create the shortcuts in your folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_agg_example
4. Drag EMP from source in mapping.
5. Click Transformation -> Create -> Select AGGREGATOR from list. Give name and click Create. Now click done.
6. Pass SAL and DEPTNO only from SQ_EMP to AGGREGATOR Transformation.
7. Edit AGGREGATOR Transformation. Go to Ports Tab
8. Create 4 output ports: OUT_MAX_SAL, OUT_MIN_SAL, OUT_AVG_SAL, OUT_SUM_SAL
9. Open Expression Editor one by one for all output ports and give the calculations. Ex: MAX(SAL), MIN(SAL), AVG(SAL),SUM(SAL)

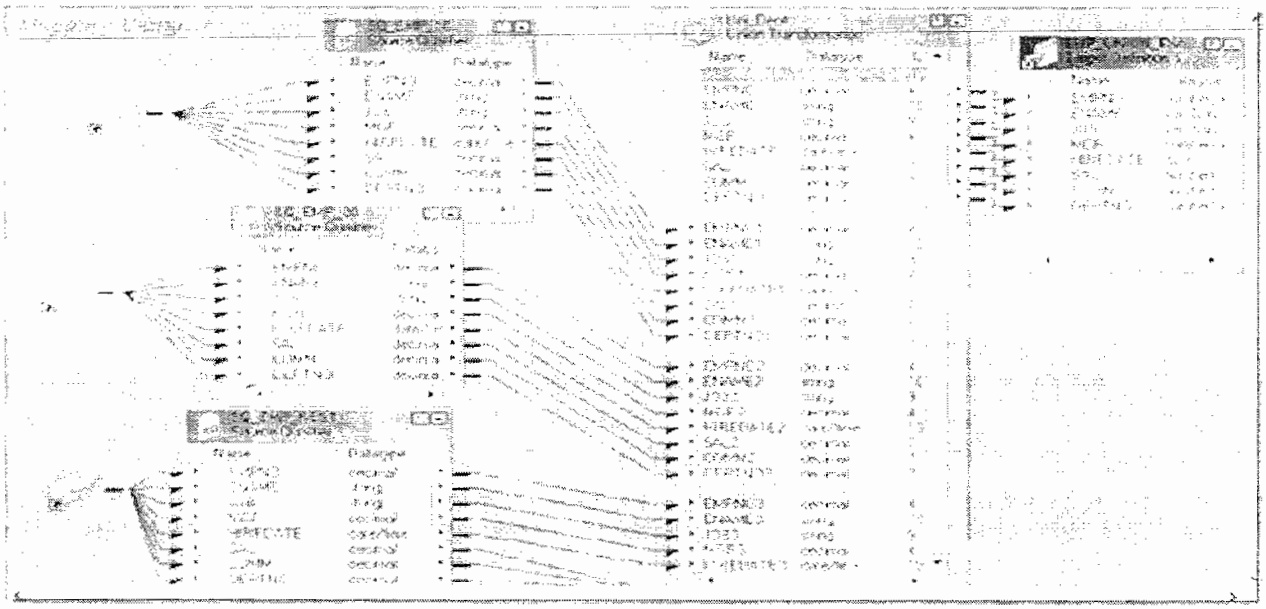
10. Click Apply -> Ok.
11. Drag target table now.
12. Connect the output ports from Rank to target table.
13. Click Mapping -> Validate
14. Repository -> Save

- Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.
- Make sure to give connection information for all tables.

UNION TRANSFORMATION

- Active and Connected transformation.
  Union transformation is a multiple input group transformation that you can use to merge data from multiple pipelines or pipeline branches into one pipeline branch. It merges data from multiple sources similar to the **UNION ALL** SQL statement to Combine the results from two or more SQL statements.
  Union Transformation Rules and Guidelines
- we can create multiple input groups, but only one output group.
- we can connect heterogeneous sources to a Union transformation.
- all input groups and the output group must have matching ports. The Precision, data type, and scale must be identical across all groups.
- The Union transformation does not remove duplicate rows. To remove Duplicate rows, we must add another transformation such as a Router or Filter Transformation.
- we cannot use a Sequence Generator or Update Strategy transformation upstream from a Union transformation.
  Union Transformation Components
  When we configure a Union transformation, define the following components:
  Transformation tab: We can rename the transformation and add a description.
  Properties tab: We can specify the tracing level.
  Groups tab: We can create and delete input groups. The Designer displays groups we create on the Ports tab.
  Group Ports tab: We can create and delete ports for the input groups. The Designer displays ports we create on the Ports tab.

  We cannot modify the Ports, Initialization Properties, Metadata Extensions, or Port Attribute Definitions tabs in a Union transformation.
  Create input groups on the Groups tab, and create ports on the Group Ports tab. We can create one or more input groups on the Groups tab. The Designer creates one output group by default. We cannot edit or delete the default output group.
  Example: to combine data of tables EMP_10, EMP_20 and EMP_REST
- Import tables EMP_10, EMP_20 and EMP_REST in shared folder in Sources.
- Create a target table EMP_UNION_EXAMPLE in target designer. Structure should be same EMP table.
- Create the shortcuts in your folder.



Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_union_example
4. Drag EMP_10, EMP_20 and EMP_REST from source in mapping.
5. Click Transformation -> Create -> Select Union from list. Give name and click Create. Now click done.
6. Pass ports from SQ_EMP_10 to Union Transformation.
7. Edit Union Transformation. Go to Groups Tab
8. One group will be already there as we dragged ports from SQ_DEPT_10 to Union Transformation.
9. As we have 3 source tables, we 3 need 3 input groups. Click add button to add 2 more groups. See Sample Mapping
10. We can also modify ports in ports tab.
11. Click Apply -> Ok.
12. Drag target table now.
13. Connect the output ports from Union to target table.
14. Click Mapping -> Validate
15. Repository -> Save

- Create Session and Workflow as described earlier. Run the Workflow and see the data in target table.
- Make sure to give connection information for all 3 source Tables.

JOINER TRANSFORMATION

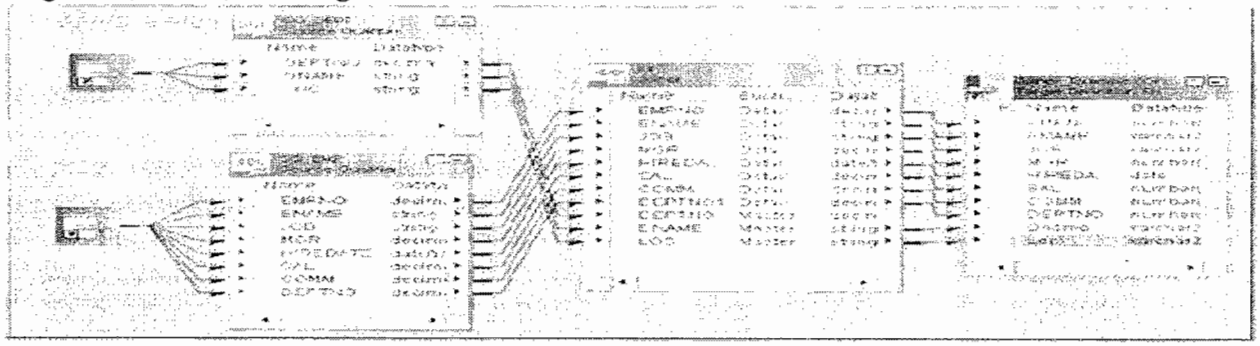
- Connected and Active Transformation
- Used to join source data from two related heterogeneous sources residing in Different locations or file systems. Or, we can join data from the same source.
- If we need to join 3 tables, then we need 2 Joiner Transformations.
- The Joiner transformation joins two sources with at least one matching port. The Joiner transformation uses a condition that matches one or more pairs of Ports between the two sources.

Example: To join EMP and DEPT tables.

- EMP and DEPT will be source table.
- Create a target table JOINER_EXAMPLE in target designer. Table should Contain all ports of EMP table plus DNAME and LOC as shown below.
- Create the shortcuts in your folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give mapping name. Ex: m_joiner_example
4. Drag EMP, DEPT, and Target. Create Joiner Transformation. Link as shown below.



5. Specify the join condition in Condition tab. See steps on next page.
6. Set Master in Ports tab. See steps on next page.
7. Mapping -> Validate
8. Repository -> Save.

- Create Session and Workflow as described earlier. Run the Work flow and see the data in target table.
- Make sure to give connection information for all tables.

JOIN CONDITION:

The join condition contains ports from both input sources that must match for the Power Center Server to join two rows.

Example: DEPTNO=DEPTNO1 in above.

1. Edit Joiner Transformation -> Condition Tab
2. Add condition

- We can add as many conditions as needed.
- Only = operator is allowed.

If we join Char and Varchar data types, the Power Center Server counts any spaces that pad Char values as part of the string. So if you try to join the following:
Char (40) = "abcd" and Varchar (40) = "abcd"

Then the Char value is "abcd" padded with 36 blank spaces, and the Power Center Server does not join the two fields because the Char field contains trailing spaces.

Note: The Joiner transformation does not match null values.

MASTER and DETAIL TABLES

In Joiner, one table is called as MASTER and other as DETAIL.

- MASTER table is always cached. We can make any table as MASTER.
- Edit Joiner Transformation -> Ports Tab -> Select M for Master table.

Table with less number of rows should be made MASTER to improve Performance.

Reason:

- When the Power Center Server processes a Joiner transformation, it reads rows from both sources concurrently and builds the index and data cache based on the master rows. So table with fewer rows will be read fast and cache can be made as table with more rows is still being read.
- The fewer unique rows in the master, the fewer iterations of the join comparison occur, which speeds the join process.

JOINER TRANSFORMATION PROPERTIES TAB

- **Case-Sensitive String Comparison:** If selected, the Power Center Server uses case-sensitive string comparisons when performing joins on string columns.
- **Cache Directory:** Specifies the directory used to cache master or detail rows and the index to these rows.
- **Join Type:** Specifies the type of join: Normal, Master Outer, Detail Outer, or Full Outer.

Tracing Level
Joiner Data Cache Size
Joiner Index Cache Size
Sorted Input

JOIN TYPES

In SQL, a join is a relational operator that combines data from multiple tables into a single result set. The Joiner transformation acts in much the same manner, except that tables can originate from different databases or flat files.

Types of Joins:

- Normal
- Master Outer
- Detail Outer
- Full Outer

Note: A normal or master outer join performs faster than a full outer or detail outer join.

Example: In EMP, we have employees with DEPTNO 10, 20, 30 and 50. In DEPT, we have DEPTNO 10, 20, 30 and 40. DEPT will be MASTER table as it has less rows.

Normal Join:

With a normal join, the Power Center Server discards all rows of data from the master and detail source that do not match, based on the condition.

- All employees of 10, 20 and 30 will be there as only they are matching.

Master Outer Join:

This join keeps all rows of data from the detail source and the matching rows from the master source. It discards the unmatched rows from the master source.

- All data of employees of 10, 20 and 30 will be there.
- There will be employees of DEPTNO 50 and corresponding DNAME and LOC Columns will be NULL.

Detail Outer Join:

This join keeps all rows of data from the master source and the matching rows from the detail source. It discards the unmatched rows from the detail source.

- All employees of 10, 20 and 30 will be there.
- There will be one record for DEPTNO 40 and corresponding data of EMP columns will be NULL.

Full Outer Join:

A full outer join keeps all rows of data from both the master and detail sources.

- All data of employees of 10, 20 and 30 will be there.
- There will be employees of DEPTNO 50 and corresponding DNAME and LOC Columns will be NULL.
- There will be one record for DEPTNO 40 and corresponding data of EMP Columns will be NULL.

USING SORTED INPUT

- Use to improve session performance.
- to use sorted input, we must pass data to the Joiner transformation sorted by the ports that are used in Join Condition.
- We check the Sorted Input Option in Properties Tab of the transformation.
- If the option is checked but we are not passing sorted data to the Transformation, then the session fails.
- We can use SORTER to sort data or Source Qualifier in case of Relational tables.

JOINER CACHES

Joiner always caches the MASTER table. We cannot disable caching. It builds Index cache and Data Cache based on MASTER table.

1) Joiner Index Cache:

- All Columns of MASTER table used in Join condition are in JOINER INDEX CACHE.
  · **Example:** DEPTNO in our mapping.

2) Joiner Data Cache:

- Master column not in join condition and used for output to other transformation or target table are in Data Cache.
  · **Example:** DNAME and LOC in our mapping example.

Performance Tuning:

- Perform joins in a database when possible.
- Join sorted data when possible.

- For a sorted Joiner transformation, designate as the master source the source with fewer duplicate key values.
- Joiner can't be used in following conditions:

1. Either input pipeline contains an Update Strategy transformation.
2. We connect a Sequence Generator transformation directly before the Joiner transformation.

UPDATED STRATEGY TRANSFORMATION

- Active and Connected Transformation

Till now, we have only inserted rows in our target tables. What if we want to update, delete or reject rows coming from source based on some condition?

Example: If Address of a CUSTOMER changes, we can update the old address or keep both old and new address. One row is for old and one for new. This way we maintain the historical data.

Update Strategy is used with Lookup Transformation. In DWH, we create a Lookup on target table to determine whether a row already exists or not. Then we insert, update, delete or reject the source record as per business need.

