DETAILED SQL DOCUMENT

Introduction

Purpose

The purpose of this document is to provide the detailed information about oracle, Data Warehousing Concepts, Informatica and Unix, based on real – time scenarios.

ORACLE

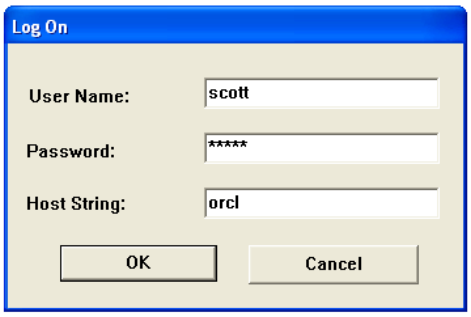
What is Oracle?

- Oracle is a database which is developed by Oracle Corporation.
- Organizations can store data on various media and in different formats, such as a hard-copy document in a filing cabinet or data stored in electronic spreadsheets or in databases.
- A database is an organized collection of information.
- To manage databases, you need database management systems (DBMS). A DBMS is a program that stores, retrieves, and modifies data in the database on request. There are four main types of databases: hierarchical, network, relational, and more recently object relational (ORDBMS).

***SEQUEL** (Structure English Query Language)*

To start with ***sqlplus*** we have to go to

Start → program files → oracle → oracle - oradb10g_home3
→ application development → sql plus.lnk



Oracle divided into 2 parts

1. **SQL** (*Structure Query Language*)
2. **PL/SQL** (*Procedure Language For SQL*)

Database : Its collection of schemas or users

Schema/User : Its collection of objects like Tables, Views etc..

Creating Schema/User :

Syntax*:*

```
Create User USERNAME  Identified By PASSWORD
Grant DBA to USERNAME
```

Example:

```
SQL> Create user abc_User identified by ABC_Password
SQL> Grant DBA  to abc_user
```

*To connect with **Scott user***

```
SQL> conn (or) connect
     Enter user-name :    scott
     Enter Password  :    tiger
```

To know number of tables or views/objects in a database:

In ***Scott@orcl,***

SQL> *select * from tab;*

SQL is divided into the following

- **DDL -** Data Definition Language : Create, Alter, Drop, Truncate, Rename
- **DML –** Data Manipulation Language : Insert, Update, Delete
- **DRL –** Data Retrieval Language : Select
- **TCL –** Transaction Control Lang. : Commit, Rollback, Savepoint
- **DCL –** Data Control Language : Grant, Revoke

Oracle Data Types:

DATA TYPES	DESCRIPTION	MAX SIZE: ORACLE 11G
VARCHAR2(SIZE)	• Variable length character string having maximum length *size* bytes. • You must specify size. We can use to store string values	• **4000** bytes • minimum is 1
N**UMBER(P,S)**	Number having precision *P* and scale *S*, which we can use to store number values	• The precision *P* can range from 1 to 38. • The scale can range from -84 to 127.
DATE	Valid Date range. We can use to store Date values	

NULL : It is not a value and does not occupy any space.

Table : Table is the basic element in a database or object in database to store the data in the form of rows and columns.

CREATE Statement

Syntax:

```
SQL> CREATE TABLE Student
  (
    Col1     Datatype,
    Col2    Datatype,
    …
    Coln    Datatype,
  );
```

Example:

```
SQL> CREATE TABLE Student
  (
    No      NUMBER(2),
    Name    VARCHAR2(10),
    Address VARCHAR2(50)
  );
```

INSERT Statement : Insert statement is used to insert the records into table.

We have two methods to insert.

- By Value Method
- By Address Method

Using Value Method

Syntax:

```
Insert into <table_name> values (val1, val2, …valn);
```

Eg:

```
SQL>  Insert into student values (1, 'Sham', 'Bang');
SQL>  Insert into student values (2, 'Mohan', 'Hyd');
```

To insert **date column**,

```
SQL>  Insert into Emp values (100, 'Sham',
                to_date('01 – 23 – 2010', 'mm – dd – yyyy'));
```

Emp

No	Name	Doj

To insert a new record again you have to type entire insert command, if there is lot of records this will be difficult. This will be avoided by using address method.

Using Address Method

Syntax:

```
Insert  into <table_name> values (&Col1, &Col2, …, &Coln);
```

This will prompt you for the values but for every insert you have to use forward slash.

***Example*:**

```
SQL> insert into student values (&no, '&name', '&address');
     Enter value for no: 1
     Enter value for name: Jagan
     Enter value for address: mpl
SQL> /
     Enter value for no: 2
     Enter value for name: Naren
     Enter value for address: Chennai
```

- **Inserting data into specified columns using Value Method**

***Syntax*:**

```
Insert into <table_name>
      (col1, col2, col3 … coln) values
                    (val1, val2, val3 ….  valn);
```

Example:

```
SQL> insert into student (no, name) values (3, 'Ramesh');
SQL> insert into student (no, name) values (4, 'Madhu');
```

- **Inserting NULL data into specified columns**

```
SQL> insert into student values (1, 'sham', NULL);
```

Here *null* is not a value you can't use it for comparison.

COMMIT Statement

- To save transaction permanently in the database we need to issue commit.
- Every DML should follow TCL.

```
SQL> commit;
```

SELECT Statement

Syntax:

```
Select * from <table_name>;        --    here * indicates all columns
        or
Select col1, col2, … coln from <table_name>;
```

Example:

To retrieve **all columns** from student table

```
SQL>  Select * from Student;
```

(Or)

```
SQL>  Select  No, Name, Address
      From    Student
```

Output

```
  NO NAME       ADDRESS
---- ---------- -------------
   1 Sham       Bang
   2 mohan      hyd
```

To retrieve **only required** from student table

```
SQL>  Select No, Name from Student;
```

Output

```
  NO NAME
---- ----------
   1 Sham
   2 mohan
```

UPDATE Statement : This can be used to modify the table data.

Syntax :

```
UPDATE <table_name>
SET <col1> = val1, <col2> = val2
WHERE <condititon>;
```

Example:

```
SQL>    UPDATE student
        SET    marks = 500;
```

If you are not specifying any condition this will update entire table.

To update specific data in a table using Where clause

Example

```
SQL>  UPDATE student
      SET marks = 500
      WHERE no = 2;
```

Example

```
SQL>  UPDATE student
      SET marks = 500, name = 'Venu'
      WHERE no = 1;
```

DELETE Statement

- This can be used to delete the table data temporarily.
- Need to issue commit since Delete is DML
- We can use delete to delete specific rows or data.
- We can use where clause with delete statement.

Syntax:

```
Delete <table_name>
where <condition>;
```

If there is no where condition, then, it will delete entire table.

Example

```
SQL>  DELETE student;
```

If there is any where condition, then according to condition, it will delete the records

Example

```
SQL>  DELETE student
      WHERE no = 2;
```

TRUNCATE Statement

- This can be used to delete the entire table data permanently.
- No need commit since truncate is DDL
- We can't use truncate to delete specific rows or data.
- We can't use where clause with truncate statement.

Syntax*:*

```
TRUNCATE TABLE <table_name>;
```

Example:

```
SQL> TRUNCATE TABLE student;
```

DROP (DDL)

- This will be used to drop the database objects;
- Drop will delete data as well structure.
- No need commit since truncate is DDL

Syntax

```
DROP TABLE <table_name>;
```

Example

```
SQL>  DROP TABLE student;
```

USING DDL

ALTER Statement

This can be used to add or remove columns and to modify the precision of the datatype.

Adding Column

Syntax

```
ALTER TABLE <table_name>
ADD <col_name> <datatype>;
```

Example

```
ALTER TABLE Student
ADD sdob DATE;
```

Removing Column

Syntax

```
ALTER TABLE <table_name>
DROP [COLUMN] <col_name>;
```

Example

```
ALTER TABLE Student
DROP COLUMN sdob;
                (OR)
ALTER TABLE Student DROP sdob;
```

Increasing or Decreasing Precision of a Column

Syntax

```
ALTER TABLE <table_name>
MODIFY <col_name> <datatype>;
```

Example

```
ALTER TABLE Student
MODIFY marks number(5);
```

Note: To decrease precision, the column should be empty.

Renaming a Column

Syntax

```
ALTER TABLE <table_name>
RENAME COLUMN <Old_Column> TO
<New_Column>;
```

Example

```
ALTER TABLE Student
RENAME COLUMN marks TO SMARKS;
```

Rename Statement: Rename statement is used to rename the database object

Syntax

```
RENAME <Old_table_name> TO
<New_table_name>;
```

Example

```
RENAME Student TO Stud;
```

USING TCL Statements

Commit : is used to save the work.

Commit is of two types.

- **Implicit:** When any DDL operation is performed, Implicit commit issued by oracle internally.
- **Explicit:** When any DML operation is performed, Explicit commit issued by the user explicitly.

Syntax / Example:

```
COMMIT  (or)  COMMIT WORK;
```

Note: *Whenever you committed, then the transaction was completed.*

Rollback : will undo the operation.

This will be applied in two methods.

- Up to previous commit
- Up to previous rollback

Syntax / Example:

```
SQL> ROLL (or) ROLL WORK;
```

(OR)

```
SQL> ROLLBACK (or) ROLLBACK WORK;
```

** While process is going on, if suddenly power goes then oracle will rollback the transaction.*

Savepoint : Savepoints used to rollback portions of your current set of transactions

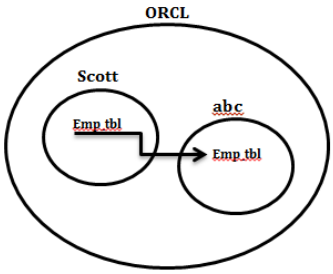
Syntax / Example:

```
SAVEPOINT <Savepoint_Name>;
```

USING DCL Statements

- DCL commands are used to granting and revoking the permissions on objects.
- To access tables from one schema to another schema.

Grant Privilege: is used to grant the privileges to the users.



- If ABC schema wants to access the object Emp_tbl, which is available in Scott schema.
- Before accessing the object, it should have privileges, which is granted by either SCOTT or DBA.
- Once, after getting the accessing privileges, the ABC user can access the object Emp_tbl, which is in Scott schema.

SCOTT@orcl

```
SQL> GRANT ON Emp_tbl TO ABC;
SQL> REVOKE ON Emp_tbl TO ABC;
```

ABC@orcl

```
SQL> SELECT * FROM SCOTT.Emp_tbl;
```

DBA@orcl

```
SQL>GRANT Select ON Emp_tbl TO ABC;
SQL>GRANT All ON Emp_tbl TO Public;
```

Syntax:

```
GRANT <privileges> on <object_name> to <user_name>
                                        [with grant option];
```

Example:

```
SQL> GRANT Select ON Student TO ABC;          -- you can give individual privilege
SQL> GRANT Select, Insert ON Student to ABC;  -- you can give set of privileges
```

The **ABC user** has to use dot method to access the object.

```
Eg: SQL> SELECT * FROM SCOTT.Student;
```

The ABC user can not grant permission on student table to other users. To get this type of option use the following.

```
Eg: SQL> GRANT ALL ON Student to ABC WITH GRANT OPTION;
```

Now, ABC user also grant permissions on Student table.

Revoke Privilege: used to revoke the privileges from the users to which you granted the privileges.

Syntax:

```
REVOKE <privileges> ON <object_name> FROM <user_name>;
```

Example:

```
SQL> REVOKE Select ON student FROM ABC;      -- you can revoke individual privilege
SQL> REVOKE Select, Insert ON Student FROM ABC; -- you can revoke set of privileges
SQL> REVOKE all ON Student FROM ABC;  -- you can revoke all privileges
```

SYNONYM:

Synonym is the similar named object. If we want to access a table which is in another schema without specifying table prefix schema name (*scott.emp*) then we use synonym *abc@orcl*.

Eg:

```
SQL> CREATE SYNONYM Emp_tbl FOR SCOTT.Emp_tbl;

SQL> SELECT * FROM Emp_tbl;
     5 rows selected
```

- First it will look for table if it doesnot exist it looks for view then Synonym.
- Where you find synonym emp, at runtime it refer to scott.emp

Conditional Selections & Operators

We have two clauses used in this

- WHERE Clause
- ORDER BY Clause

Where Clause: We can retrieve or select specific data or rows with *WHERE* clause.

Syntax:

```
SELECT * FROM <table_name>
WHERE   <Condition>;
```

The following are the different types of operators used in where clause.

- Arithmetic operators
- Comparison operators
- Logical operators

Arithmetic Operators : +, -, *, /

Comparison Operators :

- =, !=, >, <, >=, <=, <>
- BETWEEN, NOT BETWEEN
- IN, NOT IN
- NULL, NOT NULL
- LIKE

Logical Operators :

- AND
- OR
- NOT

Using =, >, <, >=, <=, !=, <>

```
SQL> SELECT * FROM Student
     WHERE DOJ = To_Date('01-23-
               2010','MM-DD-YYYY');
```

NAME	NO	DOB	MARKS
Madhu	4	20-JUL-87	

```
SQL> SELECT * FROM Student
     WHERE No < 2;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100

```
SQL> SELECT * FROM Student
     WHERE No > 2;
```

NAME	NO	DOB	MARKS
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

```
SQL> SELECT * FROM Student
     WHERE No <= 2;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400

```
SQL> SELECT * FROM Student
     WHERE No >= 2;
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

6 rows selected.

```
SQL> SELECT * FROM Student
     WHERE No != 2;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

```
SQL> SELECT * FROM Student
     WHERE No <> 2;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

Using AND : This will give the output when all the conditions become true.

Syntax

```
SELECT * FROM <table_name>
WHERE <condition - 1> AND <condition - 2> AND … <condition - n>
```

```
SQL> SELECT * FROM Student
     WHERE No = 2 AND Marks>=200;
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400

Using OR : This will gives the output when anyone of the conditions become true.

Syntax

```
SELECT * FROM <table_name>
WHERE <condition - 1> OR <condition - 2> OR … <condition - n>
```

```
SQL> SELECT * FROM Student
       WHERE No = 2 OR Marks>= 200;
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400

Using BETWEEN :

This will gives the output based on the column and its lower bound, upper bound.

Syntax

```
SELECT * FROM <table_name>
WHERE        <col_name>
BETWEEN      <Lower Bound> AND <Upper Bound>
```

```
SQL> SELECT * FROM Student
     WHERE    marks
     BETWEEN  200 AND 400;
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400

Using NOT BETWEEN :

This will gives the output based on the column which values are not in its lower bound, upper bound.

Syntax

```
SELECT * FROM <table_name>
WHERE        <col_name>
NOT BETWEEN  <Lower Bound> AND <Upper Bound>
```

```
SQL> SELECT * FROM Student
     WHERE    marks
     NOT BETWEEN  200 AND 400;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100

Using IN: This will give the output based on the column and its list of values specified.

Syntax

```
SELECT * FROM <table_name>
WHERE        <col_name>
IN           (val1, val2, …, valn)
```

```
SQL> SELECT * FROM STUDENT
     WHERE  No
     IN    (1, 2, 3);
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	

Using NOT IN:

This will give the output based on the column which values are not in the list of values specified.

Syntax

```
SELECT * FROM <table_name>
WHERE        <col_name>
NOT IN       (val1, val2, …, valn)
```

```
SQL> SELECT * FROM STUDENT
     WHERE  No
     NOT IN    (1, 2, 3);
```

NAME	NO	DOB	MARKS
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

Using NULL: This will gives the output based on the null values in the specified column.

Syntax

```
SELECT * FROM <table_name>
WHERE <col_name> IS NULL
```

```
SQL> SELECT * FROM STUDENT
     WHERE  Marks IS NULL;
```

NAME	NO	DOB	MARKS
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

Using NOT NULL:

This will gives the output based on the not null values in the specified column.

Syntax

```
SELECT * FROM <table_name>
WHERE <col_name> IS NOT NULL
```

```
SQL> SELECT * FROM STUDENT
     WHERE  Marks IS NOT NULL;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400

Using LIKE:

This will be used to search through the rows of database column based on the pattern you specify.

Syntax

```
SELECT * FROM <table_name>
WHERE <col_name> LIKE <pattern>
```

Example 1:

```
SQL> SELECT * FROM STUDENT
     WHERE  Marks LIKE 100;
```

NAME	NO	DOB	MARKS
Sudha	1	06-OCT-89	100

Example 2: This will give the rows whose name start with 'S'.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE 'S%';
```

NAME	NO	DOB	MARKS
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200

Example 3: This will give the rows whose name ends with 'h'.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE '%h';
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200
Ramesh	3	03-AUG-92	
Prakash	6	18-DEC-88	

Example 4: This will give the rows whose name's second letter start with 'a'.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE '_a%';
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	

Example 5: This will give the rows whose name's third letter start with 'd'.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE '__d%';
```

NAME	NO	DOB	MARKS
Sudha	1	06-OCT-89	100
Madhu	4	20-JUL-87	

Example 6: This will give the rows whose name's second letter start with 't' from ending.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE '%_t%';
```

NAME	NO	DOB	MARKS
Saketh	2	19-MAY-88	200

Example 7: This will give the rows whose name contains 2 a's.

```
SQL> SELECT * FROM Student
     WHERE  Name LIKE '%a%a%';
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Prakash	6	18-DEC-88	

USING ORDER BY :

This will be used to ordering the columns data (ascending or descending).

Syntax

```
SELECT * FROM <table_name>
ORDER BY <col_name> DESC;
```

- By default oracle will use *ascending order*.
- If you want output in descending order you have to use *desc* keyword after the column.

```
SQL> SELECT * FROM Student
     ORDER BY No;
```

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

```
SQL> SELECT * FROM Student
     ORDER BY No DESC;
```

NAME	NO	DOB	MARKS
Prakash	6	18-DEC-88	
Vishu	5	05-AUG-86	
Madhu	4	20-JUL-87	
Ramesh	3	03-AUG-92	
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100

ALL_OBJECTS:

- Whenever you create any object in any schema *all_objects* will make one record in it.
- *All_Objects* contains what object is created in which schema and when it is created.
- *All_Objects* will be in DBA schema

Owner	Object_name	Object_type	Date of creation	...
Scott	Files$	Table	10/02/2013	...

To access All_Objects:

Example 1:

```
SQL> SELECT * FROM All_Objects
     where Object_Name='EMP';
```

Example 2:

```
SQL> SELECT * FROM All_Objects;
```

Whenever you create a schema from admin or DBA it will grant select on all_objects to public

dba@orcl

```
SQL> GRANT SELECT ON All_Objects TO PUBLIC;
```

scott@orcl

```
SQL> CREATE SYNONYM All_Objects FOR DBA.All_Objects;
```

Advantage: We can hide the name and owner of the object.

Creating and Dropping the Synonyms

```
SQL> CREATE SYNONYM  Emp_tbl  FOR Scott.Emp_tbl;
SQL> CREATE PUBLIC SYNONYM Emp_tbl FOR Scott.Emp_tbl;
```

Login to **ABC schema** then try

```
SQL> SELECT * FROM  Emp_tbl;
SQL> DROP SYNONYM Emp_tbl;
```

DBLINK : DBLinks required to access tables from one database to another database.

Accessing object or table from another database:
Using dblink we access table of another DB. First we request to DBA to connect dblink.

Syntax:

```
SQL> CREATE  DATABASE LINK  XYZ_dblink _Name
       CONNECT  TO  User_Name IDENTIFIED BY Password
       USING  'Database_Name';
SQL> SELECT * FROM  Emp @ XYZ_dblink
```

TOAD

Toad is a GUI (Graphical User Interface) tool which is user friendly, we use toad tool for Oracle to connect to database.

<table>
<tr><td rowspan="2">Navigator</td><td>Editor (workspace)</td></tr>
<tr><td>Output</td></tr>
</table>

How to take Backup table data?

Create with select (Take Backup table data)

We can create a table using existing table [along with data] Called it as Back up tables.

Syntax:

```
CREATE TABLE <Tbl_Name_Bkp>
     (col1, col2,col3 ... coln) AS
SELECT  *  FROM    <Old_Table_Name>;
```

Example:

```
CREATE TABLE Student_Bkp
     as
SELECT * FROM Student;
```

Creating table with your own column names.

```
SQL> CREATE TABLE Student2(Sno, Sname, Smarks)
          AS
     SELECT * FROM Student;
```

Creating table with specified columns.

```
SQL> CREATE TABLE Student3
     AS
     SELECT No,Name FROM Student;
```

Creating table without table data.

```
SQL> CREATE TABLE Student2(Sno, Sname, Smarks)
     AS
     SELECT * FROM Student WHERE 1 = 2;
```

-- In where clause give any condition which does not satisfy.

INSERT WITH SELECT

- Using this we can insert existing table data to another table in a single trip.
- But the table structure should be same.

Syntax:

```
     INSERT INTO < Table_Name_2>
     SELECT * FROM <Table_Name_1>;
```

Example:

```
SQL> INSERT INTO Tbl_Student_2
     SELECT * FROM Tbl_Student_1;
```

Inserting data into specified columns

```
SQL> INSERT INTO Tbl_Student_2(NO, NAME)
     SELECT No, Name FROM Tbl_Student_1;
```

ALIASES

Column Aliases

Syntax:

```
SELECT <Orginal_Col> [AS]
     <Alias_Name> FROM
<Table_Name>;
```

Table Aliases

If you are using table aliases, you can use DOT method to the columns.

Syntax:

```
SELECT
     <Alias_Name>.<Col1>,
     <Alias_Name>.<Col2>,…,
     <Alias_Name>.<Coln>,
FROM <Table_Name> AS <Alias_Name>;
```

Example:

```
SSELECT No SNo FROM Tbl_Student;
      (or)
SELECT No AS SNo FROM
Tbl_Student;
```

Example:

```
SELECT
      S.No,
      S.Name
FROM Tbl_Student S;
```

FUNCTIONS

SQL functions are built into Oracle Database and are available for use in various appropriate SQL statements.

Functions can be categorized as follows.

- Row Level Functions
- Group Functions

Row Level Functions

Single row functions can be categorized into five. These will be applied for each row and produces individual output for each row.

- Numeric functions
- String functions
- Date functions
- Miscellaneous functions
- Conversion functions

Numeric Functions : Numeric functions accept numeric input and return numeric values.

Mod: This will give the remainder.

Syntax: **MOD** (value, divisor)

Example:

```
SELECT
      MOD(7,4), MOD(1,5), MOD(null,null), MOD(0,0), MOD(-7,4)
FROM Dual;
```

Output:

MOD(7,4)	MOD(1,5)	MOD(NULL,NULL)	MOD(0,0)	MOD(-7,4)
3	1		0	-3

Dual is dummy table, it does not have data or structure.

Example:

```
SELECT * FROM Dual;
```

Output:

D
X

Isnull:

Examples:

SQL> SELECT * FROM Emp_tbl WHERE Sal IS NULL;

SQL> SELECT * FROM Emp_tbl WHERE Sal NOT IS NULL;

NVL : It is null value function, this will substitutes the specified value in the place of null values.

Syntax: **NVL(**Null_Column, Replacement_Value**)**

SQL> SELECT * FROM Student;

-- All the marks column is null in the output

NAME	NO	DOB	MARKS
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	
Madhu	4	20-JUL-87	
Vishu	5	05-AUG-86	
Prakash	6	18-DEC-88	

SQL> SELECT
Name, No, DOB, NVL(Marks, 999)
FROM Student;

-- All the marks column is null in the output

NAME	NO	DOB	NVL(MARKS,999)
Jagan	1	10-JAN-87	99
Sudha	1	06-OCT-89	100
Saketh	2	19-MAY-88	200
Naren	2	03-APR-90	400
Ramesh	3	03-AUG-92	999
Madhu	4	20-JUL-87	999
Vishu	5	05-AUG-86	999
Prakash	6	18-DEC-88	999

Round : will rounds numbers to a given number of digits of precision.

Syntax: **ROUND** (Value, Precision)

Example:

SQL> SELECT
Round(123.2345), Round(123.2345,2), Round(123.2354,2)
FROM DUAL;

Output:

ROUND(123.2345)	ROUND(123.2345,2)	ROUND(123.2354,2)
123	123.23	123.24

Greatest: will gives the greatest number.

Syntax: **GREATEST** (val1, val2, . . . , valn)

Example:

SQL> SELECT
Greatest(1,2,3), Greatest(-1, -2, -3)
FROM DUAL;

Output:

GREATEST(1,2,3)	GREATEST(-1,-2,-3)
3	-1

Truncate : returns n truncated to m decimal places

Syntax: `TRUNC( number, [ decimal_places ] )`

Example:

```
SQL> SELECT
       TRUNC(125.815), TRUNC(125.815, 2), TRUNC(125.815, -2)
     FROM DUAL;
```

Output:

```
TRUNC(125.815) TRUNC(125.815,2) TRUNC(125.815,-2)
-------------- ---------------- -----------------
           125           125.81               100
```

String Functions:

Functions that return character values, return values of the same datatype as the input argument.

- Initcap
- Upper
- Lower
- Length
- Rpad
- Lpad
- Ltrim
- Rtrim
- Trim
- Translate
- Replace
- Soundex
- Concat ('||' – Concatenation Operator)
- Ascii
- Chr
- Substr
- Instr
- Decode
- Greatest
- Least
- Coalesce

Initcap: will capitalize the initial letter of the string.

Syntax: `INITCAP(string)`

Example:

```
SQL> SELECT INITCAP ('data warehouse') FROM DUAL;
```

Output:

```
Data Warehouse
```

Upper: will convert the string into *upper case*.

Syntax: `UPPER(string)`

Example:

```
SQL> SELECT UPPER ('data warehouse') FROM DUAL;
```

Output:

```
DATA WAREHOUSE
```

Lower: will convert the string into *lower case*.

Syntax: `LOWER(string)`

Example:

```
SQL> SELECT LOWER ('DATA WAREHOUSE') FROM DUAL;
```

Output:

```
data warehouse
```

Length: will give the length of the string.

Syntax: `LENGTH(string)`

Example:

```
SQL> SELECT LENGTH('DATA WAREHOUSE') FROM DUAL;
```

Output:

```
LENGTH('DATAWAREHOUSE')
-----------------------
                     14
```

Rpad : allows you to pad the right side of a column with any set of characters.

Syntax: `RPAD(string, length [, padding_char])`

Example:

```
SQL> SELECT RPAD('computer',15,'*'),
            RPAD('computer',15,'*#')
     FROM DUAL;
```

Output:

```
computer*******
computer*#*#*#*
```

Note: *Default padding character was blank space.*

Lpad : allows you to pad the left side of a column with any set of characters.

Syntax: `LPAD(string, length [, padding_char])`

Example:

```
SQL> SELECT LPAD('computer',15,'0'),
            LPAD('computer',15,'*#')
     FROM DUAL;
```

Output:

```
0000000computer
*#*#*#*computer
```

Note: *Default padding character was blank space.*

Ltrim: This will trim off unwanted characters from the left end of string.

Syntax: `LTRIM(string, [, unwanted_chars])`

Example:

```
SQL> SELECT LTRIM('computer', 'co')FROM DUAL;
SQL> SELECT LTRIM('computer', 'com')FROM DUAL;
SQL> SELECT LTRIM('computer','puter')FROM DUAL;
SQL> SELECT LTRIM('computer', 'omputer')FROM DUAL;
```

Output:

```
mputer
puter
computer
computer
```

If you haven't specify any unwanted characters it will display entire String.

Rtrim: This will trim off unwanted characters from the Right end of string.

Syntax: `RTRIM(string, [, unwanted_chars])`

Example:

```
SQL> SELECT RTRIM('computer', 'er')FROM DUAL;
SQL> SELECT RTRIM('computer', 'ter')FROM DUAL;
SQL> SELECT RTRIM('computer','comput')FROM DUAL;
SQL> SELECT RTRIM('computer', 'compute')FROM DUAL;
```

Output:

```
comput
compu
computer
computer
```

If you haven't specify any unwanted characters it will display entire string.

Trim : This will trim off unwanted characters from the both sides of string

Syntax: `TRIM(unwanted chars from string)`

Example: **Output:**

```
SQL> SELECT TRIM('i' from 'indiani')FROM DUAL;                    ndian

SQL> SELECT TRIM( Leading 'i' from 'indiani')FROM DUAL;           ndiani

SQL> SELECT TRIM( Trailing 'i' from 'indiani')FROM DUAL;          indian
```

Translate : This will replace the set of characters, character by character.

Syntax: TRANSLATE(String, Old_Chars, New_Chars)

Example: **Output:**

```
SQL> SELECT TRANSLATE('india','in', 'xy')FROM DUAL;              xydxa
```

Replace : This will replace the set of characters, string by string.

Syntax: REPLACE(String, Old_Chars [,New_Chars])

Example: **Output:**

```
SQL> SELECT REPLACE('india','in', 'xy')FROM DUAL;                 xydia

SQL> SELECT REPLACE('india','in')FROM DUAL;                         dia
```

Concat: This will be used to combine two string only.

Syntax: CONCAT(String1, String2)

Example: **Output:**

```
SQL> SELECT CONCAT('computer','operator')FROM DUAL;          computeroperator
```

If you want to combine more than two strings you have to use concatenation Operator(||).

Example: **Output:**

```
SQL> SELECT 'How' ||' are' || ' you…???' FROM DUAL;          How are you...???
```

ASCII: This will return the decimal representation in the database character set of the first character of the string.

Syntax: ASCII(String)

Example: **Output:**

```
SQL> SELECT ASCII('a'), ASCII('apple')FROM
DUAL;

ASCII('A') ASCII('APPLE')
---------- --------------
        97             97
```

Char: This will return the character having the binary equivalent to the string in either the database character set or the national character set.

Syntax: CHR(Number)

Example: **Output:**

```
SQL> SELECT CHR(97)FROM DUAL;
C
-
a
```

Substr: This will be used to extract substrings.

Syntax: SUBSTR (string, start_chr_count [, no_of_chars])

Example: **Output:**

```
SQL> SELECT SUBSTR('computer',2) FROM DUAL;      omputer

SQL> SELECT SUBSTR('computer',2,5) FROM DUAL;     omput
SQL> SELECT SUBSTR('computer',3,7) FROM DUAL;     mputer
```

Instr: This will allows you for searching through a string for set of characters.

Syntax: INSTR (string, search_str [, start_chr_count [, occurrence]])

Example:

```
SQL> SELECT INSTR('information','o',4,1)
     FROM DUAL;
```

Output:

```
INSTR('INFORMATION','O',4,1)
----------------------------
                           4
```

```
SQL> SELECT INSTR('information','o',4,2)
     FROM DUAL;
```

```
INSTR('INFORMATION','O',4,2)
----------------------------
                          10
```

- If you are not specifying *start_chr_count* and *occurrence* then it will start search from the beginning and finds first occurrence only.
- If both parameters *start_chr_count* and *occurrence* are null, it will display nothing.

Decode: Decode will act as value by value substitution. For every value of field, it will checks for a match in a series of if/then tests.

Syntax: `DECODE (value, if1, then1, if2, then2, ……. else);`

```
SQL> SELECT Supplier_Id,
    DECODE(Supplier_Id,  10000, 'IBM',
                    10001, 'Microsoft',
                    10002, 'Hewlett Packard',
                    'Gateway') Supplier_Name
     FROM Suppliers;
```

```
SUPPLIER_ID SUPPLIER_NAME
----------- ---------------
      10000 IBM
      10001 Microsoft
      10002 Hewlett Packard
      10003 Gateway
```

```
SQL> SELECT DECODE(1,1,3),
DECODE(1,2,3,4,4,6)
     FROM DUAL;
```

```
DECODE(1,1,3) DECODE(1,2,3,4,4,6)
------------- -------------------
            3                   6
```

- If the number of parameters are odd and different then decode will display nothing.
- If the number of parameters are even and different then decode will display last Value.
- If all the parameters are null then decode will display nothing.
- If all the parameters are zeros then decode will display zero.

Greatest: This will give the greatest string.

Syntax: `GREATEST (String 1, String 2, String 3,..., String n);`

Example:

```
SQL> SELECT
          GREATEST('a', 'b', 'c'),
          GREATEST('satish','srinu','saketh')
     FROM DUAL;
```

Output:

```
c
srinu
```

- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

Least: This will give the least string.

Syntax: `LEAST (String1, String2, String3, ..., Stringn);`

Example:

```
SQL> SELECT LEAST('a', 'b', 'c'),
          LEAST('satish','srinu','saketh')
     FROM DUAL;
```

Output:

```
a
saketh
```

- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

Date Functions: Oracle default date format is DD – MON – YY.

- Sysdate
- Current_date
- Current_timestamp
- Systimestamp
- Localtimestamp
- Dbtimezone
- Sessiontimezone
- To_char
- To_date
- Add_months
- Months_between
- Next_day
- Last_day
- Extract
- Greatest
- Least
- Round
- Trunc
- New_time
- Coalesce

Sysdate: This will give the current date and time.

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
25-SEP-13
```

Current_Date: This will returns the current date in the session's timezone.

```
SQL> SELECT CURRENT_DATE FROM DUAL;

25-SEP-13
```

Current_TimeStamp:

This will returns the current timestamp with the active time zone information.

```
SQL> SELECT CURRENT_TIMESTAMP FROM DUAL;
```

OUTPUT:

```
CURRENT_TIMESTAMP
-----------------------------------
25-SEP-13 12.51.21.037000 AM +05:30
```

SysTimeStamp:

This will returns the system date, including fractional seconds and time zone of the database.

```
SQL> SELECT SYSTIMESTAMP FROM DUAL;
```

OUTPUT:

```
SYSTIMESTAMP
-----------------------------------
25-SEP-13 12.54.51.032000 AM +05:30
```

Date Formats

- D -- No of days in week
- DD -- No of days in month
- DDD -- No of days in year
- MM -- No of month
- MON -- Three letter abbreviation of month
- MONTH -- Fully spelled out month
- RM -- Roman numeral month
- DY -- Three letter abbreviated day
- DAY -- Fully spelled out day
- Y -- Last one digit of the year
- YY -- Last two digits of the year
- YYY -- Last three digits of the year
- YYYY -- Full four digit year
- SYYYY -- Signed year
- I -- One digit year from ISO standard
- IY -- Two digit year from ISO standard
- IYY -- Three digit year from ISO standard
- IYYY -- Four digit year from ISO standard
- Y, YYY -- Year with comma
- YEAR -- Fully spelled out year
- CC -- Century
- Q -- No of quarters
- W -- No of weeks in month
- WW -- No of weeks in year
- IW -- No of weeks in year from ISO standard
- HH -- Hours
- MI -- Minutes
- SS -- Seconds
- FF -- Fractional seconds
- AM or PM -- Displays AM or PM depending upon time of day
- A.M or P.M -- Displays A.M or P.M depending upon time of day
- AD or BC -- Displays AD or BC depending upon the date
- A.D or B.C -- Displays AD or BC depending upon the date
- FM -- Prefix to month or day, suppresses padding of month or day
- TH -- Suffix to a number
- SP -- suffix to a number to be spelled out
- SPTH -- Suffix combination of TH and SP to be both spelled out
- THSP -- same as SPTH

Examples:

SQL> SELECT TO_CHAR(SYSDATE,'DD MONTH YEAR') FROM DUAL;

Output:

```
TO_CHAR(SYSDATE,'DDMONTHYEAR')
------------------------------
25 september twenty thirteen
```

SQL> SELECT TO_CHAR(SYSDATE,'DD FMMONTH YEAR') FROM DUAL;

Output:

```
TO_CHAR(SYSDATE,'DDFMMONTHYEAR')
--------------------------------
25 SEPTEMBER TWENTY THIRTEEN
```

SQL> SELECT TO_CHAR(SYSDATE,'DDth') FROM DUAL;

Output:

```
TO_CHAR(SYSDATE,'DDth')
-----------------------
25TH
```

To_Date: This is used to convert the String into Date format.

Syntax: TO_DATE (date,'date_format');

Example:

```
SELECT TO_CHAR(
      TO_DATE('24/DEC/2006','DD/MON/YYYY'), 'DD * MONTH * DAY')
FROM DUAL;
```

OUTPUT:

```
TO_CHAR(TO_DATE('24/DEC/2013','DD/MON/YYYY'), 'DD * MONTH * DAY')
-----------------------------------------------------------------
24 * DECEMBER  * TUESDAY
```

If you are not using to_char oracle will display output in default date format

Add_Months: Adds the specified months to the given date.

Syntax: ADD_MONTHS (date, No_Of_Months);

Example:

```
SELECT ADD_MONTHS(
      TO_DATE('24/DEC/2006','DD/MON/YYYY'), 5)
FROM DUAL;
```

OUTPUT:

```
ADD_MONTHS(TO_DATE('24/DEC/2006','DD/MON/YYYY'), 5)
---------------------------------------------------
24-MAY-07
```

Example:

```
SELECT ADD_MONTHS(
      TO_DATE('24/DEC/2006','DD/MON/YYYY'), -5)
FROM DUAL;
```

OUTPUT:

```
ADD_MONTHS(TO_DATE('24/DEC/2006','DD/MON/YYYY'),-5)
---------------------------------------------------
24-JUL-06
```

- If no_of_months is zero then it will display the same date.
- If no_of_months is null then it will display nothing

Months_Between: This will give difference of months between two dates.

Syntax: MONTHS_BETWEEN (Date 1, Date 2);

Example:

```
SELECT MONTHS_BETWEEN(
      TO_DATE('24/AUG/1990','DD/MON/YYYY'),
      TO_DATE('11/JAN/1990','DD/MON/YYYY')) MONTHS_BETWEEN
FROM DUAL;
```

OUTPUT:

```
MONTHS_BETWEEN
--------------
    7.41935484
```

Example:

```
SELECT MONTHS_BETWEEN(
      TO_DATE('11/JAN/1990','DD/MON/YYYY'),
      TO_DATE('24/AUG/1990','DD/MON/YYYY')) MONTHS_BETWEEN
FROM DUAL;
```

OUTPUT:

```
MONTHS_BETWEEN
--------------
   -7.4193548
```

Next_Day: This will produce next day of the given day from the specified date.

Syntax: NEXT_DAY (Date, Day);

Example:

```
SELECT NEXT_DAY(
      TO_DATE('24/AUG/1990','DD/MON/YYYY'), 'SUN')AS NEXT_DAY
FROM DUAL;
```

OUTPUT:

```
NEXT_DAY
---------
26-AUG-90
```

If the day parameter is null then it will display nothing.

Last_Day: This will produce last day of the given day.

Syntax: LAST_DAY (Date);

Example:

```
SELECT LAST_DAY(
TO_DATE('24/AUG/1990','DD/MON/YYYY'))AS LAST_DAY FROM DUAL;
```

OUTPUT:

```
LAST_DAY
---------
31-AUG-90
```

Greatest: This will give the greatest date.

Syntax: `GREATEST(Date 1, Date 2, ..., Date n);`

Example:

```
SELECT GREATEST(
      TO_DATE('11/Jan/1990','DD/MON/YYYY'),
      TO_DATE('11/Mar/1990','DD/MON/YYYY'),
      TO_DATE('11/Apr/1990','DD/MON/YYYY')) AS GREATEST
FROM DUAL;
```

OUTPUT:

```
GREATEST
---------
11-APR-90
```

Least: This will give the least date.

Syntax: `LEAST(Date 1, Date 2, ..., Date n);`

Example:

```
SELECT LEAST(
      TO_DATE('11/Jan/1990','DD/MON/YYYY'),
      TO_DATE('11/Mar/1990','DD/MON/YYYY'),
      TO_DATE('11/Apr/1990','DD/MON/YYYY')) AS LEAST
FROM DUAL;
```

OUTPUT:

```
LEAST
---------
11-JAN-90
```

Trunc : Trunc will chops off the date to which it was equal to or less than the given date.

Syntax: `TRUNC(Date, (Day | Month | Year))`

- If the second parameter was *year* then it always returns the first day of the current year.
- If the second parameter was *month* then it always returns the first day of the current month.
- If the second parameter was *day* then it always returns the previous Sunday.
- If the second parameter was null then it returns nothing.

If you are not specifying the second parameter then trunk will resets the time to the beginning of the current day.

Example:

```
SQL> SELECT
      TRUNC(TO_DATE('24-DEC-04','DD-MON-YY'),'YEAR'),
      TRUNC(TO_DATE('11-MAR-06','DD-MON-YY'),'YEAR')
FROM DUAL;
```

Output:

```
TRUNC(TO_DATE('24-DEC-04','DD-MON-YY'),'YEAR')
----------------------------------------------
01-JAN-04

TRUNC(TO_DATE('11-MAR-06','DD-MON-YY'),'YEAR')
----------------------------------------------
01-JAN-06
```

Example:

```
SQL> SELECT
          TRUNC(TO_DATE('11-JAN-04','DD-MON-YY'),'MONTH'),
          TRUNC(TO_DATE('18-JAN-04','DD-MON-YY'),'MONTH')
     FROM DUAL;
```

Output:

```
TRUNC(TO_DATE('11-JAN-04','DD-MON-YY'),'MONTH')
-----------------------------------------------
01-JAN-04

TRUNC(TO_DATE('18-JAN-04','DD-MON-YY'),'MONTH')
-----------------------------------------------
01-JAN-04
```

Miscellaneous Functions:

Rank : Rank() function will give the non – sequential ranking.

Example:

```
SQL> SELECT NAME, NUM, SALARY, R FROM
          (SELECT NAME, NUM, SALARY, RANK()
                    OVER (ORDER BY SALARY DESC) R
            FROM EMP);
```

Output:

NAME	NUM	SALARY	R
Russell	10	8800	1
Sara	7	7000	2
Ben	6	7000	2
James	9	6500	4
Valaire	8	5500	5
Tom	1	4000	6
John	3	3500	7
Sam	4	3500	7
Todd	5	2800	9

Dense_Rank : Dense_Rank() function will give the sequential ranking.

Example:

```
SQL> SELECT NAME, NUM, SALARY, R FROM
          (SELECT NAME, NUM, SALARY, DENSE_RANK()
                    OVER (ORDER BY SALARY DESC) R
            FROM EMP);
```

Output:

NAME	NUM	SALARY	R
Russell	10	8800	1
Sara	7	7000	2
Ben	6	7000	2
James	9	6500	3
Valaire	8	5500	4
Tom	1	4000	5
John	3	3500	6
Sam	4	3500	6
Todd	5	2800	7

Group By & Having Clause

Group By :

- Using group by, we can create groups of related information.
- Columns used in select must be used with group by, otherwise it was not a group by expression.

Example:

```
SQL> SELECT       DEPARTMENT_ID, SUM(SALARY)
     FROM         EMPLOYEES
     GROUP BY     DEPARTMENT_ID;
```

Output:

DEPARTMENT_ID	SUM(SALARY)
100	51600
30	24900
	7000
90	58000
20	19000
70	10000
110	20300
50	156400
80	304500
40	6500
60	28800
10	4400

Example:

```
SQL> SELECT       DEPARTMENT_ID, JOB_ID, SUM(SALARY)
     FROM         EMPLOYEES
     GROUP BY     DEPARTMENT_ID, JOB_ID
     ORDER BY     DEPARTMENT_ID;
```

Output:

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
30	PU_CLERK	13900
30	PU_MAN	11000
40	HR_REP	6500
50	SH_CLERK	64300
50	ST_CLERK	55700
50	ST_MAN	36400
60	IT_PROG	28800
70	PR_REP	10000
80	SA_MAN	61000
80	SA_REP	243500
90	AD_PRES	24000
90	AD_VP	34000
100	FI_ACCOUNT	39600
100	FI_MGR	12000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

HAVING

- This will work as where clause which can be used only with group by because of absence of where clause in group by.

- On top of the Group by if we want to filter the groups then we use having clause.

Example:

```
SQL> SELECT      Num, Name, Place, COUNT(*)
     FROM        EMP
     GROUP BY    Num, Name, Place
     HAVING      COUNT(*)>1;
```

Output:

```
       NUM NAME                 PLACE                COUNT(*)
---------- -------------------- -------------------- ----------
        10 Russell              London                        2
         3 John                 London                        2
         6 Ben                  New York                      3
         7 Sara                 Sydney                        3
```

Example:

```
SQL> SELECT      Employee_ID,Job_Id,Sum(Salary) Total_Salary
     FROM        Employees
     GROUP BY    Employee_ID, Job_ID
     HAVING      SUM(Salary) >12000
     order by JOB_ID;
```

Output:

```
EMPLOYEE_ID JOB_ID     TOTAL_SALARY
----------- ---------- ------------
        100 AD_PRES            24000
        101 AD_VP              17000
        102 AD_VP              17000
        201 MK_MAN             13000
        145 SA_MAN             14000
        146 SA_MAN             13500
```

Order of execution

- Group the rows together based on group by clause.
- Calculate the group functions for each group.
- Choose and eliminate the groups based on the having clause.
- Order the groups based on the specified column.

