

Practical No. 01

Aim: Implementation of Data Partitioning through Range and List Partitioning

a) Range Partitioning

1.1 Create table sales with the following columns:

prod_id	Number
cust_id	Number
time_id	Date
channel_id	Char
promo_id	Number
quantity_sold	Number
amount_sold	Number

- Partition this table into 4 using range partition and time_id as partitioning key.
Give partition names as:
sales_q1_2006, sales_q2_2006, sales_q3_2006, sales_q4_2006.

Query:

✓ create table sales(

```
prod_id number(5),
cust_id number(5),
time_id date, channel_id
char(5), promo_id
number(5),
quantity_sold number(5),
amount_sold number(10))
partition by range(time_id)
(partition sales_q1_2006 values less than (to_date('31-MAR-2006','dd-MON-yyyy')),
partition sales_q2_2006 values less than (to_date('31-MAY-2006','dd-MON-yyyy')),
partition sales_q3_2006 values less than (to_date('30-SEP-2006','dd-MON-yyyy')),
partition sales_q4_2006 values less than (to_date('31-DEC-2006','dd-MON-yyyy')));
```

```
SQL> create table sales(
  2 prod_id number(5),
  3 cust_id number(5),
  4 time_id date,
  5 channel_id char(5),
  6 promo_id number(5),
  7 quantity_sold number(5),
  8 amount_sold number(10))
  9 partition by range(time_id)
 10 (partition sales_q1_2006 values less than (to_date('31-MAR-2006','dd-MON-yyyy')),
 11 partition sales_q2_2006 values less than (to_date('31-MAY-2006','dd-MON-yyyy')),
 12 partition sales_q3_2006 values less than (to_date('30-SEP-2006','dd-MON-yyyy')),
 13 partition sales_q4_2006 values less than (to_date('31-DEC-2006','dd-MON-yyyy')));

Table created.

SQL> |
```

- Store quarterly data into each partition. For example, partition sales_q1_2006 will store records for first quarter 01-jan-2006 to 01-mar-2006 sales_q1_2006 will store records for second quarter 01-apr-2006 to 01- jun2006. And so on.

Query:

✓ insert into sales values(2015,103,'15-Mar-2006',101,234,23,1029);

```
insert into sales values(3013,204,'03-Apr-2006',103,456,56,2938);
insert into sales values(6016,302,'27-Jun-2006',105,567,43,3847);
insert into sales values(8101,404,'05-Aug-2006',204,678,95,4756);
insert into sales values(4106,501,'17-Nov-2006',206,789,62,2374);
insert into sales values(8014,603,'23-Feb-2006',303,901,83,6358);
insert into sales values(2761,625,'13-Sep-2006',403,432,57,9341);
```

Output:

```
SQL> insert into sales values(2015,103,'15-Mar-2006',101,234,23,1029);
1 row created.

SQL> insert into sales values(3013,204,'03-Apr-2006',103,456,56,2938);
1 row created.

SQL> insert into sales values(6016,302,'27-Jun-2006',105,567,43,3847);
1 row created.

SQL> insert into sales values(8101,404,'05-Aug-2006',204,678,95,4756);
1 row created.

SQL> insert into sales values(4106,501,'17-Nov-2006',206,789,62,2374);
1 row created.

SQL> insert into sales values(8014,603,'23-Feb-2006',303,901,83,6358);
1 row created.

SQL> insert into sales values(2761,625,'13-Sep-2006',403,432,57,9341);
1 row created.

SQL> |
```

- Write a command to view records in each partition.

Query:

✓ select * from sales partition (sales_q1_2006);

select * from sales partition (sales_q2_2006);

select * from sales partition (sales_q3_2006);

select * from sales partition (sales_q4_2006);

Output:						
SQL> select * from sales partition (sales_q1_2006);						
PROD_ID	CUST_ID	TIME_ID	CHANN	PROMO_ID	QUANTITY SOLD	AMOUNT SOLD
2815	103	15-MAR-06	101	234	23	1029
8014	603	23-FEB-06	303	901	83	6358
SQL> select * from sales partition (sales_q2_2006);						
PROD_ID	CUST_ID	TIME_ID	CHANN	PROMO_ID	QUANTITY SOLD	AMOUNT SOLD
3013	204	03-APR-06	103	456	56	2938
SQL> select * from sales partition (sales_q3_2006);						
PROD_ID	CUST_ID	TIME_ID	CHANN	PROMO_ID	QUANTITY SOLD	AMOUNT SOLD
6016	302	27-JUN-06	105	567	43	3847
8101	404	05-AUG-06	204	678	95	4756
2761	625	13-SEP-06	403	432	57	9341
SQL> select * from sales partition (sales_q4_2006);						
PROD_ID	CUST_ID	TIME_ID	CHANN	PROMO_ID	QUANTITY SOLD	AMOUNT SOLD
4106	501	17-NOV-06	206	789	62	2374

- Write a command to display the partition structure.

Query:

✓ select table_name,partition_name,high_value,num_rows from user_tab_partitions where table_name='SALES';

SQL> select table_name,partition_name,high_value,num_rows from user_tab_partitions where table_name='SALES';	
TABLE_NAME	PARTITION_NAME
HIGH_VALUE	
HIGH_ROWS	
SALES	SALES_Q1_2006
TO_DATE(' 2006-03-31 00:00:00', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA')	
SALES	SALES_Q2_2006
TO_DATE(' 2006-05-31 00:00:00', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA')	
TABLE_NAME	PARTITION_NAME
HIGH_VALUE	
HIGH_ROWS	
SALES	SALES_Q3_2006
TO_DATE(' 2006-09-30 00:00:00', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA')	
SALES	SALES_Q4_2006
TO_DATE(' 2006-12-31 00:00:00', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA')	
TABLE_NAME	PARTITION_NAME
HIGH_VALUE	
HIGH_ROWS	

- Write a command to add a new partition called sales_q1_2007 for the next quarter value.

Query:

- ✓ alter table sales add partition sales_q1_2007 values less than(TO_DATE('31-MAR-2007','dd-MON-yyyy'));

Output:

```
SQL> alter table sales add partition sales_q1_2007 values less than(TO_DATE('31-MAR-2007','dd-MON-yyyy'));
Table altered.
```

```
SQL> select table_name,partition_name,high_value,num_rows from user_tab_partitions where table_name='SALES';

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----
SALES           SALES_Q1_2006
TO_DATE(' 2006-03-31 00:00:00', 'SYYYYY-MON-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
          0

SALES           SALES_Q1_2007
TO_DATE(' 2007-03-31 00:00:00', 'SYYYYY-MON-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----
SALES           SALES_Q2_2006
TO_DATE(' 2006-05-31 00:00:00', 'SYYYYY-MON-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
          0

SALES           SALES_Q3_2006
TO_DATE(' 2006-09-30 00:00:00', 'SYYYYY-MON-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----
          0

SALES           SALES_Q4_2006
TO_DATE(' 2006-12-31 00:00:00', 'SYYYYY-MON-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

- Write a command to delete all records from partition sales_q1_2006.

Query:

- ✓ alter table sales truncate partition sales_q1_2006;

```

SQL> alter table sales truncate partition sales_q1_2006;
Table truncated.

SQL> select * from sales partition (sales_q1_2006);
no rows selected

SQL> |

```

- Write a command to delete a partition.

Query:

✓ alter table sales drop partition sales_q3_2006;

Output:

```

SQL> alter table sales drop partition sales_q3_2006;
Table altered.

SQL> select table_name,partition_name,high_value,num_rows from user_tab_partitions where table_name='SALES';

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----
SALES              SALES_Q1_2006
TO_DATE(' 2006-03-31 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
  0

SALES              SALES_Q1_2007
TO_DATE(' 2007-03-31 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----
SALES              SALES_Q2_2006
TO_DATE(' 2006-05-31 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
  0

SALES              SALES_Q4_2006
TO_DATE(' 2006-12-31 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

TABLE_NAME          PARTITION_NAME
-----            -----
HIGH_VALUE
-----            -----
NUM_ROWS
-----            -----

```

b) List Partitioning

1.2 Create table Student with the following columns:

Student_id	Number
Student_name	Number
Student_dob	Date

- Create list partition with student_name as partition key. Create following two partitions.
stu_divA with values 'a','b','c','d','e','f','g','h','i','j','k'
stu_divB with values 'n','o','p','q','r','s','t','u','v','w','x','y','z'

Query:

✓ create table Student(
student_id number(5),
student_name varchar2(20),
student_dob date)
partition by list(student_name)
(partition stu_divA values('a','b','c','d','e','f','g','h','i','j','k'),
partition stu_divB values('n','o','p','q','r','s','t','u','v','w','x','y','z'));

Output:

```
SQL> create table Student(  
 2 student_id number(5),  
 3 student_name varchar2(20),  
 4 student_dob date)  
 5 partition by list(student_name)  
 6 (partition stu_divA values('a','b','c','d','e','f','g','h','i','j','k'),  
 7 partition stu_divB values('n','o','p','q','r','s','t','u','v','w','x','y','z'));  
  
Table created.  
  
SQL> |
```

Query:

✓ insert into Student values(0121,'m',to_date('23-AUG-1963','dd-MON-yyyy'));

insert into Student values(0221,'j',to_date('15-SEP-1958','dd-MON-yyyy'));
insert into Student values(0321,'a',to_date('19-AUG-1992','dd-MON-yyyy'));
insert into Student values(0421,'h',to_date('10-JUL-1991','dd-MON-yyyy'));
insert into Student values(0521,'p',to_date('27-JUN-1965','dd-MON-yyyy'));
insert into Student values(0621,'z',to_date('18-MAR-1968','dd-MON-yyyy'));

```
SQL> insert into Student values(0221,'j',to_date('15-SEP-1958','dd-MON-yyyy'));
1 row created.

SQL> insert into Student values(0321,'a',to_date('19-AUG-1992','dd-MON-yyyy'));
1 row created.

SQL> insert into Student values(0421,'h',to_date('10-JUL-1991','dd-MON-yyyy'));
1 row created.

SQL> insert into Student values(0521,'p',to_date('27-JUN-1965','dd-MON-yyyy'));
1 row created.

SQL> insert into Student values(0621,'z',to_date('18-MAR-1968','dd-MON-yyyy'));
1 row created.
```

- Write a command to view records in each partitions.

Query:

✓ select * from Student partition(stu_divA);

select * from Student partition(stu_divB);

Output:

```
SQL> select * from Student partition(stu_divA);

STUDENT_ID STUDENT_NAME          STUDENT_D
----- -----
  221   j                      15-SEP-58
  321   a                      19-AUG-92
  421   h                      10-JUL-91

SQL> select * from Student partition(stu_divB);

STUDENT_ID STUDENT_NAME          STUDENT_D
----- -----
  521   p                      27-JUN-65
  621   z                      18-MAR-68
```

- Write a command to display the partition structure.

Query:

✓ SELECT TABLE_NAME, PARTITION_NAME,HIGH_VALUE, NUM_ROWS FROM USER_TAB_PARTITIONS WHERE TABLE_NAME='STUDENT';

Output:

```
SQL> SELECT TABLE_NAME, PARTITION_NAME,HIGH_VALUE, NUM_ROWS FROM USER_TAB_PARTITIONS
2 WHERE TABLE_NAME='STUDENT';

TABLE_NAME          PARTITION_NAME
-----              -----
HIGH_VALUE
-----
NUM_ROWS
-----
STUDENT             STU_DIVA
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'

STUDENT             STU_DIVB
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'

TABLE_NAME          PARTITION_NAME
-----              -----
HIGH_VALUE
-----
NUM_ROWS
-----
```

- Write a command to add a new partition called stu_null for the null values.

Query:

✓ alter table student add partition stu_null values(NULL);

Output:

```
SQL> alter table Student add partition stu_null values(NULL);

Table altered.

SQL> SELECT TABLE_NAME, PARTITION_NAME,NUM_ROWS FROM USER_TAB_PARTITIONS WHERE
2 TABLE_NAME='STUDENT';

TABLE_NAME          PARTITION_NAME          NUM_ROWS
-----              -----                  -----
STUDENT             STU_DIVA
STUDENT             STU_DIVB
STUDENT             STU_NULL
```

- Write a command to display records from stu_null partition.

Query:

- ✓ select * from Student partition(stu_null);

Output:

```
SQL> select * from Student partition(stu_null);
no rows selected

SQL> SELECT TABLE_NAME, PARTITION_NAME,NUM_ROWS FROM USER_TAB_PARTITIONS WHERE
2  TABLE_NAME='STUDENT';

TABLE_NAME          PARTITION_NAME          NUM_ROWS
-----              -----                  -----
STUDENT             STU_DEFAULT
STUDENT             STU_DIVA
STUDENT             STU_DIUB
STUDENT             STU_NULL
```

- Write a command to add a new partition called stu_default for the default values.

Write a command to display records from stu_default partition.

Query:

- ✓ alter table Student add partition stu_default values(DEFAULT);
- ✓ select * from Student partition(stu_default);

Output:

```
SQL> alter table Student add partition stu_default values(DEFAULT);
Table altered.

SQL> SELECT TABLE_NAME, PARTITION_NAME,NUM_ROWS FROM USER_TAB_PARTITIONS WHERE
2  TABLE_NAME='STUDENT';

TABLE_NAME          PARTITION_NAME          NUM_ROWS
-----              -----                  -----
STUDENT             STU_DEFAULT
STUDENT             STU_DIVA
STUDENT             STU_DIUB
STUDENT             STU_NULL
```

```

SQL> SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, NUM_ROWS FROM USER_TAB_PARTITIONS
  2 WHERE TABLE_NAME='STUDENT';

TABLE_NAME          PARTITION_NAME
-----              -----
HIGH_VALUE
NUM_ROWS
STUDENT             STU_DEFAULT
DEFAULT

STUDENT             STU_DIVA
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'

TABLE_NAME          PARTITION_NAME
-----              -----
HIGH_VALUE
NUM_ROWS
STUDENT             STU_DIVB
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'

STUDENT             STU_NULL
NULL

TABLE_NAME          PARTITION_NAME
-----              -----
HIGH_VALUE
NUM_ROWS

```

- Write a command to add values 'l' and 'm' in a partition stu_divA

Query:

✓ alter table Student modify partition stu_divA add values('l','m');

Output:

```

SQL> alter table Student modify partition stu_divA add values('l','m');

Table altered.

```

TABLE_NAME	PARTITION_NAME
HIGH_VALUE	
NUM_ROWS	
STUDENT	STU_DEFAULT
DEFAULT	
STUDENT	STU_DIVA
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm'	

- Write a command to display records from stu_divA partition.

Query:

✓ select * from Student partition(stu_divA);

Output:

```
SQL> select * from Student partition(stu_divA);

STUDENT_ID STUDENT_NAME          STUDENT_D
-----  -----
  221    j                  15-SEP-58
  321    a                  19-AUG-92
  421    h                  10-JUL-91
  521    l                  02-JUL-65
  621    m                  08-MAY-68

SQL> |
```

Practical No. 02

Aim: Implementation of Analytical Queries like Roll_Up, CUBE, First, Last, Lead, Lag, Row_number, Rank and Dense Rank.

- Creating a Table: Employee Table and Inserting values in the Table.

Query:

- ✓ create table Employee(
 EmpNo number(6), Name
 varchar(10), Position
 varchar(6), Manager
 number(6), JoinDate date,
 Salary number(7),
 DeptNo number(3));

- ✓ insert into Employee values(1632,'Sam','Asstt',131632,to_date('9-6-1963','dd-mm-yyyy'),11370,102);
insert into Employee values(1461,'Lark','Asstt',131461,to_date('2-4-1965','dd-mm-yyyy'),25410,203);
insert into Employee values(2345,'Herman','Officer',130245,to_date('19-4-1978','dd-mm-yyyy'),37520,102);
insert into Employee values(2893,'Daniel','Officer',131461,to_date('3-12-1988','dd-mm-yyyy'),8230,203);
insert into Employee values(5845,'Park','Mngr',130245,to_date('17-12-1991','dd-mm-yyyy'),45840,203);
insert into Employee values(6459,'Muel','Mngr',131461,to_date('20-2-1993','dd-mm-yyyy'),64559,304);
insert into Employee values(7583,'Shane','SrMngr',130245,to_date('22-2-1995','dd-mm-yyyy'),77580,304);
insert into Employee values(9534,'Murr','SrMngr',131632,to_date('25-2-1989','dd-mm-yyyy'),95450,304);

Output:

```
SQL> select * from Employee;
```

EMPNO	NAME	POSITION	MANAGER	JOINDATE	SALARY	DEPTNO
1632	Sam	Asstt	131632	09-JUN-63	11370	102
1461	Lark	Asstt	131461	02-APR-65	25410	203
5845	Park	Mngr	130245	17-DEC-91	45840	203
6459	Muel	Mngr	131461	20-FEB-93	64559	304
7583	Shane	SrMngr	130245	22-FEB-95	77580	304
9534	Murr	SrMngr	131632	25-FEB-89	95450	304
2345	Herman	Officer	130245	19-APR-78	37520	102
2893	Daniel	Officer	131461	03-DEC-88	8230	203

```
8 rows selected.
```

□ **Roll_Up**

Syntax:

- ✓ select c1, c2, aggregate_function(c3)
from table_name
GROUP BY ROLLUP (c1, c2);
- ✓ **Roll_Up** is a modifier used to produce the summary output, including extra rows that represent super-aggregate (higher-level) summary operations. It enables us to sum-up the output at multiple levels of analysis using a single query.

Query:

- ✓ Without RollUp:

```
select DeptNo, sum(Salary) as
Total from Employee group by
DeptNo
order by DeptNo;
```

Output:

SQL> select DeptNo, sum(Salary)	
2 as Total from Employee	
3 group by DeptNo	
4 order by DeptNo;	

DEPTNO	TOTAL
-----	-----
102	48890
203	79480
304	237589

- ✓ With RollUp:

```
select DeptNo, sum(Salary) as
Total from Employee group by
rollup(DeptNo) order by
DeptNo;
```

Output:

SQL> select DeptNo, sum(Salary)	
2 as Total from Employee	
3 group by rollup(DeptNo)	
4 order by DeptNo;	

DEPTNO	TOTAL
-----	-----
102	48890
203	79480

✓ Multiple Column without RollUp:

```
select DeptNo, Position, sum(Salary) as  
Total from Employee  
group by DeptNo, Position  
order by 1,2;
```

Output:

DEPTNO	POSITION	TOTAL
102	Asstt	11370
102	Oficer	37520
203	Asstt	25410
203	Mngr	45840
203	Oficer	8230
304	Mngr	64559
304	SrMngr	173030

✓ Multiple Column with RollUp:

```
select DeptNo, Position, sum(Salary) as  
Total from Employee  
group by rollup(DeptNo, Position) order  
by 1,2;
```

Output:

```
SQL> select DeptNo,Position,sum(Salary)
  2  as Total from Employee
  3  group by rollup(DeptNo,Position)
  4  order by 1,2;
```

DEPTNO	POSITION	TOTAL
102	Asstt	11370
102	Oficer	37520
102		48890
203	Asstt	25410
203	Mngr	45840
203	Oficer	8230
203		79480
304	Mngr	64559
304	SrMngr	173030
304		237589
		365959

```
11 rows selected.
```

□ **CUBE**

Syntax:

✓ select c1, c2, aggregate_function(c3)
from table_name
GROUP BY CUBE (c1, c2);

✓ **CUBE** is an extension similar to ROLLUP, enabling a single statement to calculate all possible combinations of subtotals. CUBE can generate the information needed in cross-tab reports with a single query.

Query:

✓ Without CUBE:

```
select DeptNo, sum(Salary) as  
Total from Employee group by  
DeptNo  
order by DeptNo;
```

Output:

```
SQL> select DeptNo, sum(Salary)  
2  as Total from Employee  
3  group by DeptNo  
4  order by DeptNo;
```

DEPTNO	TOTAL
102	48890
203	79480
304	237589

✓ With CUBE:

```
select DeptNo, sum(Salary) as  
Total from Employee group by  
cube(DeptNo) order by  
DeptNo;
```

Output:

```
SQL> select DeptNo, sum(Salary)
  2  as Total from Employee
  3  group by cube(DeptNo)
  4  order by DeptNo;
```

DEPTNO	TOTAL
102	48898
203	79488
304	237589
	365959

✓ Multiple Column without Cube:

```
select DeptNo, Position, sum(Salary) as  
Total from Employee  
group by DeptNo, Position  
order by 1,2;
```

Output:

```
SQL> select DeptNo,Position,sum(Salary)  
2 as Total from Employee  
3 group by DeptNo,Position  
4 order by 1,2;  
  
DEPTNO POSITI      TOTAL  
----- -----  
    102 Asstt        11370  
    102 Oficer       37520  
    203 Asstt        25410  
    203 Mngr         45840  
    203 Oficer       8230  
    304 Mngr         64559  
    304 SrMngr       173030  
  
7 rows selected.
```

✓ Multiple Column with Cube:

```
select DeptNo,Position,sum(Salary) as  
Total from Employee  
group by cube(DeptNo,Position) order  
by 1,2;
```

Output:

```
SQL> select DeptNo,Position,sum(Salary)  
2 as Total from Employee  
3 group by cube(DeptNo,Position)  
4 order by 1,2;  
  
DEPTNO POSITI      TOTAL  
----- -----  
    102 Asstt        11370  
    102 Oficer       37520  
    102              48890  
    203 Asstt        25410  
    203 Mngr         45840  
    203 Oficer       8230  
    203              79480  
    304 Mngr         64559  
    304 SrMngr       173030  
    304              237589  
    Asstt           36780  
  
DEPTNO POSITI      TOTAL  
----- -----  
    Mngr            110399  
    Oficer          45750  
    SrMngr          173030  
    365959
```

15 rows selected.

□ **Rank**

Syntax:

- ✓ select c1, c2, c3 rank()

over(order by c1) RANK
from table_name;

- ✓ **Rank** function assigns a rank to each row within a partition such that each row's rank is one more than total number of rows that have been ranked in that partition.

Query:

- ✓ select Salary, rank() over(order

by Salary) RANK from
Employee;

Output:

```
SQL> select Salary, rank()  
2 over(order by Salary) RANK  
3 from Employee;
```

SALARY	RANK
8230	1
11370	2
25410	3
37520	4
45840	5
64559	6
77580	7
95450	8

8 rows selected.

- ✓ select EmpNo, DeptNo, Salary, JoinDate, rank()

over(order by Salary) RANK

```
SQL> select EmpNo, DeptNo, Salary, JoinDate, rank()  
2 over(order by Salary) RANK  
3 from Employee;
```

EMPNO	DEPTNO	SALARY	JOINDATE	RANK
2893	203	8230	03-DEC-88	1
1632	102	11370	09-JUN-63	2
1461	203	25410	02-APR-65	3
2345	102	37520	19-APR-78	4
5845	203	45840	17-DEC-91	5
6459	304	64559	20-FEB-93	6
7583	304	77580	22-FEB-95	7
9534	304	95450	25-FEB-89	8

8 rows selected.

Dense Rank

Syntax:

- ✓ select c1, c2, c3 dense_rank()

```
over(order by c1) DENSE_RANK  
from table_name;
```

- ✓ **DENSE_RANK()** is a window function that assigns a rank to each row within a

partition or result set with no gaps in ranking values. The rank of a row is increased by one from the number of distinct rank values which come before the row. **Query:**

- ✓ select Salary, dense_rank()

```
over(order by Salary) DENSE_RANK from  
Employee;
```

Output:

```
SQL> select Salary, dense_rank()  
2 over(order by Salary) DENSE_RANK  
3 from Employee;
```

Salary	DENSE_RANK
8230	1
11370	2
25410	3
37520	4
45840	5
64559	6
77580	7
95450	8

8 rows selected.

- ✓ select EmpNo, DeptNo, Salary, JoinDate, dense_rank()

```
over(order by Salary) DENSE_RANK  
from Employee;
```

```
SQL> select EmpNo, DeptNo, Salary, JoinDate, dense_rank()  
2 over(order by Salary) DENSE_RANK  
3 from Employee;
```

EMPNO	DEPTNO	Salary	JOINDATE	DENSE_RANK
2893	203	8230	03-DEC-88	1
1632	102	11370	09-JUN-63	2
1461	203	25410	02-APR-65	3
2345	102	37520	19-APR-78	4
5845	203	45840	17-DEC-91	5
6459	304	64559	20-FEB-93	6
7583	304	77580	22-FEB-95	7
9534	304	95450	25-FEB-89	8

8 rows selected.

□ **Row_Number**

Syntax:

✓ select column_name, row_number()

over (order by col)

ROW_NUM from table_name;

✓ **ROW_NUMBER()** function is a type of function that returns a number for each row in sequence or serial, beginning from 1 for the first record of the result set to the end in ascending order.

Query:

✓ select EmpNo, row_number()

over (order by Salary)

ROW_NUM from Employee;

Output:

```
SQL> select EmpNo, row_number()
  2 over (order by Salary)
  3 ROW_NUM from Employee;
```

EMPNO	ROW_NUM
2893	1
1632	2
1461	3
2345	4
5845	5
6459	6
7583	7
9534	8

8 rows selected.

✓ select EmpNo, DeptNo, Salary, JoinDate,

row_number() over (order by Salary)

```
SQL> select EmpNo,DeptNo,Salary,JoinDate,
  2 row_number() over (order by Salary)
  3 ROW_NUM from Employee;
```

EMPNO	DEPTNO	SALARY	JOINDATE	ROW_NUM
2893	203	8230	03-DEC-88	1
1632	102	11370	09-JUN-63	2
1461	203	25410	02-APR-65	3
2345	102	37520	19-APR-78	4
5845	203	45840	17-DEC-91	5
6459	304	64559	20-FEB-93	6
7583	304	77580	22-FEB-95	7
9534	304	95450	25-FEB-89	8

8 rows selected.

□ **LEAD**

Syntax:

✓ **LEAD(<sql expr>, <offset>, <default>)**
OVER(<analytic_clause>)

✓ **LEAD()** function is a window function that allows you to look forward a number of rows and access data of that row from the current row. It is very useful for calculating the difference between the current row and the subsequent row within the same result set.

Query:

✓ select DeptNo, EmpNo, Salary,

LEAD(Salary,1,0)

over(partition by DeptNo order by Salary desc) next_low_sal from Employee

where DeptNo in(203,304) order

by DeptNo, Salary desc;

```
SQL> select DeptNo, EmpNo, Salary,
  2 LEAD(Salary,1,0)
  3 over(partition by DeptNo order by Salary desc) next_low_sal
  4 from Employee
  5 where DeptNo in(203,304)
  6 order by DeptNo, Salary desc;
```

DEPTNO	EMPNO	salary	next_low_sal
203	5845	45840	25410
203	1461	25410	8230
203	2893	8230	0
304	9534	95450	77580
304	7583	77580	64559
304	6459	64559	0

6 rows selected.

□ LAG

Syntax:

✓ LAG(<sql expr>, <offset>, <default>)

OVER(<analytic_clause>)

✓ LAG() function is a window function that allows you to look back a number of rows and access data of that row from the current row. It returns the value of the expression from the row that precedes the current row by offset number of rows within its partition or result set.

Query:

✓ select DeptNo, EmpNo, Salary,

LAG(Salary,1,0)

over(partition by DeptNo order by Salary desc) next_high_sal from Employee

where DeptNo in(203,304) order

by DeptNo, Salary desc;

```
SQL> select DeptNo, EmpNo, Salary,
  2  LAG(Salary,1,0)
  3  over(partition by DeptNo order by Salary desc) next_high_sal
  4  from Employee
  5  where DeptNo in(203,304)
  6  order by DeptNo, Salary desc;
```

DEPTNO	EMPNO	salary	next_high_sal
203	5845	45840	0
203	1461	25410	45840
203	2893	8230	25410
304	9534	95450	0
304	7583	77580	95450
304	6459	64559	77580

6 rows selected.

FIRST

Syntax:

✓ Function() KEEP(DENSE_RANK FIRST

ORDER BY<expr>)OVER(<partitioning clause>)

✓ **FIRST** function is used to return the first value of the selected column. Here, we use limit clause to select first record or more.

Query:

✓ select DeptNo, EmpNo, Salary,

min(Salary)keep(dense_rank FIRST order by Salary)

over(partition by DeptNo) as lowest

from Employee

order by DeptNo,Salary;

```
SQL> select DeptNo, EmpNo, Salary,
  2 min(Salary)keep(dense_rank FIRST order by Salary)
  3 over(partition by DeptNo) as lowest
  4 from Employee
  5 order by DeptNo,Salary;
```

DEPTNO	EMPNO	SALARY	LOWEST
102	1632	11370	11370
102	2345	37520	11370
203	2893	8230	8230
203	1461	25410	8230
203	5845	45840	8230
304	6459	64559	64559
304	7583	77580	64559
304	9534	95450	64559

8 rows selected.

□ LAST

Syntax:

✓ Function() KEEP(DENSE_RANK LAST ORDER

BY<expr>)OVER(<partitioning clause>)

✓ LAST function is used to return the last value of the selected column.

Query:

✓ select DeptNo, EmpNo, Salary,

min(Salary)keep(dense_rank LAST order by Salary)

over(partition by DeptNo) as highest

from Employee

order by DeptNo,Salary;

```
SQL> select DeptNo, EmpNo, Salary,
  2 min(Salary)keep(dense_rank LAST order by Salary)
  3 over(partition by DeptNo) as highest
  4 from Employee
  5 order by DeptNo,Salary;
```

DEPTNO	EMPNO	SALARY	HIGHEST
102	1632	11370	37520
102	2345	37520	37520
203	2893	8230	45840
203	1461	25410	45840
203	5845	45840	45840
304	6459	64559	95450
304	7583	77580	95450
304	9534	95450	95450

8 rows selected.

Practical No. 03

Aim: Implementation of ORDBMS using ADT (Abstract Data Type), Reference.

c) Abstract Data Type

- Create Object Type ‘data_type1’ with Object Attribute ‘Year’ and Function ‘Prod(Invest number)’, use this function to return the Sum of Year and Invest value.

Query:

- ✓ create type data_type1 as
 - object(Year number,
 - member function Prod(Invest number)
 - return number);
 - /
- ✓ create type body data_type1 is member
 - function Prod(Invest number) return
 - number is
 - begin return(Year+Invest);
 - end;
 - end;
 - /

Output:

```
SQL> create type data_type1 as
  2  object(Year number,
  3  member function Prod(Invest number)
  4  return number);
  5 /
```

Type created.

```
SQL> create type body data_type1 is
  2  member function Prod(Invest number)
  3  return number is
  4  begin
  5  return(Year+Invest);
  6  end;
  7  end;
  8 /
```

Type body created.

- Create Table ‘Data’ with Attribute ‘Col’ of type ‘data_type1’. Insert a value in the table. Display the result using Select statement.

Query:

✓ create table Data(Col data_type1); insert
into Data values(data_type1(2)); select
d.Col.Prod(12) from Data d;

Output:

```
SQL> create table Data(Col data_type1);
Table created.

SQL> insert into Data values(data_type1(2));
1 row created.

SQL> select d.Col.Prod(12) from Data d;
D.COL.PROD(12)
-----
14
```

- Create Object Type ‘Name’ with Object Attributes ‘FName’ and ‘LName’. Display the First and Last Name of a Person using Table ‘Person’.

Query:

✓ create type Name as
object(
FName varchar(10),
LName varchar(10));
/

Output:

```
SQL> create type Name as
 2 object(
 3   FName varchar(10),
 4   LName varchar(10));
 5 /
```

Type created.

- Creating Table ‘Person’ with Attribute ‘PName’ of type ‘Name’.
- Inserting values in the table ‘Person’.
- Displaying the Description of table ‘Person’.

Query:

- ✓ create table

```
Person(PName Name);
```

- ✓ insert into Person values(Name('Lekha','Naik'));

- ✓ desc Person;

```
SQL> create table
      2 Person(PName Name);
```

Output:

```
Table created.
```

```
SQL>
SQL> insert into Person values(Name('Lekha','Naik'));

1 row created.
```

```
SQL> desc Person;
```

Name	Null?	Type
PNAME		NAME

- Displaying First Name.
- Displaying First and Last Name.

Query:

- ✓ select p.PName.FName from Person p;

- ✓ select p.PName.FName || ' ' || p.PName.LName from Person p;

Output:

```
SQL> select p.PName.FName from Person p;
```

```
PNAME.FNAME
```

```
-----  
Lekha
```

```
SQL> select p.PName.FName || ' ' || p.PName.LName from Person p;
```

```
P.PNAME.FNAME||'|||P.
```

```
-----  
Lekha Naik
```

- Create Object Type ‘Address’ with Object Attributes ‘Street’ and ‘City’.
Display the Street and City of a person using Table ‘People’, also Display the Person ‘Name’ and ‘DOB’ using ‘Name’ and ‘Date’ Object Type.

Query:

- ✓ Creating Object Type ‘Address’ with Attributes ‘Street’ and ‘City’:

```
create type Address as
object(
  Street varchar(10),
  City varchar(10));
/
```

- ✓ Creating Table ‘People’:

```
create table
People(Name Name,
Addrs Address, DOB
Date);
```

- ✓ Inserting values in Table:

```
insert into People values(Name('Lekha','Naik'),
Address('Mira Road','Thane'),
to_date('02-06-1997','dd-mm-yyyy'));
```

```
SQL> create type Address as
 2  object(
 3    Street varchar(10),
 4    City varchar(10));
 5 /
Type created.

SQL>
SQL> create table
 2  People(Name Name,
 3  Addrs Address,
 4  DOB Date);

Table created.

SQL>
SQL> insert into People values(Name('Lekha','Naik'),
 2  Address('Mira Road','Thane'),
 3  to_date('02-06-1997','dd-mm-yyyy'));

1 row created.
```

✓ Displaying the Table:

```
select * from People;
```

✓ Displaying the First and Last Name:

```
select pd.Name.FName || ' ' || pd.Name.LName from People pd;
```

✓ Displaying the ‘Street’ and ‘City’:

```
select pd.Addrs.Street || ' ' || pd.Addrs.City from People pd;
```

✓ Displaying the ‘Date of Birth(DOB)’:

```
select DOB from People;
```

```
SQL> select * from People;
```

```
NAME(FNAME, LNAME)
```

```
ADDRS(STREET, CITY)
```

```
DOB
```

```
NAME('Lekha', 'Naik')
ADDRESS('Mira Road', 'Thane')
02-JUN-97
```

```
SQL>
```

```
SQL> select pd.Name.FName || ' ' || pd.Name.LName from People pd;
```

```
PD.NAME.FNAME||'|||PD
```

```
Lekha Naik
```

```
SQL>
```

```
SQL> select pd.Addrs.Street || ' ' || pd.Addrs.City from People pd;
```

```
PD.ADDRS.STREET||'|||
```

```
Mira Road Thane
```

```
SQL>
```

```
SQL> select DOB from People;
```

```
DOB
```

```
02-JUN-97
```

- Create Object Type ‘Demo’ with Object Attribute ‘ID’ and Function ‘get_square’. Use this Function to return the Square of ID attribute value.

Query:

- Creating Object Type ‘Demo’ with Attribute ‘ID’ and Function ‘get_square’:

```
create type Demo as
object(ID number,
member function get_square
return number);
/
```

- Creating Table ‘Demo_Tbl’ with Attribute ‘Col’ of type ‘Demo’:

```
create table
Demo_Tbl(Col Demo);
```

- Inserting Values in the Table:

```
insert into Demo_Tbl values(Demo(3));
```

- Output:**

```
SQL> create type Demo as
 2 object(ID number,
 3 member function get_square
 4 return number);
 5 /
```

Type created.

```
SQL>
SQL> create table
 2 Demo_Tbl(Col Demo);
```

Table created.

