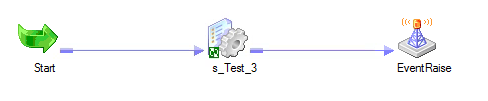
Types of tasks in Informatica 9.6.1:

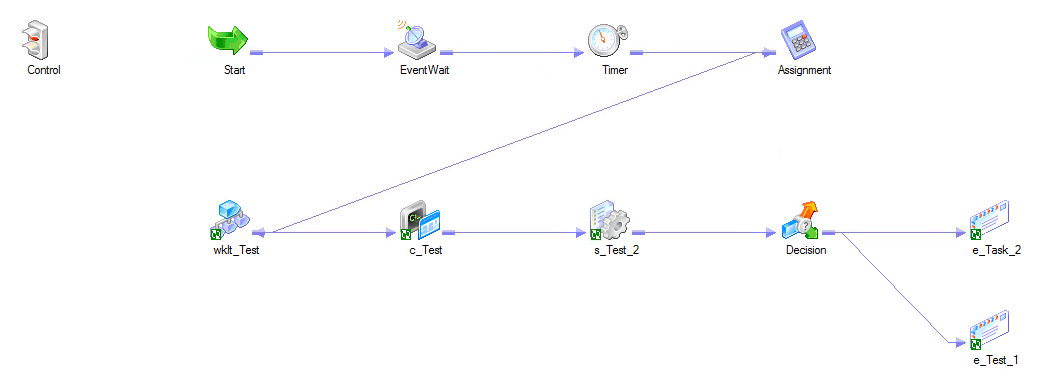
1. Session
2. Command
3. Decision
4. Event Wait
5. Event raise
6. Email
7. Timer
8. Assignment
9. Control
10. Worklet

**Types of Tasks in Informatica**

There are different types of tasks in Informatica Workflow manager which we use while running a workflow. I have listed it below.

|  |  |
| --- | --- |
| **Tasks** | **Description** |
| Assignment | Used to assign a value to a workflow variable |
| Command | Used to run a shell command during the workflow |
| Control | Used to stop or abort the workflow |
| Decision | Tells a condition to evaluate |
| Email | Used to send email during the workflow |
| Event-Raise | Notifies the Event-Wait task that an event has occurred |
| Event-Wait | It waits for the event to completed in order to start the next task |
| Session | Used to run the mapping created in Designer buy linking to session |
| Timer | It waits for an already timed event to start |





**Change data capture** (**CDC**)

It is the process of capturing changes made at the **data** source and applying them throughout the enterprise. **CDC** minimizes the resources required for ETL (extract, transform, load) processes because it only deals with **data** changes. The goal of **CDC** is to ensure **data** synchronicity.

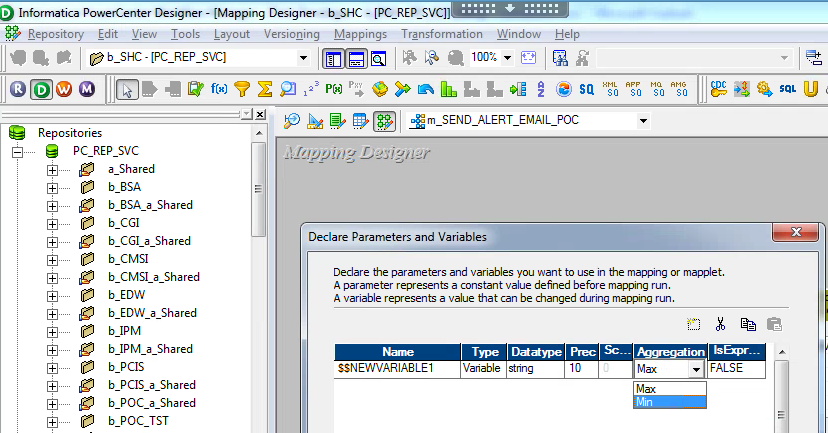
Difference b/w Snowflake schema and star schema

100 records, load to same target with 10 records at a time.

Normalization

Mapping parameters and variables.

How does aggregation happen in mapping variables?



**Snowflake Schema vs. Star Schema**

# When choosing a database schema for a data warehouse, **snowflake** and **star schemas** tend to be popular choices. This comparison discusses suitability of star vs. snowflake schemas in different scenarios and their characteristics.

## Comparison chart

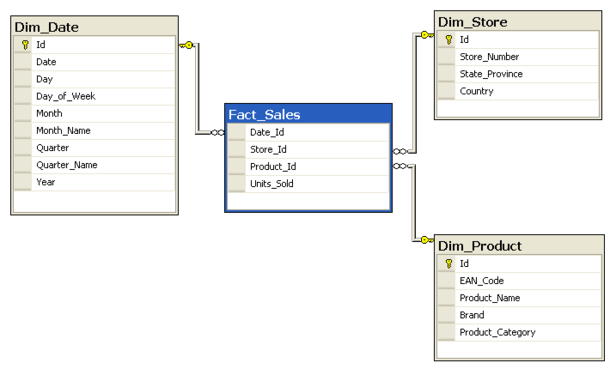
| Snowflake Schema versus Star Schema comparison chart | | |
| --- | --- | --- |
|  | **Snowflake Schema** | **Star Schema** |
| **Ease of maintenance / change** | No redundancy, so snowflake schemas are easier to maintain and change. | Has redundant [data](http://www.diffen.com/difference/Data_vs_Information) and hence less easy to maintain/change |
| **Ease of Use** | More complex queries and hence less easy to understand | Lower query complexity and easy to understand |
| **Query Performance** | More foreign keys and hence longer query execution time (slower) | Less number of foreign keys and hence shorter query execution time (faster) |
| **Type of Data warehouse** | Good to use for data warehouse core to simplify complex relationships (many : many) | Good for DataMart with simple relationships (1:1 or 1:many) |
| **Joins** | Higher number of [Joins](http://www.diffen.com/difference/Inner_Join_vs_Outer_Join) | Fewer [Joins](http://www.diffen.com/difference/Inner_Join_vs_Outer_Join) |
| **Dimension table** | A snowflake schema may have more than one dimension table for each dimension. | A star schema contains only single dimension table for each dimension. |
| **When to use** | When dimension table is relatively big in size, snowflaking is better as it reduces space. | When dimension table contains less number of rows, we can choose Star schema. |
| **Normalization/ De-Normalization** | Dimension Tables are in Normalized form but Fact Table is in De-Normalized form | Both Dimension and Fact Tables are in De-Normalized form |
| **Data model** | Bottom up approach | Top down approach |

## Examples

Consider a database for a retailer that has many stores, with each store selling many products in many product categories and of various brands. A data warehouse or data mart for such a retailer would need to provide analysts the ability to run sales reports grouped by store, date (or month, quarter or year), or product category or brand.

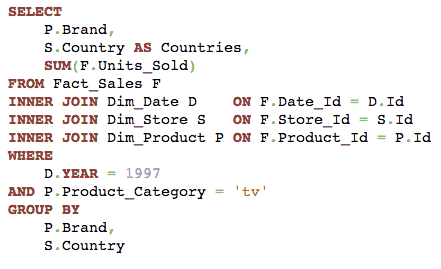
### Star Schema Example

If this data mart was using a star schema, it would look as follows:

[](http://www.diffen.com/difference/Image:Star-schema-4.png)

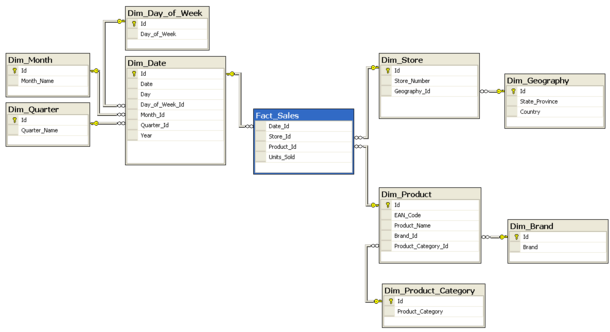
The fact table would be a record of sales transactions, while there are dimension tables for date, store and product. Dimension tables are each connected to the fact table via their primary key, which is a foreign key for the fact table. For example, instead of storing the actual transaction date in a row of the fact table, the date\_id is stored. This date\_id corresponds to a unique row in the Dim\_Date table, and that row also stores other attributes of the date that are required for grouping in reports. e.g., day of the week, month, quarter of the year and so on. The data is denormalized for easier reporting.

Here is how one would get a report of number of televisions sold by brand and by country with the help of [inner joins](http://www.diffen.com/difference/Inner_Join_vs_Outer_Join).

[](http://www.diffen.com/difference/Image:Star-schema-SQL-query.png)

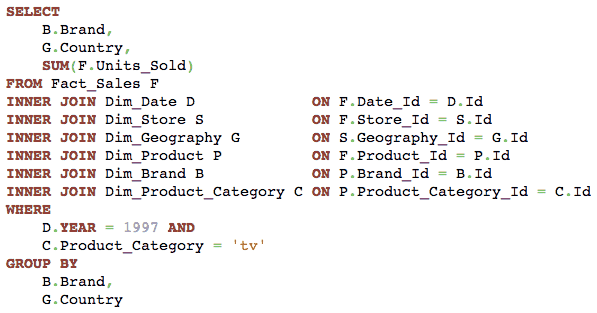
### Snowflake Schema Example

The same scenario can also use a snowflake schema, in which case it would be structured as follows:

[](http://www.diffen.com/difference/Image:Snowflake-schema-4.png)

The main difference, when compared with the star schema, is that data in dimension tables is more [normalized](https://en.wikipedia.org/wiki/Database_normalization). For example, instead of storing month, quarter and day of the week in each row of the Dim\_Date table, these are further broken out into their own dimension tables. Similarly for the Dim\_Store table, the state and country are geographical attributes that are one step removed — instead of being stored in the Dim\_Store table, they are now stored in a separate Dim\_Geography table.

The same report — the number of televisions sold by country and by brand — is now a little more complicated than in a star schema:

[](http://www.diffen.com/difference/Image:Snowflake-schema-SQL-query.png)

*SQL query to get number of products sold by country and brand, when the database uses a snowflake schema.*

**Snowflake vs. Star Schemas: Which Should You Use?**

Now that we’ve looked at theory and query speeds, let’s get right into the heart of the matter: how do you know which schema to use on any given project?

Consider using the **snowflake schema**:

* **In data warehouses.** As the warehouse is Data Central for the company, we could save lot of space this way.
* **When dimension tables require a significant amount of storage space.** In most cases, the fact tables will be the ones that take most of the space. They’ll probably also grow much faster than dimension tables. But there are certain situations where that doesn’t apply. For instance, the dimension tables could contain a lot of redundant-but-needed attributes. In our example, we used the city attribute to describe the city where the store is located. What if we wanted a much more detailed description of the city, including the population, postal code, demographic data, etc.? Describing other subdimensions – for example, store, region, state and country – with more attributes would turn the **dim\_store** dimension table into one large table with a lot of redundancy.
* **If you use tools that require a snowflake schema in the background. (Fortunately, most modern tools support both schemas and even the galaxy schema.)**

Consider using the **star schema**:

* **In data marts.** Data marts are subsets of data taken out of the central data warehouse. They are usually created for different departments and don’t even contain all the history data. In this setting, saving storage space is not a priority.

On the other hand, the star schema does simplify analysis. This is not just about query efficiency but also about simplifying future actions for business users. They may understand databases and know how to write queries, but why complicate things and include more joins if we can avoid it? A business user could have a template query that joins the fact table with all the dimension tables. Then they only need to add the appropriate selections and groupings. (This approach is close to Excel’s pivot tables.)

* **If you use tools that require a star schema in the background. (Again, this usually isn’t an issue.)**

Both the star schema and the snowflake schema are relational models used to organize data warehouses and/or data marts. No matter how similar they are, they demonstrate two different approaches and have their own benefits and disadvantages. Personally, I would go with the snowflake schema when implementing a data warehouse (to save storage space) and with the star schema for data marts (to make life easier for business users).

**Normalization of Database**

Database Normalisation is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

* Eliminating reduntant(useless) data.
* Ensuring data dependencies make sense i.e data is logically stored.

Normalization Rule

Normalization rule are divided into following normal form.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF

#### First Normal Form (1NF)

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary key** is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

**Student Table :**

|  |  |  |
| --- | --- | --- |
| **Student** | **Age** | **Subject** |
| Adam | 15 | Biology, Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

**Student Table following 1NF will be :**

|  |  |  |
| --- | --- | --- |
| **Student** | **Age** | **Subject** |
| Adam | 15 | Biology |
| Adam | 15 | Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

#### Second Normal Form (2NF)

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form**.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is {**Student**, **Subject**}, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

**New Student Table following 2NF will be :**

|  |  |
| --- | --- |
| **Student** | **Age** |
| Adam | 15 |
| Alex | 14 |
| Stuart | 17 |

In Student Table the candidate key will be **Student** column, because all other column i.e **Age** is dependent on it.

**New Subject Table introduced for 2NF will be :**

|  |  |
| --- | --- |
| **Student** | **Subject** |
| Adam | Biology |
| Adam | Maths |
| Alex | Maths |
| Stuart | Maths |

In Subject Table the candidate key will be {**Student**, **Subject**} column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

#### Third Normal Form (3NF)

**Third Normal form** applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. Therefore, this ***transitive functional dependency* should be removed** from the table and the table must be in **Second Normal form**. For example, consider a table with following fields.

**Student\_Detail Table :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student\_id** | **Student\_name** | **DOB** | **Street** | **city** | **State** | **Zip** |

In this table Student\_id is Primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called **transitive dependency**. Hence to apply **3NF**, we need to move the street, city and state to new table, with **Zip** as primary key.

**New Student\_Detail Table :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_id** | **Student\_name** | **DOB** | **Zip** |

**Address Table :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Zip** | **Street** | **city** | **state** |

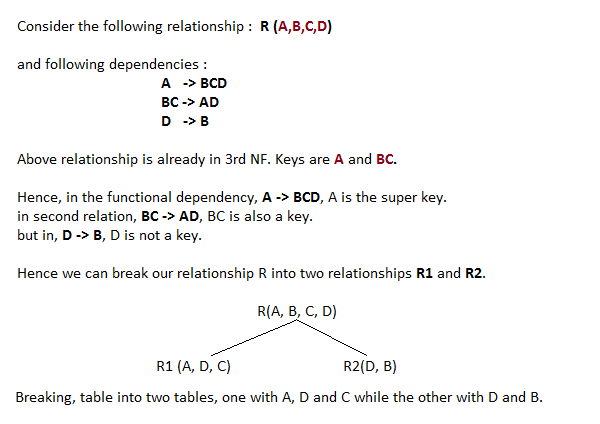
The advantage of removing transtive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.

#### Boyce and Codd Normal Form (BCNF)

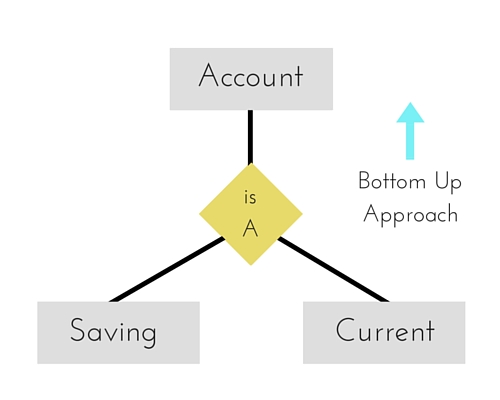
**Boyce and Codd Normal Form** is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

* R must be in 3rd Normal Form
* and, for each functional dependency ( X -> Y ), X should be a super Key.