Set Operators : Used To Combine two queries we need set operators.

Types of Set Operators

- Union
- Union all
- Intersect
- Minus

Union : This will combine the records of multiple tables having the same structure.

Example:

```
SQL> SELECT * FROM Student1
          UNION
     SELECT * FROM Student2;
```

Union All: This will combine the records of multiple tables having the same structure but including duplicates.

Example:

```
SQL> SELECT * FROM Student1
          UNION ALL
     SELECT * FROM Student2;
```

Intersect: This will give the common records of multiple tables having the same structure.

Example:

```
SQL> SELECT * FROM Student1
          INTERSECT
     SELECT * FROM Student2;
```

Minus: This will give the records of a table whose records are not in other tables having the same structure.

Example:

```
SQL> SELECT * FROM Student1
          MINUS
     SELECT * FROM Student2;
```

Constraints: Constraints required maintaining consistence data in a database.

Constraints are categorized as follows.

- **Domain integrity constraints**
  - ✓ Not null
  - ✓ Check
- **Entity integrity constraints**
  - ✓ Unique
  - ✓ Primary key
- **Referential integrity constraints**
  - ✓ Foreign key

Constraints are always attached to a column not a table.

We can add constraints in three ways.

- ✓ **Column level** -- *along with the column definition*
- ✓ **Table Level** -- *after the table definition*
- ✓ **Alter level** -- *using alter command*

- While adding constraints you need not specify the name but the type only, oracle will internally name the constraint.
- If you want to give a name to the constraint, you have to use the constraint clause.

Not Null: used to avoid null values. We can add this constraint in *column level only*.

Example:

```
CREATE TABLE Student(
    No      Number(2) Not Null,
    Name    Varchar(10),
    Marks   Number(3)
);
```

Example:

```
CREATE TABLE Student (
     No    Number(2) CONSTRAINT
           NN_Costraint Not Null,
     Name  Varchar(10),
     Marks Number(3)
);
```

Check: used to insert the values based on specified condition. We can add this constraint in all three levels.

COLUMN LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2),
 Name    Varchar(10),
 Marks   Number(3)CHECK (Marks > 300)
);
```

Example:

```
CREATE TABLE Student (
     No    Number(2),
     Name  Varchar(10),
     Marks Number(3)
     CONSTRAINT Chk_Constraint
     CHECK (Marks > 300)
);
```

Table LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2),
 Name    Varchar(10),
 Marks   Number(3),
 CHECK (Marks > 300)
);
```

Example:

```
CREATE TABLE Student (
     No    Number(2),
     Name  Varchar(10),
     Marks Number(3),
     CONSTRAINT Chk_Constraint
     CHECK (Marks > 300)
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE Student ADD CHECK (Marks > 300);    (OR)

SQL> ALTER TABLE Student ADD
          CONSTRAINT Chk_Constraint CHECK (Marks > 300);
```

Unique: used to avoid duplicates but it allow nulls. We can add this constraint in all three levels.

COLUMN LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2) UNIQUE,
 Name    Varchar(10),
 Marks   Number(3)
);
```

Example:

```
CREATE TABLE Student (
  No    Number(2) CONSTRAINT
          Unique_Constraint UNIQUE,
  Name  Varchar(10),
  Marks Number(3)
);
```

Table LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2),
 Name    Varchar(10),
 Marks   Number(3),
 UNIQUE (No)
);
```

Example:

```
CREATE TABLE Student (
  No    Number(2),
  Name  Varchar(10),
  Marks Number(3),
  CONSTRAINT Unq_Constraint
                    UNIQUE (No)
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE Student ADD UNIQUE (No);    (OR)
SQL> ALTER TABLE Student ADD CONSTRAINT Unq_Constraint UNIQUE (No);
```

Primary Key

- This is used to avoid duplicates and nulls. This will work as combination of unique and not null.
- Primary key always attached to the parent table.
- Primary key will create Unique index by default.
- We can add this constraint in all three levels.

COLUMN LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2) PRIMARY KEY,
 Name    Varchar(10),
 Marks   Number(3)
);
```

Example:

```
CREATE TABLE Student (
  No    Number(2) CONSTRAINT
       PKey_Constraint PRIMARY KEY,
  Name  Varchar(10),
  Marks Number(3)
);
```

Table LEVEL

Example:

```
CREATE TABLE Student(
 No      Number(2),
 Name    Varchar(10),
 Marks   Number(3),
 PRIMARY KEY (No)
);
```

Example:

```
CREATE TABLE Student (
  No    Number(2),
  Name  Varchar(10),
  Marks Number(3),
  CONSTRAINT PKey_Constraint
              PRIMARY KEY(No)
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE Student ADD PRIMARY KEY (No);    (OR)
SQL> ALTER TABLE Student ADD CONSTRAINT PKey_Constraint PRIMARY KEY (No);
```

Foreign Key

- This is used to reference the parent table primary key column which allows duplicates.
- Foreign key always attached to the child table.
- We can add this constraint in *table and alter levels* only.

Table LEVEL

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  PRIMARY KEY(Empno),
  FOREIGN KEY(Deptno)
          REFERENCES DEPT(Deptno)
);
```

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  CONSTRAINT PKey_Constraint
          PRIMARY KEY(Empno),
  CONSTRAINT FKey_Constraint
        FOREIGN KEY(Deptno)
         REFERENCES DEPT(Deptno)
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE EMP ADD FOREIGN KEY (Deptno)
                          REFERENCES DEPT(Deptno);    (OR)
SQL> ALTER TABLE EMP ADD CONSTRAINT FKey_Constraint
                    FOREIGN KEY (Deptno) REFERENCES DEPT(Deptno);
```

Once the primary key and foreign key relationship has been created then you cannot remove any parent record if the dependent child's exists.

Using On Delete Cascade

- By using this clause you can remove the parent record even it child's exists.
- Because whenever you remove parent record oracle automatically removes all its dependent records from child table, if this clause is present while creating foreign key constraint.

Table LEVEL

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  PRIMARY KEY(Empno),
```

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  CONSTRAINT PKey_Constraint
```

```
  FOREIGN KEY(Deptno)
          REFERENCES DEPT(Deptno)
          ON DELETE CASCADE
);
```

```
          PRIMARY KEY(Empno),
     CONSTRAINT FKey_Constraint
          FOREIGN KEY(Deptno)
           REFERENCES DEPT(Deptno)
           ON DELETE CASCADE
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE EMP ADD FOREIGN KEY (Deptno)
          REFERENCES DEPT(Deptno) ON DELETE CASCADE;    (OR)

SQL> ALTER TABLE EMP ADD CONSTRAINT FKey_Constraint
          FOREIGN KEY (Deptno) REFERENCES DEPT(Deptno)
          ON DELETE CASCADE;
```

Composite Keys

- A Composite Key can be defined on a combination of columns.
- Composite key can be defined in table and alter levels only.

Table LEVEL

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  UNIQUE(Empno, Ename)
);
```

Example:

```
CREATE TABLE EMP(
  Empno Number(2),
  Ename Varchar(10),
  Deptno Number(2),
  CONSTRAINT CompKey_Constraint
          UNIQUE(Empno, Ename)
);
```

Alter LEVEL

Example:

```
SQL> ALTER TABLE EMP ADD UNIQUE (Empno, Ename);    (OR)

SQL> ALTER TABLE EMP ADD CONSTRAINT Comp_Key_Constraint
                         UNIQUE(Empno, Ename);
```

JOINS

- The purpose of a join is to combine the data across tables.
- A join is actually performed by the where clause which combines the specified rows of tables.
- If a join involves in more than two tables then oracle joins first two tables based on the joins condition and then compares the result with the next table and so on.

Types

- Equi join
- Non-equi join
- Self join
- Natural join
- Cross join
- Outer join
  - Left outer
  - Right outer
  - Full outer
- Inner join
- Using clause
- On clause

Assume that we have the following tables.

```
SQL> SELECT * FROM EMPLOYEES;
```

EMPNO	ENAME	JOB	MGR	DEPTNO
111	Saketh	Analyst	444	10
222	Sudha	Clerk	333	20
333	Jagan	Manager	111	10
444	Madhu	Engineer	222	40

```
SQL> SELECT * FROM DEPARTMENT;
```

DEPTNO	DNAME	LOCATION
10	MKT	Hyderabad
20	FIN	Bangalore
30	HR	Bombay

Equi Join: A join which contains an '=' operator in the joins condition.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E, DEPARTMENT D
     WHERE  E.Deptno = D.Deptno;
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad

Using Clause:

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E JOIN DEPARTMENT D
     USING  (Deptno);
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad

On Clause:

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno);
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad

Non – Equi Join : A join which contains an operator other than '=' in the joins condition.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E, DEPARTMENT D
     WHERE  (E.Deptno > D.Deptno);
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
222	Sudha	Clerk	MKT	Hyderabad
444	Madhu	Engineer	MKT	Hyderabad
444	Madhu	Engineer	FIN	Bangalore
444	Madhu	Engineer	HR	Bombay

Self Join : Joining the table itself is called self join.

Example:

```
SQL> SELECT E1.Empno, E2.Ename
     FROM   EMPLOYEES E1, EMPLOYEES E2
     WHERE  E1.Mgr = E2.Empno;
```

Output:

EMPNO	ENAME
333	Saketh
444	Sudha
222	Jagan
111	Madhu

Natural Join: Natural join compares all the common columns.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES  NATURAL JOIN DEPARTMENT;
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad

Cross Join: This will gives the cross product.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES  CROSS JOIN DEPARTMENT;
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	MKT	Hyderabad
333	Jagan	Manager	MKT	Hyderabad
444	Madhu	Engineer	MKT	Hyderabad
111	Saketh	Analyst	FIN	Bangalore
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	FIN	Bangalore
444	Madhu	Engineer	FIN	Bangalore
111	Saketh	Analyst	HR	Bombay
222	Sudha	Clerk	HR	Bombay
333	Jagan	Manager	HR	Bombay
444	Madhu	Engineer	HR	Bombay

OUTER JOIN

- Outer join gives the non-matching records along with matching records from both the tables.
- It is divided into Left, Right and Full outer join.

Left Outer Join

This will display the all matching records from both the tables and also non matching records from opposite side outer join symbol (+) table.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E LEFT OUTER JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno);
                              (OR)
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E, DEPARTMENT D
     WHERE  (E.Deptno = D.Deptno(+));
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
333	Jagan	Manager	MKT	Hyderabad
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
444	Madhu	Engineer		

Right Outer Join

This will display the all matching records and the records which are in right hand side table those that are not in left hand side table.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E RIGHT OUTER JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno);

                          (OR)

SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E, DEPARTMENT D
     WHERE  (E.Deptno(+) = D.Deptno);
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad
			HR	Bombay

Full Outer Join

This will display the all matching records and the non – matching records from both tables. Union between Left and Right outer joins nothing but Full Outer join.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E LEFT OUTER JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno)
               UNION
     SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E RIGHT OUTER JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno);

                          (OR)

SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E FULL OUTER JOIN DEPARTMENT D
     ON     (E.Deptno = D.Deptno)
```

Output:

EMPNO	ENAME	JOB	DNAME	LOCATION
111	Saketh	Analyst	MKT	Hyderabad
222	Sudha	Clerk	FIN	Bangalore
333	Jagan	Manager	MKT	Hyderabad
444	Madhu	Engineer		
			HR	Bombay

Inner Join : This will display all the records that have matched.

Example:

```
SQL> SELECT Empno, Ename, Job, Dname, Location
     FROM   EMPLOYEES E INNER JOIN DEPARTMENT D
     USING  (Deptno);
```

Output:

```
EMPNO ENAME                JOB                 DNAME      LOCATION
------ -------------------- ------------------- ---------- ----------
   111 Saketh               Analyst             MKT        Hyderabad
   222 Sudha                Clerk               FIN        Bangalore
   333 Jagan                Manager             MKT        Hyderabad
```

CASE ( ) or DECODE ( )

Case( ) : Case is similar to decode but easier to understand while going through coding.

Example:

```
SQL> SELECT Salary,
            CASE Salary
                 WHEN 2500 THEN 'Low'
                 WHEN 4000 THEN 'High'
                 ELSE 'Medium'
            END CASE
       FROM   EMPLOYEES;
```

Output:

```
    SALARY CASE
---------- ------
      2500 Low
      3000 Medium
      3500 Medium
      4000 High
```

Decode( ) :

Example:

```
SQL> SELECT Salary,
            DECODE(Salary,  2500,'Low',
                            4000,'High',
                            'Medium') AS GRADE
       FROM   EMPLOYEES;
```

Output:

```
    SALARY GRADE
---------- ------
      2500 Low
      3000 Medium
      3500 Medium
      4000 High
```

Example:

```
SQL> SELECT Location,
            DECODE(Location,  'Hyderabad','Hyd',
                              'Bangalore','Bang',
                             'Address') AS Loc
       FROM   DEPARTMENT;
```

Output:

```
LOCATION           LOC
------------------ -------
Hyderabad          Hyd
Bangalore          Bang
Bombay             Address
```

Merge Statement:

You can use merge command to perform insert and update in a single command.

Example:

```
SQL> MERGE INTO Student1 S1
        USING (SELECT * FROM Student2) S2
        ON (S1.NO = S2.NO)
        WHEN MATCHED THEN
           UPDATE
              SET Marks = S2.Marks
        WHEN NOT MATCHED THEN
           INSERT (S1.NO, S1.NAME, S1.Marks)
           VALUES (S2.NO, S2.NAME, S2.Marks);
```

SUB – QUERIES, CO – RELATED QUERIES and EXISTS clause

Sub Queries : If we write select statement in where clause that we called it as sub queries or inner queries.

Types

- Single Row Sub – Queries
- Multi Row Sub – Queries
- Correlated Sub – Queries

Single Row Sub – Queries : In single row *Sub - Query*, it will return one value.

Example:

```
SQL>  SELECT * FROM Employees
       WHERE Salary > (SELECT Salary FROM Employees
                         WHERE Empno = 333);
```

Output:

EMPNO	ENAME	JOB	MGR	DEPTNO	SALARY
444	Madhu	Engineer	222	40	4000

Multi Row Sub – Queries

In *multi row sub - query*, it will return more than one value. In such cases we should include operators like any, all, in or not in between the comparison operator and the *sub - query*. We can't use = operator for multi row sub - query.

Example:

```
SQL>  SELECT * FROM Employees
      WHERE Salary > ANY (SELECT Salary FROM Employees
                             WHERE Salary BETWEEN 2500 AND 3000);
```

Output:

EMPNO	ENAME	JOB	MGR	DEPTNO	SALARY
444	Madhu	Engineer	222	40	4000
333	Jagan	Manager	111	10	3500
222	Sudha	Clerk	333	20	3000

Correlated Sub – Queries

A ***Sub – Query*** is evaluated only once for the entire parent statement where as a correlated ***Sub – Query*** is evaluated once for each and every row processed by the parent statement.

Example: Find all employees who earn more than the average salary in their department.

```
SQL> SELECT Last_Name, Salary, Department_Id
       FROM EMPLOYEES a
      WHERE Salary > (SELECT   AVG (Salary)
                        FROM EMPLOYEES b
                       WHERE b.Department_Id = a.Department_Id
                       GROUP BY b.Department_Id)
```

Exists :

The Exists operator tests for existence of rows in the results set of the Sub - Query.

```
SQL> SELECT Department_Name FROM DEPARTMENTS
      WHERE EXISTS (SELECT 1 FROM EMPLOYEES
                     WHERE DEPARTMENTS.DEPARTMENT_ID =
                           EMPLOYEES.DEPARTMENT_ID);
```

Output:

```
DEPARTMENT_NAME
------------------------
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
Accounting
```

VIEWS, INLINE VIEWS AND MATERIALIZED VIEWS

Views

- A view is a database object that is a logical representation of a table.
- It is delivered from a table but has no storage of its own.
- View will fetch the date from base table. It will run the Base query.
- A view takes the output of the query and treats it as a table

Types

  - ✓ Simple view
  - ✓ Complex view
- Simple view can be created from one table where as complex view can be created from multiple tables.
- We can do DML on Simple view but not on Complex views.

WHY VIEWS?

- Provides additional level of security by restricting access to a predetermined set of rows and/or columns of a table.

Example:

```
SQL> CREATE VIEW dept_v
     AS
       SELECT * FROM dept;

SQL> CREATE VIEW dept_v
     AS
        SELECT   deptno, SUM (sal) t_sal FROM emp
        GROUP BY deptno;

SQL> SELECT * FROM dept_v;
```

Dropping Views

```
SQL> DROP VIEW dept_v
```

Inline View:

If we write a select statement in from clause that is nothing but inline view.

Example:

```
SQL> SELECT a.First_Name, a.Employee_ID, b.Salary, b.Department_ID
     FROM Employees a,
            (SELECT   MAX (Salary) Salary, Department_ID FROM Employees
             GROUP BY Department_ID) b
      WHERE a.Salary = b.Salary AND a.Department_ID = b.Department_ID;
```

Output:

FIRST_NAME	EMPLOYEE_ID	SALARY	DEPARTMENT_ID
Steven	100	24000	90
Alexander	103	9000	60
Nancy	108	12000	100
Den	114	11000	30
Adam	121	8200	50
John	145	14000	80
Jennifer	200	4400	10
Michael	201	13000	20
Susan	203	6500	40
Hermann	204	10000	70
Shelley	205	12000	110

Example:

```
SQL>  SELECT Ename, Salary, ROWNUM RANK
          FROM (SELECT   * FROM Employees
                      ORDER BY Salary);
```

Output:

ENAME	SALARY	RANK
Saketh	2500	1
Sudha	3000	2
Jagan	3500	3
Madhu	4000	4

Materialized View

In DWH materialized views are very essential because in reporting side if we do aggregate calculations as per the business requirement report performance would be de graded. So to improve report performance rather than doing report calculations and joins at reporting side if we put same logic in the MV then we can directly select the data from MV without any joins and aggregations. We can also schedule MV (Materialize View).

Example:

```
SQL> Create MATERIALIZED  HWMD_Mth_All_Metrics_Curr_Mv
     REFRESH COMPLETE
     START WITH Sysdate
     NEXT TRUNC(Sysdate+1)+  4/24
     WITH Primary Key
     AS
     SELECT * FROM HWMD_MTH_ALL_METRICS_CURR_VW;
```

Another Method to refresh MV:

```
SQL> DBMS_MVIEW.REFRESH('MV_COMPLEX', 'C');
```

Or

We can use Informatica mapping to refresh Materialized views.

What is the difference between view and materialized view?

VIEW	MATERIALIZED VIEW
A view has a logical existence. It does not contain data.	A materialized view has a physical existence.
We cannot perform DML operation on complex view.	We can perform DML operation on materialized view.
When we do select * from view it will fetch the data from base table.	When we do select * from materialized view it will fetch the data from materialized view.
In view we cannot schedule to refresh.	In materialized view we can schedule to refresh.
	We can keep aggregated data into materialized view. Materialized view mostly used for reporting purpose.

Normalization:

- Some Oracle databases were modeled according to the rules of normalization that were intended to eliminate redundancy.
- Obviously, the rules of normalization are required to understand your relationships and functional dependencies

First Normal Form:

A row is in first normal form (1NF) if all underlying domains contain atomic values only.

- Eliminate duplicative columns from the same table.

Second Normal Form:

An entity is in Second Normal Form (2NF) when it meets the requirement of being in First Normal Form (1NF) and additionally:

- Does not have a composite primary key. Meaning that the primary key can not be subdivided into separate logical entities.
- All the non-key columns are functionally dependent on the entire primary key.
- A row is in second normal form if, and only if, it is in first normal form and every non-key attribute is fully dependent on the key.

Third Normal Form:

An entity is in Third Normal Form (3NF) when it meets the requirement of Second Normal Form (2NF) and additionally:

- Functional dependencies on non-key fields are eliminated by putting them in a separate table. At this level, all non-key fields are dependent on the primary key.
- A row is in third normal form if and only if it is in second normal form and if attributes that do not contribute to a description of the primary key are move into a separate table. An example is creating look-up tables.

Boyce - Codd Normal Form:

Boyce Codd Normal Form (BCNF) is a further refinement of 3NF. In his later writings Codd refers to BCNF as 3NF. A row is in Boyce Codd normal form if, and only if, every determinant is a candidate key. Most entities in 3NF are already in BCNF.

Fourth Normal Form:

An entity is in Fourth Normal Form (4NF) when it meets the requirement of being in Third Normal Form (3NF) and additionally:

- Should not have any multiple sets of multi – valued dependencies. In other words, 4NF states that no entity can have more than a single one-to-many relationship.

Sequence: is an object in an oracle used to generate oracle sequence numbers to a column.

For example, suppose you had inserted 110 records last years and this year you want to add 50 more records to generate automatic sequence numbers we use sequence.

- Sequence can be used for any column of a table

```
SQL>    Create or replace sequence seq_name
        Start with 100 (end with 10000)
        Increment by 1
```

- If you mention no cycle, after meeting end value it throws error that your sequence may not return value as you reached maximum value.

  It will start with 1 and no end

```
SQL> INSERT INTO Emp VALUES ('p',seq.next,9000);
```

When you call for 1st time, *next = starting values*.

Now, next will send starting value (100) to current value current=100.

2nd time when you call next,

Next = current + increment----------the process continues...

Indexes

We can create indexes explicitly to speed up SQL statement execution on a table. The index points directly to the location of the rows containing the value.

Why indexes?

Indexes are most useful on larger tables, on columns that are likely to appear in where clauses as simple equality.

Types

- Unique index
- Non-unique index
- ***B - Tree*** index
- ***Bitmap*** index
- Composite index
- Function-based index
- Cluster index

Unique Index

Unique indexes guarantee that no two rows of a table have duplicate values in the columns that define the index. Unique index is automatically created when primary key or unique constraint is created.

Example:

```
SQL> CREATE UNIQUE INDEX Stud_Ind on Student(Sno);
```

Non – Unique Index

Non-Unique indexes do not impose the above restriction on the column values.

Example:

```
SQL> CREATE INDEX Stud_Ind ON Student(Sno);
```

***B - Tree* Index or Ascending Index**

The default type of index used in an oracle database is the B - Tree index. A B - Tree

index is designed to provide both rapid access to individual rows and quick access to groups of rows within a range. The B - Tree index does this by performing a succession of value comparisons. Each comparison eliminates many of the rows.

Example:

```
SQL> CREATE INDEX Stud_Ind ON Student(Sno);
```

Bitmap Index

This can be used for low cardinality columns: that is columns in which the number of distinct values is small when compared to the number of the rows in the table.

Example:

```
SQL> CREATE BITMAP INDEX Stud_Ind ON Student(Sex);
```

Composite Index

A composite index also called a concatenated index is an index created on multiple columns of a table. Columns in a composite index can appear in any order and need not be adjacent columns of the table.

Example:

```
SQL> CREATE BITMAP INDEX Stud_Ind ON Student(Sno, Sname);
```

PARTITIONS

A single logical table can be split into a number of physically separate pieces based on ranges of key values. Each partition is work like a table.

A non-partitioned table cannot be partitioned later.

Types

- Range partitions
- List partitions
- Hash partitions
- Sub partitions

Advantages

- The advantage is when you partition the table correctly for the majority of your usage there can be significant performance benefit

DISADVANTAGES

- Partitioned tables cannot contain any columns with long or long raw datatypes, LOB types or object types.
- Additional licensing fee and administration to manage/split/add/merge partitions and so forth.

```
CREATE TABLE Sales_Range
(       salesman_id  NUMBER(5),
        salesman_nameVARCHAR2(30),
        sales_amount  NUMBER(10),
        sales_date    DATE
)PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS
THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS
THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS
THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION sales_apr2000 VALUES LESS
THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
);
```

Query Tuning Approach

What is your tuning approach if SQL query taking long time? Or how do u tune SQL query?

- If query taking long time then we need to run the query in Explain Plan it will give us execution plan of the query like whether the query is using the relevant indexes on the joining columns or not .
- If joining columns doesn't have index then it will do the full table scan if it is full table scan the cost will be more then we have to create the indexes on the joining columns and will run the query it should give better performance .
- And also needs to analyze the tables if analyzation happened long back. It may causes performance.
- The ANALYZE statement can be used to gather statistics for a specific table below is the syntax ANALYZE TABLE employees COMPUTE STATISTICS;
- If still has performance issue then will use HINTS, hint is nothing but a clue. We can use hints in select statement like Use Hint to force using index.

  ```
  SELECT /*+INDEX (TABLE_NAME INDEX_NAME) */ COL1,COL2 FROM TABLE_NAME
  ```

- **ALL_ROWS :** `SELECT /*+ ALL_ROWS */`
  One of the hints that 'invokes' the Cost based optimizer
  ALL_ROWS is usually used for batch processing or data warehousing systems.

- **FIRST_ROWS :** `(/*+ FIRST_ROWS */)`
  One of the hints that 'invokes' the Cost based optimizer
  FIRST_ROWS is usually used for OLTP systems.

- **CHOOSE** :
  One of the hints that 'invokes' the Cost based optimizer
  This hint lets the server choose (between ALL_ROWS and FIRST_ROWS, based on statistics gathered.

- **HASH**
  Hashes one table (full scan) and creates a hash index for that table. Then hashes other table and uses hash index to find corresponding records. Therefore not suitable for < or > join conditions.

  ```
  /*+ use_hash */
  Select  ( /*+ hash  */ ) empno from Emp
  ```

Hints are most useful to optimize the query performance.

Why Hints Required?

It is a perfect valid question to ask why hints should be used. Oracle comes with an optimizer that promises to optimize a query's execution plan. When this optimizer is really doing a good job, no hints should be required at all.

Sometimes, however, the characteristics of the data in the database are changing rapidly, so that the optimizer (or more accuratly, its statistics) are out of date. In this case, a hint could help.

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.

DIFFERENCES

What is the Difference between Delete, Truncate and Drop?

Delete

- The DELETE command is used to remove rows from a table.
- A WHERE clause can be used to only remove some rows.
- If no WHERE condition is specified, all rows will be removed.
- After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it.

Truncate

- TRUNCATE removes all rows from a table.
- The operation cannot be rolled back, as such, TRUNCATE is faster and doesn't use as much undo space as a DELETE.

Drop

- The DROP command removes a table from the database.
- All the tables' rows, indexes and privileges will also be removed. The operation cannot be rolled back.

Order of where and having:

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

- The WHERE clause cannot be used to restrict groups.
- You use the HAVING clause to restrict groups.

Rowid & Rownum

Row_id	Row_num
Rowid is an oracle internal id that is allocated every time a new record is inserted in a table. This ID is unique and cannot be changed by the user.	Row-num is a row number returned by a select statement.
Rowid is permanent.	Row-num is temporary.
Rowid is a globally unique identifier for a row in a database. It is created at the time the row is inserted into the table, and destroyed when it is removed from a table.	The row-num pseudo column returns a number indicating the order in which oracle selects the row from a table or set of joined rows.

Where & Having

Where clause	Having clause
Both where and having clause can be used to filter the data.	
Where as in where clause it is not mandatory.	But having clause we need to use it with the group by.
Where clause applies to the individual rows.	Where as having clause is used to test some condition on the group rather than on individual rows.
Where clause is used to restrict rows.	But having clause is used to restrict groups.
Restrict normal query by where	Restrict group by function by having
In where clause every record is filtered based on where.	In having clause it is with aggregate records (group by functions).

Sub Query & Co-Related Sub Query

Sub – Query	Correlated Sub – Query
A Sub – Query is executed once for the parent Query	Whereas Correlated Sub – Query is executed once for each row of the parent query.
Example: SQL> SELECT * FROM Emp WHERE Deptno IN (SELECT Deptno FROM Dept);	**Example:** SQL> SELECT a.* FROM Emp e WHERE Sal >= (SELECT AVG(Sal) FROM Emp a WHERE a.Deptno = e.Deptno GROUP BY a.Deptno);

Store Procedure & Function

Stored Procedure	Functions
Stored procedure may or may not return values.	Function should return at least one output parameter. Can return more than one parameter using OUT argument.
Stored procedure can be used to solve the business logic.	Function can be used to calculations
Stored procedure is a pre-compiled statement.	But function is not a pre-compiled statement.
Stored procedure accepts more than one argument.	Whereas function does not accept arguments.
Stored procedures are mainly used to process the tasks.	Functions are mainly used to compute values
Cannot be invoked from SQL statements. E.g. SELECT	Can be invoked form SQL statements e.g. SELECT
Can affect the state of database using commit.	Cannot affect the state of database.
Stored as a pseudo-code in database i.e. compiled form.	Parsed and compiled at runtime.

Trigger & Procedure

Triggers	Stored Procedures
In trigger no need to execute manually. Triggers will be fired automatically.	Where as in procedure we need to execute manually.
Triggers that run implicitly when an INSERT, UPDATE, or DELETE statement is issued against the associated table.	Stored procedure normally used for performing tasks
Trigger normally used for tracing and auditing logs.	Stored procedures should be called explicitly by the user in order to execute
Trigger should be called implicitly based on the events defined in the table.	Stored Procedure can run independently
Trigger should be part of any DML events on the table.	Stored procedure can be executed from the Trigger
Trigger cannot be executed from the Stored procedures.	Stored Procedures can have parameters.

Trigger cannot have any parameters.	Stored procedures are compiled collection of programs or SQL statements in the database. Using stored procedure we can access and modify data present in many tables. Also a stored procedure is not associated with any particular database object.
triggers are event-driven special procedures which are attached to a specific database object say a table	Stored procedures are not automatically run and they have to be called explicitly by the user.
Triggers get executed when the particular event associated with the event gets fired	

Store Procedure

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name that's stored in the database in compiled form so that it can be shared by a number of programs.

Pre – Requisites:

Before creating a procedure, the user SYS must run a SQL script commonly called DBMSSTDX.SQL. The exact name and location of this script depend on your operating system.

To create a procedure in your own schema, you must have the CREATE PROCEDURE system privilege. To create a procedure in another user's schema, you must have the CREATE ANY PROCEDURE system privilege. To replace a procedure in another schema, you must have the ALTER ANY PROCEDURE system privilege.

How to create and execute store proc?

Example :

```
SQL> CREATE OR REPLACE
     Procedure Add_Sp(
         A In Number,
         b IN NUMBER,
         c OUT NUMBER)
     AS
     Begin
       c := (a + b);
       DBMS_OUTPUT.put_line (c);
     END;
     /
SQL> VAR C Number;

SQL> EXEC add_sp(10,15,:C);

SQL> PRINT C;
```

Example :

```
SQL> CREATE OR REPLACE
     PROCEDURE concat_sp(
         a IN VARCHAR2,
         b IN VARCHAR2,
         c OUT VARCHAR2)
     AS
     Begin
       C := concat(a,b);
       dbms_output.put_line(c);
     END;
     /
```

```
SQL> VAR C VARCHAR2(20);

SQL> EXEC concat_sp('ABC','XYZ', :C);

SQL> PRINT C;
```

Packages:

- Packages provide a method of encapsulating related procedures, functions, and associated cursors and variables together as a unit in the database.
- Package that contains several procedures and functions that process related to same transactions.
- A package is a group of related procedures and functions, together with the cursors and variables they use,
- Packages provide a method of encapsulating related procedures, functions, and associated cursors and variables together as a unit in the database.

Triggers:

- Oracle lets you define procedures called triggers that run implicitly when an INSERT, UPDATE, or DELETE statement is issued against the associated table
- Triggers are similar to stored procedures. A trigger stored in the database can include SQL and PL/SQL

Types of Triggers

This section describes the different types of triggers:

- Row Triggers and Statement Triggers
- BEFORE and AFTER Triggers

A **Row Trigger** is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.

A **statement trigger** is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

BEFORE and AFTER Triggers

When defining a trigger, you can specify the trigger timing--whether the trigger action is to be run before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

BEFORE and AFTER triggers fired by DML statements can be defined only on tables, not on views.

```
SQL> CREATE OR REPLACE TRIGGER emp_aur
        AFTER INSERT
        ON emp
     DECLARE
     BEGIN
        DELETE FROM x;
        COMMIT;
     END;
```

Data files Overview:

A table space in an Oracle database consists of one or more physical **data files**. A data file can be associated with only one table space and only one database.

Table Space:

- Oracle stores data logically in table spaces and physically in data files associated with the corresponding table space.
- A database is divided into one or more logical storage units called table spaces. Table spaces are divided into logical units of storage called segments.

Control File:

A control file contains information about the associated database that is required for access by an instance, both at startup and during normal operation. Control file information can be modified only by Oracle; no database administrator or user can edit a control file.

Important Queries

1. Get duplicate rows from the table:

```
SQL> SELECT   empno, COUNT (*)
     FROM emp
     GROUP BY empno
     HAVING COUNT (*) > 1;
```

2. Remove duplicates in the table:

```
SQL> DELETE FROM emp
       WHERE ROWID NOT IN (SELECT   MAX (ROWID) FROM emp
                           GROUP BY empno);
```

3. **Below query transpose columns into rows.**

Name	No	Add1	Add2
abc	100	Hyd	bang
xyz	200	Mysore	pune

```
SQL> SELECT NAME, NO, add1 FROM a
     UNION
     SELECT NAME, NO, add2 FROM a;
```

4. **Below query transpose rows into columns.**

```
SQL> SELECT   emp_id, MAX (DECODE (row_id, 0, address)) AS address1,
         MAX (DECODE (row_id, 1, address)) AS address2,
         MAX (DECODE (row_id, 2, address)) AS address3
    FROM (SELECT   emp_id, address, MOD (ROWNUM, 3) row_id FROM temp
          ORDER BY emp_id)
   GROUP BY emp_id
```

Other query:

```
SQL> SELECT   emp_id, MAX (DECODE (rank_id, 1, address)) AS add1,
         MAX (DECODE (rank_id, 2, address)) AS add2,
         MAX (DECODE (rank_id, 3, address)) AS add3
    FROM (SELECT emp_id, address,
                 RANK () OVER (PARTITION BY emp_id ORDER BY emp_id,
                  address) rank_id
            FROM temp)
     GROUP BY emp_id
```

5. **Rank query:**

```
SQL> SELECT empno, ename, sal, r
          FROM (SELECT empno, ename, sal, RANK ()
                OVER (ORDER BY sal DESC) r FROM emp);
```

6. **Dense rank query:**

The DENSE_RANK function works acts like the RANK function except that it assigns consecutive ranks:

```
SQL> SELECT empno,  ename, Sal, FROM
     (SELECT empno, ename, sal, DENSE_RANK ()
           OVER (ORDER BY sal DESC) r FROM emp);
```

7. **Top 5 salaries by using rank:**

```
SQL> SELECT empno, ename, sal, r
     FROM (SELECT empno, ename, sal, DENSE_RANK () OVER
               (ORDER BY sal DESC) r FROM emp)
          WHERE r <= 5;

      (OR)

SQL> SELECT * FROM
        (SELECT   * FROM emp ORDER BY sal DESC)
    WHERE ROWNUM <= 5;
```

8. 2nd highest Sal:

```
SQL> SELECT empno, ename, sal, r
       FROM (SELECT empno, ename, sal, DENSE_RANK () OVER
                  (ORDER BY sal DESC) r FROM emp)
      WHERE r = 2;
```

9. Top Sal:

```
SQL> SELECT * FROM emp
         WHERE sal = (SELECT MAX (sal) FROM emp);
```

10. How to display alternative rows in a table?

```
SQL> SELECT * FROM emp
         WHERE (ROWID, 0) IN
            (SELECT ROWID, MOD (ROWNUM, 2) FROM emp);
```

11. Hierarchical queries

Starting at the root, walk from the top down, and eliminate employee Higgins in the result, but process the child rows.

```
SQL> SELECT department_id, employee_id, last_name, job_id, salary
     FROM employees
     WHERE last_name ! = 'Higgins'
     START WITH manager_id IS NULL
          CONNECT BY PRIOR employee_id = manager_id;
```

12. greatest( ) function

The ***greatest* function** returns the greatest value in a list of expressions.

Syntax:

GREATEST (expr1, expr2, …, exprn)

Example:

```
SQL> SELECT NAME, sub1, sub2, sub3, sub4,
       GREATEST (sub1, sub2, sub3, sub4) AS greatest_value
       FROM student;
```

Student

Name	Sub1	Sub2	Sub3	Sub4	Greatest Value
Arun	14	17	19	13	**19**
Tarun	15	16	10	8	**16**
Varun	16	18	14	2	**18**

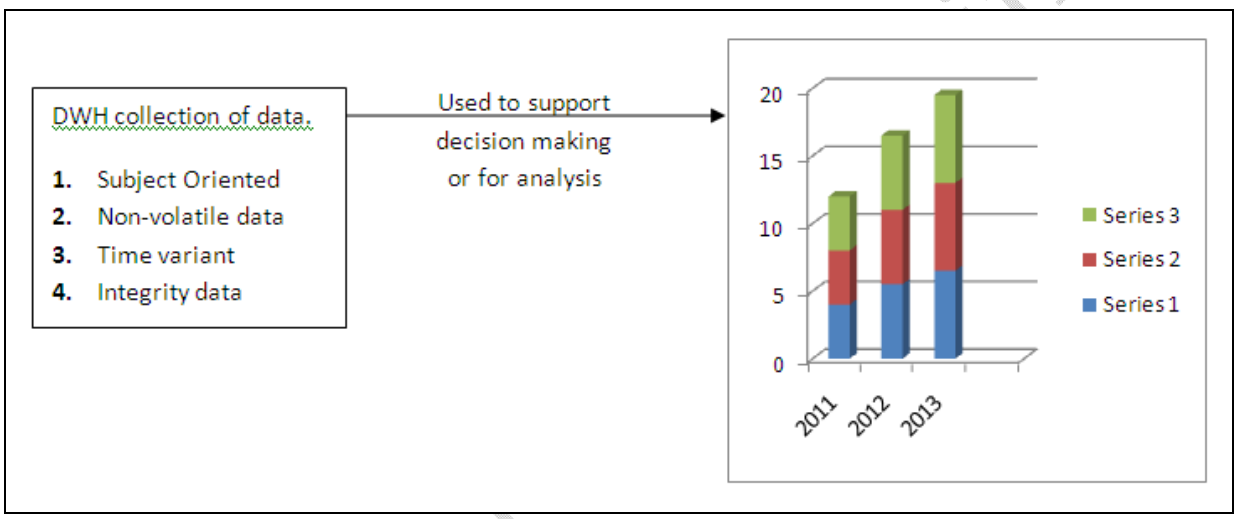
DWH CONCEPTS

What is BI?

Business Intelligence refers to a set of methods and techniques that are used by organizations for tactical and strategic decision making. It leverages methods and technologies that focus on counts, statistics and business objectives to improve business performance.

The objective of Business Intelligence is to better understand customers and improve customer service, make the supply and distribution chain more efficient, and to identify and address business problems and opportunities quickly.

Warehouse is used for high level data analysis purpose. It is used for predictions, time series analysis, financial Analysis, what -if simulations etc. Basically it is used for better decision making.



OLTP:

a. Online Transaction Processing will store only current data.
b. Current may be daily, weekly, monthly, quarterly, yearly but not more than 1 year.
c. Based on time daily, weekly, monthly, quarterly, yearly OLTP transfers data to DWH.
d. Oracle, UNIX, SAP, Flat files stores only current data they won't store previous data..
e. OLTP system maintains only current data because performance degrades by storing historical data into OLTP systems hence we store historical data in DWH.
f. The business people connect every day or every week OLTP (current data) to DWH through ETL Informatica.

OLTP	DWH / DSS / OLAP
1. OLTP contains only current data	**1.** DWH contains historical data (current data + previous data)
2. It is not used for reporting purpose.	**2.** It is purely designed for reporting purpose.
3. OLTP is a volatile system	**3.** OLAP is a Non-volatile system
4. It is not integrated system	**4.** It is a integrated system.
5. Not supported for analysis and decision	**5.** Supports for analysis and decision
6. It is not specific to subject area	**6.** Specific to subject area.

7. Data stored in oltp in a relational model (parent-child relationship). **8.** Normalized structure(large tables into small tables)	**7.** Data stored in DWH in a dimensional model(facts & dimension tables) **8.** Demoralized structure. The data once loaded in DWH will remain same for next 100years

What is a Data Warehouse?

Data warehouse is the de-normalized structure of database, which stores historical data in summary level format. It is specifically meant for heavy duty querying and analysis.

Data Warehouse is a "*Subject-Oriented, Integrated, Time-Variant Nonvolatile collection of data in support of decision making*".

In terms of design *data warehouse* and *data mart* are almost the same.

In general a Data Warehouse is used on an enterprise level and a Data Marts is used on a business division/department level.

Subject Oriented:

Data that gives information about a particular subject instead of about a company's ongoing operations.

Integrated:

Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

Time – variant:

All data in the data warehouse is identified with a particular time period.

Non – volatile:

Data is stable in a data warehouse. More data is added but data is never removed.

What is a Data Mart?

Data mart is usually sponsored at the department level and developed with a specific details or subject in mind, a Data Mart is a subset of data warehouse with a focused objective.

What is the difference between a Data Warehouse and a Data Mart?

- In terms of design data warehouse and data mart are almost the same.
- In general a Data Warehouse is used on an enterprise level and a Data Marts is used on a business division/department level.
- A data mart only contains data specific to a particular subject areas.

Difference between Data Mart and Data Warehouse

Data Mart	Data Warehouse
Data mart is usually sponsored at the department level and developed with a specific issue or subject in mind, a data mart is a data warehouse with a focused objective.	Data warehouse is a "Subject-Oriented, Integrated, Time-Variant, Nonvolatile collection of data in support of decision making".
A data mart is used on a business division/ department level.	A data warehouse is used on an enterprise level
A Data Mart is a subset of data from a Data Warehouse. Data Marts are built for specific user groups.	A Data Warehouse is simply an integrated consolidation of data from a variety of sources that is specially designed to support strategic and tactical decision making.
By providing decision makers with only a subset of data from the Data Warehouse, Privacy, Performance and Clarity Objectives can be attained.	The main objective of Data Warehouse is to provide an integrated environment and coherent picture of the business at a point in time.

What is a Schema?

- Graphical Representation of the data structure.
- First Phase in implementation of Universe.

What are the most important features of a data warehouse?

- Drill Down, Drill Across, Graphs, PI charts, Dashboards and Time Handling.
- To be able to Drill Down/Drill Across is the most basic requirement of an end user in a data warehouse. Drilling down most directly addresses the natural end-user need to see more detail in result. Drill down should be as generic as possible. Because there is absolutely no good way to predict users drill-down path.

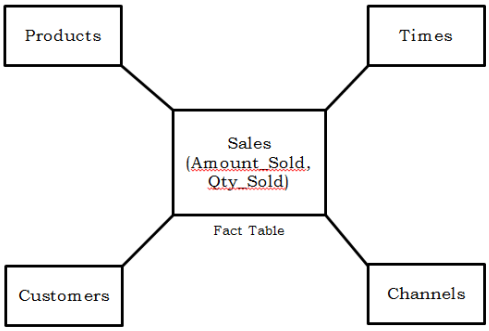
What does it mean by grain of the star schema?

- In Data warehousing grain refers to the level of detail available in a given fact table as well as to the level of detail provided by a star schema.
- It is usually given as the number of records per key within the table. In general, the grain of the fact table is the grain of the star schema.

What is a star schema?

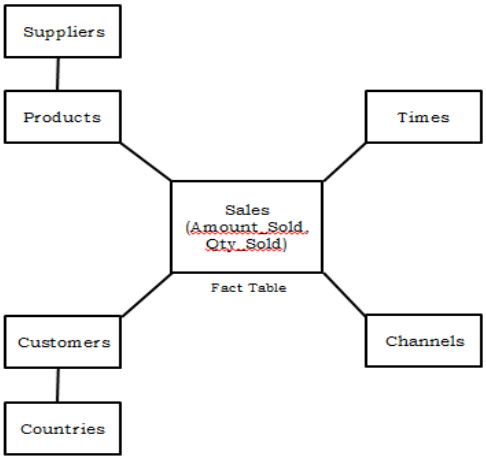
Star schema is a data warehouse schema where there is only one "fact table" and many denormalized dimension tables.

Fact table contains primary keys from all the dimension tables and other numeric columns columns of additive, numeric facts.



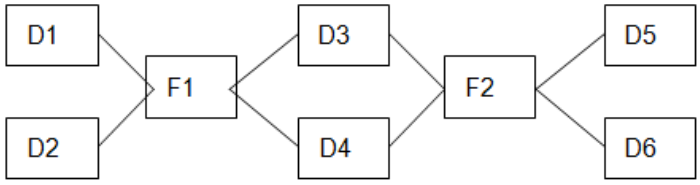
What is a snowflake schema?

- Unlike Star-Schema, Snowflake schema contain normalized dimension tables in a tree like structure with many nesting levels.
- Snowflake schema is easier to maintain but queries require more joins.



What is Galaxy Schema?

The Schema which contain many fact tables surrounded by the common dimensions. It a combination of multiple Data Marts.



What is the difference between Snow Flake and Star Schema

Star Schema	Snow Flake Schema
The star schema is the simplest data warehouse scheme.	Snowflake schema is a more complex data warehouse model than a star schema.
In star schema each of the dimensions is represented in a single table .It should not have any hierarchies between dims.	In snow flake schema at least one hierarchy should exists between dimension tables.
It contains a fact table surrounded by dimension tables. If the dimensions are	It contains a fact table surrounded by dimension tables. If a dimension is

de-normalized, we say it is a star schema design.	normalized, we say it is a snow flaked design.
In star schema only one join establishes the relationship between the fact table and any one of the dimension tables.	In snow flake schema since there is relationship between the dimensions tables it has to do many joins to fetch the data.
A star schema optimizes the performance by keeping queries simple and providing fast response time. All the information about the each level is stored in one row.	Snowflake schemas normalize dimensions to eliminated redundancy. The result is more complex queries and reduced query performance.
It is called a star schema because the diagram resembles a star.	It is called a snowflake schema because the diagram resembles a snowflake.

What is Fact and Dimension?

- A "fact" is a numeric value that a business wishes to count or sum. A "dimension" is essentially an entry point for getting at the facts. Dimensions are things of interest to the business.

- A set of level properties that describe a specific aspect of a business, used for analyzing the factual measures.

What is Fact Table?

A Fact Table in a dimensional model consists of one or more numeric facts of importance to a business. Examples of facts are as follows:

- the number of products sold
- the value of products sold
- the number of products produced
- the number of service calls received

What is Factless Fact Table?

A fact table that contains only primary keys from the dimension tables, and that do not contain any measures that type of fact table is called fact less fact table.

Factless fact table captures the many-to-many relationships between dimensions, but contains no numeric or textual facts. They are often used to record events or coverage information.

Common examples of Factless fact tables include:

- Identifying product promotion events (to determine promoted products that didn't sell)
- Tracking student attendance or registration events
- Tracking insurance-related accident events

Types of facts?

There are three types of facts:

- **Additive**: Additive facts are facts that can be summed up through all of the dimensions in the fact table.

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

- **Non-Additive**: Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table.

What is Granularity?

Principle: create fact tables with the most granular data possible to support analysis of the business process.

In Data warehousing grain refers to the level of detail available in a given fact table as well as to the level of detail provided by a star schema.

It is usually given as the number of records per key within the table. In general, the grain of the fact table is the grain of the star schema.

Facts: Facts must be consistent with the grain. All facts are at a uniform grain.

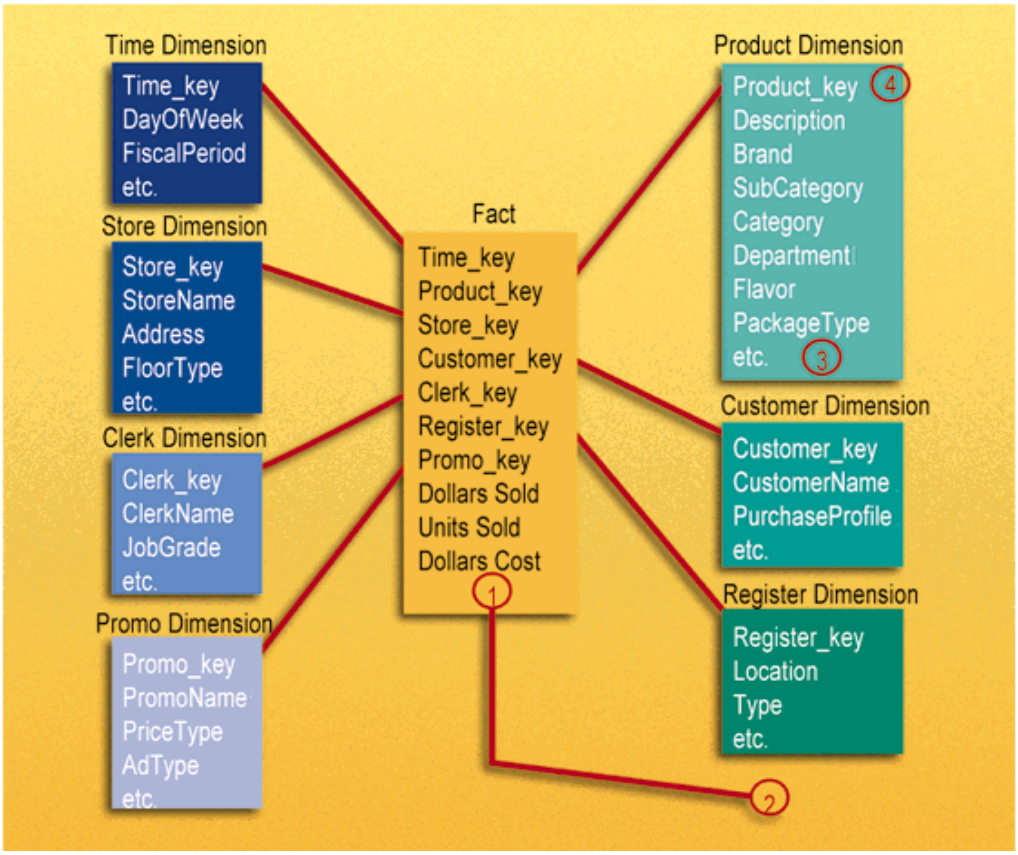
- Watch for facts of mixed granularity
- Total sales for day & montly total

Dimensions:

Each dimension associated with fact table must take on a single value for each fact row.

- Each dimension attribute must take on one value.
- Outriggers are the exception, not the rule.

Dimensional Model



What is Slowly Changing Dimension?

Slowly changing dimensions refers to the change in dimensional attributes over time.

An example of slowly changing dimension is a Resource dimension where attributes of a particular employee change over time like their designation changes or dept changes etc.

What is Conformed Dimension?

Conformed Dimensions (CD): these dimensions are something that is built once in your model and can be reused multiple times with different fact tables. For example, consider a model containing multiple fact tables, representing different data marts. Now look for a dimension that is common to these facts tables. In this example let's consider that the product dimension is common and hence can be reused by creating short cuts and joining the different fact tables. Some of the examples are time dimension, customer dimensions, product dimension.

What is Junk Dimension?

A "junk" dimension is a collection of random transactional codes, flags and/or text attributes that are unrelated to any particular dimension. The junk dimension is simply a structure that provides a convenient place to store the junk attributes. A good example would be a trade fact in a company that brokers equity trades.

When you consolidate lots of small dimensions and instead of having 100s of small dimensions, that will have few records in them, cluttering your database with these mini 'identifier' tables, all records from all these small dimension tables are loaded into ONE dimension table and we call this dimension table Junk dimension table. (Since we are storing all the junk in this one table) For example: a company might have handful of manufacture plants, handful of order types, and so on, so forth, and we can consolidate them in one dimension table called junked dimension table. It's a dimension table which is used to keep junk attributes

What is De – Generated Dimension?

An item that is in the fact table but is stripped off of its description, because the description belongs in dimension table, is referred to as Degenerated Dimension. Since it looks like dimension, but is really in fact table and has been degenerated of its description, hence is called degenerated dimension.

Degenerated Dimension:

A dimension which is located in fact table known as Degenerated dimension

Dimensional Model:

A type of data modeling suited for data warehousing.

In a dimensional model, there are two types of tables:

- Dimensional tables and
- Fact tables.

Dimensional table records information on each dimension, and fact table records all the "fact", or measures.

Data modeling

There are three levels of data modeling. They are conceptual, logical, and physical. This section will explain the difference among the three, the order with which each one is created, and how to go from one level to the other.

Conceptual Data Model

Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.

At this level, the data modeler attempts to identify the highest-level relationships among the different entities.

Logical Data Model

Features of logical data model include:

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

At this level, the data modeler attempts to describe the data in as much detail as possible, without regard to how they will be physically implemented in the database.

In data warehousing, it is common for the conceptual data model and the logical data model to be combined into a single step (deliverable).

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

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

Physical Data Model

Features of physical data model include:

- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Demoralization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.

At this level, the data modeler will specify how the logical data model will be realized in the database schema.

The steps for physical data model design are as follows:

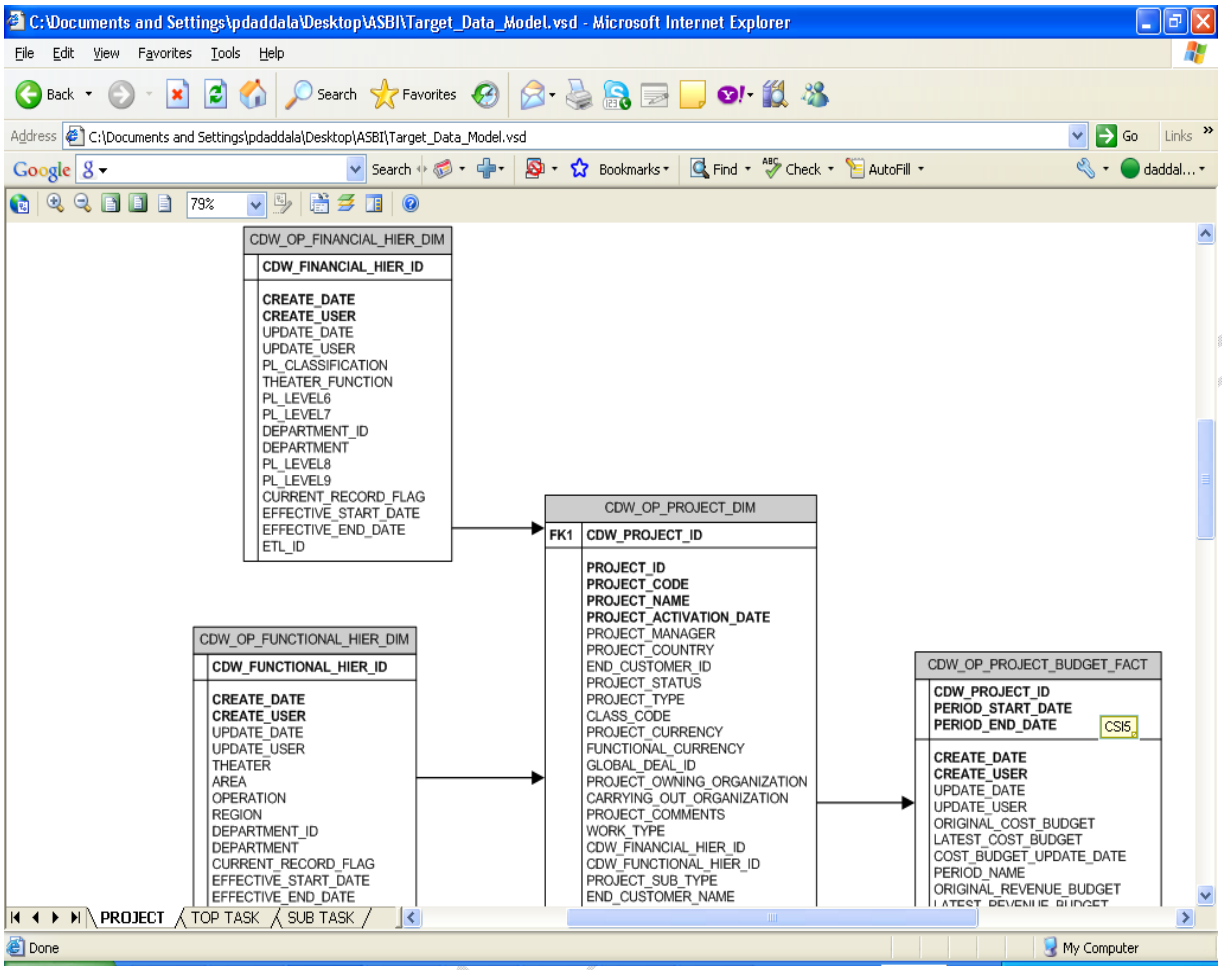
1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
   a. http://www.learndatamodeling.com/dm_standard.htm
   b. Modeling is an efficient and effective way to represent the organization's needs; It provides information in a graphical way to the members of an organization to understand and communicate the business rules and processes. Business Modeling and Data Modeling are the two important types of modeling.

The differences between a Logical Data Model and physical data model is shown below.

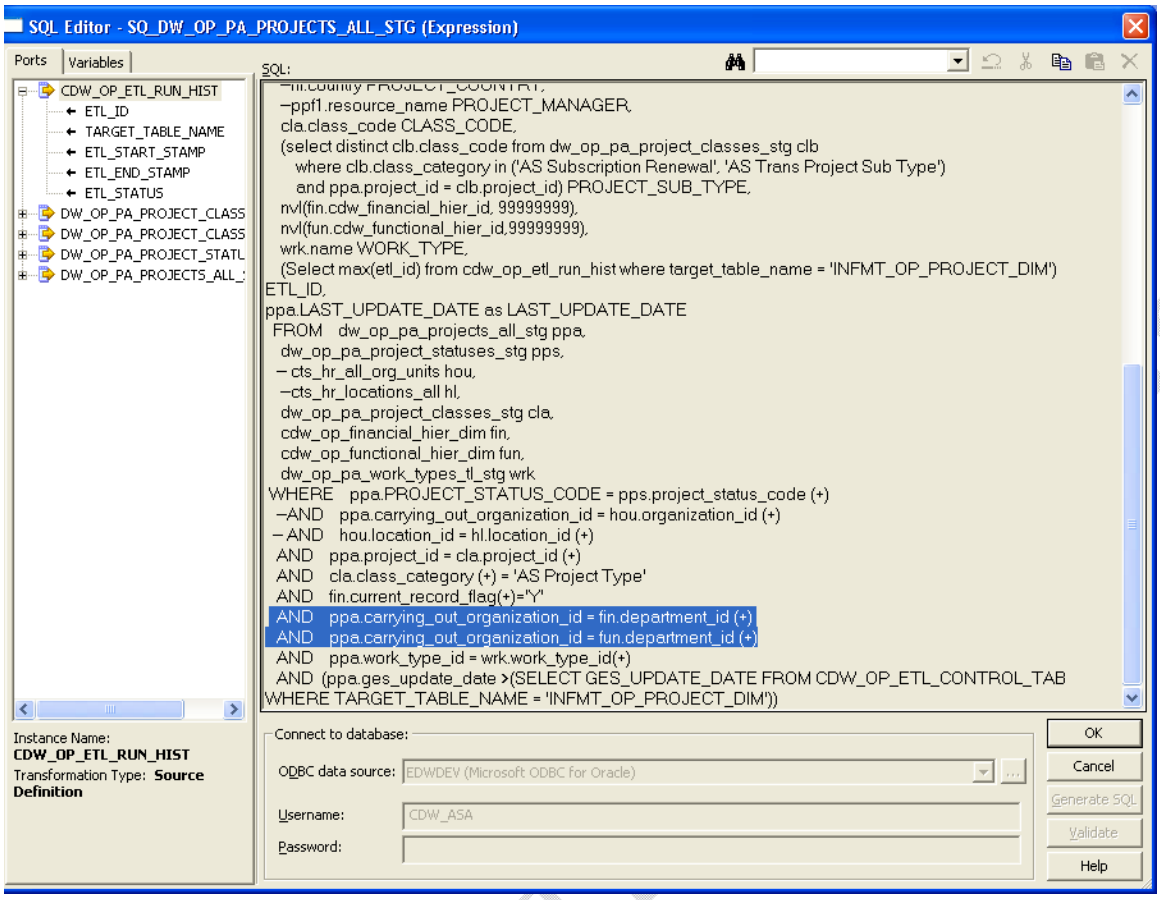
Logical vs Physical Data Modeling

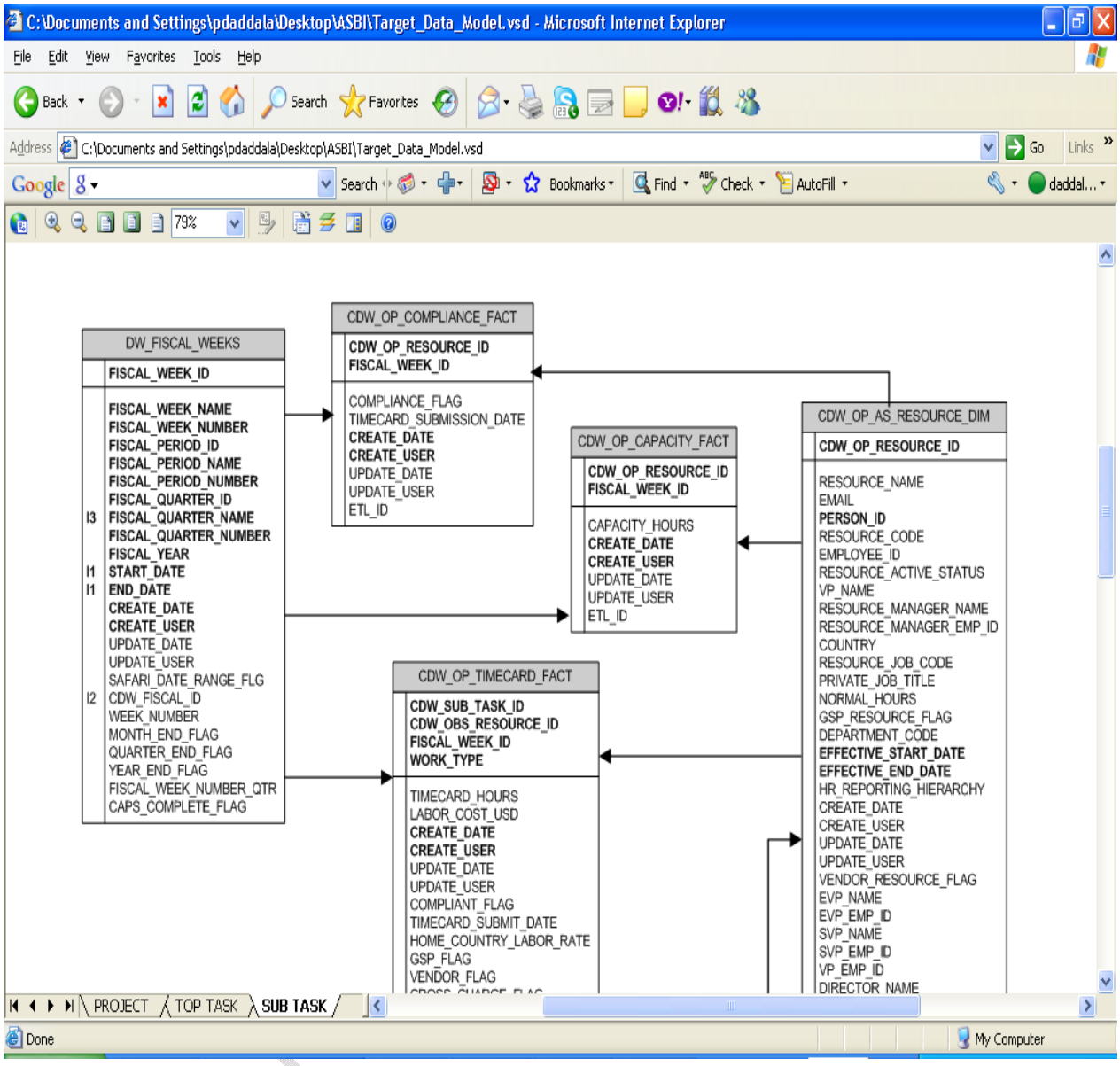
Logical Data Model	Physical Data Model
Represents business information and defines business rules	Represents the physical implementation of the model in a database.
Entity	Table
Attribute	Column
Primary Key	Primary Key Constraint
Alternate Key	Unique Constraint or Unique Index
Inversion Key Entry	Non Unique Index
Rule	Check Constraint, Default Value
Relationship	Foreign Key
Definition	Comment

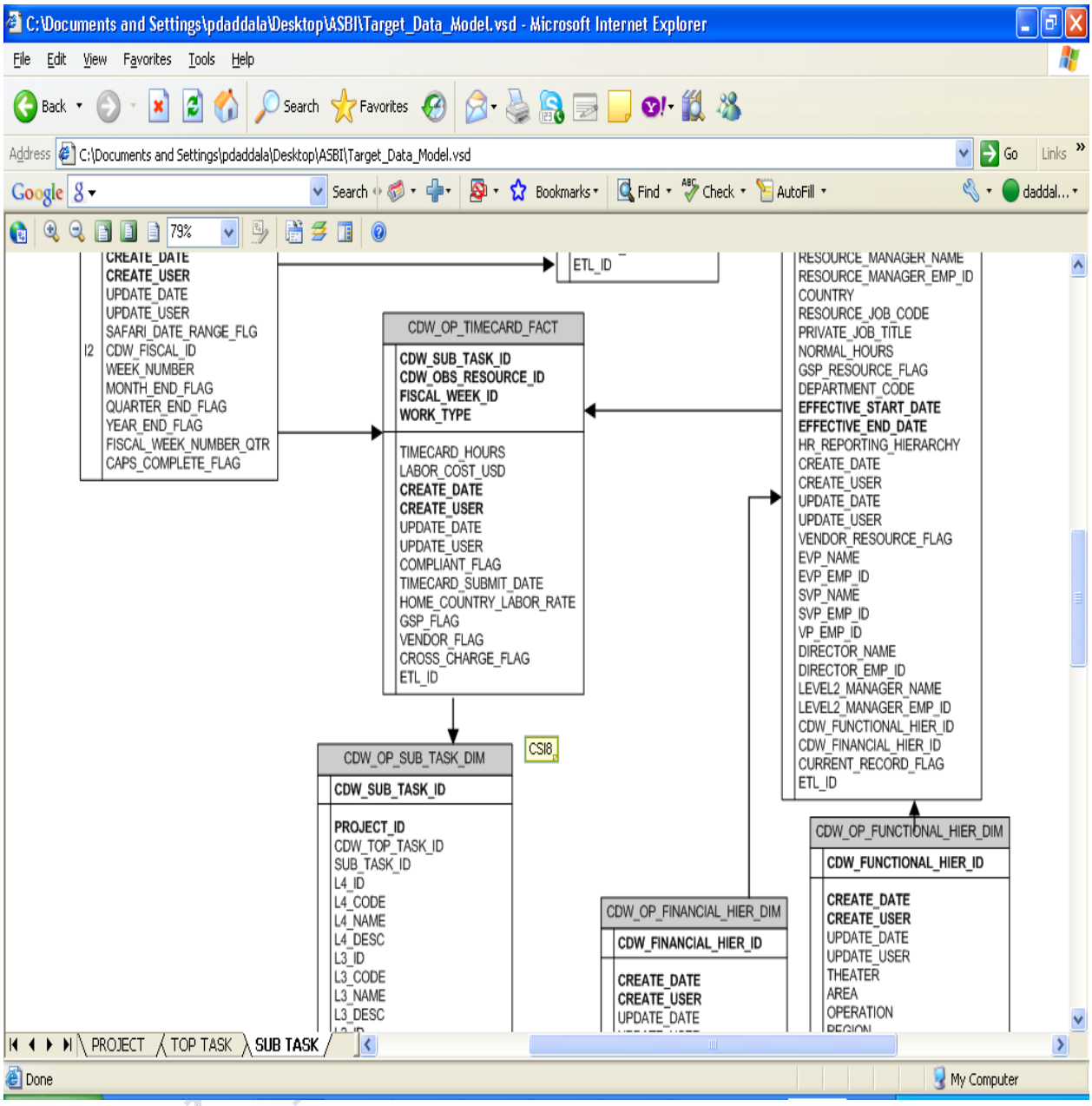
Below is the Simple Data Model



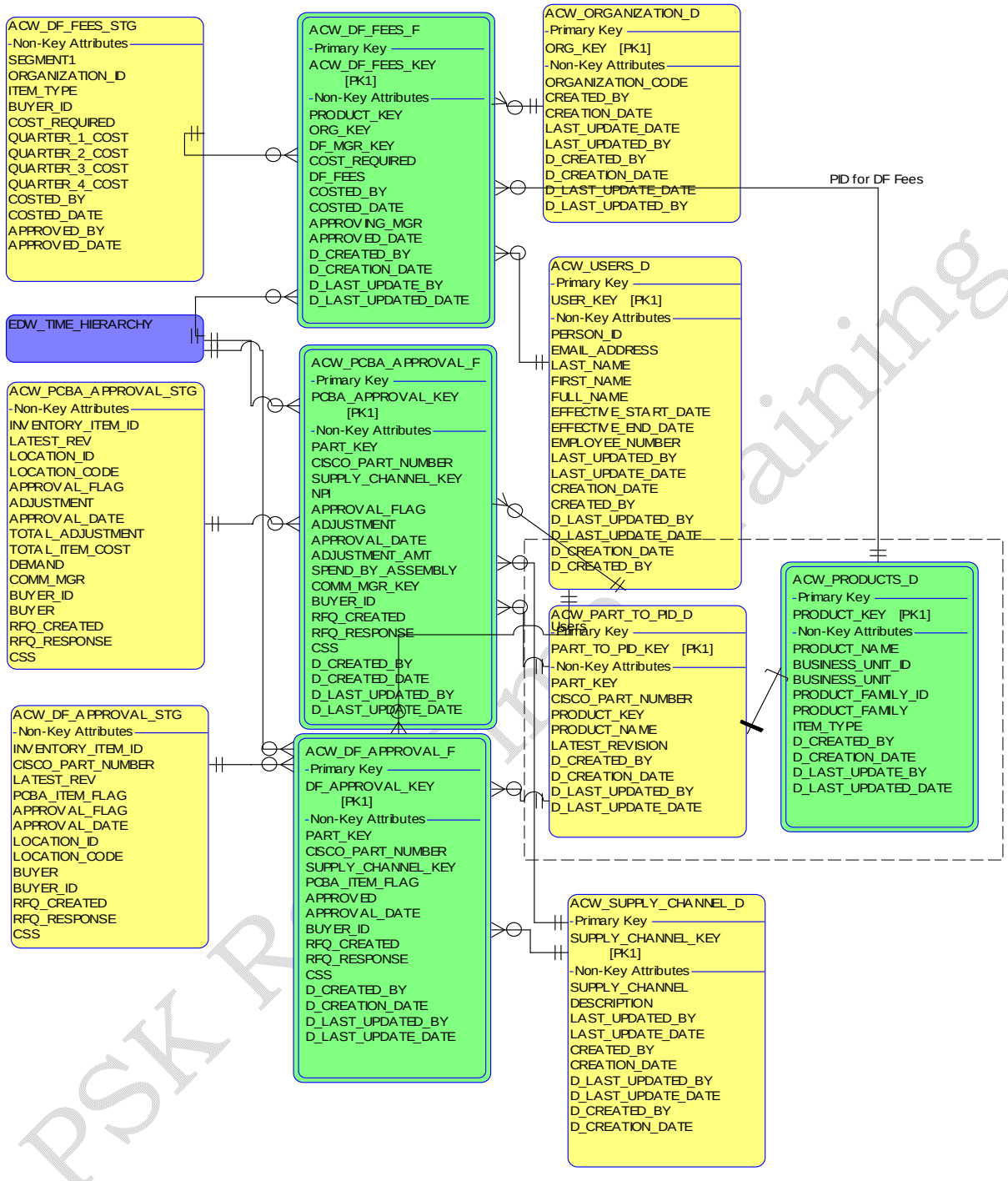
Below is the SQL Query for one of SourceQualifier (SQ) for the dimension table load



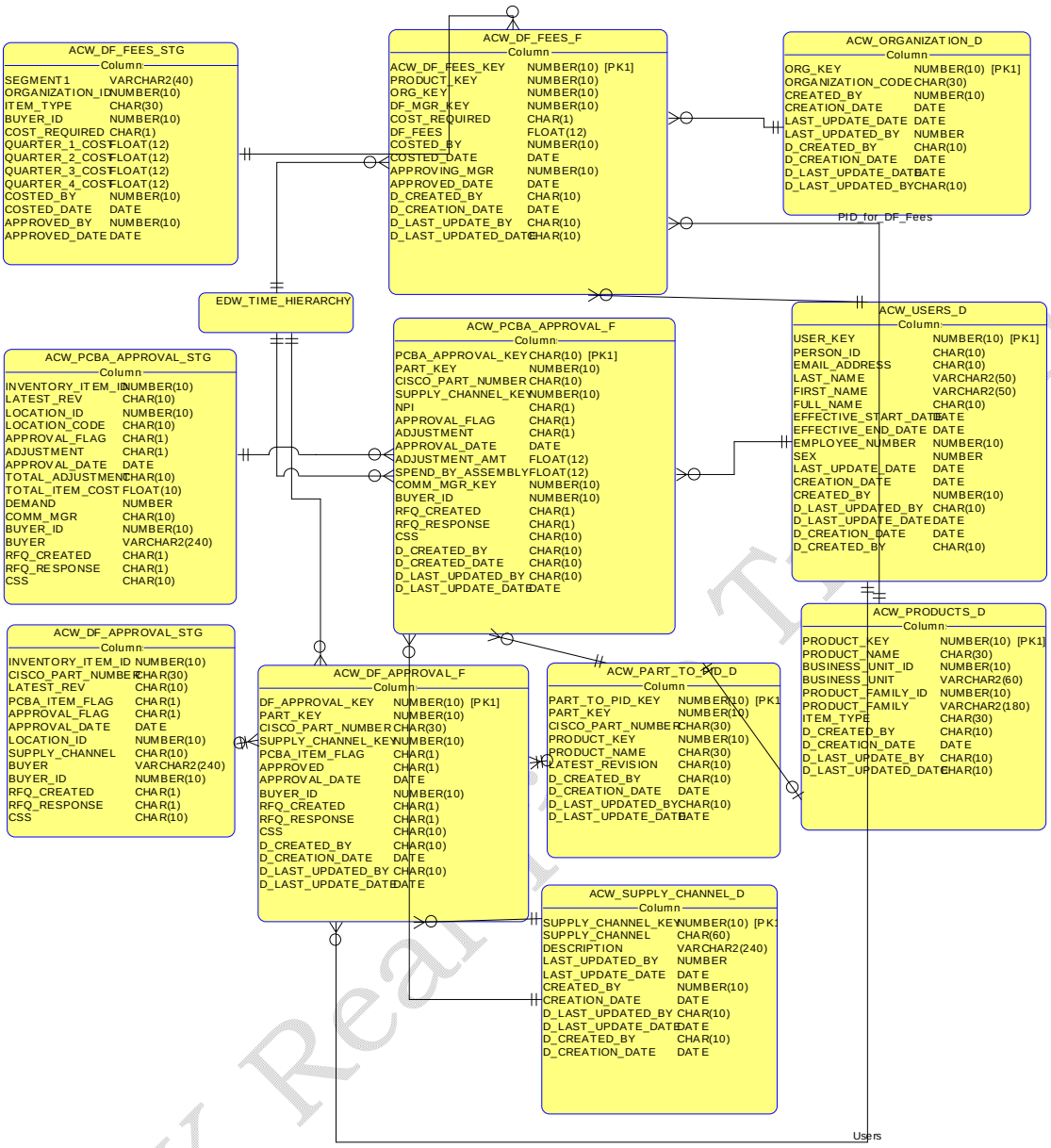

C:\Documents and Settings\pdaddala\Desktop\ASBI\Target_Data_Model.vsd - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Documents and Settings\pdaddala\Desktop\ASBI\Target_Data_Model.vsd
Go Links
Google Search Bookmarks Find Check AutoFill daddal...
79%
DW_FISCAL_WEEKS
FISCAL_WEEK_ID
FISCAL_WEEK_NAME
FISCAL_WEEK_NUMBER
FISCAL_PERIOD_ID
FISCAL_PERIOD_NAME
FISCAL_PERIOD_NUMBER
FISCAL_QUARTER_ID
I3 FISCAL_QUARTER_NAME
FISCAL_QUARTER_NUMBER
FISCAL_YEAR
I1 START_DATE
I1 END_DATE
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
SAFARI_DATE_RANGE_FLG
I2 CDW_FISCAL_ID
WEEK_NUMBER
MONTH_END_FLAG
QUARTER_END_FLAG
YEAR_END_FLAG
FISCAL_WEEK_NUMBER_QTR
CAPS_COMPLETE_FLAG
CDW_OP_COMPLIANCE_FACT
CDW_OP_RESOURCE_ID
FISCAL_WEEK_ID
COMPLIANCE_FLAG
TIMECARD_SUBMISSION_DATE
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
ETL_ID
CDW_OP_CAPACITY_FACT
CDW_OP_RESOURCE_ID
FISCAL_WEEK_ID
CAPACITY_HOURS
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