```
SQL>
SQL> insert into Demo_Tbl values(Demo(3));
1 row created.
```

- ✓ Creating type body to return the ‘Square of ID’ using ‘get_square’ function:

```
create type body Demo
is member function get_square
return number
is n number;
begin
select s.Col.ID*s.Col.ID into n
from Demo_Tbl s;
return(n); end;
end;
/
```

- ✓ Displaying the Square of ID attribute value:

```
select v.Col.get_square() from Demo_Tbl v;
```

Output:

```
SQL> create type body Demo
  2  is member function get_square
  3  return number
  4  is n number;
  5  begin
  6  select s.Col.ID*s.Col.ID into n
  7  from Demo_Tbl s;
  8  return(n);
  9  end;
10 end;
11 /
```

Type body created.

```
SQL>
SQL> select v.Col.get_square() from Demo_Tbl v;
V.COL.GET_SQUARE()
-----
```

Practical No. 04

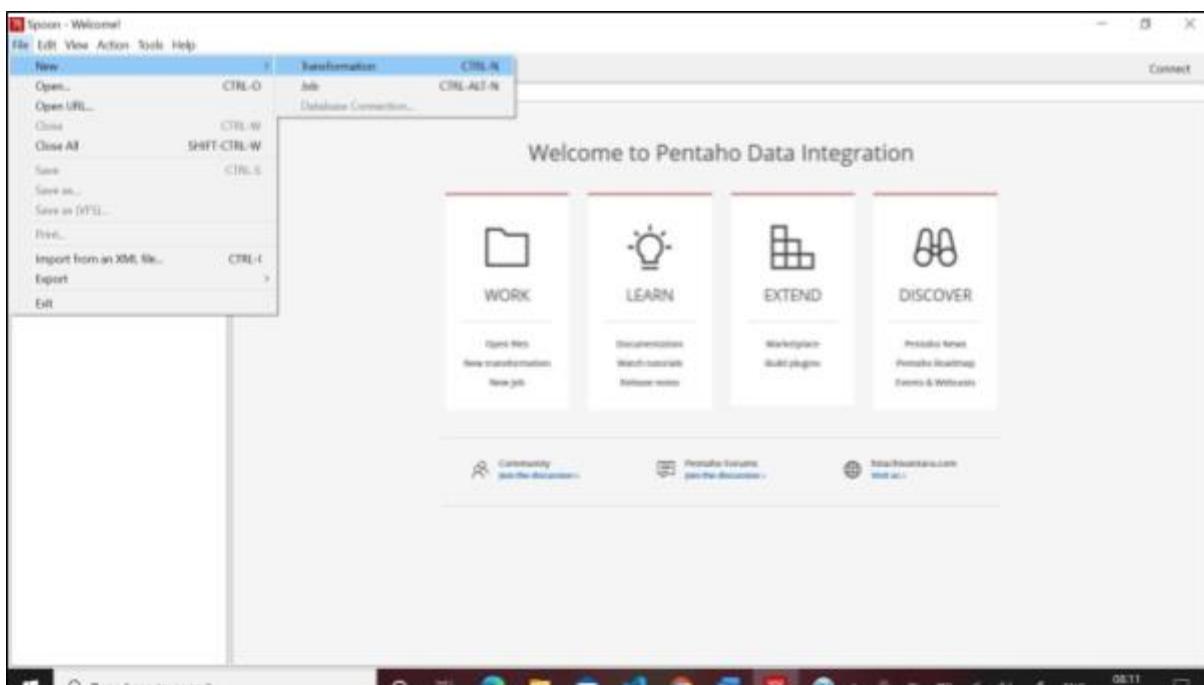
Aim: Implementation of ETL Transformation with Pentaho.

- Transforming Source Table and Storing to Output Table in SQL.

Steps:

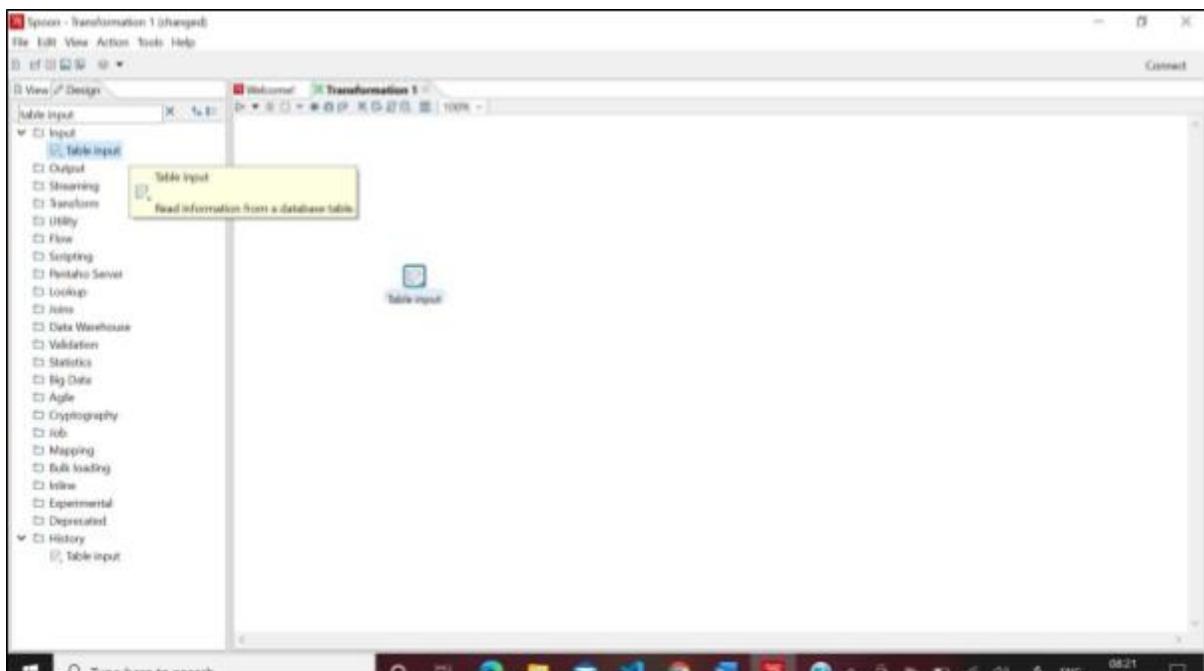
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



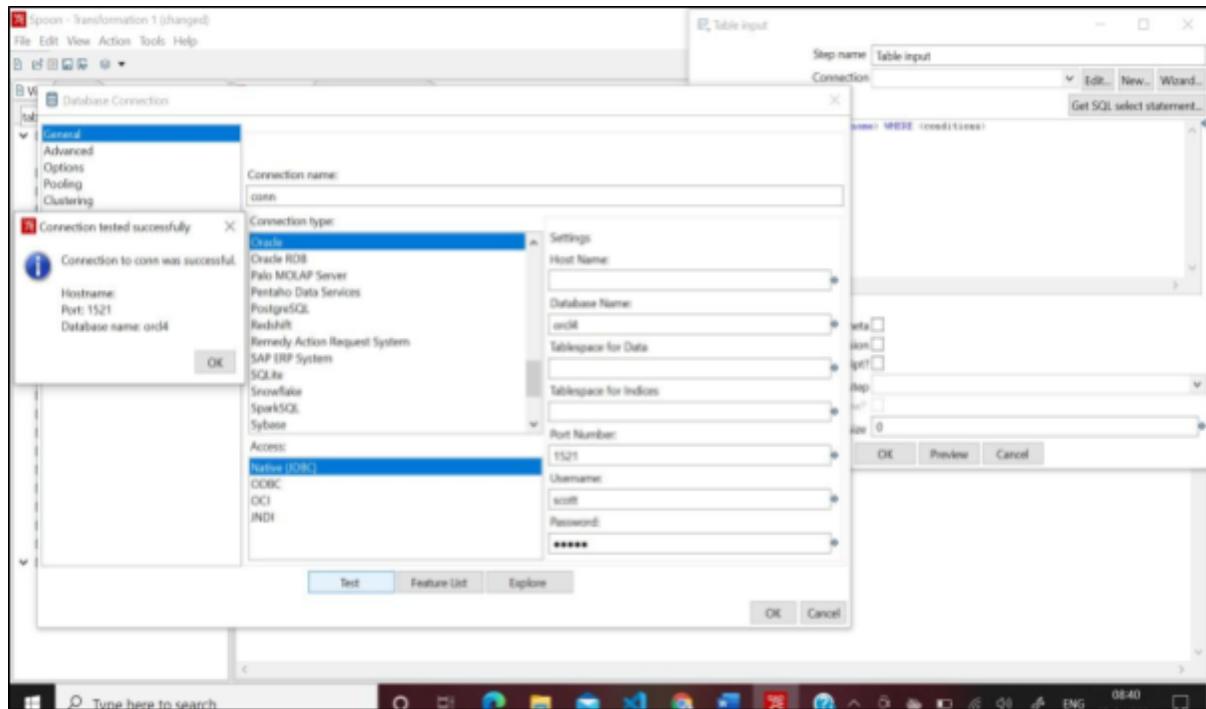
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



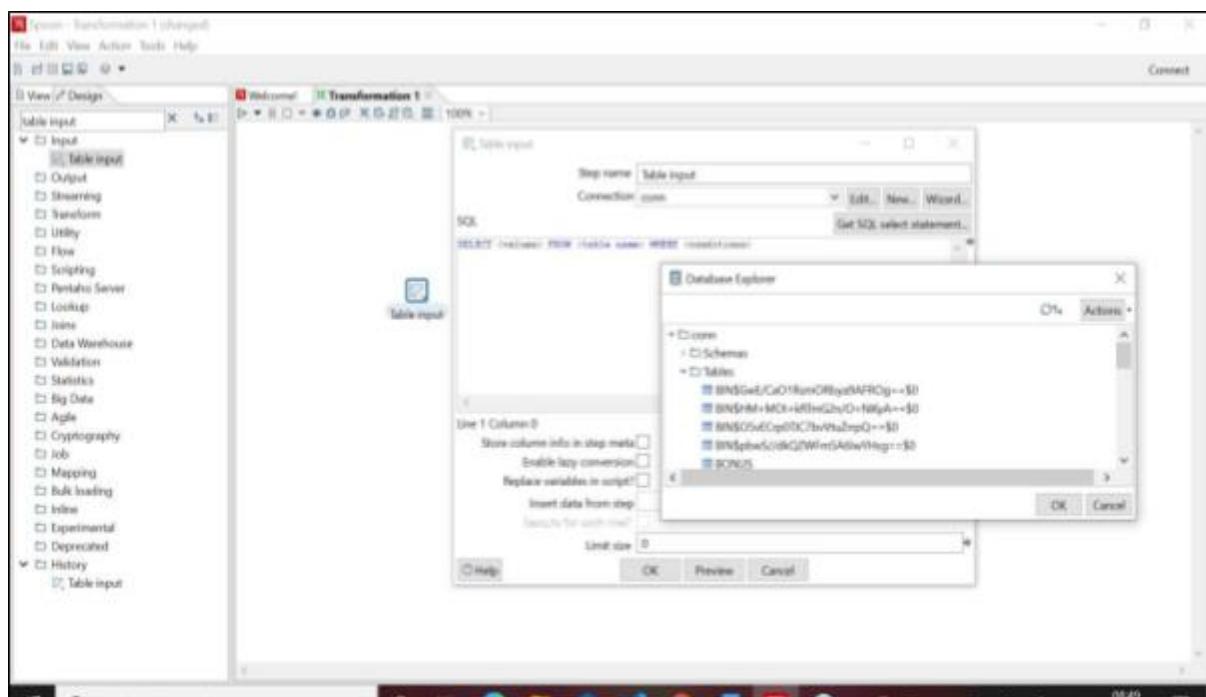
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle Access: Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



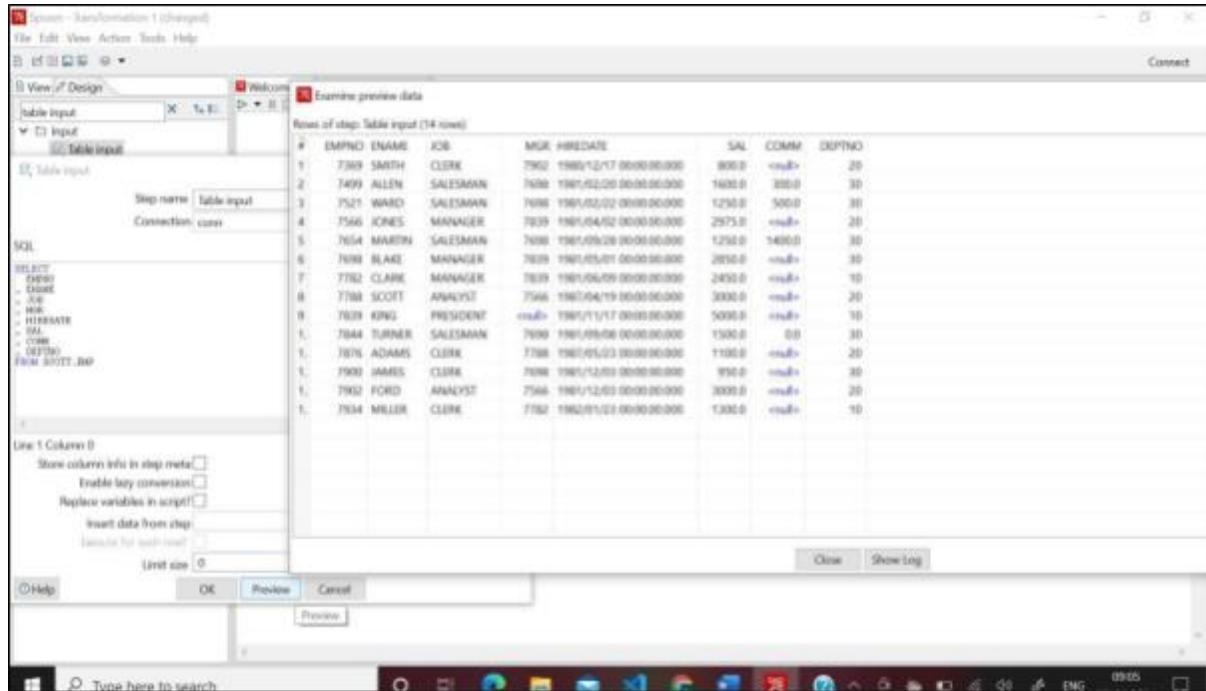
✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

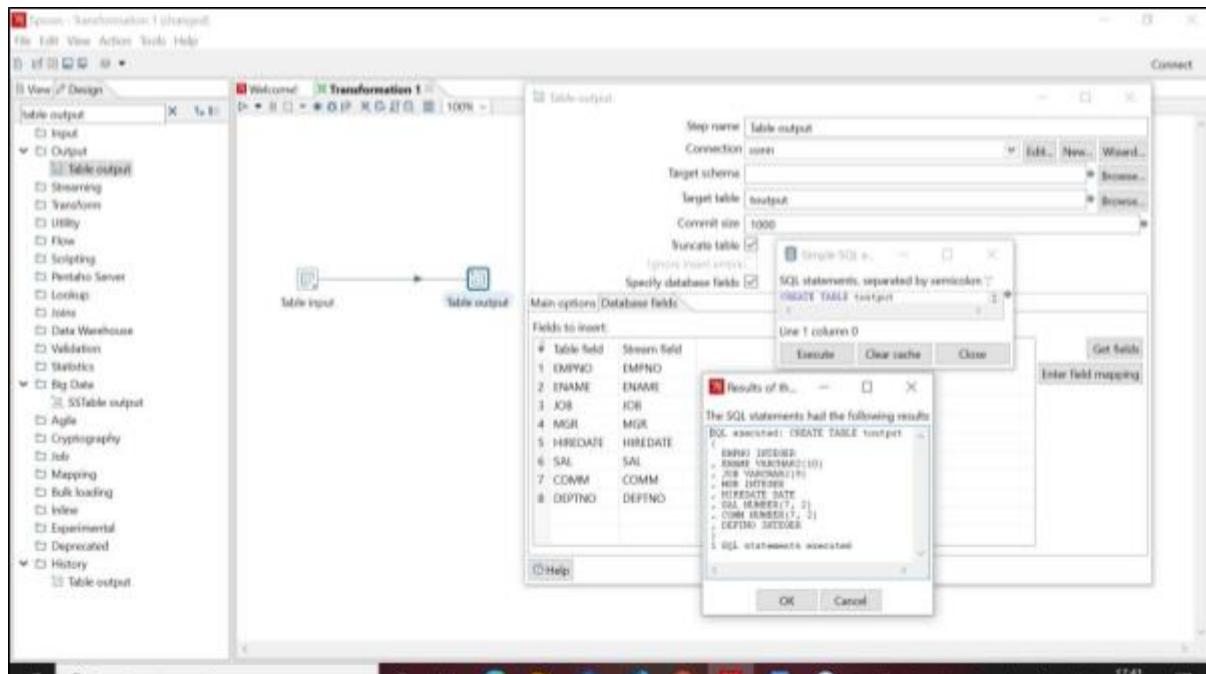
Table Input -> Preview -> OK -> Close -> OK



✓ Storing to Output Table:

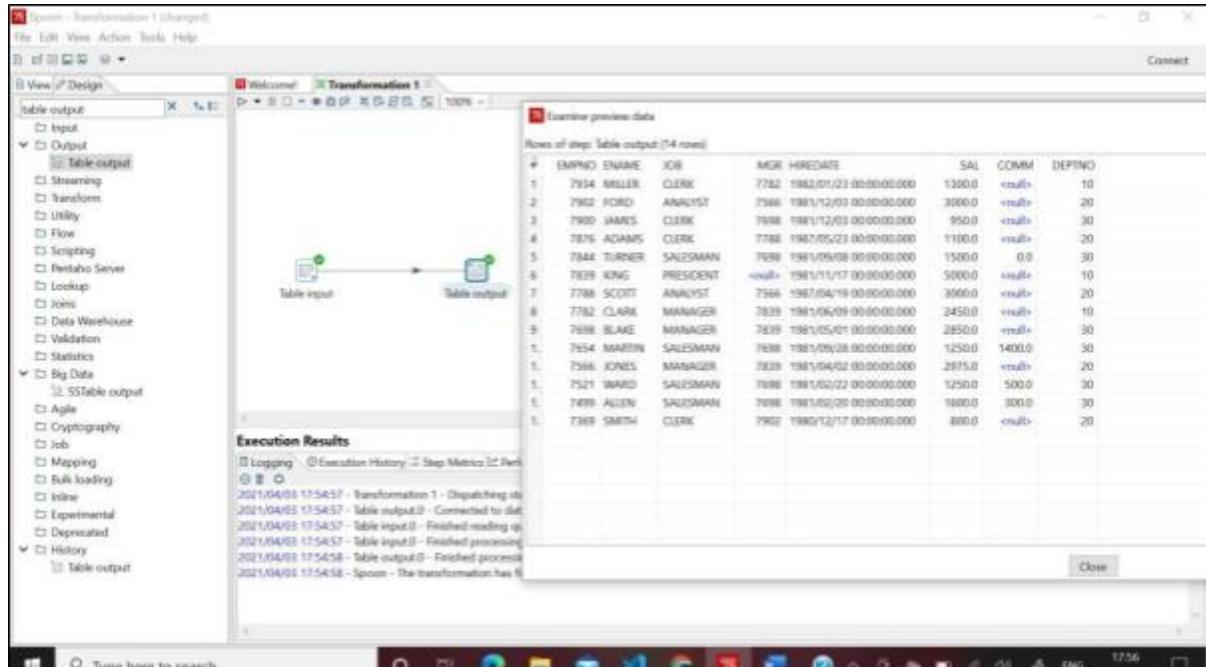
Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Input Table)

Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> Execute -> OK -> Close



✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes



✓ View Output Table in SQL:

SQL Plus -> select * from output;

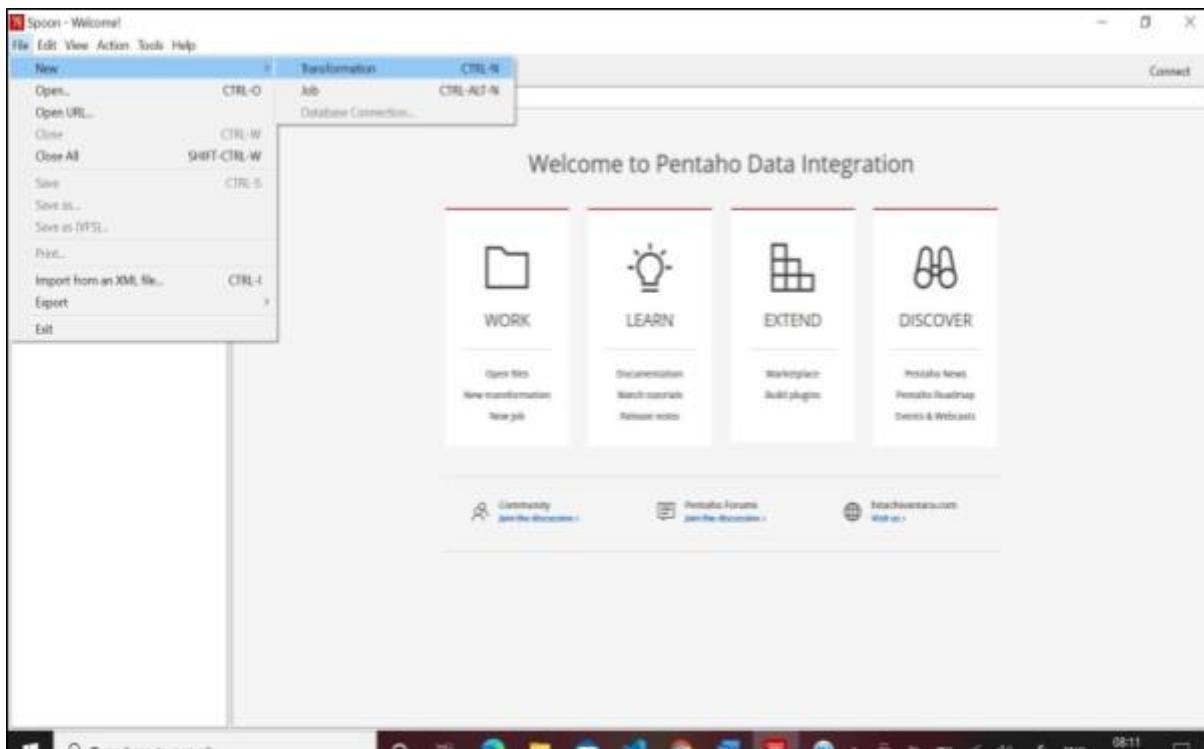
SQL> select * from toutput;						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
7369	SMITH	CLERK	7902	17-DEC-80	800	
20						
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30						
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
7566	JONES	MANAGER	7839	02-APR-81	2975	
20						
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30						
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	
30						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
7782	CLARK	MANAGER	7839	09-JUN-81	2450	
10						
7788	SCOTT	ANALYST	7566	19-APR-87	3000	
20						
7839	KING	PRESIDENT		17-NOV-81	5000	
10						

- Implementation of Sorting Operation and Adding Sequence.

Steps:

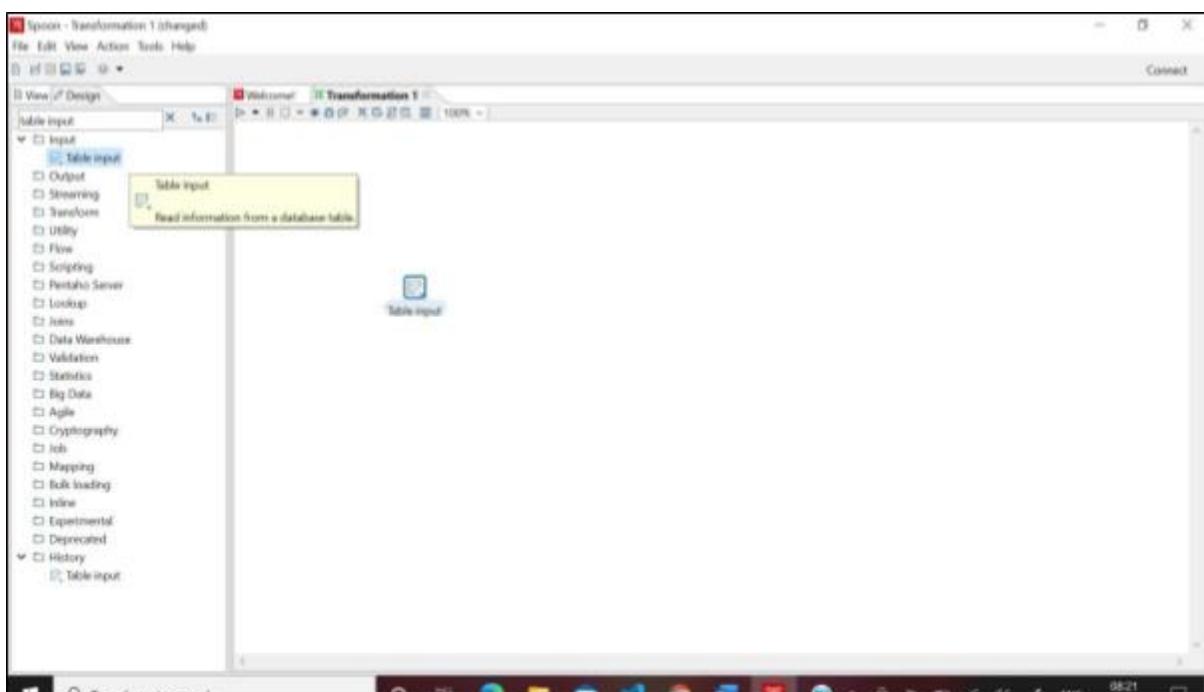
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



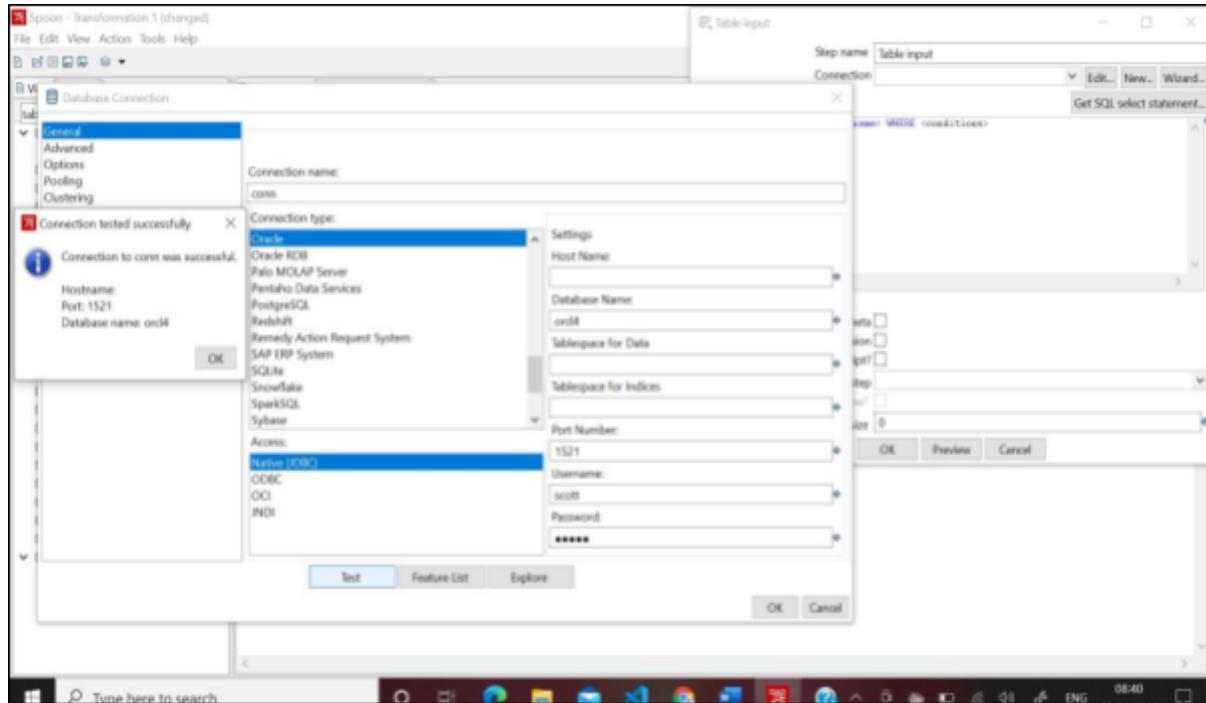
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



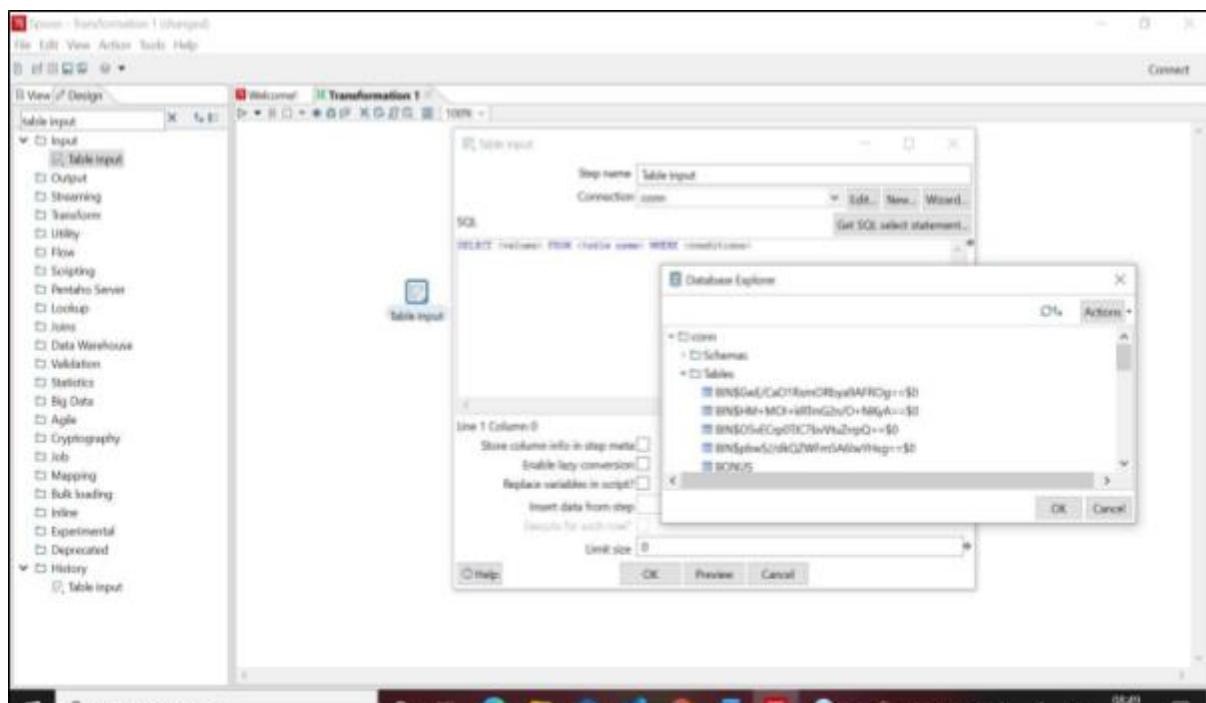
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



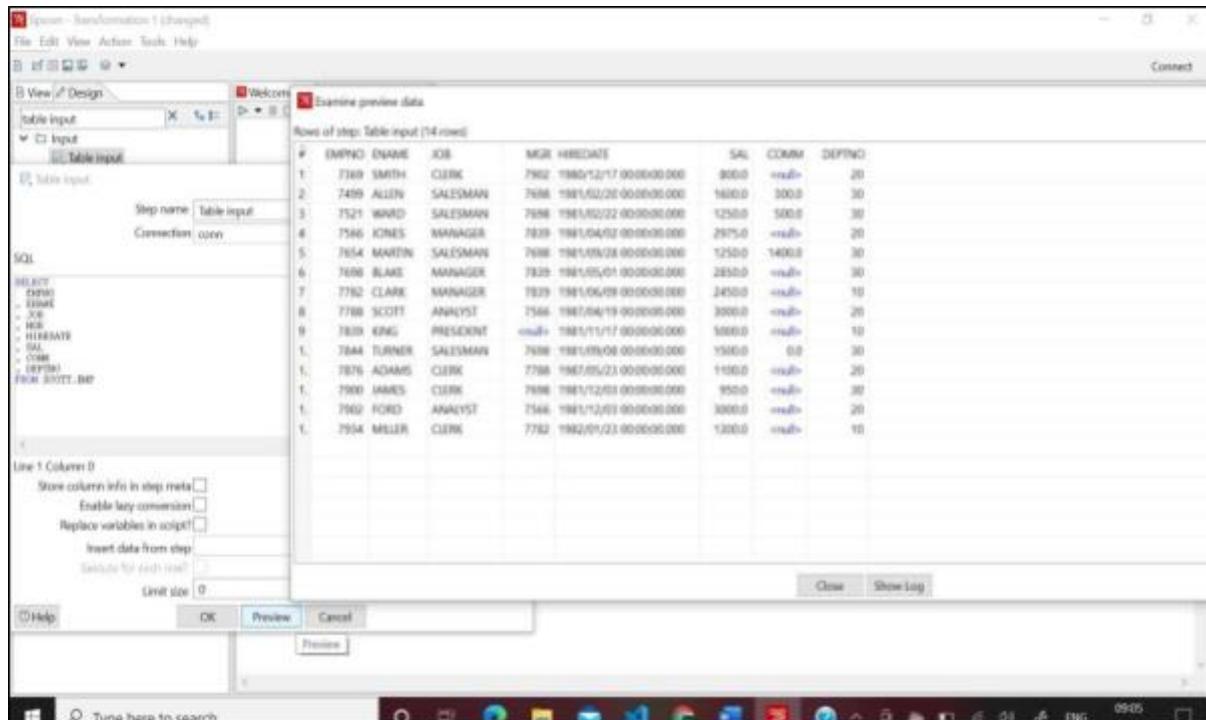
✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

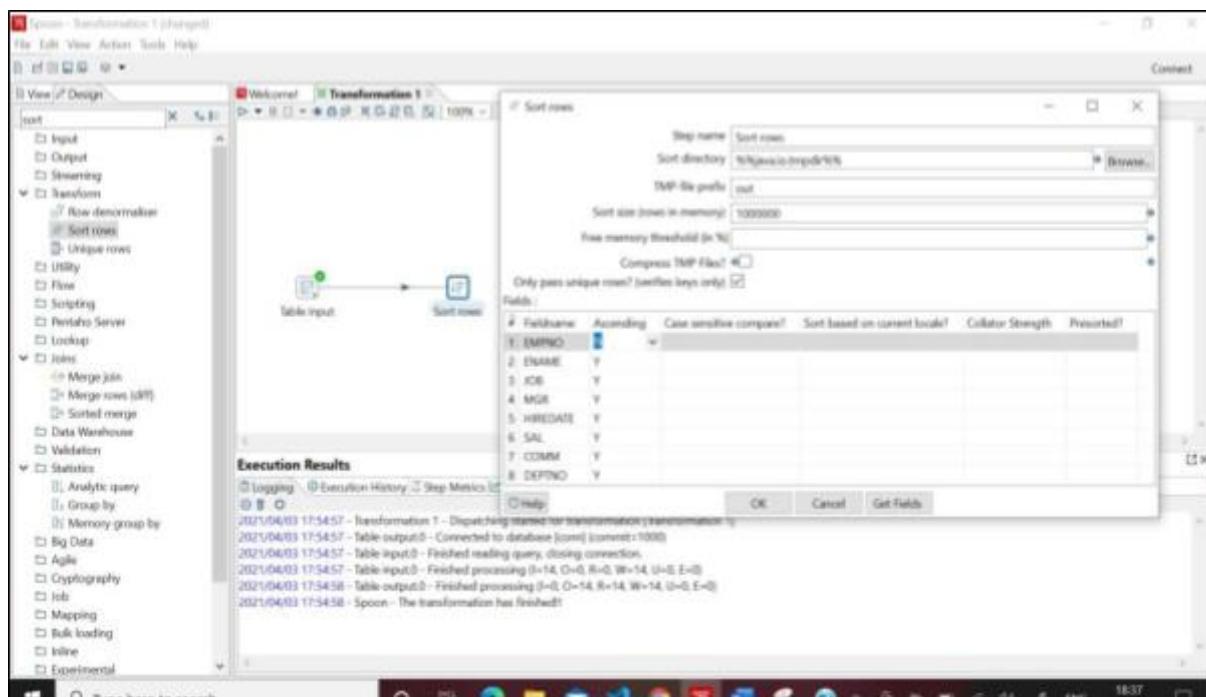
Table Input -> Preview -> OK -> Close -> OK



✓ Sorting Operation:

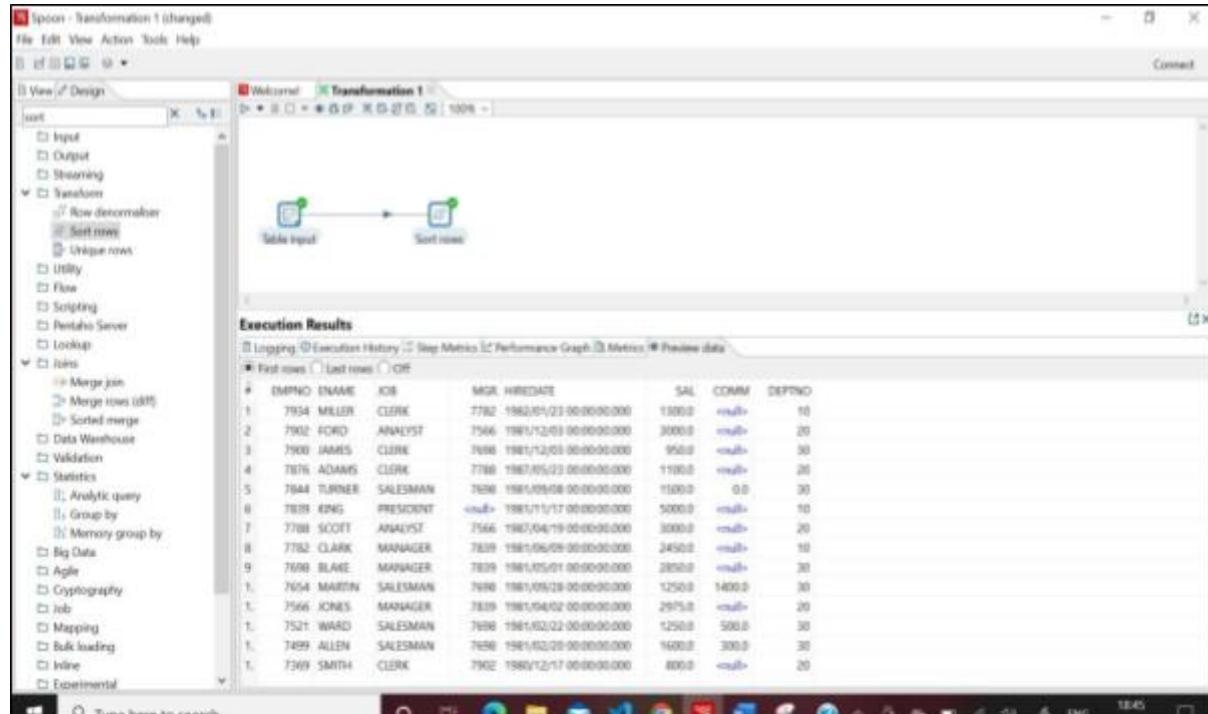
Design -> Transform -> Sort Rows (Drag & Drop in Transformation) -> Create Hop Connection with Sort Rows (Click on Output Connector arrow in Input Table)

Double Click Sort Rows -> Only pass unique rows? -> Get Fields -> Set desired Order in Fields -> OK



✓ Debug Transformation:

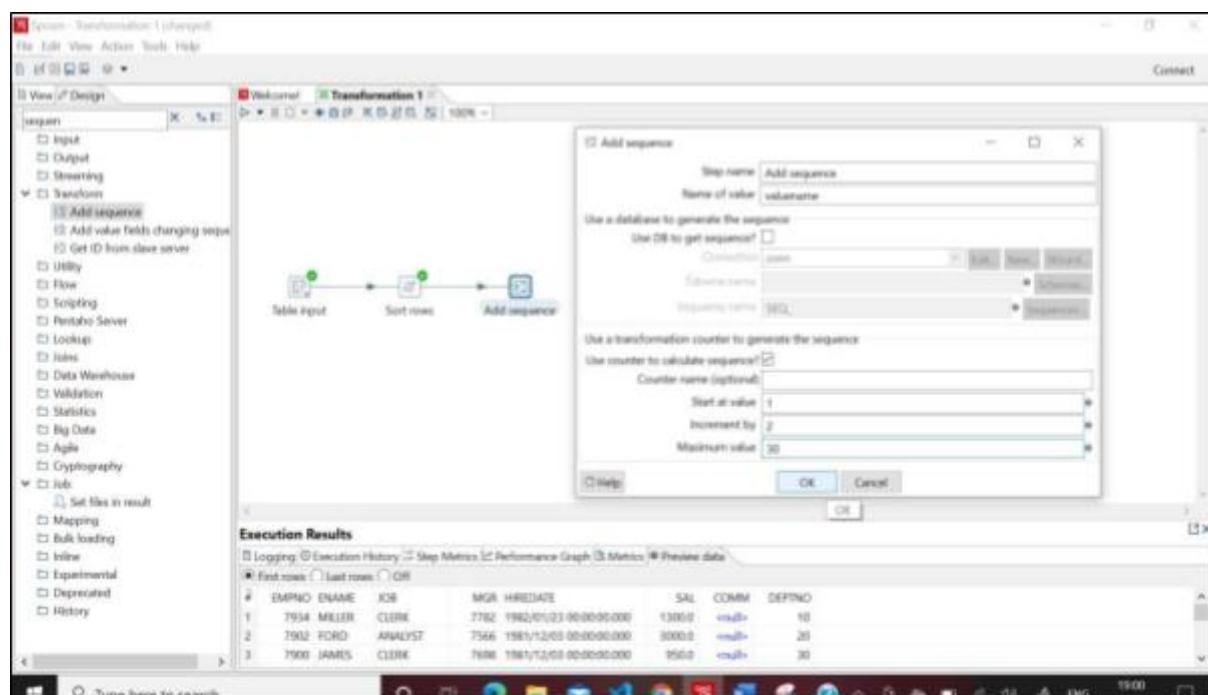
Click Sort Rows -> Debug -> Quick Launch -> Yes



✓ Adding Sequence:

Design -> Transform -> Add Sequence (Drag & Drop in Transformation) -> Create Hop Connection with Add Sequence (Click on Output Connector arrow in Sort Rows)

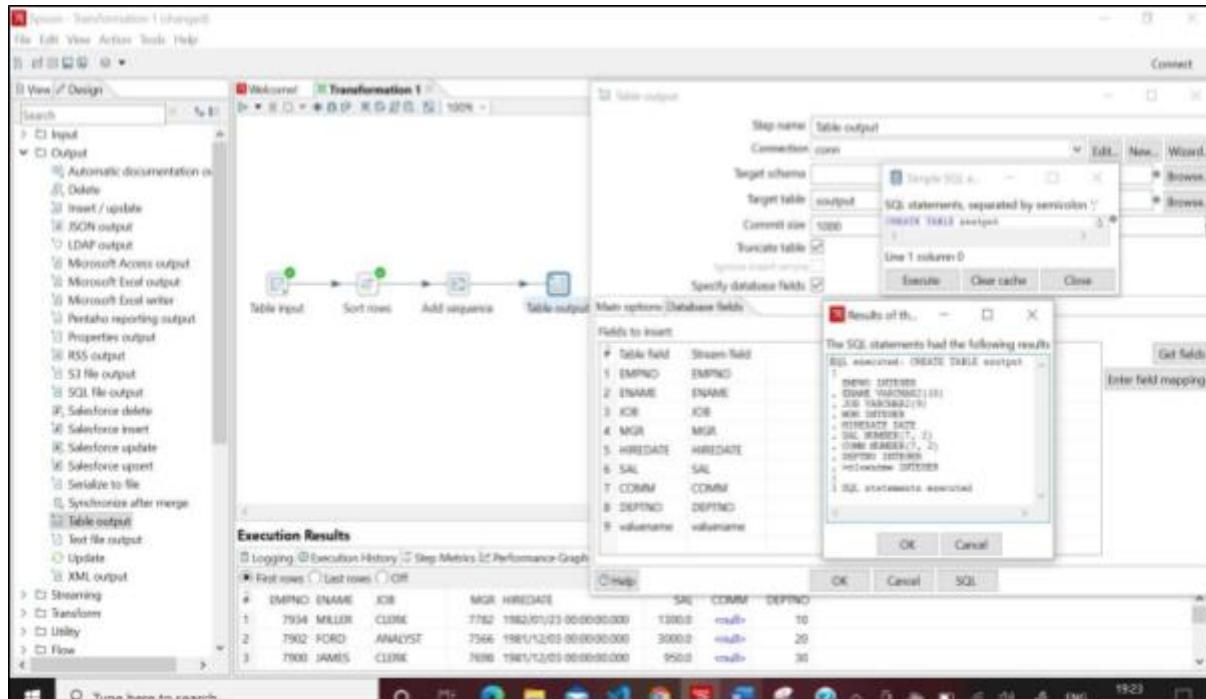
Double Click Add Sequence -> Start at value: 1 -> Increment by: 2 -> Maximum value: 30 -> OK



✓ Storing to Output Table:

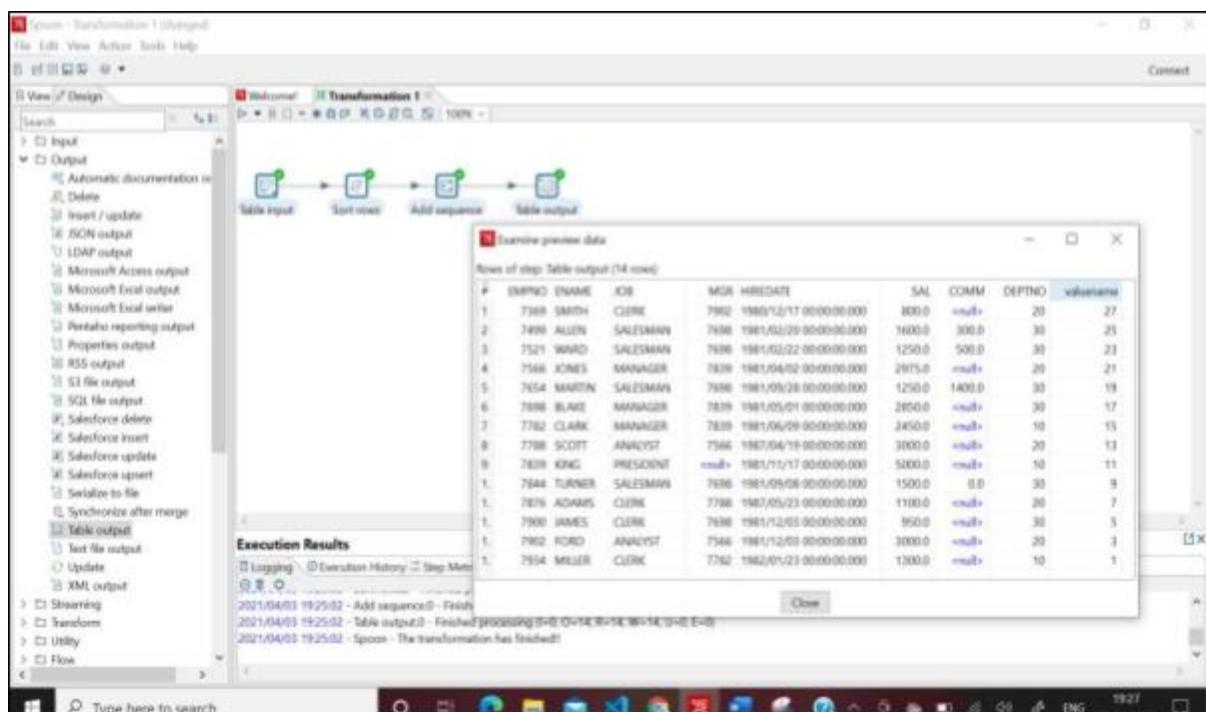
Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Add Sequence)

Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close



✓ Debug Transformation:

Click Sort Rows -> Debug -> Quick Launch -> Yes

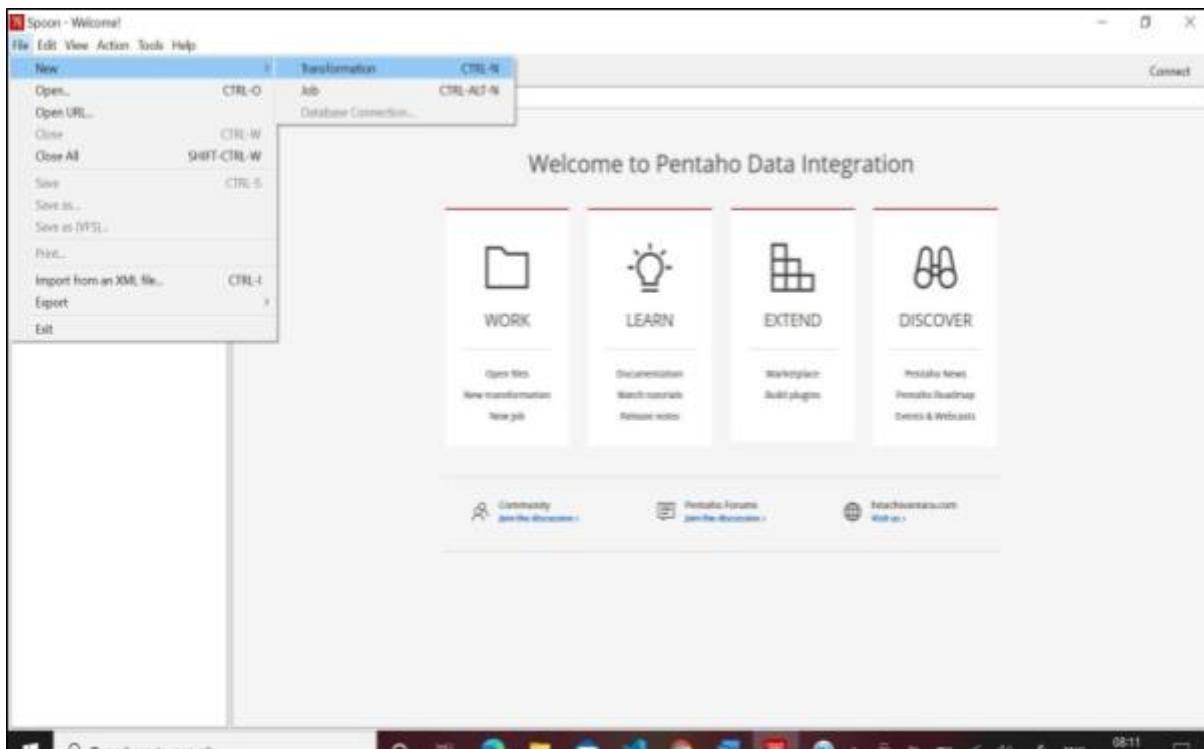


- Implementation of Calculator Operation.

Steps:

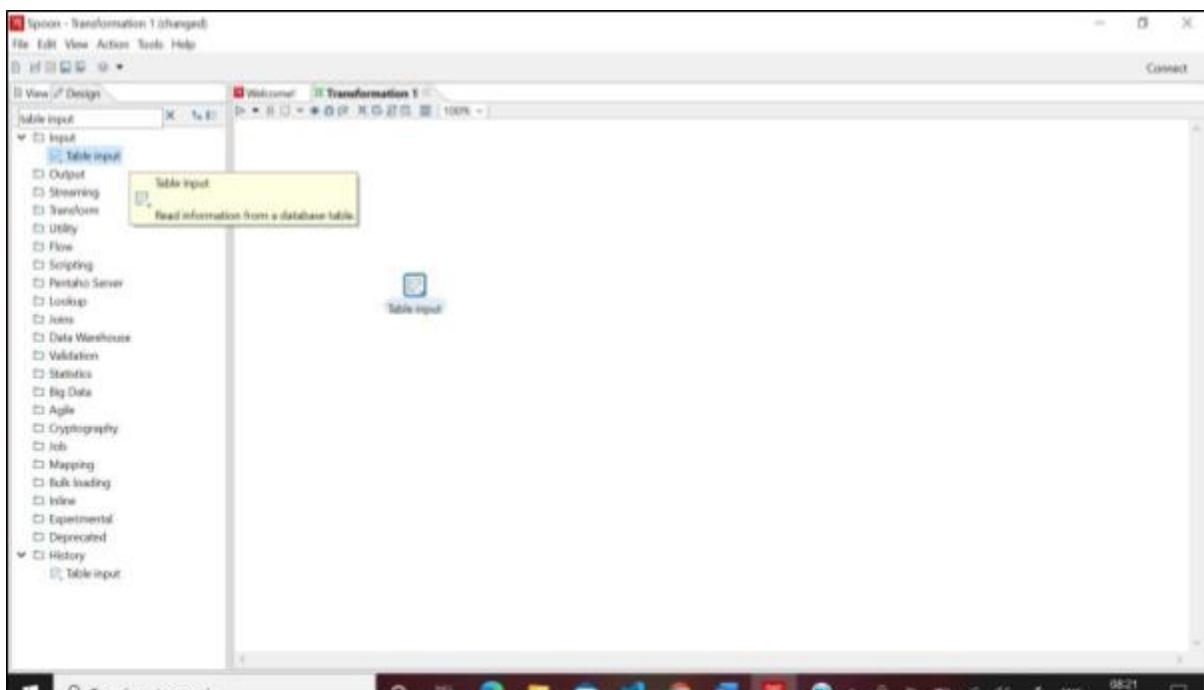
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



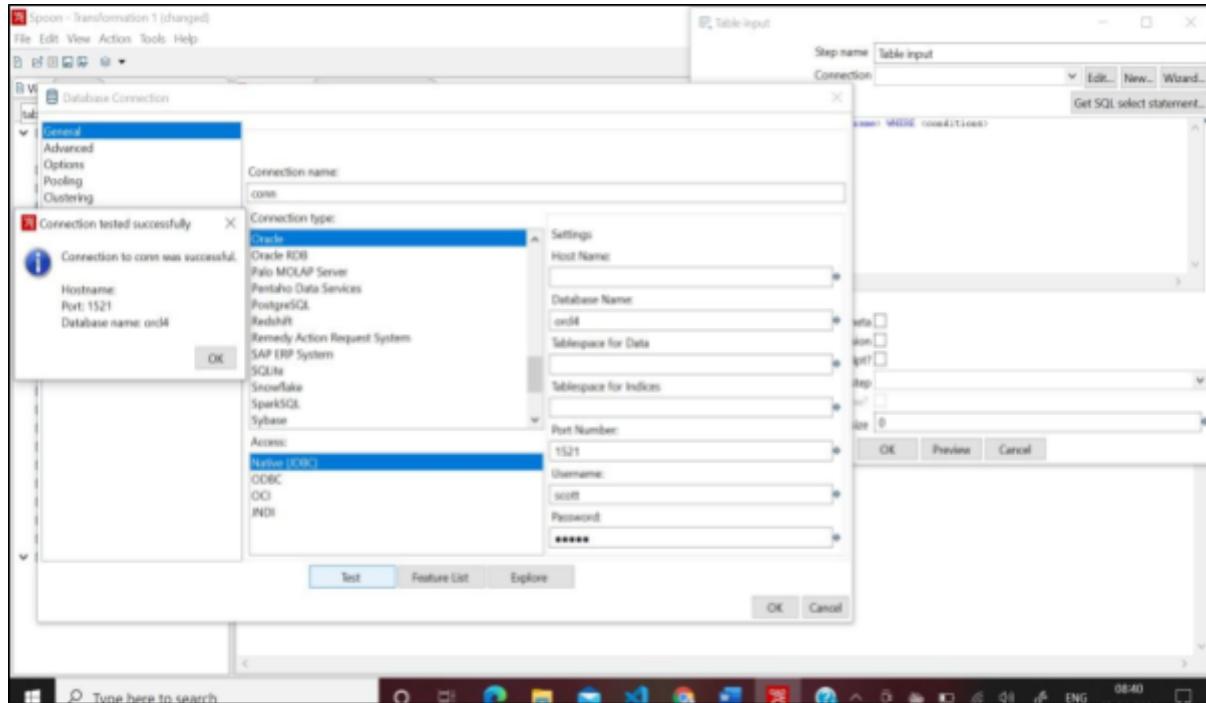
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



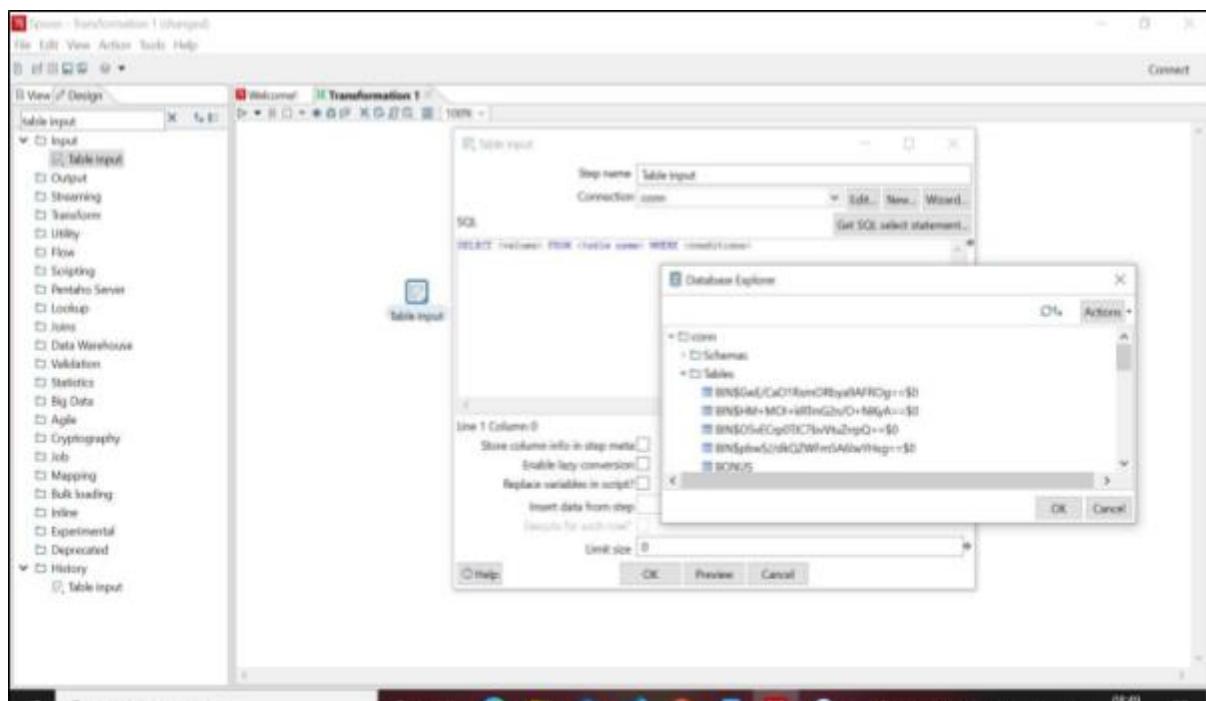
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



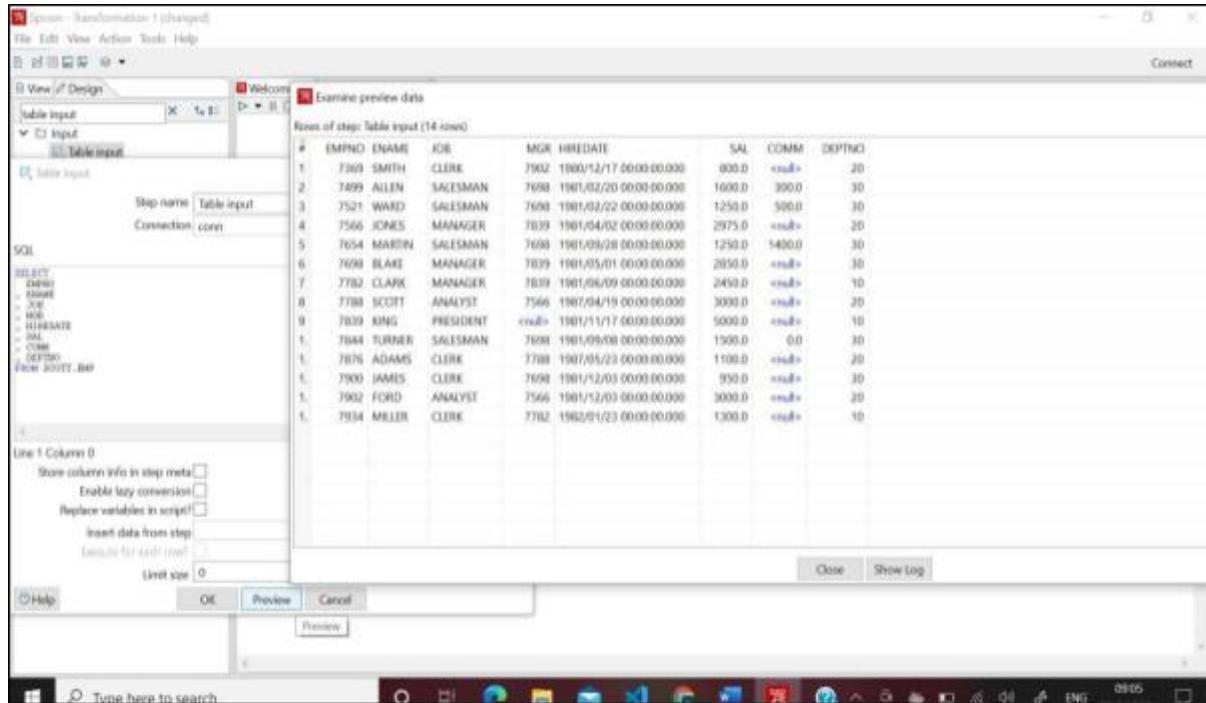
✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

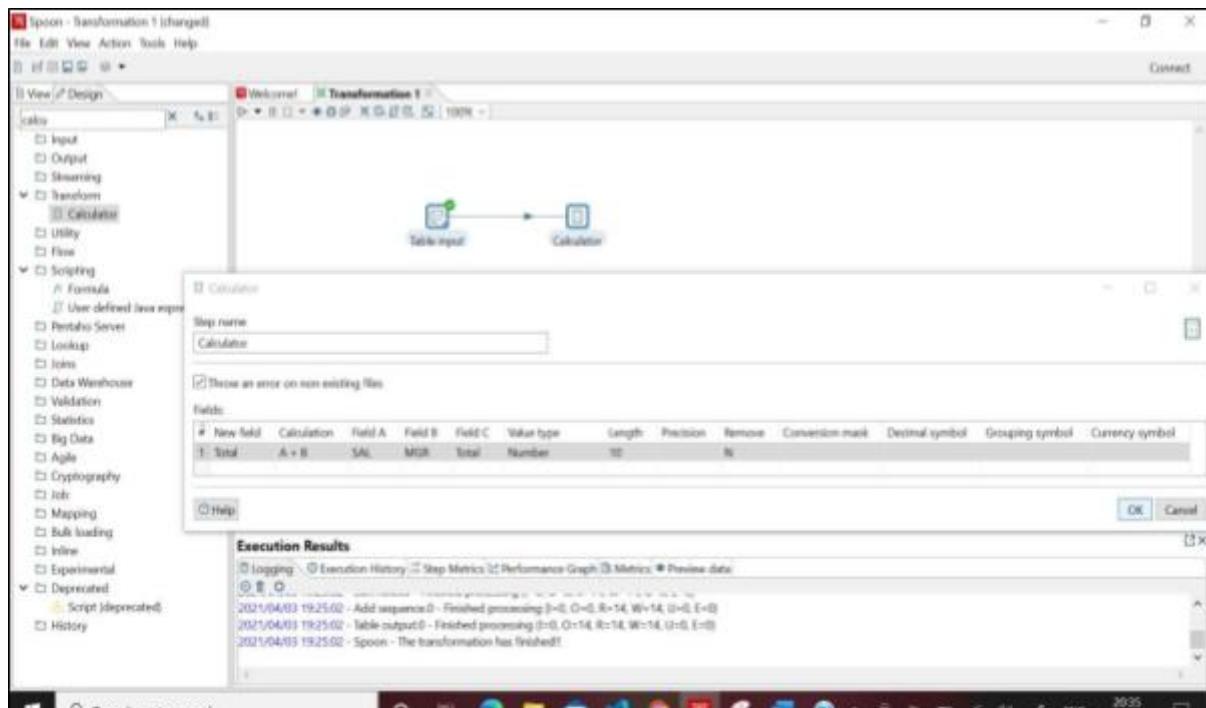
Table Input -> Preview -> OK -> Close -> OK



✓ Calculator Operation:

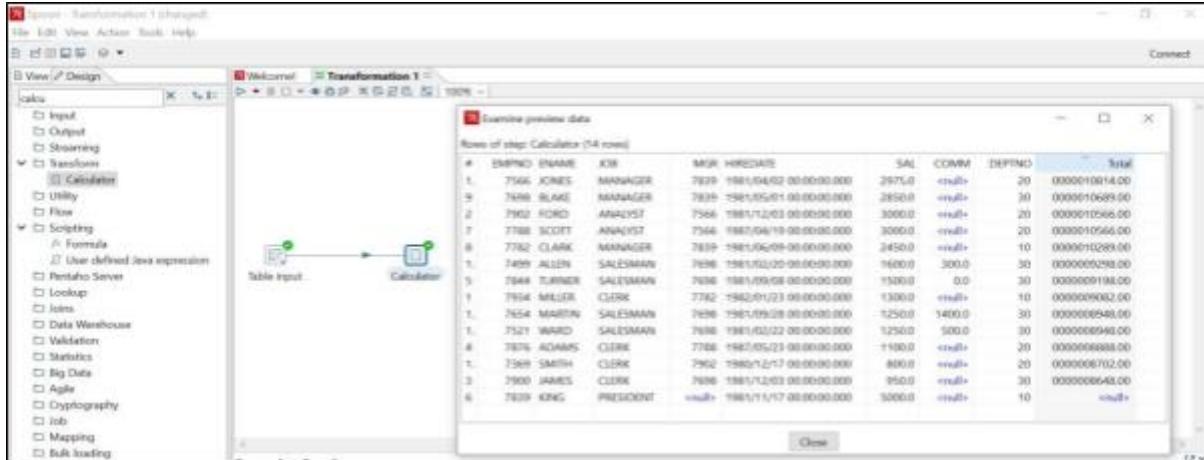
Design -> Transform -> Calculator (Drag & Drop in Transformation) -> Create Hop Connection with Calculator (Click on Output Connector arrow in Input Table)

Double Click Calculator -> New field: *Total* -> Calculation: *A+B* -> Field A: *SAL* -> Field B: *MGR* -> Field C: *Total* -> Value type: Number -> Length: *10* -> Remove: *N* -> OK



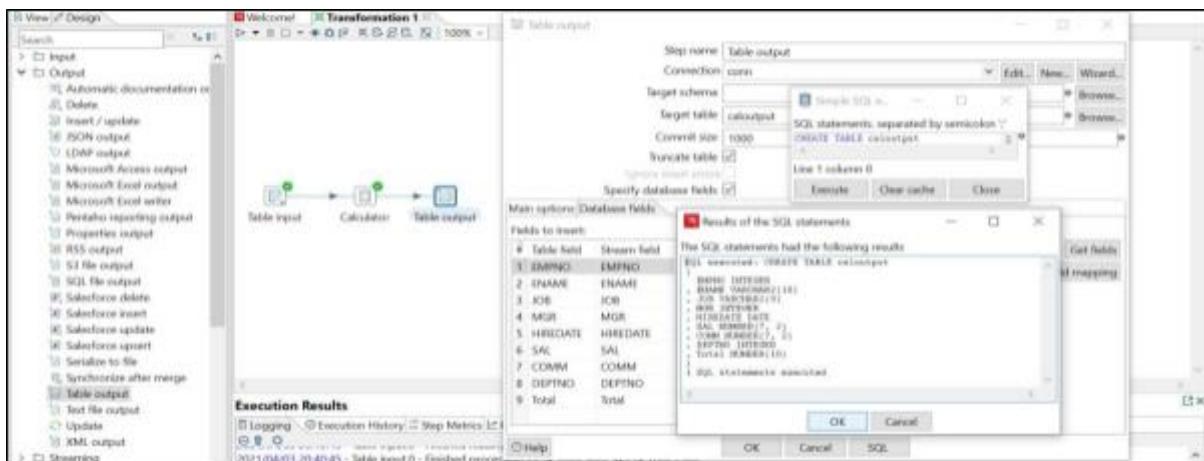
✓ Debug Transformation:

Click Calculator -> Debug -> Quick Launch -> Yes



✓ Storing to Output Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Calculator) Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close



✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes

The screenshot shows the Talend Data Integration environment. On the left, the 'View of Design' sidebar lists various output types, including Table output, which is selected. The main workspace displays a transformation flow with three steps: Table Input, Calculator, and Table output. The Table output step is highlighted with a green border. A preview window titled 'Preview previous data' is open, showing 14 rows of data from the Table output step. The data includes columns such as EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO, and Total. The execution results window at the bottom shows logs indicating the successful completion of the transformation.

#	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	Total
1	7564	MILLER	CLERK	7782	1982/01/23 00:00:00.000	1300.0	NULL	10	0000000032.00
2	7602	PORD	ANALYST	7566	1981/12/03 00:00:00.000	3100.0	NULL	20	00000017568.00
3	7693	JAMES	CLERK	7690	1981/12/03 00:00:00.000	950.0	NULL	30	00000008643.00
4	7679	ADAMS	CLERK	7686	1981/01/01 00:00:00.000	1100.0	NULL	20	00000008888.00
5	7844	TURNER	SALESMAN	7690	1981/09/08 00:00:00.000	1500.0	0.0	30	00000009198.00
6	7839	ING	PRESIDENT	NULL	1981/11/17 00:00:00.000	5000.0	NULL	10	00000005000.00
7	7789	SCOTT	ANALYST	7566	1982/06/19 00:00:00.000	3000.0	NULL	20	00000017566.00
8	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00.000	2450.0	NULL	10	00000010299.00
9	7690	BLAKE	MANAGER	7690	1981/05/01 00:00:00.000	2850.0	NULL	30	00000010699.00
10	7654	MARTIN	SALESMAN	7690	1981/08/28 00:00:00.000	1250.0	1400.0	30	00000008948.00
11	7598	JONES	MANAGER	7839	1981/04/02 00:00:00.000	2975.0	NULL	20	00000010814.00
12	7524	WARD	SALESMAN	7498	1981/02/02 00:00:00.000	1250.0	500.0	30	00000008948.00
13	7499	ALLEN	SALESMAN	7498	1981/02/09 00:00:00.000	1600.0	300.0	30	00000009298.00
14	7360	SMITH	CLERK	7852	1980/12/17 00:00:00.000	800.0	NULL	20	00000008752.00

Execution Results

Logging Execution History Step Metrics Performance Graph Metrics Preview data

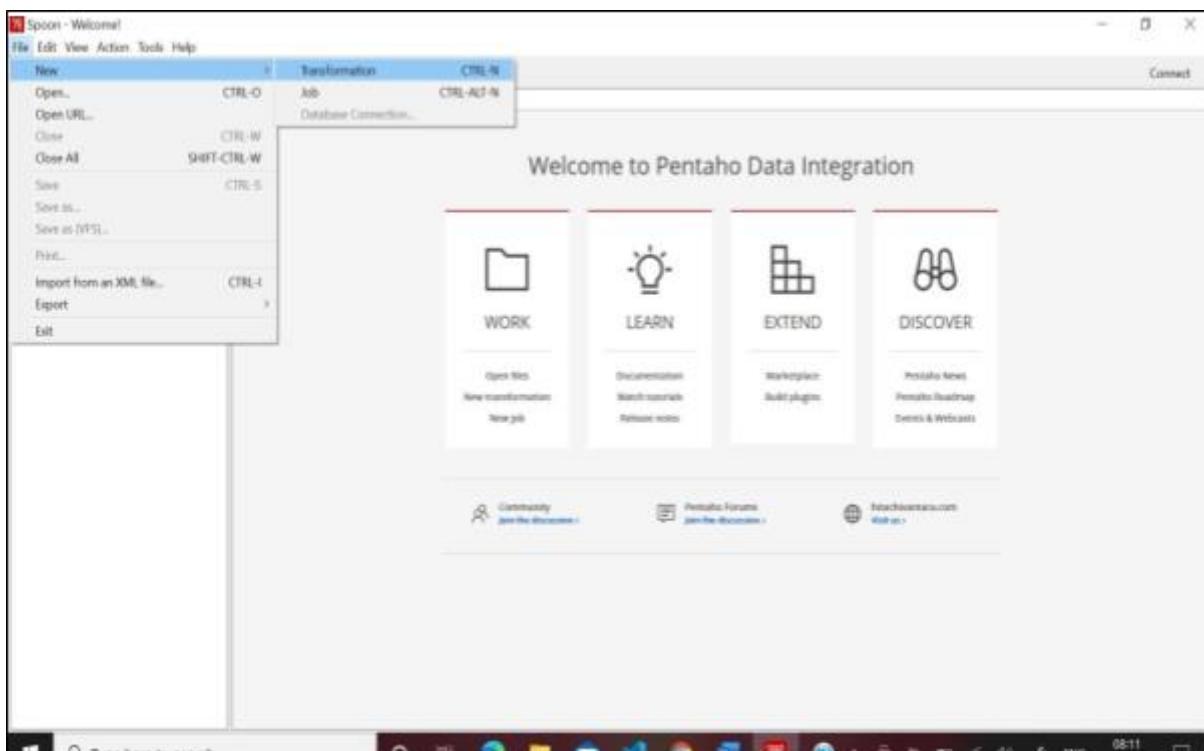
2021/04/03 21:29:59 - Calculator[0] - Finished processing 0=0, O=0, R=14, W=14, U=0, E=0
2021/04/03 21:29:59 - Table output[0] - Finished processing I=0, O=14, R=14, W=14, U=0, E=0
2021/04/03 21:29:59 - Spoon - The transformation has finished!

- Implementation of Concatenation Operation.

Steps:

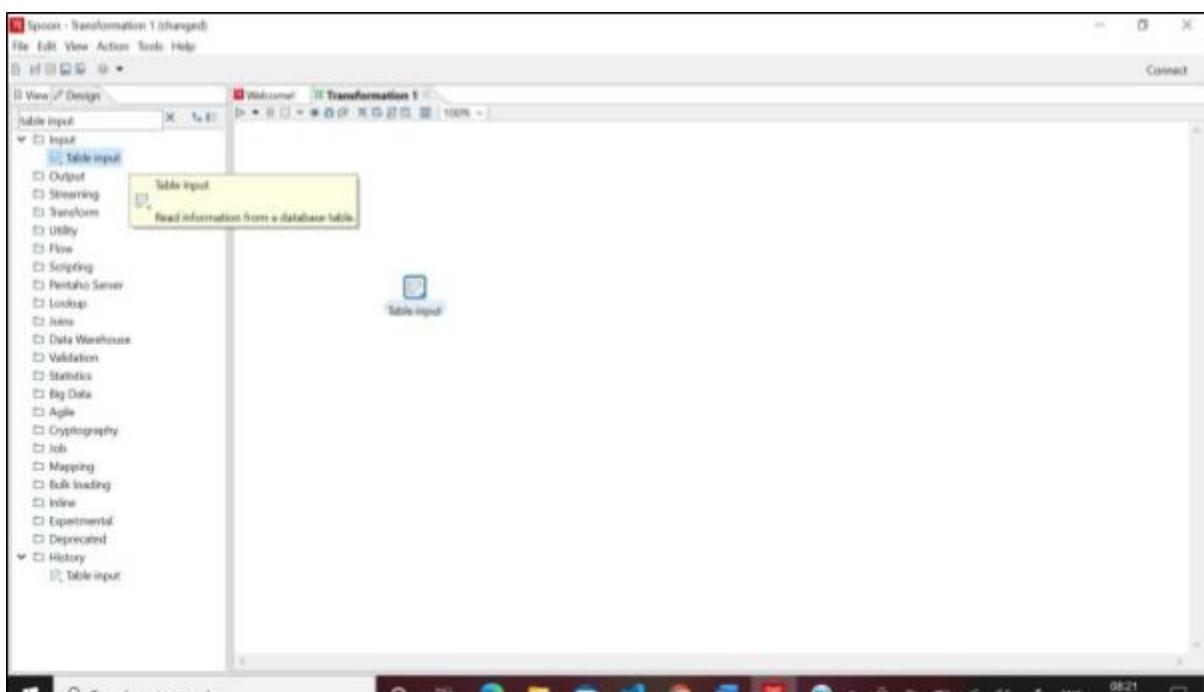
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



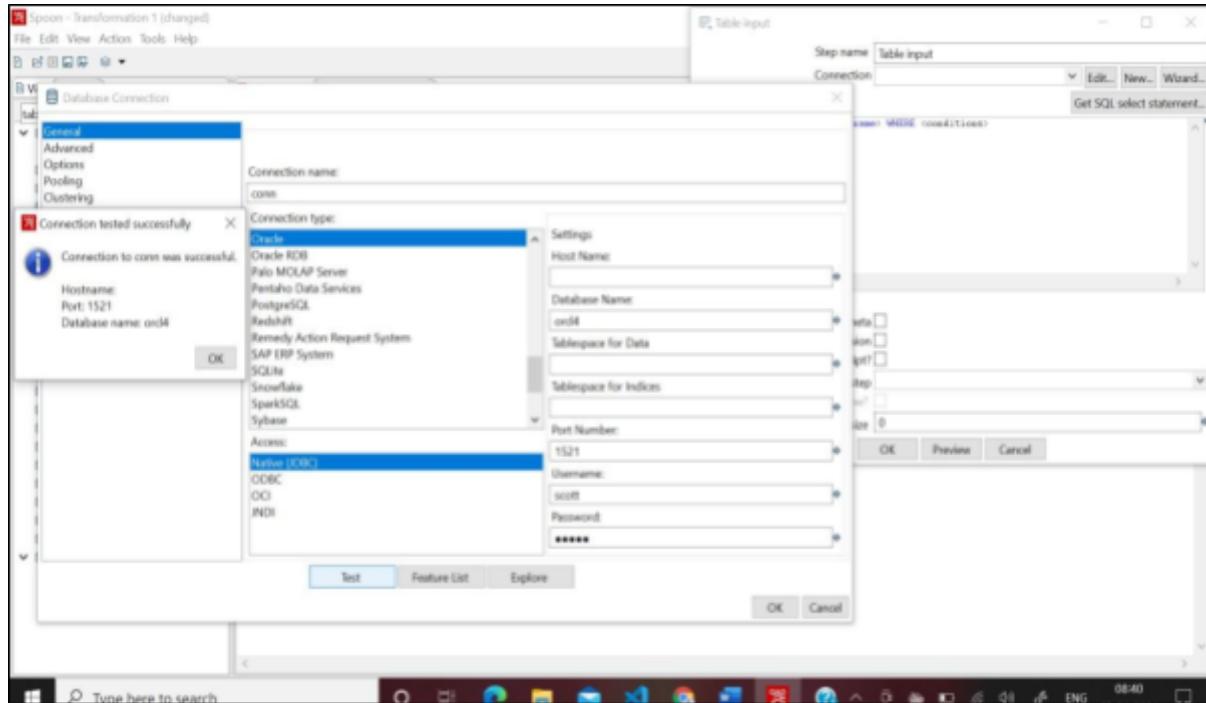
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



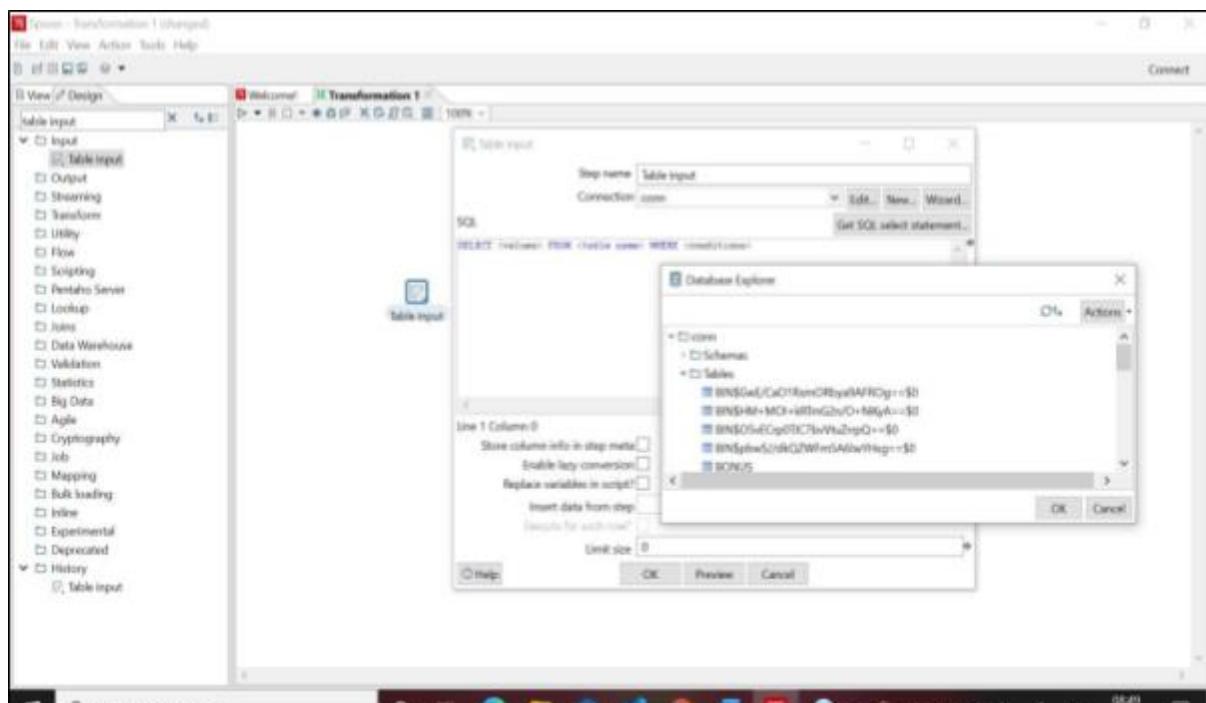
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

Table Input -> Preview -> OK -> Close -> OK