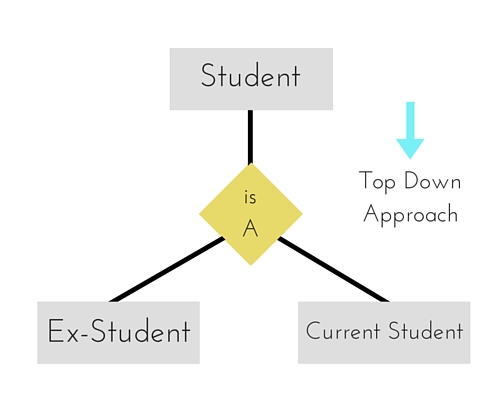
#### Generalization

**Generalization** is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity.



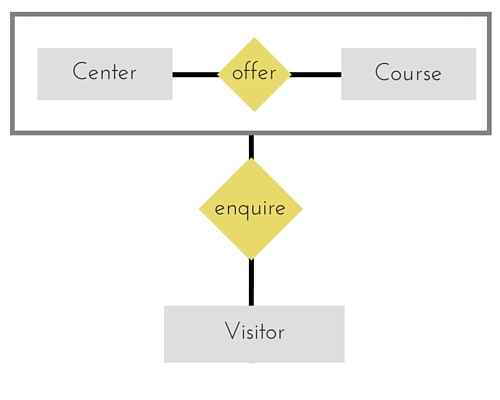
#### Specialization

**Specialization** is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, some higher level entities may not have lower-level entity sets at all.



#### Aggregration

Aggregration is a process when relation between two entity is treated as a single entity. Here the relation between Center and Course, is acting as an Entity in relation with Visitor.



# [**Change Data Capture (CDC) Made Easy Using Mapping Variables**](http://www.disoln.org/2012/10/change-data-capture-cdc-made-easy-using-mapping-variables.html)

At times we may need to implement [Change Data Capture](http://www.disoln.org/2013/03/Change-Data-Capture-Implementation-for-Multi-Sourced-ETL-Process.html) for small data integration projects which includes just couple of workflows.  Introducing a [Change Data Capture framework](http://www.disoln.org/2012/10/An-ETL-Framework-for-Change-Data-Capture-CDC.html) for such project is not a recommended way to handle this, just because of the efforts required to build the framework may not be justified. Here in this article lets discuss about a simple, easy approach handle Change Data Capture.

We will be using Informatica Mapping Variables to building our Change Data Capture logic. Before even we talk about the implementation, lets understand the Mapping Variable

## **Informatica Mapping Variable**

### What is Mapping Variable

These are variables created in PowerCenter Designer, which you can use in any expression in a mapping, and you can also use the mapping variables in a source qualifier filter, user-defined join, or extract override, and in the Expression Editor of reusable transformations.

### Mapping Variable Starting Value

Mapping variable can take the starting value from

* 1. Parameter file
  2. Pre-session variable assignment
  3. Value saved in the repository
  4. Initial value
  5. Default Value

The Integration Service looks for the start value in the order mentioned above. Value of the mapping variable can be changed with in the session using an expression and the final value of the variable will be saved into the repository. The saved value from the repository is retrieved in the next session run and used as the session start value.

### Setting Mapping Variable Value

You can change the mapping variable value with in the mapping or session using the Set Function. We need to use the set function based on the Aggregation Type of the variable. Aggregation Type of the variable can be set when the variable is declared in the mapping.

* **SetMaxVariable.** Sets the variable to the maximum value of a group of values. To use the SetMaxVariable with a mapping variable, the aggregation type of the mapping variable must be set to Max.
* **SetMinVariable.**Sets the variable to the minimum value of a group of values. use the SetMinVariable with a mapping variable, the aggregation type of the mapping variable must be set to Min.
* **SetCountVariable.**Increments the variable value by one. In other words, it adds one to the variable value when a row is marked for insertion, and subtracts one when the row is marked for deletion. To use the SetCountVariable with a mapping variable, the aggregation type of the mapping variable must be 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.

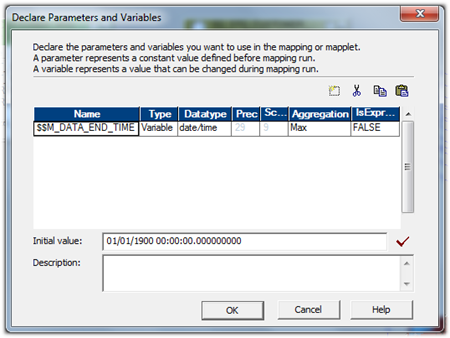
## **Change Data Capture Implementation**

Now we understand the mapping variables, lets go ahead and start building our mapping with [Change Data Capture](http://www.disoln.org/2013/03/Change-Data-Capture-Implementation-for-Multi-Sourced-ETL-Process.html).

Here we are going to implement [Change Data Capture](http://www.disoln.org/2012/12/Change-Data-Capture-CDC-Implementation-Using-CHECKSUM-Number.html) for CUSTOMER data load. We need to load any new customer or changed customers data to a flat file. Since the column UPDATE\_TS value changes for any new or updated customer record, we will be able to find the new or changed customer records using UPDATE\_TS column.

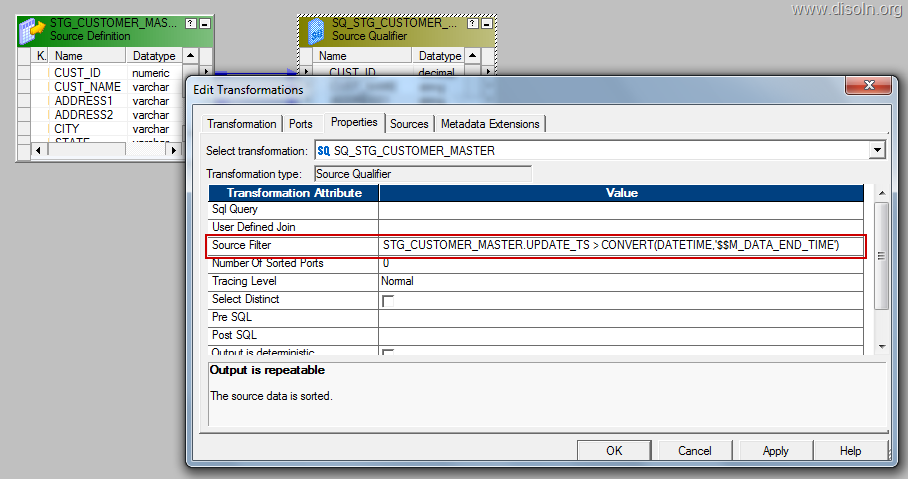
As the first step lets start the mapping and create a mapping variable as shown in below image.

* + $$M\_DATA\_END\_TIME as Date/Time

[](http://lh6.ggpht.com/-_sHvDtOQOIM/UHes3ESSEjI/AAAAAAAAFp8/cA8W_xx8ELE/s1600-h/image%5b11%5d.png)

Now bring in the source and source qualified to the mapping designer workspace. Open the source qualifier and give the filter condition to get the latest data from the source as shown below.

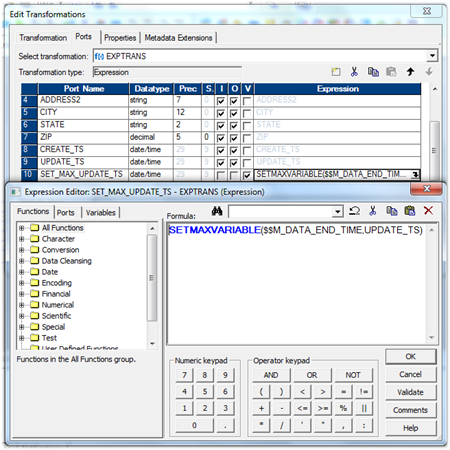
* + STG\_CUSTOMER\_MASTER.UPDATE\_TS > CONVERT(DATETIME,'$$M\_DATA\_END\_TIME')



Note : This filter condition will make sure that, latest data is pulled from the source table each and every time. Latest value for the variable $M\_DATA\_END\_TIME is retrieved from the repository every time the session is run.

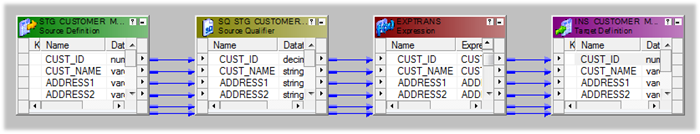
Now map the column UPDATE\_TS to an expression transformation and create a variable expression as below.

* + SETMAXVARIABLE($M\_DATA\_END\_TIME,UPDATE\_TS)

[](http://lh5.ggpht.com/-wkXWeSOXjU0/UHes30RqY7I/AAAAAAAAFqM/tK8ZTh6CFnU/s1600-h/image%5b20%5d.png)

Note : This expression will make sure that, latest value from the the column UPDATE\_TS is stored into the repository after the successful completion of the session run.

Now you can map all the remaining columns to the down stream transformation and complete all other transformation required in the mapping.

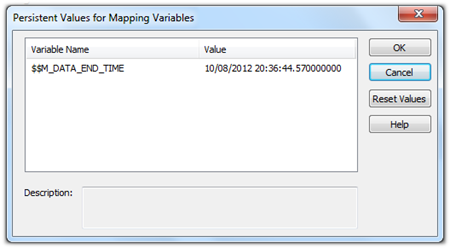
[](http://lh3.ggpht.com/-bqC0GTJxHxU/UHes4_NIJfI/AAAAAAAAFqc/IHzZOZq_gcY/s1600-h/image%5b25%5d.png)

That’s all you need to configure Change Data Capture, Now create your workflow and run the workflow.

Once you look into the session log file you can see the mapping variable value is retrieved from the repository and used in the source SQL, just like shown in the image below.

[http://lh6.ggpht.com/-boEI6Hod5wo/UHes6ZXxL9I/AAAAAAAAFq0/vFnBgzKrMmc/image_thumb%25255B15%25255D.png?imgmax=800](http://lh4.ggpht.com/-f79dyLcritU/UHes5y55zII/AAAAAAAAFqs/lKfOaVYlUfs/s1600-h/image%5b29%5d.png)

You can look at the mapping variable value stored in the repository, from workflow manager. Choose the session from the workspace, right click and select 'View Persistent Value'. You get the mapping variable in a pop up window, like shown below.

[](http://lh5.ggpht.com/--J6ZUQepFhM/UHes6txLX7I/AAAAAAAAFq8/cTRXl0Fs7Wk/s1600-h/image%5b7%5d.png)

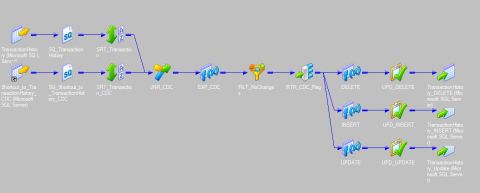
**Change Data Capture – Informatica Mapping Logic for CDC Implementation**

So, Finally here I go with an article on CDC (Change Data Capture) implementation through an informatica which had been a long waiting from my side to be posted.

Requirement – To track on the data change in the source system with the earlier data pulled into the warehouse and to track on Updates, new Inserts and Delete on the data that is no more in the source. I would continue on this CDC implementation with an another article on loading of data forward into a table to save the change history for the data user to keep a track on the change history.

**Mapping Design:**

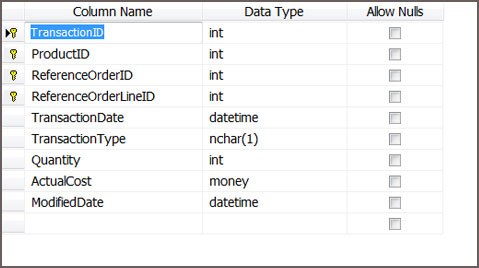
The mapping I have come up with for the CDC logic would be as below,

[](https://mahaveersingh.files.wordpress.com/2014/10/cdc_data_load_img21.jpg)

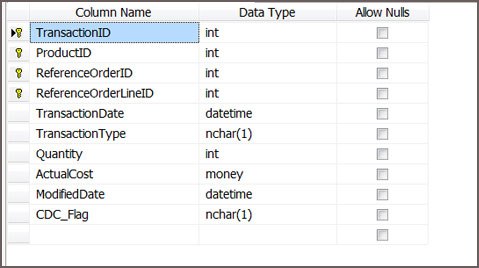
(Click on the image for a better view)

The mapping would involve the Staging/source table which would be a daily truncate load from the source system as is with the current status on the transactions from a source. And the data are then to be sorted with respect to the primary key columns, sample table used would be as below. The sample implies on the target CDC table to have the same composite primary key columns.

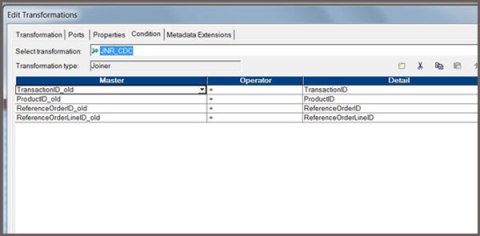
**Source Table:**

[](https://mahaveersingh.files.wordpress.com/2014/10/cdc_data_load_img3.jpg)

**Target Table:**

[](https://mahaveersingh.files.wordpress.com/2014/10/cdc_data_load_img4.jpg)

The mapping would involve a simple logic to have a joiner (Full Outer Join) with the source table and the Target CDC table and to check for the matching and non matching records on the Primary key columns as is illustrated below,

[](https://mahaveersingh.files.wordpress.com/2014/10/cdc_data_load_img5.jpg)

This being a Full Outer Join, enables the loading of all matching and non matching transactions to flow through and then the following three expressions should be defined to check on the Change type,

Flag\_DELETE:

**ISNULL**(TransactionID) AND **ISNULL**(ProductID) AND **ISNULL**(ReferenceOrderID) and **ISNULL**(ReferenceOrderLineID)

Flag\_INSERT:

ISNULL(TransactionID\_old) AND ISNULL(ProductID\_old) AND ISNULL(ReferenceOrderID\_old) AND ISNULL(ReferenceOrderLineID\_old)

Flag\_UPDATE:

TransactionID=TransactionID\_old AND

ProductID=ProductID\_old AND

ReferenceOrderID=ReferenceOrderID\_old AND

ReferenceOrderLineID=ReferenceOrderLineID\_old AND

(TransactionDate != TransactionDate\_old OR

TransactionType != TransactionType\_old OR

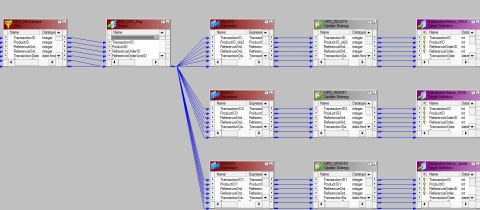
Quantity != Quantity\_old OR

**ABS**(ActualCost – ActualCost\_old)>1)

Flag => IIF(v\_Flag\_Update=1,-1,IIF(v\_Flag\_Insert=1,-2,IIF(v\_Flag\_Delete=1,-3,0)))

where ‘0’ is for no change records, and then to use a filter transformation to filter out all the no change transactions with Flag=0.