ETL_ID
CDW_OP_AS_RESOURCE_DIM
CDW_OP_RESOURCE_ID
RESOURCE_NAME
EMAIL
PERSON_ID
RESOURCE_CODE
EMPLOYEE_ID
RESOURCE_ACTIVE_STATUS
VP_NAME
RESOURCE_MANAGER_NAME
RESOURCE_MANAGER_EMP_ID
COUNTRY
RESOURCE_JOB_CODE
PRIVATE_JOB_TITLE
NORMAL_HOURS
GSP_RESOURCE_FLAG
DEPARTMENT_CODE
EFFECTIVE_START_DATE
EFFECTIVE_END_DATE
HR_REPORTING_HIERARCHY
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
VENDOR_RESOURCE_FLAG
EVP_NAME
EVP_EMP_ID
SVP_NAME
SVP_EMP_ID
VP_EMP_ID
DIRECTOR_NAME
CDW_OP_TIMECARD_FACT
CDW_SUB_TASK_ID
CDW_OBS_RESOURCE_ID
FISCAL_WEEK_ID
WORK_TYPE
TIMECARD_HOURS
LABOR_COST_USD
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
COMPLIANT_FLAG
TIMECARD_SUBMIT_DATE
HOME_COUNTRY_LABOR_RATE
GSP_FLAG
VENDOR_FLAG
PROJECT TOP TASK SUB TASK
Done
My Computer


C:\Documents and Settings\pdaddala\Desktop\ASBI\Target_Data_Model.vsd - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Documents and Settings\pdaddala\Desktop\ASBI\Target_Data_Model.vsd
Go Links
Google Search Bookmarks Find Check AutoFill
79%
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
SAFARI_DATE_RANGE_FLG
CDW_FISCAL_ID
WEEK_NUMBER
MONTH_END_FLAG
QUARTER_END_FLAG
YEAR_END_FLAG
FISCAL_WEEK_NUMBER_QTR
CAPS_COMPLETE_FLAG
ETL_ID
CDW_OP_TIMECARD_FACT
CDW_SUB_TASK_ID
CDW_OBS_RESOURCE_ID
FISCAL_WEEK_ID
WORK_TYPE
TIMECARD_HOURS
LABOR_COST_USD
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
COMPLIANT_FLAG
TIMECARD_SUBMIT_DATE
HOME_COUNTRY_LABOR_RATE
GSP_FLAG
VENDOR_FLAG
CROSS_CHARGE_FLAG
ETL_ID
RESOURCE_MANAGER_NAME
RESOURCE_MANAGER_EMP_ID
COUNTRY
RESOURCE_JOB_CODE
PRIVATE_JOB_TITLE
NORMAL_HOURS
GSP_RESOURCE_FLAG
DEPARTMENT_CODE
EFFECTIVE_START_DATE
EFFECTIVE_END_DATE
HR_REPORTING_HIERARCHY
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
VENDOR_RESOURCE_FLAG
EVP_NAME
EVP_EMP_ID
SVP_NAME
SVP_EMP_ID
VP_EMP_ID
DIRECTOR_NAME
DIRECTOR_EMP_ID
LEVEL2_MANAGER_NAME
LEVEL2_MANAGER_EMP_ID
CDW_FUNCTIONAL_HIER_ID
CDW_FINANCIAL_HIER_ID
CURRENT_RECORD_FLAG
ETL_ID
CDW_OP_SUB_TASK_DIM
CDW_SUB_TASK_ID
PROJECT_ID
CDW_TOP_TASK_ID
SUB_TASK_ID
L4_ID
L4_CODE
L4_NAME
L4_DESC
L3_ID
L3_CODE
L3_NAME
L3_DESC
CSI8
CDW_OP_FINANCIAL_HIER_DIM
CDW_FINANCIAL_HIER_ID
CREATE_DATE
CREATE_USER
UPDATE_DATE
CDW_OP_FUNCTIONAL_HIER_DIM
CDW_FUNCTIONAL_HIER_ID
CREATE_DATE
CREATE_USER
UPDATE_DATE
UPDATE_USER
THEATER
AREA
OPERATION
PROJECT TOP TASK SUB TASK
Done
My Computer

EDIII – Logical Design



EDII– Physical Design



Types of SCD Implementation:

SCD Type 1

In Type 1 Slowly Changing Dimension, the new information simply overwrites the original information. In other words, no history is kept.

In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

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

Customer Key	Name	State
1001	Christina	California

When to use Type 1

Type 1 slowly changing dimension should be used when it is not necessary for the data warehouse to keep track of historical changes.

Type 2 Slowly Changing Dimension

In Type 2 Slowly Changing Dimension, a new record is added to the table to represent the new information. Therefore, both the original and the new record will be present. The newer record gets its own primary key.

In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

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

Customer Key	Name	State
1001	Christina	Illinois
1005	Christina	California

When to use Type 2:

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

Type 3 Slowly Changing Dimension

In Type 3 Slowly Changing Dimension, there will be two columns to indicate the particular attribute of interest, one indicating the original value, and one indicating the

current value. There will also be a column that indicates when the current value becomes active.

In our example, recall we originally have the following table:

Customer Key	Name	State
1001	Christina	Illinois

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

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

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

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

When to use Type 3:

Type III slowly changing dimension should only be used when it is necessary for the data warehouse to track historical changes, and when such changes will only occur for a finite number of time.

What is Staging area why we need it in DWH?

Staging area is required in Data warehousing, when we need to process source flat file data into data warehouse or data mart, first we will load it into the staging table. While loading it into the staging table, we will do cleansing activities like LTRIM(), RTRIM(), TO_DATE(), TO_NUM(), TO_INT() to make it consistent format.

- Since it is one-to-one mapping from flat file to staging we do truncate and reload.
- Staging is the temporary storage area in Date Warehouse.

Data cleansing

Weeding out unnecessary or unwanted things (characters and spaces etc.) from incoming data to make it more meaningful and informative

Data merging

Data can be gathered from heterogeneous systems and put together

Data scrubbing

Data scrubbing is the process of fixing or eliminating individual pieces of data that are incorrect, incomplete or duplicated before the data is passed to end user.

Data scrubbing is aimed at more than eliminating errors and redundancy. The goal is

also to bring consistency to various data sets that may have been created with different, incompatible business rules.

ODS (Operational Data Sources):

My understanding of ODS is, its a replica of OLTP system and so the need of this, is to reduce the burden on production system (OLTP) while fetching data for loading targets. Hence its a mandate Requirement for every Warehouse.

So every day do we transfer data to ODS from OLTP to keep it up to date?

- OLTP is a sensitive database they should not allow multiple select statements it may impact the performance as well as if something goes wrong while fetching data from OLTP to data warehouse it will directly impact the business.
- ODS is the replication of OLTP.
- ODS is usually getting refreshed through some oracle jobs.
- Enables management to gain a consistent picture of the business.

What is a surrogate key?

A surrogate key is a substitution for the natural primary key. It is a unique identifier or number (normally created by a database sequence generator) for each record of a dimension table that can be used for the primary key to the table.

A surrogate key is useful because natural keys may change.

What is the difference between a Primary Key and a Surrogate Key?

A **Primary Key** is a special constraint on a column or set of columns. A primary key constraint ensures that the column(s) so designated have no NULL values, and that every value is unique. Physically, a primary key is implemented by the database system using a unique index, and all the columns in the primary key must have been declared NOT NULL. A table may have only one primary key, but it may be composite (consist of more than one column).

A **Surrogate Key** is any column or set of columns that can be declared as the primary key instead of a "real" or natural key. Sometimes there can be several natural keys that could be declared as the primary key, and these are all called candidate keys. So a surrogate is a candidate key. A table could actually have more than one surrogate key, although this would be unusual. The most common type of surrogate key is an incrementing integer, such as an auto increment column in MySQL, or a sequence in Oracle, or an identity column in SQL Server.

ETL

ETL (*Extract Transform Load*)

The process of extracting data from OLTP or any other remote system and transforming and then loading into data ware house is known as ETL.

ETL Tools

- Informatica – *popular ETL tool*
- Ab – Initio
- Data Stage
- SSIS (SQL server integration system)
- Oracle PL/SQL
- Oracle DWH tool

Most of DWH and integration projects use Informatica;

- 80% DWH projects
- 20% Integration projects

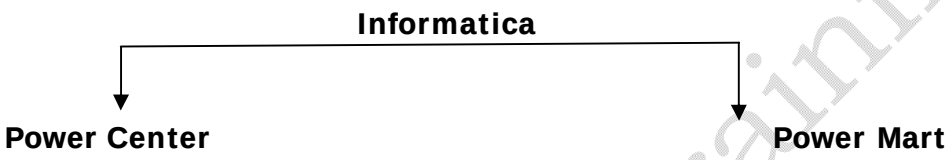
Advantages of Informatica compared to other tools:

i. User friendly
ii. More secure
iii. Easy to maintain (Easy to debug if job fails)
iv. It can extract data from any system (SAP, flat files, COBOL, systems, etc)

INFORMATICA

- Informatica is one of the ETL tool.
- It is client-server architecture
- Informatica is comprised of 2 parts
  - Power center
  - Power mart

Before purchase of Informatica you should specify whether you want power center or power mart.



Power Center: It is used by large organizations to handle larger volumes of data. We can configure multiple servers. If one server goes down then it automatically switches on to another server.

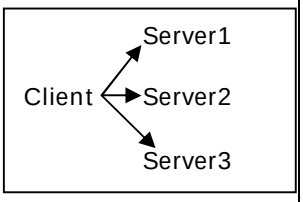
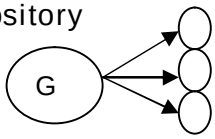
- Power center is nothing but an Informatica and Informatica is a ETL tool.
- Server installation is the job of Informatica admin team. They install Informatica server in the UNIX box.
- We can create Global Repository

Power Mart: It is used by small organization to handle smaller volumes of data.
We can create local repositories.

Client → Server

What is Repository?

- It is a Meta-data container, whatever the code (mapping / session / workflows / folders / transformations) we create in Informatica it will be saved in repository.

Power Center	Power Mart
We can configure multiple servers.  If one server goes down, it automatically switch on to other server.	It contains only one server Client---Server
Used by large organizations.	Used by small organizations
Used to create global repository  	

- Install PowerCenter(PC)client is installed in local machine.
- Informatica Server should install in Unix Box (This is admin activity).

Install PowerCenter(PC) Client

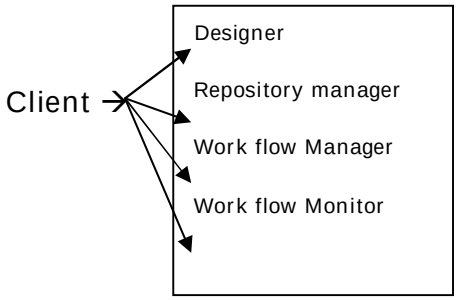
i. We need to raise a request to admin team to get install PowerCenter client. If they provide access to the software, we need to get it install in our local machine.

ii. After installing PowerCenter Client, we need to add a repository and the corresponding server.

The latest Informatica version in the market is 9.5

Start → Program Files →

Informatica PowerCenter 8.6.0 →



PowerCenter client contains 4 components or tools

1. **Designer :** To create mappings.
2. **Repository Manager :**

   a. Used to create folders.
   b. We can create Deployment Groups, Labeling, Version.

3. **Work flow Manager :**
   a. To create tasks and workflows.
   b. To run the workflow.
4. **Work flow Monitor:** To know the status of the workflow / task, whether it succeeded or failed.

In general we will have 3 repositories.

1. Development Repository
2. Testing Repository
3. Production Repository

How to add a repository and the server:

1. Go to repository tab in the toolbar, using Designer.
2. Click on Add option.
3. Type repository name and your username and click OK.
4. Configure server by clicking on more options with domain name, port number and gateway host.
5. The above steps 1 to 4 should perform one time activity, when you login first time.

In Informatica code is nothing but mapping, session and workflow

Mapping:

- Mapping is one of the basic elements in Informatica code. It is done in PC—client—designer.
- Mapping is the set of transformations between the source and target. **(OR)**
- It is the pipeline which tells how to flow data from source to target.
- To extract data from source table(S) and load in target table (Emp_t), we need to create a mapping.

Transformation: which transforms (change) the data while processing from source to target.

Session:

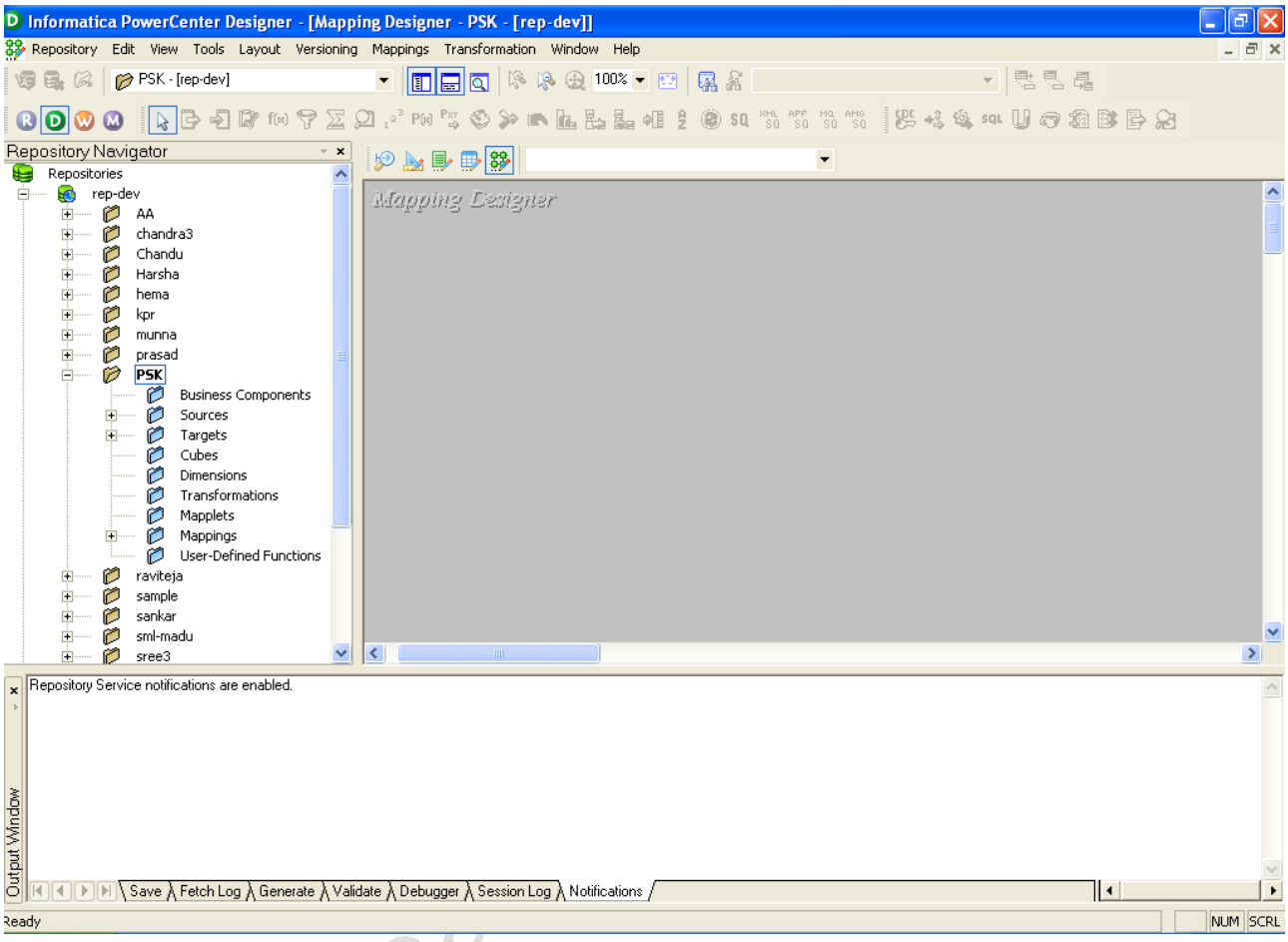
- To run a mapping we need a session. Session is an instructor which sends instructions to server what to do and server will follow session instructions.
- Mapping and session is a one-to-one relationship (1mapping-1session)
- Session is created in workflow manager.

Workflow Manager: Workflow is an engine which runs 'N' number of sessions / tasks.

Workflow Monitor: It is one of the client components where mapping is done.

Structure of Designer: Designer is basically divided into 3 parts.

- Workspace
- Navigation Window and
- Output Window



Before going to develop a mapping, you need to make active your working folder / project folder.

Designer contains 5 components.

1. Source Analyzer
2. Mapping Designer
3. Mapplet Designer
4. Target Designer
5. Transformation Developer

We have identified source as EMP_S and Target as EMP_T at Database level.

To create a mapping in Informatica Designer, we need to import source and target definitions.

Source Analyzer: used to import the source definitions.

Target Designer: used to import the target definition.

Mapping Designer: used to create the mappings.

Mapping is defined as the set of transformations between source to target.

After creating the mapping we need to validate the mapping.

- To validate the mapping, go to ***mapping tab*** → click on ***validate***, to check syntactical error not logical errors***.***
- To save the mapping, go to ***Repository*** → click on ***save (ctrl + s)***

In mapping designer when we drag and drop the source definition, it will bring source qualifier automaitcally.

Why to create session: To run mapping we have to create session in workflow manager.

Workflow Manager: It contains 3 tools

1. Task Developer
2. Worklet Designer
3. Workflow Designer

Using task developer we can create following session tasks.

1. Session Task
2. Command Task
3. Email Task

Always for 1 mapping we can create 1 session(one-to-one relationship).

In *Task Developer*

Goto *tasks* tab

→ **Cretae**

→ select ***Session*** task

→ choose corresponding mapping.

After creating session, set source, target, Lookup, Stored Procedure connections.

Target load type should not be ***Bulk***, if target contains *Indexes*.

Set ***Stop on Error = 1***

How to create WorkFlow:

In Workflow Designer,

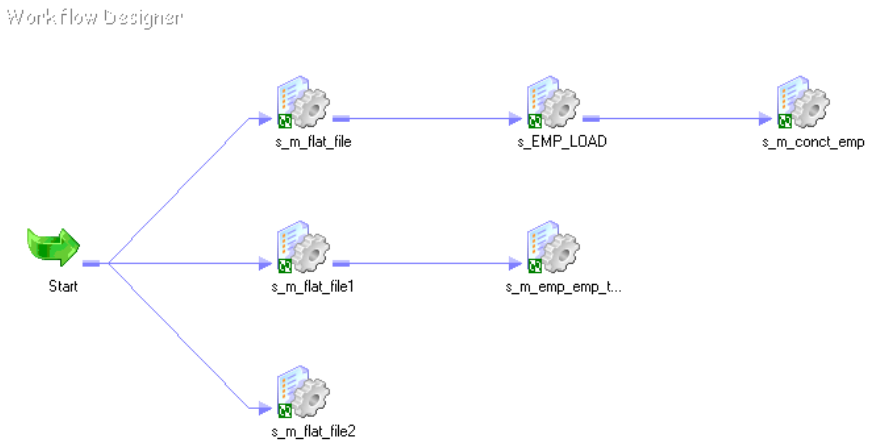
Goto *Workflow* tab

→ **Cretae**

→ Drag and Drop the session in to Workflow Designer

→ link the workflow to session using *Link Task*.

Generally for an entire project there will be one workflow.

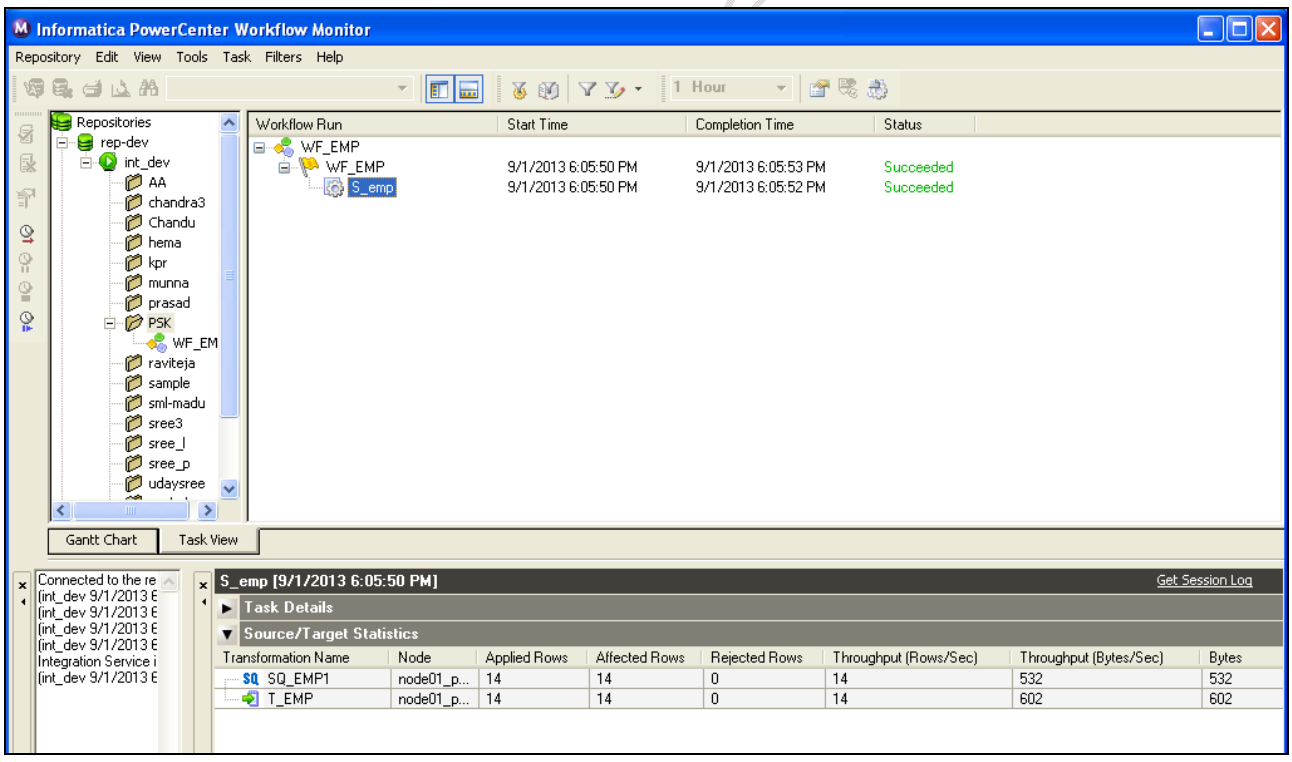


How to run workflow:

Workflow tab →start workflow **(or)**

Session → right click → start task (particular session) **(or)**

Session → right click →start workflow from task (*runs all session from this task*)



Backend execution at runtime:

1. Initiates source and target DB connection
2. Creates reader, writer and transformation threads

3. Now reader thread starts reading data from the source table. The job of reader thread is to prepare a select query based on source qualifier output ports and run that query on source DB to read the data.
4. Source Qualifier query after issuing to the database, the result of the query record by record going to process by mapping.
5. Once the target instance recieves the record, then writer thread come into the picture to write this record in to the target with insert statement if it is the relational table.
6. As per the session level commit interval, *Writer Thread* will issue the commit or at the end of success run, writer will issue one commit.

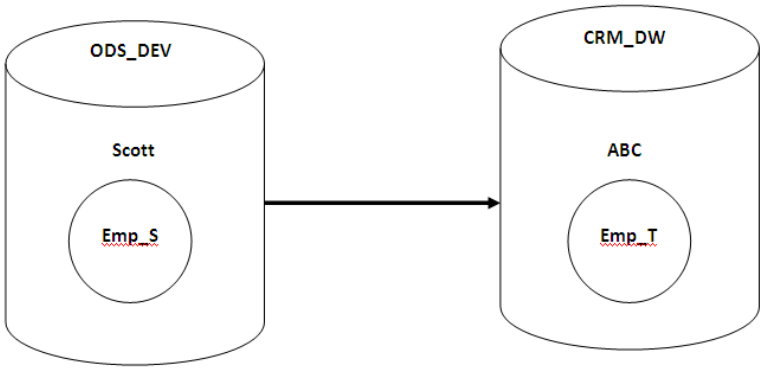
Why writer considers 'Insert' statement for source data?

Session level property, treat source rows as ***INSERT*** by default.

SCENARIOS

SCENARIO

Requirement: ***Create a simple one – to – one mapping to load data from Emp_s@ODS_DEV to Emp_T@CRM_DW***



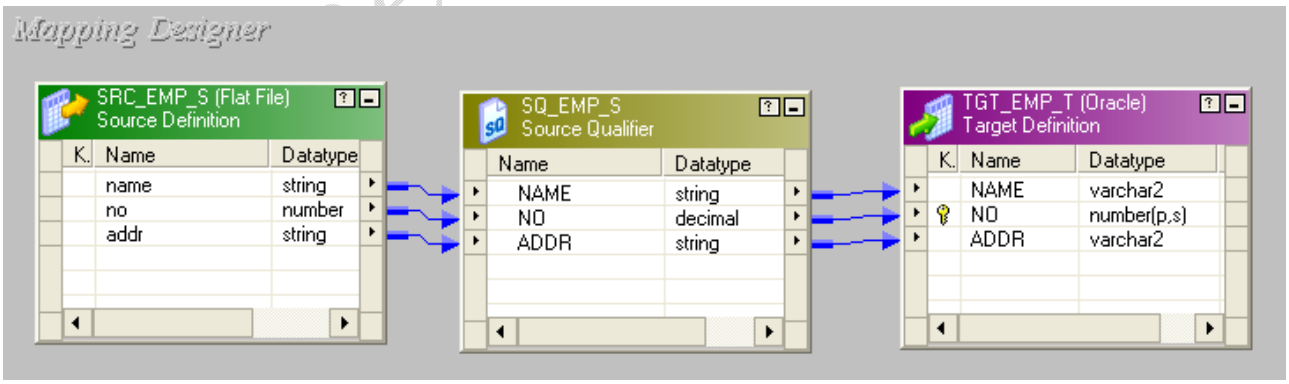
Step 1 : Import Source and Target definitions into the designer.

Step 2 : Create mapping as per development standards (referr end of the material) and drag and drop the source and target definition from navigation window in to the mapping designer.

Step 3 : As per the mapping sheet / technical design, link target columns from corresponding source columns.

Step 4 : Validate and save the mapping.

Step 5: Below is the overview of the mapping.



Step 6: To run the above mapping, we need to create session and workflow using workflow manager.

Step 7: Run the session.

Output : Validate the target data before and after the session run.

Expected Result : It should load all the records from the source table.

Actual Result : Capture this after you validate the target data.
Use space for running notes:

SCENARIO

Requirement: ***Joining the multiple table using SourQualifier Query Overwrite, if tables are in the same database.***

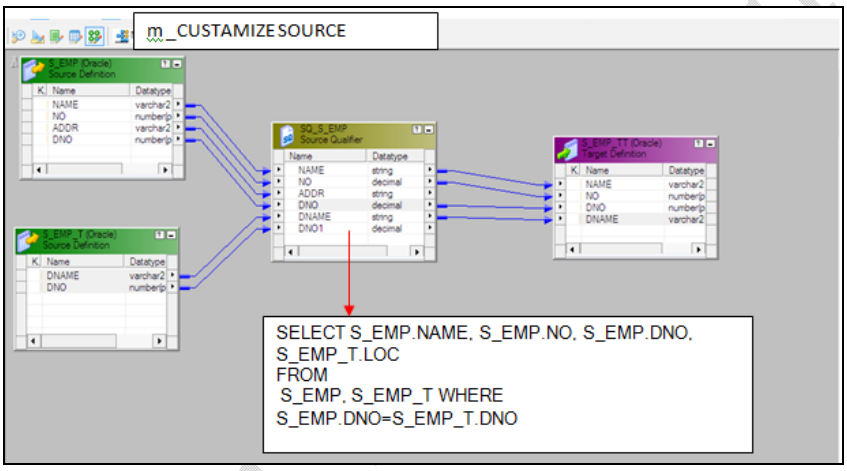


Fig. the above screenshot shows