The screenshot shows the Talend Open Studio interface with the 'Table Input' step selected. The 'View / Design' tab is active, and the 'Preview' tab is open, showing 14 rows of data from the 'SCOTT.DBF' file. The data is as follows:

#	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1.	7369	SMITH	CLERK	1980-12-17	800.0	<null>	20
2.	7499	ALLLEN	SALESMAN	1981-02-20	1600.0	300.0	30
3.	7521	WARD	SALESMAN	1981-02-22	1250.0	500.0	30
4.	7566	KING	MANAGER	1981-04-01	2975.0	<null>	10
5.	7654	MARTIN	SALESMAN	1981-05-28	1250.0	1400.0	30
6.	7680	BLAKE	MANAGER	1981-05-01	2850.0	<null>	30
7.	7782	CLARK	MANAGER	1981-06-09	2450.0	<null>	10
8.	7788	SCOTT	ANALYST	1987-04-19	3000.0	<null>	20
9.	7829	KING	PRESIDENT	<null>	5000.0	<null>	10
10.	7844	TURNER	SALESMAN	1981-08-06	1300.0	0.0	30
11.	7876	ADAMS	CLERK	1981-05-23	1150.0	<null>	20
12.	7900	JAMES	CLERK	1981-12-03	950.0	<null>	30
13.	7965	FORD	ANALYST	1981-12-01	3000.0	<null>	20
14.	7954	MILLER	CLERK	1982-01-23	1300.0	<null>	10

✓ Concatenation Operation:

Design-> Transform-> Concat Fields (Drag & Drop in Transformation) -> Create Hop Connection with Concat Fields (Click on Output Connector arrow in Input Table)

Double Click Concat Fields -> Target Field Name: *Emp_Data* -> Separator: _ -> Get Fields (Discard unwanted Fields) -> OK

The screenshot shows the Talend Open Studio interface with the 'Concat Fields' step selected in the 'Transformation 1' panel. The 'Fields' tab is active, showing two fields: ENAME and JOB. The 'Get Fields' button is highlighted. The 'Execution Results' section at the bottom shows the transformation completed successfully.

✓ Debug Transformation:

Click Concat fields -> Debug -> Quick Launch -> Yes

#	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	Emp_Data
1	7834	MILLER	CLERK	7782	1980/12/13 00:00:00.000	1300.0	<null>	10	MILLER_CLERK
2	7902	FORD	ANALYST	7566	1981/12/03 00:00:00.000	3000.0	<null>	20	FORD_ANALYST
3	7908	JAMES	CLERK	7898	1981/12/03 00:00:00.000	950.0	<null>	30	JAMES_CLERK
4	7876	ADAMS	CLERK	7788	1981/05/13 00:00:00.000	1100.0	<null>	20	ADAMS_CLERK
5	7844	TURNER	SALESMAN	7898	1981/08/08 00:00:00.000	1500.0	0.0	30	TURNER_SALESMAN
6	7839	KING	PRESIDENT	<null>	1981/11/17 00:00:00.000	5000.0	<null>	10	KING_PRESIDENT
7	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00.000	3000.0	<null>	20	SCOTT_ANALYST
8	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00.000	2400.0	<null>	10	CLARK_MANAGER
9	7866	BLAKE	MANAGER	7839	1981/05/01 00:00:00.000	2800.0	<null>	30	BLAKE_MANAGER
10	7854	MARTIN	SALESMAN	7898	1981/05/08 00:00:00.000	1250.0	1400.0	30	MARTIN_SALESMAN
11	7868	JONES	MANAGER	7839	1981/04/02 00:00:00.000	2975.0	<null>	20	JONES_MANAGER
12	7521	WARD	SALESMAN	7898	1981/02/12 00:00:00.000	1250.0	500.0	30	WARD_SALESMAN
13	7499	ALLEN	SALESMAN	7898	1981/02/20 00:00:00.000	1600.0	300.0	30	ALLEN_SALESMAN
14	7839	SMITH	CLERK	7902	1980/12/17 00:00:00.000	800.0	<null>	20	SMITH_CLERK

✓ Storing to Output Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Concat Fields) Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close

#	Table field	Stream field
1	EMPNO	EMPNO
2	ENAME	ENAME
3	JOB	JOB
4	MGR	MGR
5	HIREDATE	HIREDATE
6	SAL	SAL
7	COMM	COMM
8	DEPTNO	DEPTNO
9	Emp_Data	Emp_Data

✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes

The screenshot shows the Talend Data Integration environment. On the left, the 'Job' tree view is open, displaying various components like Input, Output, Transform, and Utilities. A transformation named 'Transformation 1' is selected. In the center, a flow diagram shows a 'Table Input' component connected to a 'Convert Fields' component, which then connects to a 'Table output' component. On the right, two windows are displayed: 'Execution preview data' and 'Execution Results'.

Execution preview data:

Row of step: Table output (24 rows)	EMPNO	ENAME	JOB	MGRAH	HIREDATE	SAL	COMM	DEPTNO	Emp_Data
1	75667	ENSMITH	CLERK	7897	1980-01-23	00-00000000	1300.0	result	10 MILLER_CLERK
2	75902	FORD	ANALYST	7894	1981-12-03	00-00000000	3000.0	result	20 TWARD_ANALYST
3	76005	HAMES	CLERK	7896	1981-12-03	00-00000000	950.0	result	30 JAMES_CLERK
4	76134	ADAMS	CLERK	7898	1980-05-22	00-00000000	1100.0	result	20 ADAMS_CLERK
5	78446	TURNER	SALESMAN	7899	1981-08-03	00-00000000	1500.0	result	30 TURNER_SALESMAN
6	78508	KING	PRESIDENT	-result	1981-01-01	17-00-00000000	50000.0	result	10 KING_PRESIDENT
7	78788	SCOTT	ANALYST	7904	1982-04-19	00-00000000	3000.0	result	20 SCOTT_ANALYST
8	79027	CLARK	MANAGER	7908	1981-06-09	00-00000000	2450.0	result	10 CLARK_MANAGER
9	79085	BLAKE	MANAGER	7929	1981-05-01	00-00000000	2850.0	result	30 BLAKE_MANAGER
10	79524	MARTIN	SALESMAN	7936	1981-05-28	00-00000000	1250.0	result	30 MARTIN_SALESMAN
11	79846	JONES	MANAGER	7939	1981-04-02	00-00000000	2875.0	result	20 JONES_MANAGER
12	75214	WARD	SALESMAN	7946	1981-02-22	00-00000000	1250.0	result	30 WARD_SALESMAN
13	74995	ALLEN	SALESMAN	7966	1981-03-23	00-00000000	1600.0	result	30 ALLEN_SALESMAN
14	73693	SMITH	CLERK	7962	1980-12-17	00-00000000	600.0	result	20 SMITH_CLERK

Execution Results:

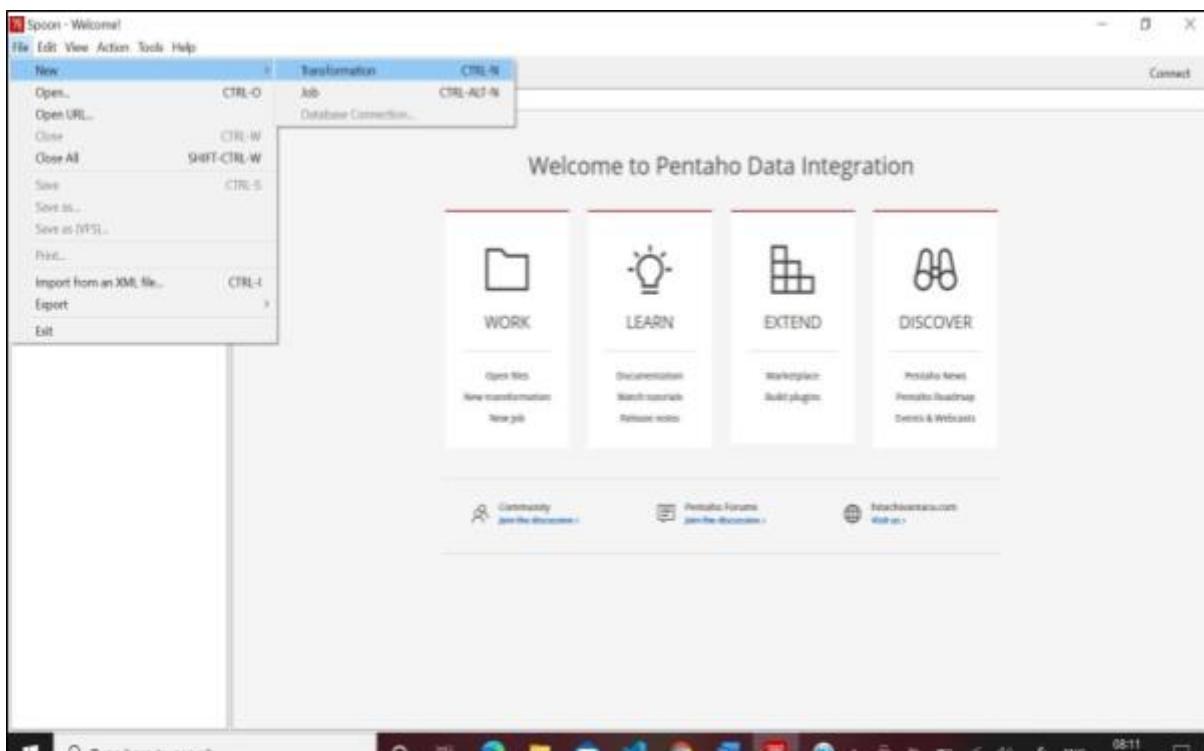
Execution completed successfully at 2023-10-06T12:22:01+00:00. Total processing time: 00:00:00.000. Rows processed: 24. Rows failed: 0. Errors: 0. Warnings: 0.

- Implementation of Splitting Operation.

Steps:

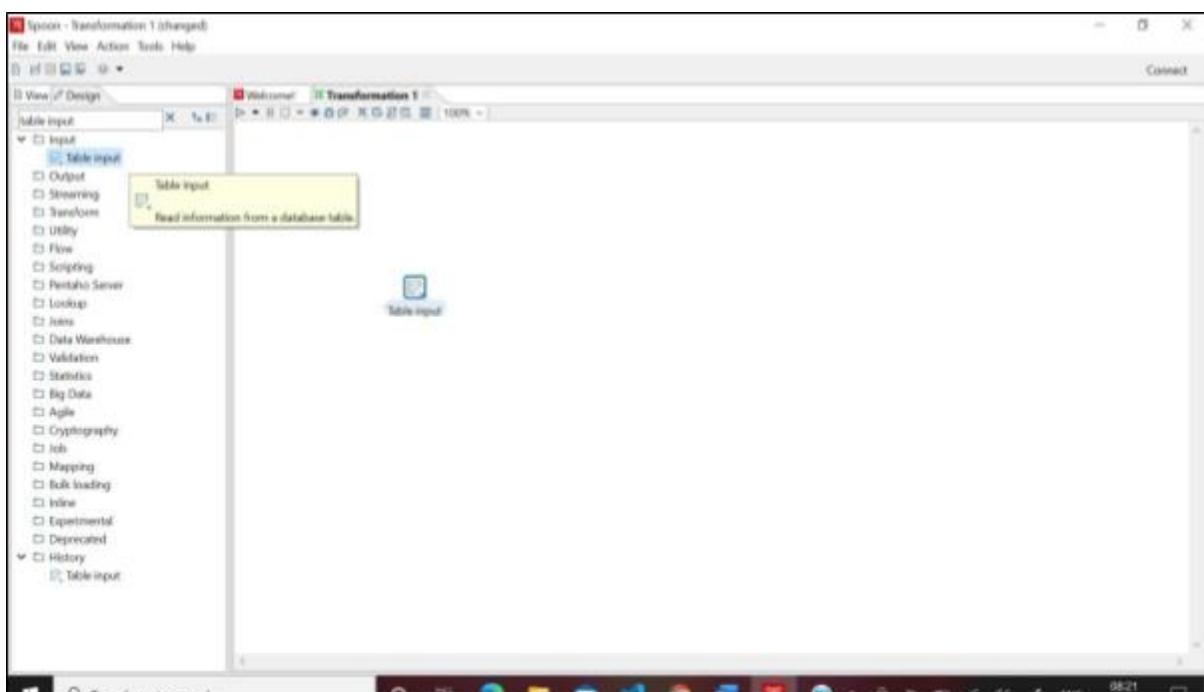
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



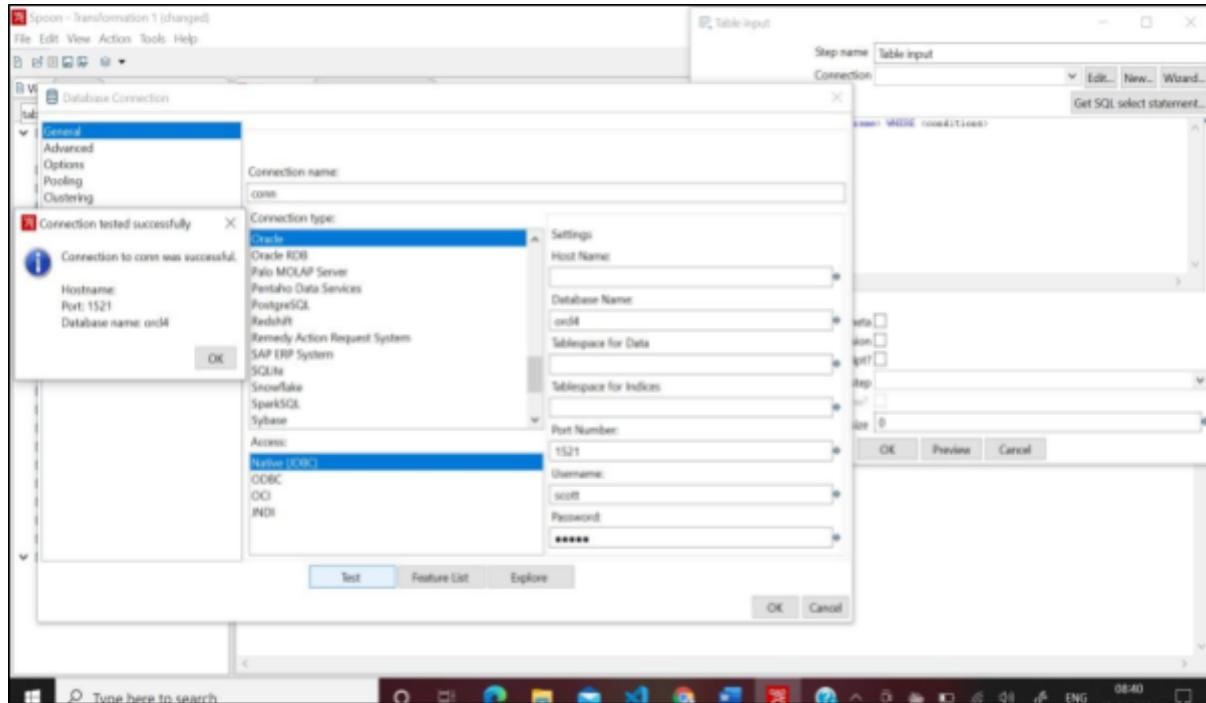
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



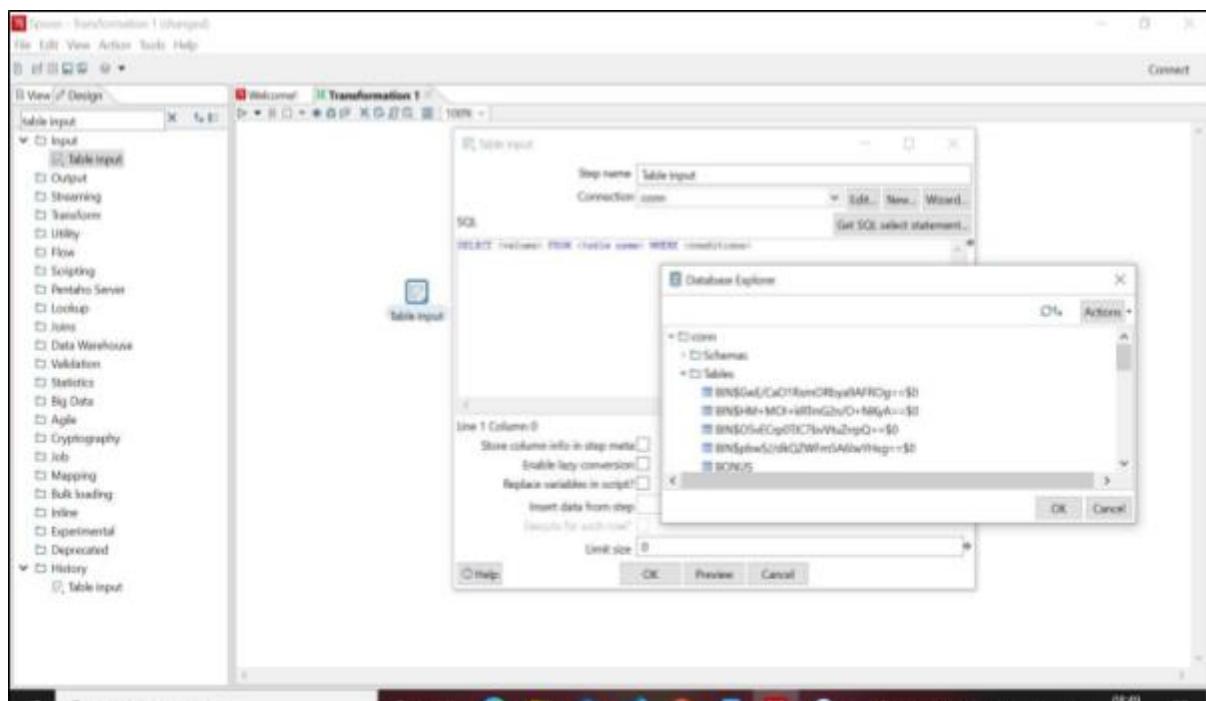
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



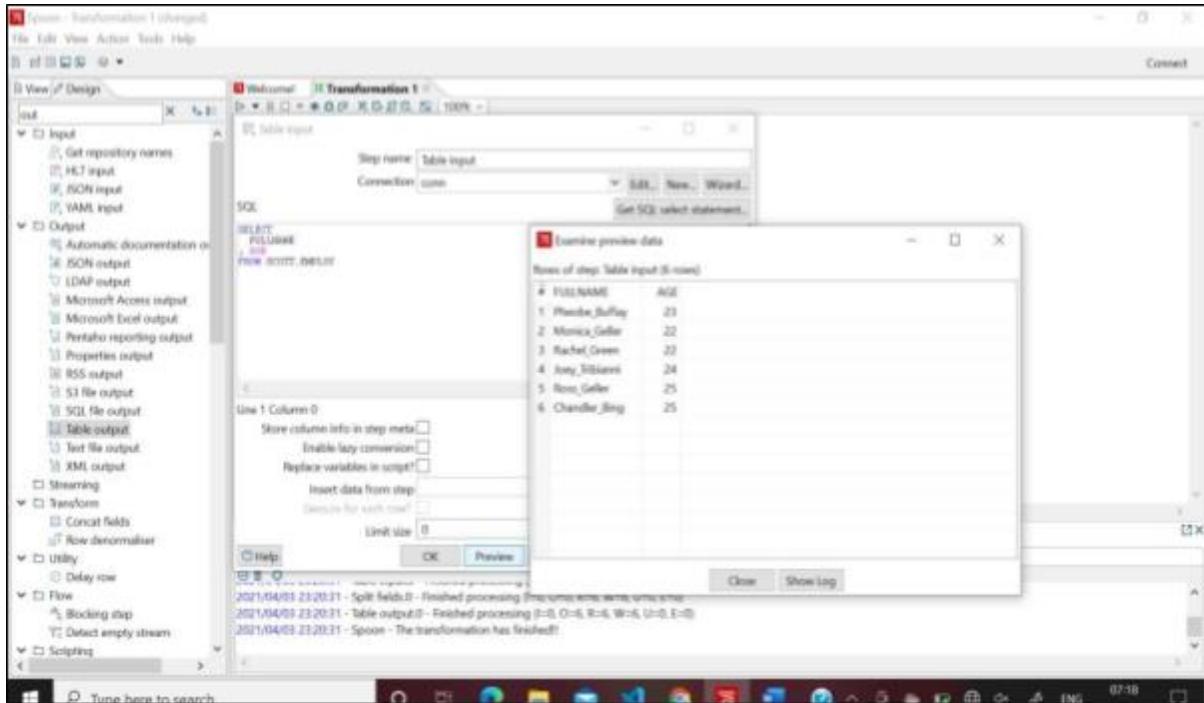
✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

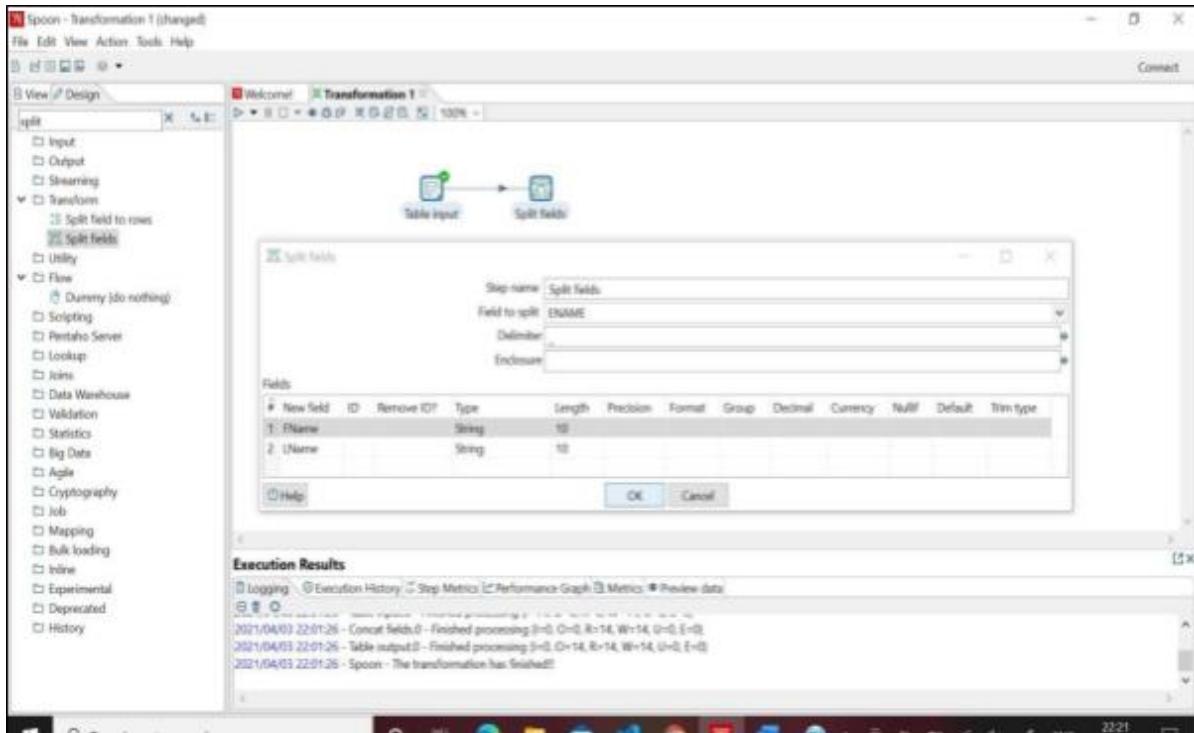
Table Input -> Preview -> OK -> Close -> OK



✓ Splitting Operation:

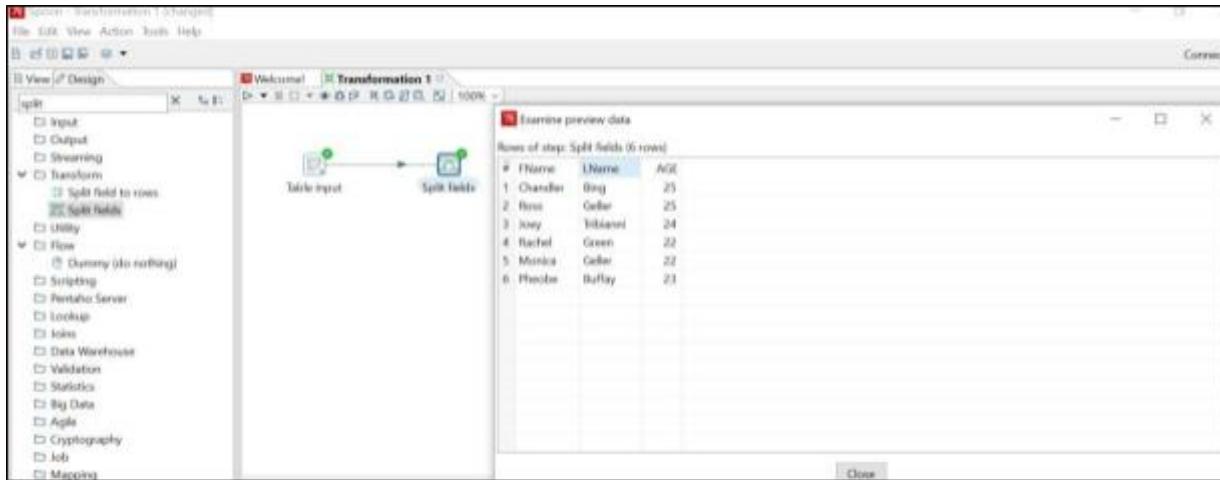
Design -> Transform -> Split Fields (Drag & Drop in Transformation) -> Create Hop Connection with Split Fields (Click on Output Connector arrow in Input Table)

Double Click Split Fields -> Field to split: ENAME -> Delimiter: _ -> New field: FName, LName -> Type: String -> Length: 10 -> OK



✓ Debug Transformation:

Click Split Fields -> Debug -> Quick Launch -> Yes



✓ Storing to Output Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Split Fields) Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close



✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes

The screenshot shows the Talend Transformation Designer interface. On the left, the 'View/Design' sidebar lists various components: Input (Get repository names, HDFS input, JSON input, YAML input), Output (Automatic documentation, LDIF output, Microsoft Access output, Microsoft Excel output, Persoaho reporting output, Properties output, RSS output, S3 file output, SQL file output, Table output, Text file output, XML output), Transform (Concat fields, Row denormalizer), Utility (Delay row), Flow (Blocking step, Detect empty changes), and Reporting.

The main workspace displays a transformation flow: Table Input → Split Fields → Table Output. A preview window titled 'Learn preview data' shows the following data:

#	FName	LName	Age
1	Chandler	Bing	25
2	Ross	Geller	25
3	Amy	Tucker	24
4	Rachel	Green	27
5	Monica	Geller	22
6	Phoebe	Buchy	29

The 'Execution Results' panel at the bottom shows the following log entries:

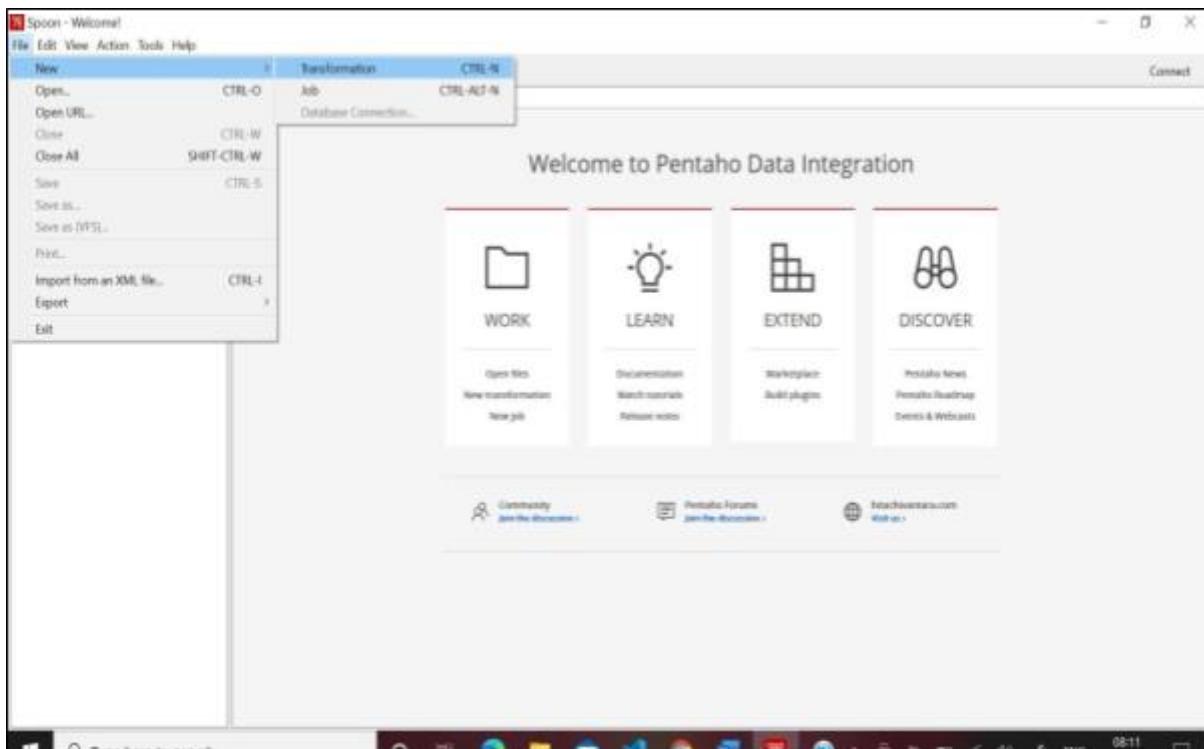
- 2011/04/03 23:20:31 - Split fields 0 - Finished processing (R=0, C=0, W=0, U=0, E=0)
- 2011/04/03 23:20:31 - Table output 0 - Finished processing (R=6, C=3, W=6, U=6, E=0)
- 2011/04/03 23:20:31 - Spoon - The transformation has finished!

- Implementation of Number Range.

Steps:

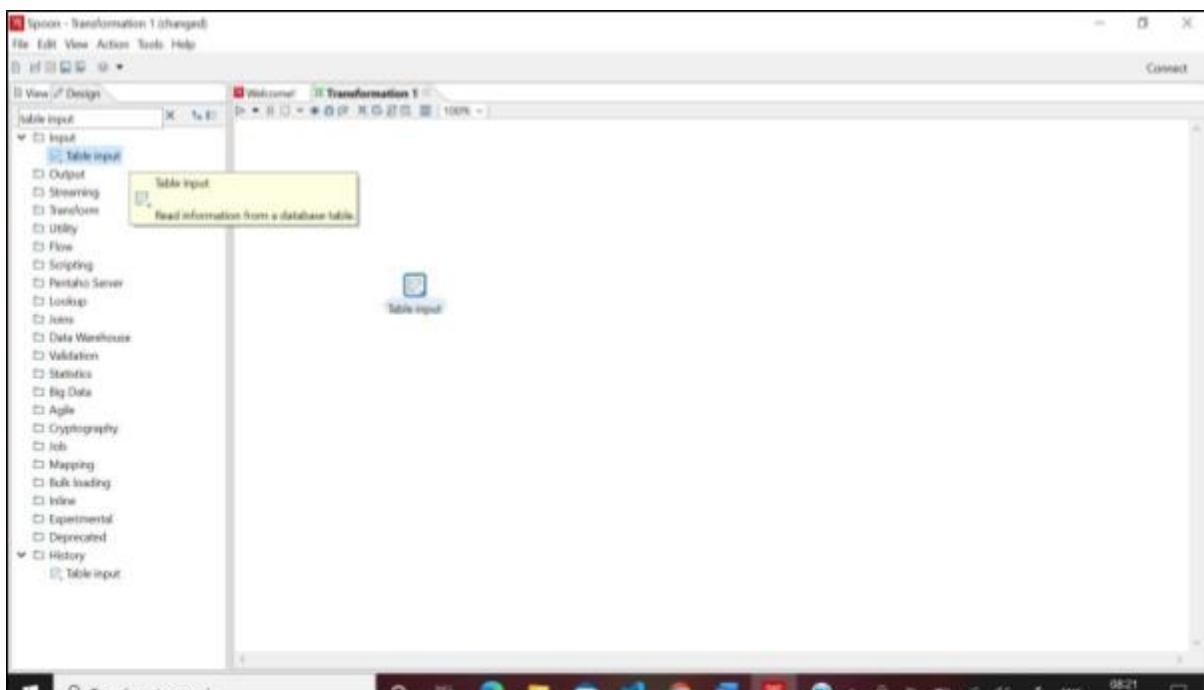
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



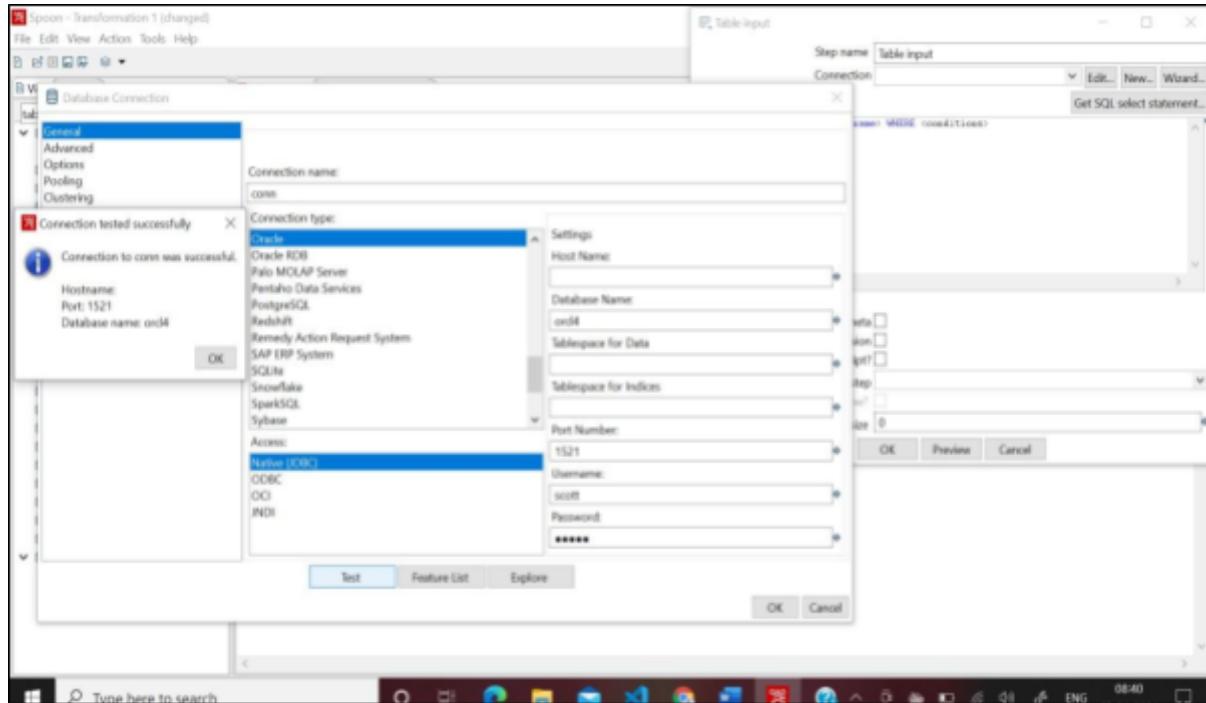
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



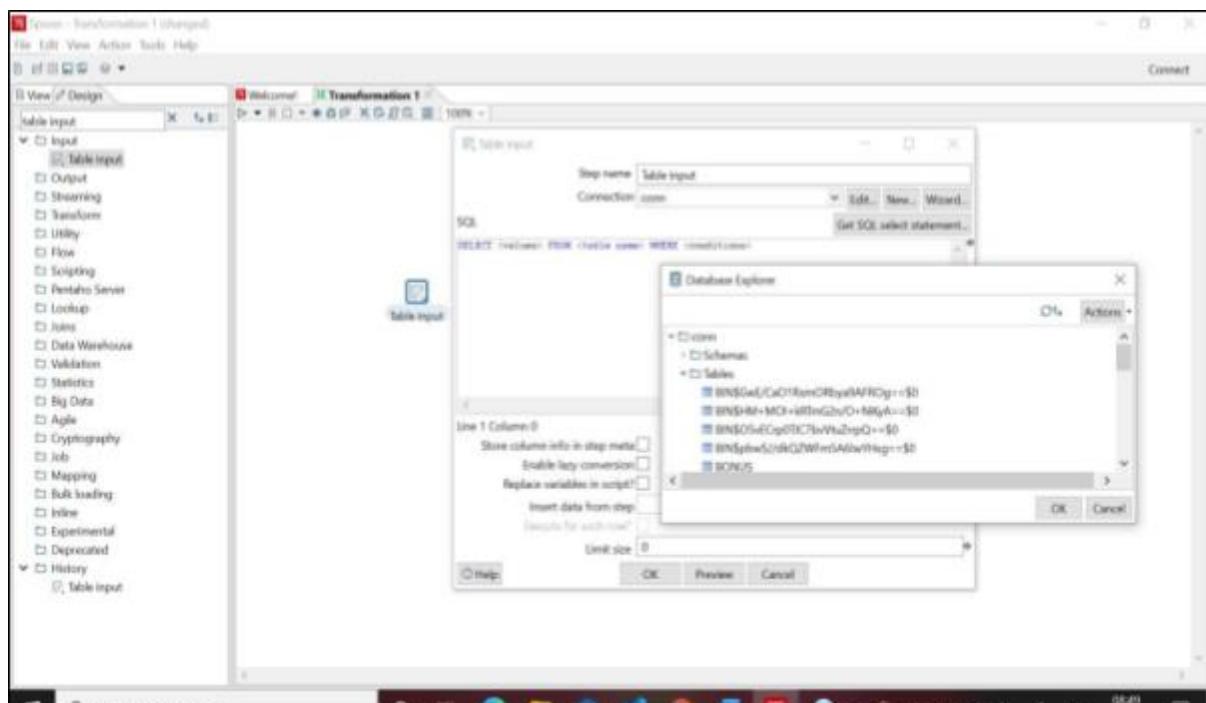
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

Table Input -> Preview -> OK -> Close -> OK

The screenshot shows the Spoon interface for a transformation named 'Transformation 1 (changed)'. The 'Table input' step is selected. In the 'Preview' tab, the first 14 rows of the 'EMPLOYEES' table are displayed. The SQL query used is:

```
SELECT * FROM SCOTT.BIGTEST
```

The preview window also includes options for column storage, lazy conversion, and limit size.

✓ Number Range:

Design-> Transform-> Number Range(Drag & Drop in Transformation)->Create Hop Connection with Number Range (Click on Output Connector arrow in Input Table)

Double Click Number Range -> Input field: *EMPNO* -> Lower Bound: *7301, 7401, 7601, 7801, 7901* -> Upper Bound: *7400, 7600, 7800, 7900, 8000* -> Value: *Interns, Officers, Managers, Executives, Retirees* -> OK

The screenshot shows the Spoon interface for 'Transformation 1 (changed)'. A hop connection is established from the 'Table input' step to the 'Number range' step. The 'Number range' configuration dialog is open, showing the following settings:

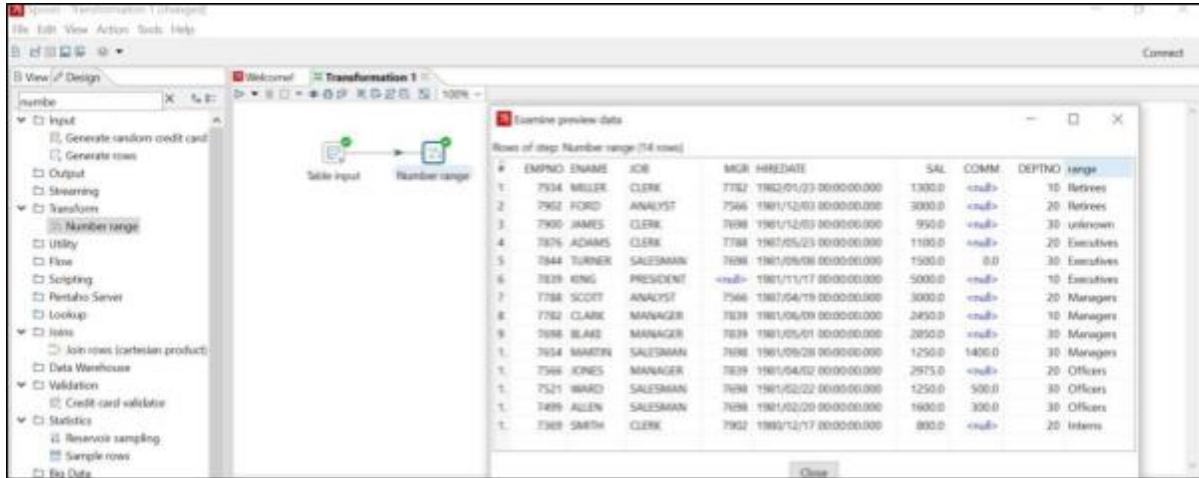
- Step name: Number range
- Input field: EMPNO
- Output field: range
- Default value if no range: unknown
- Ranges (min <= x <= max):
 - Lower Bound: 7301, Upper Bound: 7400, Value: Interns
 - Lower Bound: 7401, Upper Bound: 7600, Value: Officers
 - Lower Bound: 7601, Upper Bound: 7800, Value: Managers
 - Lower Bound: 7801, Upper Bound: 7900, Value: Executives
 - Lower Bound: 7901, Upper Bound: 8000, Value: Retirees

The 'Execution Results' pane at the bottom shows the execution log:

```
2021/04/03 23:20:31 - Split Fields 0 - Finished processing (I=0, O=0, R=6, W=6, U=0, E=0)
2021/04/03 23:20:31 - Table output 0 - Finished processing (I=0, O=0, R=6, W=6, U=0, E=0)
2021/04/03 23:20:31 - Spoon - The transformation has finished!
```

✓ Debug Transformation:

Click Number Range -> Debug -> Quick Launch -> Yes



✓ Storing to Output Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table(Connector arrow Number Range)>Main output of step Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close



✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes

The screenshot shows the Apache Nifi transformation editor. On the left, the sidebar lists various output types, with "Table output" selected. The main canvas displays a flow: Table input → Number range → Table output. A preview window titled "Examine previous data" is open, showing 14 rows of data from the Table output step. The columns in the preview are: #, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO, and RENAME. The data includes rows for employees like SMITH, ALLEN, KING, and CLARK.

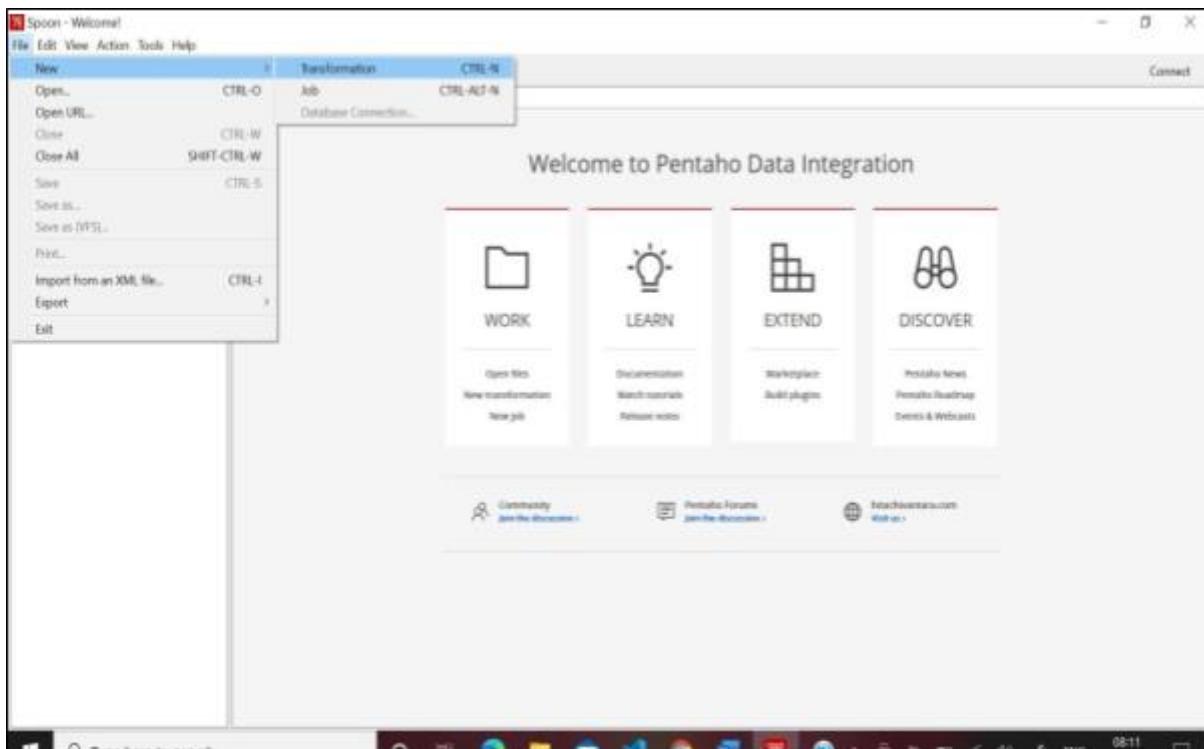
#	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	RENAME
1	SMITH	CLERK	7782	1982/01/17 00:00:00.000	1300.0	<null>	10	Replies
2	KING	PRESE	7802	1981/06/17 00:00:00.000	5000.0	<null>	20	Replies
3	CLARK	CLERK	7839	1981/01/01 00:00:00.000	950.0	<null>	30	unknown
4	ADAMS	CLERK	7788	1981/05/23 00:00:00.000	1100.0	<null>	30	Executives
5	TURNER	SALESMAN	7844	1981/08/09 00:00:00.000	1500.0	<null>	30	Executives
6	ADAMS	PRESE	7809	1981/11/17 00:00:00.000	5000.0	<null>	10	Executives
7	SCOTT	ANALYST	7866	1983/06/19 00:00:00.000	3000.0	<null>	20	Managers
8	CLARK	MANAGER	7839	1981/06/09 00:00:00.000	2450.0	<null>	10	Managers
9	BLAKE	MANAGER	7889	1981/05/01 00:00:00.000	2910.0	<null>	30	Managers
10	MARTIN	SALESMAN	7866	1981/09/28 00:00:00.000	1250.0	5400.0	30	Managers
11	JONES	MANAGER	7839	1981/04/02 00:00:00.000	2975.0	<null>	20	Officers
12	WARD	SALESMAN	7866	1981/02/22 00:00:00.000	1250.0	500.0	30	Officers
13	ALLEN	SALESMAN	7839	1981/02/03 00:00:00.000	1600.0	300.0	30	Officers
14	SMITH	CLERK	7902	1980/12/17 00:00:00.000	800.0	<null>	20	Interns

- Implementation of String Operation.

Steps:

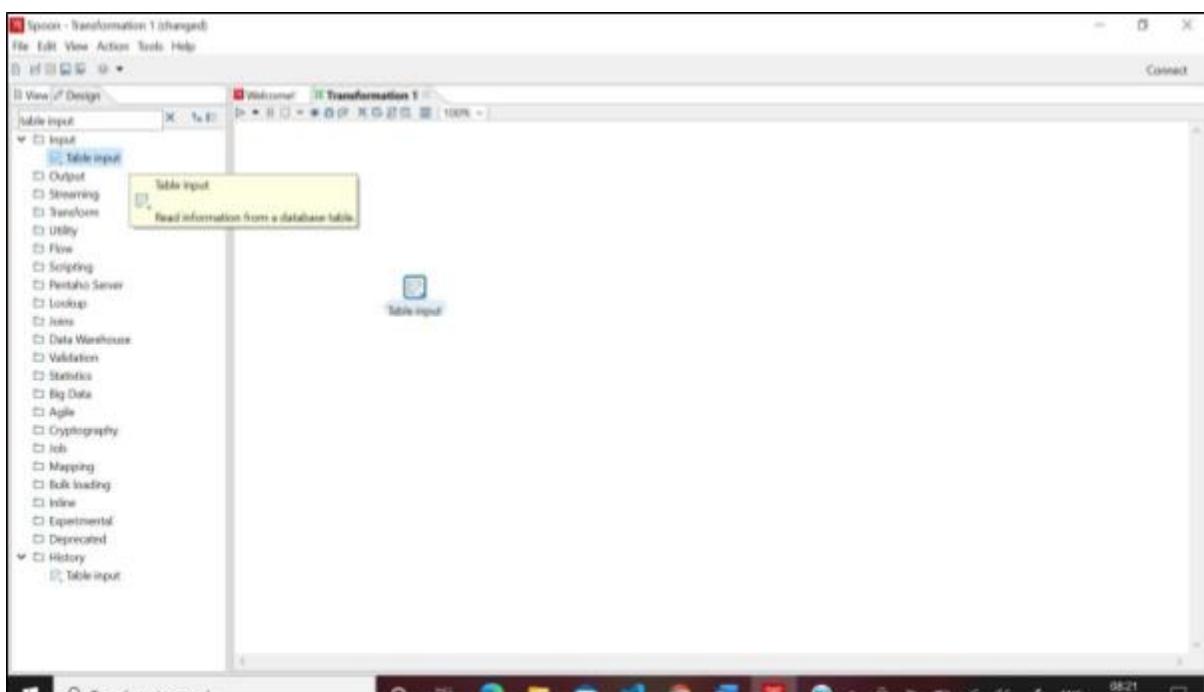
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



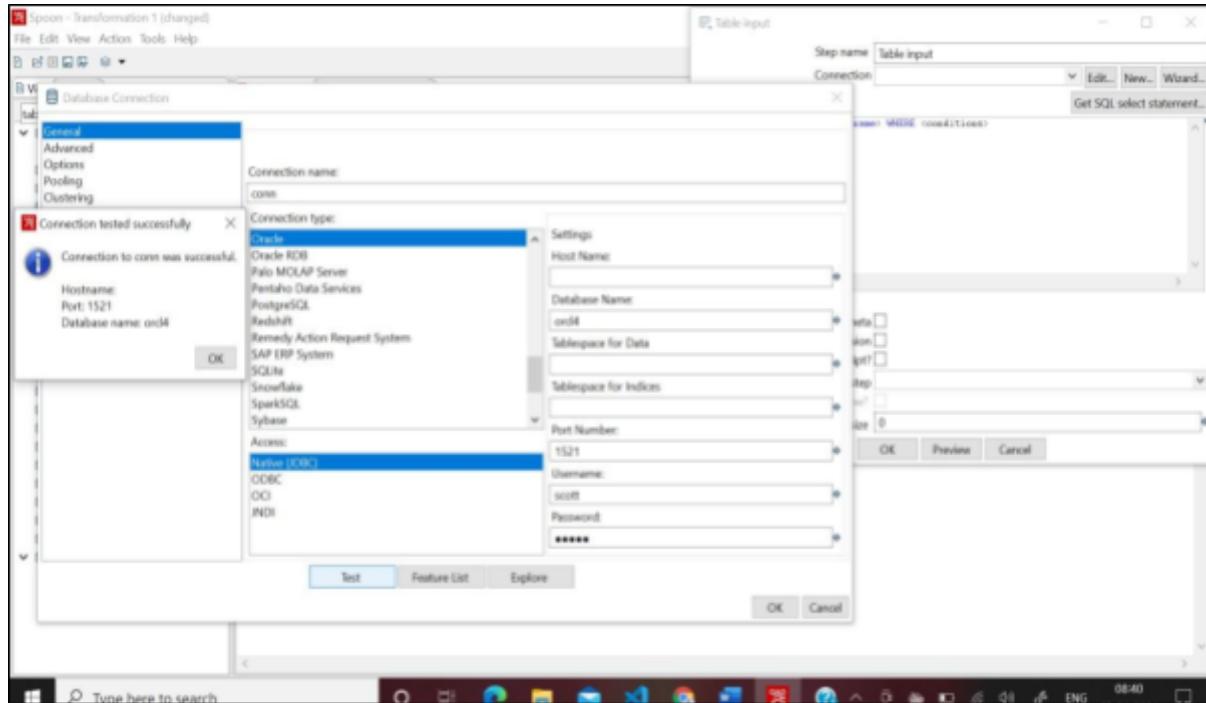
- ✓ Inserting Table Input into the Transformation:

Design -> Input -> Table Input (Drag & Drop in Transformation)



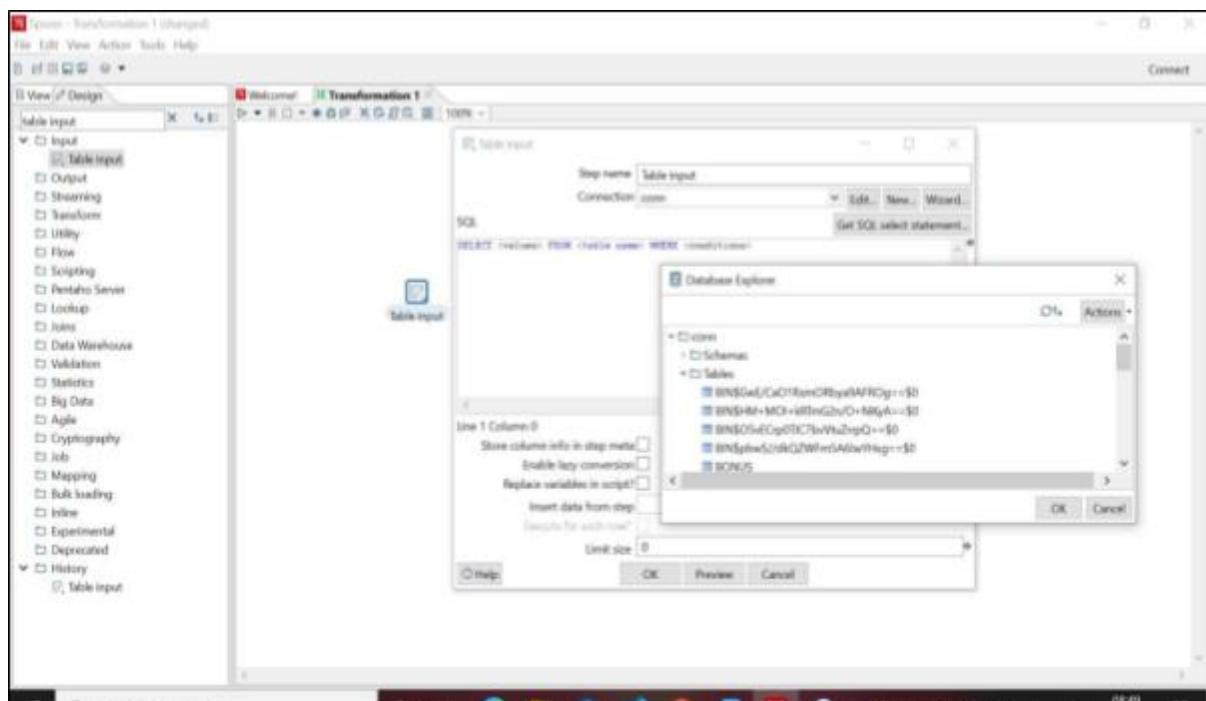
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:
Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)* Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



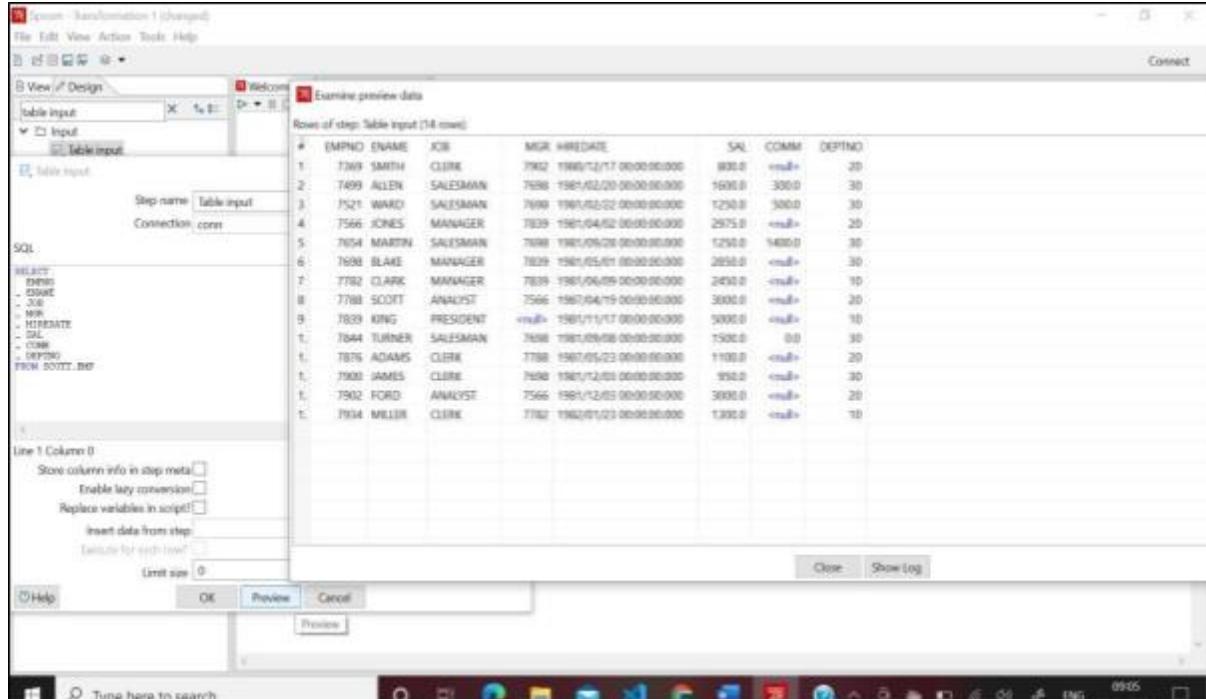
✓ Getting Source Table:

Table Input -> Get SQL Select Statement -> Data Explorer -> *conn* -> Tables -> Click on desired Table -> OK -> Yes



✓ View Input Table:

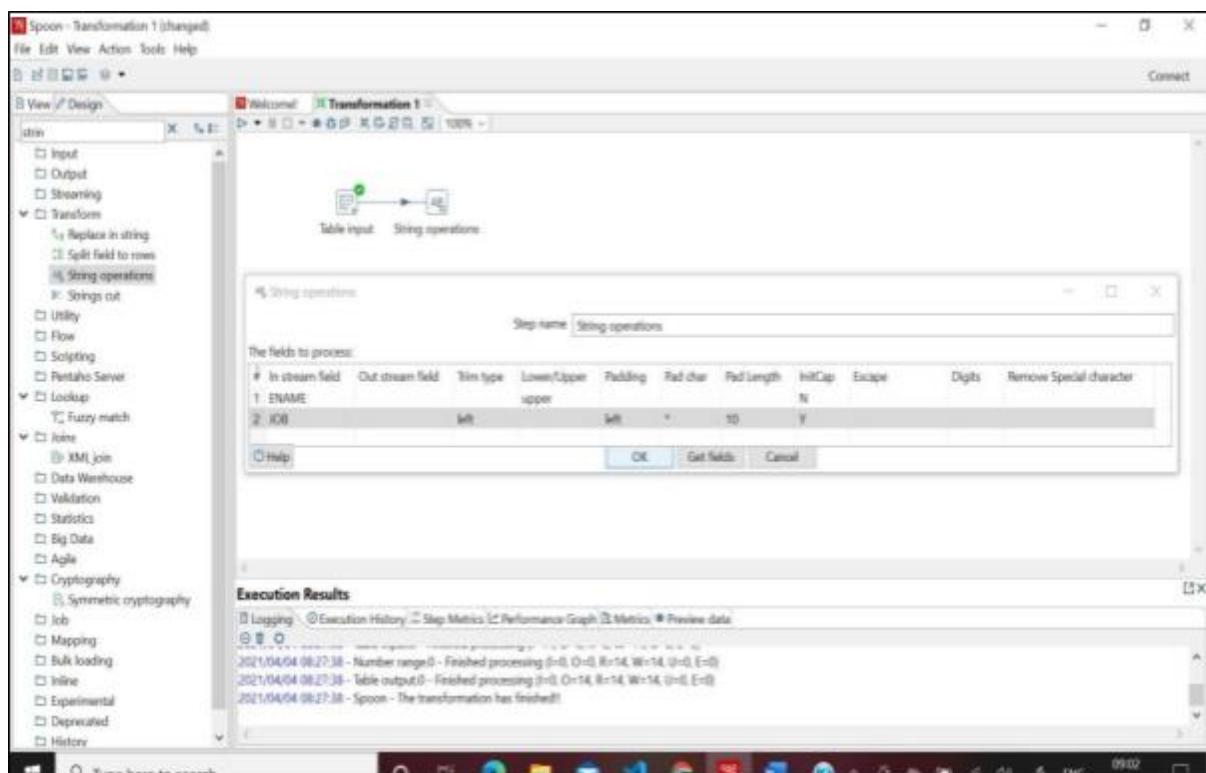
Table Input -> Preview -> OK -> Close -> OK



✓ String Operations:

Design->Transform->String Operations(Drag / Drop in Transformation)->Create Hop Connection with String Operations (Click on Output Connector arrow in Input Table)

Double Click String Operations -> In stream field: *ENAME* -> Lower/Upper: *upper* -> InitCap: *N* -> In stream field: *JOB* -> Trim type: *left* -> Padding: *left* -> Pad char: *** -> Pad Length: *10* -> InitCap: *Y* ->OK



✓ Debug Transformation:

Click String Operations -> Debug -> Quick Launch -> Yes

#	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7361	MILLER	SALESMAN	7362	1982/01/13 00:00:00.000	13000.0	NULL	10
2	7362	FORD	ANALYST	7566	1981/12/03 00:00:00.000	30000.0	NULL	20
3	7363	JAMES	CLERK	7566	1981/12/03 00:00:00.000	950.0	NULL	30
4	7364	ADAMS	CLERK	7788	1987/05/13 00:00:00.000	11000.0	NULL	20
5	7364	TURNER	SALESMAN	7898	1981/08/06 00:00:00.000	15000.0	0.0	30
6	7365	KING	PRESIDENT	7821	1981/11/17 00:00:00.000	50000.0	NULL	10
7	7366	SCOTT	ANALYST	7566	1987/04/19 00:00:00.000	30000.0	NULL	20
8	7367	CLARK	MANAGER	7839	1981/06/09 00:00:00.000	24000.0	NULL	10
9	7368	BLAKE	MANAGER	7839	1981/05/11 00:00:00.000	28500.0	NULL	30
10	7369	MARTIN	SALESMAN	7898	1981/09/28 00:00:00.000	12500.0	1400.0	30
11	7370	JONES	MANAGER	7839	1981/04/02 00:00:00.000	29750.0	NULL	20
12	7371	WARD	SALESMAN	7898	1981/02/02 00:00:00.000	12500.0	500.0	30
13	7409	ALLISON	SALESMAN	7898	1981/02/20 00:00:00.000	16000.0	300.0	30
14	7389	SMITH	CLERK	7892	1980/12/17 00:00:00.000	8000.0	NULL	20

✓ Storing to Output Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection to Output Table(Connector arrow String Operation)->Main output of step Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close

Table Field	Stream field
1	EMPNO
2	ENAME
3	JOB
4	MGR
5	HIREDATE
6	SAL
7	COMM
8	DEPTNO

✓ Debug Transformation:

Click Table Output -> Debug -> Quick Launch -> Yes

The screenshot shows the Talend Data Integration interface. On the left, the 'View' menu is open, showing various output options like 'Table output', 'Text file output', 'Update', 'Transform', and 'Delete'. The main workspace displays a transformation flow named 'Transformation 1'. The flow consists of three steps: 'Table Input', 'String operations', and 'Table output'. A preview window titled 'Examine preview data' is open, showing 14 rows of data from the 'Table output' step. The columns are labeled: #, EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, and DEPTNO. The data includes records for employees like JAMES, TURNER, and MARTIN. Below the preview window, the 'Execution Results' section shows logs of the processing steps.

#	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7934	MILLER	Clerk	7782	1982/01/23 00:00:00.000	1300.0	credit	10
2	7902	FORD	Analyst	7566	1981/12/03 00:00:00.000	3000.0	credit	20
3	7800	JAMES	Clerk	7698	1983/12/03 00:00:00.000	950.0	credit	30
4	7876	ADAMS	Clerk	7788	1987/05/03 00:00:00.000	1100.0	credit	20
5	7844	TURNER	Salesman	7908	1983/08/08 00:00:00.000	1500.0	0.0	30
6	7839	KING	President	7868	1981/11/17 00:00:00.000	5000.0	credit	10
7	7782	SCOTT	Analyst	7566	1981/04/19 00:00:00.000	3000.0	credit	20
8	7782	CLARK	Manager	7839	1981/06/09 00:00:00.000	2450.0	credit	10
9	7698	BLAKE	Manager	7839	1981/05/01 00:00:00.000	2850.0	credit	30
10	7654	MARTIN	Salesman	7986	1981/09/28 00:00:00.000	1250.0	1400.0	30
11	7566	JONES	Manager	7839	1981/06/02 00:00:00.000	2975.0	credit	20
12	7521	WARD	Salesman	7698	1981/02/22 00:00:00.000	1250.0	500.0	30
13	7499	ALLLEN	Salesman	7698	1981/02/20 00:00:00.000	1600.0	300.0	30
14	7369	SMITH	Clerk	7902	1980/12/17 00:00:00.000	800.0	credit	20

Execution Results

- Logging
- Execution History
- Step Metrics
- Performance Graph
- Metrics
- Preview data