Then a Router to define the grouping as per the result of the above expression on checking if the Transaction is an INSERT, UPDATE or DELETE.

[](https://mahaveersingh.files.wordpress.com/2014/10/cdc_data_load_img11.jpg)

(Click on the image for a better view)

And the Update strategy transformation for each to perform and Update or Insert for each, and to note even for DELETE flags I will be going for a DD\_UPDATE, which would update the CDC flag for the transactions not available in the source system to ‘D’ in the target table.

Also to note, that the Delete flow, would involve pulling the ports of CDC source table from the Router to the target and Update on the key column matching records with CDC\_Flag=’D’.

And finally to make the session level changes for each target to define ‘Update as Update’ or ‘Update as Insert’ for the targets respectively. Upon running the mapping for the first time all transactions would be tagged as INSERT and then the proceeding runs will then depend on the source system changes to be captured through the CDC logic of our mapping.

Note – The DELETE transactions are the one which are no more in the source table, and hence these once after being tagged as ‘D’ for the CDC\_Flag can be deleted from the target CDC table as well after making use of the change on to the History tracking table which will be covered more in detail with the next article shortly.

# Deployment Groups in Informatica

Deployment group is a global object which consists of objects from one or more folders. Deployment group is used to copy objects from one or more folders to another folder or repository. You can create, edit, or delete deployment groups. You can copy a deployment group and the objects in the deployment group to a target repository.

**Deployment Group Tasks**:

You can do the following tasks when working with deployment groups:

* **Create deployment group**: You can create a global object for deploying objects from one or more folders.
* **Edit deployment group**: You can modify a deployment group. You can change the static deployment group to dynamic deployment group and vice versa.
* **Configure privileges and permissions**: Configure permissions on a deployment group and the privilege to copy a deployment group.
* **View the objects in a static or dynamic deployment group**: You can preview the objects that the Repository Service will deploy.
* **Add or remove objects in a static deployment group**: You can specify what objects belong to a static deployment group. You can add or remove the objects from the deployment group.
* **Associate a query with a dynamic deployment group**: Assign a query to a deployment to dynamically update the objects that the group contains.
* **View the history of a deployment group**: View the history of a deployment group, including the source and target repositories, deployment date, and user who ran the deployment.
* **Post-deployment validation**: Validate the objects in the target repository after you copy a deployment group to verify that the objects and dependent objects are valid.
* **Roll back a deployment group**: You can roll back a deployment group to purge deployed versions of objects from the target repository.

# Create Deployment Group in Informatica

In informatica, there are two types of deployment groups. They are:

* **Static Deployment Group**: You have to add objects manually. Create static deployment group when the objects in the group are not going to change.
* **Dynamic Deployment Group**: Object query is used to create the deployment group. Create dynamic group if the objects in the group are going to change frequently.

### Creating Deployment Group:

Use the Deployment Group Editor to create and edit deployment groups.   
  
Follow the below steps for creating a deployment group:

* Login to the Repository Manager. Click on Tools > Deployment > Groups to view the existing deployment groups in the Deployment Group Browser.
* Click New to create the deployment group in the Deployment Group Editor. Enter a name for the deployment group.
* Select whether to create a static or dynamic deployment group.
* If you are creating a dynamic deployment group, click Queries to select a query from the Query Browser, and then click Close to return to the Deployment Group Editor.
* Optionally, you can enter a comment for the deployment group.
* Click OK.

After you create a deployment group, the new deployment group appears in the Deployment Groups node in the Navigator of the Repository Manager. If you have created a static deployment group, you can add objects to it.

# PMCMD Command Usage in Informatica

Informatica provides four built-in command line programs or utilities to interact with the informatica features. They are:

* infacmd
* infasetup
* pmcmd
* pmrep

PMREP:

pmrep ObjectExport -n objName -o objType -f FolderName -m -s -b -r -u objName.xml

ObjectExport exports objects by name, to export multiple objects from a folder you can either supply names one at a time, re-executing the command for each name or you can supply an input file containing a list of object names. You also must supply the object name's folder (-f). The last parameter is the output file name.

ObjectExport creates XML that can be imported back into PowerCenter. The switches -m -s -b –r are required to get dependent objects (tables, shortcuts, non-reusable objects) for the object you are exporting. For some objects, if you do not export its dependent objects, it will not re-import into the repository correctly.

PMCMD:

The pmcmd is a command line utility provided by the informatica to perform the following tasks.

* Start workflows.
* Start workflow from a specific task.
* Stop, Abort workflows and Sessions.
* Schedule the workflows.

**How to use PMCMD Command in Informatica**:   
  
1. Scheduling the workflow   
  
The pmcmd command syntax for scheduling the workflow is shown below: 

pmcmd scheduleworkflow -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name

You cannot specify the scheduling options here. This command just schedules the workflow for the next run.   
  
2. Start workflow   
  
The following pmcmd command starts the specified workflow: 

pmcmd startworkflow -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name

3. Stop workflow   
  
Pmcmd command to stop the infromatica workflow is shown below: 

pmcmd stopworkflow -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name

4. Start workflow from a task   
  
You can start the workflow from a specified task. This is shown below: 

pmcmd startask -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name -startfrom task-name

5. Stopping a task.   
  
The following pmcmd command stops the specified task instance: 

pmcmd stoptask -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name task-name

6. Aborting workflow and task.   
  
The following pmcmd commands are used to abort workflow and task in a workflow: 

pmcmd abortworkflow -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name

pmcmd aborttask -service informatica-integration-Service -d domain-name -u user-name -p password -f folder-name -w workflow-name task-name

Usage: pmcmd **startworkflow**

                           <<-service|-sv> service <-domain|-d> domain [<-timeout|-t> timeout]>

                           [<<-user|-u> username|<-uservar|-uv> userEnvVar>]

                           [<<-password|-p> password|<-passwordvar|-pv> passwordEnvVar>]

                           [<<-usersecuritydomain|-usd> usersecuritydomain|<-usersecuritydomainvar|-usdv> userSecuritydomainEnvVar>]

                           [<-folder|-f> folder]

                           [<-startfrom> taskInstancePath [<-recovery|-norecovery>]]

                           [<-paramfile> paramfile]

                           [<-localparamfile|-lpf> localparamfile]

                           [<-osprofile|-o> OSProfile] [-wait|-nowait]

                           [<-runinsname|-rin> runInsName] workflow

INFASETUP:

Command line program "infasetup" can be used for Domain and Node management.  
Here are some of the commands.

* BackupDomain
* DefineDomain
* DefineGatewayNode
* DeleteDomain
* RestoreDomain
* UpdateGatewayNode

Example -

cd /opt/Informatica/9.5.1/server  
./infasetup.sh updateGatewayNode -databaseaddress 172.20.4.117:1521  
  
*New Database host - 172.20.4.117*  
*New Database Port – 1521*

INFACMD:

To disable an Informatica service use the below command:(The variables starting with $ should be defined in your environment or in your script)  
  
*$INFA\_HOME/server/bin/infacmd.sh isp disableService ­dn $DOMAIN\_NAME ­un $DOMAIN\_USER ­pd $DOMAIN\_PASSWORD ­sn $Service ­mo stop*  
 To get status of an Informatica service use the below command:  
  
 $INFA\_HOME/server/bin/infacmd.sh getServiceStatus ­dn $DOMAIN\_NAME ­un $DOMAIN\_USER ­pd $DOMAIN\_PASSWORD ­sn $Service  
  
  
To enable an Informatica service use the below command:   
    
$INFA\_HOME/server/bin/infacmd.sh enableService ­dn $DOMAIN\_NAME ­un $DOMAIN\_USER ­pd $DOMAIN\_PASSWORD sn $Service  
  
  
To recycle Informatica service using Infacmd (command line) then you have disable and enable a service using the commands above.

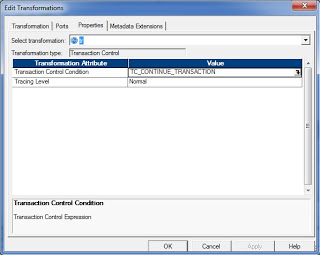
# Transaction Control Transformation in Informatica

Transaction Control is an active and connected transformation. The transaction control transformation is used to control the commit and rollback of transactions. You can define a transaction based on varying number of input rows. As an example, you can define a transaction on a group rows in the employees data using the department Id as a key.  
  
In the informatica power center, you can define the transaction at the following levels:

* **Mapping level**: Use the transaction control transformation to define the transactions.
* **Session level**: You can specify the "Commit Type" option in the session properties tab. The different options of "Commit Type" are Target, Source and User Defined. If you have used the transaction control transformation in the mapping, then the "Commit Type" will always be "User Defined"

When you run a session, the integration service evaluates the expression for each row in the transaction control transformation. When it evaluates the expression as commit, then it commits all the rows in the transaction to the target(s). When the integration service evaluates the expression as rollback, then it rolls back all the rows in the transaction from the target(s).  
  
When you have flat file as the target, then the integration service creates an output file for each time it commits the transaction. You can dynamically name the target flat files. Look at the example for creating flat files dynamically - Dynamic flat file creation.  
  
**Creating Transaction Control Transformation**  
  
Follow the below steps to create transaction control transformation:

* Go to the mapping designer, click on transformation in the toolbar, Create.
* Select the transaction control transformation, enter the name and click on Create and then Done.
* You can drag the ports in to the transaction control transformation or you can create the ports manually in the ports tab.
* Go to the properties tab. Enter the transaction control expression in the Transaction Control Condition.

[](http://1.bp.blogspot.com/-I5NNjMcL2fE/T5kXAD4vv0I/AAAAAAAAAW4/mpqpdyVUjLg/s1600/transaction_control_transofrmation.jpg)

**Configuring Transaction Control Transformation**  
  
You can configure the following components in the transaction control transformation:

* **Transformation Tab**: You can rename the transformation and add a description.
* **Ports Tab**: You can create input/output ports
* **Properties Tab**: You can define the transaction control expression and tracing level.
* **Metadata Extensions Tab**: You can add metadata information.

**Transaction Control Expression**  
  
You can enter the transaction control expression in the Transaction Control Condition option in the properties tab. The transaction control expression uses the IIF function to test each row against the condition. Use the following syntax for the expression

Syntax:

IIF (condition, value1, value2)

Example:

IIF(dept\_id=10, TC\_COMMIT\_BEFORE,TC\_ROLLBACK\_BEFORE)

Use the following built-in variables in the expression editor of the transaction control transformation:

* **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 transformation evaluates to a value other than the commit, rollback or continue, then the integration service fails the session.  
  
**Transaction Control Transformation in Mapping**  
  
Transaction control transformation defines or redefines the transaction boundaries in a mapping. It creates a new transaction boundary or drops any incoming transaction boundary coming from upstream active source or transaction control transformation.   
  
Transaction control transformation can be effective or ineffective for the downstream transformations and targets in the mapping. The transaction control transformation can become ineffective for downstream transformations or targets if you have used transformation that drops the incoming transaction boundaries after it. The following transformations drop the transaction boundaries.

* Aggregator transformation with Transformation scope as "All Input".
* Joiner transformation with Transformation scope as "All Input".
* Rank transformation with Transformation scope as "All Input".
* Sorter transformation with Transformation scope as "All Input".
* Custom transformation with Transformation scope as "All Input".
* Custom transformation configured to generate transactions
* Transaction Control transformation
* A multiple input group transformation, such as a Custom transformation, connected to multiple upstream transaction control points.

**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.

# [**INFORMATICA - Target Update Override : Updating Target Table without any Primary Keys defined**](http://ramakantshankar.blogspot.com/2014/03/informatica-target-update-override.html)

One can perform updates in the target in mapping using 2 approaches:

1. Using a UPDATE STRATEGY transformation
2. Configuring Session and defining properties to "Treat all rows as UPDATE"

But both these approach works only when there is a Primary Key defined in the target table. The session will fail when there will not be any key defined in the target.

Alternatively, if you want to perform TARGET updates,  with putting condition on COLUMNs not included as PRIMARY KEY in target table, it will not be feasible with out changing the Keys on target table. i.e. If PRIMARY KEY is defined on COLUMN\_A in TABLE\_A and you want to Update TABLE\_A with conditions on or including COLUMN\_B

In such cases the UPDATE\_OVERRIDE property of TARGET Object can be useful.

* ***How to access UPDATE\_OVERRIDE property?***

1. Go to Mapping
2. Double Clink on the concerned TARGET or edit the Target
3. Click on Properties Tab
4. The second Transformation Attribute is the property we are looking for

* ***Syntax for UPDATE\_OVERIDE SQL:***

*UDATE      <TARGET TABLE>  
SET            <COLUMN NAME to be updated> = :TU.<TARGET COLUMN PORT NAME (As in Designer)>*

*, [Other columns need to be updated]  
WHERE      <COLUMN NAME to be treated as KEY > = :TU.<corresponding TARGET COLUMN PORT NAME (As in Designer)>  
AND          [other conditions]*

* ***Example:***

*UPDATE    EMPL\_POST\_HIST*

*SET            POST = :TU.POST*

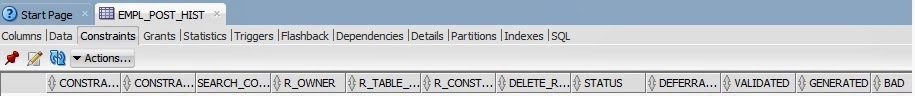
*, UPDATE\_DATE = :TU.UPDATE\_DATE*

*WHERE EMPL = :TU.EMPL*

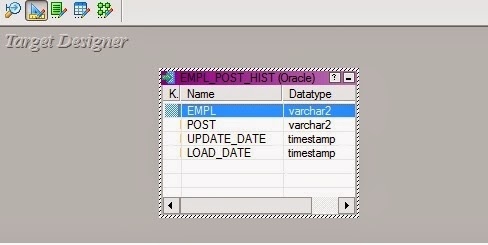
**Following is an example showing Problem statement and step by step Instructions, of two approaches, of suggested solution.**

**Problem Scenario:**

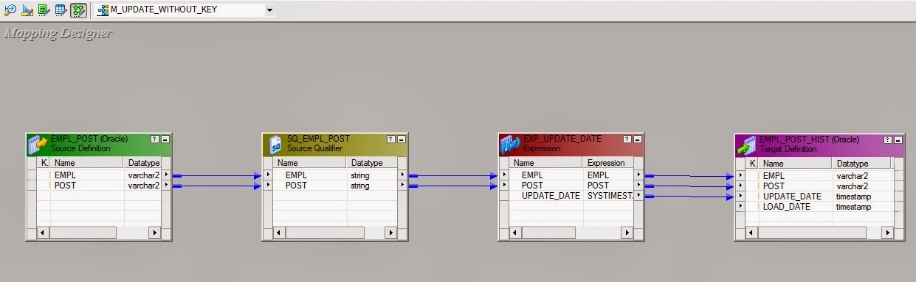
1. Target table has not defined Primary key

[](http://3.bp.blogspot.com/-CxRPYaZhLkU/UxWNOPFJByI/AAAAAAAAMLQ/-VFeA3OC9q0/s1600/TargetTableNoPrimary_key.jpg)