how to customize the source qualifier with Query overwrite.

Rules to be followed when you overwrite a query:

1. Number of selected columns in the *Query Overwrite* should be same as the number of output ports, which we are connecting to the next transformation from Source Qualifier**.**
2. Order of columns in the select query should be same as output ports of source qualifier i.e., the datatype of first column in query should match with first output port in the source qualifier and same applicable for all selected columns in the query.

Use space for running notes:

SCENARIO

Requirement: ***Concat the first_name and last_name values from the source and then load it into the target.***

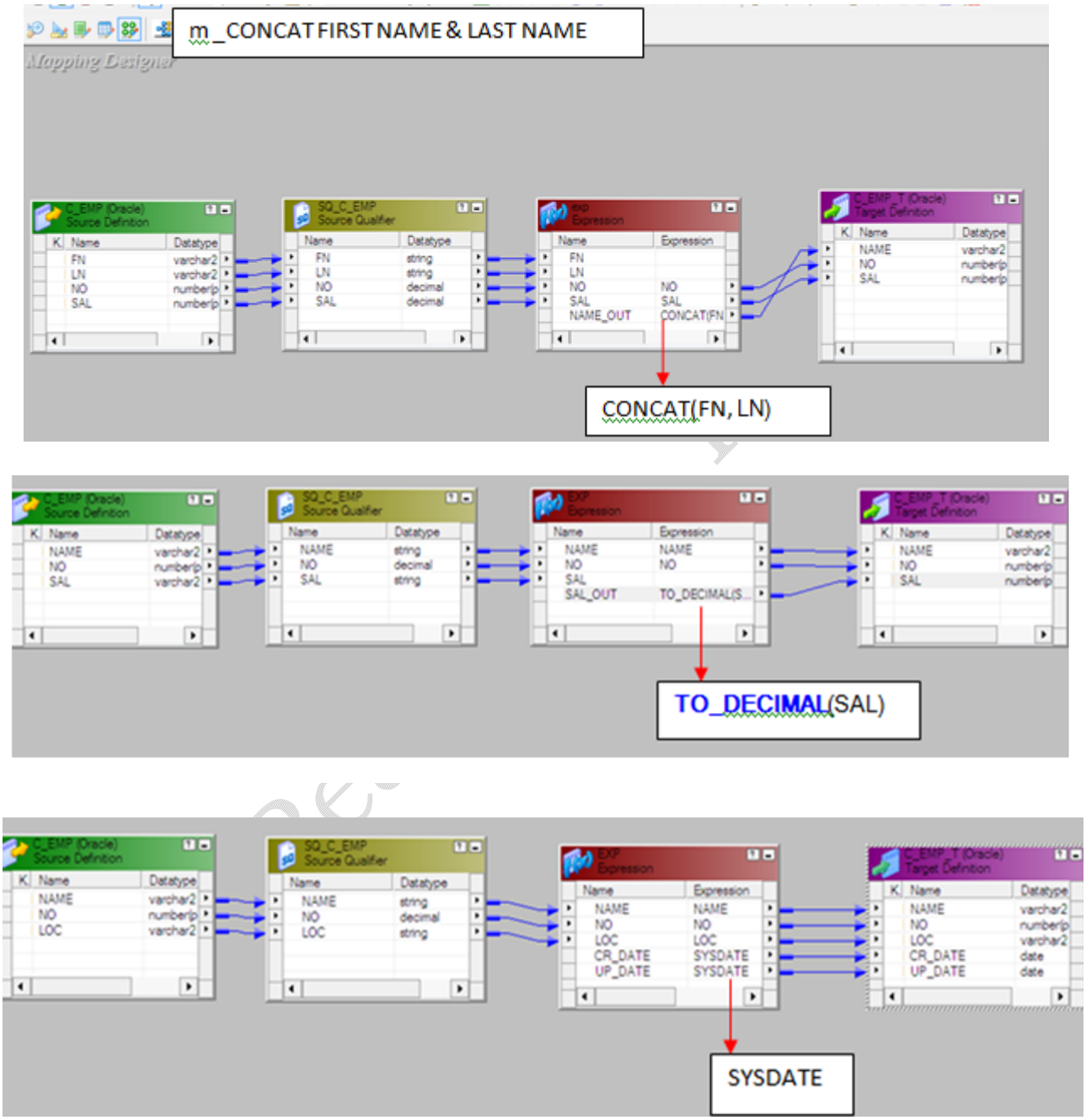


Fig.: The above screenshots shows how to perform rowlevel operations in informatica using Expression Transofrmation

Use of Expression Transformation:

1. Perform row level calculation like to_date(), to_number(), ltrim(), rtrim(), substr(), instr(), concat(), etc.

2. We cannot use aggregate functions like min(), max(), etc.

Use space for running notes:

SCENARIO

Requirement: ***Check source salary, if salary < 5000, then add salary + 1000 else send source salary itself to target.***

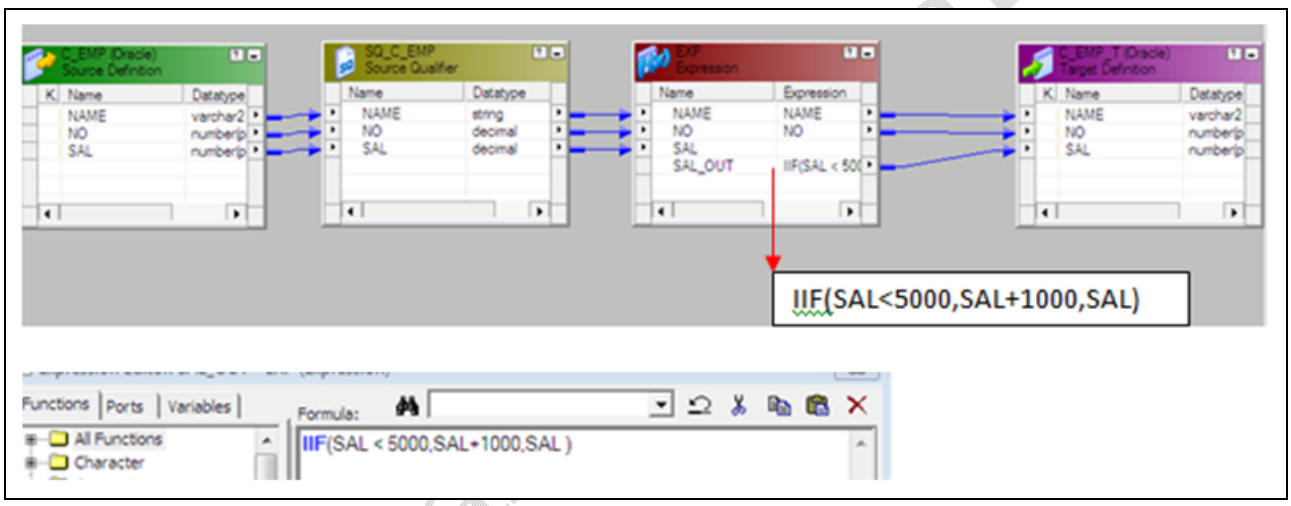


Fig.: The above screenshot shows how to user if else statement in informatica

Use space for running notes:

SCENARIO

Requirement: ***After modifying source data, before sending it to the target, how to filter the rows?***

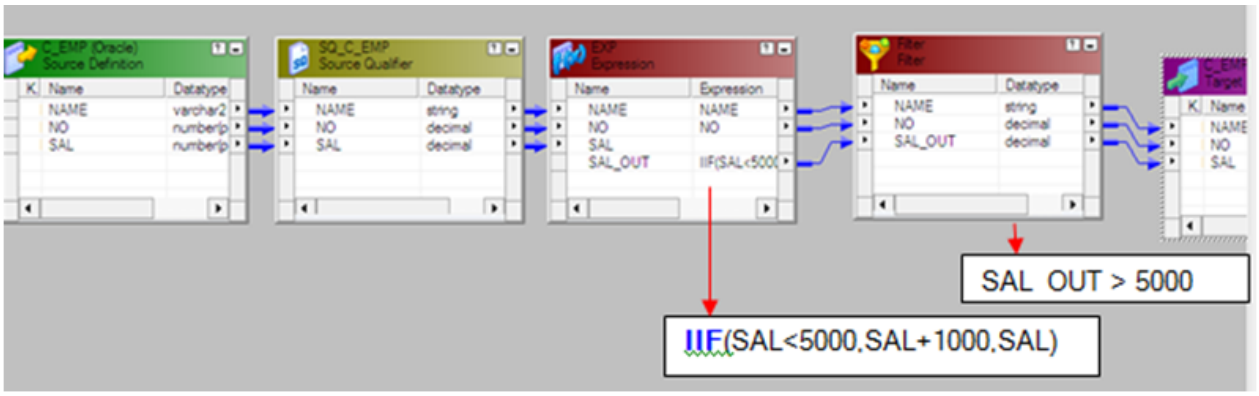


Fig.: The above screenshot shows how to use a filter transforamtion in between the pipeline.

Use of Filter Transformation:

1. To filter the source data based on the single condition. If source is a relational, we can filter the rows using source qualifier filter.
2. If source is a flat file or after modifying the data in the pipeline to filter data, before sending to the target, we need to use the filter transformation.

Use space for running notes:

SCENARIO

Requirement: ***Load the 10 department employees into one target and 20 department employees into another target.***

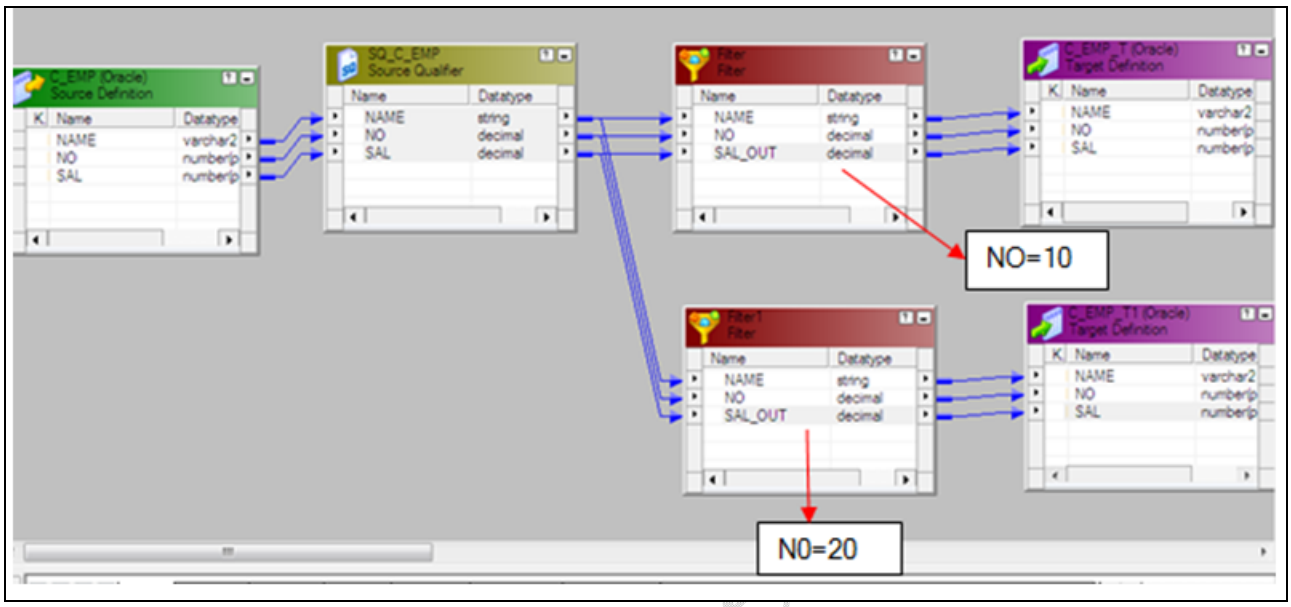


Fig.: The above screenshot shows how to use a filter transforamtion in between the pipeline.

Use of Filter Transformation:

1. To filter the source data based on the single condition. If source is a relational, we can filter the rows using source qualifier filter.
2. If source is a flat file or after modifying the data in the pipeline to filter data, before sending to the target, we need to use the filter transformation.

Use space for running notes:

SCENARIO

Requirement: ***Load the 10 department employees into one target and 20 department employees into another target using Router Transformation.***

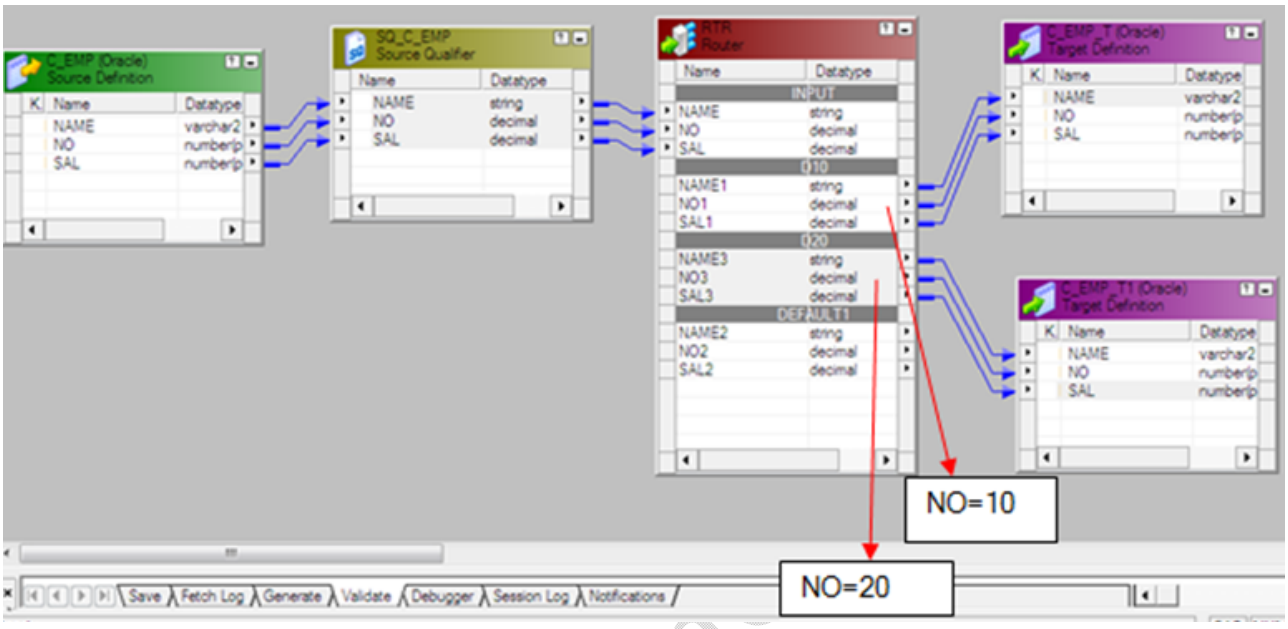


Fig.: The above screenshot shows how to use a router transforamtion to test the source data with multiple codition.

Use of Router Transformation:

1. To filter the source data based on multiple condition. Instead of multiple filter we can use single router with multiple groups.

Use space for running notes:

SCENARIO

Requirement: ***Load Emp data along with the Dname into target table consider EMP table and Dept table are in different database.***

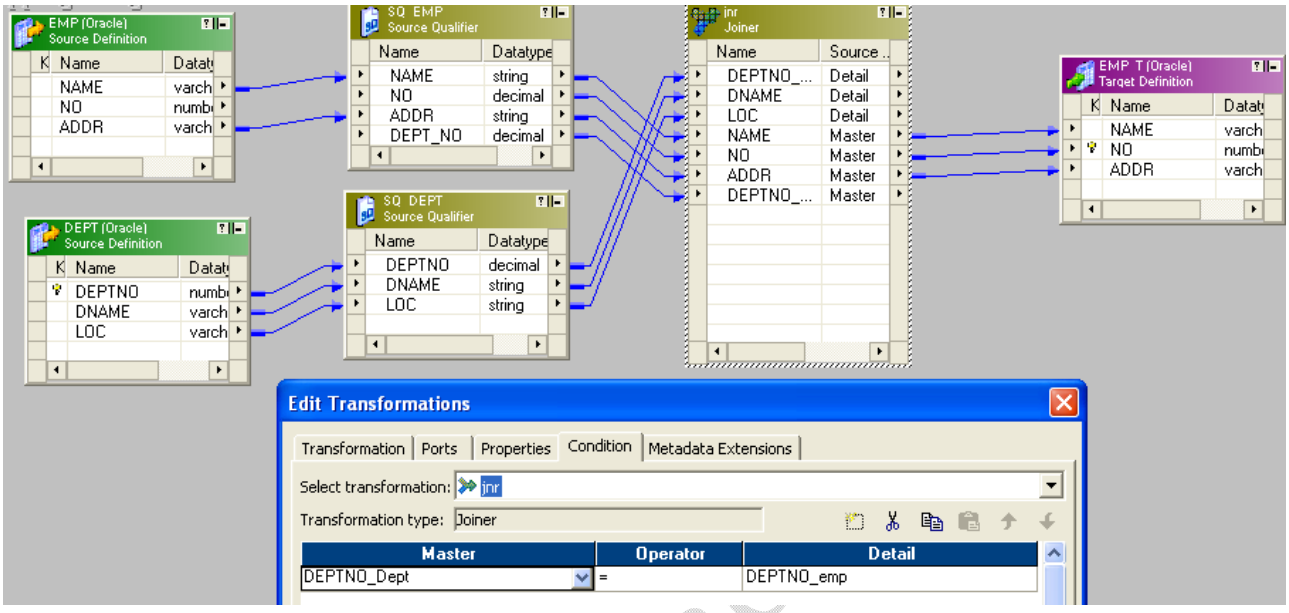


Fig.: The above screenshot shows how to join two tables if tables are in different databases or different applications using Joiner Transformation.

Joiner Transformation:

1. We can join two sources if they are in different databases.
2. It support only *Normal Join, Master outer join, Detail outer join and Full outer join*.
3. On multiple match it would return all multiple match record.
4. We can't do Querty override

Use space for running notes:

SCENARIO

Requirement: ***Transpose column into rows, load based on Addr1 one record in target and based on Addr2 another record***

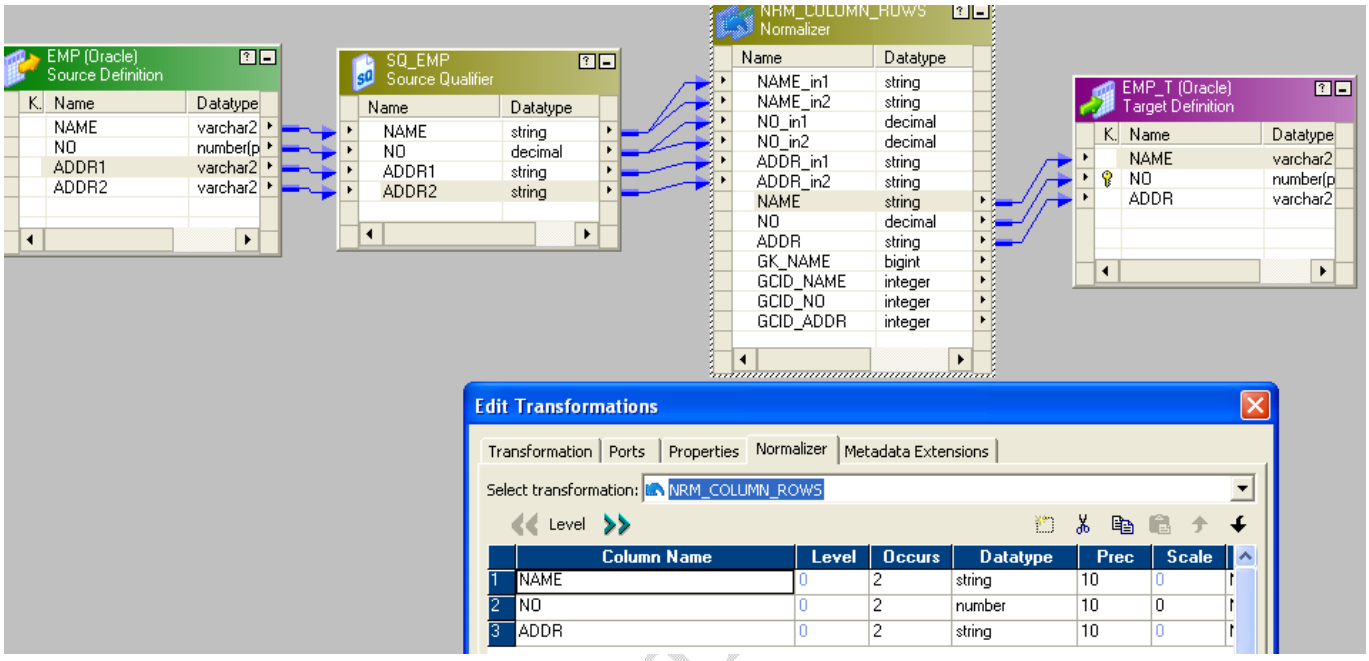


Fig.: The above screenshot shows how to transpose columns into rows using Normalizer Transformation.

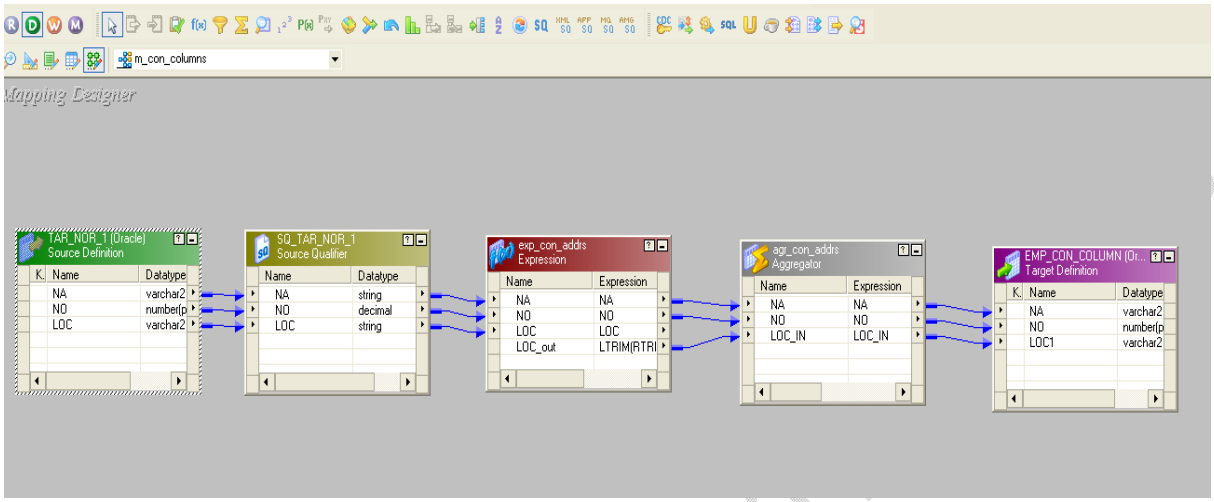
Use of Normalizer Transformation:

1. We can transpose columns into rows.
2. To generate multiple records in target based on single source record
3. To process Cobol source data we use Narmalizer instead of Source Qualifer

Use space for running notes:

SCENARIO

Requirement: ***Transposing rows into columns using variable port and aggregator transformation.***



Transformation | Ports | Properties | Metadata Extensions

Select transformation: exp_con_addrs

Transformation type: Expression

	Port Name	Datatype	Prec	Scale	I	O	V	Expression
1	NA	string	2	0	☑	☑	☐	NA
2	NO	decimal	3	0	☑	☑	☐	NO
3	LOC	string	3	0	☑	☑	☐	LOC
4	Vari_pt	string	50	0	☐	☐	☑	IIF(NA != NA_V AND NO != NO_V,LOC,LOC_V || ', ' || LOC)
5	LOC_V	string	50	0	☐	☐	☑	iif(NO != NO_V,LOC,LOC_V || ', ' || LOC)
6	NA_V	string	10	0	☐	☐	☑	NA
7	NO_V	decimal	10	0	☐	☐	☑	NO
8	LOC_out	string	50	0	☐	☑	☐	LTRIM(RTRIM(Vari_pt,','),',')

Variable ports logic in detail

Vari_pt	**IIF (NA != NA_V AND NO != NO_V,LOC,LOC_V || ', ' || LOC)**
LOC_V	**IIF (NO != NO_V,LOC,LOC_V || ', ' || LOC)**
LOC_out	**LTRIM(RTRIM(Vari_pt,','),',')**

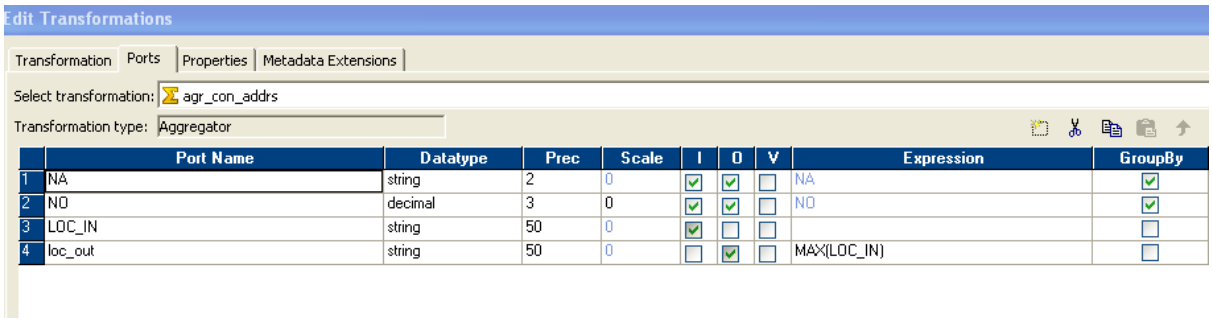


Fig.: The above screenshots shows how to set the logic in expression and aggregator transformation.

Use of Normalizer Transformation:

1.
2.
3.

Use space for running notes:

SCENARIO

Requirement: ***How to Combine multiple pipelines in a informatica before loading into the target.***

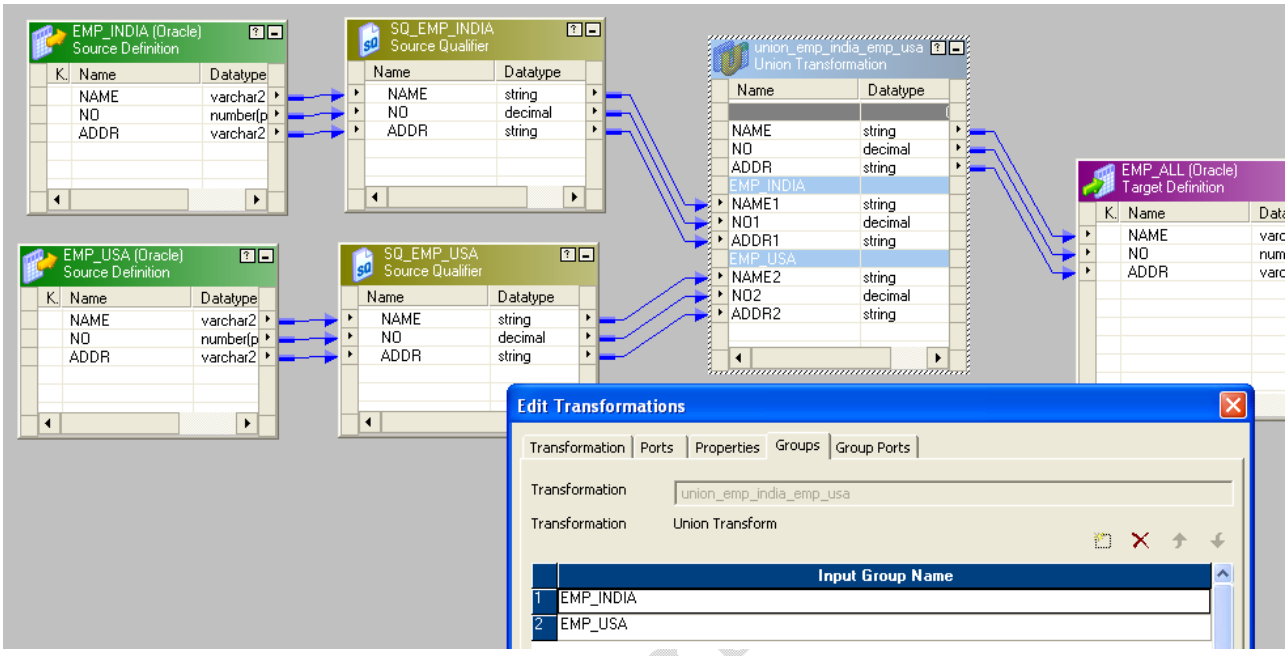


Fig.: The above screenshot shows how to combine to multiple pipelines in informatica using Union Transformation.

Use of Union Transformation:

1. We can use union to combine multiple pipeline.
2. If Sources are relational we can combine multiple select statement in Source Qualifier query override using union query.

Rule: All groups should have same number of columns/ports.

Use space for running notes:

SCENARIO

Requirement: ***Load Emp data into target along with Dname values consider Emp and Dept are in different databases.***

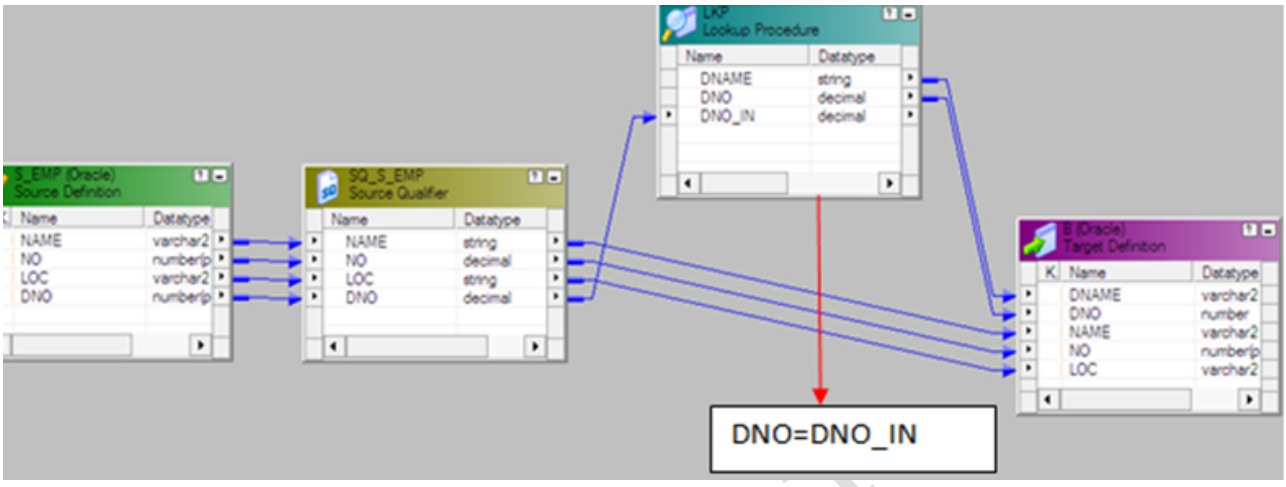


Fig.: The above screenshot shows how to use lookup Transformation to get relavent data based on the condition.

Use of Lookup Transformation:

1. To check the value exist or not in a table or a file based on condition.
2. If lookup does not find a match in cache based on the condition it would return null value thorugh output port.
3. If lookup find a match in cache based on the condition it would return the relavent value thorugh output port.

Use space for running notes:

SCENARIO

Requirement: ***Load Emp data into target along with Dname values consider Emp and Dept are in different databases using unconnected Lookup.***

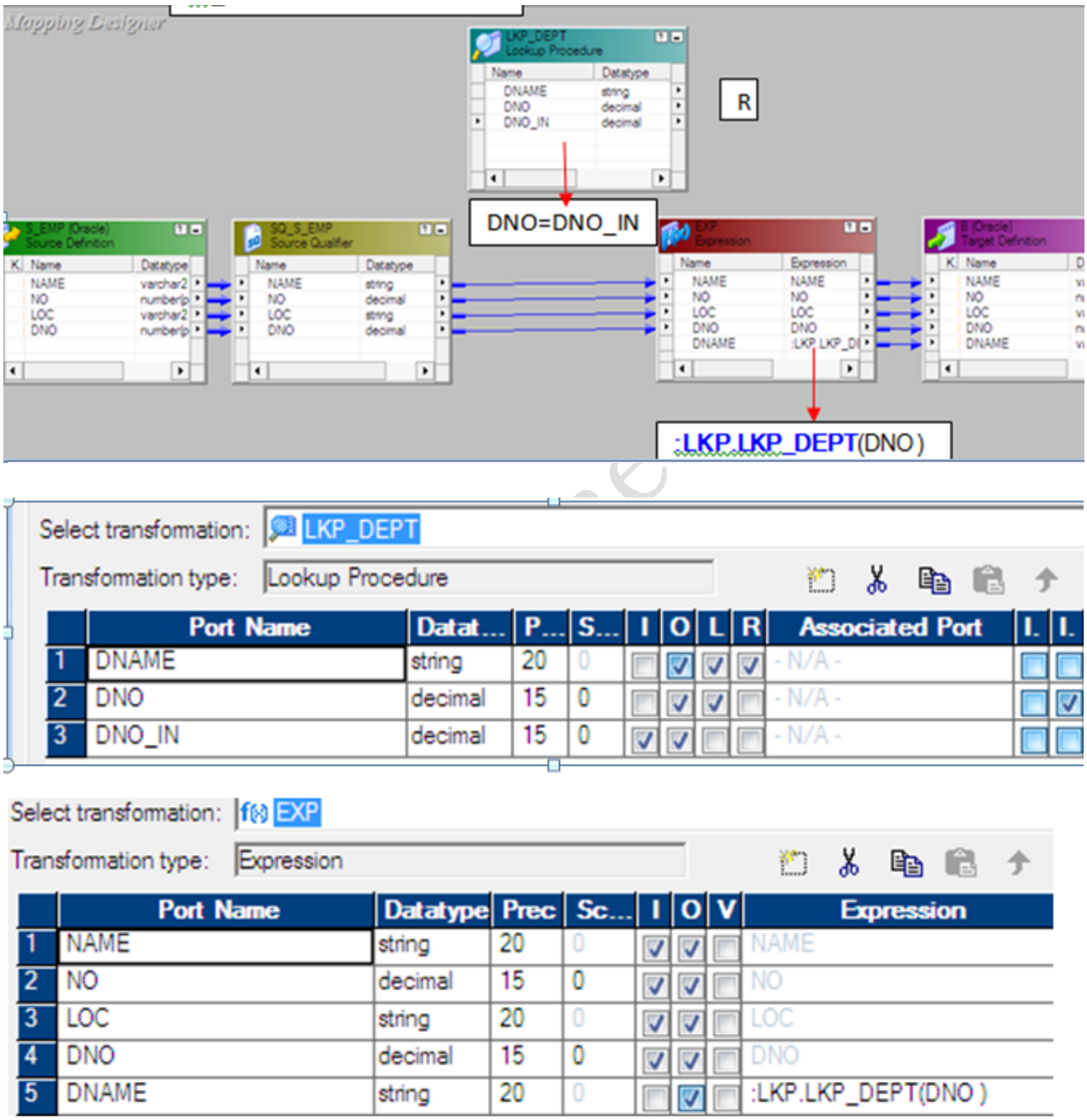


Fig.: The above screenshot shows how t o use un-connected lookup Transformation to get relavent data based on the condition.

Use of Un connected Lookup Transformation:

1. We can use multiple times within the mapping by calling in expression using :LKP function.
2. It would return single port..
3. To return multiple ports using unconnected lookup we need to override a query in lookup with CONCAT multiple columns later in expression we need to split
4. It receives input values via argument

Use space for running notes:

SCENARIO

Requirement: ***How to perform Update else insert using Session level properties***

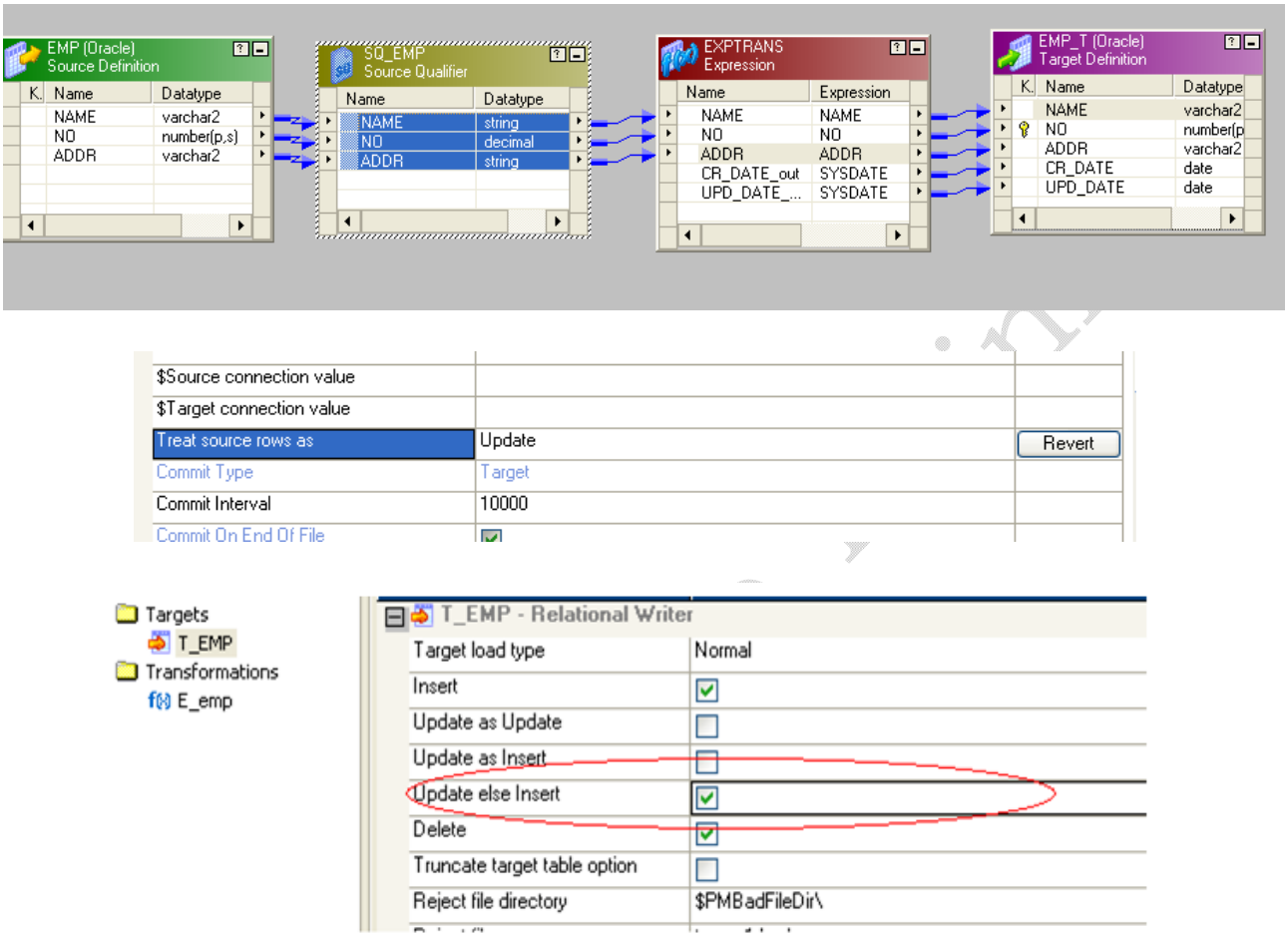


Fig.: The above screenshot shows how to do the session level update else insert.

What is Update else Insert

If source record doesn't exist in the target table then insert,if source record is already present in the target then update existing record with source record

Steps:

1. Pre reuest Target table must contain Primary key
2. Set session level property treat source row as UPDATE
3. Select Target property at session level update else insert

Drawbacks:

1. It would update all columns except primary key

2. We can't update the specific columns in the target
3. Source record present in the target even there is no change in the between the source and target it will mark for update

Use space for running notes:

SCENARIO

Requirement: ***Find a Maximum salary Dept wise from the source File by using Aggregator Transformation***

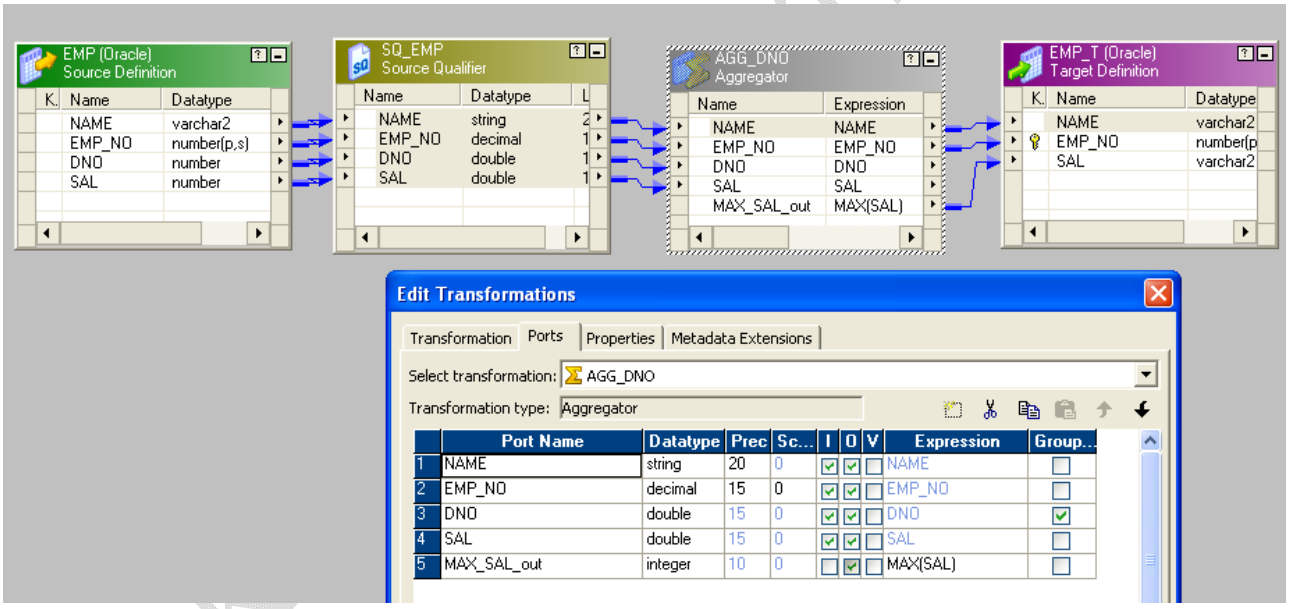


Fig.: The above screenshot shows how to use the Aggregate function in Informatica by using Aggregator Transformation

Use of Aggregator Transformation:

1. To perform the Aggregate calculations betweeen the transformations
2. If source is flat file we can't do aggregations in the source qualifier Query
3. To eliminate the duplicate rows from the source we can use the aggregator group by on all columns
4. Always Group will return the last record

Use space for running notes:

SCENARIO

Requirement: ***How to update the data in a target table using mapping***

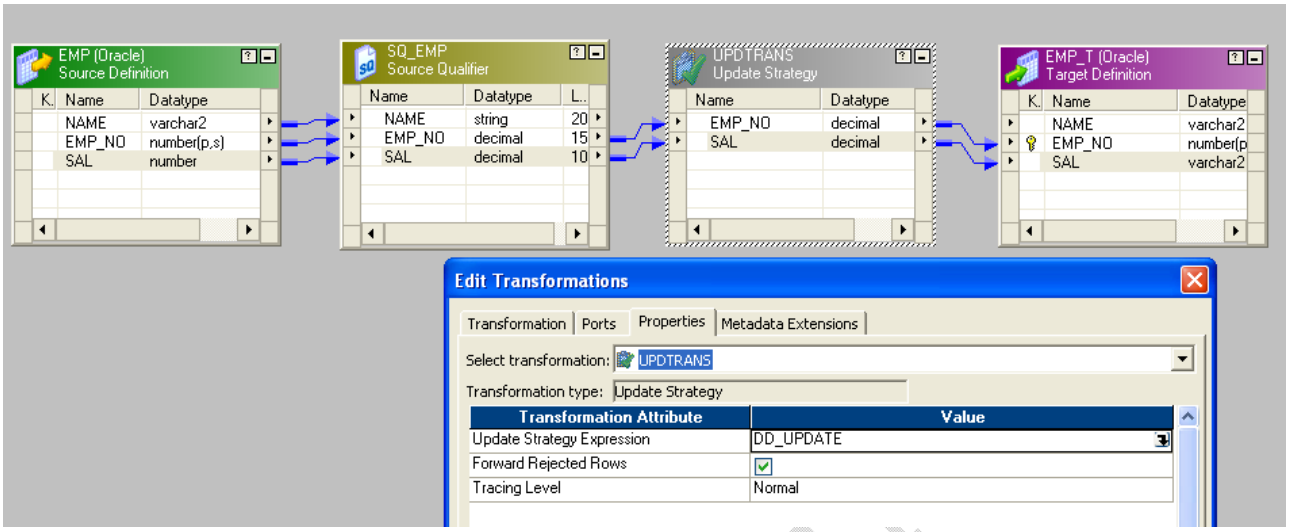


Fig.: The above screenshot shows how to do the updates in target by using mapping approach

Use of Update strategy Transformation:

1. To mark the data in the target for Update using mapping level
2. To flag/mark for Delete/Reject/Insert/Update
3. Make sure to set treat source rows as Data Driven in the session level property when we use upadte strategy transformation in the mapping
4. Pre request to update the target should have primary key

Use space for running notes:

SCENARIO

Requirement: ***Update else Insert (SCD Type – 1) through mapping***

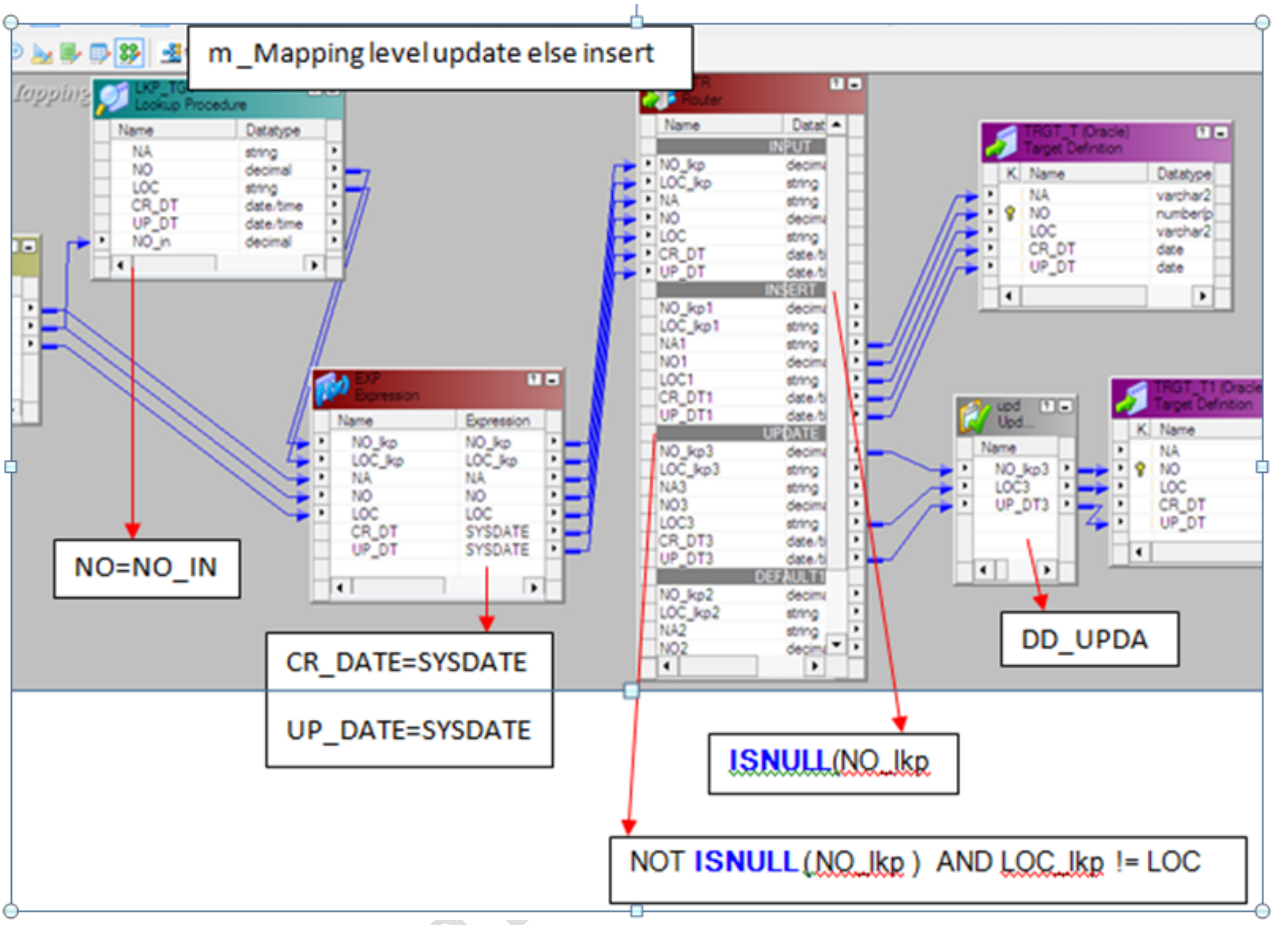


Fig.: The above screenshot shows how to perform the Update else Insert(SCD Type-1)

Steps:

1. Pre reuest Target table must contain Primary key
2. Do lookup on the target table based on the primary key Condition
3. If source and target both are in the same database instead of lookup we can perform outer join in the soruce qualifer over ride to check the record exists or not in the target table
4. Uisng Update strategy transformation we are flagging/Mark source data for Update

Use space for running notes:

SCENARIO

Requirement: ***How to Sort the Source data in between the transformations***

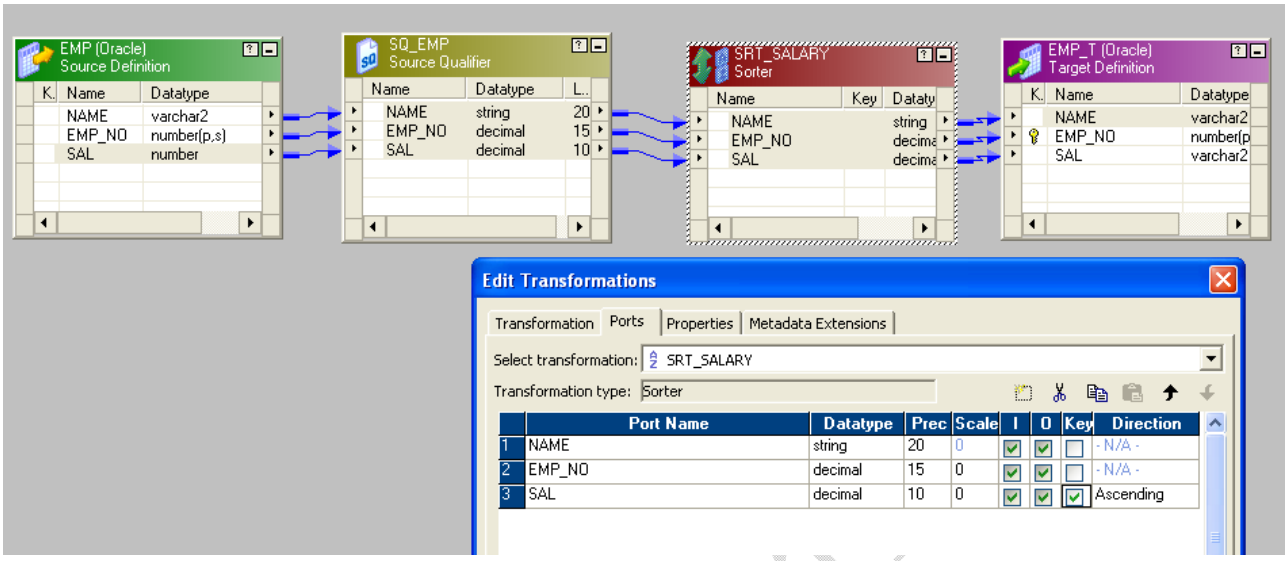


Fig.: The above screenshot shows how to Sort the Source data in between the transformations

Use of Sorter Transformation:

1. Used to sort the data before sending to the target.If source is a relational we can do it in a source qualifer query
2. If source is a flat file ,we can't do the query over ride .So we need to use the Sorter Transformation
3. We can eliminate the duplicate data from flat file soucre using Sorter With distinct option

Use space for running notes:

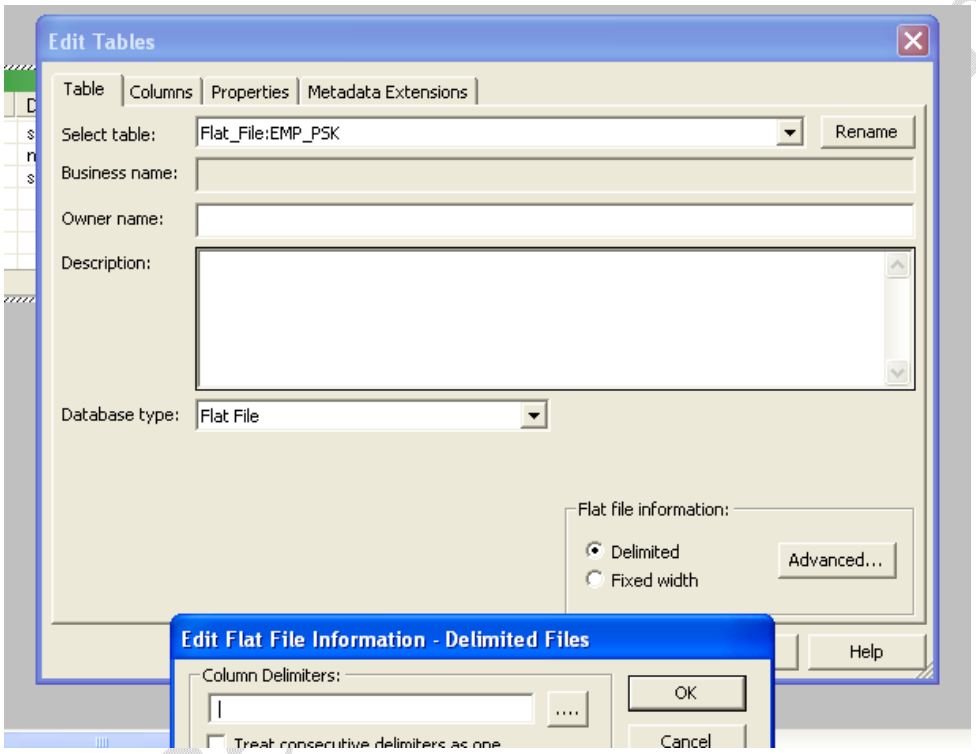
SCENARIO

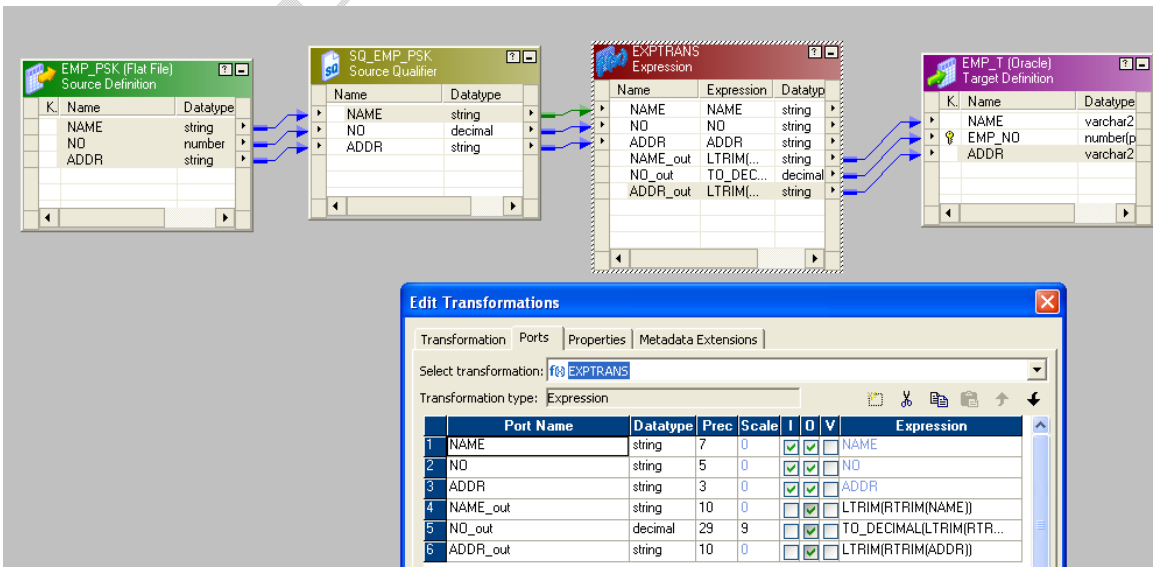
We can process Fixed Width and Deliemeted Files in Informatica

Requirement: *How to process Delimeted Flat File Source through Informatica*

Steps to Import or Create Flat File Definition

- Go to Soure tab → Create → Select type as a Flat File → Add a cloumns → Select the Delimeted as per the requirement(Select the type of a deleimeter like **,,~,|**





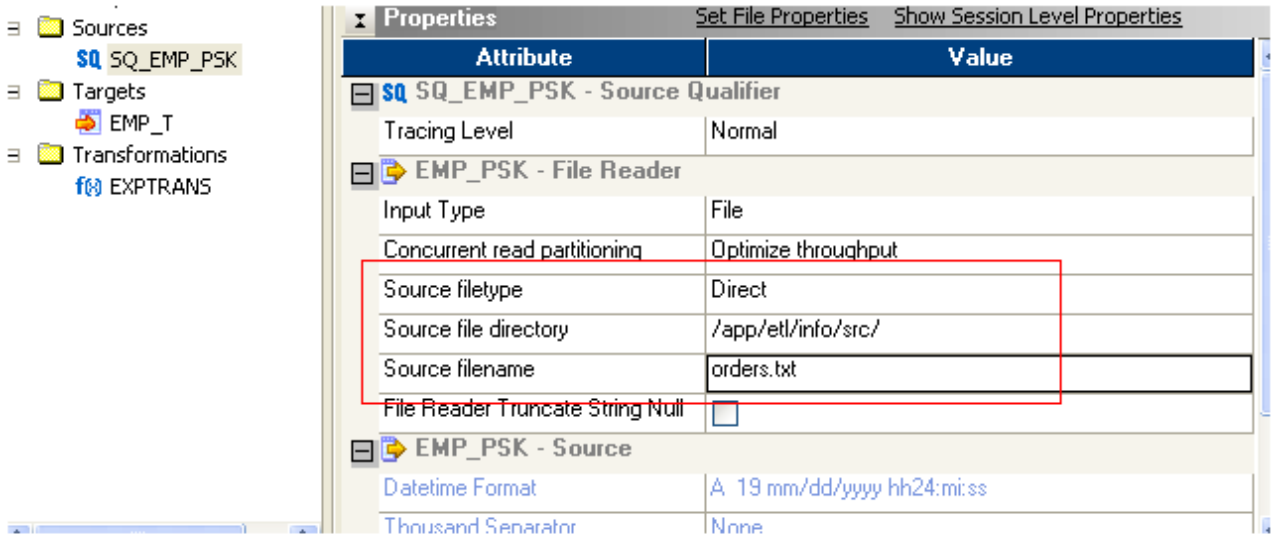
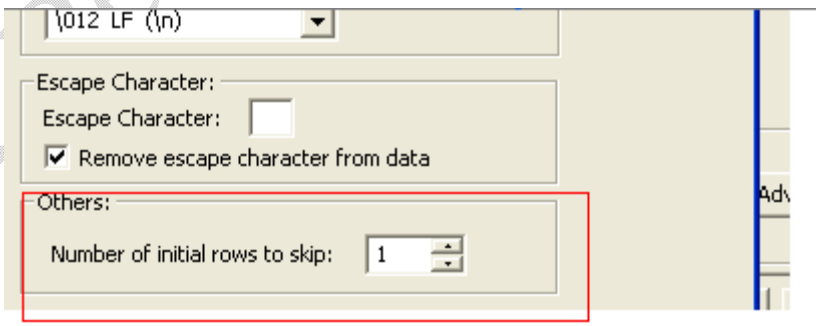


Fig.: The above screenshot shows how to load flat file source data in to the target table

Steps:

1. Make sure to upload source File in to the informatica server Unix Box
2. Specify the Source file Directory path
3. If soruce is Delimeted then we need to specify the delimeter as per the requirement in the source Definition at Source Analyzer
4. If soruce is Fixed Width then we need to specify the size of each column as per the file structure in the source Definition at Source Analyzer
5. If Source contains Header to skip the first row ,choose the number of Rows to skip option as shown in the below



Use space for running notes:

SCENARIO

Requirement: ***How to populate the surrogate key in target table using sequence generator transformation***

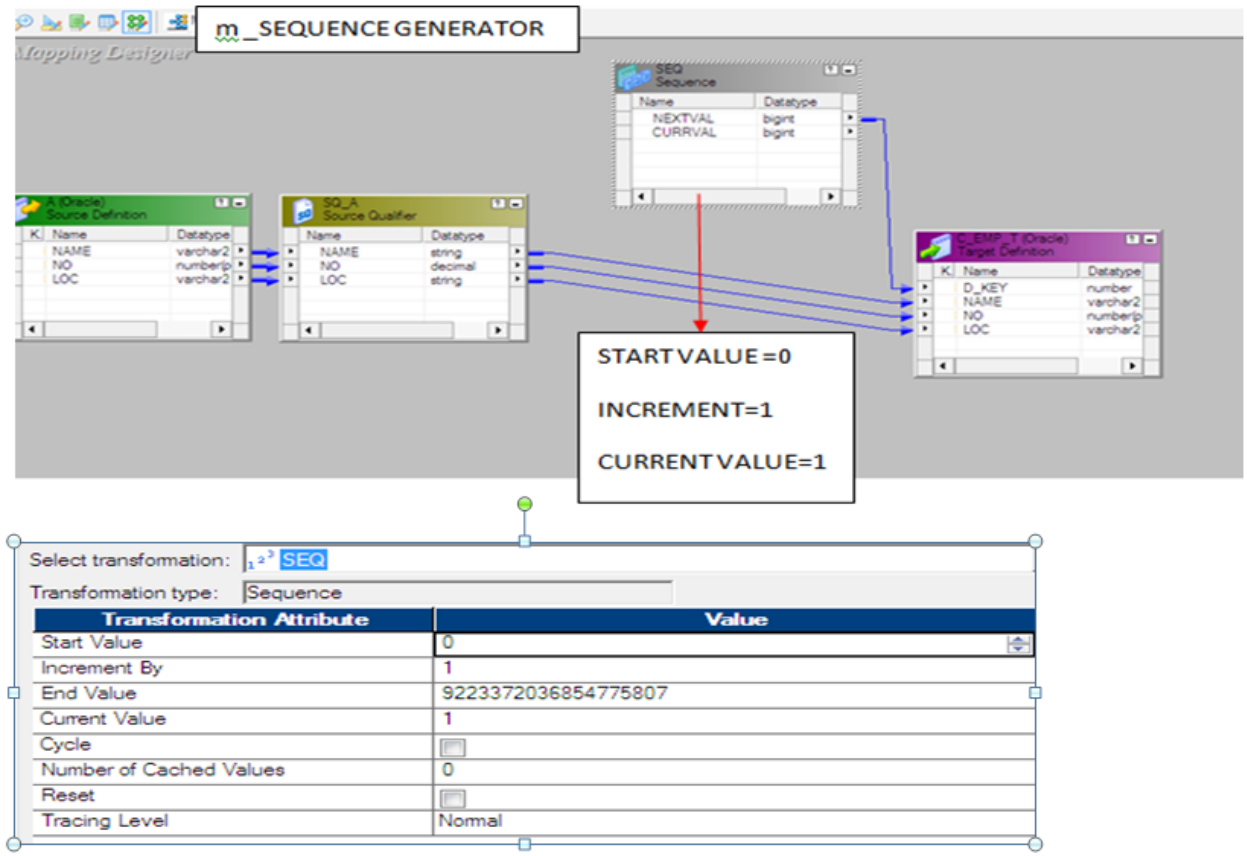


Fig.: The above screenshot shows how to use the sequnece generator transformation

Uses of sequence generator transformation

1. To populate sequence numbers in target table
2. It would generate maximum 19 digits number
3. I f we select a cycle ,after reaching the end value it will restart
4. I f we select the reset option it will restart for every session run

Use space for running notes:

SCENARIO

Requirement: ***How to send First record into First target and second record into second target***

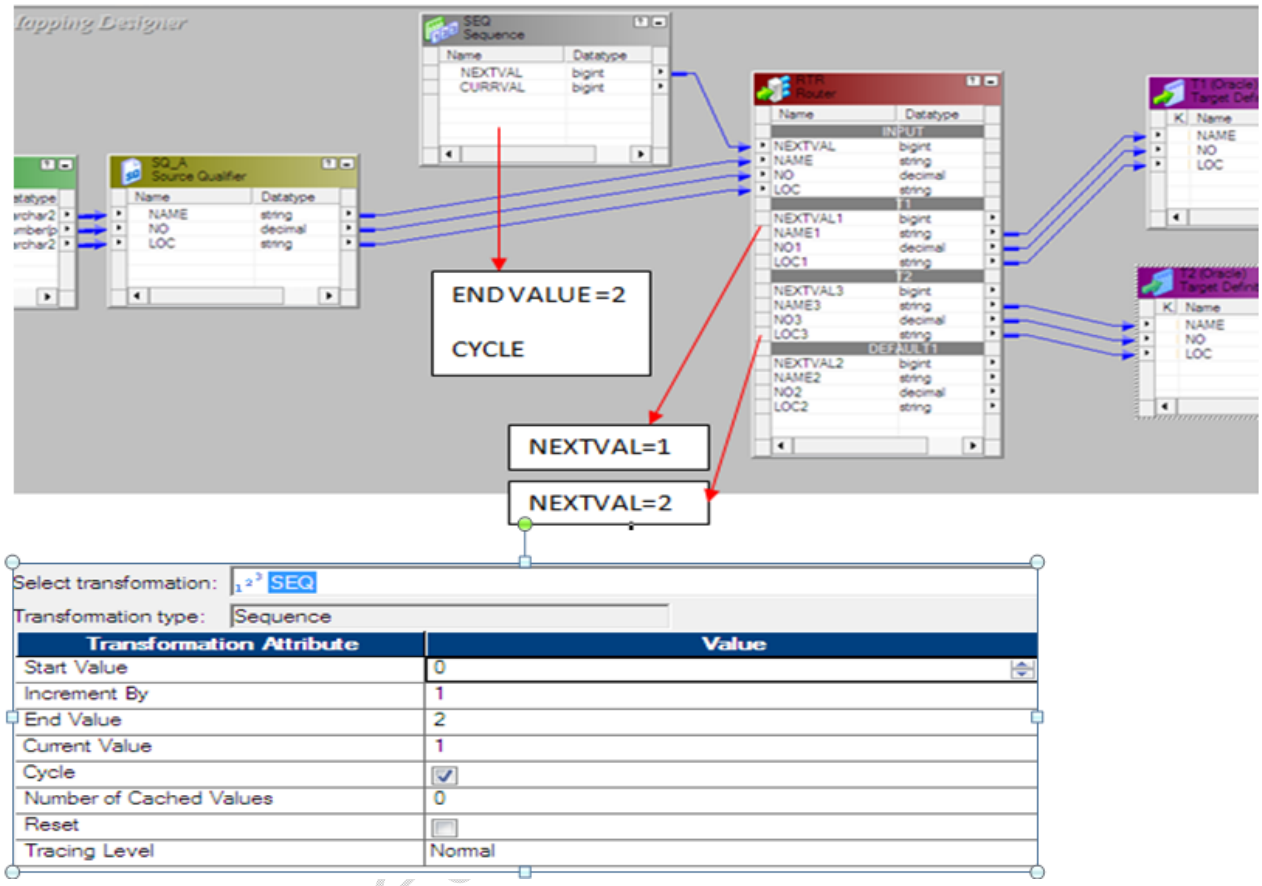


Fig.: The above screenshot shows how to load first record into first target,second record into second target

Steps:

Using sequece generator by setting the end value 2, and selecting cycle option we can route the data in to two targets using router

Use space for running notes:

SCENARIO

Requirement: ***How to load multiple flat files in to target table if all files are the same structure***

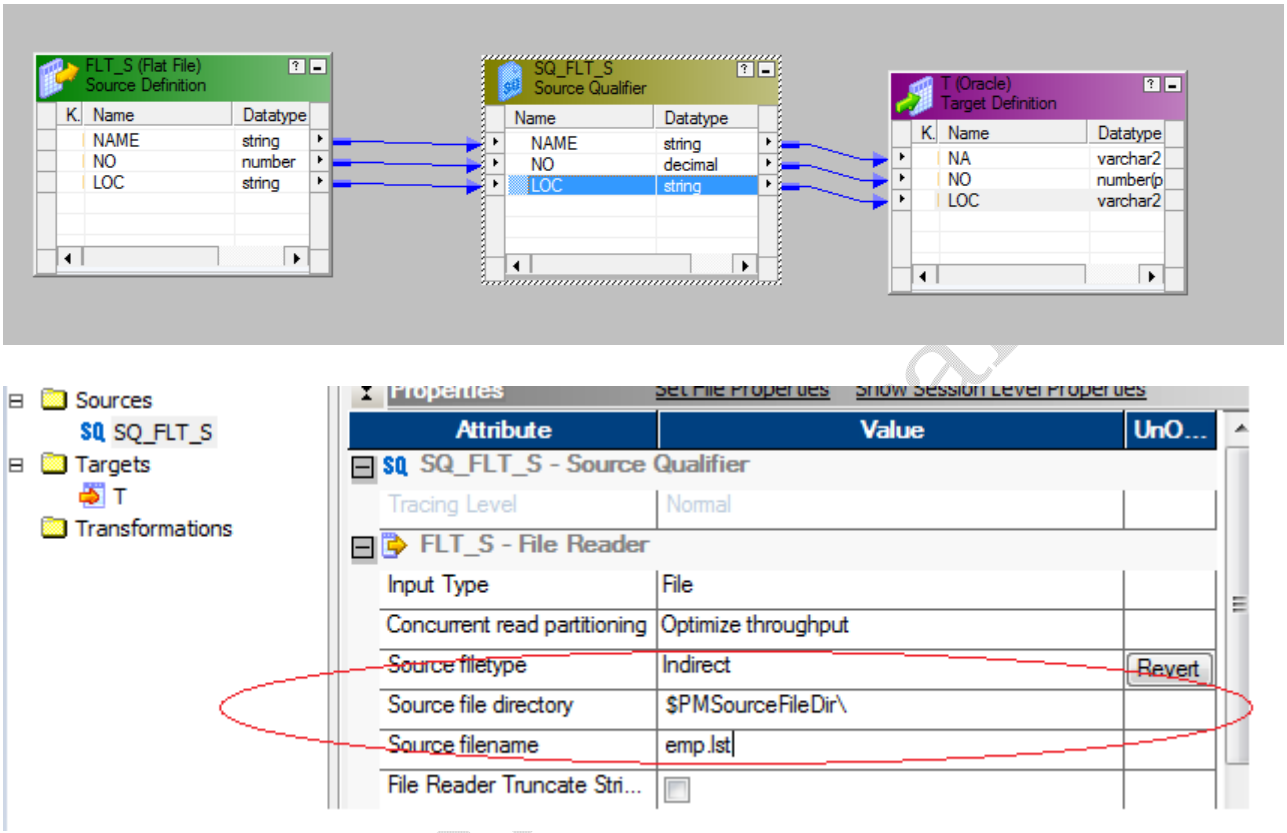


Fig.: The above screenshot shows how to set indirect option at session level to process lsit file

Steps:

1. First we need to create a mapping to load source file data in to a target table as per the source file structure
2. At session level we need to set the source file directory as a list file path
3. Source file type should be Indirect
4. Before the sesssio run we need to run a shell script to create a list file.This needs to be run before the session using command task or presession command task
5. List file which contains all the source file names not data

Use space for running notes:

SCENARIO

Requirement: ***How to populate Source flat file name to the target while processing list file***

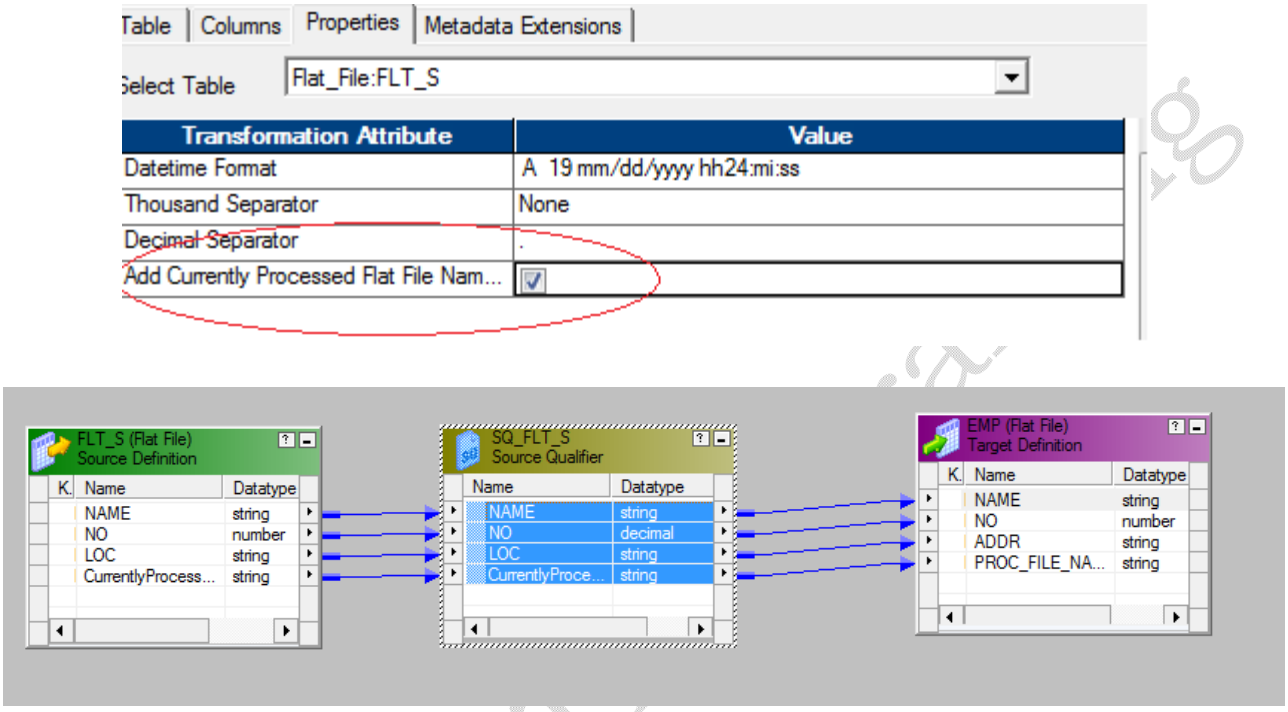


Fig.: The above screenshot shows how to read source file name in informatica

Steps:

1. In Informatica 8.6 onwards we have advanced option **add currently processing file name port** for source file definition at source analyzer
2. When we enable this option ,it will add one extra File name column in to the source definition and source qualifier

Use space for running notes:

SCENARIO

Requirement: ***How to implement SCD Type -2 using Effective date approach to maintain history in the target table***

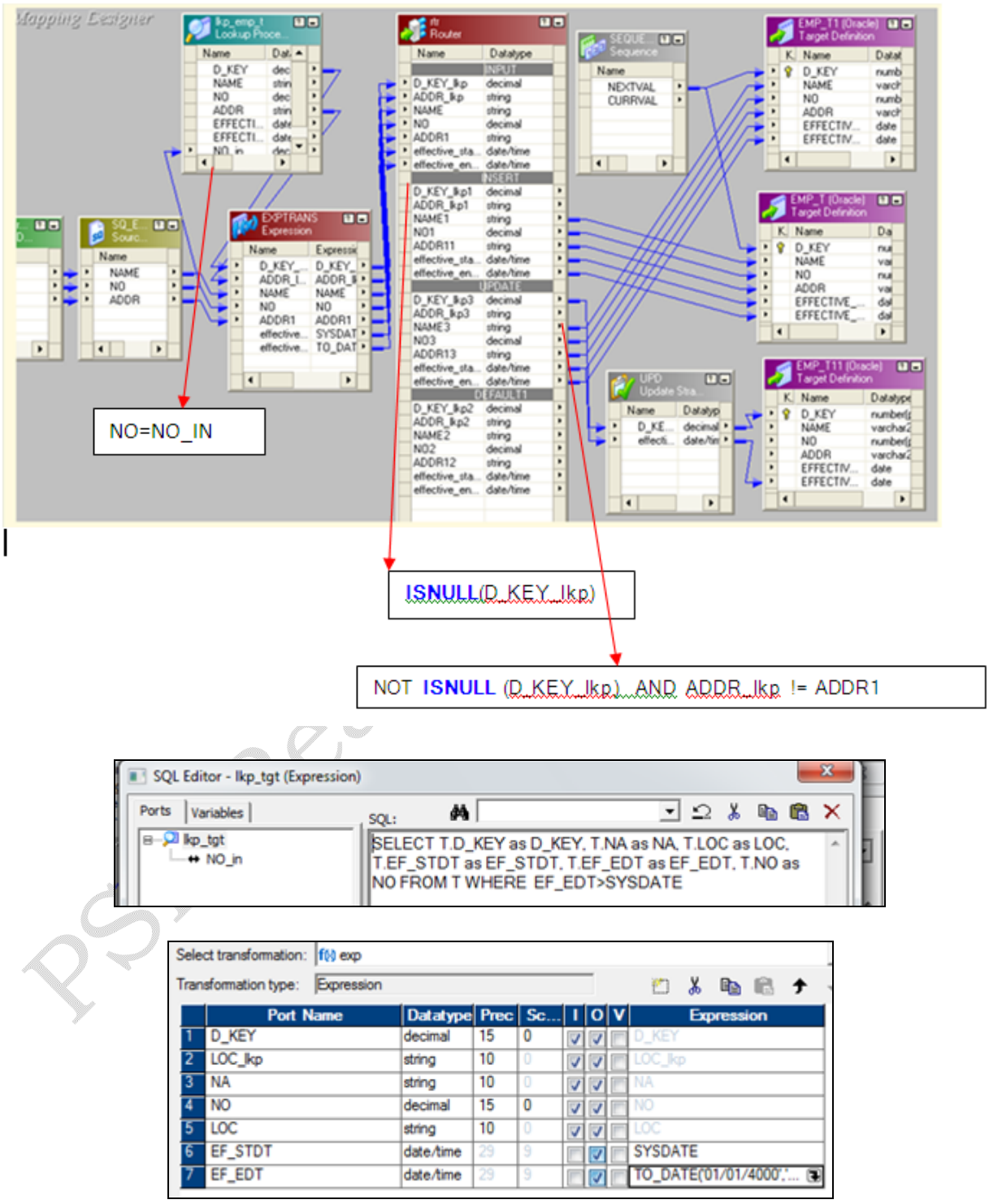


Fig.: The above screenshot shows how to implement scd type 2 Effective Date Approach

Steps:

1. In target table we need three extra columns like D_KEY, Effective_Start_Date, Effective_End_date.
2. If source record doesn't exist in target then we need insert into target,Dimension key with surrogate key ,effective_stdt with system date and effective_Eddt with the future date
3. If source record already present ,then compare source SCD columns with target SCD columns.

   If there is any change in any of the column then we need to do two tasks

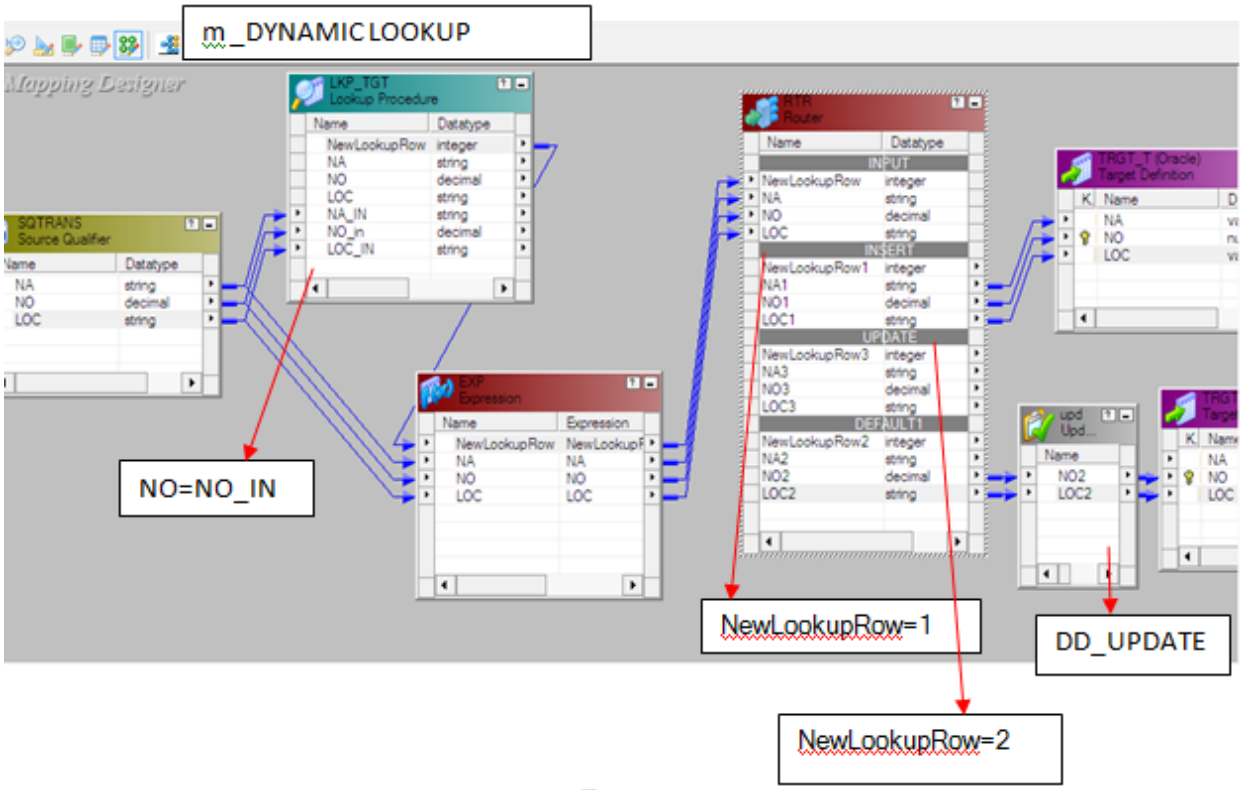
   Task 1: Insert this as New record

   Task2: Update previous existing active record end date with the sysdate
4. In Router we need create two groups one for Newly insert records and second one for insert and Update(Previous record effective end date with sysdate)
5. Lookup override write a query to fetch only active records into Cache

Use space for running notes:

SCENARIO

Requirement: ***How to perform update else insert between source and target if source contains duplicate data***



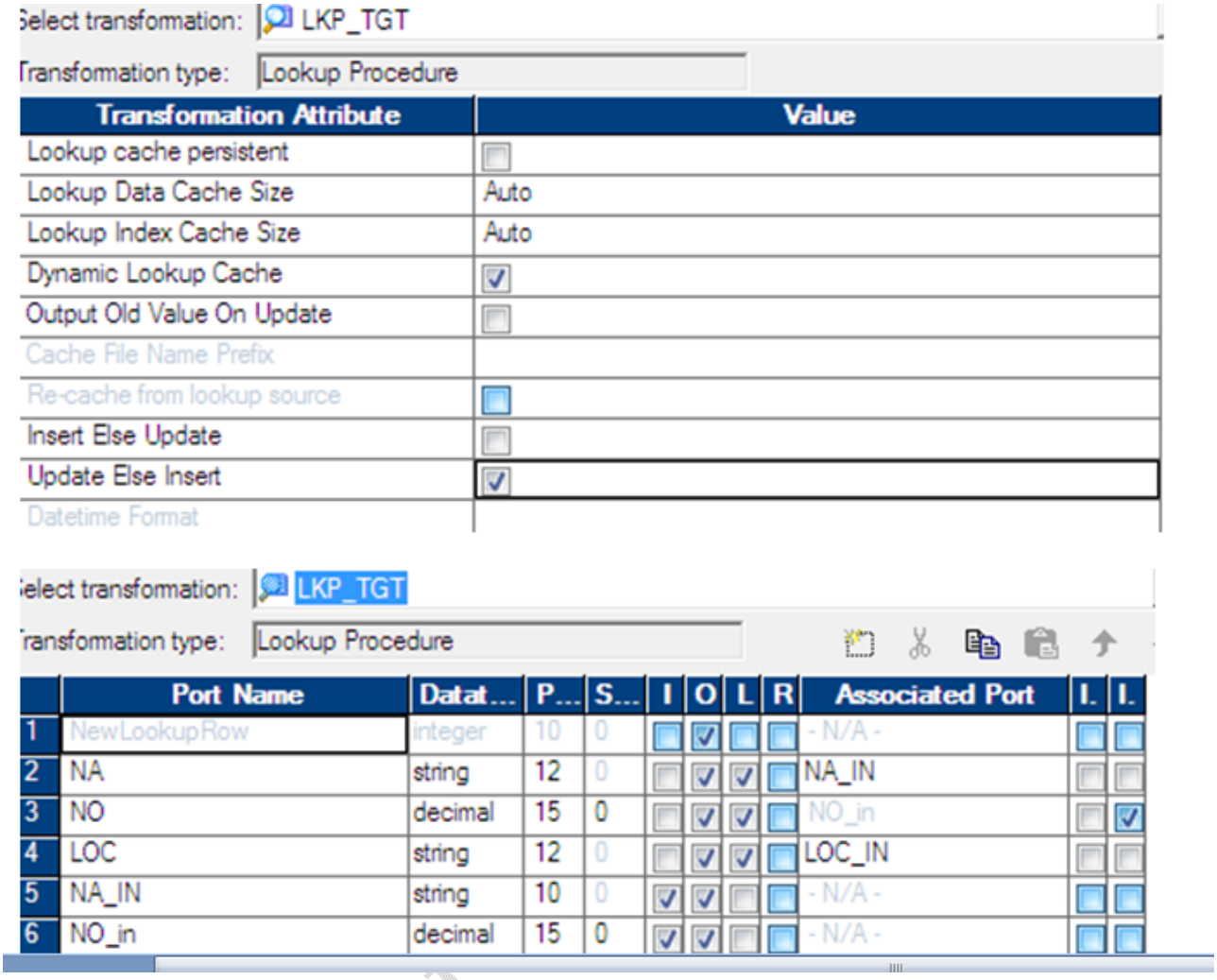


Fig.: The above screenshot shows how to use a dynamic lookup

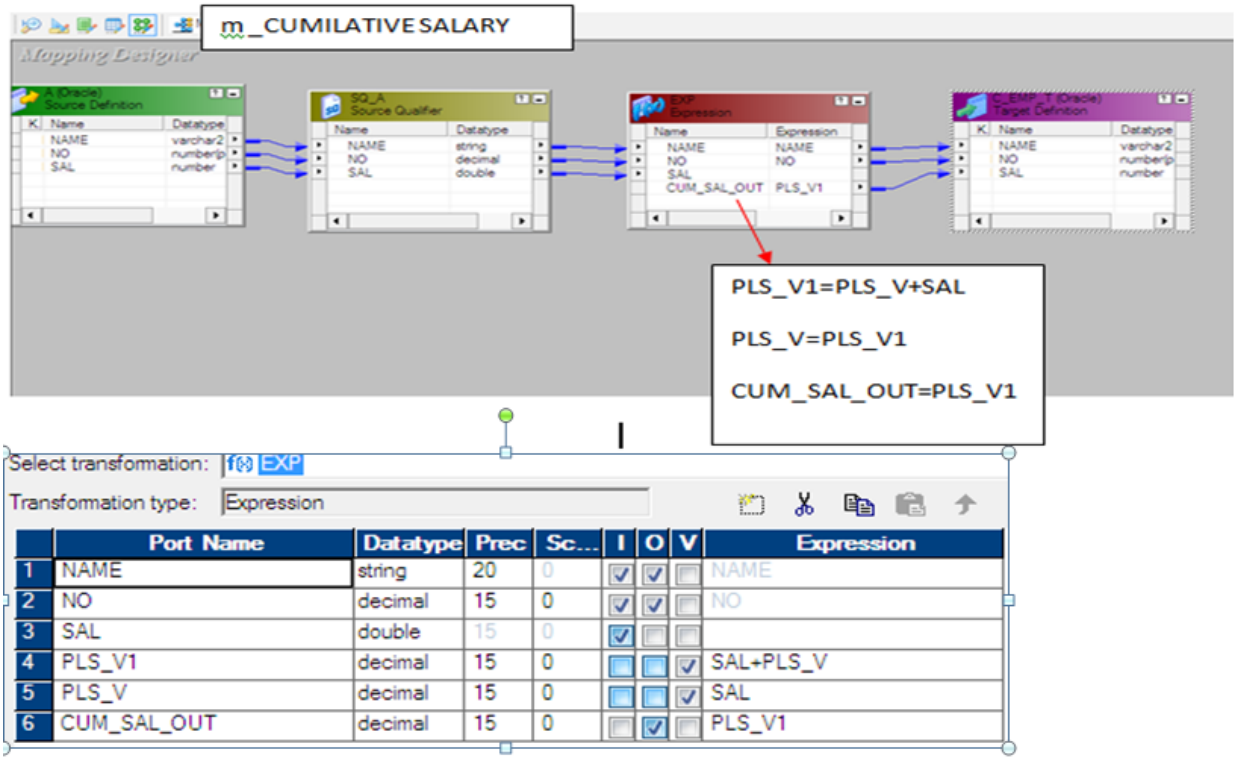
Steps:

1. If source contains duplicate data to perform update else insert we should use dynamic cache
2. If we use a static lookup ,for example in source we have first and last record are same but there is a change Loc column
3. For the first record lookup search in the cache since it doesn't exist so it will send it for insert.Same it will process 2nd,3rd and etc.
4. When it start process last record ,it will not find this records in the cache even though it processed in to target as a first record.The reason is static lookup will not get refreshed as and when lookup table get refreshed.
5. So this record also send it for insert,it suppose to go for update

Use space for running notes:

SCENARIO

Requirement: ***How to add a previous record salary while processing to the target table***



	Port Name	Datatype	Prec	Sc...	I	O	V	Expression
1	NAME	string	20	0	☑	☑	☐	NAME
2	NO	decimal	15	0	☑	☑	☐	NO
3	SAL	double	15	0	☑	☐	☐	
4	PLS_V1	decimal	15	0	☐	☐	☑	SAL+PLS_V
5	PLS_V	decimal	15	0	☐	☐	☑	SAL