In Power Center, we set the update strategy at two different levels:

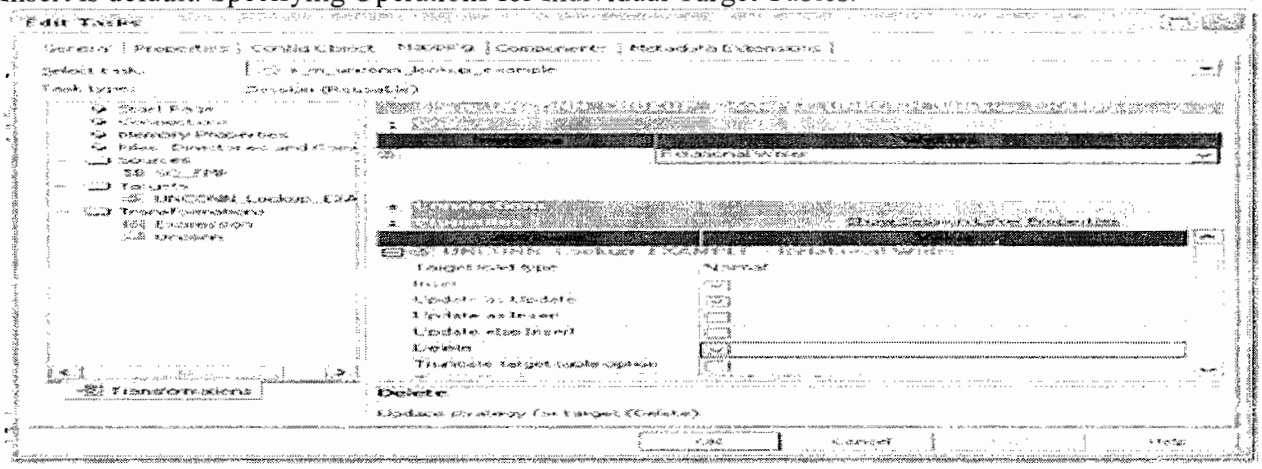
1. Within a session
2. Within a Mapping

1. Update Strategy within a session:

When we configure a session, we can instruct the IS to either treat all rows in the same way or use instructions coded into the session mapping to flag rows for different database operations.

Session Configuration:

Edit Session -> Properties -> Treat Source Rows as: (Insert, Update, Delete, and Data Driven). Insert is default. Specifying Operations for Individual Target Tables:



You can set the following update strategy options:

Insert: Select this option to insert a row into a target table.

Delete: Select this option to delete a row from a table.

Update: We have the following options in this situation:

- **Update as Update.** Update each row flagged for update if it exists in the target table.
- **Update as Insert.** Inset each row flagged for update.
- **Update else Insert.** Update the row if it exists. Otherwise, insert it.

Truncate table: Select this option to truncate the target table before loading data.

2. Flagging Rows within a Mapping

Within a mapping, we use the Update Strategy transformation to flag rows for insert, delete, update, or reject.

Operation	Constant	Numeric Value
INSERT	DD_INSERT	0
UPDATE	DD_UPDATE	1
DELETE	DD_DELETE	2
REJECT	DD_REJECT	3

Update Strategy Expressions:

Frequently, the update strategy expression uses the IIF or DECODE function from the transformation language to test each row to see if it meets a particular condition.

IIF((ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE)

Or

IIF((ENTRY_DATE > APPLY_DATE), 3, 2)

- The above expression is written in Properties Tab of Update Strategy T/f.
- DD means DATA DRIVEN

Forwarding Rejected Rows:

We can configure the Update Strategy transformation to either pass rejected rows to the next transformation or drop them.

Steps:

1. Create Update Strategy Transformation
2. Pass all ports needed to it.
3. Set the Expression in Properties Tab.
4. Connect to other transformations or target.

Performance tuning:

1. Use Update Strategy transformation as less as possible in the mapping.
2. Do not use update strategy transformation if we just want to insert into target table, instead use direct mapping, direct filtering etc.
3. For updating or deleting rows from the target table we can use Update Strategy transformation itself.

LOOKUP TANSFORMATION

- Passive Transformation
- Can be Connected or Unconnected. Dynamic lookup is connected.
- Use a Lookup transformation in a mapping to look up data in a flat file or a relational table, view, or synonym.
- We can import a lookup definition from any flat file or relational database to which both the PowerCenter Client and Server can connect.
- We can use multiple Lookup transformations in a mapping.

The Power Center Server queries the lookup source based on the lookup ports in the transformation. It compares Lookup transformation port values to lookup source column values based on the lookup condition. Pass the result of the lookup to other transformations and a target.

We can use the Lookup transformation to perform following:

- **Get a related value:** EMP has DEPTNO but DNAME is not there. We use Lookup to get DNAME from DEPT table based on Lookup Condition.
- **Perform a calculation:** We want only those Employees who's SAL > Average (SAL). We will write Lookup Override query.
- **Update slowly changing dimension tables:** Most important use. We can use a Lookup transformation to determine whether rows already exist in the target.

1. LOOKUP TYPES

We can configure the Lookup transformation to perform the following types of lookups:

- Connected or Unconnected
- Relational or Flat File
- Cached or Un cached

Relational Lookup:

When we create a Lookup transformation using a relational table as a lookup source, we can connect to the lookup source using ODBC and import the table definition as the structure for the Lookup transformation.

- We can override the default SQL statement if we want to add a WHERE clause or query multiple tables.
- We can use a dynamic lookup cache with relational lookups.

Flat File Lookup:

When we use a flat file for a lookup source, we can use any flat file definition in the repository, or we can import it. When we import a flat file lookup source, the Designer invokes the Flat File Wizard.

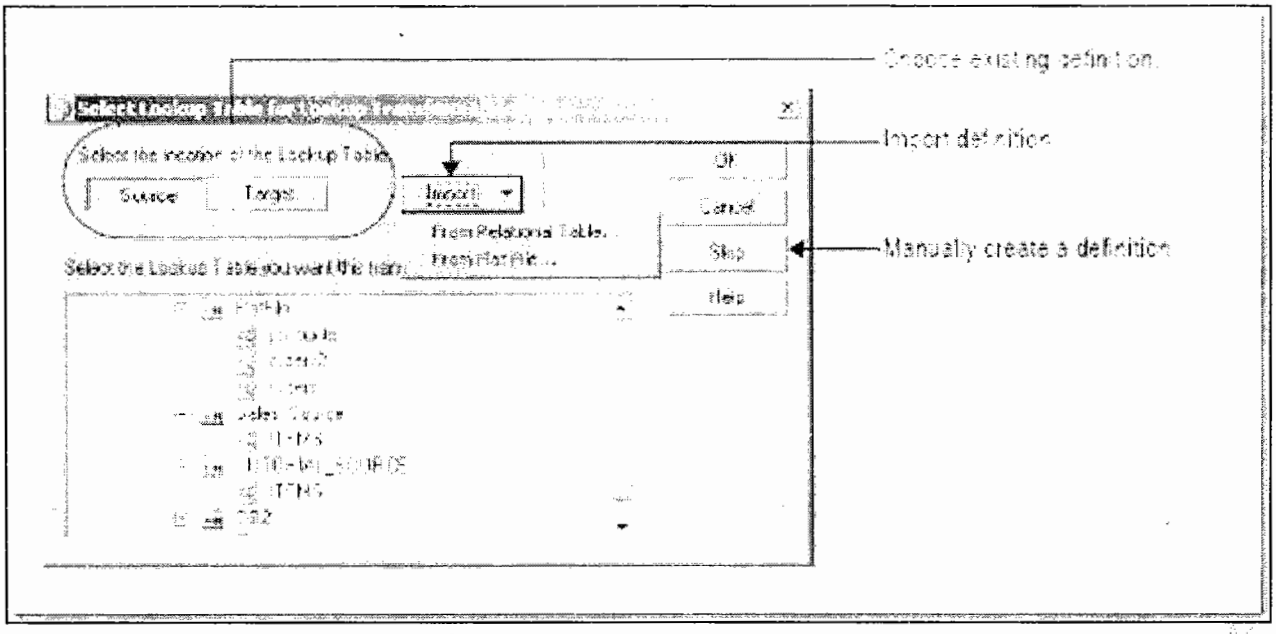
Cached or Un cached Lookup:

We can check the option in Properties Tab to Cache to lookup or not. By default, lookup is cached.

Connected and Unconnected Lookup

Connected Lookup	Unconnected Lookup
Receives input values directly from the pipeline.	Receives input values from the result of a :LKP expression in another transformation.
We can use a dynamic or static cache.	We can use a static cache.
Cache includes all lookup columns used in the mapping.	Cache includes all lookup/output ports in the lookup condition and the lookup/return port.
If there is no match for the lookup condition, the Power Center Server returns the default value for all output ports.	If there is no match for the lookup condition, the Power Center Server returns NULL.
If there is a match for the lookup condition, the Power Center Server returns the result of the lookup condition for all lookup/output	If there is a match for the lookup condition,the Power Center Server returns the result of the lookup condition into the return port.

ports.	
Pass multiple output values to another transformation.	Pass one output value to another transformation.
Supports user-defined default values	Does not support user-defined default values.



2 .LOOKUP T/F COMPONENTS

Define the following components when we configure a Lookup transformation in a mapping:

- Lookup source
- Ports
- Properties
- Condition

1. Lookup Source:

We can use a flat file or a relational table for a lookup source. When we create a Lookup t/f, we can import the lookup source from the following locations:

- Any relational source or target definition in the repository
- Any flat file source or target definition in the repository
- Any table or file that both the Power Center Server and Client machine can connect to The lookup table can be a single table, or we can join multiple tables in the same database using a lookup SQL override in Properties Tab.

2. Ports:

Ports	**Lookup Type**	**Number Needed**	**Description**
I	Connected Unconnected	Minimum 1	Input port to Lookup. Usually ports used for Join condition are Input ports.

O	Connected Unconnected	Minimum 1	Ports going to another transformation from Lookup.
L	Connected Unconnected	Minimum 1	Lookup port. The Designer automatically Designates each column in the lookup source as a lookup (L) and output port (O).
R	Unconnected	1 Only	Return port. Use only in unconnected Lookup t/f only.

3. Properties Tab

Options	Lookup Type	Description
Lookup SQL Override	Relational	Overrides the default SQL statement to query the lookup table.
Lookup Table Name	Relational	Specifies the name of the table from which the transformation looks up and caches values.
Lookup Caching Enabled	Flat File, Relational	Indicates whether the Power Center Server caches lookup values during the session.
Lookup Policy on Multiple Match	Flat File, Relational	Determines what happens when the Lookup transformation finds multiple rows that match the lookup condition. Options: Use First Value or Use Last Value or Use Any Value or Report Error
Lookup Condition	Flat File, Relational	Displays the lookup condition you set in the Condition tab.
Connection Information	Relational	Specifies the database containing the lookup table.
Source Type	Flat File, Relational	Lookup is from a database or flat file.
Lookup Cache Directory Name	Flat File, Relational	Location where cache is build.
Lookup Cache Persistent	Flat File, Relational	Whether to use Persistent Cache or not.
Dynamic Lookup Cache	Flat File, Relational	Whether to use Dynamic Cache or not.
Recache From Lookup Source	Flat File, Relational	To rebuild cache if cache source changes and we are using Persistent Cache.
Insert Else Update	Relational	Use only with dynamic caching enabled. Applies to rows entering the Lookup transformation with the row type of insert.

Lookup Data Cache Size	Flat File, Relational	Data Cache Size
Lookup Index Cache Size	Flat File, Relational	Index Cache Size
Cache File Name Prefix	Flat File, Relational	Use only with persistent lookup cache. Specifies the file name prefix to use with persistent lookup cache files.

Some other properties for Flat Files are:

- Date time Format
- Thousand Separator
- Decimal Separator
- Case-Sensitive String Comparison
- Null Ordering
- Sorted Input

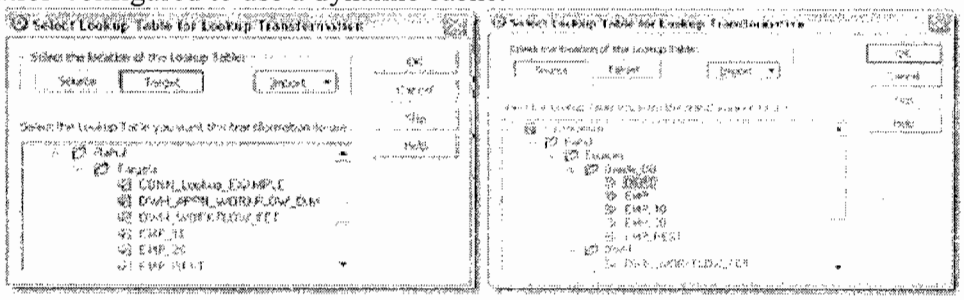
4: Condition Tab

We enter the Lookup Condition. The Power Center Server uses the lookup condition to test incoming values. We compare transformation input values with values in the lookup source or cache, represented by lookup ports.

- The data types in a condition must match.
- When we enter multiple conditions, the Power Center Server evaluates each condition as an AND, not an OR.
- The Power Center Server matches null values.
- The input value must meet all conditions for the lookup to return a value.
- =, >, <, >=, <=, != Operators can be used.
- Example: IN_DEPTNO = DEPTNO
  In_DNAME = 'DELHI'
  Tip: If we include more than one lookup condition, place the conditions with an equal sign first to optimize lookup performance.

Note:

1. We can use = operator in case of Dynamic Cache.
2. The Power Center Server fails the session when it encounters multiple keys for a Lookup transformation configured to use a dynamic cache.