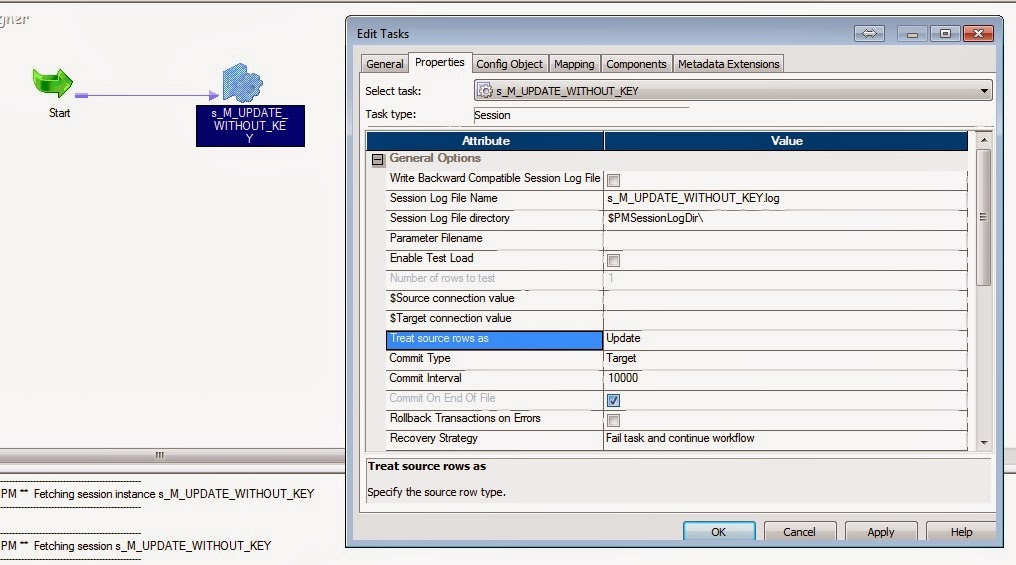
1. Informatica target object has no Keys - Since the Database table has not Keys the Imported Object in Informatica will not have any Key in the Target definition

[](http://1.bp.blogspot.com/-kASJ024nyMo/UxWNNNgIV3I/AAAAAAAAMLE/Zwsc2xi6nLM/s1600/INFA_Target.jpg)

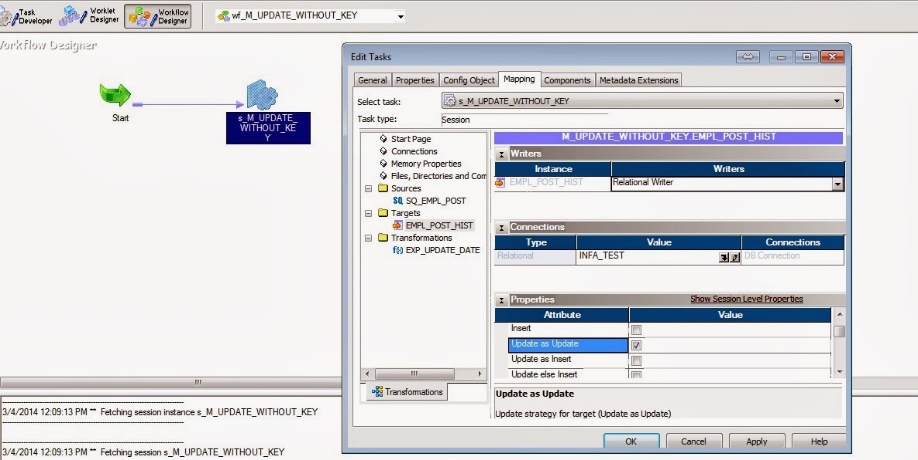
1. Original Mapping:  The mapping created using this very Target definition

[](http://4.bp.blogspot.com/-WHnkLetpRkU/UxWNSzNlMEI/AAAAAAAAMMA/ZDbLWpW9TvE/s1600/mapping.jpg)

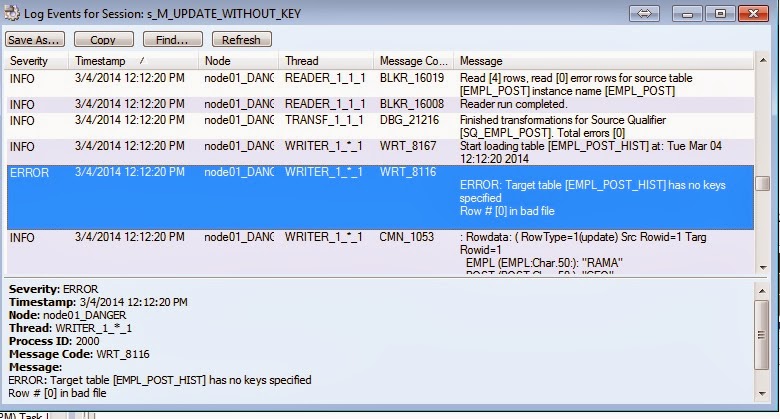
1. Defining update logic: configuring session for UPDATE Logic
2. Configure session for Treat All Source Row as "UPDATE"

[](http://2.bp.blogspot.com/-UtvZUFoux6k/UxWNTcebgxI/AAAAAAAAMMM/FeTwmer59iI/s1600/session_config1.jpg)

1. Configure in the session target property to UPDATE as UPDATE - optional

[](http://1.bp.blogspot.com/-lTNnG3Bv2t0/UxWNWHvPwII/AAAAAAAAMMU/knNlL2BrQaI/s1600/session_config2.jpg)

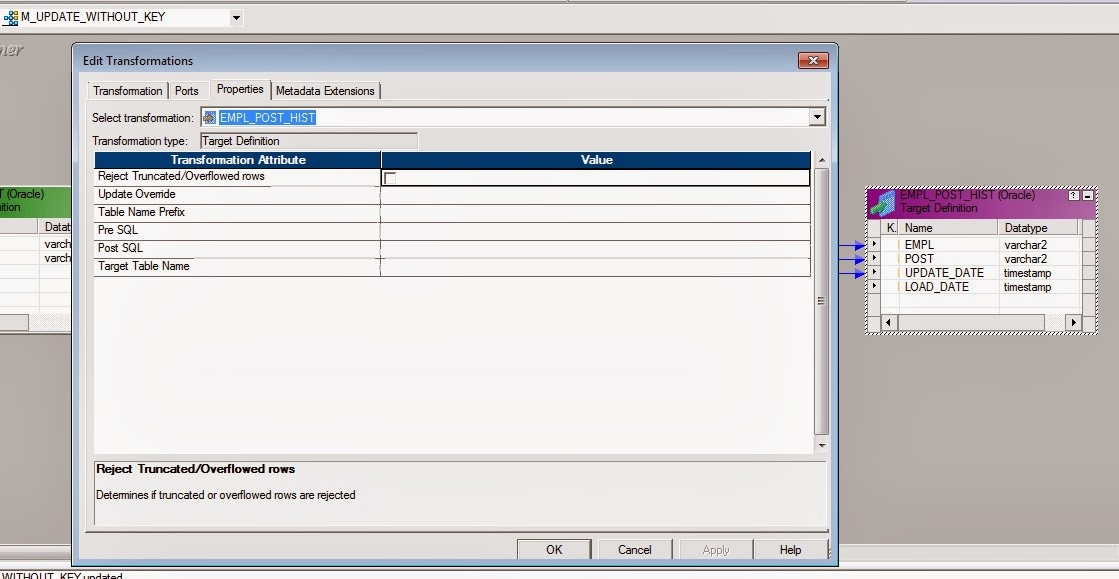
1. The error you will get while execution - the session will fail

[](http://4.bp.blogspot.com/-sfAOeQypXEk/UxWNWyxQ0xI/AAAAAAAAMMc/Ks0lrsryDkc/s1600/session_error_log.jpg)

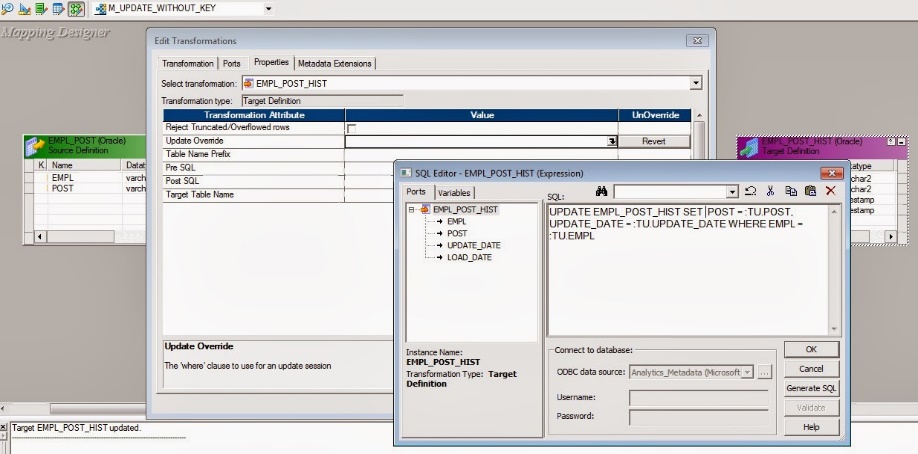
**Solution:**

**Approach 1: Using Update in the Target**

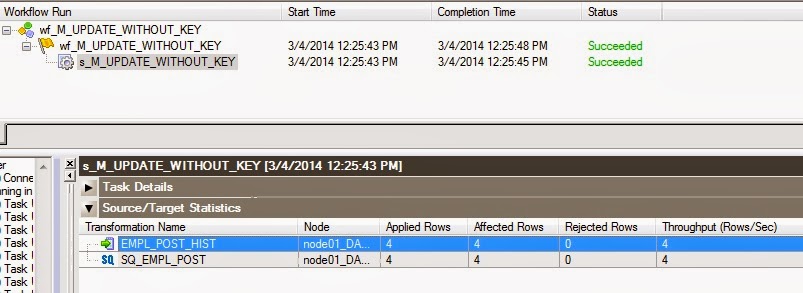
1. Go to properties tab of Target in Mapping:

[](http://1.bp.blogspot.com/-srmwggwf2F8/UxWNQFHeNAI/AAAAAAAAMLk/WhqF4sHRcsw/s1600/Target_property.jpg)

1. Edit the "Update Override" attribute - generate SQL and add where Clause with the column name on which you want to update:

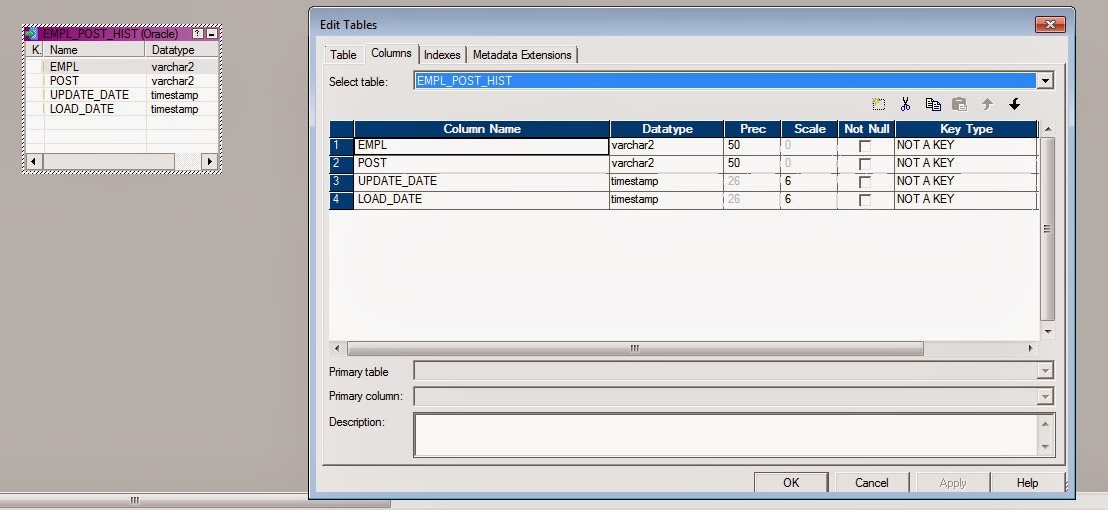
[](http://3.bp.blogspot.com/-7H_z0voxWaI/UxWNP-wxkNI/AAAAAAAAMLo/dANV819BNn0/s1600/Target_override.jpg)

1. Please note that you need to remove the very column, used in WHERE clause from the SET Clause list
2. Go to Workflow and refresh changes in the mapping and save them
3. Execute the Workflow
4. Now it has successfully updated the rows:

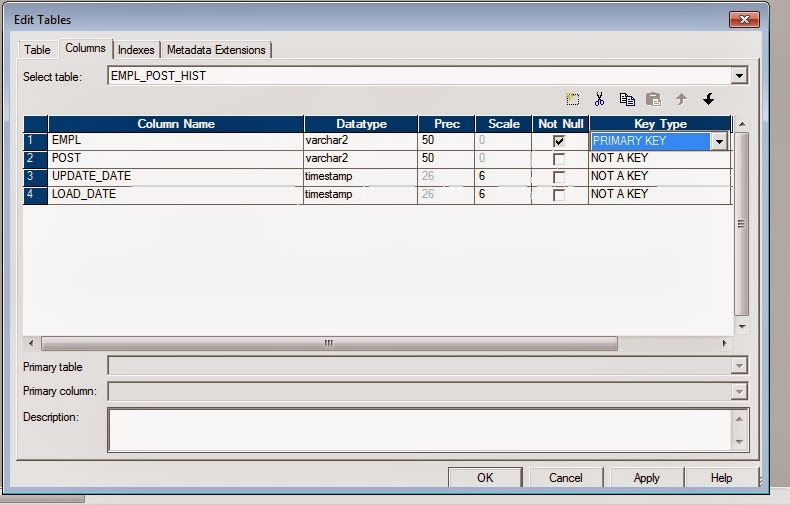
[](http://1.bp.blogspot.com/-dNQoAu8F6sQ/UxWNQbbKTOI/AAAAAAAAMLs/cQGO69j2S7c/s1600/Updated_rows1.jpg)

**Approach 2: Forcing Keys in the Target object definition in designer**

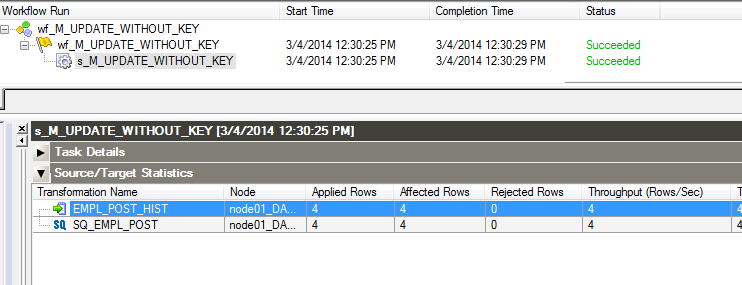
1. Edit the Target in the target designer

[](http://2.bp.blogspot.com/-MTp0RoRprhE/UxWNOuyVsfI/AAAAAAAAMLU/zHpxJ2aR8Os/s1600/Key1.jpg)

1. Define your Primary key column. Please note that in this case the Database need not have keys defined on the same table:

[](http://1.bp.blogspot.com/-a9fk4myqVUQ/UxWNS1VN54I/AAAAAAAAMME/33_QBe0CYzs/s1600/primary_key.jpg)

1. Save the changes
2. Execute the workflow
3. The workflow will update the records in the target table:

[](http://3.bp.blogspot.com/-P2FxdXjHOy4/UxWNQxKs9AI/AAAAAAAAML0/peUvLho_gTQ/s1600/Updated_rows2.jpg)

How to find files based on the file type?  
  