```

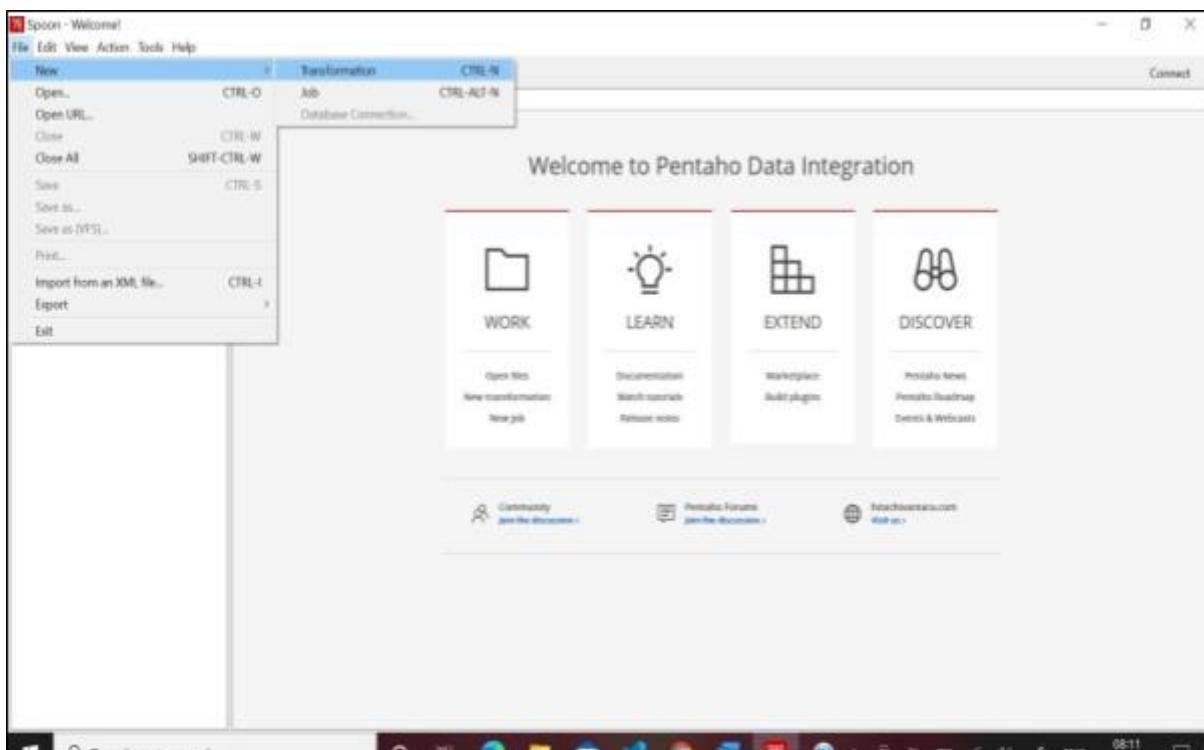
2021/04/04 09:20:07 - String operations:0 - Finished processing (I=0, O=0, R=14, W=14, U=0, E=0)
2021/04/04 09:20:07 - Table output:0 - Finished processing (I=0, O=14, R=14, W=14, U=0, E=0)
2021/04/04 09:20:07 - Spooler - The transformation has finished
    
```

- Importing .CSV file to the target table.

Steps:

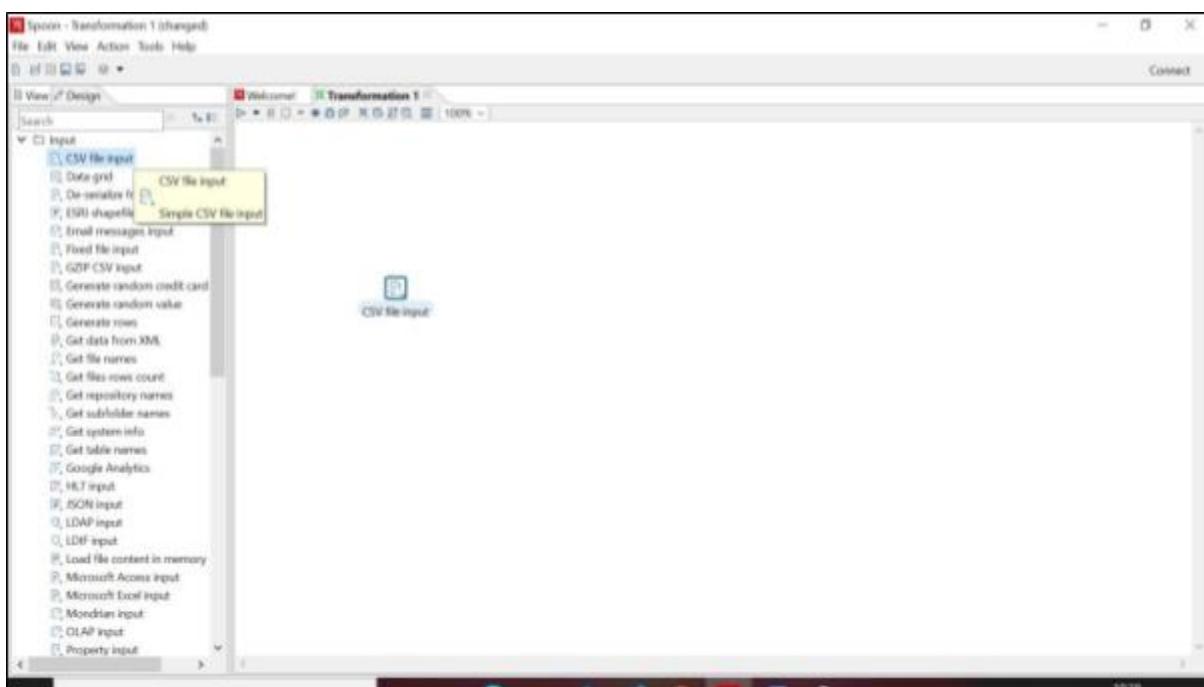
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



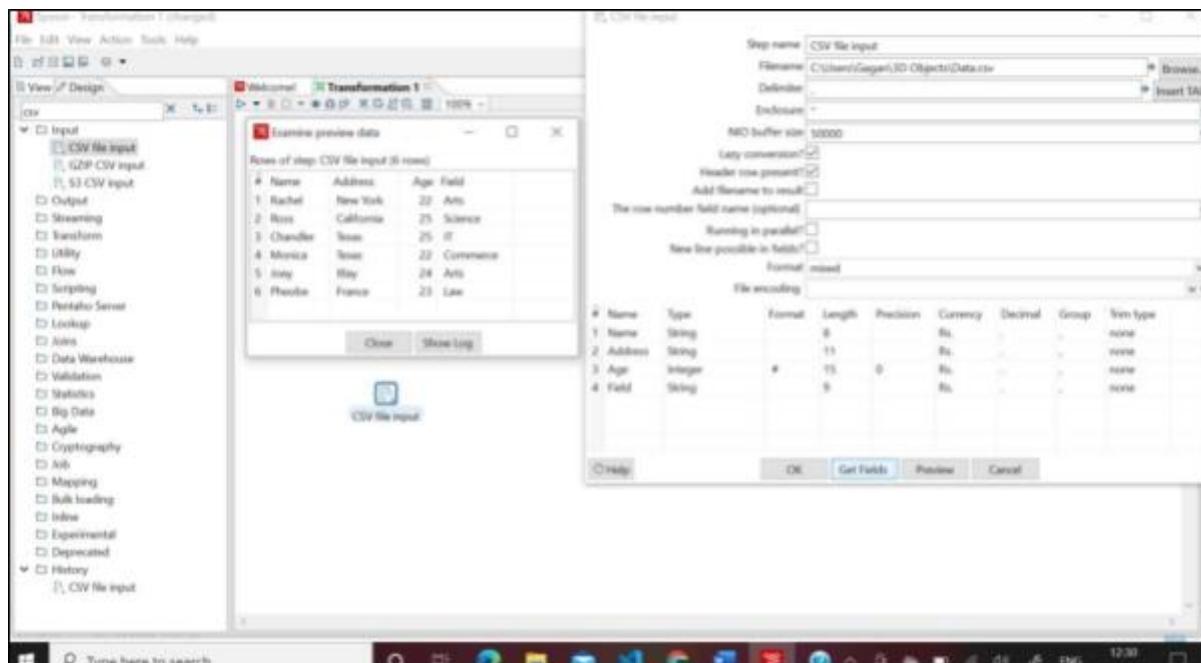
- ✓ Inserting CSV File Input into the Transformation:

Design -> Input -> CSV File Input (Drag & Drop in Transformation)



✓ Importing .CSV File:

Double Click CSV File Input -> Filename (Browse): *Data.csv* -> Get Fields -> OK -> Preview -> OK -> Close -> OK



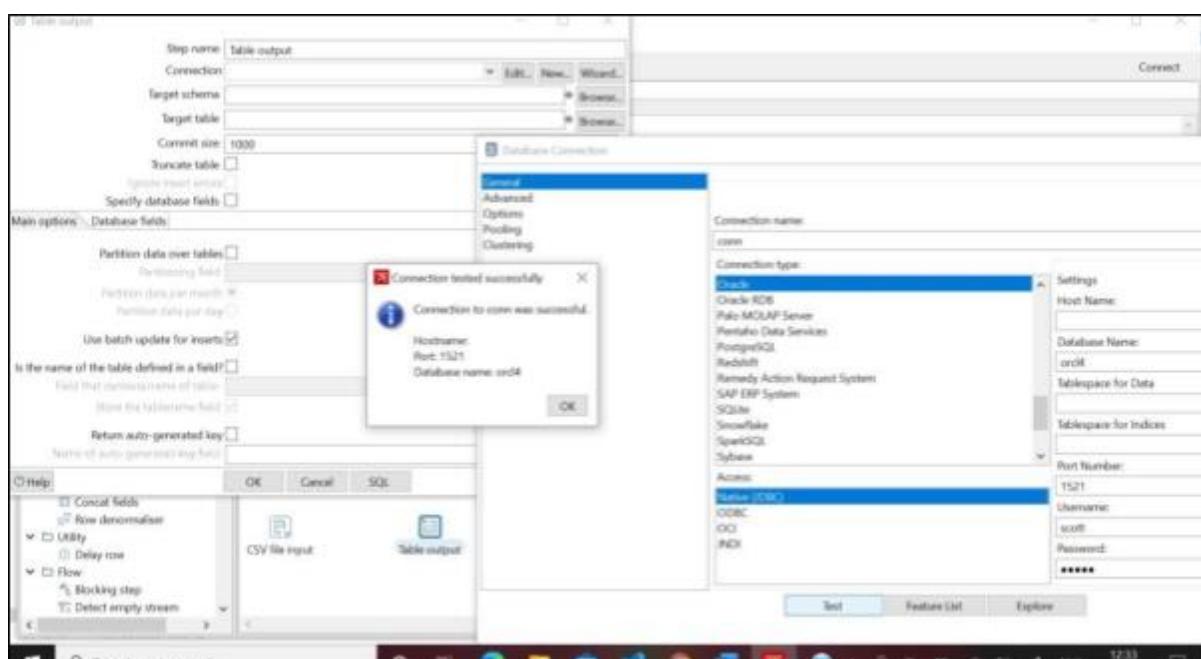
✓ Storing to Target Table:

Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection to Output Table (Connector arrow CSV file input) -> Main output of step

✓ Establishing Database Connection:

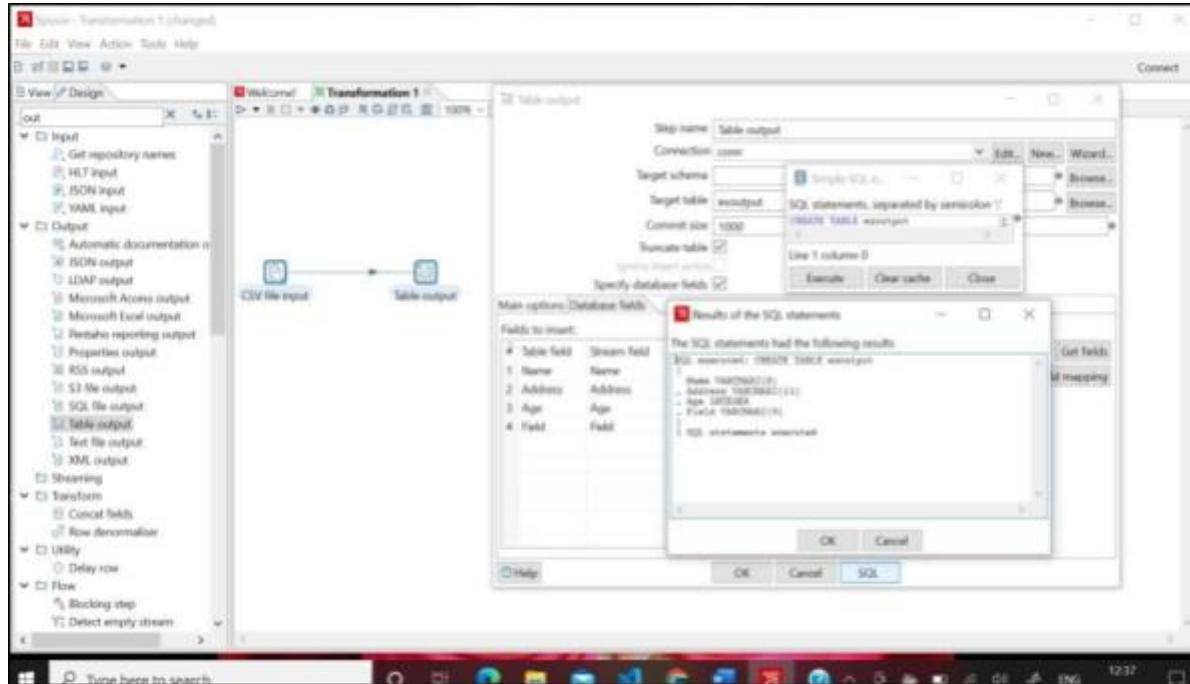
New -> Enter following respective Details in *General*:

Connection name: *conn* Connection Type: Oracle Access: Native (JDBC) Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger* Click Test -> OK -> OK



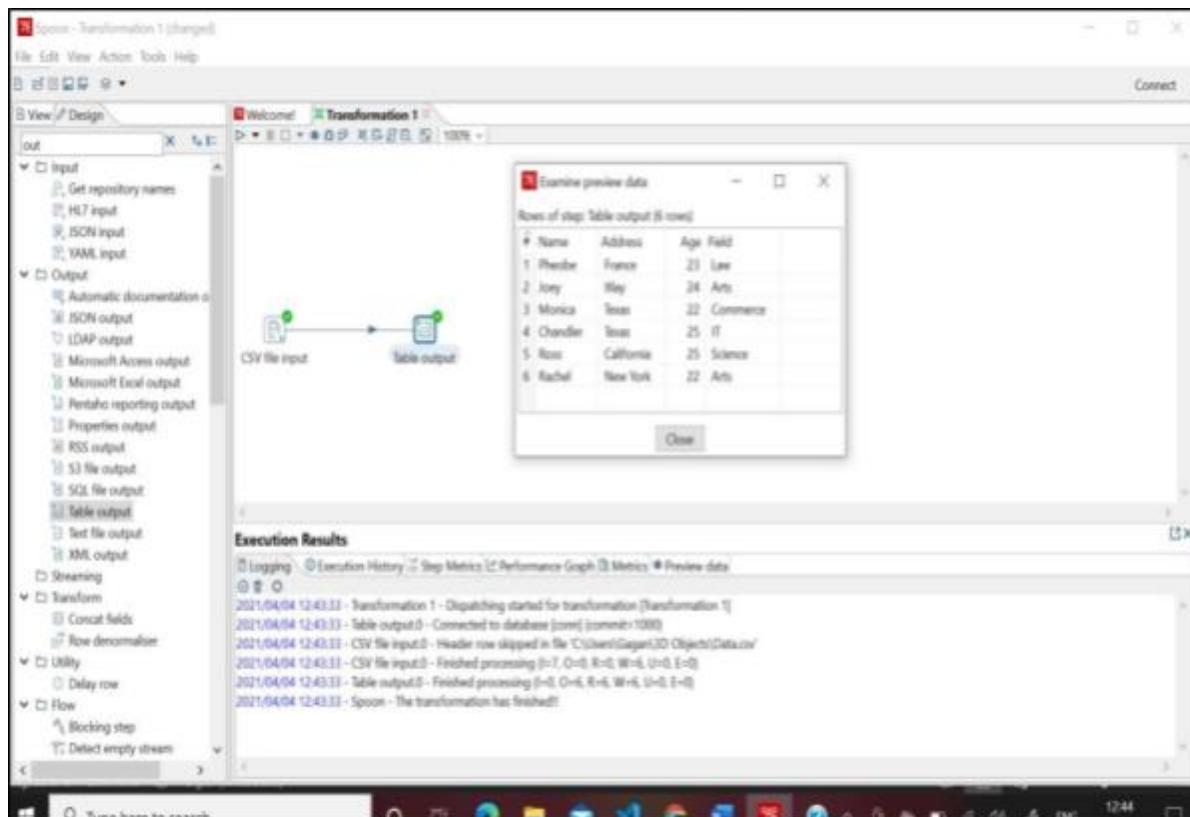
✓ Importing to Target Table:

Double Click Table Output -> Target Table: *output* -> Truncate table -> Specify database fields -> Database fields -> Get fields -> SQL -> Execute -> OK -> Close



✓ Debug Transformation (View Target Table):

Click Table Output -> Debug -> Quick Launch -> Yes

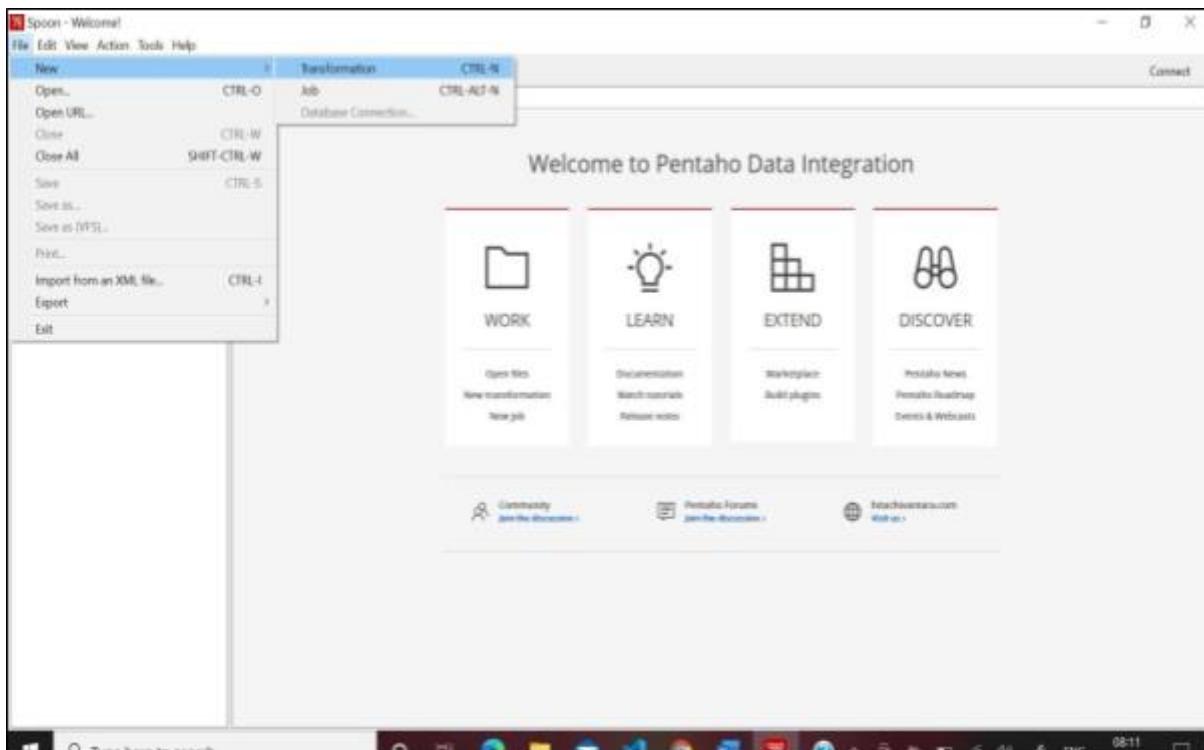


- Implement the Merge Join Transformation on Tables.

Steps:

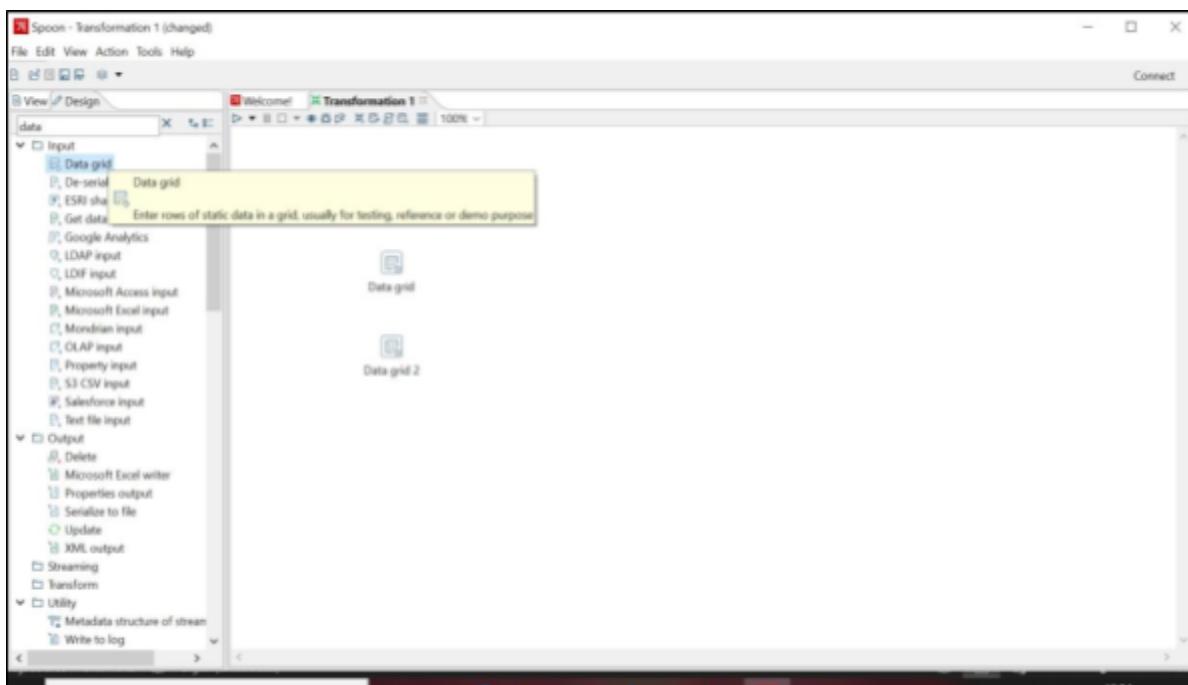
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



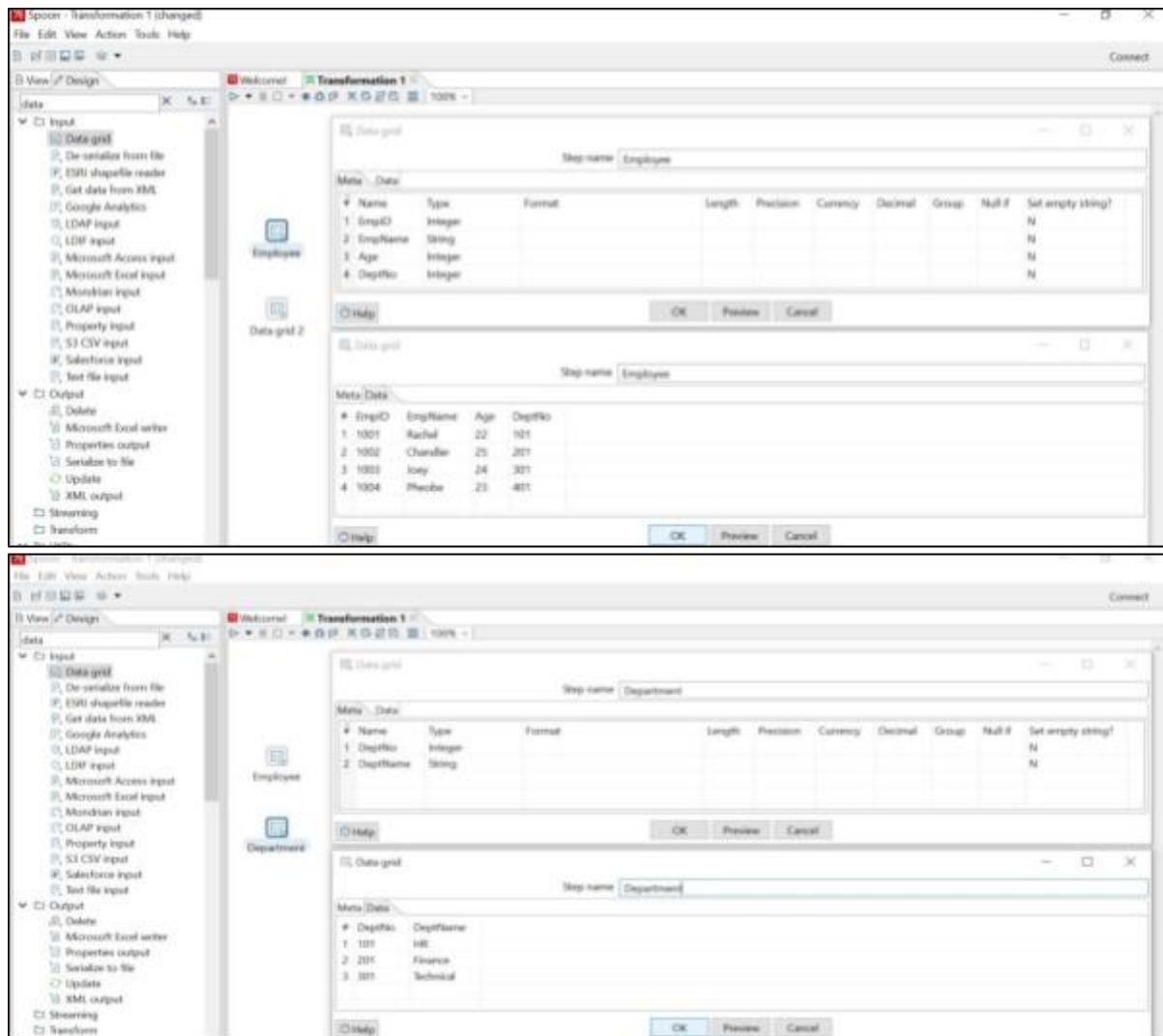
- ✓ Inserting Data Grid into the Transformation:

Design -> Input -> Data Grid (Twice Drag & Drop in Transformation)



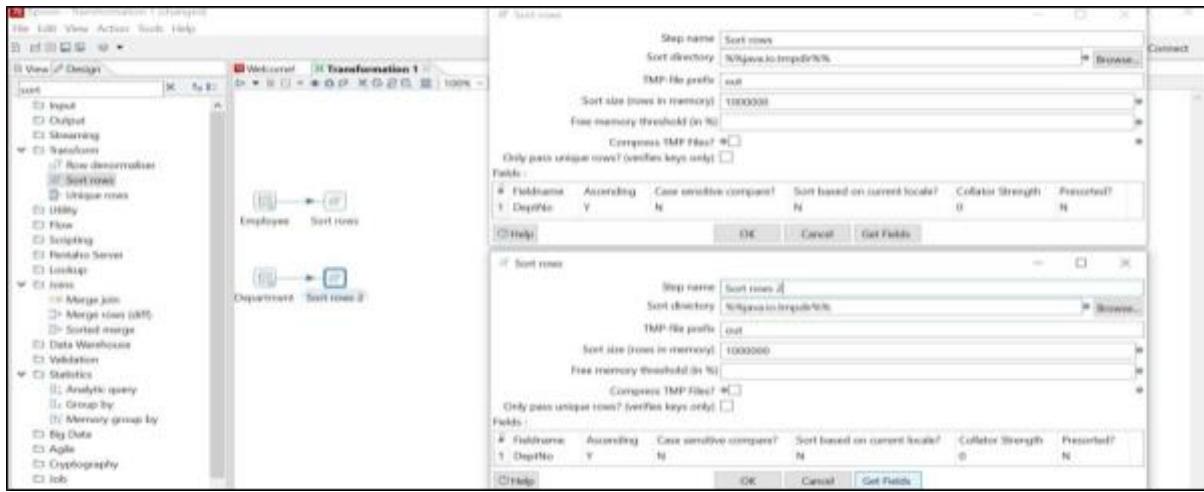
✓ Inserting Data in Data Grid:

Double Click Data Grids -> Step name : *Employee/ Department* -> Meta:
Enter Fields Details -> Data: *Enter actual Data* -> OK



✓ Inserting Sort Rows:

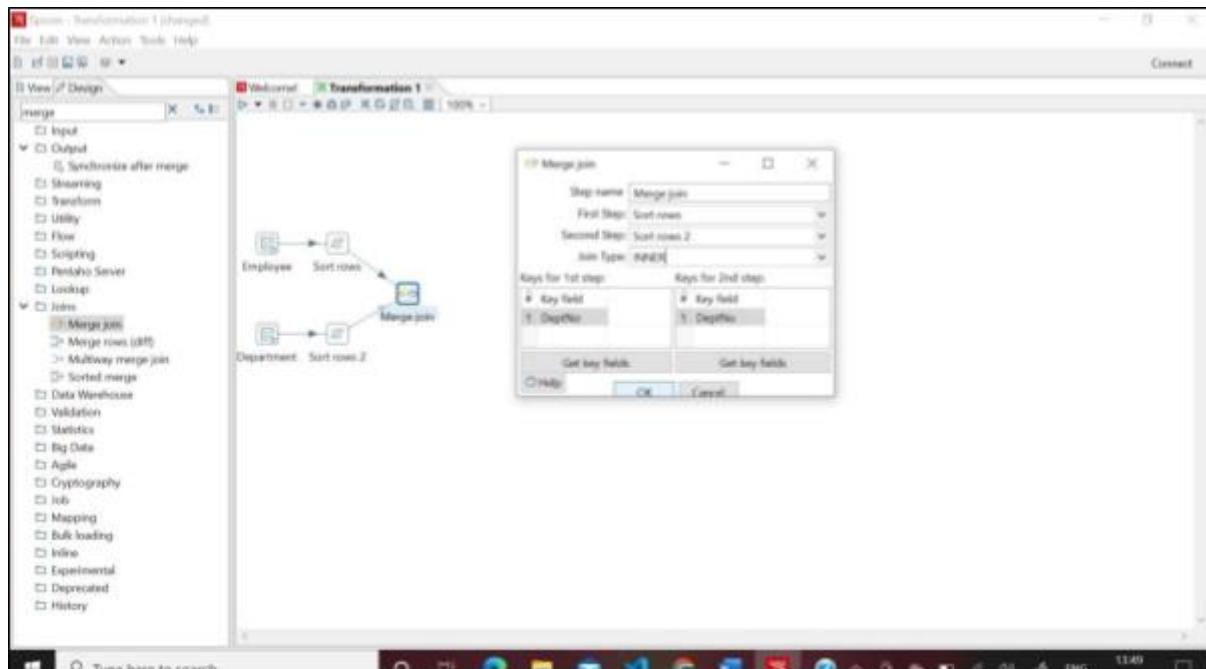
Design -> Transform -> Sort Rows (Twice Drag & Drop in Transformation) -> Create Hop Connection with Sort Rows (Click on Output Connector arrow in Data Grids) Double Click Sort Rows -> Get Fields (Discard unwanted Fields) -> OK



✓ Inserting Merge Join into the Transformation:

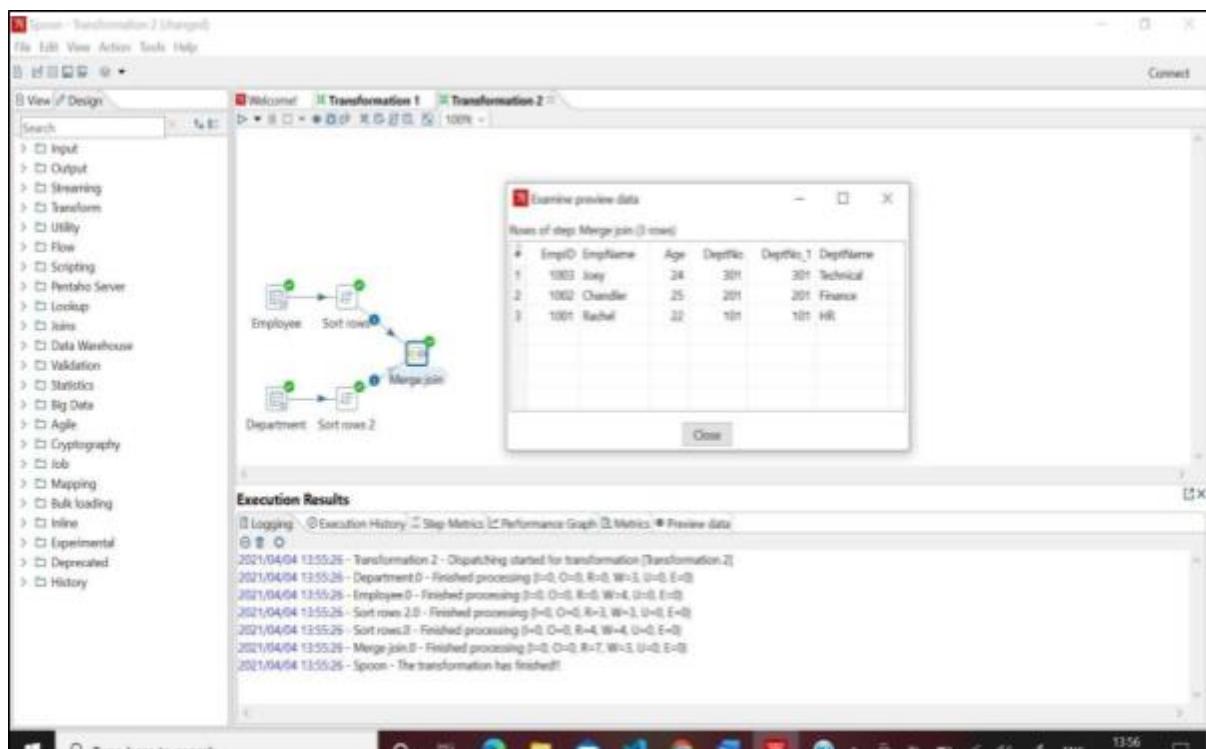
Design -> Joins -> Merge Join (Drag & Drop in Transformation) -> Create Hop Connection with Merge Join (Click on Output Connector arrow in Sort Rows)

Double Click Merge Join -> First Step: Sort rows -> Second Step: Sort Rows 2 -> Join Type: INNER -> Get key fields (Both, Discard unwanted Fields) -> OK -> Close



✓ Debug Transformation:

Click Merge Join -> Debug -> Quick Launch -> Yes

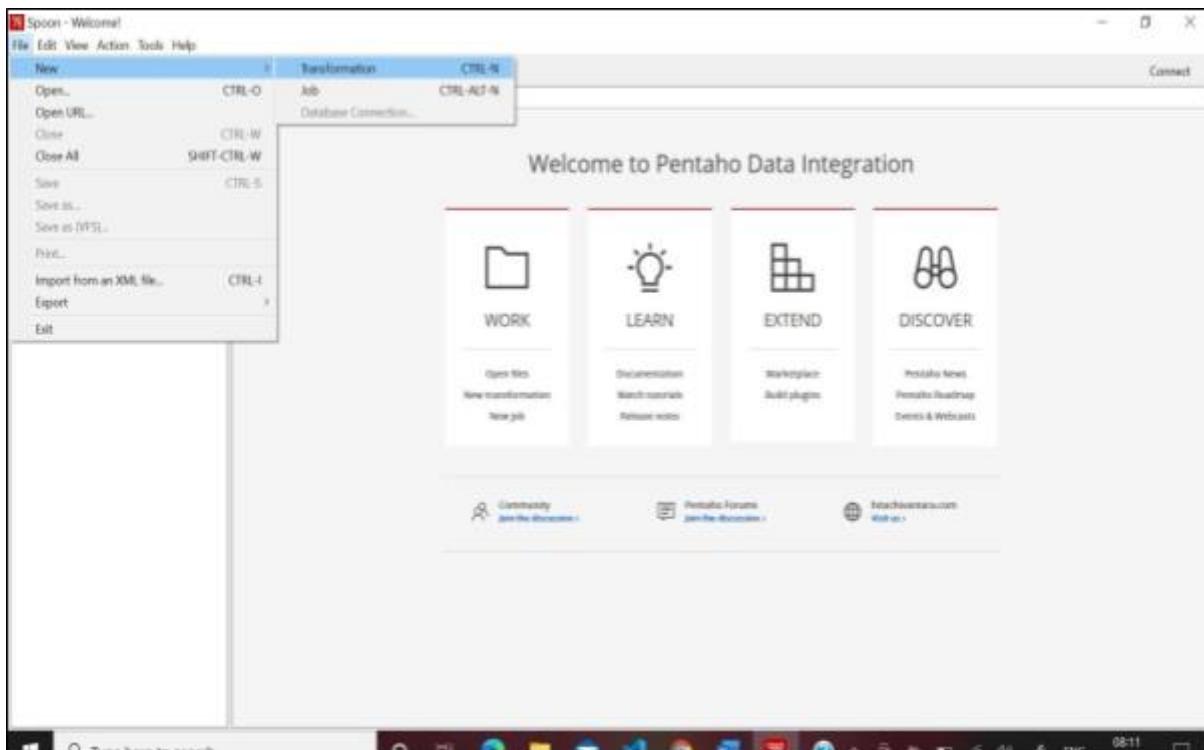


- Implement different Data Validations on Table.

Steps:

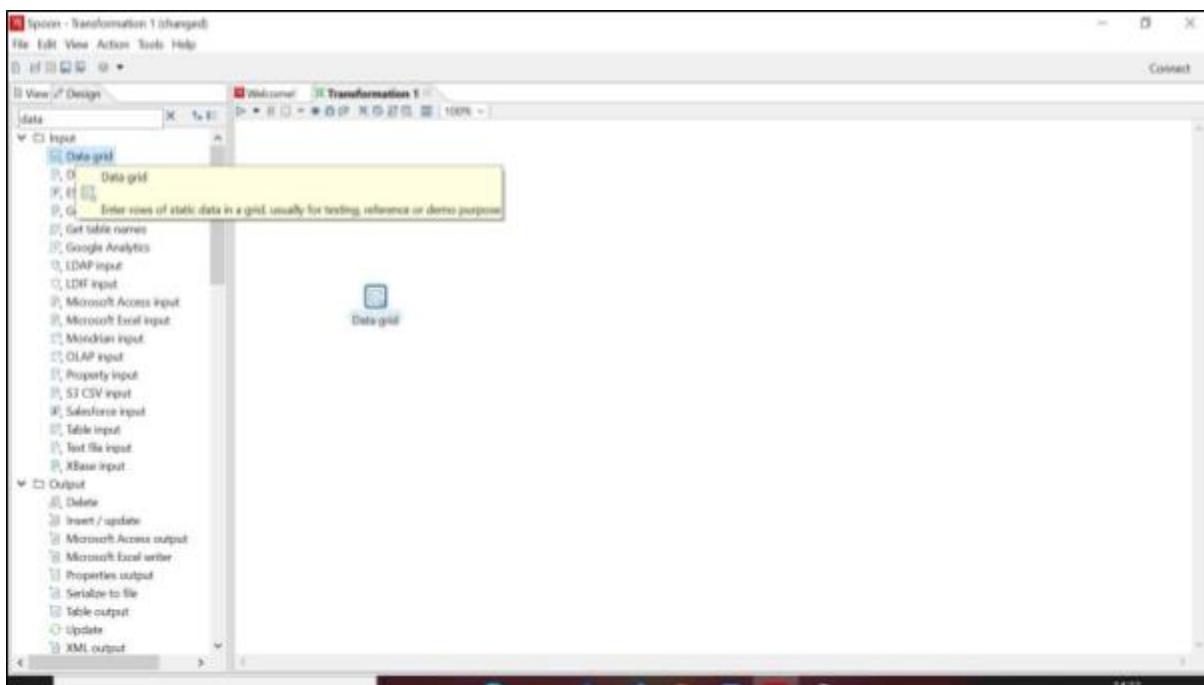
- ✓ Creating a New Transformation:

Open Pentaho Spoon -> File -> New -> Transformation (or CTRL-N)



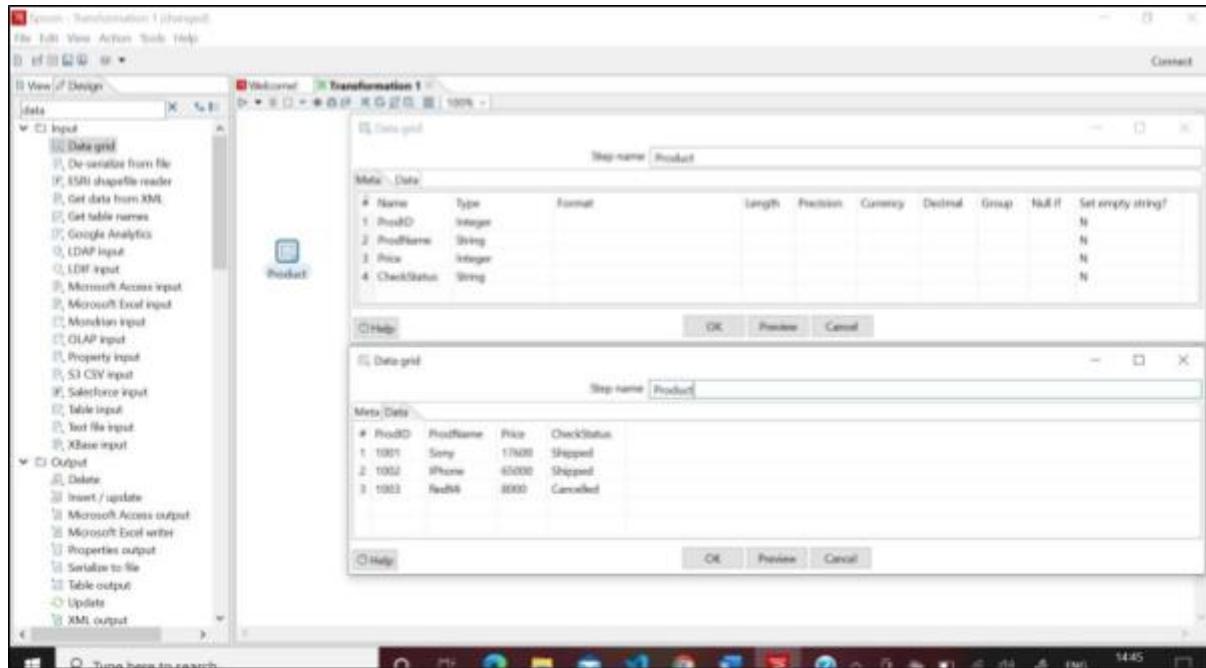
- ✓ Inserting Data Grid into the Transformation:

Design -> Input -> Data Grid (Drag & Drop in Transformation)



✓ Inserting Data in Data Grid:

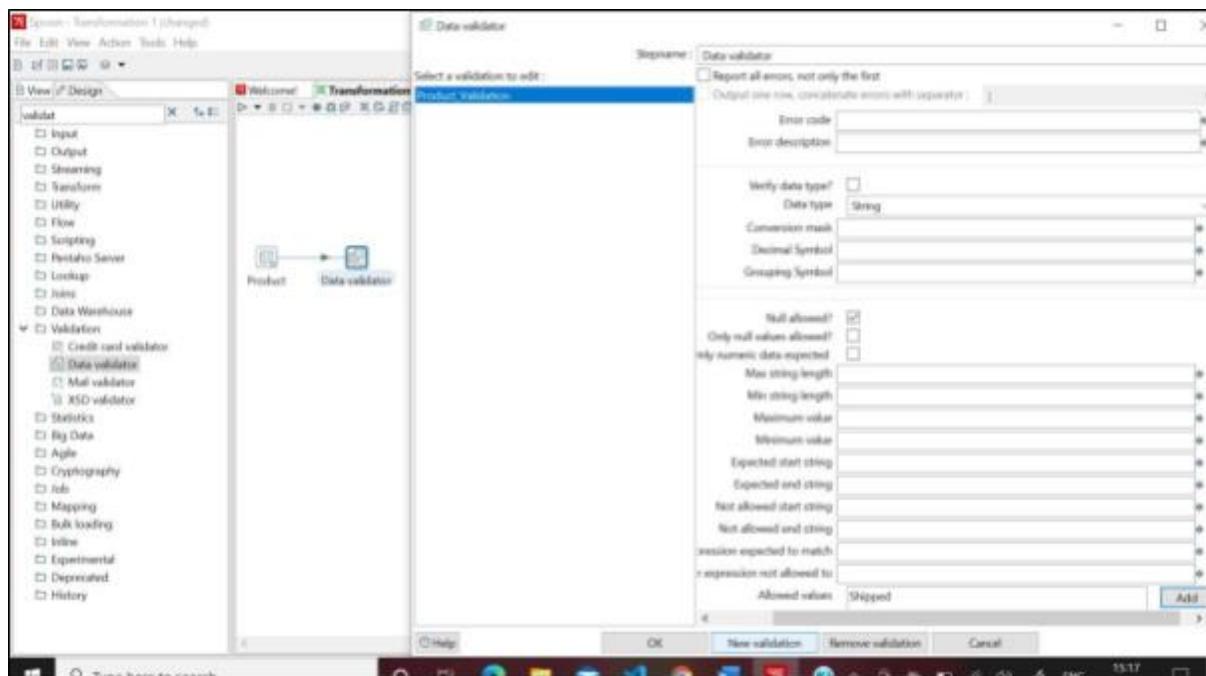
Double Click Data Grid -> Step name : *Product* ->
Meta: *Enter Fields Details* -> Data: *Enter actual Data* -> OK



✓ Inserting Data Validator into the Transformation:

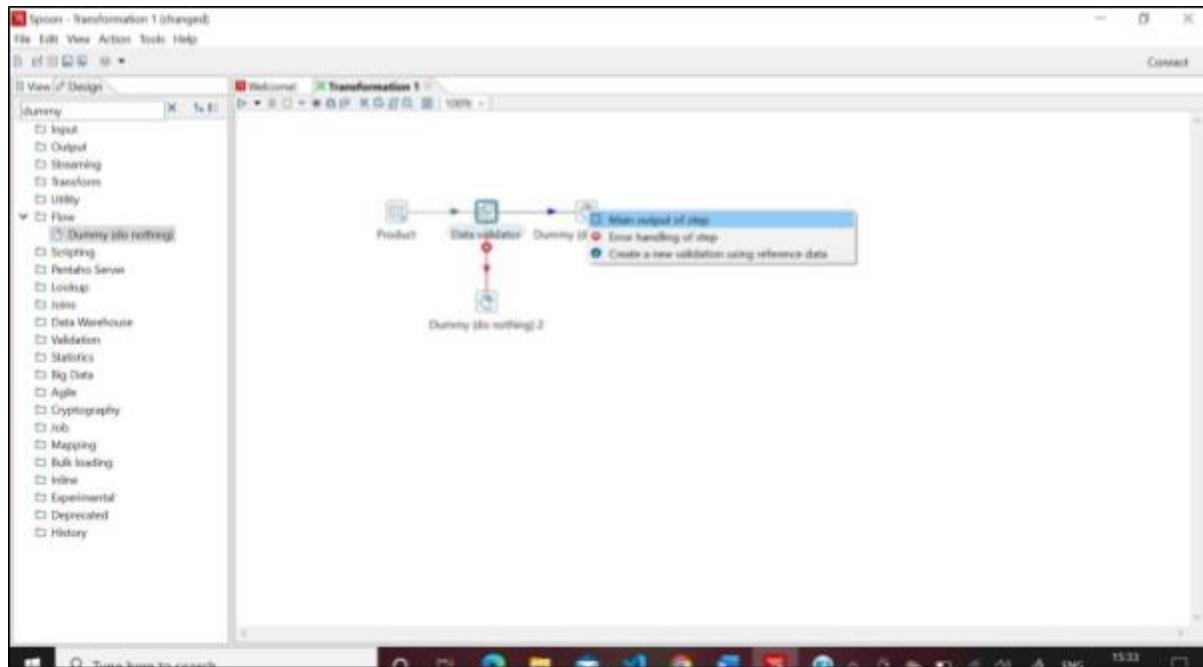
Design -> Validation -> Data Validator (Drag & Drop in Transformation) -> Create Hop Connection with Data Validator (Click on Output Connector arrow in Data Grid)

Double Click Data Validator -> Step Name: *Product_Validation* -> New validation -> Enter validation name: *Product_Validation* -> OK -> Select a validation to edit: *Click*
Product_Validation -> Name of field to validate: *CheckStatus* -> Data type: *String* -> Add Button -> Enter the allowed value to add: *Shipped* -> OK -> OK



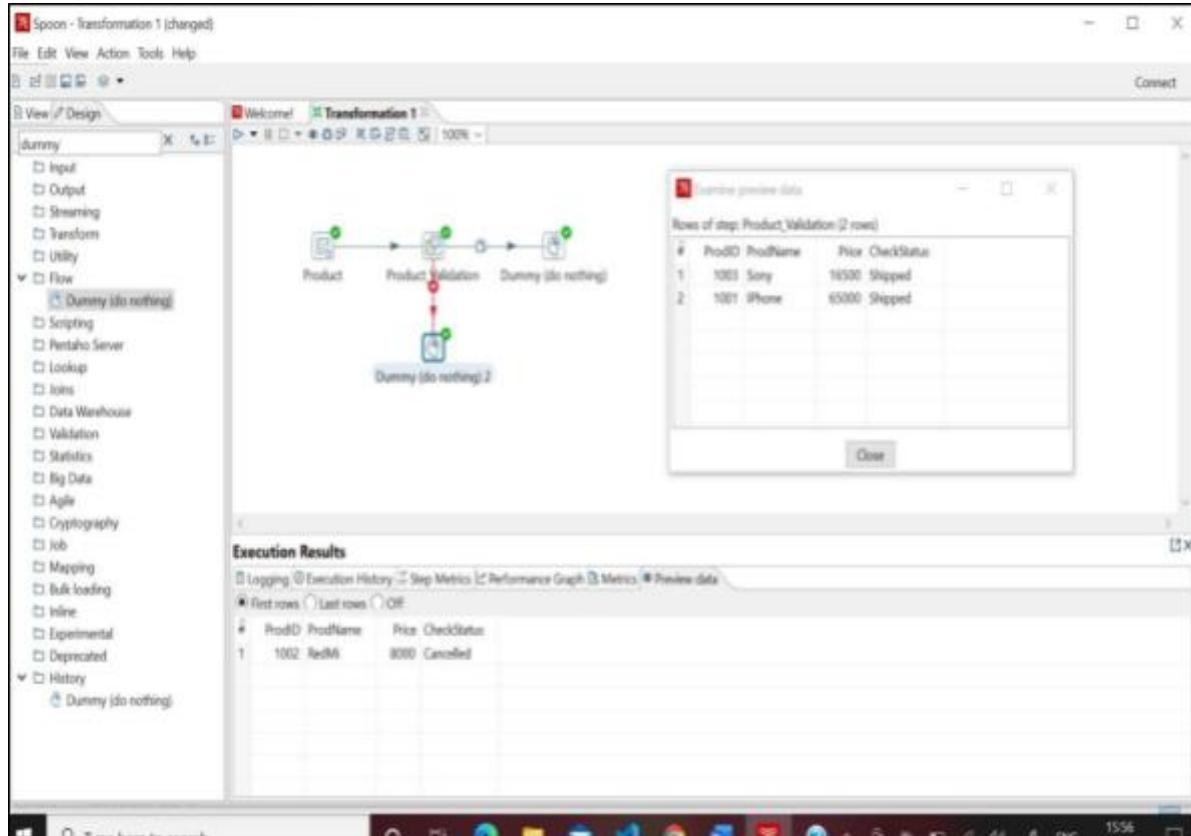
✓ Inserting **Dummy** (do nothing) into the Transformation:

Design -> Flow -> Dummy (Twice Drag & Drop in Transformation) -> Create Hop Connection with Dummy (Click on Output Connector arrow in Data Validator) -> Dummy:
Main output of step -> Dummy 2: *Error Handling of Step* -> Copy



✓ Debug Transformation:

Click Data Validator -> Debug -> Quick Launch -> Yes



Practical No. 05

Aim: Basic R Commands.

- Setting working directory, getting current working directory, listing contents of current working directory.

Commands:

✓ Setting Working Directory