3. Connected Lookup Transformation

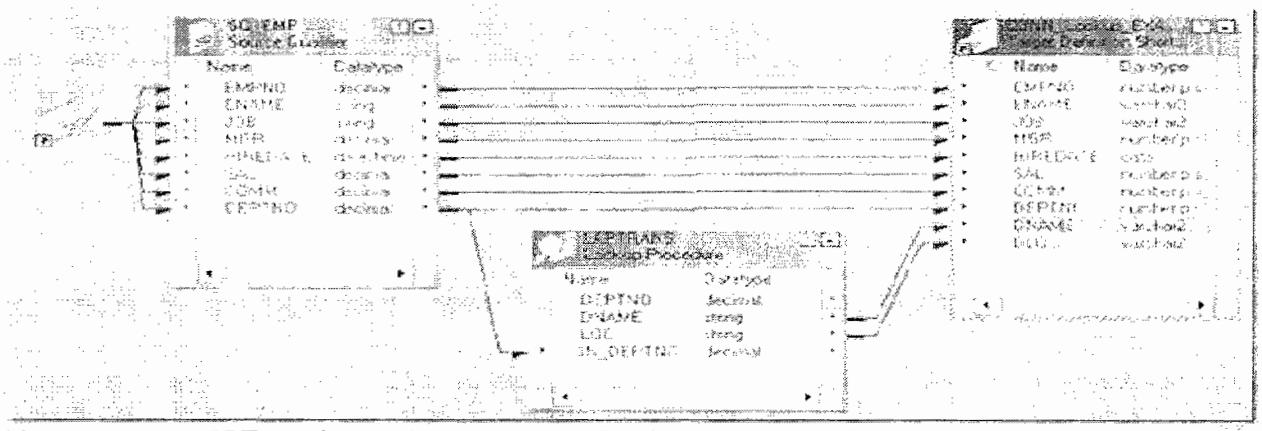
Example: To create a connected Lookup Transformation

- EMP will be source table. DEPT will be LOOKUP table.
- Create a target table CONN_Lookup_EXAMPLE in target designer. Table should contain all ports of EMP table plus DNAME and LOC as shown below.

- Create the shortcuts in your folder.

Creating Mapping:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_CONN_LOOKUP_EXAMPLE
4. Drag EMP and Target table.
5. Connect all fields from SQ_EMP to target except DNAME and LOC.
6. Transformation-> Create -> Select LOOKUP from list. Give name and click Create.
7. The Following screen is displayed.
8. As DEPT is the Source definition, click Source and then Select DEPT.
9. Click Ok.



10. Now Pass DEPTNO from SQ_EMP to this Lookup. DEPTNO from SQ_EMP will be named as DEPTNO1. Edit Lookup and rename it to IN_DEPTNO in ports tab.
11. Now go to CONDITION tab and add CONDITION.

DEPTNO = IN_DEPTNO and Click Apply and then OK.

Link the mapping as shown below:

12. We are not passing IN_DEPTNO and DEPTNO to any other transformation from LOOKUP; we can edit the lookup transformation and remove the OUTPUT check from them.
13. Mapping -> Validate
14. Repository -> Save

- Create Session and Workflow as described earlier. Run the workflow and see the data in target table.
- Make sure to give connection information for all tables.
- Make sure to give connection for LOOKUP Table also.

Lookup Cache Files

1. **Lookup Index Cache:**

- Stores data for the columns used in the lookup condition.

2. **Lookup Data Cache:**

- For a connected Lookup transformation, stores data for the connected output ports, not including ports used in the lookup condition.
- For an unconnected Lookup transformation, stores data from the return port.

Types of Lookup Caches:

1. Static Cache

By default, the IS creates a static cache. It caches the lookup file or table and Looks up values in the cache for each row that comes into the transformation.The IS does not update the cache while it processes the Lookup transformation.

2. Dynamic Cache

To cache a target table or flat file source and insert new rows or update existing rows in the cache, use a Lookup transformation with a dynamic cache.

The IS dynamically inserts or updates data in the lookup cache and passes data to the target. Target table is also our lookup table. No good for performance if table is huge.

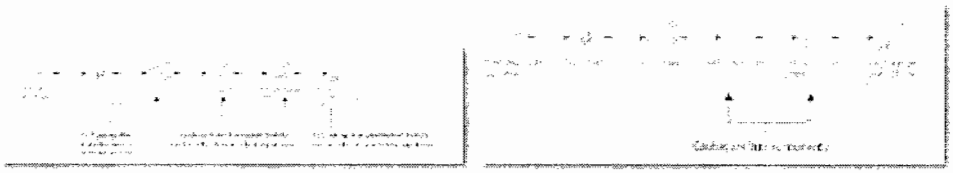
3. Persistent Cache

If the lookup table does not change between sessions, we can configure the Lookup transformation to use a persistent lookup cache.

The IS saves and reuses cache files from session to session, eliminating the time Required to read the lookup table.

4. Recache from Source

If the persistent cache is not synchronized with the lookup table, we can Configure the Lookup transformation to rebuild the lookup cache.If Lookup table has changed, we can use this to rebuild the lookup cache.



5. Shared Cache

- **Unnamed cache:** When Lookup transformations in a mapping have compatible caching structures, the IS shares the cache by default. You can only share static unnamed caches.
- **Named cache:** Use a persistent named cache when we want to share a cache file across mappings or share a dynamic and a static cache. The caching structures must match or be compatible with a named cache. You can share static and dynamic named caches.

MAPPING PARAMETERS AND VARIABLES

Mapping parameters and variables represent values in mappings and mapplets.

When we use a mapping parameter or variable in a mapping, first we declare the mapping parameter or variable for use in each mapplet or mapping. Then, we define a value for the mapping parameter or variable before we run the session.

MAPPING PARAMETERS

- A mapping parameter represents a constant value that we can define before running a session.
- A mapping parameter retains the same value throughout the entire session.

  Example: When we want to extract records of a particular month during ETL process, we will create a Mapping Parameter of data type and use it in query to compare it with the timestamp field in SQL override.
- After we create a parameter, it appears in the Expression Editor.
- We can then use the parameter in any expression in the mapplet or mapping.
- We can also use parameters in a source qualifier filter, user-defined join, or extract override, and in the Expression Editor of reusable transformations.

MAPPING VARIABLES

- Unlike mapping parameters, mapping variables are values that can change between sessions.
- The Integration Service saves the latest value of a mapping variable to the repository at the end of each successful session.
- We can override a saved value with the parameter file.
- We can also clear all saved values for the session in the Workflow Manager.

We might use a mapping variable to perform an incremental read of the source. For example, we have a source table containing time stamped transactions and we want to evaluate the transactions on a daily basis. Instead of manually entering a session override to filter source data each time we run the session, we can create a mapping variable, $$IncludeDateTime. In the source qualifier, create a filter to read only rows whose transaction date equals $$IncludeDateTime, such as:

TIMESTAMP = $$IncludeDateTime

In the mapping, use a variable function to set the variable value to increment one day each time the session runs. If we set the initial value of $$IncludeDateTime to 8/1/2004, the first time the Integration Service runs the session, it reads only rows dated 8/1/2004. During the session, the Integration Service sets $$IncludeDateTime to 8/2/2004. It saves 8/2/2004 to the repository at the end of the session. The next time it runs the session, it reads only rows from August 2, 2004.

Used in following transformations:

- Expression
- Filter
- Router
- Update Strategy

Initial and Default Value:

When we declare a mapping parameter or variable in a mapping or a mapplet, we can enter an initial value. When the Integration Service needs an initial value, and we did not declare an initial value for the parameter or variable, the Integration Service uses a default value based on the data type of the parameter or variable.

Data ->Default Value
Numeric ->0
String ->Empty String
Date time ->1/1/1

Variable Values: Start value and current value of a mapping variable

Start Value:

The start value is the value of the variable at the start of the session. The Integration Service looks for the start value in the following order:

1. Value in parameter file
2. Value saved in the repository
3. Initial value
4. Default value

/

Current Value:
The current value is the value of the variable as the session progresses. When a session starts, the current value of a variable is the same as the start value. The final current value for a variable is saved to the repository at the end of a successful session. When a session fails to complete, the Integration Service does not update the value of the variable in the repository.
Note: If a variable function is not used to calculate the current value of a mapping variable, the start value of the variable is saved to the repository.
Variable Data type and Aggregation Type When we declare a mapping variable in a mapping, we need to configure the Data type and aggregation type for the variable. The IS uses the aggregate type of a Mapping variable to determine the final current value of the mapping variable.

Aggregation types are:

- **Count:** Integer and small integer data types are valid only.
- **Max:** All transformation data types except binary data type are valid.
- **Min:** All transformation data types except binary data type are valid.

Variable Functions
Variable functions determine how the Integration Service calculates the current value of a mapping variable in a pipeline.
SetMaxVariable: Sets the variable to the maximum value of a group of values. It ignores rows marked for update, delete, or reject. Aggregation type set to Max.
SetMinVariable: Sets the variable to the minimum value of a group of values. It ignores rows marked for update, delete, or reject. Aggregation type set to Min.
SetCountVariable: Increments the variable value by one. It adds one to the variable value when a row is marked for insertion, and subtracts one when the row is Marked for deletion. It ignores rows marked for update or reject. Aggregation type set to Count.
SetVariable: Sets the variable to the configured value. At the end of a session, it compares the final current value of the variable to the start value of the variable. Based on the aggregate type of the variable, it saves a final value to the repository.

Creating Mapping Parameters and Variables

1. Open the folder where we want to create parameter or variable.
2. In the Mapping Designer, click Mappings > Parameters and Variables. -or- In the Mapplet Designer, click Mapplet > Parameters and Variables.
3. Click the add button.
4. Enter name. Do not remove $$ from name.
5. Select Type and Data type. Select Aggregation type for mapping variables.
6. Give Initial Value. Click ok.

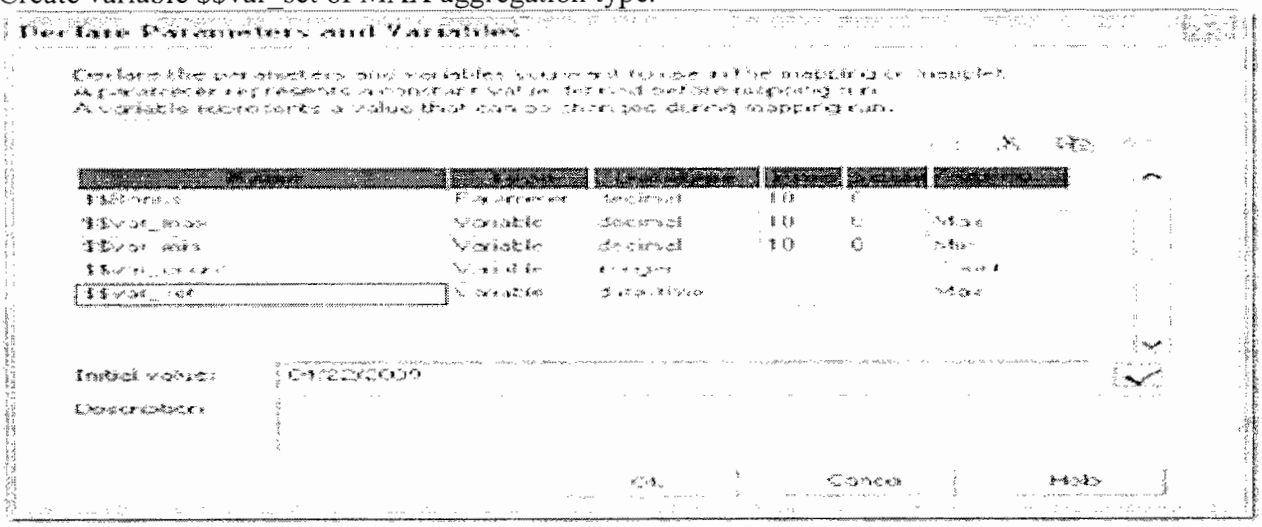
Example: Use of Mapping of Mapping Parameters and Variables

- EMP will be source table.
- Create a target table MP_MV_EXAMPLE having columns: EMPNO, ENAME, DEPTNO, TOTAL_SAL, MAX_VAR, MIN_VAR, COUNT_VAR and SET_VAR.
- TOTAL_SAL = SAL+ COMM + $$BONUS (Bonus is mapping parameter that changes every month)
- SET_VAR: We will be added one month to the HIREDATE of every employee.

- Create shortcuts as necessary.

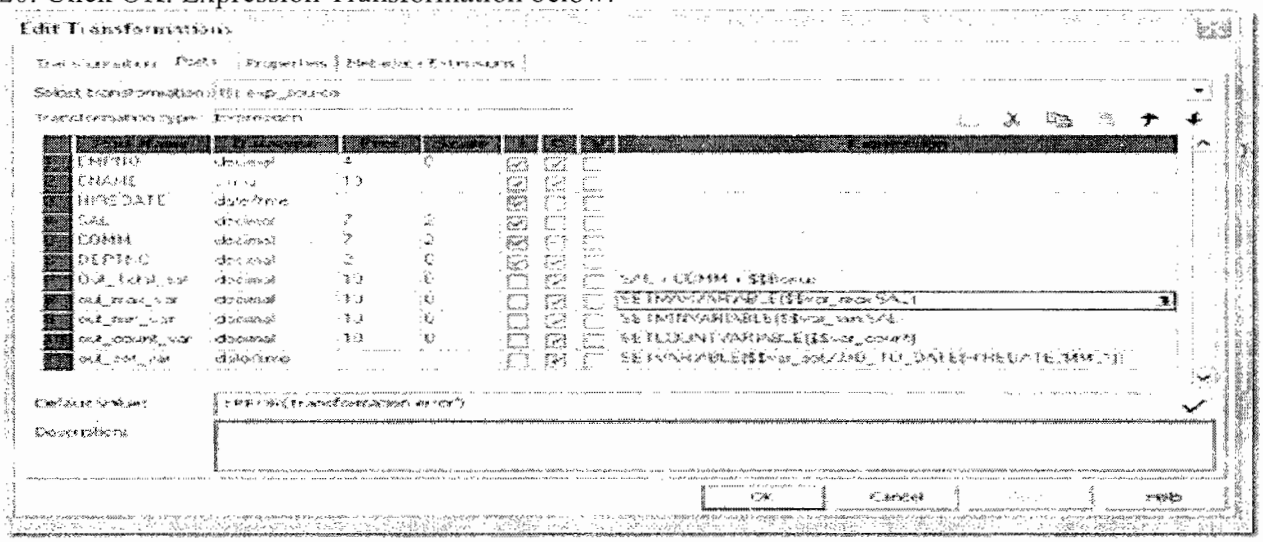
Creating Mapping

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_mp_mv_example
4. Drag EMP and target table.
5. Transformation -> Create -> Select Expression for list -> Create –> Done.
6. Drag EMPNO, ENAME, HIREDATE, SAL, COMM and DEPTNO to Expression.
7. Create Parameter $$Bonus and Give initial value as 200.
8. Create variable $$var_max of MAX aggregation type and initial value 1500.
9. Create variable $$var_min of MIN aggregation type and initial value 1500.
10. Create variable $$var_count of COUNT aggregation type and initial value 0. COUNT is visible when datatype is INT or SMALLINT.
11. Create variable $$var_set of MAX aggregation type.