**a.** Finding socket files

find . -type s

**b.** Finding directories

find . -type d

**c.** Finding hidden directories

find . -type d -name ".\*"

**d.** Finding regular files

find . -type f

**e.** Finding hidden files

find . -type f -name ".\*"

**13.** How to find files based on the size?  
  
**a.** Finding files whose size is exactly 10M

find . -size 10M

**b.** Finding files larger than 10M size

find . -size +10M

**c.** Finding files smaller than 10M size

find . -size -10M

## **Unit Testing**

[AddThis Social Bookmark Button](http://www.addthis.com/bookmark.php)

Unit Testing  
In unit testing what we need  do is something like below

1. Validate source and target  
              -  Analyze & validate your transformation business rules.  
             -  We need review field by field from source to target and ensure that the required  
                  transformation logic is applied.  
               -   We generally check the source and target counts for each mapping.
2. Analyze the success and reject rows  
   -    In this stage we generally customized sql queries to check source and target.  
   -    Analyze the rejected rows and build the process to handle this rejection.
3. Calculate the load time  
   -    Run the session and view the statistics  
   -    We observe how much time is taken by reader and writer .  
   -    We should look at lesion log and workflow log to view the load statistics
4. Testing performance  
   -    Source performance  
   -    Target performance  
   -    Session performance  
   -    Network performance  
   -    Database performance  
   After unit testing we generally prepare one document  as described below
5. **UNIT TEST CASE FOR LOAN\_MASRER**

| **FUNCTIONALITY\_ID** | **FIELD\_NAME** | **DETAIL** | **VALUE PASSED** | **EXPECTED RESULT** | **ACTUAL RESULT** | **PASS/FAIL RESULT** | **REMARK** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| STG\_SCHM\_DTLS\_001 | LOAN  \_ID | \_TYPE\_ID SHOULD BE NOT NULL ,FIRST CHARACHER ALPHABET(INSCH) AND LAST 10 CHARACTER NUMERIC VALUES AND ALSO ITS LENGTH IS 16 | INSCH00000000002 | ACCEPT RECORD | RECORD ACCEPTED | PASS |  |
| STG\_SCHM\_DTLS\_002 | LOAN\_TYPE\_ID | REJECT WHEN , NOT NULL ,FIRST 5 CHARACHER NOT (INSCH) OR LAST 10 CHARACTER NON NUMERIC VALUES AND ALSO ITS LENGTH <>16 | INSCP001000000002 | REJECT RECORDRECORD REJECTED | PASS | RECORD INSERTED INTO REJECTED FILE WITH AN ERROR\_ID &ERROR\_DETAILS INTO ERROR\_TABLE |  |
| STG\_SCHM\_DTLS\_003 | LOAN\_COMPANY\_ID | LOAN\_COMPANY\_ID  MUST BE NOT NULL,FIRST 4 CHRACTER ALPHABET(INCO) AND LAST 11 CHRACTER NUMERIC VALUES AND ALSO LENGTH IS 15 | INCO00000000003 | ACCEPT RECORD | RECORD ACCEPTED | PASS |  |
| STG\_SCHM\_DTLS\_004 | LOAN\_COMPANY\_ID | REJECT WHEN , NOT NULL ,FIRST 4 CHARACHER NOT (INCO) OR LAST 11 CHARACTER NON NUMERIC VALUES AND ALSO ITS LENGTH <>15 | INSO00000060003 | REJECT RECORD | RECORD REJECTED | PASS | RECORD INSERTED INTO REJECTED FILE WITH AN ERROR\_ID &ERROR\_DETAILS INTO ERROR\_TABLE |
| STG\_SCHM\_DTLS\_005 | START\_DATE | START DATE SHOULD  BE  A VALID DATE | 12/9/1988 | ACCEPT RECORD | RECORD ACCEPTED | PASS |  |
| STG\_SCHM\_DTLS\_006 | START\_DATE | START DATE SHOULD NOT BE LOADED WHEN IT IS NOT A VALID DATE | 33FeB/88 | REJECT RECORD | RECORD REJECTED | PASS | RECORD INSERTED INTO REJECTED FILE WITH AN ERROR\_ID &ERROR\_DETAILS INTO ERROR\_TABLE |
| STG\_SCHM\_DTLS\_007 | SCHEME\_DESC | SCHEME-DESC SHOULD BE ALPHABETIC TYPE | AUTOMOBILE | ACCEPT RECORD | RECORD ACCEPTED | PASS |  |
| |  | | --- | | STG\_SCHM\_DTLS\_008 | | SCHEME\_DESC | REJECT WHEN SCHEME DISCOUNT IS NOT ALPHABETIC TYPE | MOTO124 | REJECT RECORD | RECORD REJECTED | PASS | RECORD INSERTED INTO REJECTED FILE WITH AN ERROR\_ID &ERROR\_DETAILS INTO ERROR\_TABLE |
| STG\_SCHM\_DTLS\_009 | PREMIUM\_PER\_LACS | PREMIUM\_PER\_LACSSHOULD BE NUMERIC | 5000 | ACCEPT RECORD | RECORD ACCEPTED | PASS |  |

Source System :- Oracle - Cadence

SQL - Lending-Pro

Flatfile - Gallagher into Informent, received via Momentum processes and loaded via SQL\_loader

Target System: - Oracle CRDM datamart

Tools: Informatica 9.6.1, Oracle11g for Db structure and Stored procedures,Momentum for file transfer, HP QC11 fro tracking defects, Control-M for job scheduling.

Architecture: - three stage process

1. Incremental data is loaded into ISA.(Schema - DW\_STG)

2. Current data from Target is stored into PSA. Delta is calculated in PSA.(Schema - DW\_STG)

3. Delta is stored into CRDM.(Schema - DW)

Fact table - DW.F\_LOAN\_ACCT - 147 columns - main Fact table

DW.F\_HELOC\_MTH - 19 Columns - for reporting

DW.F\_COHORT\_MTH - 35 Columns - for reporting

contains information about loan transactions happened in a month. updated on 1st calender date of every month.

Main Dimention tables :-

DW.D\_DATE

DW.D\_PRODUCT

DW.D\_PURCHASE\_POOL

Staging Tables:

DW\_STG.ISA\_CR\_DRR

DW\_STG.ISA\_CR\_PRAIRIE\_ISLAND\_LOAN

DW\_STG.ISA\_CR\_FDR\_LIEN

DW\_STG.ISA\_CR\_LOAN\_ORGNL\_ACCT

STAGING TABLES - 42

Roles and responsibilities:

Understanding requirements. has discussions with client on the initial requirements put by them.

Documents prepared:

HLD:

LLD:

Field-to-Field mapping and logic.

Three-tier architecture.

CRDM Monthly job flow diagram in MS Visio.

Requirement-Traceability Matrix Via HP QC11.

69 mappings and 31 workflows

FICO\_Score – Credit Score from Cadence and Gallagher

**Dynamic Partitioning:**

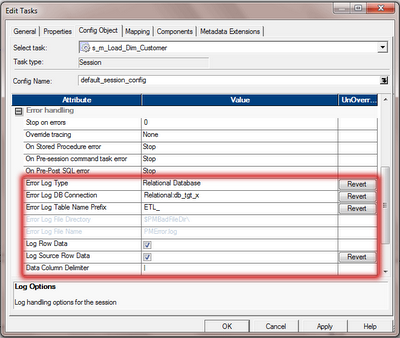
If the volume of data grows or you add more CPUs, you might need to adjust partitioning so that the session run time does not increase. You can use dynamic partitioning, so that the Integration Service determines the number of partitions to create at run time based on factors such as source database partitions or the number of nodes in a grid.

### Error handling made easy using Informatica Row Error Logging

Every time we start with a Data Warehouse or Data Integration project we spent lot of time defining our error handling approach. Still there are good chances that we might miss some scenarios because of unexpected data issues. Here in this article, lets us discuss an approach to handle the unexpected error.

## ERROR HANDLING CONFIGURATION.

We do not have any error configuration required during the mapping development. So after the mapping is created, during the session configuration set the session properties as shown in below image.

[](http://3.bp.blogspot.com/-EBtdaXQ-iNs/UANftysNu5I/AAAAAAAAEd4/bk_M9YC_940/s1600/error-handling-config.png)

Here is the details on what each property means.

* **Error Log Type** :-  Specifies the type of error log to create. It can be relational database or flat file.
* **Error Log DB Connection** :- Database connection for a relational log.
* **Error Log Table Name Prefix** :- Specifies the table name prefix for relational logs.
* **Log Row Data** :- Specifies whether or not to log transformation row data.
* **Log Source Row Data** :- Specifies whether or not to log source row data.
* **Data Column Delimiter** :- Data will be delimited by the specified character in DB column.

With this configuration we specified, Informatica PowerCenter will create four different tables for error logging and the table details as below.

* **ETL\_PMERR\_DATA** :- Stores data about a transformation row error and its corresponding source row.
* **ETL\_PMERR\_MSG**:- Stores metadata about an error and the error message.
* **ETL\_PMERR\_SESS**:- Stores metadata about the session.
* **ETL\_PMERR\_TRANS** :- Stores metadata about the source and transformation ports, when error occurs.

With this configuration, we are done with the setting required to capture any error during the session execution. Now lets see how do we retrieve the data from the error log tables and report it to Business Users or IT Department.

## **REPORT THE ERROR DATA.**

Now we have the error data stored in the error table, it is important to share the error data to the Business Users or to IT Department. Lets see how we can pull data from these tables.  
  
We can pull the basic error report using the SQL. We can be more fancy with the SQL and get more information from the error tables.

select

       sess.FOLDER\_NAME as 'Folder Name',

       sess.WORKFLOW\_NAME as 'WorkFlow Name',

       sess.TASK\_INST\_PATH as 'Session Name',

       data.SOURCE\_ROW\_DATA as 'Source Data',

       msg.ERROR\_MSG as 'Error MSG'  
from

       ETL\_PMERR\_SESS sess  
left outer join ETL\_PMERR\_DATA data  
      on data.WORKFLOW\_RUN\_ID = sess.WORKFLOW\_RUN\_ID and

      data.SESS\_INST\_ID = sess.SESS\_INST\_ID  
left outer join ETL\_PMERR\_MSG msg  
      on msg.WORKFLOW\_RUN\_ID = sess.WORKFLOW\_RUN\_ID and

      msg.SESS\_INST\_ID = sess.SESS\_INST\_ID  
where

      sess.FOLDER\_NAME = <Project Folder Name> and  
      sess.WORKFLOW\_NAME = <Workflow Name> and  
      sess.TASK\_INST\_PATH = <Session Name> and

      sess.SESS\_START\_TIME = <Session Run Time>

The above SQL can be converted to an Informatica mapping and can be scheduled to run after all your ETL jobs are completed to get a daily error report and the same can be scheduled for an email delivery to the data owners to take relevant action on the errors.

## **PROS AND CONS OF THIS APPROACH.**

We should know the Pros and Cons of this approach  before applying this to your project.

### Pros.

1. Out of the box Solution Provided by Informatica.
2. Captures all the Error, Including the unthought error.
3. Less Coding and Testing efforts required by the development team.

### Cons.

1. If an error occurs after an active transformation, Integration service will not be able to capture source data into the erro table.
2. In sessions with multiple non-pass through partitions, Integration service can not capture source data into error table.
3. Added overhead to the Session performance (Which is expected and acceptable).

pmrep ObjectExport -n objName -o objType -f FolderName -m -s -b -r -u objName.xml

ObjectExport exports objects by name, to export multiple objects from a folder you can either supply names one at a time, re-executing the command for each name or you can supply an input file containing a list of object names. You also must supply the object name's folder (-f). The last parameter is the output file name.

ObjectExport creates XML that can be imported back into PowerCenter. The switches -m -s -b –r are required to get dependent objects (tables, shortcuts, non-reusable objects) for the object you are exporting. For some objects, if you do not export its dependent objects, it will not re-import into the repository correctly.

|  |
| --- |
| Oracle SQL Hints Tuning  *Oracle Database Tips by Burleson Consulting* |

## Using Hints to tune SQL **Mike Ault**

There are many Oracle hints available to the developer for use in tuning SQL statements that are embedded in PL/SQL.

You should first get the explain plan of your SQL and determine what changes can be done to make the code operate without using hints if possible. However, Oracle hints such as ORDERED, LEADING, INDEX, FULL, and the various AJ and SJ Oracle hints can tame a wild optimizer and give you optimal performance.

Oracle hints are enclosed within comments to the SQL commands DELETE, SELECT or UPDATE or are designated by two dashes and a plus sign. To show the format the SELECT statement only will be used, but the format is identical for all three commands.

|  |
| --- |
| SELECT   /\*+ hint --or-- text \*/  statement body  -- or --  SELECT   --+ hint --or-- text  statement body |

Where:

* + - /\*, \*/  -  These are the comment delimiters for multi-line comments
    - --  -  This is the comment delimiter for a single line comment (not usually used for hints)
    - +  -  This tells Oracle a hint follows, it must come immediately after the /\*
    - **hint**  -  This is one of the allowed hints
    - **text**  -  This is the comment text