```
setwd("C:/Users/Gagan/Documents")
```

✓ Getting Current Working Directory

```
getwd()
```

✓ Listing Contents of Current Working Directory

```
list.files("C:/Users/Gagan/Documents")
```

Output:

The screenshot shows the RStudio interface. The code editor window has a file named "BasicComnd.R" with the following content:

```
1 setwd("C:/Users/Gagan/Documents")
2 getwd()
3 list.files("C:/Users/Gagan/Documents")
4:1 (Top Level) ↴
```

The console tab is active, showing the following output:

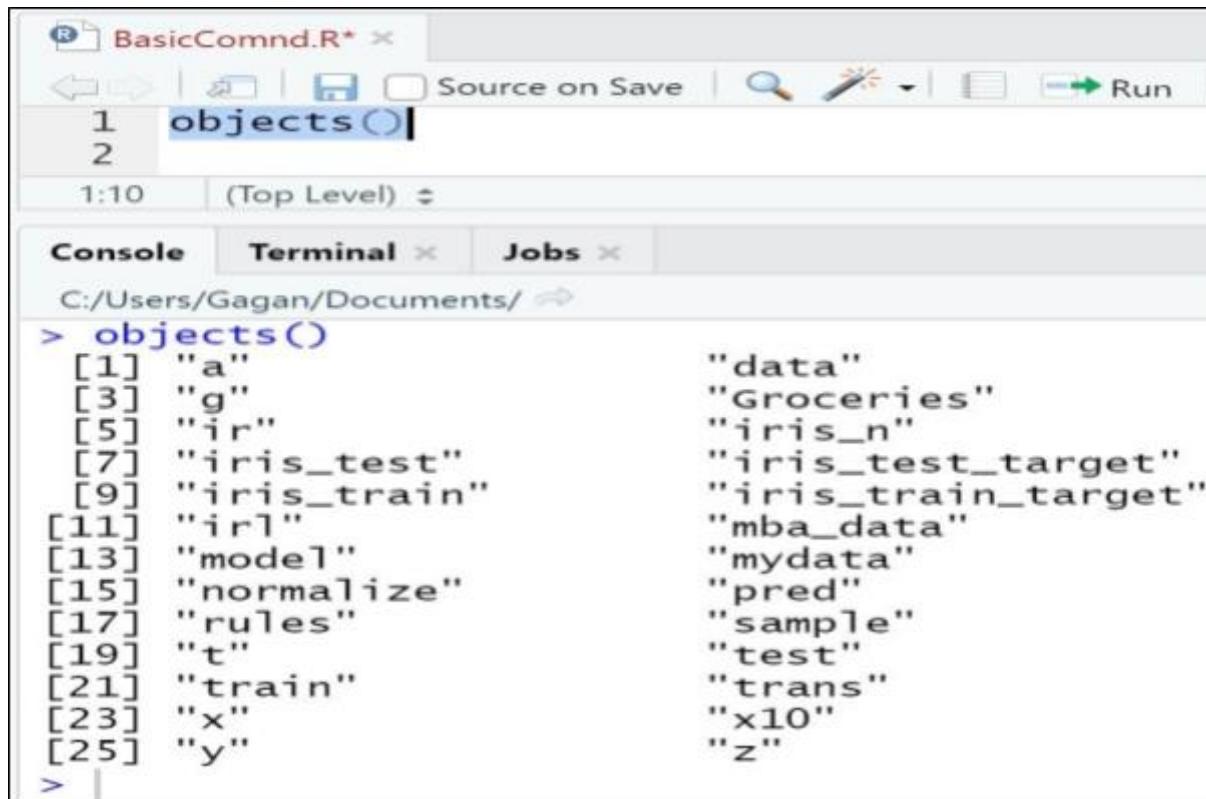
```
C:/Users/Gagan/Documents/
> setwd("C:/Users/Gagan/Documents")
> getwd()
[1] "C:/Users/Gagan/Documents"
> list.files("C:/Users/Gagan/Documents")
[1] "9778d5d219c5080b9a6a17bef029331c.pdf"
[2] "Adani Electricity Payment Receipt.pdf"
[3] "certificate.pdf"
[4] "Chapter 12-b052255ec3c806670967091b15a55b1f.pdf"
[5] "Gas.pdf"
[6] "Lecture 11_ Project Risk Management.pdf"
[7] "Mahanagar Gas.pdf"
[8] "My Music"
[9] "My Pictures"
[10] "My Videos"
[11] "pizza.jpg"
[12] "Screenshot 1 - IC"
```

- List Names of Objects in R Environment

Command:

✓ `objects()`

Output:



The screenshot shows the RStudio interface. In the top-left corner, there's a file named "BasicComnd.R*". Below the title bar, there are several icons: back, forward, file, save, and search. A "Source on Save" checkbox is checked. On the right side of the top bar are "Run" and "Help" buttons. The main area has two tabs: "Console" (which is active) and "Terminal". The "Console" tab shows the command `> objects()` and its output, which lists various object names. The "Terminal" tab shows the path `C:/Users/Gagan/Documents/`. The "Jobs" tab is also visible.

```
1  objects()
2
1:10 (Top Level) ⇣
Console Terminal × Jobs ×
C:/Users/Gagan/Documents/
> objects()
[1] "a"                      "data"
[3] "g"                      "Groceries"
[5] "ir"                     "iris_n"
[7] "iris_test"               "iris_test_target"
[9] "iris_train"              "iris_train_target"
[11] "irl"                    "mba_data"
[13] "model"                  "mydata"
[15] "normalize"              "pred"
[17] "rules"                  "sample"
[19] "t"                      "test"
[21] "train"                  "trans"
[23] "x"                      "x10"
[25] "y"                      "z"
> |
```

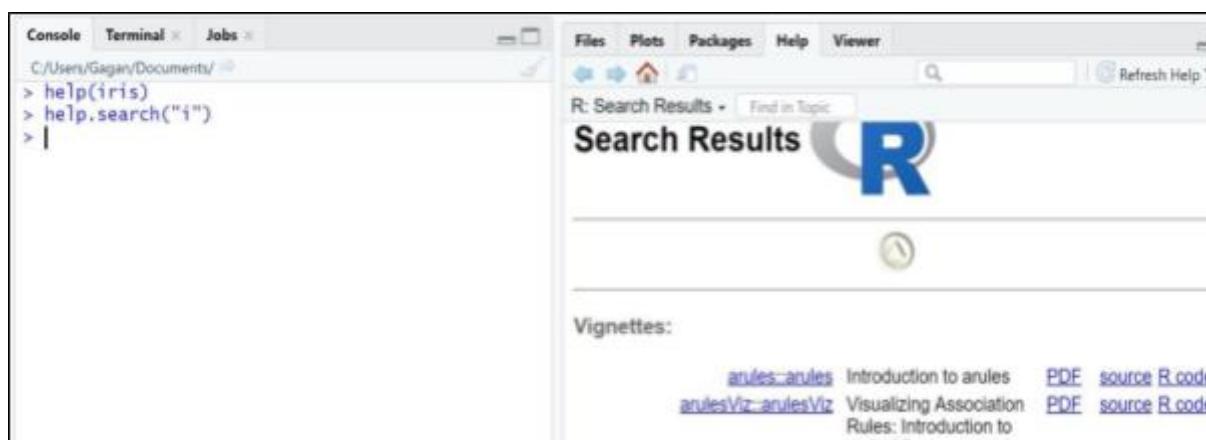
- Different Help Commands

Command:

✓ `help(iris)`

`help.search("i")`

Output:



The screenshot shows the RStudio interface with the "Help" tab selected. In the top-left corner, there's a file named "BasicComnd.R*". Below the title bar, there are several icons: back, forward, file, save, and search. A "Refresh Help" button is also present. The main area shows the "Search Results" for the query "i". It features a large blue "R" logo at the top. Below it, there's a section titled "Vignettes:" with a list of items. At the bottom, there are links for "arules" and "arulesViz".

```
Console Terminal × Jobs ×
C:/Users/Gagan/Documents/
> help(iris)
> help.search("i")
> |
```

Files Plots Packages Help Viewer

R: Search Results Find in Topic Refresh Help

Search Results

Vignettes:

arules:arules Introduction to arules PDF source R code
arulesViz:arulesViz Visualizing Association Rules: Introduction to PDF source R code

- Installing Packages (XLConnect, readxl, writexl), Displaying Installed Packages, Loading a Package

Command:

- ✓ **Installing Packages (XLConnect, readxl, writexl)**

```
install.packages("XLConnect")
install.packages("readxl")
install.packages("writexl")
```

Output:

The screenshot shows the RStudio interface with the 'BasicComnd.R' script open in the top panel. The 'Console' tab is selected, displaying the command history and its output. The user has run three commands to install packages: XLConnect, readxl, and writexl. The output shows the download of readxl from cran.rstudio.com, the unpacking of readxl, and a warning message about Rtools.

```
R BasicComnd.R*
1 install.packages("XLConnect")
2 install.packages("readxl")
3 install.packages("writexl")
4

4:1 (Top Level) : R Script

Console Terminal Jobs

C:/Users/Gagan/Documents/ ↵
R/win-library/4.0
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/readxl_1.3.1.zip'
Content type 'application/zip' length 1709039 bytes (1.6 M
B)
downloaded 1.6 MB

package 'readxl' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\Gagan\AppData\Local\Temp\RtmpwTOfQR\downlo
aded_packages
> install.packages("writexl")
WARNING: Rtools is required to build R packages but is not
currently installed. Please download and install the appro
priate version of Rtools before proceeding:
```

- ✓ **Displaying Installed Packages**

```
installed.packages()
```

Output:

The screenshot shows the RStudio interface with the 'BasicComnd.R' script open in the top panel. The 'Console' tab is selected, displaying the command 'installed.packages()' and its output. The output lists numerous installed R packages along with their versions, including lmtest, magrittr, mime, munsell, openssl, pillar, pkgconfig, plotly, prettyunits, progress, promises, proxy, purrr, qap, R6, RColorBrewer, and Rcpp.