6	CUM_SAL_OUT	decimal	15	0	☐	☑	☐	PLS_V1

Fig.: The above screenshot shows how to calculate the cummulative salary

Steps:

Using Varible ports in the expression transformation we can capture or store previous processed record details

Use space for running notes:

SCENARIO

Variables:

In Informatica we have session level variables and mapping variables

Mapping level variables are two types

- *Varaible type as Variable*
- *Varaible type as Parameter*

Four types of Session Level Variables

- $DBConnection_Source
- $DBConnection_Target
- $InputFile
- $OutputFile

Parameter File : It supplies the values for the session level variable and the mapping level variable.

Requirement: ***How to extract latest data/Changed data Capture/Delta /Incremental Data from source in informatica***

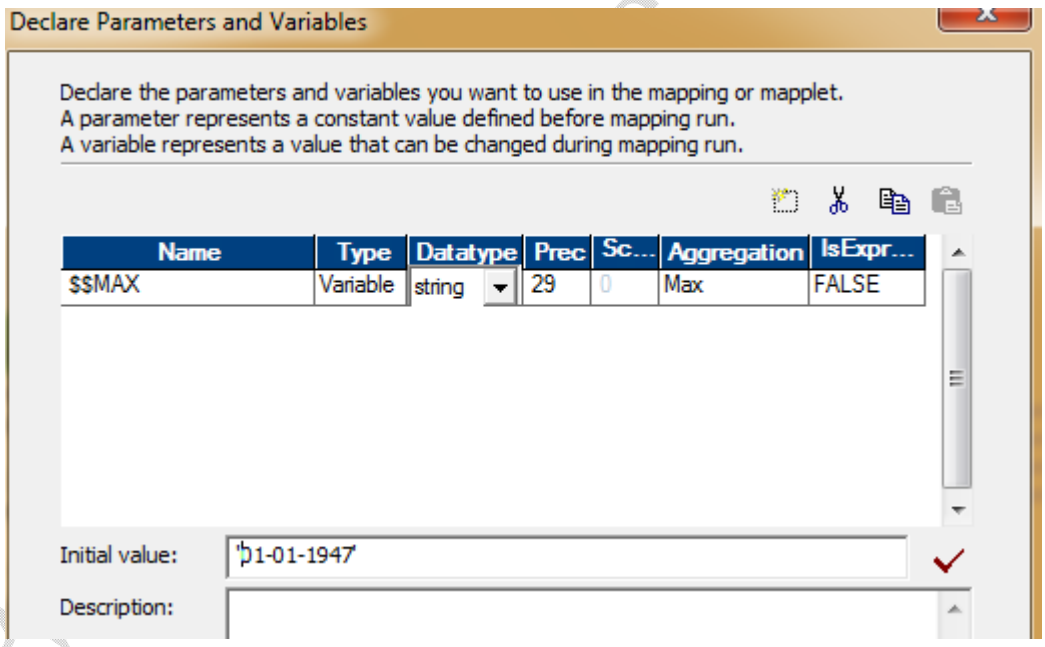


Fig.: The above screenshot shows how to create a mapping level variable

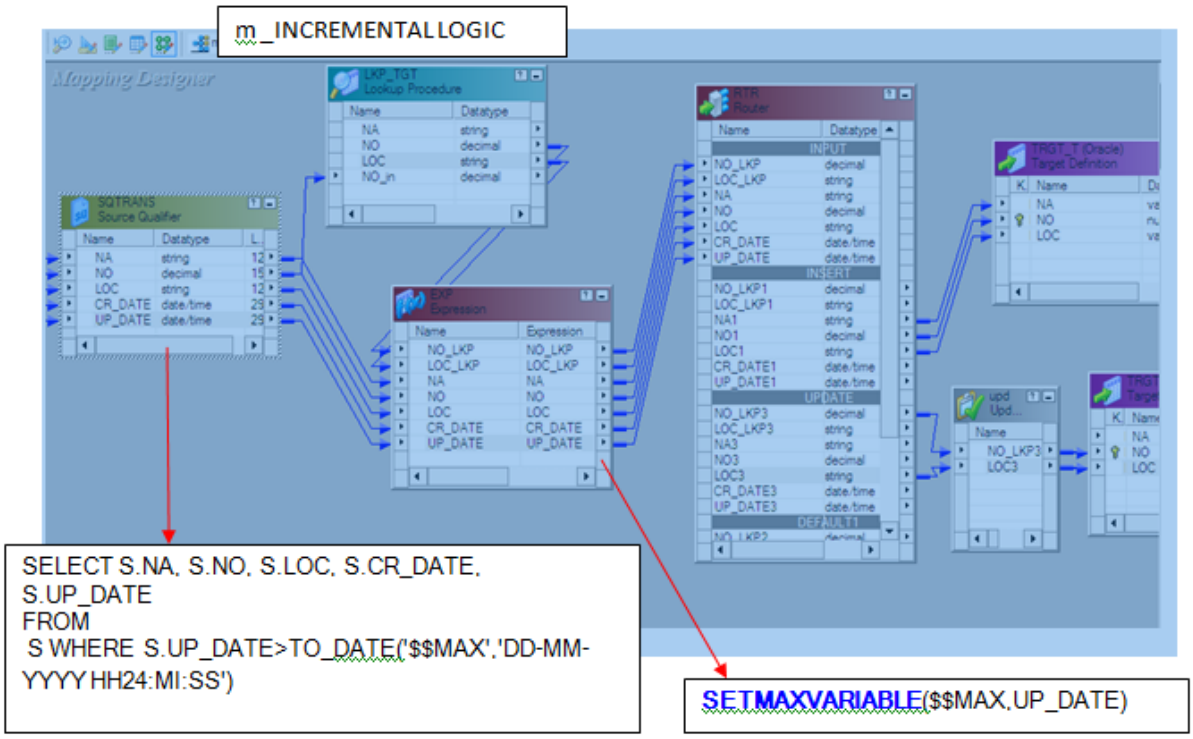


Fig.: The above screenshot shows how to pull the incremental data using setmax variable approach

Steps:

1. We need to Create mapping level variable and assign a old value
2. Write a query in source qualifier query to select data from the source table whose updated date is greater than the mapping variable date
3. While processing the data from source to target capture maximum update date value to the mapping level variable using SETMAX VARIABLE function in the expression transformation

Use space for running notes:

SCENARIO

Requirement: ***Pulling Incremental Data using Control tables Logic***

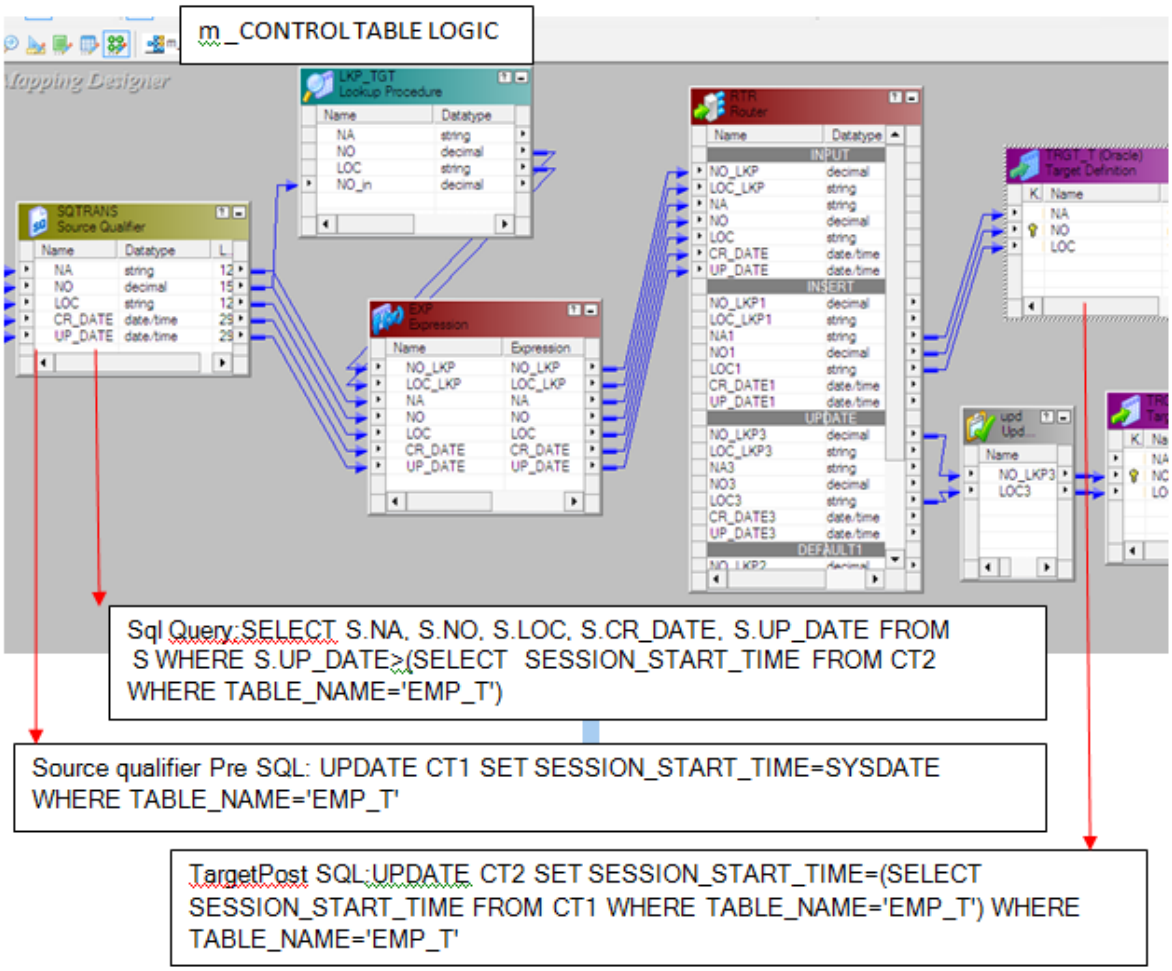


Fig.: The above screenshot shows how to pull the incremental data using control tables

Steps:

1. We need to have two control tables in the database one for to capture session start time and second one for to store session start time after session successful run
2. Update the control table 1 with system date by usinf source qualifier Pre sql query
3. Update the control table 2 with control table 1 session start time using Target Post sql

Use space for running notes:

SCENARIO

Requirement: ***How to control commits through mapings by using Transaction control Transformation***

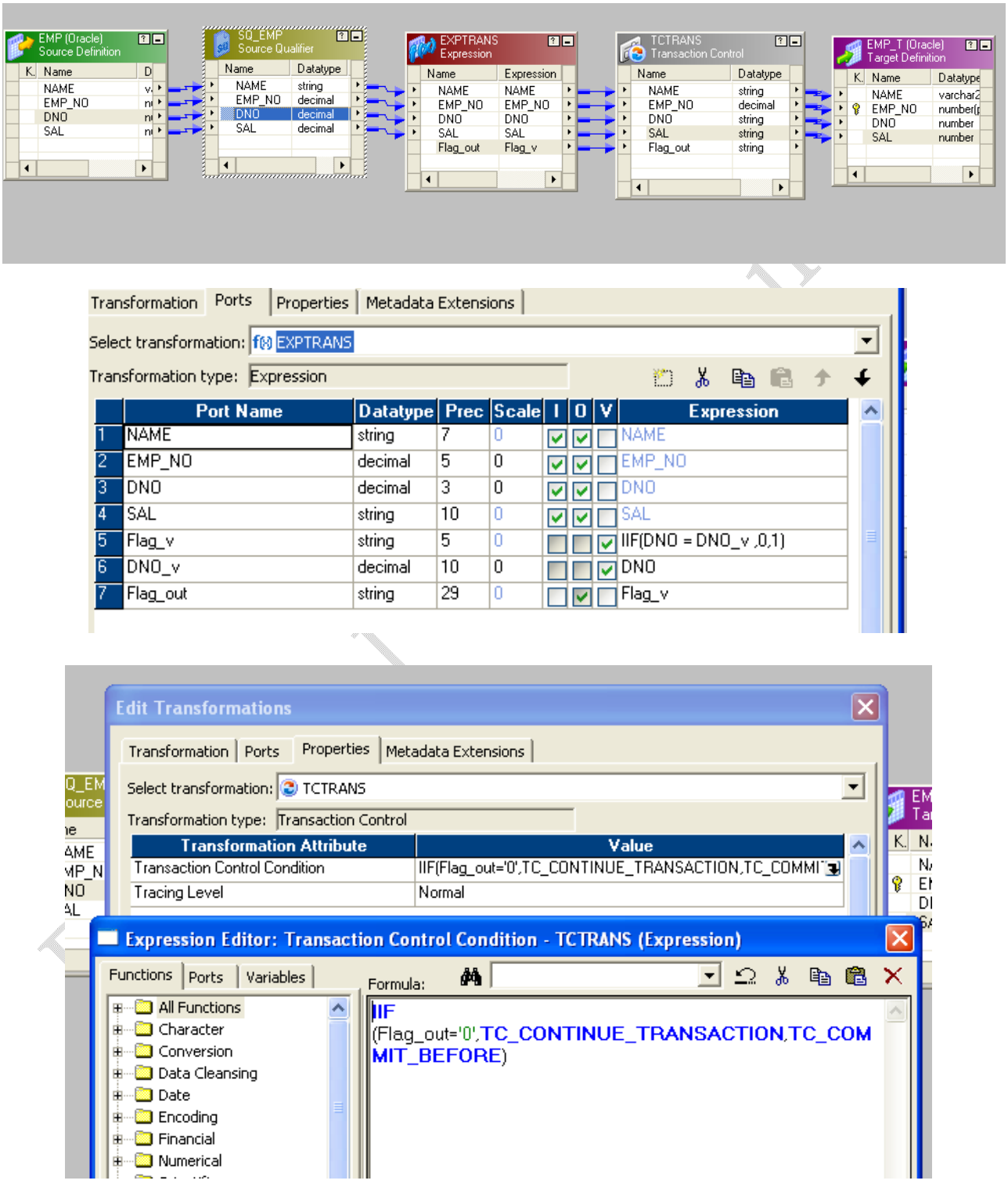


Fig.: The above screenshot shows how to use the transaction control transformation

Uses of Transactional Conrtrol Transformation:

1. Transactional control transformation we use to control the commits through mapping

Use space for running notes:

SCENARIO

Requirement: ***How to generate targegt files Dynamically***

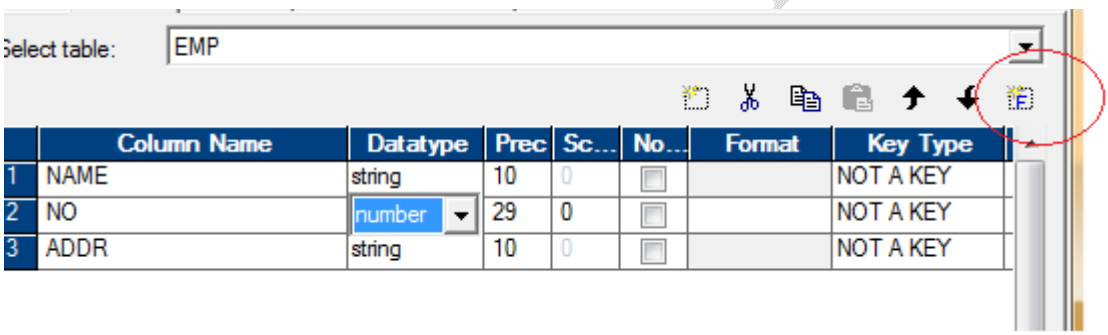
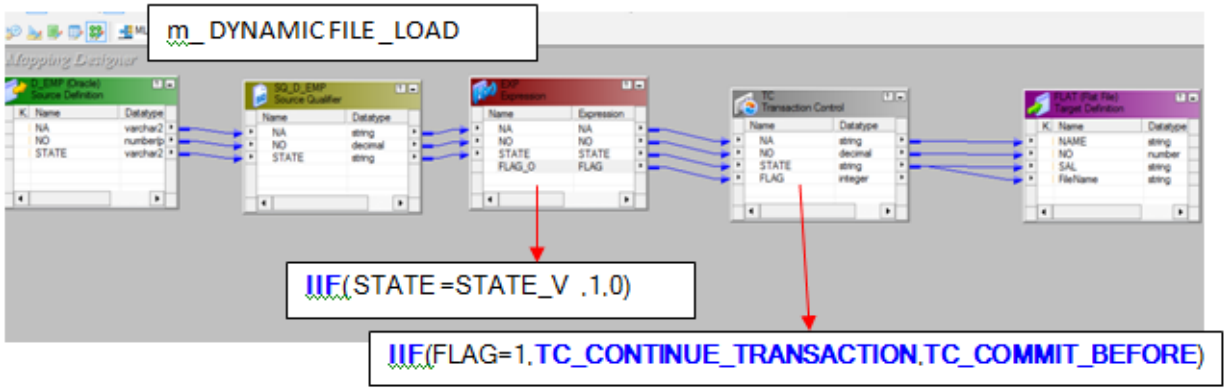


Fig.: The above screenshot shows how to enable F label option in target definition at traget desingner



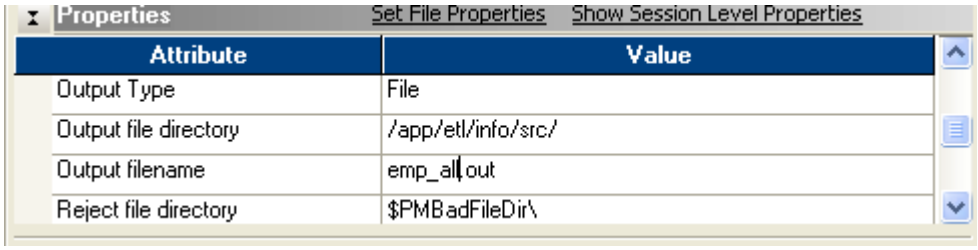


Fig.: The above screenshot shows how to generate the files dynamically

Use default output file name as it is.

Steps:

1. In Informatica 8 onwards we have advanced option (F label) for Target definition at target designer
2. Using this we can generate target files Dynamically
3. Before the target we need to use the transaction control transformation.
4. In transactional control transformation we need to flag either TC CONTINUE TRANSACTION OT TC COMMIT BEFORE based on the flag value
5. Ine expression before transaction control transformation we need to derive the flag value by comparing previous record state with current record state.

Use space for running notes:

SCENARIO

Requirement: ***How to generate sequence numbers without using sequence generator transformation in informatica***

Create one mapping variable and set default value is 0.

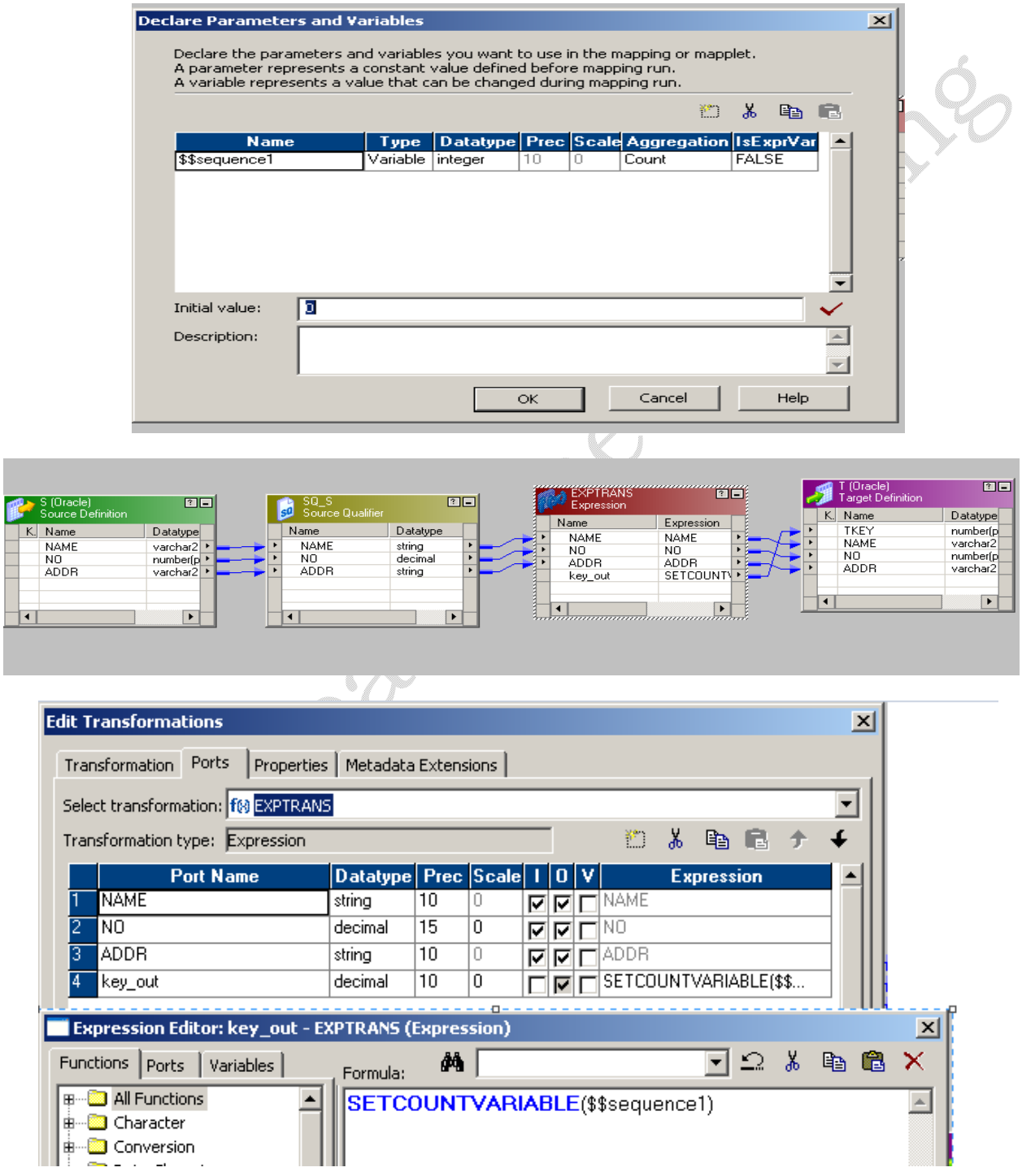


Fig.: The above screenshot shows how to generate the sequence numbers with out using sequence generator

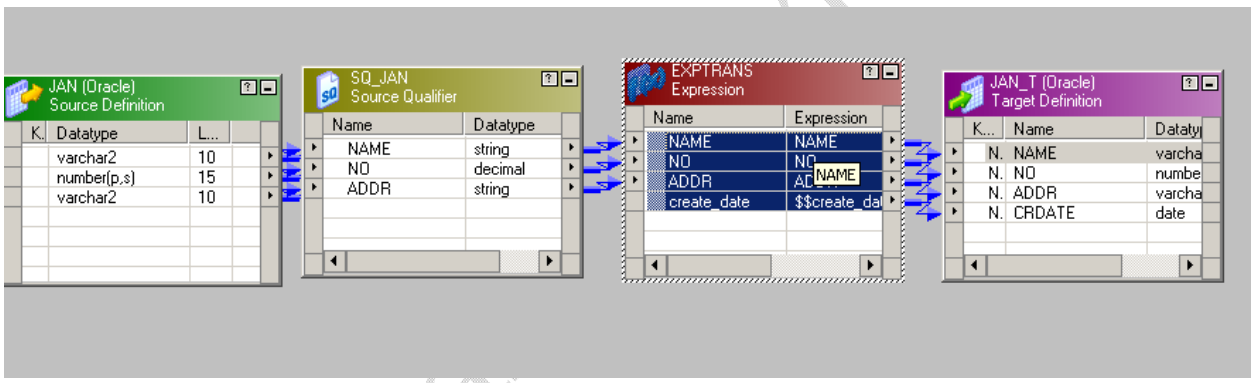
Steps:

1. We need to Create a mapping level variable and assign initial value as 0 and set Aggregation value to Count
2. In Expression using SETCOUNT VARIABLE fuction assign a count to mapping level variable

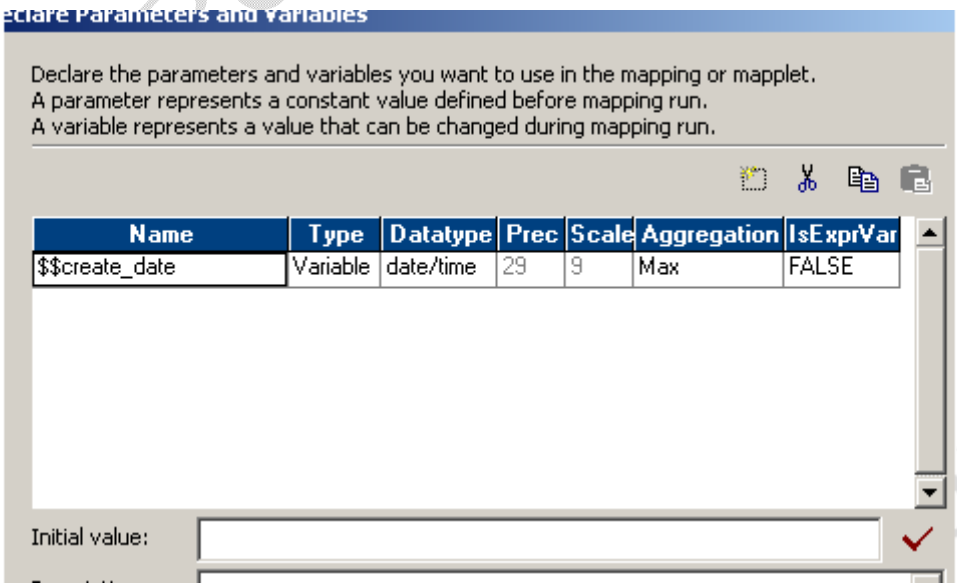
Use space for running notes:

SCENARIO

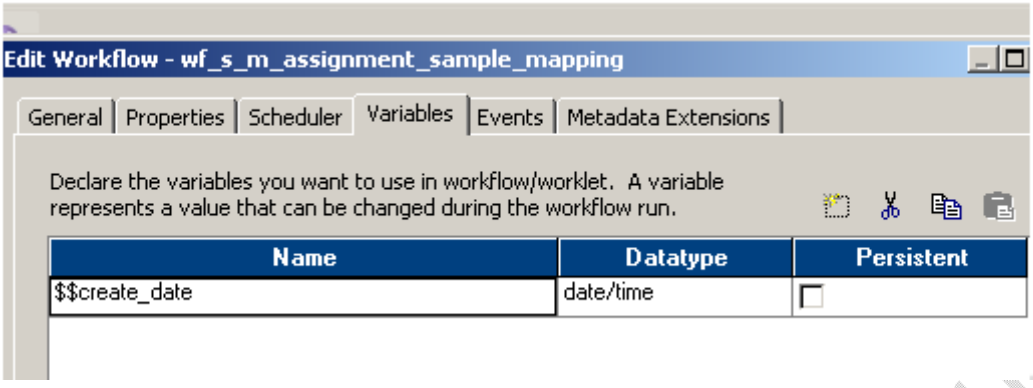
Requirement: ***How to Assign worflow variable to mapping level variable using Assignement task***



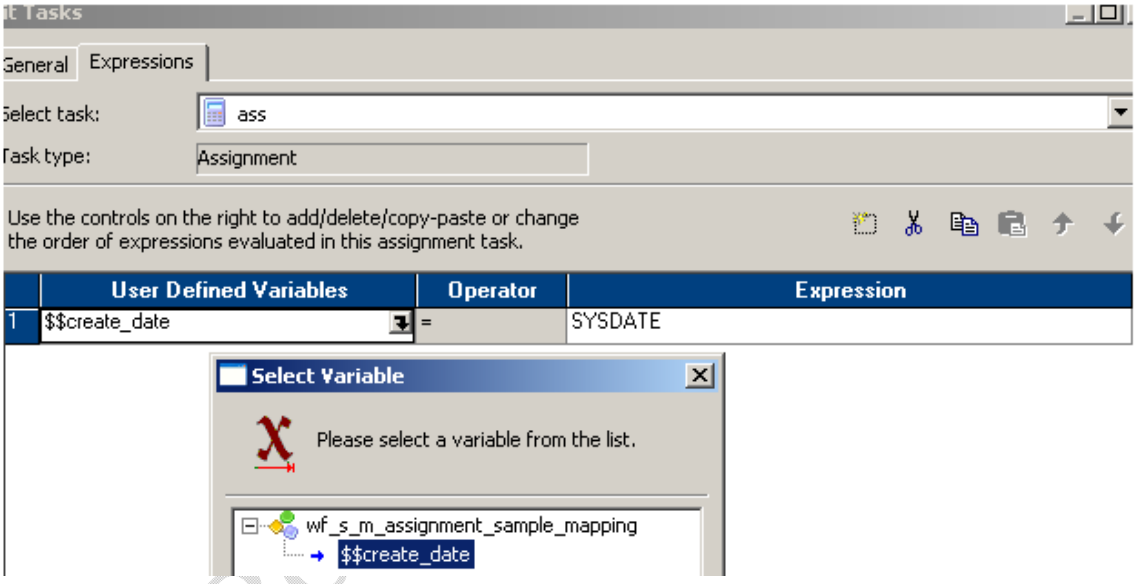
Create mapping level variable without initial value



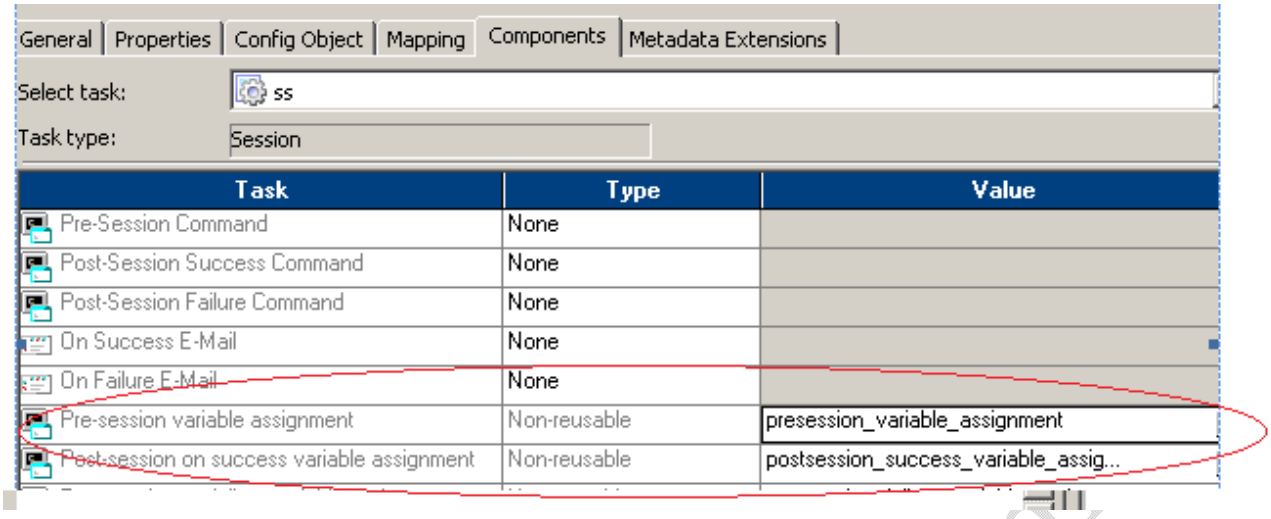
Create workflow level variable as shown below



Create Assignment task to assign a value to the workflow variable as shown below



Using pre-session variable assignment property at session level, assign workflow level variable to mapping level variable as shown below



In the Presession_varaible_assignemnt make them equal mapping variable and workflow variable .

Work flow variable assigns sysdate and these values supplies to mapping variable as intial values

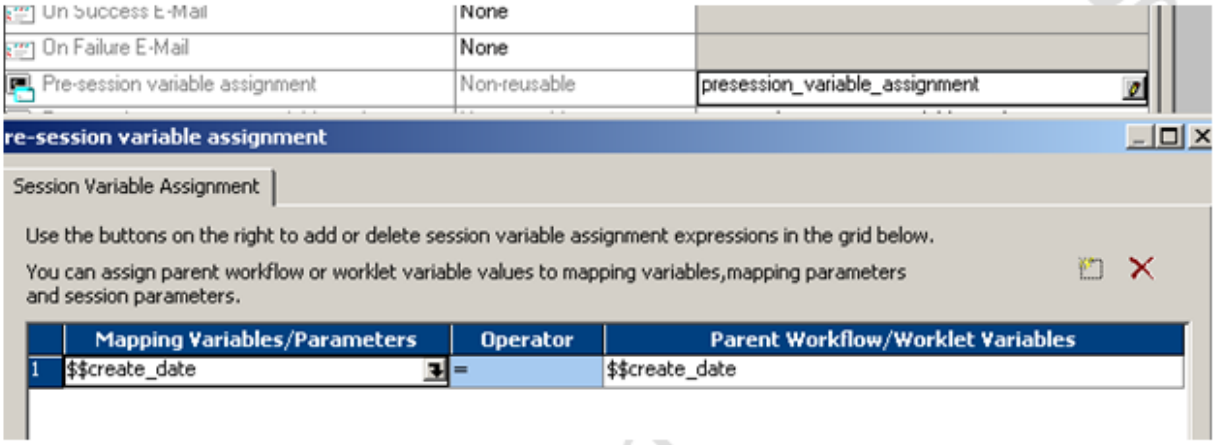


Fig.: The above screenshot shows how to assign a workflow variable to mapping level variable

Use space for running notes:

SCENARIO

Requirement: ***How to do file watch mechanism in Informatica. (or)***

How to run a session after status file/Indicator file available in Informatica server.

Sol:

Using event wait task we can do file watch but Event wait it will wait forever till file available

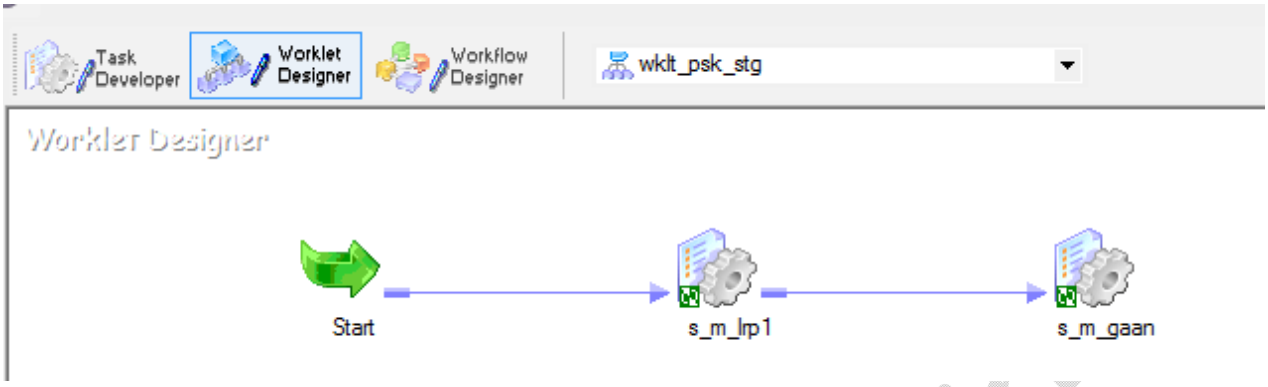
Event-Wait task:

The Event-Wait task waits for an event to occur. Once the event triggers, the Integration Service continues executing the rest of the workflow.

Use space for running notes:

SCENARIO

Requirement: *Working with worklets*



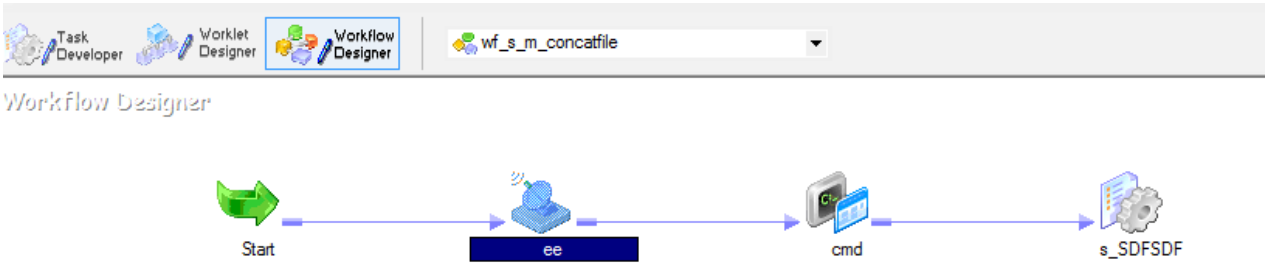
Uses of Worklets:

- Worklet is set of reusable tasks
- To set a dependency between theworkflows rather than creating worlflows we will create a worklet and then will call them in a workflow
- Worklet we cannot run without workflow
- Generally we create one worklet for all stage sessions ,one worket for the dimensions and one worklet for fact session

Use space for running notes:

SCENARIO

Requirement: *Working with Command tasks*



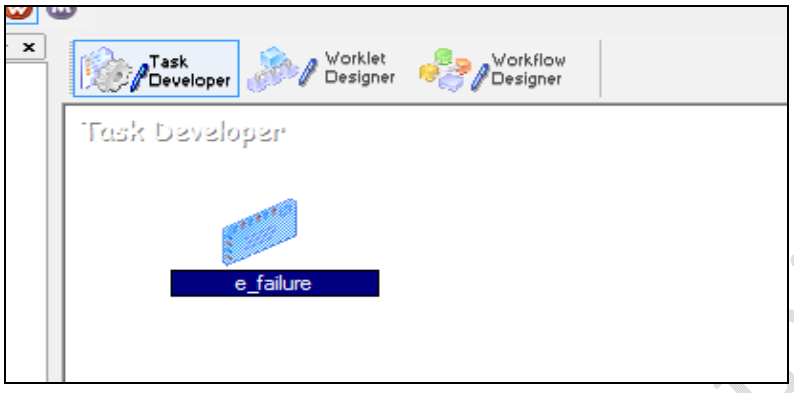
Uses of Command Task:

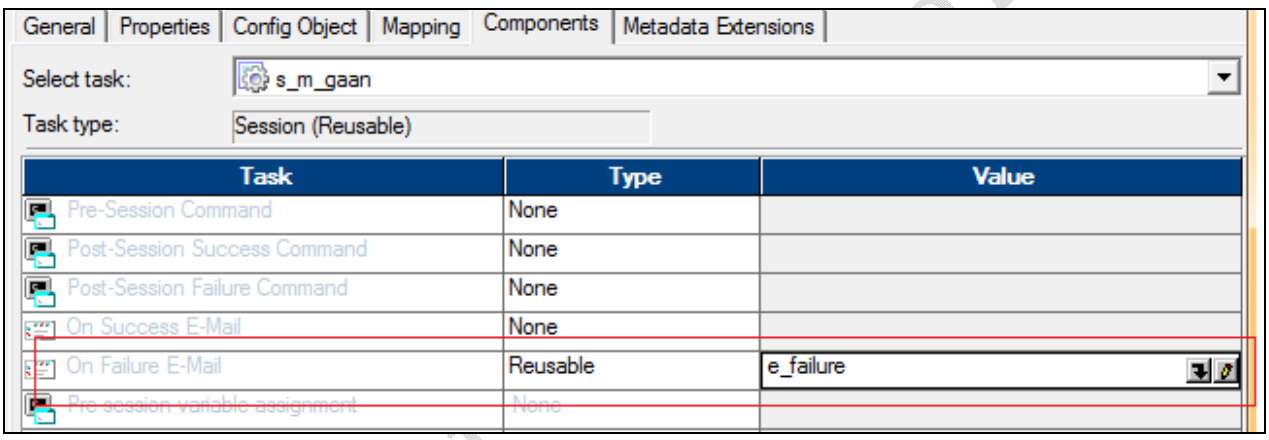
- Command Task is used to run a unix commands like execute the shell scripts.

Use space for running notes

SCENARIO

Requirement: *Working with Email Tasks*





Uses of Email Task:

- Email Task is used to send a failure or success notification emails
- As per above daigram we need to configure reusable email at session level property on failure Email or on Success Email

Use this for Running Notes

SCENARIO

Requirement: ***How to Execute database Stored procedure in Informatica***

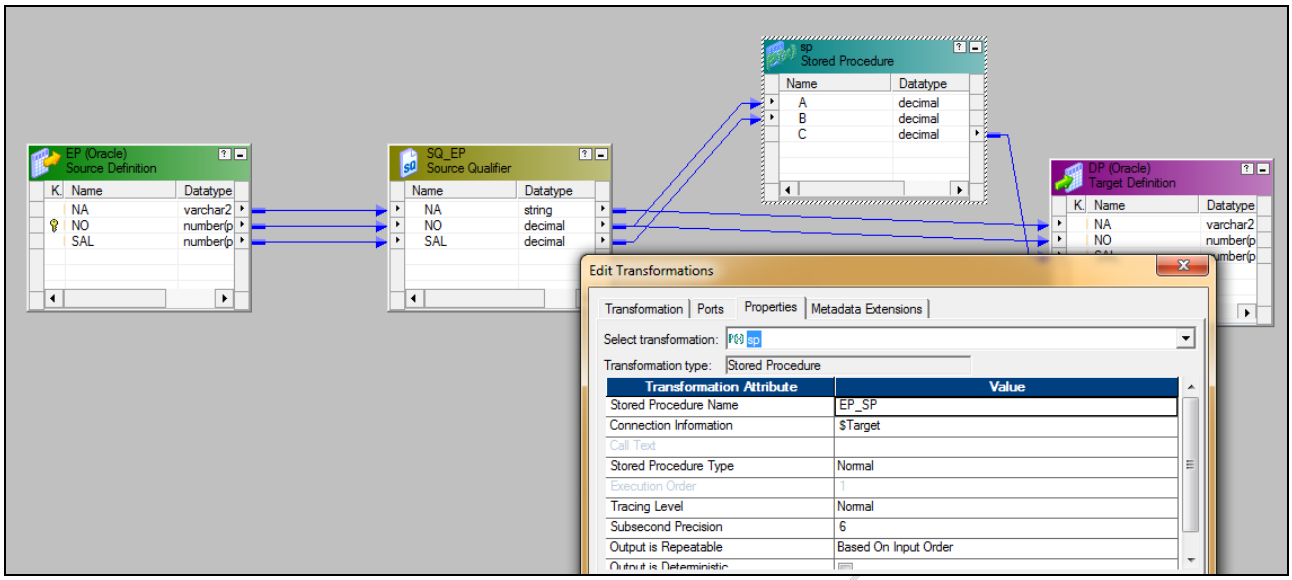


Fig.: The above screenshot shows how to call stored procedure in informatica using stored procedure transformation

Steps:

1. If type is a Normal it execute for every record once
2. If type is other than Normal it executes only once for entire mapping run
3. If Type=Source Pre Load →Stored Procedure will execute before source qualifier extracts
4. If Type=Source Post Load →Stored Procedure will execute after source qualifier extracts
5. If Type=Target Pre Load →Stored Procedure will execute before Target load
6. If Type=Target Post Load →Stored Procedure will execute after Target load
7. If stored procedure is unconnedted we can call in Expression using :SP. But type should be Normal

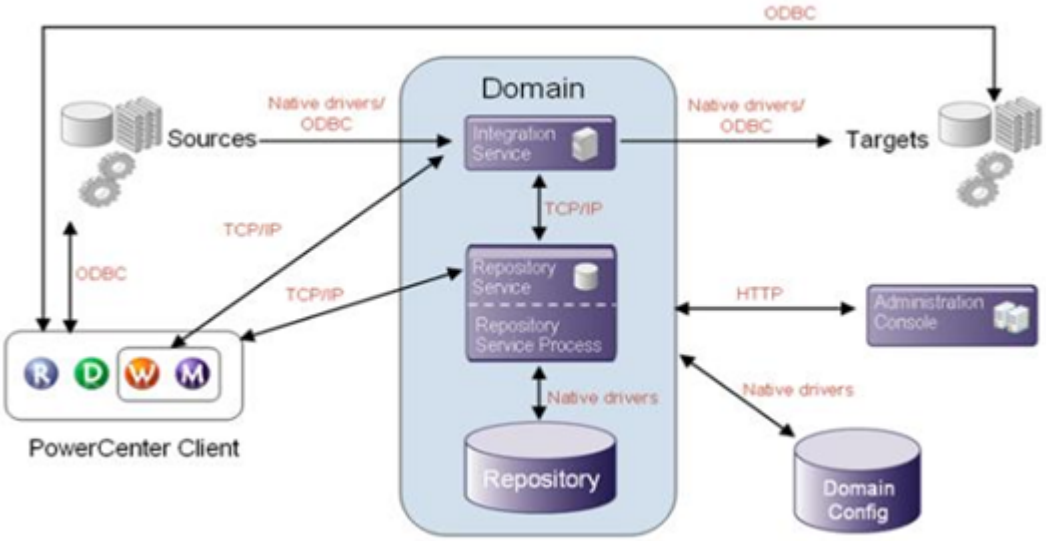
Use space for running notes:

SCENARIO

Requirement: ***Difference between Informatica 7,8 and 9 Versions***

There is lot of difference in Architecture between 7 and 8 higher end versions. Till Informatica 7 version it was client and server Architecture, from 8 onwards they made it SOA (*Service oriented Architecture*).

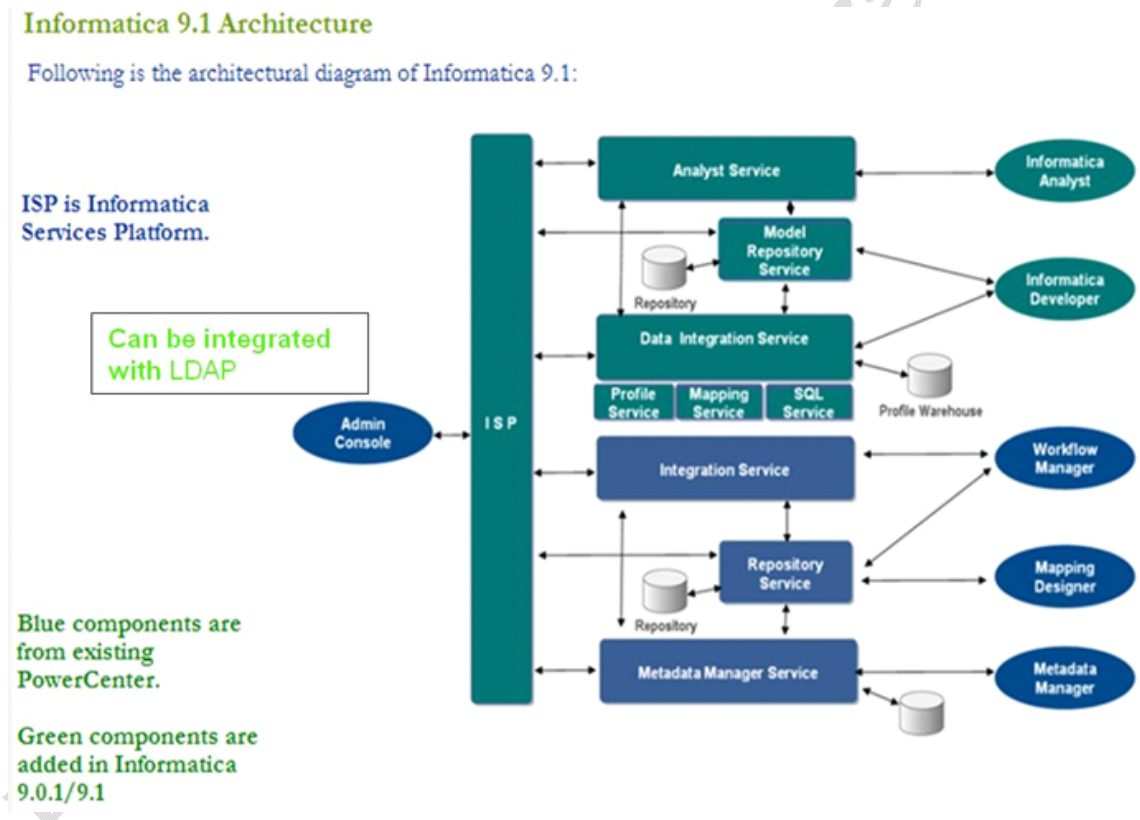
- In 7 when client request to server to start workflow ,Informatica server will start DTM engine. Below are the task performed by DTM
  - Retrieves and validates session information from the repository.
  - Validates source and target code pages.
  - Verifies connection object permissions.
  - Creates the session log.



- Where as in Informatica 8 when client request to Integration service to start the workflow Then IS will start ISP(informatica Services Platform)
- ISP will consult Load Balancer to check available node on a domain
- The node which choosen by load balance ,ISP will run DTM engine on that particular node
- Developer Perspective they introduced new options like
- In Informatica 8 we have advanced option to append the data to the existing target file
- We can populate header and footer option in Informatica 8 using target file option in session level

- We can read current processing file name in Informatica 8 using advanced option of source definition Add current processing file name port
- Grid concept is additional feature
- Java and sqltransformation is added
- They made autocache options in 8
- push down optimization techniques
- Power Exchange plugins for SAP, Webmethods,Mainframes and SFDC they have introduced in 8 onwards.
- Introduced IDQ in 8 :The **Informatica Data Quality** Solution delivers pervasive data quality to all stakeholders

Informatica 9.x.



They introduced integrated admin console frame work for all newly added plugins where as in 8 we have separate admin console for each plug in like SFDC one separate admin console SAP one separate admin console etc.

- Added informatica developer and analyst components.

  Informatica Developer:

  This tool is eclipse-based and supports both data integration and data quality for enhanced productivity.

Informatica Analyst:

This is a browser-based tool for analysts, stewards and line of business managers. This tool supports data profiling, specifying and validating rules, and monitoring data quality.

- In 9 they have added new value for lookup policy on multiple match **Use All Values,** From 9 version LKP is Active.
- We can write lookup query override for un cached lookup in informatica 9.x.

  Restrictions:
  - We cannot return multiple rows from an unconnected Lookup transformation
  - We cannot enable dynamic cache for Active Lookup transformation.
- **Database deadlock resilience:** In 9.x the session will not fail for database deadlock during lookup.
- **Connection management:** Database connections are centralized in the domain. We can create, view, edit, and grant permissions on database connections in Informatica Administrator.

Use space for running notes:

SCENARIO

Requirement: ***Incrementl Logic using parameter file logic***

Steps:

1. First need to create mapping parameter ($$Pre_sess_start_tmst)and assign initial value as old date (01/01/1940) in the parameterfile.

2. Then override source qualifier query to fetch only LAT_UPD_DATE >= $$Pre_sess_start_tmst (Mapping var)

3. Update mapping parameter($$Pre_sess_start_tmst) values in the parameter file using shell script or another mapping after first session get completed successfully

4. Because its mapping parameter so every time we need to update the value in the parameter file after comptetion of main session.

Below is the *parameter file* format Parameter file:

[GEHC_APO_DEV.WF:w_GEHC_APO_WEEKLY_HIST_LOAD.WT:wl_GEHC_APO_WEEKL_HIST_B

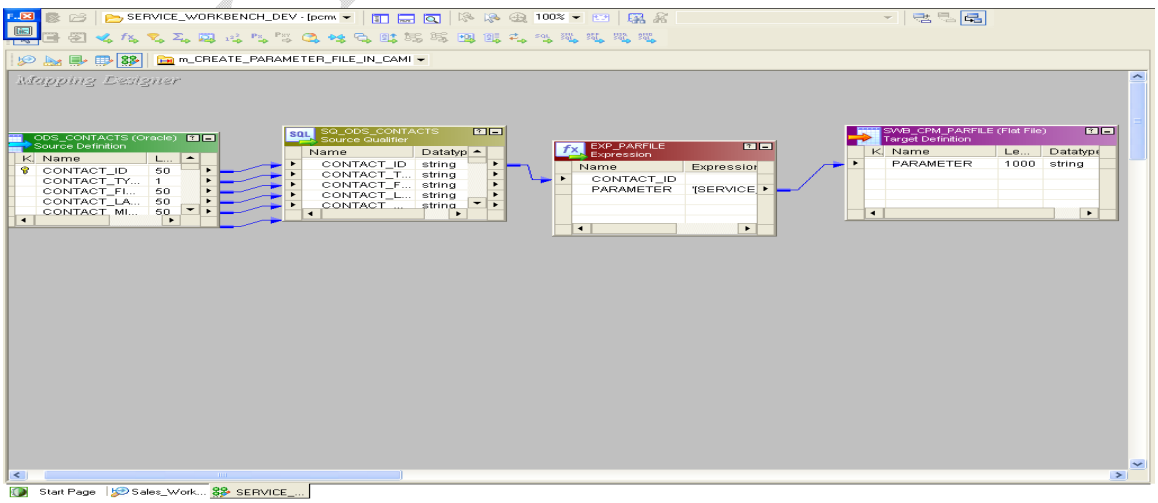
AAN.ST:s_m_GEHC_APO_BAAN_SALES_HIST_AUSTRI]

$DBConnection_Source=DMD2_GEMS_ETL

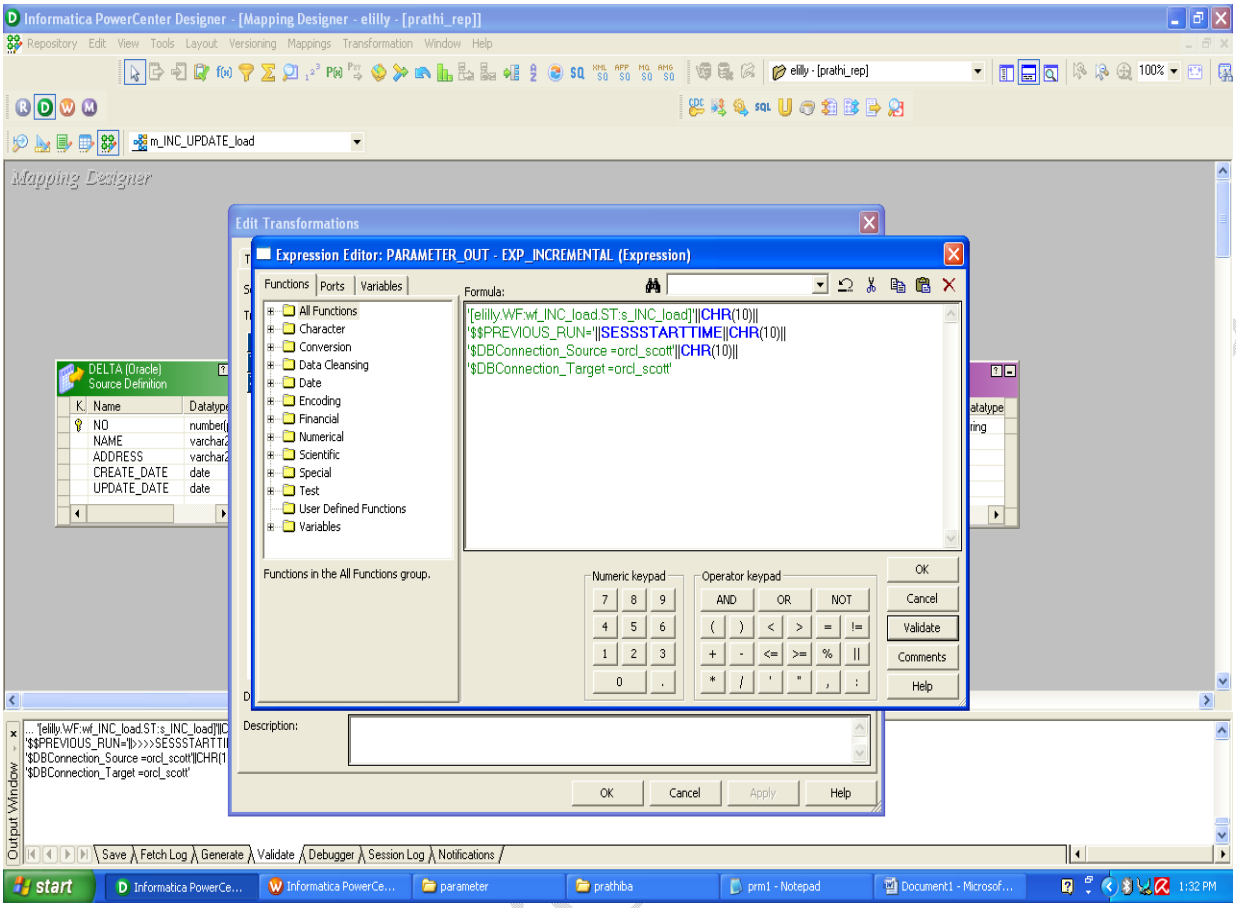
$DBConnection_Target=DMD2_GEMS_ETL

$$LastUpdateDate Time =01/01/1940

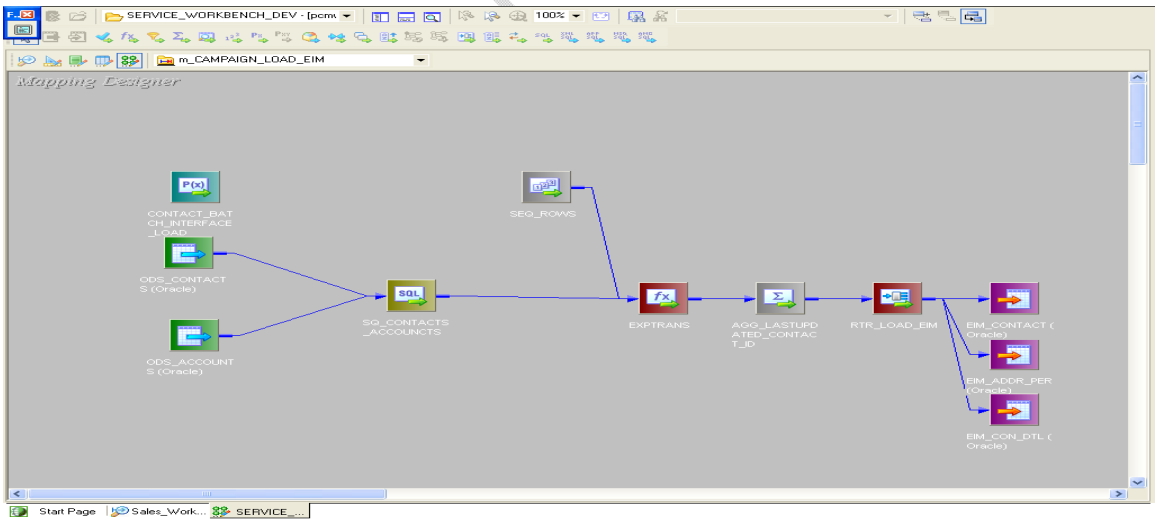
Updating parameter File



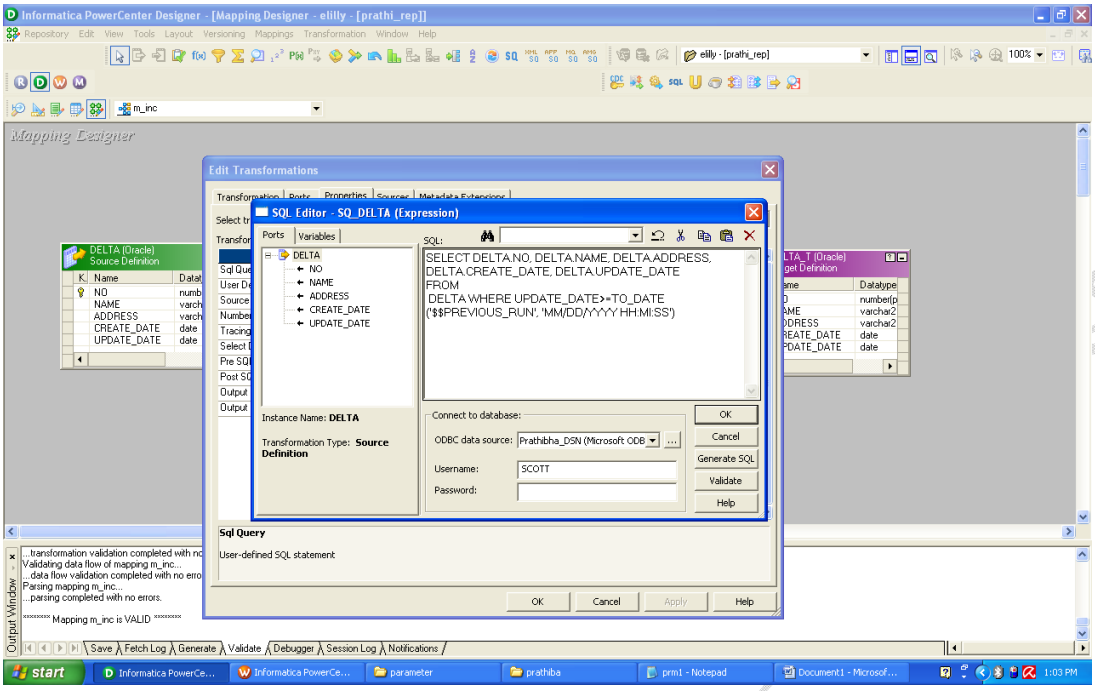
Logic in the expression



Main mapping



SQL override in SQ Transformation



Workflow Design

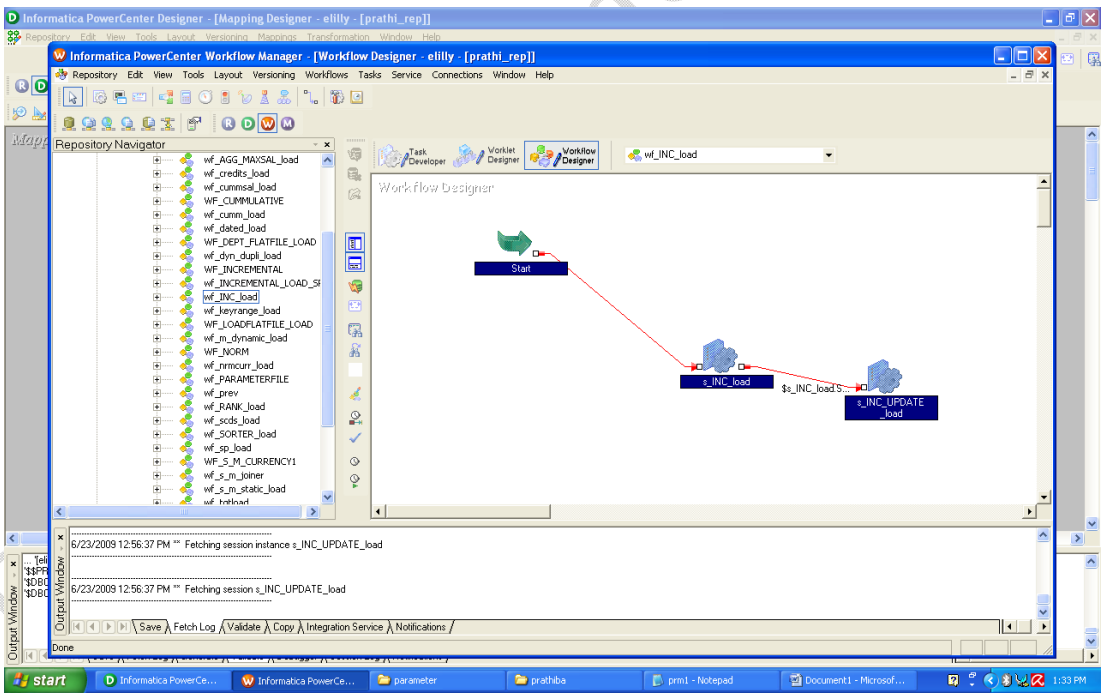


Fig.: The above screenshots shows how to use a parameter file to pull the incremental data

Use space for running notes:

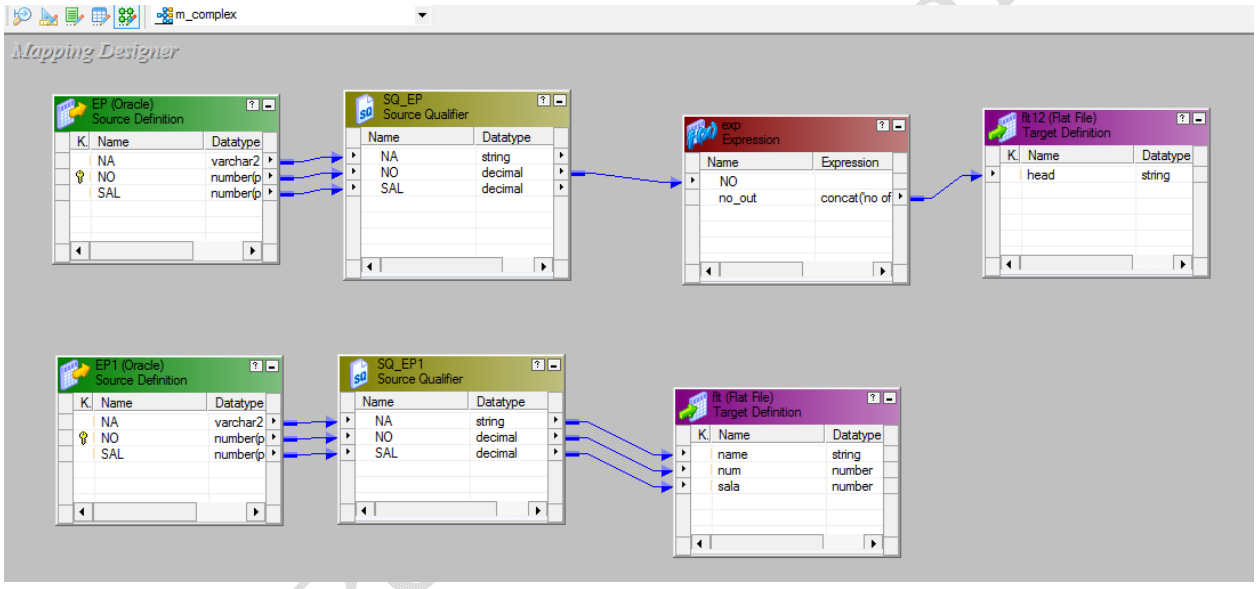
SCENARIO

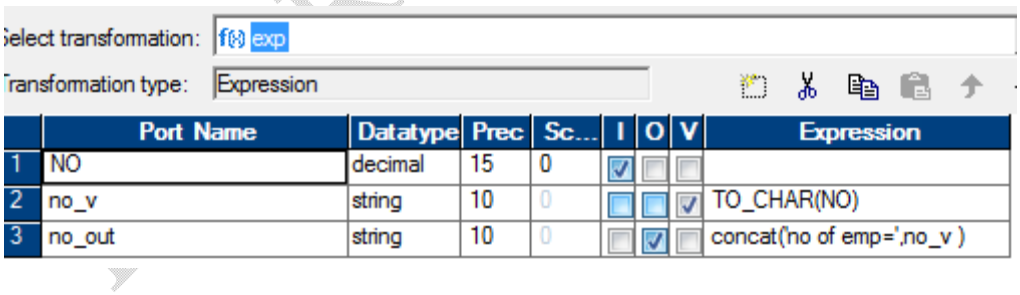
Requirement: ***Explain one complex scenario.***

Recently I faced one complicates scenario to populate target file header with count of the rows which we are going to load to target file like

Psk.txt

No. of loaded records = 3
P,100,Hyd
S, 200, Bang
M,300, Chn





Use above concat logic in the first pipeline expression

Below is the query we need to use in the first source qualifier to get the count

Select count(*) from emp

Steps:

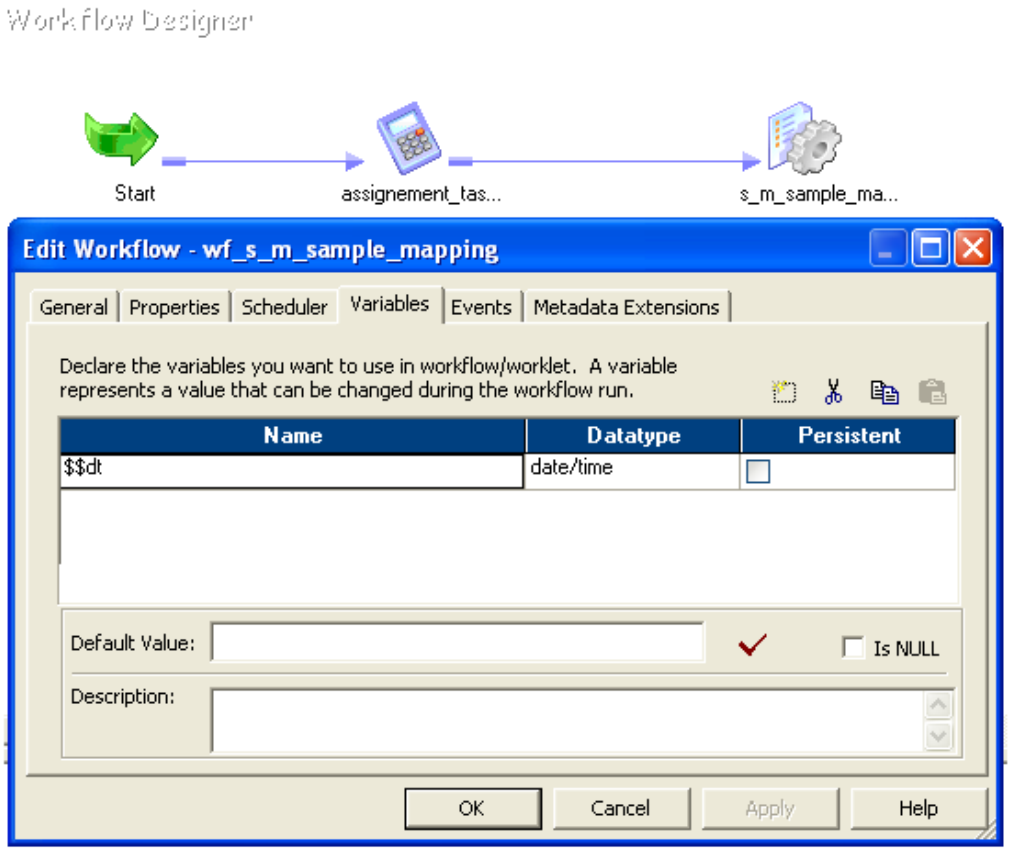
1. We need to create two pipelines first pipeline populate target file message with record count
2. In second pipeline populate target file with actual data
3. Set second target file session property Append if exists
4. Set target load plan to load first pipeline and second pipeline second

Use space for running notes

SCENARIO

Requirement: ***How to assign value to workflow level variable – using Assignment task.***

Goto Workflow → Edit → Variables → Create



No need to specify the default value.

Double Click on the Assignment Task → Go to Expressions Tab → Add a new expression

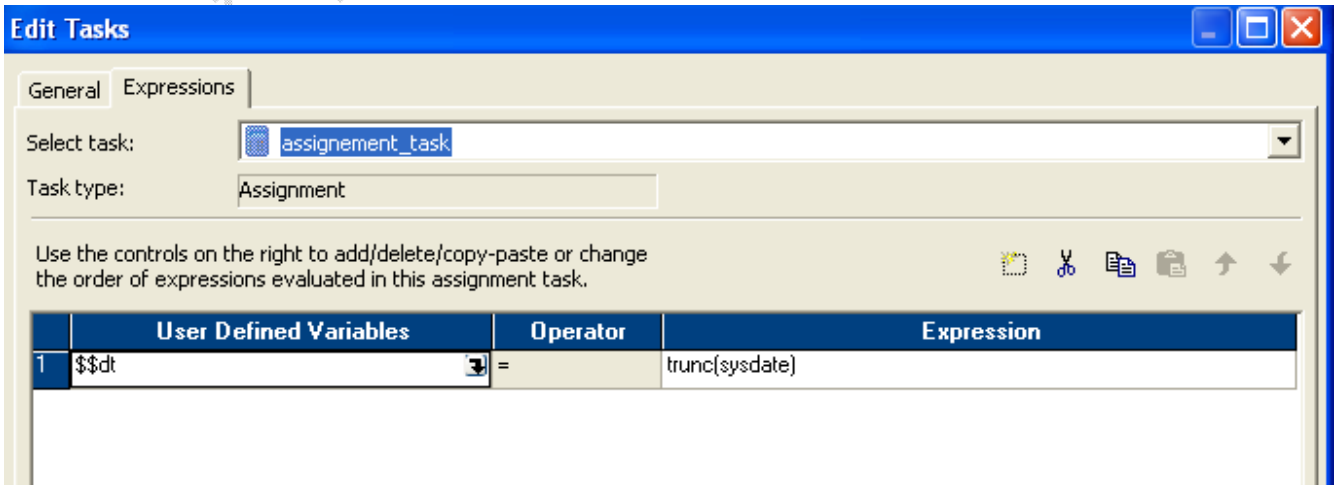
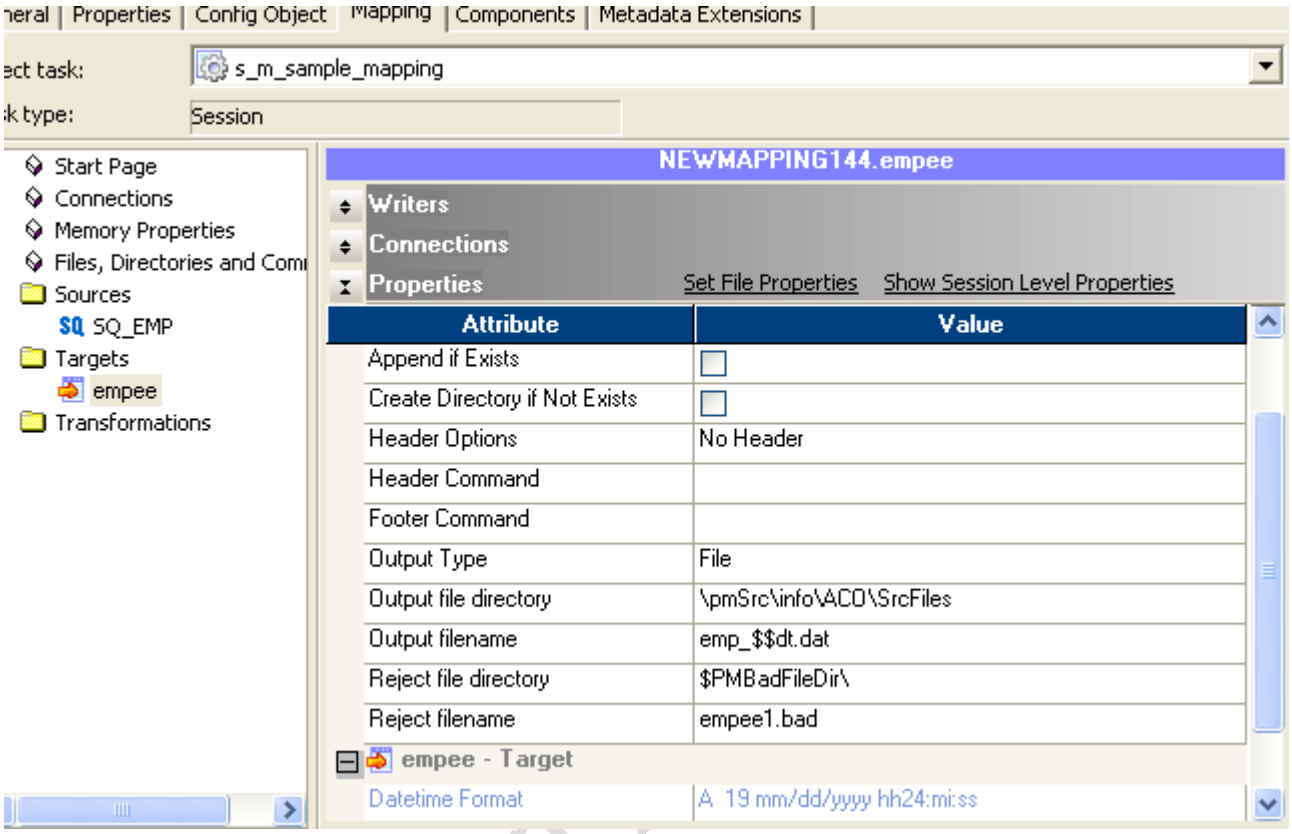


Fig. The above screenshot represents assigning a value to workflow level variable by using Assignment Task

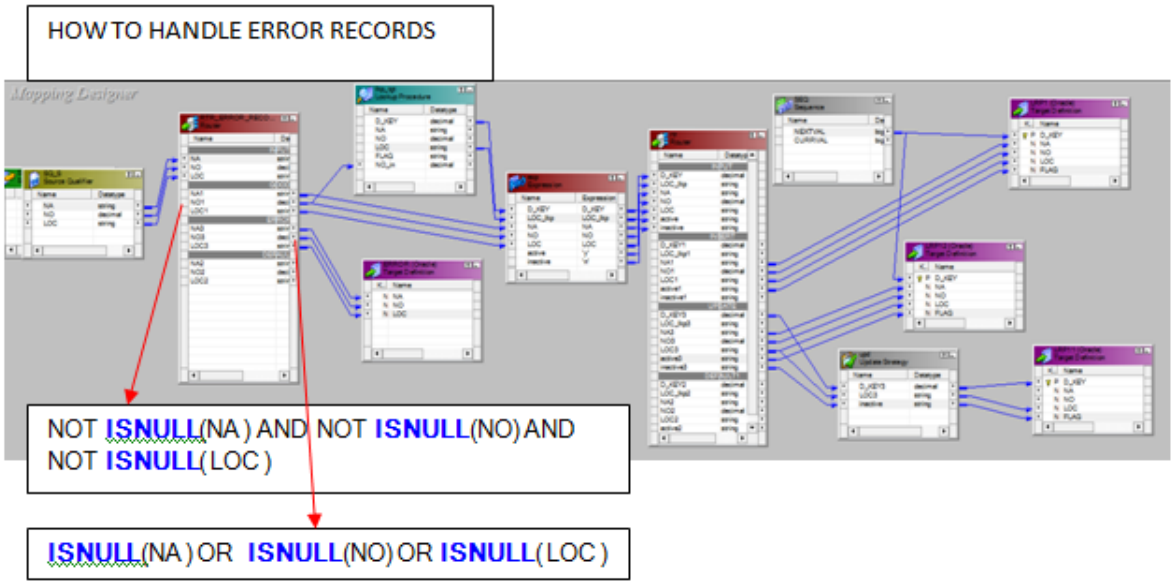
The below screenshot shows, how to generate the target file with the date appended to the filename.



Use space for running notes

SCENARIO

Requirement: ***How to handle the error data in Informatica***



Group Name	Group Filter Condition
GOOD	NOT ISNULL(NA) AND NOT ISNULL(NO) AND NOT IS...
ERROR	ISNULL(NA) OR ISNULL(NO) OR ISNULL(LOC)
DEFAULT1	

Sol:

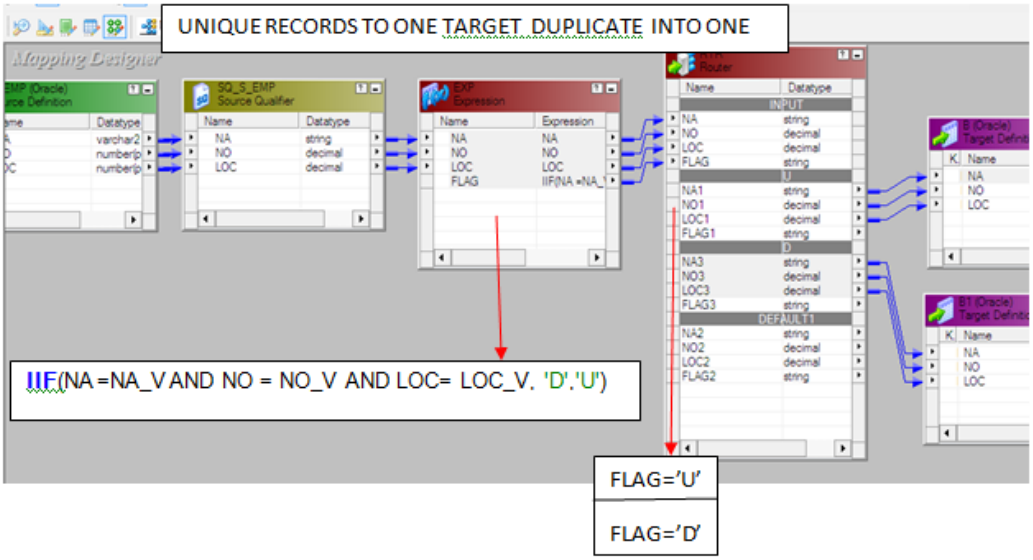
We will process error records in one target file and good records in to target file

So after execution of the mapping we will check error file size through unix, if file size is greater than 0 then we will send email to source system team with error file attachment

Use space for running notes

SCENARIO

Requirement: ***How to load unique records in to one table and duplicate records in to other target table***



Sol:

Approach – 1

- Using Variable ports we will compare previous record and current record
- Based on the comparison we will derive the flag as a U or D
- If flag=D,then will send it to Duplicate target table
- If flag=U, then we will send it to Unique target table
- Make sure to sort the data before expression

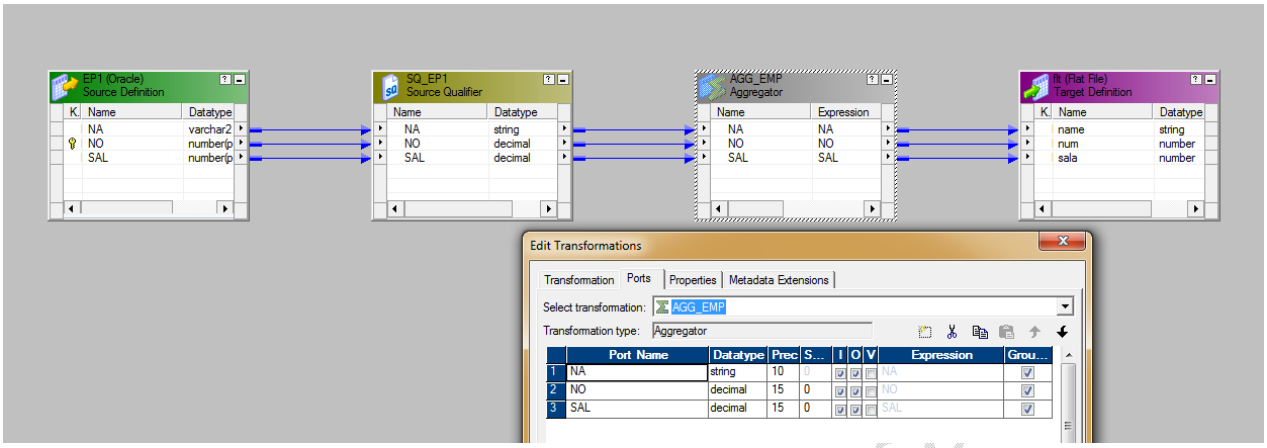
Approach – 2

- Using dynamic lookup on unique table we can separate duplicate in to one table and unique in to one table

Use space for running notes

SCENARIO

Requirement: ***How to eliminate duplicate records while processing from flat file to target table***



Sol:

Approach-1

- If source is relational we can select distinct option in a source qualifier
- If source is a flat file we can not do in source qualifier,so either we need to use aggregator with group by all columns as shown above

Approach-2

- If source is a flat file to eliminate duplicate records we can go with variable port approach

Approach-3

- If source is a flat file to eliminate duplicate records by using sorter transformation with distinct option

Approach-4

- If source is a flat file to eliminate duplicate records we can use a dynamic lookup on target table

Use space for running notes

SCENARIO

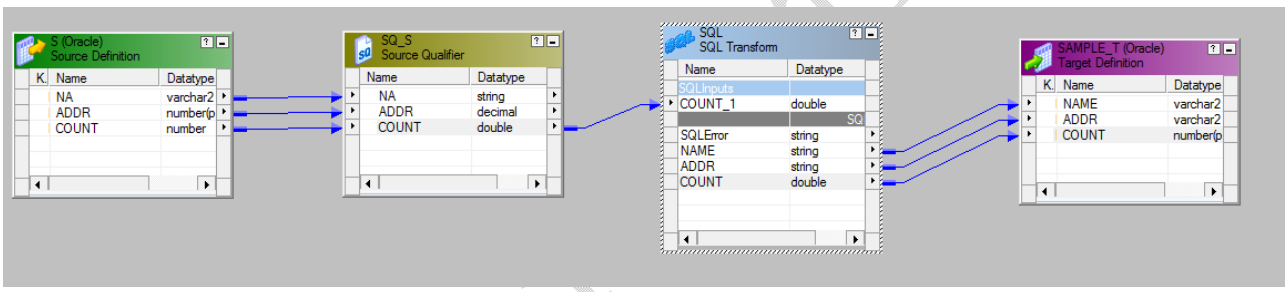
Requirement: ***How to generate multiple records in target based on source column value by using SQL Transformation***

Source table

Name	Address	Count
Prabhakar	Hyd	2
Shyam	bang	3

Output of Target able

Name	Address	Count
Prabhakar	Hyd	2
Prabhakar	Hyd	2
Shyam	Bang	3
Shyam	Bang	3
Shyam	Bang	3



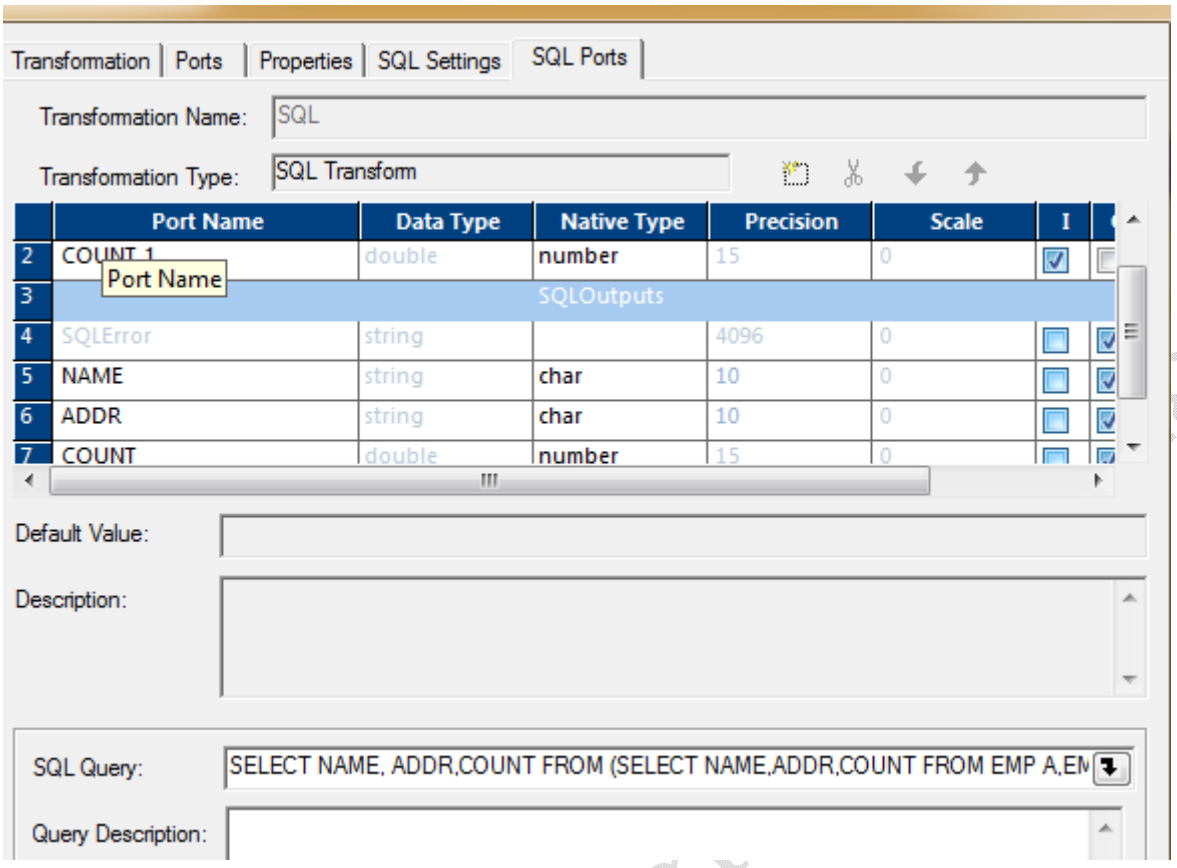


Fig.: The above screenshots shows how to generate multiple records in target based on source column value by using SQL Transformation with Query Mode option use a SQL Transformation

Below is the query

SELECT NAME, ADDR, COUNT FROM

(SELECT NAME,ADDR,COUNT FROM EMP A,EMP B WHERE COUNT=?COUNT_1?)

WHERE ROWNUM <=? COUNT_1?

Use space for running notes:

SCENARIO

Requirement:

How to update the target table data if target doesn't have Primary key

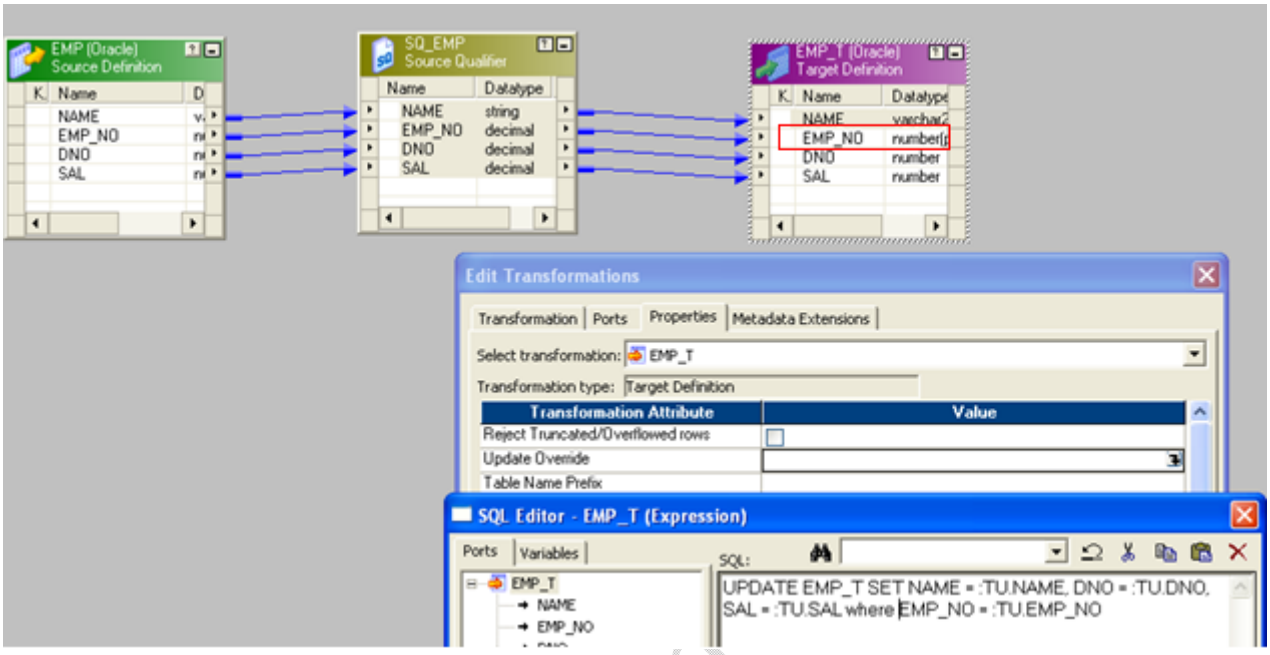


Fig.: The above screenshot shows,
how to update the target table data,
if target doesn't have Primary key

Solution

Approach 1:

- By using the target update override in the mapping we can update target table data without Primary key but this update is special update statement so we need to use ***:TU*** syntax

  Query:

```
UPDATE EMP_T
    SET NAME = :TU.NAME,
    DNO = :TU.DNO,
    SAL =:TU.SAL
WHERE EMP_NO = :TU.EMP_NO
```

Approach 2:

- By setting primary key at Informatica level in target definition at target designer.

Approach 3:

- If you want to update primary key value in target we can use rowid concept in target update override query.

Use this space for running notes

SCENARIO

Requirement: ***How to do performance tuning in informatica if mapping is taking long time***

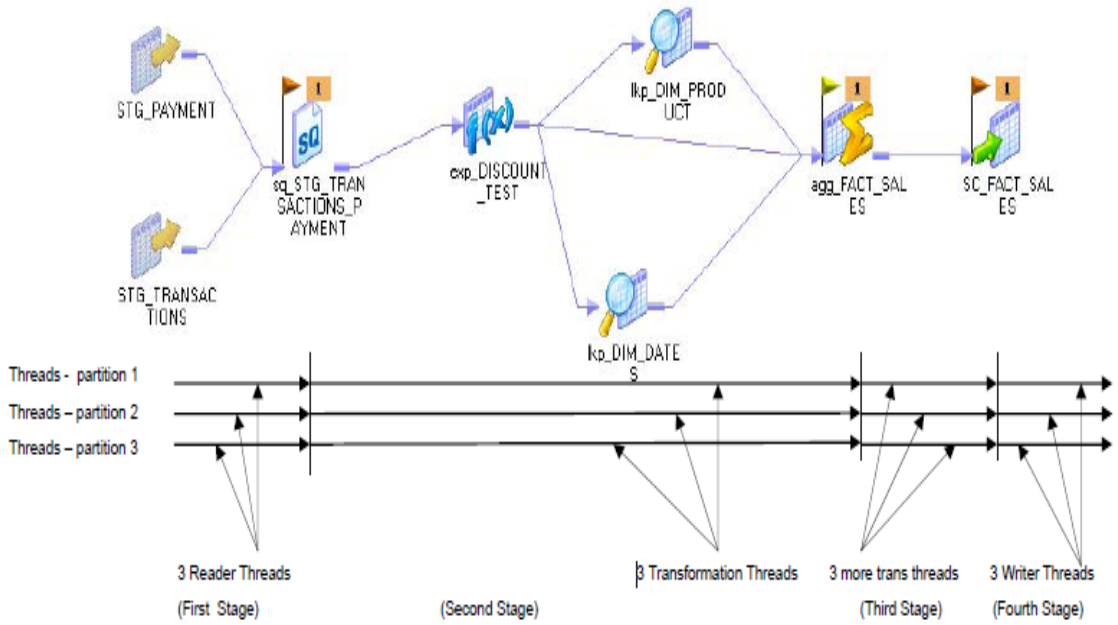
Steps:

- As per best practices we need to design a mapping with minimum transformations
- As much as possible we need to filtered out unwanted data in source qualifier itself
- If any mapping taking long time ,then we need to get session statistics in monitor by get run properties
- If source reading very less records like throughput 5,10,100 etc less records we need to look session log .
- Check busy percentages of reader,Writer and trasformation threads.
- The total run time for the transformation thread is 506 seconds and the busy percentage is 99.7%. This means the transformation thread was never idle for the 506 seconds. The reader and writer busy percentages were significantly smaller, about 9.6% and 24%. In this session, the transformation thread is the bottleneck in the mapping.
- **Hint :** Thread with the highest busy percentage is the bottleneck.
- If end of the session log says query issued to database then there could be issue at source query.So we need to tune the query
- If suppose lookup is taking long time end of the session log says so and so lookup taking time to build the cache we need to use uncached lookup
- If reader and writer both doing good job then we need to do parallel processing using session level partitions

Partitioning Sessions

- Performance can be improved by processing data in parallel in a single session by creating multiple partitions of the pipeline.
- By default session will have one partition that is pass through partition it will create 1 reader and 1 writer thread.
- Rather than processing larger volumes of data through single reader and single writer, we will share with multiple reader and multiple writer using partitions.
- Increasing the number of partitions allows the Integration Service to create multiple connections to sources and process partitions of source data concurrently.
- We have 4 types of partiotions at session level
    - Key range
    - Hash
    - Pass through
    - Round Robin

- If we create key range then we need to specify range values for each partion based on key column.
- If it is pass through then we need to specify SQ queries at session for each partiotion.



SCENARIO

Requirement: ***How many ways we can create a parameter file (Types of parameter files)***

Scope of Parameter files:

- **[Global]** -> All Integration Services, Workflows, Worklets, Sessions.
- **[Service:IntegrationService_Name]** -> The Named Integration Service and Workflows, Worklets, Sessions that runs under this IS.
- **[Folder_Name.WF:Workflow_Name]** -> The Named workflow and all sessions within the workflow.
- **[Folder_Name.WF:Workflow_Name.WT:Worklet_Name]** -> The Named worklet and all sessions within the worklet.
- **[Folder_Name.WF:Workflow_Name.WT:Worklet_Name.WT:Nested_Worklet_Name]** -> The Named nested worklet and all sessions within the nested worklet.
- **[Folder_Name.WF:Workflow_Name.WT:Worklet_Name.ST:Session_Name]** -> The Named Session.
- **[Folder_Name.WF:Workflow_Name.ST:Session_Name]** -> The Named Session.
- **[Folder_Name.ST:Session_Name]** -> The Named Session.
- **[Session_Name]** -> The Named Session.

SCENARIO

Development Code Standards:

The starting point of the development is the logical model created by the Data Architect. This logical model forms the foundation for metadata, which will be continuously be maintained throughout the Data Warehouse Development Life Cycle (DWDLC). The logical model is formed from the requirements of the project. At the completion of the logical model technical documentation defining the sources, targets, requisite business rule transformations, mappings and filters. This documentation serves as the basis for the creation of the Extraction, Transformation and Loading tools to actually manipulate the data from the applications sources into the Data Warehouse/Data Mart.

To start development on any data mart you should have the following things set up by the Informatica Load Administrator

- Informatica Folder. The development team in consultation with the BI Support Group can decide a three-letter code for the project, which would be used to create the informatica folder as well as Unix directory structure.
- Informatica User IDs for the developers
- Unix directory structure for the data mart.
- A schema XXXLOAD on DWDEV database.

Transformation Specifications

Before developing the mappings you need to prepare the specifications document for the mappings you need to develop. A good template is placed in the templates folder You can use your own template as long as it has as much detail or more than that which is in this template.

While estimating the time required to develop mappings the thumb rule is as follows.

- Simple Mapping – 1 Person Day
- Medium Complexity Mapping – 3 Person Days
- Complex Mapping – 5 Person Days.

Usually the mapping for the fact table is most complex and should be allotted as much time for development as possible.

Data Loading from Flat Files

It's an accepted best practice to always load a flat file into a staging table before any transformations are done on the data in the flat file.

Always use LTRIM, RTRIM functions on string columns before loading data into a stage table. You can also use UPPER function on string columns but before using it you need

to ensure that the data is not case sensitive (e.g. ABC is different from Abc)

If you are loading data from a delimited file then make sure the delimiter is not a character which could appear in the data itself. Avoid using comma-separated files. Tilde (~) is a good delimiter to use.

Failure Notification

Once in production your sessions and batches need to send out notification when then fail to the Support team. You can do this by configuring email task in the session level.

Naming Conventions and usage of Transformations

Port Standards:

Input Ports – ***Prefixed with: IN_*** : It will be necessary to change the name of input ports for lookups, expression and filters where ports might have the same name. If ports do have the same name then will be defaulted to having a number after the name. Change this default to a prefix of "in_". This will allow you to keep track of input ports through out your mappings.

Variable Ports – Variable ports that are created within an expression Transformation should be **Prefixed with: V_.** This will allow the developer to distinguish between input/output and variable ports. For more explanation of Variable Ports see the section "VARIABLES".

Output Ports – *Prefixed with: O_*

If organic data is created with a transformation that will be mapped to the target, make sure that it has the same name as the target port that it will be mapped to.

Quick Reference

Object Type	Syntax
Folder	XXX_<Data Mart Name>
Mapping	m_fXY_ZZZ_<Target Table Name>_x.x
Session	s_fXY_ZZZ_<Target Table Name>_x.x
Batch	b_<Meaningful name representing the sessions inside>
Source Definition	<Source Table Name>
Target Definition	<Target Table Name>
Aggregator	AGG_<Purpose>
Expression	EXP_<Purpose>
Filter	FLT_<Purpose>
Joiner	JNR_<Names of Joined Tables>
Lookup	LKP_<Lookup Table Name>
Normalizer	Norm_<Source Name>
Rank	RNK_<Purpose>
Router	RTR_<Purpose>
Sequence Generator	SEQ_<Target Column Name>
Source Qualifier	SQ_<Source Table Name>
Stored Procedure	STP_<Database Name>_<Procedure Name>
Update Strategy	UPD_<Target Table Name>_xxx
Mapplet	MPP_<Purpose>
Input Transformation	INP_<Description of Data being funneled in>
Output Transformation	OUT_<Description of Data being funneled out>
Database Connections	XXX_<Database Name>_<Schema Name>

SCENARIO

Requirement: ***How to skip header and footer records while processing Source file.***

Step 1: To eliminate Header Record, we can use Source Definition advanced option in source analyzer**,**

skip No. of initial rows = 1.

Step 2: To delete Footer Record from the source file, we need to execute sed command in pre – session command property.

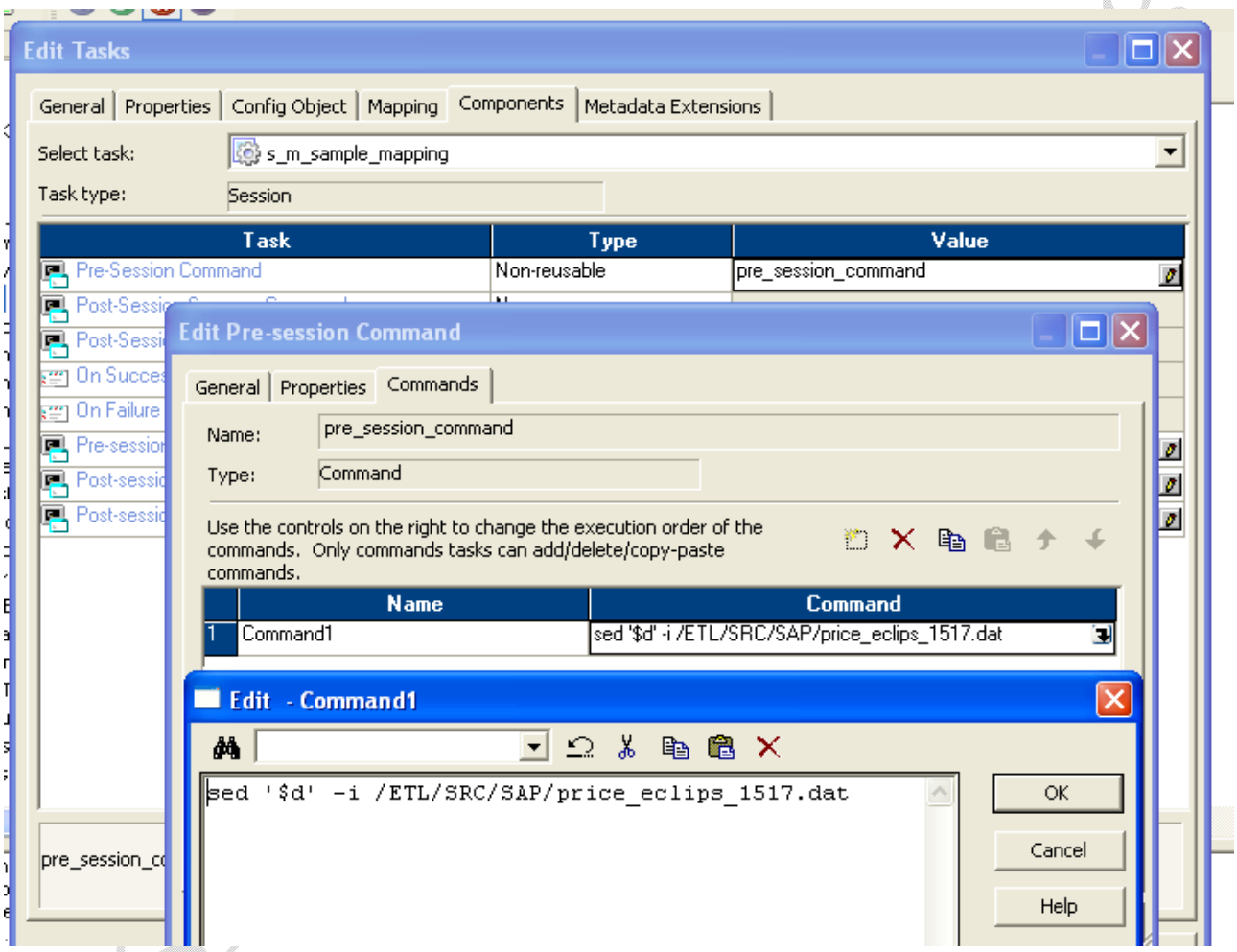


Fig. The above screenshot shows
The command to delete the last line (footer) of the file

Use Space for Running Notes:

SCENARIO

Requirement: ***If same target table populated by two mappings. If these two corresponding sessions execute parallelly, how do you handle Surrogate key in your target table?***

Step 1:

We need to create non – reusable sequence generator transformation, in each mapping individually.

Step 2:

In first mapping's sequence generator, set start value = 1, increment by 2.

Step 3:

In Second mapping's sequence generator, set start value = 2, increment by 2.
So that these two sequence generator will not produce same value at any point of time.

Use Space for Running Notes:

UNIT TESTING:

Unit testing is developer responsibility to make sure mapping is meeting the all the business scenarios

System Integration Testing:

Which will be performed by ETL testing team to test all the functional and technical scenarios for all of the application modules in the same environment.

User Acceptance Testing(UAT): Which will perform by Business users

Unit Test Plan Template

Step#	Description	Test Conditions	Expected Results	Actual Results,	Pass or Fail (P or F)	Tested By
SAP- CMS Interfaces						
1	Check for the total count of records in source tables that is fetched and the total records in the PRCHG table for a particular session timestamp	**SOURCE:** SELECT count(*) FROM XST_PRCHG_STG **TARGET:** Select count(*) from _PRCHG	Both the source and target table load record count should match.	Should be same as the expected	Pass	Stev
2	Check for all the target columns whether they are getting populated correctly with source data.	select PRCHG_ID, PRCHG_DESC, DEPT_NBR, EVNT_CTG_CDE, PRCHG_TYP_CDE, PRCHG_ST_CDE, from T_PRCHG **MINUS** select PRCHG_ID, PRCHG_DESC, DEPT_NBR, EVNT_CTG_CDE, PRCHG_TYP_CDE, PRCHG_ST_CDE, from PRCHG	Both the source and target table record values should return zero records	Should be same as the expected	Pass	Stev
3	Check for Insert strategy to load records into target table.	Identify a one record from the source which is not in target table. Then run the session	It should insert a record into target table with source data	Should be same as the expected	Pass	Stev
4	Check for Update strategy to load records into target table.	Identify a one Record from the source which is already present in the target table with different PRCHG_ST_CDE or PRCHG_TYP_CDE values Then run the session	It should update record into target table with source data for that existing record	Should be same as the expected	Pass	Stev

Scenario :

Difference between Source Qualifier, Joiner and Lookup transformation.

Source Qualifier	Joiner	Lookup
In source qualifier it will push all the matching records.	We cannot handle multiple match data to return one record either first / last / any of the record.	We can handle multiple match data to return either first / last / any of the record / all matching data / error report using multiple match policy.
When both source and lookup are in same database we can use source qualifier.	We cannot override the query in joiner	We can override the query in lookup to fetch the data from multiple tables.
When we want to join multiple sources, if the sources are in the same database, we can use a Source Qualifier Query Override.	We can't perform any filters along with join condition in joiner transformation.	We can apply filters along with lkp conditions using lkp query override lookup transformation.
In source qualifier there is no concept of cache.	Except '=' operator, we cannot use relational operators in joiner transformation.(i.e. <,>,<= and so on)	We can use any relational operators in lookup transformation.(i.e. =, <,>,<= and so on)
Along with the join condition, we can apply filters.		

SCENARIO

Difference between Dynamic Lookup Cache and Static Lookup Cache.

Dynamic Lookup Cache	Static Lookup Cache
In dynamic lookup the cache memory will get refreshed as soon as the record get inserted or updated/deleted in the lookup table.	In static lookup the cache memory will not get refreshed even though record inserted or updated in the lookup table it will refresh only in the next session run.
When we configure a lookup transformation to use a dynamic lookup cache, you can only use the equality operator in the lookup condition. NewLookupRow port will enable automatically.	It is a default cache.
Best example where we need to use dynamic cache is if suppose first record and last record both are same but there is a change in the address. What informatica mapping has to do here is first record needs to get insert and last record should get update in the target table.	If we use static lookup first record it will go to lookup and check in the lookup cache based on the condition it will not find the match so it will return null value then in the router it will send that record to insert flow. But still this record dose not available in the cache memory so when the last record comes to lookup it will check in the cache it will not find the match so it returns null value again it will go to insert flow through router but it is suppose to go to update flow because cache didn't get refreshed when the first record get inserted into target table.
NewLookupRow output Port will enable, when we configure dynamic lookup.	

NewLookupRow Value	Description
0	The PowerCenter Server does not update or insert the row in the cache.
1	The PowerCenter Server inserts the row into the cache.
2	The PowerCenter Server updates the row in the cache.

SCENARIO

Difference between Connected Lookup and Unconnected Lookup.

Connected Lookup	Unconnected Lookup
This is connected to pipleline and receives the input values from pipleline.	Which is not connected to pipeline and receives input values from the result of a: LKP expression in another transformation via arguments.
We cannot use this lookup more than once in a mapping.	We can use this transformation more than once within the mapping
We can return multiple columns from the same row.	Designate one return port (R), returns one column from each row.
We can configure to use dynamic cache.	We cannot configure to use dynamic cache.
Pass multiple output values to another transformation. Link lookup/output ports to another transformation.	Pass one output value to another transformation. The lookup/output/return port passes the value to the transformation calling: LKP expression.
Use a dynamic or static cache	Use a static cache
Supports user defined default values.	Does not support user defined default values.
Cache includes the lookup source column in the lookup condition and the lookup source columns that are output ports.	Cache includes all lookup/output ports in the lookup condition and the lookup/return port.

IMPORTANT UNIX COMMANDS

How to list down the files and directories