|  |  |
| --- | --- |
| **Oracle Hint** | **Meaning** |
| + | Must be immediately after comment indicator, tells Oracle this is a list of hints. |
| ALL\_ROWS | Use the cost based approach for best throughput. |
| CHOOSE | Default, if statistics are available will use cost, if not, rule. |
| FIRST\_ROWS | Use the cost based approach for best response time. |
| RULE | Use rules based approach; this cancels any other hints specified for this statement. |
| Access Method Oracle Hints: |  |
| CLUSTER(table) | This tells Oracle to do a cluster scan to access the table. |
| FULL(table) | This tells the optimizer to do a full scan of the specified table. |
| HASH(table) | Tells Oracle to explicitly choose the hash access method for the table. |
| HASH\_AJ(table) | Transforms a NOT IN subquery to a hash anti-join. |
| ROWID(table) | Forces a rowid scan of the specified table. |
| INDEX(table [index]) | Forces an index scan of the specified table using the specified index(s). If a list of indexes is specified, the optimizer chooses the one with the lowest cost. If no index is specified then the optimizer chooses the available index for the table with the lowest cost. |
| INDEX\_ASC (table [index]) | Same as INDEX only performs an ascending search of the index chosen, this is functionally identical to the INDEX statement. |
| INDEX\_DESC(table [index]) | Same as INDEX except performs a descending search. If more than one table is accessed, this is ignored. |
| INDEX\_COMBINE(table index) | Combines the bitmapped indexes on the table if the cost shows that to do so would give better performance. |
| INDEX\_FFS(table index) | Perform a fast full index scan rather than a table scan. |
| MERGE\_AJ (table) | Transforms a NOT IN subquery into a merge anti-join. |
| AND\_EQUAL(table index index [index index index]) | This hint causes a merge on several single column indexes. Two must be specified, five can be. |
| NL\_AJ | Transforms a NOT IN subquery into a NL anti-join (nested loop). |
| HASH\_SJ(t1, t2) | Inserted into the EXISTS subquery; This converts the subquery into a special type of hash join between t1 and t2 that preserves the semantics of the subquery. That is, even if there is more than one matching row in t2 for a row in t1, the row in t1 is returned only once. |
| MERGE\_SJ (t1, t2) | Inserted into the EXISTS subquery; This converts the subquery into a special type of merge join between t1 and t2 that preserves the semantics of the subquery. That is, even if there is more than one matching row in t2 for a row in t1, the row in t1 is returned only once. |
| NL\_SJ | Inserted into the EXISTS subquery; This converts the subquery into a special type of nested loop join between t1 and t2 that preserves the semantics of the subquery. That is, even if there is more than one matching row in t2 for a row in t1, the row in t1 is returned only once. |
| Oracle Hints for join orders and transformations: |  |
| ORDERED | This hint forces tables to be joined in the order specified. If you know table X has fewer rows, then ordering it first may speed execution in a join. |
| STAR | Forces the largest table to be joined last using a nested loops join on the index. |
| STAR\_TRANSFORMATION | Makes the optimizer use the best plan in which a start transformation is used. |
| FACT(table) | When performing a star transformation use the specified table as a fact table. |
| NO\_FACT(table) | When performing a star transformation do not use the specified table as a fact table. |
| PUSH\_SUBQ | This causes nonmerged subqueries to be evaluated at the earliest possible point in the execution plan. |
| REWRITE(mview) | If possible forces the query to use the specified materialized view, if no materialized view is specified, the system chooses what it calculates is the appropriate view. |
| NOREWRITE | Turns off query rewrite for the statement, use it for when data returned must be concurrent and can't come from a materialized view. |
| USE\_CONCAT | Forces combined OR conditions and IN processing in the WHERE clause to be transformed into a compound query using the UNION ALL set operator. |
| NO\_MERGE (table) | This causes Oracle to join each specified table with another row source without a sort-merge join. |
| NO\_EXPAND | Prevents OR and IN processing expansion. |
| Oracle Hints for Join Operations: |  |
| USE\_HASH (table) | This causes Oracle to join each specified table with another row source with a hash join. |
| USE\_NL(table) | This operation forces a nested loop using the specified table as the controlling table. |
| USE\_MERGE(table,[table, -]) | This operation forces a sort-merge-join operation of the specified tables. |
| DRIVING\_SITE | The hint forces query execution to be done at a different site than that selected by Oracle. This hint can be used with either rule-based or cost-based optimization. |
| LEADING(table) | The hint causes Oracle to use the specified table as the first table in the join order. |
| Oracle Hints for Parallel Operations: |  |
| [NO]APPEND | This specifies that data is to be or not to be appended to the end of a file rather than into existing free space. Use only with INSERT commands. |
| NOPARALLEL (table | This specifies the operation is not to be done in parallel. |
| PARALLEL(table, instances) | This specifies the operation is to be done in parallel. |
| PARALLEL\_INDEX | Allows parallelization of a fast full index scan on any index. |
| Other Oracle Hints: |  |
| CACHE | Specifies that the blocks retrieved for the table in the hint are placed at the most recently used end of the LRU list when the table is full table scanned. |
| NOCACHE | Specifies that the blocks retrieved for the table in the hint are placed at the least recently used end of the LRU list when the table is full table scanned. |
| [NO]APPEND | For insert operations will append (or not append) data at the HWM of table. |
| UNNEST | Turns on the UNNEST\_SUBQUERY option for statement if UNNEST\_SUBQUERY parameter is set to FALSE. |
| NO\_UNNEST | Turns off the UNNEST\_SUBQUERY option for statement if UNNEST\_SUBQUERY parameter is set to TRUE. |
| PUSH\_PRED | Pushes the join predicate into the view. |

As you can see, a dilemma with a stubborn index can be easily solved using FULL or NO\_INDEX Oracle hints. You must know the application to be tuned. The DBA can provide guidance to developers but in all but the smallest development projects, it will be nearly impossible for a DBA to know everything about each application. It is clear that responsibility for application tuning rests solely on the developer's shoulders with help and guidance from the DBA.

## Using Global Hints

While Oracle hints normally refer to table in the query it is possible to specify a hint for a table within a view through the use of what are known as Oracle GLOBAL HINTS. This is done using the Oracle global hint syntax. Any table hint can be transformed into an Oracle global hint.

The syntax is:

|  |
| --- |
| /\*+ hint(view\_name.table\_in\_view) \*/ |

For example:

|  |
| --- |
| /\*+ full(sales\_totals\_vw.s\_customer)\*/ |

If the view is an inline view, place an alias on it and then use the alias to reference the inline view in the Oracle global hint.

# Oracle's DBMS\_PROFILER: PL/SQL Performance Tuning

**By**[**Amar Kumar Padhi**](http://www.databasejournal.com/feedback.php/http:/www.databasejournal.com/features/oracle/article.php/2197231/Oracles-DBMSPROFILER-PLSQL-Performance-Tuning.htm)

An application can always be fine-tuned for better performance with the use of better alternatives or with the new features introduced with every release of Oracle.

Simply inspecting the code can bring out the bottlenecks eating up your processing time. Usingexplain plan to fine tune the SQL statements resolves issues most of the time. However, sometimes it may not be that simple. It is baffling when all the SQL statements are well tuned but the routine still takes noticeable time to execute.

**DBMS\_PROFILER Package**   
Oracle 8i provides a new tool called PL/SQL Profiler. This is a powerful tool to analyze a Program unit execution and determine the runtime behavior. The results generated can then be evaluated to find out the hot areas in the code. This tool helps us identify performance bottlenecks, as well as where excess execution time is being spent in the code. The time spent in executing an SQL statement is also generated. This process is implemented with DBMS\_PROFILER package.

The possible profiler statistics that are generated:   
   1. Total number of times each line was executed.   
   2. Time spent executing each line. This includes SQL statements.  
   3. Minimum and maximum duration spent on a specific line of code.   
   4. Code that is actually being executed for a given scenario.

**DBMS\_PROFILER.START\_PROFILER**   
The DBMS\_PROFILER.START\_PROFILER tells Oracle to start the monitoring process. An identifier needs to be provided with each run that is used later to retrieve the statistics.

e.g.:

l\_runstatus := dbms\_profiler.start\_profiler('am' ||

to\_char(sysdate));

**DBMS\_PROFILER.STOP\_PROFILER**   
The DBMS\_PROFILER.STOP\_PROFILER tells Oracle to stop the monitoring.

e.g.:

l\_runstatus := dbms\_profiler.stop\_profiler;

**DBMS\_PROFILER.FLUSH\_DATA**   
The data collected for an execution is held in the memory. Calling the DBMS\_PROFILER.FLUSH\_DATA routine tells Oracle to save this data in profiler tables and clear the memory.

e.g.:

l\_runstatus := dbms\_profiler.flush\_data;

The above functions return the following status'.   
   0 : Successful completion   
   1 : Incorrect parameters passed (error\_parm).   
   2 : data flush operation failed (error\_io).   
   -1 : mismatch between package and database implementation (error\_version).

|  |
| --- |
| DBMS\_PROFILER |
|  |

Before DBMS\_PROFILER was introduced tuning long PL/SQL programs was a great pain. Tuning PL/SQL was done by SQL TRACE and TKPROF. Code of thousands of lines was monitored via DBMS\_UTILITY.GET\_TIME to measure time consumed by individual statements.

With the presence of DBMS\_PROFILER this has become quite easy to monitor the performance of the PL/SQL programs and identifying the culprit statements consuming too much time while the run of the program.

In this article we have two procedures. One of them uses literals in the queries and the other uses bind variables. We noticed that the program using literals taking too much time. We will then use DBMS\_PROFILER to pin point the statement consuming most of the time.

## **Understanding EXPLAIN PLAN**

The EXPLAIN PLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETEstatements. A statement's execution plan is the sequence of operations Oracle performs to run the statement.

The row source tree is the core of the execution plan. It shows the following information:

* An ordering of the tables referenced by the statement
* An access method for each table mentioned in the statement
* A join method for tables affected by join operations in the statement
* Data operations like filter, sort, or aggregation

In addition to the row source tree, the plan table contains information about the following:

* Optimization, such as the cost and cardinality of each operation
* Partitioning, such as the set of accessed partitions
* Parallel execution, such as the distribution method of join inputs

The EXPLAIN PLAN results let you determine whether the optimizer selects a particular execution plan, such as, nested loops join. It also helps you to understand the optimizer decisions, such as why the optimizer chose a nested loops join instead of a hash join, and lets you understand the performance of a query.

### How Execution Plans Can Change

With the query optimizer, execution plans can and do change as the underlying optimizer inputs change. EXPLAIN PLAN output shows how Oracle runs the SQL statement when the statement was explained. This can differ from the plan during actual execution for a SQL statement, because of differences in the execution environment and explain plan environment.

Execution plans can differ due to the following:

* [Different Schemas](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i18891)
* [Different Costs](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i16344)

#### **19.1.1.1 Different Schemas**

* The execution and explain plan happen on different databases.
* The user explaining the statement is different from the user running the statement. Two users might be pointing to different objects in the same database, resulting in different execution plans.
* Schema changes (usually changes in indexes) between the two operations.

#### **19.1.1.2 Different Costs**

Even if the schemas are the same, the optimizer can choose different execution plans if the costs are different. Some factors that affect the costs include the following:

* Data volume and statistics
* Bind variable types and values
* Initialization parameters - set globally or at session level

### 19.1.2 Minimizing Throw-Away

Examining an explain plan lets you look for throw-away in cases such as the following:

* Full scans
* Unselective range scans
* Late predicate filters
* Wrong join order
* Late filter operations

For example, in the following explain plan, the last step is a very unselective range scan that is executed 76563 times, accesses 11432983 rows, throws away 99% of them, and retains 76563 rows. Why access 11432983 rows to realize that only 76563 rows are needed?

***Example 19-1 Looking for Throw-Away in an Explain Plan***

Rows Execution Plan

-------- ----------------------------------------------------

12 SORT AGGREGATE

2 SORT GROUP BY

76563 NESTED LOOPS

76575 NESTED LOOPS

19 TABLE ACCESS FULL CN\_PAYRUNS\_ALL

76570 TABLE ACCESS BY INDEX ROWID CN\_POSTING\_DETAILS\_ALL

76570 INDEX RANGE SCAN (object id 178321)

76563 TABLE ACCESS BY INDEX ROWID CN\_PAYMENT\_WORKSHEETS\_ALL

11432983 INDEX RANGE SCAN (object id 186024)

### 19.1.3 Looking Beyond Execution Plans

The execution plan operation alone cannot differentiate between well-tuned statements and those that perform poorly. For example, an EXPLAIN PLAN output that shows that a statement uses an index does not necessarily mean that the statement runs efficiently. Sometimes indexes can be extremely inefficient. In this case, you should examine the following:

* The columns of the index being used
* Their selectivity (fraction of table being accessed)

It is best to use EXPLAIN PLAN to determine an access plan, and then later prove that it is the optimal plan through testing. When evaluating a plan, examine the statement's actual resource consumption.

#### **19.1.3.1 Using V$SQL\_PLAN Views**

In addition to running the EXPLAIN PLAN command and displaying the plan, you can use the V$SQL\_PLAN views to display the execution plan of a SQL statement:

After the statement has executed, you can display the plan by querying the V$SQL\_PLAN view. V$SQL\_PLAN contains the execution plan for every statement stored in the cursor cache. Its definition is similar to the PLAN\_TABLE. See ["PLAN\_TABLE Columns"](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i18300).

The advantage of V$SQL\_PLAN over EXPLAIN PLAN is that you do not need to know the compilation environment that was used to execute a particular statement. For EXPLAIN PLAN, you would need to set up an identical environment to get the same plan when executing the statement.

The V$SQL\_PLAN\_STATISTICS view provides the actual execution statistics for every operation in the plan, such as the number of output rows and elapsed time. All statistics, except the number of output rows, are cumulative. For example, the statistics for a join operation also includes the statistics for its two inputs. The statistics in V$SQL\_PLAN\_STATISTICS are available for cursors that have been compiled with the STATISTICS\_LEVEL initialization parameter set to ALL.

The V$SQL\_PLAN\_STATISTICS\_ALL view enables side by side comparisons of the estimates that the optimizer provides for the number of rows and elapsed time. This view combines both V$SQL\_PLAN and V$SQL\_PLAN\_STATISTICS information for every cursor.

### EXPLAIN PLAN Restrictions

Oracle does not support EXPLAIN PLAN for statements performing implicit type conversion of date bind variables. With bind variables in general, the EXPLAIN PLAN output might not represent the real execution plan.

From the text of a SQL statement, TKPROF cannot determine the types of the bind variables. It assumes that the type is CHARACTER, and gives an error message if this is not the case. You can avoid this limitation by putting appropriate type conversions in the SQL statement.

## **The PLAN\_TABLE Output Table**

The PLAN\_TABLE is automatically created as a global temporary table to hold the output of an EXPLAIN PLAN statement for all users. PLAN\_TABLE is the default sample output table into which the EXPLAIN PLAN statement inserts rows describing execution plans. See ["PLAN\_TABLE Columns"](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i18300) for a description of the columns in the table.

While a PLAN\_TABLE table is automatically set up for each user, you can use the SQL script utlxplan.sql to manually create a local PLAN\_TABLE in your schema. The exact name and location of this script depends on your operating system. On Unix, it is located in the $ORACLE\_HOME/rdbms/admin directory.

For example, run the commands in [Example 19-2](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i24492) from a SQL\*Plus session to create the PLAN\_TABLE in the HR schema.

***Example 19-2 Creating a PLAN\_TABLE***

CONNECT HR/*your\_password*

@$ORACLE\_HOME/rdbms/admin/utlxplan.sql

Table created.

Oracle Corporation recommends that you drop and rebuild your local PLAN\_TABLE table after upgrading the version of the database because the columns might change. This can cause scripts to fail or cause TKPROF to fail, if you are specifying the table.

If you want an output table with a different name, first create PLAN\_TABLE manually with the utlxplan.sql script and then rename the table with the RENAME SQL statement. For example:

RENAME PLAN\_TABLE TO my\_plan\_table;

## **19.3 Running EXPLAIN PLAN**

To explain a SQL statement, use the EXPLAIN PLAN FOR clause immediately before the statement. For example:

EXPLAIN PLAN FOR

SELECT last\_name FROM employees;

This explains the plan into the PLAN\_TABLE table. You can then select the execution plan from PLAN\_TABLE. See ["Displaying PLAN\_TABLE Output"](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i16938).

### 19.3.1 Identifying Statements for EXPLAIN PLAN

With multiple statements, you can specify a statement identifier and use that to identify your specific execution plan. Before using SETSTATEMENT ID, remove any existing rows for that statement ID.

In [Example 19-3](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i24641), st1 is specified as the statement identifier:

***Example 19-3 Using EXPLAIN PLAN with the STATEMENT ID Clause***

EXPLAIN PLAN

SET STATEMENT\_ID = 'st1' FOR

SELECT last\_name FROM employees;

### 19.3.2 Specifying Different Tables for EXPLAIN PLAN

You can specify the INTO clause to specify a different table.

***Example 19-4 Using EXPLAIN PLAN with the INTO Clause***

EXPLAIN PLAN

INTO my\_plan\_table

FOR

SELECT last\_name FROM employees;

You can specify a statement Id when using the INTO clause.

EXPLAIN PLAN

SET STATEMENT\_ID = 'st1'

INTO my\_plan\_table

FOR

SELECT last\_name FROM employees;

**See Also:**

[*Oracle Database SQL Language Reference*](https://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm) for a complete description of EXPLAIN PLAN syntax.