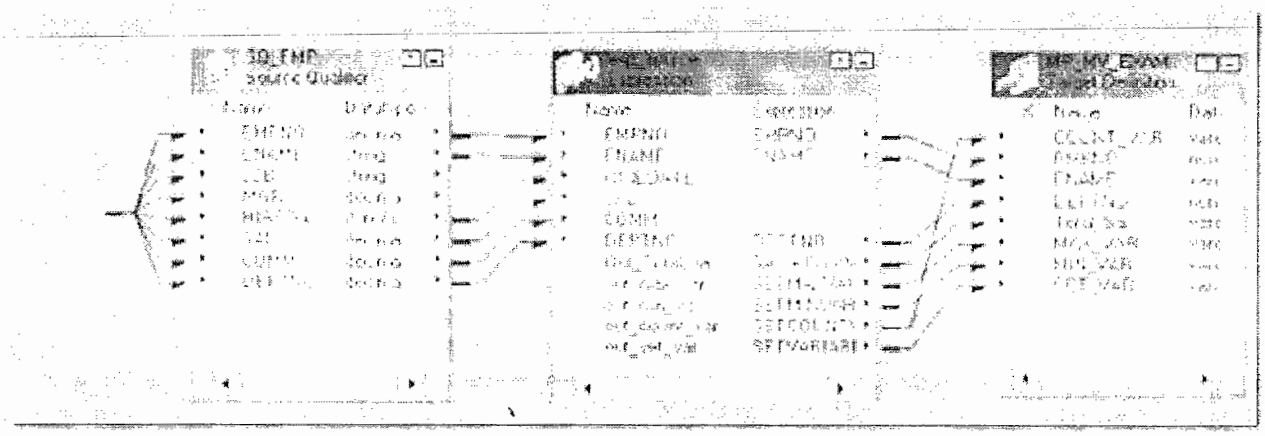


12. Create 5 output ports out_ TOTAL_SAL, out_MAX_VAR, out_MIN_VAR, out_COUNT_VAR and out_SET_VAR.
13. Open expression editor for TOTAL_SAL. Do the same as we did earlier for SAL+ COMM. To add $$BONUS to it, select variable tab and select the parameter from mapping parameter. SAL + COMM + $$Bonus
14. Open Expression editor for out_max_var.
15. Select the variable function SETMAXVARIABLE from left side pane. Select $$var_max from variable tab and SAL from ports tab as shown below.SETMAXVARIABLE($$var_max,SAL)

17. Open Expression editor for out_min_var and write the following expression: SETMINVARIABLE($$var_min,SAL). Validate the expression.
18. Open Expression editor for out_count_var and write the following expression: SETCOUNTVARIABLE($$var_count). Validate the expression.
19. Open Expression editor for out_set_var and write the following expression: SETVARIABLE($$var_set,ADD_TO_DATE(HIREDATE,'MM',1)). Validate.
20. Click OK. Expression Transformation below:



21. Link all ports from expression to target and Validate Mapping and Save it.
22. See mapping picture on next page.



PARAMETER FILE

- A parameter file is a list of parameters and associated values for a workflow, worklet, or session.
- Parameter files provide flexibility to change these variables each time we run a workflow or session.
- We can create multiple parameter files and change the file we use for a session or workflow. We can create a parameter file using a text editor such as WordPad or Notepad.
- Enter the parameter file name and directory in the workflow or session properties.

A parameter file contains the following types of parameters and variables:

- **Workflow variable:** References values and records information in a workflow.
- **Worklet variable:** References values and records information in a worklet. Use predefined worklet variables in a parent workflow, but we cannot use workflow variables from the parent workflow in a worklet.
- **Session parameter:** Defines a value that can change from session to session, such as a database connection or file name.
- Mapping parameter and Mapping variable

USING A PARAMETER FILE

Parameter files contain several sections preceded by a heading. The heading identifies the Integration Service, Integration Service process, workflow, worklet, or session to which we want to assign parameters or variables.

- Make session and workflow.
- Give connection information for source and target table.
- Run workflow and see result.

Heading	Scope
[Global]	All Integration Services, Integration Service processes, workflows, worklets, and sessions.
[Integration Service name]	The named Integration Service and workflows, worklets, and sessions that run on the service.
[Integration Service name.wf:workflow name]	The named Integration Service and workflow, worklet, and sessions that run on the service.
[folder name.WF:workflow name]	The named workflow and all sessions within the workflow.
[folder name.WF:workflow name.WT:worklet name]	The named worklet and all sessions within the worklet.
[folder name.WF:workflow name.WT:worklet name...]	The nested worklet and all sessions within the nested worklet.
[folder name.WF:workflow name.ST:session name] -or- [folder name.WF:workflow name.WT:worklet name.ST:session name] -or- [folder name.session name] -or- [session name]	The named session.

Sample Parameter File for Our example:

In the parameter file, folder and session names are case sensitive.

Create a text file in notepad with name Para_File.txt

[Practice.ST:s_m_MP_MV_Example]

$$Bonus=1000

$$var_max=500

$$var_min=1200

$$var_count=0

CONFIGURING PARAMTER FILE

We can specify the parameter file name and directory in the workflow or session properties.

To enter a parameter file in the workflow properties:

1. Open a Workflow in the Workflow Manager.
2. Click Workflows > Edit.
3. Click the Properties tab.
4. Enter the parameter directory and name in the Parameter Filename field.
5. Click OK.

To enter a parameter file in the session properties:

1. Open a session in the Workflow Manager.
2. Click the Properties tab and open the General Options settings.
3. Enter the parameter directory and name in the Parameter Filename field.
4. Example: D:\Files\Para_File.txt or $PMSourceFileDir\Para_File.txt
5. Click OK.

MAPPLETS

- A mapplet is a reusable object that we create in the Mapplet Designer.
- It contains a set of transformations and lets us reuse that transformation logic in multiple mappings.
- Created in Mapplet Designer in Designer Tool.
  We need to use same set of 5 transformations in say 10 mappings. So instead of making 5 transformations in every 10 mapping, we create a mapplet of these 5 transformations. Now we use this mapplet in all 10 mappings. Example: To create a surrogate key in target. We create a mapplet using a stored procedure to create Primary key for target table. We give target table name and key column name as input to mapplet and get the Surrogate key as output.
  Mapplets help simplify mappings in the following ways:
- Include source definitions: Use multiple source definitions and source qualifiers to provide source data for a mapping.
- Accept data from sources in a mapping
- Include multiple transformations: As many transformations as we need.
- Pass data to multiple transformations: We can create a mapplet to feed data to multiple transformations. Each Output transformation in a mapplet represents one output group in a mapplet.
- Contain unused ports: We do not have to connect all mapplet input and output ports in a mapping.

Mapplet Input:

Mapplet input can originate from a source definition and/or from an Input transformation in the mapplet. We can create multiple pipelines in a mapplet.

- We use Mapplet Input transformation to give input to mapplet.
- Use of Mapplet Input transformation is optional.

Mapplet Output:

The output of a mapplet is not connected to any target table.

- We must use Mapplet Output transformation to store mapplet output.
- A mapplet must contain at least one Output transformation with at least one connected port in the mapplet.

Example1: We will join EMP and DEPT table. Then calculate total salary. Give the output to mapplet out transformation.

· EMP and DEPT will be source tables.

· Output will be given to transformation Mapplet_Out.

Steps:

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapplet Designer.
3. Click Mapplets-> Create-> Give name. Ex: mplt_example1
4. Drag EMP and DEPT table.
5. Use Joiner transformation as described earlier to join them.
6. Transformation -> Create -> Select Expression for list -> Create -> Done
7. Pass all ports from joiner to expression and then calculate total salary as described in expression transformation.
8. Now Transformation -> Create -> Select Mapplet Out from list –> Create -> Give name and then done.

9. Pass all ports from expression to Mapplet output.
10. Mapplet -> Validate
11. Repository -> Save

Use of mapplet in mapping:

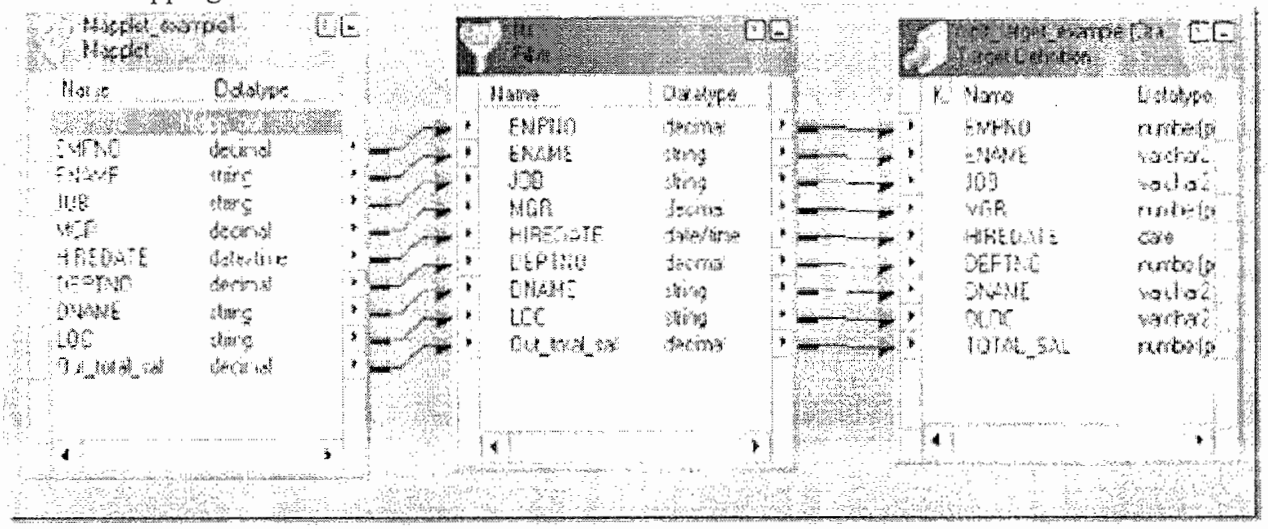
- We can mapplet in mapping by just dragging the mapplet from mapplet folder on left pane as we drag source and target tables.
- When we use the mapplet in a mapping, the mapplet object displays only the ports from the Input and Output transformations. These are referred to as the mapplet input and mapplet output ports.
- Make sure to give correct connection information in session.

Making a mapping: We will use mplt_example1, and then create a filter transformation to filter records whose Total Salary is >= 1500.

· mplt_example1 will be source.

· Create target table same as Mapplet_out transformation as in picture above. Creating Mapping

1. Open folder where we want to create the mapping.
2. Click Tools -> Mapping Designer.
3. Click Mapping-> Create-> Give name. Ex: m_mplt_example1
4. Drag mplt_Example1 and target table.
5. Transformation -> Create -> Select Filter for list -> Create -> Done.
6. Drag all ports from mplt_example1 to filter and give filter condition.
7. Connect all ports from filter to target. We can add more transformations after filter if needed.
8. Validate mapping and Save it.



- Make session and workflow.
- Give connection information for mapplet source tables.
- Give connection information for target table.
- Run workflow and see result.

WORKING WITH TASKS - 1

The Workflow Manager contains many types of tasks to help you build workflows and worklets. We can create reusable tasks in the Task Developer.

Types of tasks:

Task Type	Tool where task can be created	Reusable or not
Session	Task Developer	Yes
Email	Workflow Designer	Yes
Command	Worklet Designer	Yes
Event-Raise	Workflow Designer	No
Event-Wait	Worklet Designer	No
Timer		No
Decision		No
Assignment		No
Control		No

SESSION TASK

- A session is a set of instructions that tells the Power Center Server how and when to move data from sources to targets.
- To run a session, we must first create a workflow to contain the Session task.
- We can run as many sessions in a workflow as we need. We can run the Session tasks sequentially or concurrently, depending on our needs.
- The Power Center Server creates several files and in-memory caches depending on the transformations and options used in the session.

EMAIL TASK

- The Workflow Manager provides an Email task that allows us to send email during a workflow.
- Created by Administrator usually and we just drag and use it in our mapping.

Steps:

1. In the Task Developer or Workflow Designer, choose Tasks-Create.
2. Select an Email task and enter a name for the task. Click Create.
3. Click Done.
4. Double-click the Email task in the workspace. The Edit Tasks dialog box appears.
5. Click the Properties tab.
6. Enter the fully qualified email address of the mail recipient in the Email User Name field.
7. Enter the subject of the email in the Email Subject field. Or, you can leave this field blank.
8. Click the Open button in the Email Text field to open the Email Editor.
9. Click OK twice to save your changes.

Example: To send an email when a session completes:

Steps:

1. Create a workflow wf_sample_email

2. Drag any session task to workspace.
3. Edit Session task and go to Components tab.
4. See On Success Email Option there and configure it.
5. In Type select reusable or Non-reusable.
6. In Value, select the email task to be used.
7. Click Apply -> Ok.
8. Validate workflow and Repository -> Save

- We can also drag the email task and use as per need.
- We can set the option to send email on success or failure in components tab of a session task.