```
$ ls:Olny list not in long list format
$ ls –ltr:To display in long list format
```

Display Hidden Files

To show all the hidden files in the directory, use '-a option'. Hidden files in Unix starts with '.' in its file name.

```
$ ls -a
```

tar command examples

The primary uses of this command is to compress multiple files

```
$ tar cvf archive_name.tar dirname/
```

Extract from an existing tar archive.

```
$ tar xvf archive_name.tar
```

gzip command examples

To zip or compress a single file

To create a *.gz compressed file : `$ gzip test.txt`
To uncompress a *.gz file : `$ gzip -d test.txt.gz`

grep command examples

Search for a given string in a file (*case in-sensitive search*) :

```
$ grep -i "the" demo_file
```

Print the matched line, along with the 3 lines after it. :

```
$ grep -A 3 -i "example" demo_text
```

find command examples

Find files using file-name (case in-sensitve find): `# find -iname "MyCProgram.c"`

sed command examples

Mainly we use SED cmd to find replace in a file or used for modifying the files in unix

- Replacing or substituting string
- *Sed command* is mostly used to replace the text in a file.
- The below simple sed command replaces the word "unix" with "linux" in the file.

```
$> sed 's/unix/linux/' file.txt
$> sed –i '5,7 d' file.txt
```

The above command will delete line 5 to line 7 from file.txt

awk command examples

Remove duplicate lines using awk

```
$ awk '!($0 in array) { array[$0]; print }' temp
```

vim command examples

Get from nth line onwards file content : `$ vim +5 filename.txt`

The above command will display 5th line onwards the data from filename.txt

sort command examples

Sort a file in ascending order

```
$ sort psk.txt
```

Sort a file in descending order

```
$ sort -r psk.txt
```

pwd command : pwd is Print working directory.

```
$ pwd
```

ftp command examples

Both ftp and secure ftp (sftp) has similar commands.

To connect to a remote server and download multiple files

SFTP: It doesn't require password

FTP: It Required username and password

```
$ ftp IP/hostname
  Username--- password---
$sftp username@hostname
```

crontab command examples

Crontab command.

Crontab command is used to schedule jobs. You must have permission to run this command by Unix Administrator. Jobs are scheduled in five numbers, as follows.

Minutes (0-59) Hour (0-23) Day of month (1-31) month (1-12) Day of week (0-6) (0 is Sunday)

ps command examples

ps command is used to display information about the processes that are running in the system. To view current running processes.

```
$ ps -ef | more
```

df command examples

Displays the file system disk space usage. By default df -k displays output in bytes

```
$ df -k
```

df -h displays output in human readable form. i.e size will be displayed in GB's.

kill command examples

Use kill command to terminate a process. First get the process id using ps -ef command, then use kill -9 to kill the running Linux process as shown below. You can also use kill all, pkill, xkill to terminate a unix process.

```
$ kill -9 7243
```

rm command examples

Get confirmation before removing the file.

```
$ rm -i filename.txt
```

cp command examples

Copy file1 to file2 preserving the mode, ownership and timestamp. `$ cp -p file1 file2`

mv command examples

Rename file1 to file2. : `$ mv  file1 file2`

cat command examples

You can view multiple files at the same time. Following example prints the content of file1 followed by file2 to stdout.

```
$ cat file1 file2
```

chmod command examples

chmod command is used to change the permissions for a file or directory.

Give full access to user and group (i.e read, write and execute) on a specific file.

```
$ chmod ug+rwx file.txt
```

mkdir command examples

How to create a directory : `$ mkdir temp`

rmdir - remove a directory

To remove the Dircetories : `$ rmdir info`

wc - display a count of lines, words and characters

wc counts the number of lines, words, and characters in files.

The command gives no of lines in a file `$ wc-l  psk.txt`

echo command

echo is used to print the message

```
echo "PSK training Istitute" > PSK.txt
```

How to print/display first n lines of a file?

There are many ways to do this. However the easiest way to display the first line of a file is using the [head] command.

Below command will display first 5 lines from the file

```
$> head -5 file.txt
```

How to print/display the last n lines of a file?

Below command will display last 5 lines

```
$> tail -10 file.txt
```

How to display n-th line of a file?

```
$> sed –n '<n> p' file.txt
```

You need to replace <n> with the actual line number. So if you want to print the 4th line,

```
$> sed –n '4 p' test
```

How to show the non-printable characters in a file?

Open the file in VI editor. Go to VI command mode by pressing [Escape] and then [:]. Then type [set list]. This will show you all the non-printable characters, e.g. Ctrl-M characters (^M) etc., in the file.

How to check if the last command was successful in Unix?

$? Will gives the status of previous command.

To display the value assigned to shell variable

```
$> echo $Home  Output: /home/useri3
```

How to connect to the oracle database from within shell script?

```
$ res = `sqlplus –s username/password@database_name <<EOF`
```