## **19.4 Displaying PLAN\_TABLE Output**

After you have explained the plan, use the following SQL scripts or PL/SQL package provided by Oracle to display the most recent plan table output:

* UTLXPLS.SQL

This script displays the plan table output for serial processing. [Example 13-2, "EXPLAIN PLAN Output"](https://docs.oracle.com/cd/B19306_01/server.102/b14211/optimops.htm#i73898) is an example of the plan table output when using the UTLXPLS.SQL script.

* UTLXPLP.SQL

This script displays the plan table output including parallel execution columns.

* DBMS\_XPLAN.DISPLAY procedure

This procedure accepts options for displaying the plan table output. You can specify:

* + A plan table name if you are using a table different than PLAN\_TABLE
  + A statement Id if you have set a statement Id with the EXPLAIN PLAN
  + A format option that determines the level of detail: BASIC, SERIAL, and TYPICAL, ALL,

Some examples of the use of DBMS\_XPLAN to display PLAN\_TABLE output are:

SELECT PLAN\_TABLE\_OUTPUT FROM TABLE(DBMS\_XPLAN.DISPLAY());

SELECT PLAN\_TABLE\_OUTPUT

FROM TABLE(DBMS\_XPLAN.DISPLAY('MY\_PLAN\_TABLE', 'st1','TYPICAL'));

**See Also:**

[*Oracle Database PL/SQL Packages and Types Reference*](https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_xplan.htm#ARPLS378) for more information on the DBMS\_XPLAN package

### 19.4.1 Customizing PLAN\_TABLE Output

If you have specified a statement identifier, then you can write your own script to query the PLAN\_TABLE. For example:

* Start with ID = 0 and given STATEMENT\_ID.
* Use the CONNECT BY clause to walk the tree from parent to child, the join keys being STATEMENT\_ID = PRIOR STATEMENT\_ID and PARENT\_ID = PRIOR ID.
* Use the pseudo-column LEVEL (associated with CONNECT BY) to indent the children.
* SELECT cardinality "Rows",
* lpad(' ',level-1)||operation||' '||options||' '||object\_name "Plan"
* FROM PLAN\_TABLE
* CONNECT BY prior id = parent\_id
* AND prior statement\_id = statement\_id
* START WITH id = 0
* AND statement\_id = 'st1'
* ORDER BY id;
* Rows Plan
* ------- ----------------------------------------
* SELECT STATEMENT
* TABLE ACCESS FULL EMPLOYEES

The NULL in the Rows column indicates that the optimizer does not have any statistics on the table. Analyzing the table shows the following:

Rows Plan

------- ----------------------------------------

16957 SELECT STATEMENT

16957 TABLE ACCESS FULL EMPLOYEES

You can also select the COST. This is useful for comparing execution plans or for understanding why the optimizer chooses one execution plan over another.

**Note:**

These simplified examples are not valid for recursive SQL.

## **19.5 Reading EXPLAIN PLAN Output**

This section uses EXPLAIN PLAN examples to illustrate execution plans. The statement in [Example 19-5](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i28423) is used to display the execution plans.

***Example 19-5 Statement to display the EXPLAIN PLAN***

SELECT PLAN\_TABLE\_OUTPUT

FROM TABLE(DBMS\_XPLAN.DISPLAY(NULL, '*statement\_id*','BASIC'));

Examples of the output from this statement are shown in [Example 19-6](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i28490) and [Example 19-7](https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i28494).

***Example 19-6 EXPLAIN PLAN for Statement Id ex\_plan1***

EXPLAIN PLAN

SET statement\_id = 'ex\_plan1' FOR

SELECT phone\_number FROM employees

WHERE phone\_number LIKE '650%';

---------------------------------------

| Id | Operation | Name |

---------------------------------------

| 0 | SELECT STATEMENT | |

| 1 | TABLE ACCESS FULL| EMPLOYEES |

---------------------------------------

This plan shows execution of a SELECT statement. The table employees is accessed using a full table scan.

* Every row in the table employees is accessed, and the WHERE clause criteria is evaluated for every row.
* The SELECT statement returns the rows meeting the WHERE clause criteria.

***Example 19-7 EXPLAIN PLAN for Statement Id ex\_plan2***

EXPLAIN PLAN

SET statement\_id = 'ex\_plan2' FOR

SELECT last\_name FROM employees

WHERE last\_name LIKE 'Pe%';

SELECT PLAN\_TABLE\_OUTPUT

FROM TABLE(DBMS\_XPLAN.DISPLAY(NULL, 'ex\_plan2','BASIC'));

----------------------------------------

| Id | Operation | Name |

----------------------------------------

| 0 | SELECT STATEMENT | |

| 1 | INDEX RANGE SCAN| EMP\_NAME\_IX |

----------------------------------------

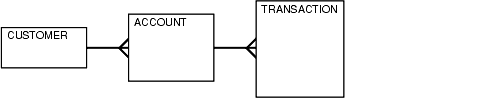
This plan shows execution of a SELECT statement.

* Index EMP\_NAME\_IX is used in a range scan operation to evaluate the WHERE clause criteria.
* The SELECT statement returns rows satisfying the WHERE clause conditions.

## **19.6 Viewing Parallel Execution with EXPLAIN PLAN**

Tuning a parallel query begins much like a non-parallel query tuning exercise by choosing the driving table. However, the rules governing the choice are different. In the non-parallel case, the best driving table is typically the one that produces fewest number of rows after limiting conditions are applied. The small number of rows are joined to larger tables using non-unique indexes. For example, consider a table hierarchy consisting of CUSTOMER, ACCOUNT, and TRANSACTION.

***Figure 19-1 A Table Hierarchy***

  
[Description of "Figure 19-1 A Table Hierarchy"](https://docs.oracle.com/cd/B19306_01/server.102/b14211/img_text/pfgrf211.htm)

**Question 1: What are the results returned from the T-SQL below?**

*CREATE TABLE a (id INT, second\_id INT);*  
*INSERT a (id, second\_id)*

*SELECT 1, 3*  
*UNION ALL*  
*SELECT 2, NULL*  
*UNION ALL*  
*SELECT 3, 3;*

*SELECT*  
*COUNT(\*) AS CountTotal*  
*, COUNT(second\_id) AS CountSecondId*  
*, COUNT(DISTINCT second\_id) AS CountDistinctSecondId*  
*FROM a;*

**Answer 1:**3,2,1  
(Because Count(Column\_Name) doesn’t consider NULL Value)

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2. Given the following tables:>**

SELECT \* FROM runners;  
+----+--------------+  
| id | name |  
+----+--------------+

| 1 | John Doe     |  
| 2 | Jane Doe     |  
| 3 | Alice Jones  |  
| 4 | Bobby Louis  |  
| 5 | Lisa Romero  |  
+----+--------------+  
  
 SELECT \* FROM races;  
+----+----------------+-----------+  
| id | event                 | winner\_id |  
+----+----------------+-----------+  
| 1 | 100 meter dash  | 2        |  
| 2 | 500 meter dash | 3        |  
| 3 | cross-country     |  2        |  
| 4 | triathalon            |  NULL     |

*select \* from runners where id not in (select winner\_id from races)*

 Explain your answer and also provide an alternative version of this query that will avoid the issue that it exposes.

**Answer 2:**

As races table contains the record with NULL value the Query will not yield the desired result

***Correct Query****: select \* from runners where id not in (select winner\_id from races where winner\_id is not null)*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 3:**

What will be the result of the query below? Explain your answer and provide a version that behaves correctly.

*select case when null = null then 'Yup' else 'Nope' end as Result;*

**Answer 3:**

Result: Nope

Because a NULL value cannot be compared with NULL, we can make use of ISNULL() function to replace the value with certain number say 1

**Correct Query:** *select case when Isnull(null,1) = Isnull(null,1) then 'Yup' else 'Nope' end as Result;*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 4:**

Given a table, such as the one below, that hasm = male and f = female  Swap all f and m values (i.e., change all f values to m and vice versa) with a single update query and no intermediate temp table.

Id    Name Sex    Salary

1     A          m      25002

2     B           f      15003

3      C         m      55004

4       D          f       500

**Answer 4:**

*Update table*  
*Set Sex=(Case Sex when ‘m’ then ‘f’ else ‘m’ end)*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Question 5:**

Assume a schema of

Emp ( Id, Name, DeptId )

, Dept ( Id, Name).

If there are 10 records in the Emp table and 5 records in the Dept table, how many rows will be displayed in the result of the following SQL query:

*Select \* From Emp, Dept*

**Answer 5:**

It’s 10\*5=50 rows

Explanation: So simply its multiplication of the 2 numbers,say if table A have 17 records and table B have 3 records then

No. of rows for the Query : *Select \* from TableA,TableB*

Would be 17\*3=51

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 6:**

What is wrong with this SQL query? Correct it so it executes properly.

*SELECT Id, YEAR(BillingDate) AS BillingYear FROM Invoices*  
*WHERE BillingYear >= 2010;*

**Answer 6:**

The where clause cannot read the name given to the column in the Select clause because the execution of the Query takes place in following order

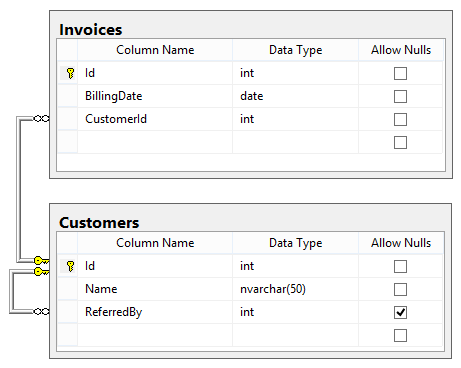
From  
Where  
Groub by  
Having  
Select  
Order by

**Correct Query:***SELECT Id, YEAR(BillingDate) AS BillingYear*  
*FROM Invoices*  
*WHERE YEAR(BillingDate) >= 2010;*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Question 7:**

Considering the database schema displayed in the SQLServer-style diagram below, write a SQL query to return a list of all the invoices. For each invoice, show the Invoice ID, the billing date, the customer’s name, and the name of the customer who referred that customer (if any). The list should be ordered by billing date.

****

**Answer 7:**

*Select I.id,I.billingdate,C.Name,V.Name as ReferredByz*  
*From Invoices I*  
*Join Customer on C.Id=I.Customerid*  
*Left Join Customer V on C.id=V.ReferredBy*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Question 8:**  
  
I have a table which contains positive and negative numbers. I have to find out sum of positive and negative numbers

**Answer 8**:

*Select*  
*sum(case when num > 0 then num else null end) pos,*  
*sum(case when num < 0 then num else null end) neg*  
*from temp*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 9:**

**Example of RECURSIVE CTE**

Say you have table called  
dbo.Policy

CREATE TABLE [dbo].[Policy](  
[PolicyNo] [int] NULL,  
[StartPg] [int] NULL,  
[EndPg] [int] NULL)

It contains the following record

|  |  |  |
| --- | --- | --- |
| **PolicyNo** | **StartPg** | **EndPg** |
| 100 | 1 | 5 |
| 200 | 50 | 55 |
| 300 | 90 | 95 |

Now in another table

CREATE TABLE [dbo].[PolicyPg](  
[PolicyNo] [int] NULL,  
[PageNo] [int] NULL  
) ON [PRIMARY]

you have to insert the records dynamically determining the page range between the StartPg and EndPg from the first table i.e. [dbo].[Policy] and insert the page numbers individually for each policy number in second table i.e. [dbo].[PolicyPg]

***Desired Result****in [dbo].[PolicyPg] should look like this:*

|  |  |
| --- | --- |
| **PolicyNo** | **PageNo** |
| 100 | 1 |
| 100 | 2 |
| 100 | 3 |
| 100 | 4 |
| 100 | 5 |
| 200 | 51 |
| 200 | 52 |
| 200 | 53 |
| 200 | 54 |
| 200 | 55 |
| 200 | 50 |
| 300 | 90 |
| 300 | 91 |
| 300 | 92 |
| 300 | 93 |
| 300 | 94 |
| 300 | 95 |

**Answer 9:**

This can be achieved by making use of Recursive CTE

*With Tab as(*  
*Select PolicyNo,StartPg from Policy*  
*union all*  
*Select P.PolicyNo,T.StartPg+1 from Policy P,Tab T*  
*where P.PolicyNo=T.PolicyNo and T.StartPg+1<=P.EndPg*  
*)*

*Select \* from Tab*  
*order by PolicyNo*

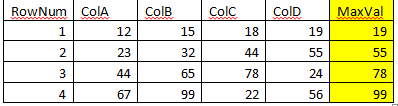
  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 10:**

There is a table called [Tab A] which contains the following record inside it

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO.** | **ColA** | **ColB** | **ColC** | **ColD** |
| 1 | 23 | 32 | 44 | 55 |
| 2 | 44 | 65 | 78 | 24 |
| 3 | 67 | 99 | 22 | 56 |
| 4 | 12 | 15 | 18 | 19 |

**Desired Result**: Get the maximum value of individual Row in a new column(highlighted in yellow) i.e.

****

**Answer 10:**

 I can think again of making use of CTE expression in this case, but experts out there can follow any approach which they feel is more simple  and easy to write

*;With tabB as*  
*(Select ColA,ColB,ColC,ColD,ROW\_NUMBER() over(order by ColA) as RowNum from TabA),*

*tabC as*  
*(Select RowNum,ColA as Col from tabB*  
*union all*  
*Select RowNum,ColB from tabB*  
*union all*  
*Select RowNum,ColC from tabB*  
*union all*  
*Select RowNum,ColD from tabB),*

*tabD as*  
*(Select B.RowNum,ColA,ColB,ColC,ColD,max(Col) as MaxVal from tabB B,tabC C where B.RowNum=C.RowNum group by  ColA,ColB,ColC,ColD,B.RowNum)*

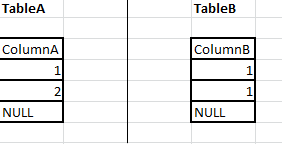
*Select \* from tabD*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Question 11:**

**Now let’s see how well you understand simple JOINS. Kindly read this carefully as there are chances of you making mistakes in this simple question.**

 Say there are 2 tables **TableA** and **TableB**



|  |
| --- |
|  |

*The main motive here is to understand that the value which are common in both table i.e. 1 in our case,so how different joins would appear.*

 A. **INNER JOIN**

*Select \* from dbo.tableA  A*  
*join tableB B on A.ColumnA=B.ColumnB*

**Answer A:**

|  |  |
| --- | --- |
| **ColumnA** | **ColumnB** |
| 1 | 1 |
| 1 | 1 |

 B. **LEFT OUTER JOIN**

*Select \* from*  
*dbo.tableA A*  
*left join tableB B on A.ColumnA=B.ColumnB*

**Answer B:**

|  |  |
| --- | --- |
| **ColumnA** | **ColumnB** |
| 1 | 1 |
| 1 | 1 |
| 2 | NULL |
| NULL | NULL |

 C. **RIGHT OUTER JOIN**

*Select \* from dbo.tableA A*  
*right join tableB B on A.ColumnA=B.ColumnB*

**Answer C:**

|  |  |
| --- | --- |
| **ColumnA** | **ColumnB** |
| 1 | 1 |
| 1 | 1 |
| NULL | NULL |

Write a SQL query to find the 10th highest employee salary from an Employee table. Explain your answer.