COMMAND TASK

The Command task allows us to specify one or more shell commands in UNIX or DOS commands in Windows to run during the workflow.

For example, we can specify shell commands in the Command task to delete reject files, copy a file, or archive target files.

Ways of using command task:

1. **Standalone Command task:** We can use a Command task anywhere in the workflow or worklet to run shell commands.

2. Pre- and post-session shell command: We can call a Command task as the pre- or post-session shell command for a Session task. This is done in COMPONENTS TAB of a session. We can run it in Pre-Session Command or Post Session Success Command or Post Session Failure Command. Select the Value and Type option as we did in Email task.

Example: to copy a file sample.txt from D drive to E.

Command: COPY D:\sample.txt E:\ in windows

Steps for creating command task:

1. In the Task Developer or Workflow Designer, choose Tasks-Create.
2. Select Command Task for the task type.
3. Enter a name for the Command task. Click Create. Then click done.
4. Double-click the Command task. Go to commands tab.
5. In the Commands tab, click the Add button to add a command.
6. In the Name field, enter a name for the new command.
7. In the Command field, click the Edit button to open the Command Editor.
8. Enter only one command in the Command Editor.
9. Click OK to close the Command Editor.
10. Repeat steps 5-9 to add more commands in the task.
11. Click OK.

Steps to create the workflow using command task:

1. Create a task using the above steps to copy a file in Task Developer.
2. Open Workflow Designer. Workflow -> Create -> Give name and click ok.
3. Start is displayed. Drag session say s_m_Filter_example and command task.

4. Link Start to Session task and Session to Command Task.
5. Double click link between Session and Command and give condition in editor as
6. $S_M_FILTER_EXAMPLE.Status=SUCCEEDED
7. Workflow-> Validate
8. Repository –> Save

UNIT TESTING

Unit testing can be broadly classified into 2 categories.

Quantitative Testing

Validate your Source and Target

a) Ensure that your connectors are configured properly.
b) If you are using flat file make sure have enough read/write permission on the file share.
c) You need to document all the connector information.

Analyze the Load Time

a) Execute the session and review the session statistics.
b) Check the Read and Write counters. How long it takes to perform the load.
c) Use the session and workflow logs to capture the load statistics.
d) You need to document all the load timing information.

Analyze the success rows and rejections.

a) Have customized SQL queries to check the source/targets and here we will perform the Record Count Verification.
b) Analyze the rejections and build a process to handle those rejections. This requires a clear business requirement from the business on how to handle the data rejections. Do we need to reload or reject and inform etc? Discussions are required and appropriate process must be developed.

Performance Improvement

a) Network Performance
b) Session Performance
c) Database Performance
d) Analyze and if required define the Informatica and DB partitioning requirements.

Qualitative Testing

Analyze & validate your transformation business rules. More of functional testing.
e) You need review field by field from source to target and ensure that the required transformation logic is applied.
f) If you are making changes to existing mappings make use of the **data lineage** feature Available with Informatica Power Center. This will help you to find the consequences of Altering or deleting a port from existing mapping.
g) Ensure that appropriate dimension lookup's have been used and your development is in Sync with your business requirements.

INTEGRATION TESTING

After unit testing is complete; it should form the basis of starting integration testing. Integration testing should

Test out initial and incremental loading of the data warehouse.

Integration testing will involve following

1. Sequence of ETL jobs in batch.
2. Initial loading of records on data warehouse.
3. Incremental loading of records at a later date to verify the newly inserted or updated data.
4. Testing the rejected records that don't fulfill transformation rules.
5. Error log generation.

Integration Testing would cover End-to-End Testing for DWH. The coverage of the tests would include the below:

Count Validation

Record Count Verification: DWH backend/Reporting queries against source and target as an initial check.

Control totals: To ensure accuracy in data entry and processing, control totals can be compared by the system with manually entered or otherwise calculated control totals using the data fields such as quantities, line items, documents, or dollars, or simple record counts

Hash totals: This is a technique for improving data accuracy, whereby totals are obtained on identifier fields (i.e., fields for which it would logically be meaningless to construct a total), such as account number, social security number, part number, or employee number. These totals have no significance other than for internal system control purposes.

Limit checks: The program tests specified data fields against defined high or low value limits (e.g., quantities or dollars) for acceptability before further processing.

Data Quality Validation

Check for missing data, negatives and consistency. Field-by-Field data verification can be done to check the consistency of source and target data.

Overflow checks: This is a limit check based on the capacity of a data field or data file area to accept data. This programming technique can be used to detect the truncation of a financial or quantity data field value after computation (e.g., addition, multiplication, and division). Usually, the first digit is the one lost.

Format checks: These are used to determine that data are entered in the proper mode, as numeric or alphabetical characters, within designated fields of information. The proper mode in each case depends on the data field definition.

Sign test: This is a test for a numeric data field containing a designation of an algebraic sign, + or - , which can be used to denote, for example, debits or credits for financial data fields.

Size test: This test can be used to test the full size of the data field. For example, a social security number in the United States should have nine digits

Granularity

Validate at the lowest granular level possible

Other validations

Audit Trails, Transaction Logs, Error Logs and Validity checks.

Note: Based on your project and business needs you might have additional testing requirements.

ETL Testing Concepts

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

Test Strategy of ETL/BI/DWH Project:

Test strategy of ETL project contains below documents, some projects will maintain all below documents in single document.

- BI Project Objective.
- Test Planning.
- Test Execution.
- Scope
- System Test Environment and Approach
- Entry and Exit Criteria

ETL/BI/DWH Project Objective:

- The success of a DW/BI program lies in meeting its key objective of ensuring data accuracy (DW construction) and providing a single version of the truth through flexibility in analysis/reporting (presentation).
- In order to achieve these critical success factors, data in a typical DW architecture passes through several steps of consolidation making it pertinent that a well defined testing framework be established.
- Consider the inputs, processes, and outputs of a source-to-target data warehouse, the high number of components and their integration that needs to be tested etc. This makes the testing process extremely complicated and important and hence should be treated as a project unto itself.

Test Planning:

- Develop Test Approach.
- Develop Test Execution Plan.
- Define E2E Scenarios.
- Define Test Conditions.
- Create Test Cases for the Test Conditions.
- Identify Test Data.

- **Develop Test Approach**

Test Approach document outlines the objective, scope, entry/exit criteria, test environment and tools used to test the system.

- **Develop Test Execution Plan**

Test execution plan outlines the tasks, resources and schedule for test plan preparation and execution.

- **Define E2E Scenarios.**

An end 2 end scenario relates to functional event that maps to overall business scenario. E2E scenarios are captured and mapped to various sub-scenarios and test cases.

- **Define Test Condition.**

A test condition is the lowest level item to be tested that is defined during the test preparation process. For each test condition expected result is defined.

- **Create Test Cases for the Test Conditions.**

Test cases document the input, expected results, and execution conditions of a given test condition. Test cases are broken down into one or more detailed test scripts and test data conditions for execution.

- **Identify Test Data.**

Test Data is identified to support the test execution. Tester is responsible to identify test data requirement and prepare test data to execute the test scripts.

Scope:

In Scope

- Data Integrity Check
- Referential Integrity Check (Orphan Check)
- Null/Column Completeness Check
- Duplicate Check
- Scenario Specific to the Work request and impacted objects.
- Check on the Dependent objects

Out of Scope

- Testing on the area that is not related to the impacted or dependant objects..

Entry Criteria:

- Signed-off requirements.
- Code signed-off by Dev and Implemented for testing.

Exit Criteria:

- All test scripts that are in scope are executed and the results documented.
- All severity defects are closed.
- All unresolved defects are fully documented and sent to the client team.

Tools/Templates/Reports:

These are Tools/Templates/Reports which are used in project.

- Tools - demo
  - Using one Automation tool i.e. QTP tool
  - Using Teradata SQL Assistant, QC tool, RTC and CDB.
- Templates/reports - walkthrough
  - Test Plan
  - Test cases
  - RTM (Requirement Traceability Matrix)
  - QA check list

Defect Reporting Template

The Defect Description should contains the minimum below information

- cc:
- Summary:
- % Impacted:

- Any trend observed?
- Samples:
- Issue in another environment (TIN/TAC/PRD)?
- Steps to Reproduce:

Process Followed in ETL/BI/DWH Project:

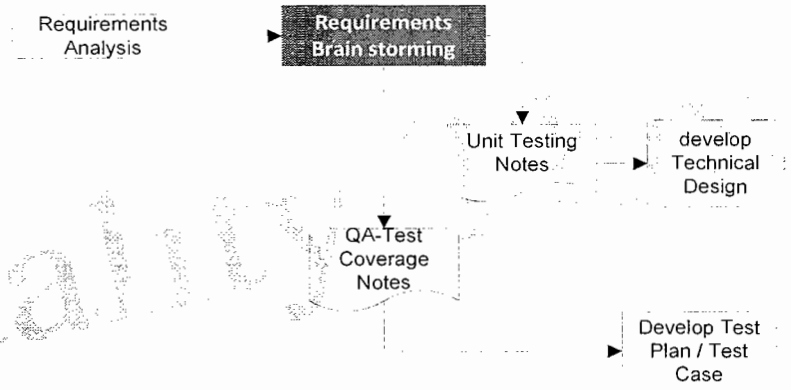
1. Requirements Brain storming process (Requirements phase)
2. Design Review process (Design phase)
3. Code Review process (Development phase)
4. Offshore Test Plan/Test Case Review process (Development phase)
5. Onsite Test Plan/Test Case Review (Development phase)

1. Requirements Brain storming process

Objective:
To ensure that the *Dev & QA understands the requirements* correctly and are on same page before starting development of Code/Test cases respectively to ensure *high quality deliverables and maximum test coverage.*

Phase	Requirements phase
Responsible	Offshore Lead + Dev + QA

Process:



- Dev and QA will have the discussion around the requirements.
- Dev discusses the *change in design and code*, facilitating QA to think about *different scenarios to test.*
- QA discusses the test strategy and major test cases to help Dev team to ensure that these are covered in the unit testing as well.
- Clarification tracker is updated and sent to Onsite ,if any requirement is not clear to DEV or/and QA

Outcome:

- Helps **Dev** team to think about different scenarios that need to be covered in Unit Testing, which would *minimize the defect leakage to TEST environments.*

- Helps **QA** team to *cover all test scenarios* and document them in test Approach document and Functional/Integration test Cases to test the change thoroughly. *Hence maximum test coverage in test environments and arrest the defects leakage to production environment.*

Communication Model:

- The Dev and QA team connects in a meeting room and discusses the requirements thoroughly.
- The clarifications would be discussed internally amongst the team and is then sent to Onsite for further confirmation.
- The Onsite would provide the answers as per the understanding of requirements gathering session or would approach the DA for further details and provide detailed explanation.

Handoffs/ Notes:

- Notes on requirements/test coverage.

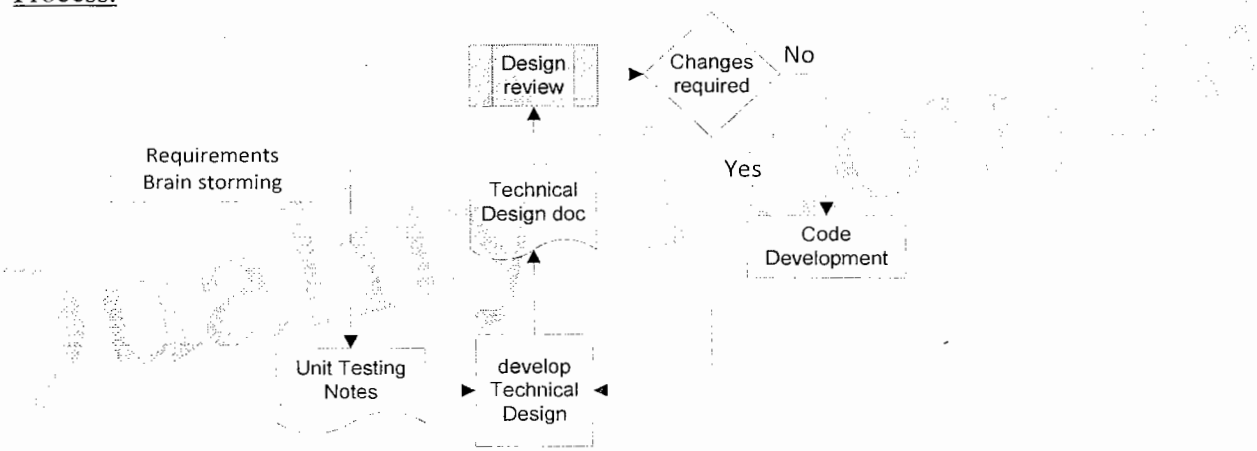
2. Design Review Process

Objective:

To make sure *effective/efficient implementation of ETL design to accommodate future Enhancements/Maintenance requests.*

Phase	Design phase
Responsible	Dev Lead

Process:



- Discuss design strategy with Offshore Dev lead.
- Review comments (if any) are implemented.
- The reviewed design is shared with Onsite coordinator.
- Review comments (if any) are implemented.
- The Design is signed off.

Out Come:

- Best Design to *serve current requirements and flexible for maintenance.*

Communication Model:

- Design is discussed with Offshore Dev lead and Shared with Onsite coordinator. The Design is signed off after the confirmation from Onsite Coordinator.

Handoffs/ Notes:

- High Level Design (HLD) and Low Level Design (LLD) document.
- Design Review Checklist - updated with review comments, if any.

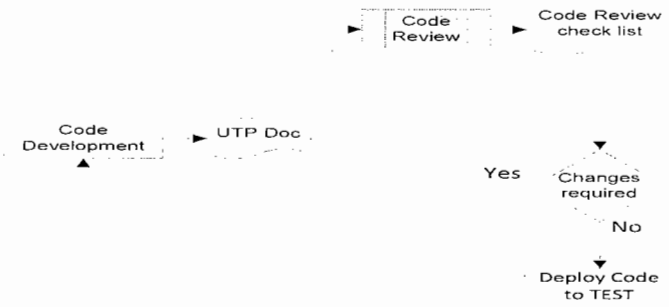
3. Code Review process

Objective:

To make sure that code written, covers each branch of the If- Else transformation rules, business logic and adheres to the coding standards.

Phase	Development phase
Responsible	Dev Lead

Process:



- Developer has to notify the reviewer through email communication once the development is completed.
- The Code is reviewed against the check list and requirements document.
- Review comments are passed to the developer and same would be updated in review check list.
- Developer has to ensure the implementation of review comments and get it signed off from reviewer.

Out Come:

- Best Code built to satisfy current requirements following coding standards and adhering to project standards.
- Code Review checklist.

Communication Model:

- Developer has to notify the reviewer through email communication once the development is completed and
- Reviewer should notify the developer in case of any review comments.

Handoffs/ Notes:

- ETL details

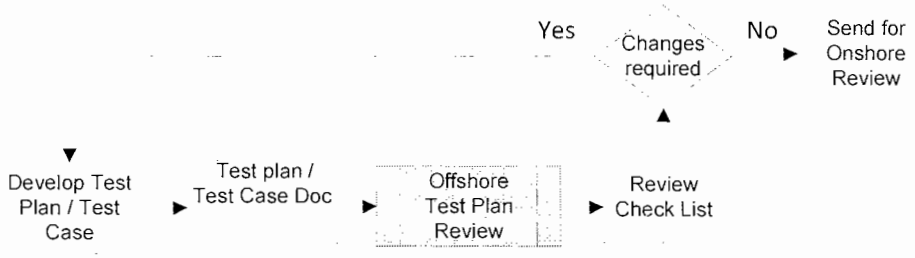
- Code Review checklist.
- Review Comments from dev lead

4. Offshore Test Plan/Test Case Review process

Objective:
To make sure that Test Approach, Test cases with SQL queries are written to cover both Positive and negative scenarios along with each branch of the If- Else transformation rules.

Phase	Development phase
Responsible	Offshore Test Lead

Process:



- Test Artifacts like Test Approach, Test cases with SQL queries are sent for the review to reviewer.
- The Review is done against the Checklist and review comments (if any) are passed to the test case developer.
- The Review comments are implemented in the respective document by the test case developer and get it signed off from review.

Out Come:
- Review Comments on test Artifacts.
- Test Approach & Test cases having maximum test coverage against requirements.

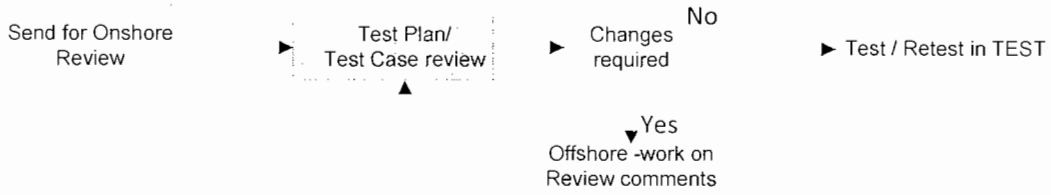
Communication Model:
Test Approach & Test cases are reviewed against Review checklist and the Review comments are provided (if any).
Handoffs/ Notes:
- Test Approach & Test cases with SQL queries
- QA Review checklist
- Review Comments from Offshore team member.

5. Onsite Test Plan/Test Case Review

Objective:
To make sure that Test Approach & Test cases with SQL queries are written to cover Positive/Negative/Business scenarios along with each branch of the If- Else transformation rules.

Phase Development phase
Responsible Onshore Lead

Process:



- Internally Reviewed Test Artifacts like Test Approach & Test cases with SQL queries are sent for the review to Onsite Coordinator.
- The Review is done against the Checklist and Review comments are passed to the offshore team.
- The Review comments are implemented in the respective document by the test case developer.
- The Docs are signed off after the confirmation from Onsite Coordinator.

Out Come:

- Review Comments on test Artifacts.
- Test Approach & Test cases having maximum test coverage.

Check List:

- QA Checklist.

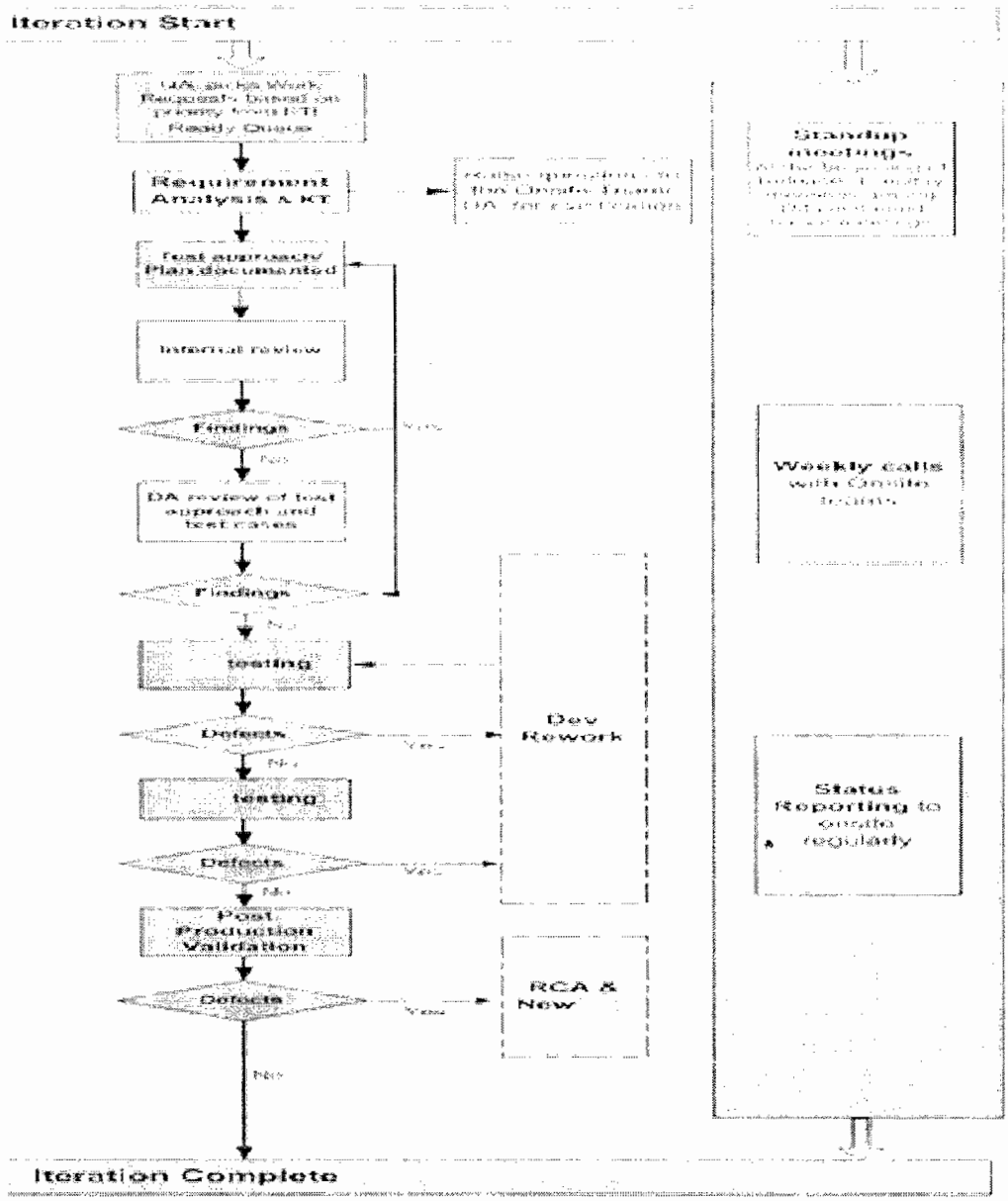
Communication Model:

- Test Artifacts like Test Approach & Test cases & SQL queries are sent for the review to Onsite testing coordinator.
- The Review is done against the Checklist and Review comments are passed to the offshore team.

Handoffs/ Notes:

- Test Approach & Test cases with SQL queries
- QA Review checklist
- Review Comments from Onsite Coordinator

Complete QA Process flow in ETL/BI/DWH Projects:



Aim: To examine a Task (or the Product) in a **Systematic** and **Comprehensive** manner in order to **Identify** the areas in which the **Standards** (Requirements) are not met and help **Correcting** them (get the impact of issue)

QA Process Flow in ETL/BI/DW Project

i. Assignment Phase

1. The DA will shortlist the list of Task from the **Work tool Database** (WTDB) and prepares the **Mini Specs** and assigns them to the respective QA Lead.
2. The QA Lead then assigns the Task to the concerned QA Resource.

Outcome: a) DA Mini Specs
b) QA Resource for particular Task

ii. Requirement Analysis & KT Phase

1. QA will analyze the assigned WR and make sure the **Requirements and Scope is clear and understood**. In case of any clarifications raise questions to DA/Onsite Client Team.
2. For QA the materials of Reference includes the **DB, Mini Specs**(prepared by DA) and **Samples** (if any provided by DA/User)
3. QA will raise concerns regarding the increase or decrease of Scope (change in Rule or reporting more tables/Columns getting impacted apart from that mentioned in DA Mini Specs) to DA keeping QA Lead in loop.
4. If QA needs KT regarding the Subject Area which the WR deals with, will be given either by DA/QA Lead or any resource which the QA Lead assigns.

Outcome: a) Updated DA Mini Specs (If any changes)
b) Requirements & Scope Freezes

iii. Estimation Phase

1. QA will estimate the total Hours required for testing the complete WR with the help of the **Standard ETL Prod Support Estimator Tool**.
2. The Estimator will have the following information:
   a) Assumptions(if any involved in testing)
   b) The phase-wise distribution of Hours (Analysis, Test Plan and Test Case Preparation, Test Plan Review, Level 1 Testing, Level 2 Testing, Production Validation, Updating the QA Repository).
3. Gets the **Estimates reviewed by QA Lead** and once it's **Signed Off,** it will be freezed.
4. Updates WTDB with Estimates.

Outcome: a) ETL Prod Support Estimator Tool with Estimates
b) WTDB with Estimates

iv. Test Approach & Test Plan Phase

1. QA will come up with a Test Approach to test the WR and document them in the **Test Plan Test Cases Spreadsheet** (MS Excel Sheet).
2. **Internal Review:** QA Lead reviews the Test Plan and documents the review comments in the **QA Review Log**. These Review comments are then

implemented in the Test Plan Sheet by QA Resource and the comments in the **QA Review Log** should be closed.

3. **External Review:** DA reviews the Test Plan and Test Cases and documents the review comments in the QA Review Log. These review comments are then implemented in the Test Plan Sheet by QA Resource and the comments in the **QA Review Log** should be closed.

Outcome: Test Plan Test Cases Sheet with the Test Plan and Test Cases

v. **Test Scripts Preparation Phase**

1. QA will prepare the Test Scripts for each Test Case and document them in the Test Plan Test Cases Spreadsheet.
2. **Peer Review:** The Test Scripts will be subjected to Peer review and QA Lead reviews the same (Optional)
3. QA will refer the QA Repository for Test Scripts if available for the particular column and table.

Outcome: Updated Test Plan Test Cases Sheet with Test Scripts

v. **Configuration Check Phase**

1. QA Lead will check if all documents are in place and correctly uploaded in the **Share Point** under the specified WR Number.
2. QA Lead will then give **Sign Off for testing** to QA Resource.

Outcome: Configuration Check List Updated and Sign-Off for Testing

vi. **Testing Phase (will go in parallel with Defect Logging Phase if Issues Found)**

1. **Level 1 Testing:** QA performs first round of testing in Level 1 environment and documents the results in Test Plan Test Cases Sheet.
2. **Level 2 Testing:** QA performs the second round of testing in Level2 Environment and documents the results in Test Plan Test Cases Sheet.
3. QA will perform retest (in each environment) after Defect Resolution.

vii. **Defect Logging Phase (Goes in parallel with the Testing Phase and iterates during each Test Environments if issues found)**

1. QA will log defects in **Clear Quest Defect Management Tool** with the necessary information.
2. QA Lead will create a **Parent Ticket** within which the QA Resource will raise the defect and assign to the Dev Lead.
3. QA Lead ensures all the necessary fields are filled by Resource while logging the defect.