(Note: You may assume that there are at least 10 records in the Employee table.)

Hide answer

This can be done as follows:

SELECT TOP (1) Salary FROM

(

SELECT DISTINCT TOP (10) Salary FROM Employee ORDER BY Salary DESC

) AS Emp ORDER BY Salary

Write a SQL query using UNION ALL (**not** UNION) that uses the WHERE clause to eliminate duplicates. Why might you want to do this?

Hide answer

You can avoid duplicates using UNION ALL and still run much faster than UNION DISTINCT (which is actually same as UNION) by running a query like this:

SELECT \* FROM mytable WHERE a=X UNION ALL SELECT \* FROM mytable WHERE b=Y AND a!=X

The key is the AND a!=X part. This gives you the benefits of the UNION (a.k.a., UNION DISTINCT) command, while avoiding much of its performance hit.

What is an execution plan? When would you use it? How would you view the execution plan?

Hide answer

An execution plan is basically a road map that graphically or textually shows the data retrieval methods chosen by the SQL server’s query optimizer for a stored procedure or ad hoc query. Execution plans are very useful for helping a developer understand and analyze the performance characteristics of a query or stored procedure, since the plan is used to execute the query or stored procedure.

In many SQL systems, a textual execution plan can be obtained using a keyword such as EXPLAIN, and visual representations can often be obtained as well. In Microsoft SQL Server, the Query Analyzer has an option called “Show Execution Plan” (located on the Query drop down menu). If this option is turned on, it will display query execution plans in a separate window when a query is run.

List and explain each of the ACID properties that collectively guarantee that database transactions are processed reliably.

Hide answer

**ACID (Atomicity, Consistency, Isolation, Durability)** is a set of properties that guarantee that database transactions are processed reliably. They are defined as follows:

* **Atomicity.** Atomicity requires that each transaction be “all or nothing”: if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
* **Consistency.** The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.
* **Isolation.** The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method (i.e. if it uses strict - as opposed to relaxed - serializability), the effects of an incomplete transaction might not even be visible to another transaction.
* **Durability.** Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

What is a key difference between Truncate and Delete?

Hide answer

Truncate is used to delete table content and the action can **not** be rolled back, whereas Delete is used to delete one or more rows in the table and **can** be rolled back.

Given a table dbo.users where the column user\_id is a unique identifier, how can you efficiently select the first 100 odd user\_id values from the table?

(Assume the table contains well over 100 records with odd user\_id values.)

Hide answer

SELECT TOP 100 user\_id FROM dbo.users WHERE user\_id % 2 = 1 ORDER BY user\_id

How can you select all the even number records from a table? All the odd number records?

Hide answer

To select all the **even** number records from a table:

Select \* from table where id % 2 = 0

To select all the **odd** number records from a table:

Select \* from table where id % 2 != 0

What are the NVL and the NVL2 functions in SQL? How do they differ?

Hide answer

Both the NVL(exp1, exp2) and NVL2(exp1, exp2, exp3) functions check the value exp1 to see if it is null.

With the NVL(exp1, exp2) function, if exp1 is *not* null, then the value of exp1 is returned; otherwise, the value of exp2 is returned, but case to the same data type as that of exp1.

With the NVL2(exp1, exp2, exp3) function, if exp1 is *not* null, then exp2 is returned; otherwise, the value of exp3 is returned.

What is the difference between the RANK() and DENSE\_RANK() functions? Provide an example.

Hide answer

The *only* difference between the RANK() and DENSE\_RANK() functions is in cases where there is a “tie”; i.e., in cases where multiple values in a set have the same ranking. In such cases, RANK() will assign non-consecutive “ranks” to the values in the set (resulting in gaps between the integer ranking values when there is a tie), whereas DENSE\_RANK() will assign consecutive ranks to the values in the set (so there will be no gaps between the integer ranking values in the case of a tie).

For example, consider the set {25, 25, 50, 75, 75, 100}. For such a set, RANK() will return {1, 1, 3, 4, 4, 6} (note that the values 2 and 5 are skipped), whereas DENSE\_RANK() will return {1,1,2,3,3,4}.

What is the difference between the WHERE and HAVING clauses?

Hide answer

When GROUP BY is not used, the WHERE and HAVING clauses are essentially equivalent.

However, when GROUP BY**is** used:

* The WHERE clause is used to filter records from a result. The filtering occurs before any groupings are made.
* The HAVING clause is used to filter values from a group (i.e., to check conditions after aggregation into groups has been performed).

I usually start with two simple tables like ‘customers and orders’ or ‘books and authors’ or similar. Believe me you could ask a lot of wonderful questions with two simple tables to test SQL and analytical skills!

1. Test on aggregate functions like sum, count, etc. And test specifically on understanding of having and where
2. Test on different type of joins. Inner, left, right, self and when to use what? I usually give them a few use cases
3. Sub and correlated queries
4. Test on analytical functions (it depends - if you’re interviewing for analytics related positions then this is mandatory). Again I usually give them a few cases on this
5. If needed, questions specific to databases. How this is done in sql server? Etc.
6. If needed, questions on data modeling. Relational or dimensional also depends
7. Few questions on query optimization. How two queries are different? Which is better

These are the FEW areas that most interviewers would like to test on.

Most SQL theory interview questions revolve around these topics:

* **Indexes**
* **Partitions**
* **Normalization**
* **ACID transactions**
* **Types of keys in databases and their significance**
* **Differences between SQL and No-SQL databases**
* **SQL and No-SQL databases application real life examples**
* **A real life design problem like a Bank, School etc.**

Questions around SQL queries revolve around:

* **GROUP BY and HAVING usage**
* **Aggregate functions**
* **Cursors**
* **Nested queries**

1. Procedure may or may not return a value, it is executed explicitly. Function must return a value. Trigger executed implicitly and does not return a value.

2. Yes function can called from procedure, trigger. Procedure can called from function but trigger cannot called from function.

3.Cursor named given to a context area and it can store the result of select statement. It processed record fetch by fetch. There is Implicit cursor, Explicit cursor, Ref cursor.

4.Index is a optional structure and give direct acess to the rows. It will improve the performance of data retrival.There are many types of index 1.Cluster index2.Non cluster index3.Domail index4.Unique index5.Function based index6.Bitmap index7.Bit tree index etc.

5.Dynamic sql is that you can create a sql statement at the runtime using 'Execute Immediate' command.

6.Global variable you can create it package specification.Temporary variable you can create for temporary basis. To use this command fir you declare the variable like this variable variable\_name datatype(size), You can use it giving semiconon befor using it like :variable\_name.after the sesson end it will disapper.

7.View is a vartual table. It stores data logically. it stores query data. it is mainly used  for sequrity perposes. you cannot create index on views.there are many types of views  1.Simple views 2.Complex views 3.Force views 4.Inline views 5.Partition views 6.Horizontal views 7. Vertical views 8.Functional views 9.Materialized vierws.

Below are few SQL question that were asked to me...

1. What is difference between Stored Procedures, triggers, function.
2. Can functions be called from within procedures and triggers and vice versa.
3. What is cursor
4. What is indexing and its types.
5. Dynamic sql query.
6. Difference between temporary and global variables (and table).
7. SQL views.

## **Overview of Indexes**

An **index** is an optional structure, associated with a table or [**table cluster**](https://docs.oracle.com/cd/E11882_01/server.112/e40540/glossary.htm#CHDJGGGF), that can sometimes speed data access. By creating an [**index**](https://docs.oracle.com/cd/E11882_01/server.112/e40540/glossary.htm#i432409) on one or more columns of a table, you gain the ability in some cases to retrieve a small set of randomly distributed rows from the table. Indexes are one of many means of reducing disk I/O.

#### **Types of Indexes**

Oracle Database provides several indexing schemes, which provide complementary performance functionality. The indexes can be categorized as follows:

* B-tree indexes

These indexes are the standard index type. They are excellent for primary key and highly-selective indexes. Used as concatenated indexes, B-tree indexes can retrieve data sorted by the indexed columns. B-tree indexes have the following subtypes:

* + Index-organized tables

An index-organized table differs from a heap-organized because the data is itself the index. See ["Overview of Index-Organized Tables"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBJEBIH).

* + Reverse key indexes

In this type of index, the bytes of the index key are reversed, for example, 103 is stored as 301. The reversal of bytes spreads out inserts into the index over many blocks. See ["Reverse Key Indexes"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBFDEAJ).

* + Descending indexes

This type of index stores data on a particular column or columns in descending order. See ["Ascending and Descending Indexes"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBFFFFG).

* + B-tree cluster indexes

This type of index is used to index a table cluster key. Instead of pointing to a row, the key points to the block that contains rows related to the cluster key. See ["Overview of Indexed Clusters"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/tablecls.htm#CFABHBAG).

* Bitmap and bitmap join indexes

In a bitmap index, an index entry uses a bitmap to point to multiple rows. In contrast, a B-tree index entry points to a single row. A bitmap join index is a bitmap index for the join of two or more tables. See ["Bitmap Indexes"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBFJFDD).

* Function-based indexes

This type of index includes columns that are either transformed by a function, such as the UPPER function, or included in an expression. B-tree or bitmap indexes can be function-based. See ["Function-Based Indexes"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBGIIFB).

* Application domain indexes

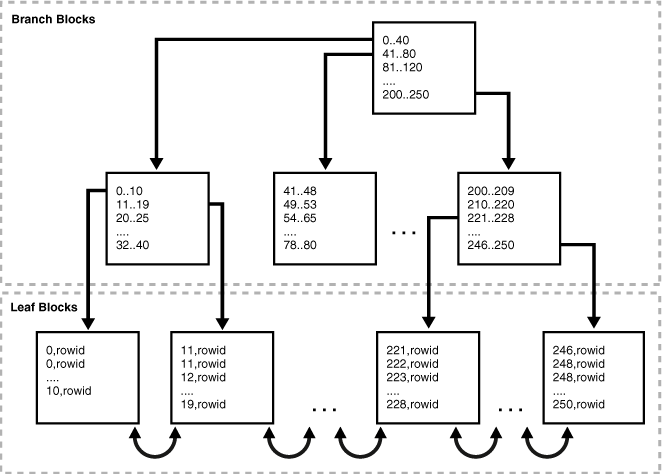
This type of index is created by a user for data in an application-specific domain. The physical index need not use a traditional index structure and can be stored either in the Oracle database as tables or externally as a file. See ["Application Domain Indexes"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CBBFEBGI).

### B-Tree Indexes

B-trees, short for **balanced trees**, are the most common type of database index. A B-tree index is an ordered list of values divided into ranges. By associating a key with a row or range of rows, B-trees provide excellent retrieval performance for a wide range of queries, including exact match and range searches.

[Figure 3-1](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#i5765) illustrates the structure of a B-tree index. The example shows an index on the department\_id column, which is a foreign key column in the employees table.

***Figure 3-1 Internal Structure of a B-tree Index***

  
[Description of "Figure 3-1 Internal Structure of a B-tree Index"](https://docs.oracle.com/cd/E11882_01/server.112/e40540/img_text/cncpt244.htm)

# T-SQL Tutorial For Beginners

01 MAY 2013 on [Sql](http://www.techumber.com/tag/sql/), [Quick Tutorial](http://www.techumber.com/tag/quick_tutorial/) |

Welcome back to another beginner tutorial. In my [Previous tutorial](http://www.techumber.com/2013/04/the-quick-sql-tutorial-for-absolute-beginners.html) I have explained you about basic SQL. In this tutorial, I will give you brief about Transact-SQL(T-SQL).

These are some differences as well as similarities between T-SQL and SQL. T-SQL is an enhanced version of SQL. In T-SQL we have SQL along with programming concepts.(SQL + Programming = T-SQL). T-SQL have most of the programming concepts like variables, loops, conditions, etc..   
Note: Oracle as has the similar concept known as PL/SQL

### 1) Variables:

Simply variable is buckets where you can store values. T-SQL support variables. You can use a variable to assign values, change values when you need. In T-SQL variables declare, assign values, retrieving values as below as below.

--Syntex:

---declaration

DECLARE @variable data-type

---Assignment

SET @variable='value'

---Retrieve

SELECT @variable

--Example

DECLARE @MyName varchar(20)

SET @MyName='Aravind'

SELECT @MyName

### 2) GO

When we execute SQL statement the will execute one by one. But in T-SQL we have a facility to execute a bunch of statement at a time. Known as Batch Execution. The main use of this is performance. Also if any one statement has syntactical error in any one statement, none of the statement will execute.

--example

USE StudentDb

INSERT INTO Student(name,email) VALUES ('aravind','aravind@techumber.com');

SELECT \* FROM Student;

GO

### 3) IF....ELSE

In programming world IF....ELSE is the basic conditional control. Have a look at the below example how we can use IF....ELSE in T-SQL.

DECLARE @temp char(2)

SELECT @temp = Gender

FROM Student

WHERE StudentId=333

IF @temp='M'

PRINT 'Hi Handsome. How are you today?'

ELSE

PRINT 'Hi beautiful. How are you today?'

### 4) BEGIN...END

BEGIN, END keyword are much like open and close parenthesis we use in high-level programming languages. Mostly we use these in condition statement where we have more than one statement to execute. See the example

IF (SELECT COUNT(\*) FROM Student WHERE Gender = 'M') > 0

BEGIN

PRINT 'This is a boys college'

PRINT ' '

SELECT fname, lname

FROM Student

END

ELSE

PRINT 'No boys'

### 5) WHILE

WHILE is used to repeatedly execute statements until the condition became false. There are many advantages of using while in T-SQL. See this example.

DECLARE @counter INT,@total INT

SET @counter=0

SET @total=SELECT COUNT(\*) FROM Student WHERE lname="Gates"

WHILE (@counter < @total)

BEGIN

UPDATE Student

SET lname= lname + CAST(@counter as char(2))

WHERE lname="Gates"

SET @counter = @counter+1

END

In this example, we finding all record with the last name as Gates and updating them with count appending to them.

### 6) CASE

In programming, we use case instead of multiple nested if..else statements. Similarly, we use CASE to evaluate an express whether it is true or false based on it uses specific values.

SELECT latitude , 'For Country' =

CASE

WHEN 'AF' THEN 'Afghanistan'

WHEN 'IN' THEN 'India'

WHEN 'RU' THEN 'Russia'

WHEN 'US' THEN 'United States'

WHEN 'LK' THEN 'Sri Lanka'

END

FROM Country

### 7) Transactions

Transactions are much like a batch. The only difference between Transaction and batch is we can undo all changes we made by statements within the single translations. We can not do this in Batch. We use COMMIT TRANSACTION or ROLLBACK TRANSACTION statement at the end. If we put ROLLBACK TRANSACTION it will undo all the changes.

BEGIN TRANSACTION

SELECT \* FROM Student

......

COMMIT TRANSACTION

### 8) Stored Procedures

Stored Procedures are a most Important concept in databases. There are many advantages using stored procedures. Mainly performance because sorted procedures are compiled and will be cached after the first time we used it. Another big advantage is we can call stored procedures from programming languages like vb,c#.

CREATE PROCEDURE uspGetStudent

@sname nvarchar(50)

AS

SELECT fname,lname,rolno

FROM Student

WHERE fname = @sname

GROUP BY fname

ORDER BY fname

GO

--Execute stored procedure

EXEC usp 'aravind'

GO

## Conclusion

These are very basic concepts of T-SQL. There are some advanced concepts like Views,Triggers,etc...