Outcome: Clear Quest Test Management Tool with the Ticket Number

viii. **QA Sign Off Phase**

1. QA signs off Level 1 testing and Level 2 Testing sequentially and notifies QA Lead, Dev Lead and DA.
2. QA updates the WTDB with the status of the Work Request as **'Ready To Deploy'.**

Outcome: QA Sign – Off and Task Ready to Deploy in Production Environment

ix. **Post Production Validation Phase**

1. QA will perform testing the Task in Production Environment and documents the results in the Test Plan Test Case Sheet.
2. If any issue comes up, will be logged as a Production Defect in Clear Quest Management Tool and may result in a new Task.
3. **Root Cause Analysis (RCA):** The document prepared by QA and Dev providing the root cause for the defect miss in Test Environments and suggestions to avoid future occurrence.
4. QA signs off the WR in Production and notify the same to DA, QA Lead and Dev Lead.

x. **QA Repository Phase**

1. QA updates Repository with the latest queries and test cases depending on the Subject Area and the Tables Impacted.

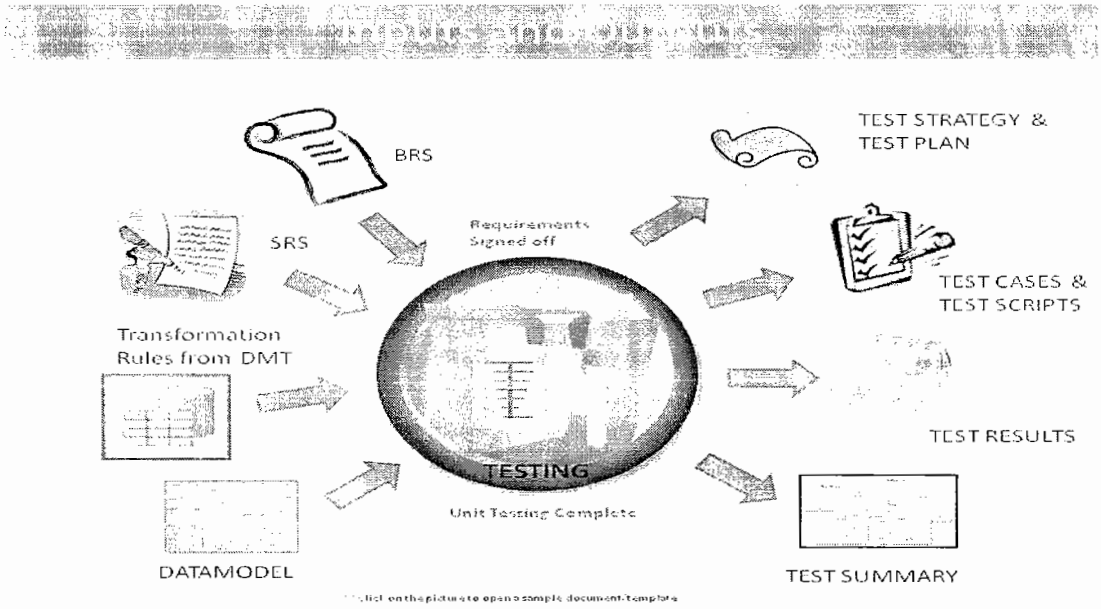
ETL Tester Responsibilities:

- Analysis of the specifications provided by the clients
- Understanding Logical flow of the Application from Business Knowledge
- Participate in reviews (Requirements, Design Documents, Data Model)
- Estimate the effort needed to design and execute the project
- Create and maintain the Test Strategy document
- Design the test cases as per Test Plan and User Specifications.
- Traceability of requirements to test cases
- Define entrance and exit criteria for code and data migration into the test and production environment
- Creating Test Plan, Test Scenarios, Test Cases and Test Scripts to validate source and Target data
- Played a key role in the development of **Teradata SQL queries** while preparing test case scripts.
- Creating, maintaining and executing Re-usable scripts
- Executing the test scripts including prerequisites, detailed instructions and anticipated results
- Analyzing the root cause of the failed Test Case and provide the related data samples to client
- Responsible for reviewing of test plans for team members to ensure high quality deliverables.
- Co-coordinating with the Client and getting proper updates from Onsite.

- Incorporate UAT Test cases in to testing
- Help with the analysis of issues and Data Reconciliation
- Onsite communication with coordinators and client.
- Create, compile, and provide Test Reports and Metrics
- UAT Support - Provide support to Business in during UAT

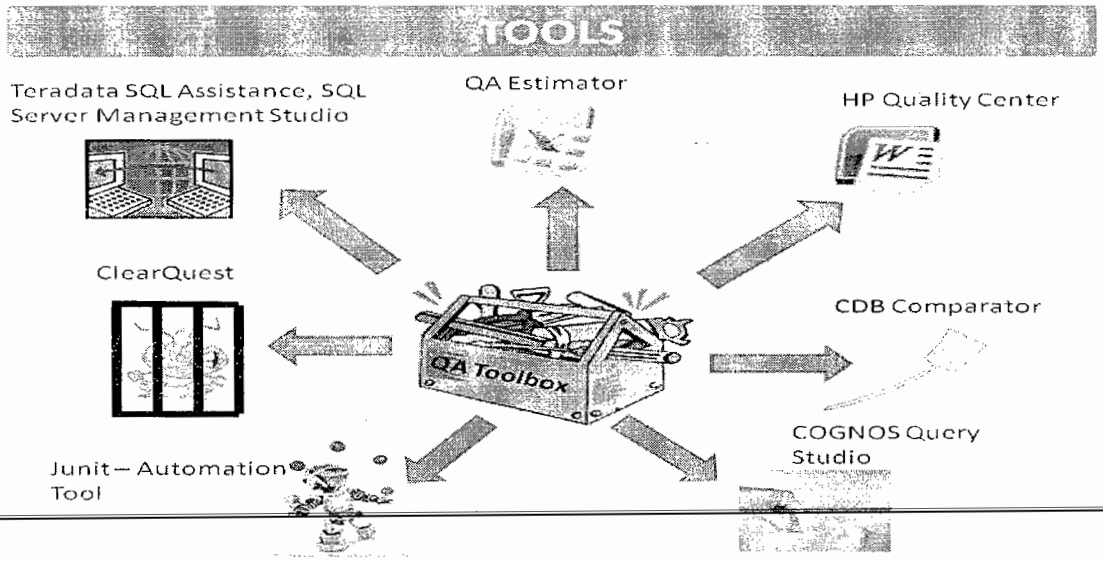
Inputs and Outputs of QA Process:

Based on project requirements these inputs and outputs will change.



Tools which are used in ETL Project:

Based on ETL project TOOLS will change.



Types of Test Cases which will arise in ETL/BI/DWH Project:

There are four different scenarios in ETL project.

1. New table creation
2. Adding new columns to Existing Table
3. Removing columns from Existing Table
4. New View creation
5. Adding columns to Existing View
6. Removing columns from Existing view

1: Metadata Check

Test case Description: QA will validate whether all column names are matched as per PDM document or not.
Expected Result: Column names in table should match as per PDM document.
Actual Result: After test case execution we have to specify actual result
Source SQL: PDM Document will act as Source here
Target SQL: Show table tablename;

2: Data type Check

Test case Description: QA will validate whether all column data types are matched as per PDM document or not.
Expected Result: Data types in table should match as per PDM document.
Actual Result: After test case execution we have to specify actual result
Source SQL: PDM Document will act as Source here
Target SQL: Show table tablename;

3: Constraint Check

Test case Description: QA will validate whether all column contraints are matched as per PDM document or not.
Expected Result: Constraints in table should match as per PDM document.
Actual Result: After test case execution we have to specify actual result
Source SQL: PDM Document will act as Source here
Target SQL: Show table tablename;

4: Duplicate Check (Based on PK/UPI/Natural Key)

Test case Description: QA will validate was there any duplicates for PK/UPI columns
Expected Result: PK/UPI column should not contain any duplicates
Actual Result: After test case execution we have to specify actual result
Source SQL: Select Col1, Count (*) from tablename Group by Col1 Having Count (*)>1
Target SQL: Select Col1, Count (*) from tablename Group by Col1 Having Count (*)>1

5: Duplicate Check (Based on Surrogate Key)

Test case Description: QA will validate was there any duplicates for Surrogate key columns
Expected Result: Surrogate key column should not contain any duplicates
Actual Result: After test case execution we have to specify actual result
Source SQL: Select Col1, Count (*) from tablename Group by Col1 Having Count (*)>1

QUALITY THOUGHT
Phone: 9963486280, 040-40025423
www.QualityThoughtTechnologies.com
Email Id:qthought99@gmail.com

Target SQL: Select Col1, Count (*) from tablename Group by Col1 Having Count (*)>1

6: Null Check

Test case Description: QA will validate was there any columns which are having all values as null.

Expected Result: No column should not contain all values as Null. (If requirement is like that then no issue)

Actual Result: After test case execution we have to specify actual result

Source SQL: Select Col1 from tablename where col1 is not null /*Like this need to check all cols*/

Target SQL: Select Col1 from tablename where col1 is not null /*Like this need to check all cols*/

7: Orphan Check

Test case Description: QA will validate was there any orphans in table.

Expected Result: There should not be any orphan records in table.

Actual Result: After test case execution we have to specify actual result

Source SQL: Select Col1 from ChildTablename
MINUS
Select Col1 from TargetTablename

Target SQL: Select Col1 from ChildTablename
MINUS
Select Col1 from TargetTablename

8: Date Check

Test case Description: QA will validate whether date format is proper or not for all columns.

Expected Result: Date format should be proper for all columns as per requirement doc.

Actual Result: After test case execution we have to specify actual result

Source SQL:

Target SQL:

9: Scenario Check: We need to this check based on business knowledge.

Example 1: Every policy should have only one policy term as open record. We need to write test case to validate this scenario.

Example 2: Load timestamp should be lower than Update time stamp. We need to write test case to validate this scenario.

10: Count Check

Test case Description: QA will validate whether count of records matching from source to target or not as per requirements/transformation rules.

Expected Result: Count of records should match between source and target as per requirement/transformation rule

Actual Result: After test case execution we have to specify actual result

Source SQL: Sel Count (*) from sourcetablename
MINUS

Sel Count (*) from targettablename

Target SQL: Sel Count (*) from targettablename
MINUS
Sel Count (*) from sourcetablename

11: Data Check

Test case Description: QA will validate whether Data matching from source to target or not as per requirements/transformation rules.

Expected Result: Data should match between source and target as per requirement/transformation rule

Actual Result: After test case execution we have to specify actual result

Source SQL: Sel Col1,Col2 from sourcetablename
MINUS
Sel Col1,Col2 from targettablename

Target SQL: Sel Col1,Col2 from targettablename
MINUS Sel Col1,Col2 from sourcetablename

12: Positive/Negative Checks: QA will validate Positive/Negative checks based on Business requirements and Business knowledge.

13: Regression Checks: QA will validate data by using regression scripts to ensure because of adding/removing columns there should not any impact on any other columns.

14: No Truncation Check: Populating the full contents of each field to validate that no truncation occurs at any step in the process. For example, if the source data field is a string (30) make sure to test it with 30 characters.

15: Data Quality :

- Number Check.
  - If in the source format of numbering the columns are as xx_30 but if the target is only 30 then it has to load not pre_fix(xx_) . We need to validate.
- Date Check
  - They have to follow Date format and it should be same across all the records. Standard format : YYYY_MM_DD
- Precision Check
  - Precision value should display as expected in the target table.
  - Example: In source 19.123456 but in the target it should display as 19.123 or round of 20.
- Data Check(Transformation Rule Check)
  - Source SQL must include the transformation logic and complete data must be validated
    By Target SQL
- Null Check
  - Few columns should display "Null" based on business requirement
  - Check for Non Null Columns

Data completeness – Balancing counts between source and target
Referential Integrity – Orphan and foreign key checks
Data Quality – **Selection Criteria**: Validate that ETL correctly excludes, substitutes default values, ignores and reports invalid data.
Data Integrity: Duplicate checks by surrogate key and natural key,
Column Null checks, date checks, and checks like rollups and aggregations
Data transformation – Ensures that all data is transformed correctly according to business rules and/or design specifications.
Integration testing – Ensures that all the ETL processes functions well with together. This is currently done in TAC
Regression testing – Ensures existing functionality remains intact with each release of new code
User-acceptance testing – Ensures the solution meets users' current expectations and anticipates their future expectations.

Test Types:

Test case	Detail
Referential Integrity Check	Validate that there should not be any Orphan
Null/Column Completeness Check	Validate that the Column should not be completely Null
Exclusion Criteria Check	Validate if there is any exclusion criteria provided on the target table.
Date Checks	Validate that the dates are valid and correct format has been loaded
Duplicate Check based on natural Key	Validate that there should not be any duplicate based on natural key
Count Balancing	Matches the Source and Target table count
Data Integrity Check	Validate the check constraints for each column of target
Absolute Duplicate check	Validate that there should not be any duplicate based on the surrogate key
Others Business Scenario Check	Validate the related business scenario
Transformation Rule check	Validate the transformation rule on each column of target table

Load Types and Validations:

In general data will be loaded in target tables based on different frequencies as below.

1. **Daily Loads :** Data will load in target table daily basis
2. **Weekly Loads :** Data will load in target on weekly basis
3. **Monthly loads:** Data will load in to target on Monthly basis.

Other than this we have below types of loads.

Initial Load: When we are loading first time in to target table we can call it as Initial load.

History Load /Full Load/Destructive Load

Also known as Full Load, is a process of completely destroying/deleting the existing data and reloading it from scratch. A lot of unchanged data is also deleted and reloaded in this process. But a destructive load ensures the highest data integrity easily.

"Delete destination data. Read data from source. Load into destination."

Incremental Load

Incremental load is a process of loading data incrementally. Only new and changed data is loaded to the destination. Data that didn't change will be left alone. Data integrity can be ensured in this process too, but ETL can get complicated.

"Read source data. Compare with destination. Filter for new and changed data. Write to destination.".

Full Load vs. Incremental Load

	Destructive Load	**Incremental Load**
How it works	Deletes all rows and reload from scratch.	Only new or updated rows are processed
Time	Requires more time.	Requires less time.
Data Integrity	Can easily be guaranteed	Difficult. ETL must check for new/updated rows.
History	Can be lost.	Retained.

Conclusion

Full load is the easiest way to load daily. But consumes a lot of time and other server resources. Incremental load processes only the new or changed data, so when time is of essence with larger data sets, incremental load is the way to go.

To validate data for all loads we should have to specify date condition in target query.

Examples:

Customer Table

CustomerID	CustomerName	Type	Entry Date
1	John	Individual	22-Mar-2012
2	Ryan	Individual	22-Mar-2012
3	Bakers'	Corporate	23-Mar-2012

Sales Table

ID	CustomerID	ProductDescription	Qty	Revenue	Sales Date
1	1	White sheet (A4)	100	4.00	22-Mar-2012
2	1	James Clip (Box)	1	2.50	22-Mar-2012
3	2	Whiteboard Marker	1	2.00	22-Mar-2012
4	3	Letter Envelop	200	75.00	23-Mar-2012
5	1	Paper Clip	12	4.00	23-Mar-2012

As you can see, above tables store data for 2 consecutive days - 22 Mar and 23 Mar. On 22 Mar, I had only 2 customers (John and Ryan) who made 3 transactions in the sales table. Next day, I have got one more customer (Bakers') and I have recorded 2 transactions - one from Bakers' and 1 from my old customer John.

Also imagine, we have a data warehouse which is loaded everyday in the night with the data from this system.

FULL LOAD METHOD FOR LOADING DATA WAREHOUSE

In case we are to opt for full load method for loading, we will read the 2 source tables (Customer and Sales) everyday in full. So,

On 22 Mar 2012: We will read 2 records from Customer and 3 records from Sales and load all of them in the target.

On 23 Mar 2012: We will read 3 records from customer (including the 2 older records) and 5 records from sales (including 3 old records) and will load or update them in the target data warehouse.

As you can clearly guess, this method of loading unnecessarily read old records that we need not read as we have already processed them before. Hence we need to implement a smarter way of loading.

INCREMENTAL LOAD METHOD FOR LOADING DATA WAREHOUSE

In case of incremental loading, we will only read those records that are not already read and loaded into our target system (data warehouse). That is, on 22 March, we will read 2 records from customer and 3 records from sales - however - on 23 March, we will read 1 record from customer and 2 records from sales.

But how do we ensure that we "only" read those records that are not "already" read? How do we know which records are already read and which records are not?

We can make use of "entry date" field in the customer table and "sales date" field in the sales table to keep track of this. After each loading we will "store" the date until which the loading has been performed in some data warehouse table and next day we only extract those records that has a date greater than our stored date. Let's create a new table to store this date. We will call this table as "Batch"

Batch

Batch_ID Loaded_Until Status
1 22-Mar-2012 Success
2 23-Mar-2012 Success

Once we have done this, all we have to do to perform incremental or delta loading is to rite our data extraction SQL queries in this format:

Customer Table Extraction SQL

```
SELECT t.*
FROM Customer t
WHERE t.entry_date > (select nvl(
                        max(b.loaded_until),
                        to_date('01-01-1900', 'MM-DD-YYYY')
                       )
                from batch b
                where b.status = 'Success');
```

Sales Table Extraction SQL

```
SELECT t.*
FROM Sales t
WHERE t.sales_date > (select nvl(
                        max(b.loaded_until),
                        to_date('01-01-1900', 'MM-DD-YYYY')
                       )
                from batch b
                where b.status = 'Success');
```

On First day (22 Mar):

There wont be any record in our batch table since we have not loaded any batch yet. So "SELECT max(b.loaded_until)" will return NULL. That is why we have put one NVL() function to replace the NULL with a very old historical date - 01 Jan 1900 in this case.
So in the first day, we are asking the select query to extract all the data having entry date (or sales date) greater than 01-Jan-1900. This will essentially extract everything from the table. Once 22 Mar loading is complete, we will make one entry in the batch table (entry 1) to mark the successful extraction of records.

Second Day (23 Mar):

Next day, the query "SELECT max(b.loaded_until)" will return me 22-Mar-2012. So in effect, above queries will reduce to this:

Customer Table Extraction SQL

```
SELECT t.*
FROM Customer t
WHERE t.entry_date > '22-Mar-2012';
```

Sales Table Extraction SQL

```
SELECT t.*
FROM Sales t
WHERE t.sales_date > '22-Mar-2012';
```

As you can understand, this will ensure that only 23-Mar records are extracted from the table thereby performing a successful incremental loading. After this loading is complete successfully, we will make one more entry in the batch table (entry number 2).

Why MAX() is used in the above query?

When we try to load 23 Mar data, there was only one entry in the batch table (that of 22nd). But when we go to load 24th data or any data after that, there will be multiple entries in the batch table. We must take the max of these entries.

Why status field is created in batch table?

This is because it might so happen that 23rd load has failed. So when we start loading again on 24th, we must take into consideration both 23rd data and 24th data.

Batch_ID	Loaded_Until	Status
1	22-Mar-2012	Success
2	23-Mar-2012	Fail
3	24-Mar-2012	Success

In the above case, 23rd batch load was a failure. That is why next day we have selected all the data after 22-Mar (including 23rd and 24th Mar).

Dimensions:

Should we do incremental loading for dimensions?

In a dimensional model, we may perform incremental loading for dimension tables also. One may argue that this won't be necessary as data volume in dimension tables is not as high as the data volumes in the fact tables, hence we can simply do a full load every time.

I personally do not agree to this argument. This is because during the last few years I have seen tremendous growth in the data in dimension tables and things can get quite heavy especially if we are trying to load SCD type 2 dimensions. Anyway, without much ado, let's delve deep.

Standard Method of Loading

Like before, for our purpose we will assume we have the below customer table in our source system from where we need to perform the data loading

CustomerID	CustomerName	Type	LastUpdatedDate
1	John	Individual	22-Mar-2012
2	Ryan	Individual	22-Mar-2012
3	Bakers'	Corporate	23-Mar-2012

As discussed in the previous article, a typical SQL query to extract data incrementally from this source system will be like this:

```
SELECT t.*
FROM Customer t
WHERE t.lastUpdatedDate > (select nvl(
                    max(b.loaded_until),
                    to_date('01-01-1900', 'MM-DD-YYYY')
                   )
                from batch b
                where b.status = 'Success');
```

Here "batch" is a separate table which stores the date until which we have successfully extracted the data.

Batch_ID	Loaded_Until	Status
1	22-Mar-2012	Success
2	23-Mar-2012	Success

Which one to use: "Entry Date" / "Load Date" or "Last Update Date"?

In an incremental load methodology, we should extract the record when it is first created and after that whenever the record is updated. Therefore, we should always look for "last update date" column for extracting records. This is because, "entry date" or "load date" columns in the source systems are not enough to determine if the record is updated in the later point in time.

Often source systems maintain 2 different columns as load_date and last_update_date. When extracting data based on "last update date", ensure that source systems always populate "last updated date" field with "load date" when the record is first created.

What are the benefits of incremental loading of dimension tables?

Once we extract records incrementally based on their last update date, we can compare each record with the target based on their natural keys and determine if the record is a new record or updated record.

However, if we do not extract incrementally (and every time extract all the records from source), then the number of records to compare against target will be much higher resulting into performance degradation. If we are doing incremental loading, records that do not have any change will not come - only new or updatable records will come. But if we are doing full load, everything will come irrespective of any change

FACT TABLES:

METHOD OF LOADING

Generally speaking, incremental loading for Fact tables is relatively easier as, unlike dimension tables, here you do not need to perform any look-up on your target table to find out if the source record already exists in the target or not. All you need to do is to select incremental records from source (as shown below for the case of "sales" table) and load them as it is to target (you may need to perform lookup to dimensional tables to assign respective surrogate keys - but that's a different story).

Like before we will assume we have a "sales" table in the source

Sales Table

ID	CustomerID	ProductDescription	Qty	Revenue	Sales Date
1	1	White sheet (A4)	100	4.00	22-Mar-2012
2	1	James Clip (Box)	1	2.50	22-Mar-2012
3	2	Whiteboard Marker	1	2.00	22-Mar-2012
4	3	Letter Envelop	200	75.00	23-Mar-2012
5	1	Paper Clip	12	4.00	23-Mar-2012

Given this table, a typical extraction query will be like this:

```
SELECT t.*
FROM Sales t
WHERE t.sales_date > (select nvl(
                        max(b.loaded_until),
```

```
                to_date('01-01-1900', 'MM-DD-YYYY')
              )
        from batch b
        where b.status = 'Success');
```

where "batch" is a separate table maintained at target system having minimal structure and data like below

Batch_ID	Loaded_Until	Status
1	22-Mar-2012	Success
2	23-Mar-2012	Success

However, things may get pretty complicated if your fact is a special type of fact called "snapshot fact". Let's understand them below

Loading Incident Fact

Incident fact is the normal fact that we encounter mostly (and that we have seen above in our sales table example). Records in these types of facts are only loaded if there are transactions coming from the source. For example, if at all there is one sale that happens in the source system, then only a new sales record will come. They are dependent on some real "incident" to happen in the source hence the name incident fact.

Loading Snapshot Fact

As opposed to incident fact, snapshot facts are loaded even if there is no real business incident in the source. Let me show you what I mean by using the above example of customer and sales tables in OLTP. Let's say I want to build a fact that would show me total revenue of sales from each customer for each day. In effect, I want to see the below data in my fact table.

Sales fact table (This is what I want to see in my target fact table)

Date	Customer	Revenue
22-Mar-2012	John	6.50
22-Mar-2012	Ryan	2.00
23-Mar-2012	John	10.50
23-Mar-2012	Ryan	2.00
23-Mar-2012	Bakers'	75.00

As you see, even if no sales was made to Ryan on 23-Mar, we still show him here with the old data. Similarly for John, even if goods totaling to $4.00 was sold to him on 23-Mar, his record shows the cumulative total of $10.50.

Now obviously the next logical question is how to load this fact using incremental loading? Because incremental loading only brings in incremental data - that is on 23rd March, we will only have Bakers' and John's records and that too with that day's sales figures. We won't have Ryan record in the incremental set.

Why not a full load

You can obviously opt-in for full load mechanism as that would solve this problem but that would take the toll on your loading performance.

Then what's the solution?

One way to resolve this issue is: creating 2 incremental channels of loading for the fact. 1 channel will bring in incremental data from source and the other channel will bring in

incremental data from the target fact itself. Let's see how does it happen below. We will take the example for loading 23-Mar data.

Channel 1: Incremental data from source
Customer Revenue
John 4.00
Bakers' 75.00

Channel 2: Incremental data from target fact table (last day's record)
Customer Revenue
John 6.50
Ryan 2.00

Next we can perform a FULL OUTER JOIN between the above two sets to come to below result
John 10.50
Ryan 2.00
Bakers' 75.00

ETL Testing Techniques:
1) Verify that data is transformed correctly according to various business requirements and rules.
2) Make sure that all projected data is loaded into the data warehouse without any data loss and truncation.
3) Make sure that ETL application appropriately rejects, replaces with default values and reports invalid data.
4) Make sure that data is loaded in data warehouse within prescribed and expected time frames to confirm improved performance and scalability.
Apart from these 4 main ETL testing methods other testing methods like integration testing and user acceptance testing is also carried out to make sure everything is smooth and reliable.
ETL **Testing Challenges**:
ETL testing is quite different from conventional testing. There are many challenges we faced while performing data warehouse testing. Here is the list of few ETL testing challenges I experienced on my project:
- Incompatible and duplicate data.
- Loss of data during ETL process.
- Unavailability of inclusive test bed.
- Testers have no privileges to execute ETL jobs by their own.
- Volume and complexity of data is very huge.
- Fault in business process and procedures.
- Trouble acquiring and building test data.
- Missing business flow information.

Defect Life Cycle In ETL Project:

The different states of a Defect can be summarized as follows:

1. New
2. Open
3. Assign
4. Test
5. Verified
6. Deferred
7. Reopened
8. Duplicate
9. Rejected and
10. Closed

Description of Various Stages:

1. New: When the Defect is posted for the first time, its state will be "NEW". This means that the Defect is not yet approved.

2. Open: After a tester has posted a Defect, the lead of the tester approves that the Defect is genuine and he changes the state as "OPEN".

3. Assign: Once the lead changes the state as "OPEN", he assigns the Defect to corresponding developer or developer team. The state of the Defect now is changed to "ASSIGN".

4. Test: Once the developer fixes the Defect, he has to assign the Defect to the testing team for next round of testing. Before he releases the software with Defect fixed, he changes the state of Defect to "TEST". It specifies that the Defect has been fixed and is released to testing team.

5. Deferred: The Defect, changed to deferred state means the Defect is expected to be fixed in next releases. The reasons for changing the Defect to this state have many factors. Some of them are priority of the Defect may be low, lack of time for the release or the Defect may not have major effect on the software.

6. Rejected: If the developer feels that the Defect is not genuine, he rejects the Defect. Then the state of the Defect is changed to "REJECTED".

7. Duplicate: If the Defect is repeated twice or the two Defects mention the same concept of the Defect, then one Defect status is changed to "DUPLICATE".

8. Verified: Once the Defect is fixed and the status is changed to "TEST", the tester tests the Defect. If the Defect is not present in the software, he approves that the Defect is fixed and changes the status to "VERIFIED".

9. Reopened: If the Defect still exists even after the Defect is fixed by the developer, the tester changes the status to "REOPENED". The Defect traverses the life cycle once again.

10. Closed: Once the Defect is fixed, it is tested by the tester. If the tester feels that the Defect no longer exists in the software, he changes the status of the Defect to "CLOSED". This state means that the Defect is fixed, tested and approved.

While defect prevention is much more effective and efficient in reducing the number of defects, most organization conducts defect discovery and removal. Discovering and removing

defects is an expensive and inefficient process. It is much more efficient for an organization to conduct activities that prevent defects.

Guidelines on deciding the Severity of Defect:

Indicate the impact each defect has on testing efforts or users and administrators of the application under test. This information is used by developers and management as the basis for assigning priority of work on defects.

A sample guideline for assignment of Priority Levels during the product test phase includes:

1. Critical / Show Stopper — An item that prevents further testing of the product or function under test can be classified as Critical Defect. No workaround is possible for such Defects. Examples of this include a missing menu option or security permission required to access a function under test.

2. Major / High — A defect that does not function as expected/designed or cause other functionality to fail to meet requirements can be classified as Major Defect. The workaround can be provided for such Defects. Examples of this include inaccurate calculations; the wrong field being updated, etc.

3. Average / Medium — The defects which do not conform to standards and conventions can be classified as Medium Defects. Easy workarounds exists to achieve functionality objectives. Examples include matching visual and text links which lead to different end points.

4. Minor / Low — Cosmetic defects which does not affect the functionality of the system can be classified as Minor Defects.