```
R BasicComnd.R*
1 installed.packages()
1:21 (Top Level) : R Script

Console Terminal Jobs

C:/Users/Gagan/Documents/ ↵
  lmtest      NA      NA    "yes"     "4.0.4"
  magrittr    NA      NA    "yes"     "4.0.4"
  mime        NA      NA    "yes"     "4.0.4"
  munsell     NA      NA    "no"      "4.0.4"
  openssl     NA      NA    "yes"     "4.0.4"
  pillar       NA      NA    "no"      "4.0.4"
  pkgconfig    NA      NA    "no"      "4.0.4"
  plotly       NA      NA    "no"      "4.0.4"
  prettyunits  NA      NA    "no"      "4.0.4"
  progress     NA      NA    "no"      "4.0.4"
  promises     NA      NA    "yes"     "4.0.4"
  proxy        NA      NA    "yes"     "4.0.5"
  purrr       NA      NA    "yes"     "4.0.4"
  qap         NA      NA    "yes"     "4.0.4"
  R6          NA      NA    "no"      "4.0.4"
  RColorBrewer NA      NA    "no"      "4.0.3"
  Rcpp         NA      NA    "yes"     "4.0.4"
  [ reached getOption("max.print") -- omitted 54 rows ]
```

✓ **Loading a Package**

```
library(XLConnect)
library(readxl)
library(writexl)
```

Output:

The screenshot shows the RStudio interface. In the top-left pane, there is an R script titled "BasicComnd.R". The code in the script is:

```
1 library(XLConnect)
2 library(readxl)
3 library(writexl)
```

In the bottom-right pane, the "Console" tab is active, showing the command-line history and the results of running the script:C:/Users/Gagan/Documents/
> library(readxl)
> library(writexl)
>

- Creating a Variable, Assigning Values to a Variable, Printing Variable Value

Command:

✓ **Creating a Variable, Assigning Values to a Variable**

```
val<-c (3,5,7,9)          OR
assign("vals", c (3,5,7,9)) OR c
(3,5,7,9) ->values
```

✓ **Printing Variable Value**

```
val                      #Auto Printing
print(vals)               #External Printing
```

Output:

The screenshot shows the RStudio interface. In the top-left pane, there is an R script titled "BasicComnd.R". The code in the script is:

```
1 val<-c (3,5,7,9)
2 assign("vals", c (3,5,7,9))
3 c (3,5,7,9) ->values
4
5 val
6 print(vals)
```

In the bottom-right pane, the "Console" tab is active, showing the command-line history and the results of running the script:C:/Users/Gagan/Documents/
> val<-c (3,5,7,9)
> assign("vals", c (3,5,7,9))
> c (3,5,7,9) ->values
> val
[1] 3 5 7 9
> print(vals)
[1] 3 5 7 9

- Performing various Calculations in R (Addition, Subtraction, Multiplication, Division, Square Root, Square, etc..)

Command:

✓ 45+28	#Addition	sqrt(64)	#Square Root
s<-45-28	#Subtraction	8^2	#Square
s		log(2)	#Logarithm
4*5	#Multiplication	exp(8)	#Exponent
8/2	#Division	factorial(5)	#Factorial

Output:

The screenshot shows the RStudio interface with the 'Console' tab active. The workspace shows the results of the following R commands:

```
C:/Users/Gagan/Documents/ ↵
> 45+28
[1] 73
> s<-45-28
> s
[1] 17
> 4*5
[1] 20
> 8/2
[1] 4
> sqrt(64)
[1] 8
> 8^2
[1] 64
> log(2)
[1] 0.6931472
> exp(8)
[1] 2980.958
> factorial(5)
[1] 120
```

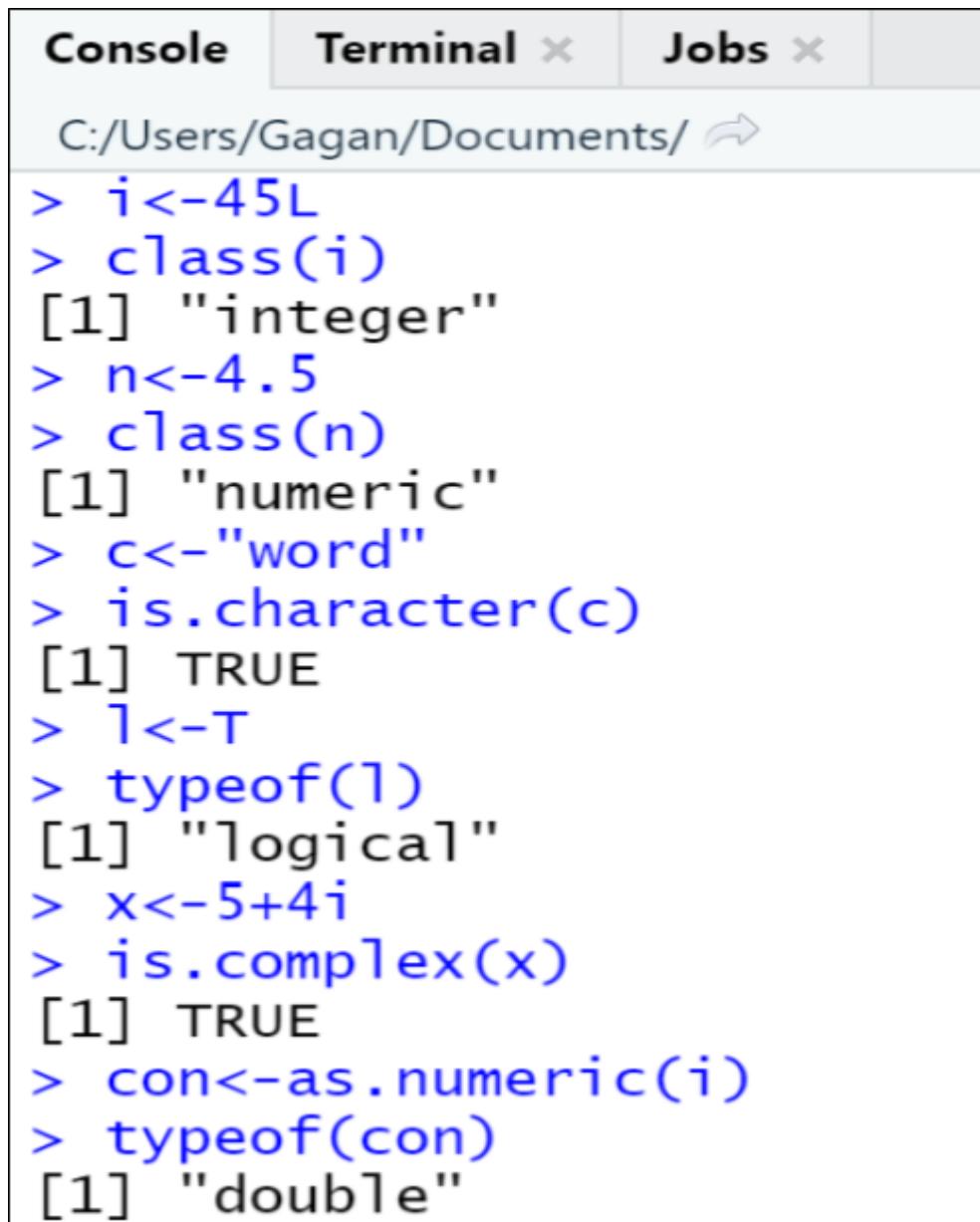
- Data Types in R (Integer, Numeric, Character, Logical, Complex), Checking Type of a Variable (using typeof() and Class()), Converting a Variable from one type to another.

Command:

✓ i<-45L #Integer
class(i) #Checking Type
n<-4.5 #Numeric
c<-"word" #Character
is.character(c) #Checking Type

l<-T #Logical
typeof(l) #CheckingType
x<-5+4i #Complex
con<-as.numeric(i) #Converting Type

Output:



The screenshot shows the RStudio interface with the 'Console' tab selected. The workspace pane displays the following R session history:

```
C:/Users/Gagan/Documents/ ↵
> i<-45L
> class(i)
[1] "integer"
> n<-4.5
> class(n)
[1] "numeric"
> c<-"word"
> is.character(c)
[1] TRUE
> l<-T
> typeof(l)
[1] "logical"
> x<-5+4i
> is.complex(x)
[1] TRUE
> con<-as.numeric(i)
> typeof(con)
[1] "double"
```

- Creating Vectors and Performing Vector Operations.

Command:

✓ v<-c(3,5,8,9) #Creating Vector
 vt<-vector(mode="logical",45) #Creating Vector print(v)
 #Accessing vector
 sort.vt<-sort(v,decreasing = TRUE) #Manipulating Vector

Output:

```
> v<-c(3,5,8,9)
> vt<-vector(mode="logical",45)
> typeof(vt)
[1] "logical"
> print(v)
[1] 3 5 8 9
> sort.vt<-sort(v,decreasing = TRUE)
> print(sort.vt)
[1] 9 8 5 3
```

- Creating Matrix, using different Methods like matrix(), cbind() and rbind(), Checking Dimension and Attributes of Matrix, Showing Matrix by Row, Performing various Matrix Operations (Mathematical Operations using Scalar Values, Matrix Addition, Subtraction, Multiplication, Division, Transpose of Matrix)

Command:

✓ Creating Matrix, using different Methods like matrix(), cbind() and rbind()

```
m<-matrix(c(1,2,3,4,5,6,7,8,9),nrow = 3,ncol = 3) #Creating Matrix
matrix(m)
c<-cbind(m1,m2)
r<-rbind(m1,m2)
```

Output:

```
> m<-matrix(c(1,2,3,4,5,6,7,8,9),nrow = 3,ncol = 3)
> matrix(m)
[,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
[7,]    7
[8,]    8
[9,]    9
> m1<-c(12,24,35)
> m2<-c(56,67,78)
> c<-cbind(m1,m2)
> c
     m1  m2
[1,] 12  56
[2,] 24  67
[3,] 35  78
> r<-rbind(m1,m2)
> r
[,1] [,2] [,3]
m1    12   24   35
```

✓ **Checking Dimension and Attributes of Matrix, Showing Matrix by Row,**

Performing various Matrix Operations

```
dim(m)           #Dimension  
attributes(m)   #Attributes  
mt<-matrix(c(18,24,33,56,65,68,10,26,44),nrow=3,ncol=3,byrow=TRUE)  #By Row  
m+mt            #Matrix Addition          mt-m          #Matrix Subtraction  
m*mt            #Matrix Multiplication      mt/m          #Matrix Division  
t(mt)           #Matrix Transpose
```

Output:

```
> dim(m)  
[1] 3 3  
> attributes(m)  
$dim  
[1] 3 3  
  
> mt<-matrix(c(18,24,33,56,65,68,10,26,44),nrow = 3,ncol =  
3,byrow = TRUE)  
> mt  
     [,1] [,2] [,3]  
[1,]    18    24    33  
[2,]    56    65    68  
[3,]    10    26    44  
> m+mt  
     [,1] [,2] [,3]  
[1,]    19    28    40  
[2,]    58    70    76  
[3,]    13    32    53  
> mt-m  
     [,1] [,2] [,3]  
[1,]    17    20    26  
[2,]    54    60    60  
[3,]     7    20    35
```

```
> m*mt  
     [,1] [,2] [,3]  
[1,]    18    96   231  
[2,]   112   325   544  
[3,]    30   156   396  
> mt/m  
     [,1]      [,2]      [,3]  
[1,] 18.000000  6.000000 4.714286  
[2,] 28.000000 13.000000 8.500000  
[3,]  3.333333  4.333333 4.888889  
> t(mt)  
     [,1] [,2] [,3]  
[1,]    18    56    10  
[2,]    24    65    26  
[3,]    33    68    44  
> t(m)  
     [,1] [,2] [,3]  
[1,]     1     2     3  
[2,]     4     5     6  
[3,]     7     8     9
```

- Creating List, Accessing List Element.

Command:

✓ list<-c("mango","banana","others")) #Creating List
print(list) #Accessing List
print(list[2]) Element

Output:

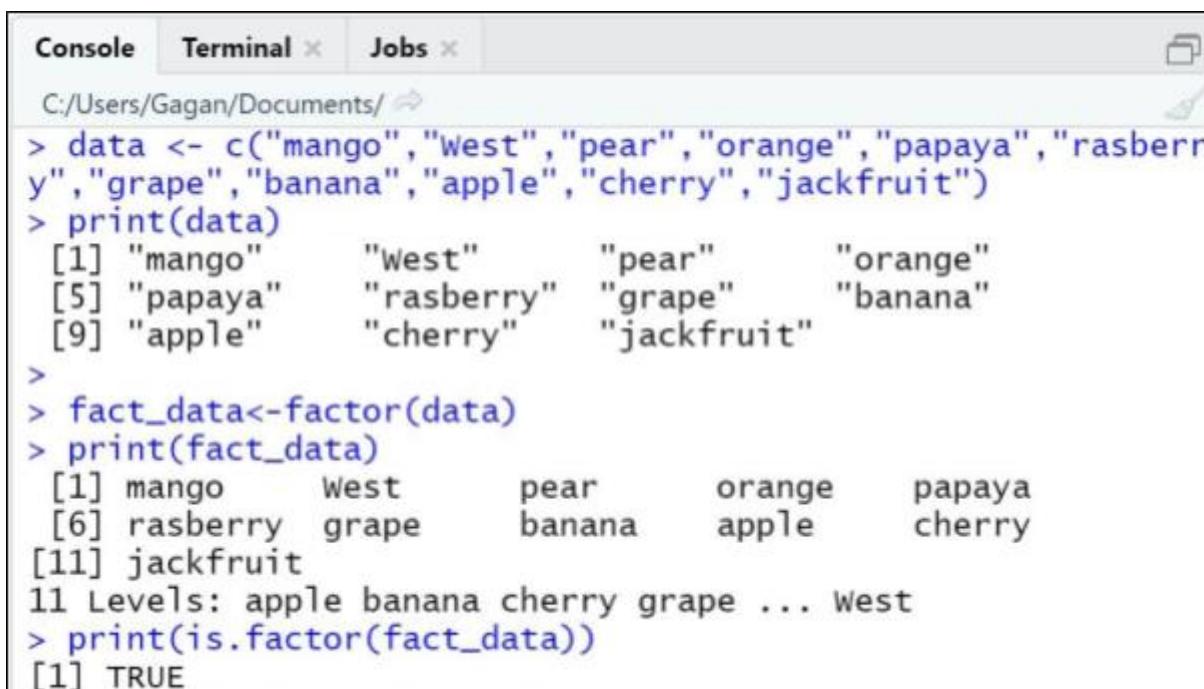
```
> list<-c("mango","banana","others"))
> print(list)
[1] "mango"  "banana" "others"
> print(list[2])
[1] "banana"
> |
```

- Creating Factors.

Command:

✓ data <- c ("mango", "West", "pear", "orange", "papaya", "rasberry", "grape",
"banana", "apple", "cherry", "jackfruit") #Creating factors
print(data)
fact_data<-factor(data)
print(fact_data)
print(is.factor(fact_data))

- **Output:**



```
Console Terminal × Jobs ×
C:/Users/Gagan/Documents/
> data <- c("mango", "West", "pear", "orange", "papaya", "rasberry", "grape", "banana", "apple", "cherry", "jackfruit")
> print(data)
[1] "mango"      "West"       "pear"       "orange"      "papaya"
[5] "rasberry"   "grape"     "banana"    "apple"      "cherry"
[9] "jackfruit"
>
> fact_data<-factor(data)
> print(fact_data)
[1] mango      West       pear      orange     papaya
[6] raspberry  grape     banana    apple     cherry
[11] jackfruit
11 Levels: apple banana cherry grape ... West
> print(is.factor(fact_data))
[1] TRUE
```

- Creating Data Frames, Checking Number of Rows, Number of Columns, Giving Names to Columns and Rows.

Command:

✓ `empid<-c(101,102,103)`
`empnm<-c("rach","pheebs","mon")`
`dtfrm<-data.frame(empid,empnm)` #Creating Data Frame
`dtfrm`
`nrow(dtfrm)` #Checking Number of Rows
`ncol(dtfrm)` #Checking Number of Columns
`names(dtfrm)[1]<-"stdid"` #Giving Names to
`names(dtfrm)[2]<-"stdnm"` Columns
`names(dtfrm)`

Output:

```
> empid<-c(101,102,103)
> empnm<-c("rach","pheebs","mon")
>
> dtfrm<-data.frame(empid,empnm)
>
> dtfrm
  empid   empnm
1    101     rach
2    102   pheebs
3    103      mon
> dtfrm$empid
[1] 101 102 103
> dtfrm$empnm
[1] "rach"   "pheebs" "mon"
> nrow(dtfrm)
[1] 3
> ncol(dtfrm)
[1] 2
> names(dtfrm)
[1] "empid" "empnm"
> names(dtfrm)[1]<-"stdid"
> names(dtfrm)[2]<-"stdnm"
> names(dtfrm)
[1] "stdid" "stdnm"
```

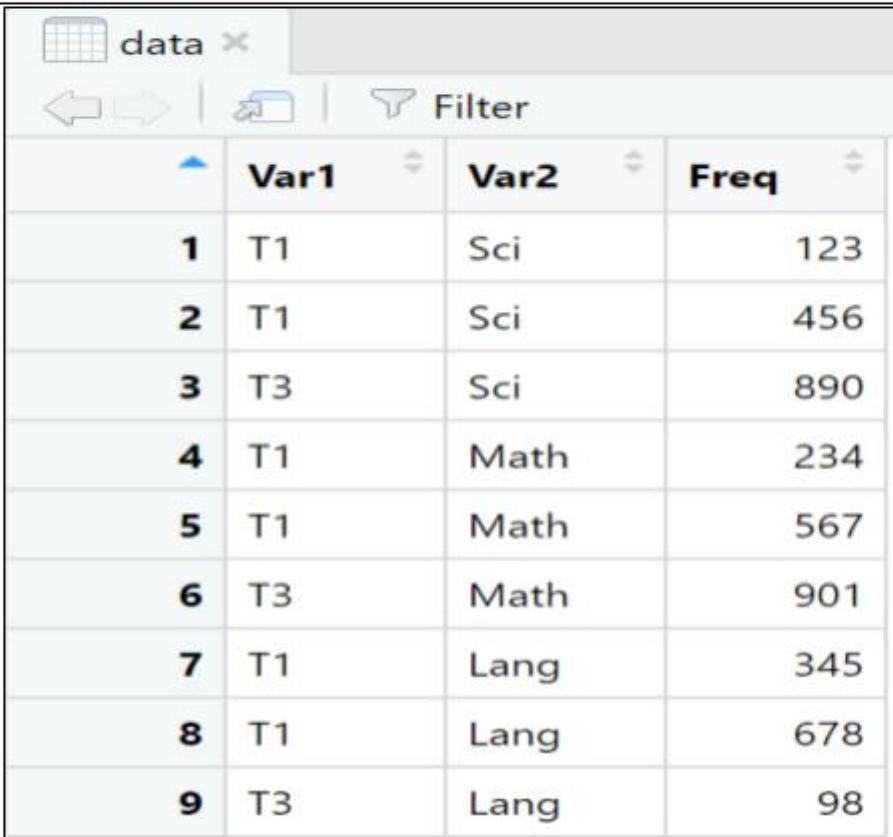
- Creating Table.

Command:

✓ `data<-matrix(c(123,234,345,456,567,678,890,901,098),ncol=3,byrow=TRUE)`
`colnames(data)<- c("Sci","Math","Lang")`
`rownames(data)<- c("T1","T1","T3")`
`data<-as.table(data)` #Creating Table
`data`
`View(data)` #Accessing table

Output:

```
> data<-matrix(c(123,234,345,456,567,678,890,901,098),ncol=3,byrow=TRUE)
> data
     [,1] [,2] [,3]
[1,] 123  234  345
[2,] 456  567  678
[3,] 890  901  98
> colnames(data)<- c("Sci","Math","Lang")
> rownames(data)<- c("T1","T1","T3")
> data<-as.table(data)
> data
  Sci Math Lang
T1 123  234  345
T1 456  567  678
T3 890  901  98
> View(data)
```



The screenshot shows the RStudio interface with a data grid titled "data". The grid has four columns: "Var1", "Var2", and "Freq", plus a primary key column labeled with numbers 1 through 9. The data is as follows:

	Var1	Var2	Freq
1	T1	Sci	123
2	T1	Sci	456
3	T3	Sci	890
4	T1	Math	234
5	T1	Math	567
6	T3	Math	901
7	T1	Lang	345
8	T1	Lang	678
9	T3	Lang	98

- Reading and Writing Data from .CSV File, Showing Dimensions, Showing First 5 Rows, Showing Last 5 Rows.

Command:

- ✓ Reading and Writing Data from .CSV File

```
cpInfo<-read.csv("Info.csv")           #Reading from .csv
write.csv(data,"Info.csv")            #Writing to .csv
newInfo<-read.csv("Info.csv")
```

Output:

```
> cpInfo<-read.csv("Info.csv")
> print(cpInfo)
  Name Class Add
1  Rachel    9   NY
2    Ross    12   NY
3 Monica     9   NY
4 Chandler   12   NY
5  Joey     11  Itlay
6 Pheobe    10  Texas
> write.csv(data,"Info.csv")
> newInfo<-read.csv("Info.csv")
> print(newInfo)
  X PR.Sl.no Material
1 1      25 250006342
2 2      4 280005173
                                         Short.Text
                                         PISTON RINGS
1 CAPACITY CONTROL VALVE ASSEMBLY (FOR 5H*
2 Manufacturer.Part.No. Quantity Unit.of.Measure Cutter
1          06EA500131       18        NO  5.51
2          5H121-417        1        NO 964.65
  Cutter.Total RC.Rates.
1      99.18      5.4
2      964.65    407.5
```

- ✓ Showing Dimensions, Showing First 5 Rows, Showing Last 5 Rows

```
cpInfo<-read.csv("Stud.csv")           dim(cpInfo)      #Dimensions
dtfrm<-data.frame(cpInfo)
head(dtfrm,5)                         #First 5 Rows
tail(dtfrm,5)                        #Last 5 Rows
```

Output:

```
> cpInfo<-read.csv("Stud.csv")
> dim(cpInfo)
[1] 11  3
> dtfrm<-data.frame(cpInfo)
> head(dtfrm, 5)
  Name Year Profession
1  Ross  28  Teaching
2 Monica 25  Cooking
3  Joey  27    Actor
4 Pheobe 26 Physician
5 Chandler 28      IT
> tail(dtfrm, 5)
  Name Year Profession
7  Penny 26 Marketing
8 Sheldon 27 Physicist
9 Leonard 28 Physicist
10 Howard 28 Astronaut
11  Rai  27 Astrophysicist
```

- Reading and Writing Data from Excel using XLConnect.

Command:

✓ library(XLConnect)

```
xlFile=XLConnect::readWorksheetFromFile("Demo.xlsx",sheet=1)
xlFile
data=XLConnect::readWorksheetFromFile("About.xlsx",sheet=1)
writeFile=XLConnect::writeWorksheetToFile("Demo.xlsx",data,sheet = "About")
xlFile=XLConnect::readWorksheetFromFile("Demo.xlsx",sheet="About")
xlFile
```

Output:

```
object 'csvFile1' not found
> library(XLConnect)
> xlFile=XLConnect::readworksheetFromFile("Demo.xlsx",sheet=1)
> xlFile
  Name. Rollno
1 Pankaj      10
2 Sunil       20
3 Lekha       30
> |

> writeFile=XLConnect::writeworksheetToFile("Demo.xlsx",data,sheet = "About")
> xlFile=XLConnect::readworksheetFromFile("Demo.xlsx",sheet="About")
> xlFile
  PR.Sl.no Material                      Short.Text
1      25 250006342                      PISTON RINGS
2      4 280005173 CAPACITY CONTROL VALVE ASSEMBLY (FOR 5H*
  Manufacturer.Part.No. Quantity Unit.of.Measure Cutter.Cutter.Total
1      06EA500131        18            NO 5.51    99.18
2      5H121-417         1             NO 964.65   964.65
  RC.Rates.
1      5.4
2     407.5
> |
```

- Reading and Writing Data from Excel using readXL and writeXL.

Command:

```
✓ library(readxl) #Loading 'readxl' Package    library(writexl) #Loading 'writexl'
  xlsmp<-read_xlsx("About.xlsx",sheet=1)          #Reading using readXL
  wrxl<-write_xlsx(data,"About.xlsx")            #Writing using writeXL
```

Output:

```
> library(readxl)
> library(writexl)
> xlsmp<-read_xlsx("About.xlsx",sheet = 1)
> print(xlsmp)
# A tibble: 11 × 3
  Name      Year Profession
  <chr>     <dbl> <chr>
1 Ross        28  Teaching
2 Monica      25  Cooking
3 Joey         27  Actor
4 Pheobe       26  Physician
5 Chandler     28  IT
6 Rachel       25  Fashion
7 Penny         26  Marketing
8 Sheldon       27  Physicist
9 Leonard       28  Physicist
10 Howard        28  Astronout
11 Raj           27  Astrophysicist
> wrxl<-write_xlsx(data,"About.xlsx")
```

PR.Sl.no	Material	Short.Text	Manufacturer.Part.No.	Quantity	Unit.of.Measure	Cutter	Cutter.Total	RC.Rates.
25	250006342	PISTON RINGS	06EA500131	18	NO	5.51	99.18	5.4
4	280005173	CAPACITY CONTROL VALVE ASSEMBLY (FOR SH*	5H121-417	1	NO	964.65	964.65	407.5

Practical No. 06

Aim: Implementation of Data pre-processing techniques like,

- Naming and Renaming variables, adding a new variable.
- Dealing with missing data.
- Dealing with categorical data.
- Data reduction using sub-setting.

- Naming and Renaming variables, adding a new variable.

✓ my_data<-mtcars

```
> my_data<-mtcars
> head(mtcars,5)
      mpg cyl disp hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0    3    2
```

✓ my_data<-my_data[1:6,1:5]

```
require(dplyr)
my_data<-rename(my_data,horse_power=hp)
my_data$new_hp<-my_data$horse_power*0.5
```

```
> my_data<-rename(my_data,horse_power=hp)
> # adding new variable
> my_data$new_hp<-my_data$horse_power*0.5
> colnames(my_data)
[1] "mpg"          "cyl"          "disp"         "horse_power" "drat"
[6] "new_hp"
> my_data
      mpg cyl disp horse_power drat new_hp
Mazda RX4     21.0   6 160        110 3.90  55.0
Mazda RX4 Wag 21.0   6 160        110 3.90  55.0
Datsun 710    22.8   4 108        93 3.85  46.5
Hornet 4 Drive 21.4   6 258        110 3.08  55.0
Hornet Sportabout 18.7   8 360        175 3.15  87.5
Valiant       18.1   6 225        105 2.76  52.5
```

✓ data <- read.table(file="missing_col.csv", sep = ",")

```
data <- read.table(file="missing_col.csv", sep =
",",col.names=c("Sno","NAME","SALARY","Date_Of_Joining","Department"))
data
```

```
> data
  Sno      NAME  SALARY Date_of_Joining Department
1   1      Rick  623.30    01/01/2012        IT
2   2      Dan   515.20   23/09/2013  Operations
3   3  Michelle  611.00   15/11/2014        IT
4   4      Ryan  729.00   11/05/2014        HR
5   NA     Gary  843.25   27/03/2015  Finance
6   6      Nina    NA    21/05/2013        IT
7   7     Simon  632.80   30/07/2013  Operations
8   8      Guru  722.50   17/06/2014  Finance
9   9      John    NA    21/05/2012
10  10     Rock  600.80   30/07/2013        HR
11  11     Brad 1032.80   30/07/2013  Operations
12  12      Ryan  729.00   11/05/2014        HR
```

□ Dealing with missing data.

- Error Detection and Correction NA: Not Available - Known as missing values
- Works as a place holder for something that is ‘missing’
- Most basic operations (addition, subtraction, multiplication, etc.) in R deal with it without crashing and return NA if one of the inputs is NA
- `is.na(VALUE)` is used to check if the input value is NA or not.
- Returns a TRUE/FALSE vector whereas in case of Excel like utilities for numeric computations it’s assumed to be 0

□ Using `na.rm()`: This will keep NA rows in data while removes them during calculation

- ✓ e.g. `V <- c(1,2,NA,3) median(V, na.rm = T)`

□ Data reduction using subsetting

✓ `complete.cases()`

Return a logical vector indicating which cases are complete, i.e., have no missing values.
Syntax: `complete.cases(...)`

It takes a sequence of vectors, matrices and data frames as arguments.

E.g. `V <- c(1,2,NA,3)`
`V [complete.cases(V)]` [1] 1 2 3

✓ `is.na()` logical indexing: indicates which elements are missing

- e.g. `naVals <- is.na(V)` `V[!naVals]` [1] 1 2 3

Detect if there are any NAs: `any(is.na(datan))` Identify positions of NAs: `which(is.na(datan$v1))`

□ Imputation: The process of estimating or deriving missing values

Missing data imputation is a statistical method that replaces missing data points with substituted values.

There are various methods for imputation

– Imputation of the mean – Imputation of the median – Imputation using linear regression models

- Package Hmisc implements many imputation methods

```
✓ library(Hmisc)  
x = c(1,2,3,NA,4,4,NA)  
# mean imputation - from package, mention name of function to be used
```

✓ v<-impute(x,fun=mean) v

```
> v<-impute(x,fun=mean)
> v
 1 2 3 4 5 6 7
 1 2 3 3* 4 3* 5
```

✓ v<-impute(x,fun=median) v

```
> v<-impute(x,fun=median)
> v
 1 2 3 4 5 6 7
 1 2 3 3* 4 3* 5
```

✓ v<-impute(data1\$SRNO,fun=mean) v

```
> v<-impute(data1$SRNO,fun=mean)
> v
 1 2 3 4 5 6 7 8 9 10 11
 2.0 3.0 4.0 7.2* 6.0 7.0 8.0 9.0 10.0 11.0 12.0
```

✓ v<-impute(data1\$SALARY,fun=median) v

```
> v<-impute(data1$SALARY,fun=median)
> v
 1 2 3 4 5 6 7 8 9
 515.20 611.00 729.00 843.25 722.50* 632.80 722.50 722.50* 600.80
 10 11
 1032.80 729.00
```

□ Dealing with categorical data.

✓ c1<-c("low","medium","high","medium","low")

```
c1<-factor(c1,levels =c("low","medium","high")) c1
```

```
> c1<-c("low","medium","high")
> c1<-factor(c1,levels=c("low","medium","high"))
> c1
[1] low   medium high
Levels: low medium high
```

□ Data reduction using sub-setting.

- ✓ `data1<-read.csv("missing_col.csv",sep=",",col.names = c("SRNO","NAME","SALARY","DOJ","DEPARTMENT"))`
- ✓ `View(data1)`

▲	SRNO	NAME	SALARY	DOJ	DEPARTMENT
1	2	Dan	515.20	23/09/2013	Operations
2	3	Michelle	611.00	15/11/2014	IT
3	4	Ryan	729.00	11/05/2014	HR
4	NA	Gary	843.25	27/03/2015	Finance
5	6	Nina	NA	21/05/2013	IT
6	7	Simon	632.80	30/07/2013	Operations
7	8	Guru	722.50	17/06/2014	Finance
8	9	John	NA	21/05/2012	
9	10	Rock	600.80	30/07/2013	HR
10	11	Brad	1032.80	30/07/2013	Operations
11	12	Ryan	729.00	11/05/2014	HR

- ✓ `x<-c(1,2,3,NA,4,NA,5)`

x

```
> x<-c(1,2,3,NA,4,NA,5)
> x
[1] 1 2 3 NA 4 NA 5
```

- ✓ `xn<-is.na(x)`
- `x[!xn]`
- `NA+4`

```
> xn<-is.na(x)
> x[!xn]
[1] 1 2 3 4 5
> NA+4
[1] NA
```

- ✓ `median(x,na.rm=T)`

```
> median(x,na.rm=T)
[1] 3
```

- ✓ `complete.cases(x)`

```
> complete.cases(x)
[1] TRUE TRUE TRUE FALSE TRUE FALSE TRUE
```

✓ is.na(data1)

```
> is.na(data1)
   SRNO NAME SALARY DOJ DEPARTMENT
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE
[4,] TRUE FALSE FALSE FALSE FALSE
[5,] FALSE FALSE TRUE FALSE FALSE
[6,] FALSE FALSE FALSE FALSE FALSE
[7,] FALSE FALSE FALSE FALSE FALSE
[8,] FALSE FALSE TRUE FALSE FALSE
[9,] FALSE FALSE FALSE FALSE FALSE
[10,] FALSE FALSE FALSE FALSE FALSE
[11,] FALSE FALSE FALSE FALSE FALSE
```

✓ datacompletecases<-data1[complete.cases(data1),]

datacompletecases

```
> datacompletecases<-data1[complete.cases(data1),]
> datacompletecases
   SRNO      NAME SALARY      DOJ DEPARTMENT
1      2        Dan  515.2 23/09/2013 Operations
2      3    Michelle  611.0 15/11/2014          IT
3      4       Ryan  729.0 11/05/2014          HR
6      7     Simon  632.8 30/07/2013 Operations
7      8      Guru  722.5 17/06/2014 Finance
9     10      Rock  600.8 30/07/2013          HR
10     11     Brad 1032.8 30/07/2013 Operations
11     12      Ryan  729.0 11/05/2014          HR
```

✓ any(is.na(x))

which(is.na(data1\$SRNO))

```
> any(is.na(x))
[1] TRUE
> which(is.na(data1$SRNO))
[1] 4
```

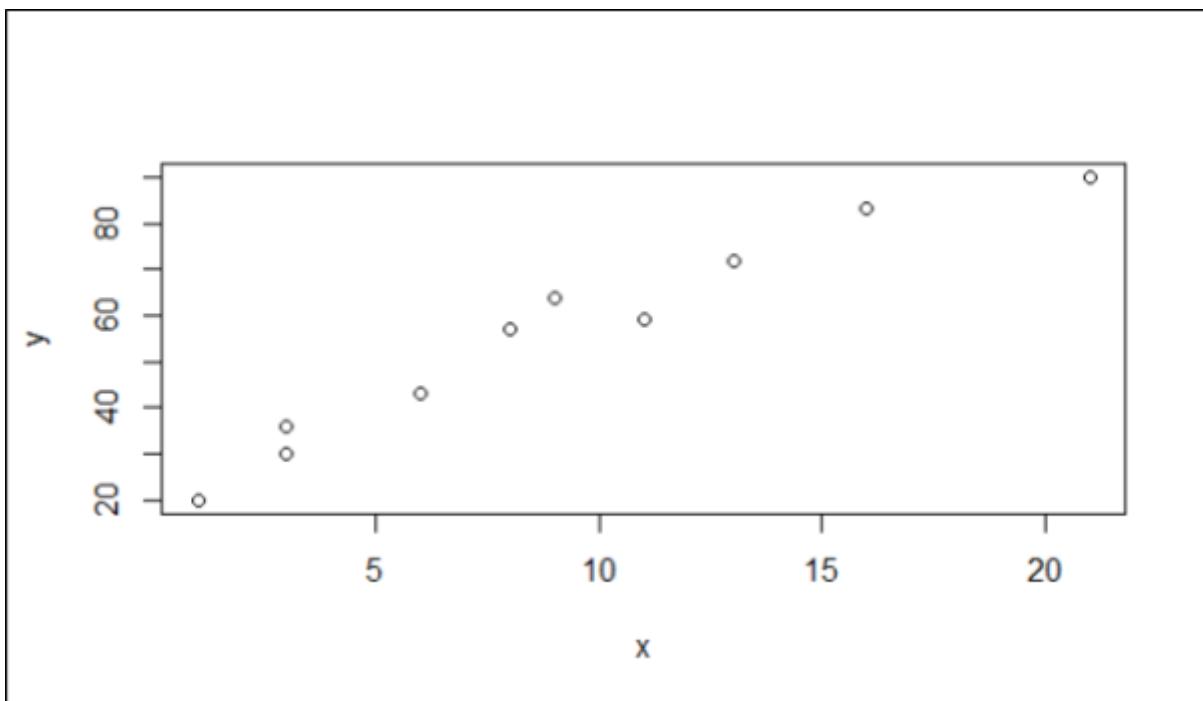
✓ Na.omit(x)

```
> na.omit(x)
[1] 1 2 3 4 5
attr(,"na.action")
[1] 4 6
attr(,"class")
[1] "omit"
```

Practical No. 07

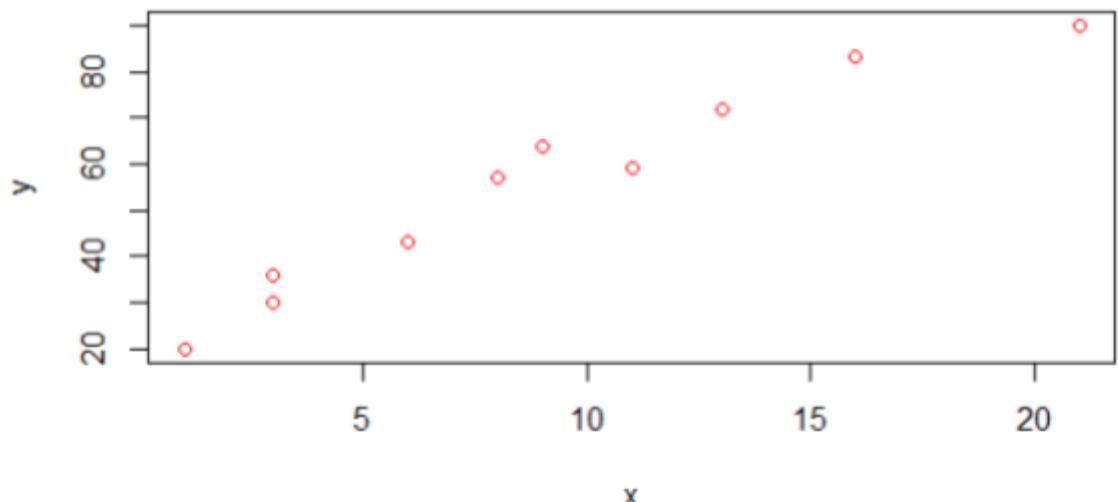
Aim: Implementation and analysis of Linear regression through graphical methods.

```
✓ x<-c(3,8,9,13,3,6,11,21,1,16)
  #response variable
  y<-c(30,57,64,72,36,43,59,90,20,83)
  plot(x,y)
```



```
✓ plot(x,y, col='red',main="scatter plot")
```

scatter plot



□ Linear Regression Model

✓ `model=lm(y~x)`

```
> model=lm(y~x)
> model

call:
lm(formula = y ~ x)

Coefficients:
(Intercept)           x
23.209          3.537
```

model

✓ `attributes(model)`

`coef(model)`

`residuals(model)`

```
> attributes(model)
$names
[1] "coefficients"   "residuals"      "effects"       "rank"
[6] "assign"          "qr"            "df.residual"   "xlevels"
[11] "terms"           "model"          "fitted.values"
[16] "call"

$class
[1] "lm"

> coef(model)
(Intercept)           x
23.208972    3.537476
> residuals(model)
     1      2      3      4      5      6      7      8
-3.821239  5.491223  8.953748  2.803845  2.178601 -1.433826 -3.121204 -7.495960
     9     10
-6.746447  3.191418
```

✓ `summary(model)`

```
> summary(model)

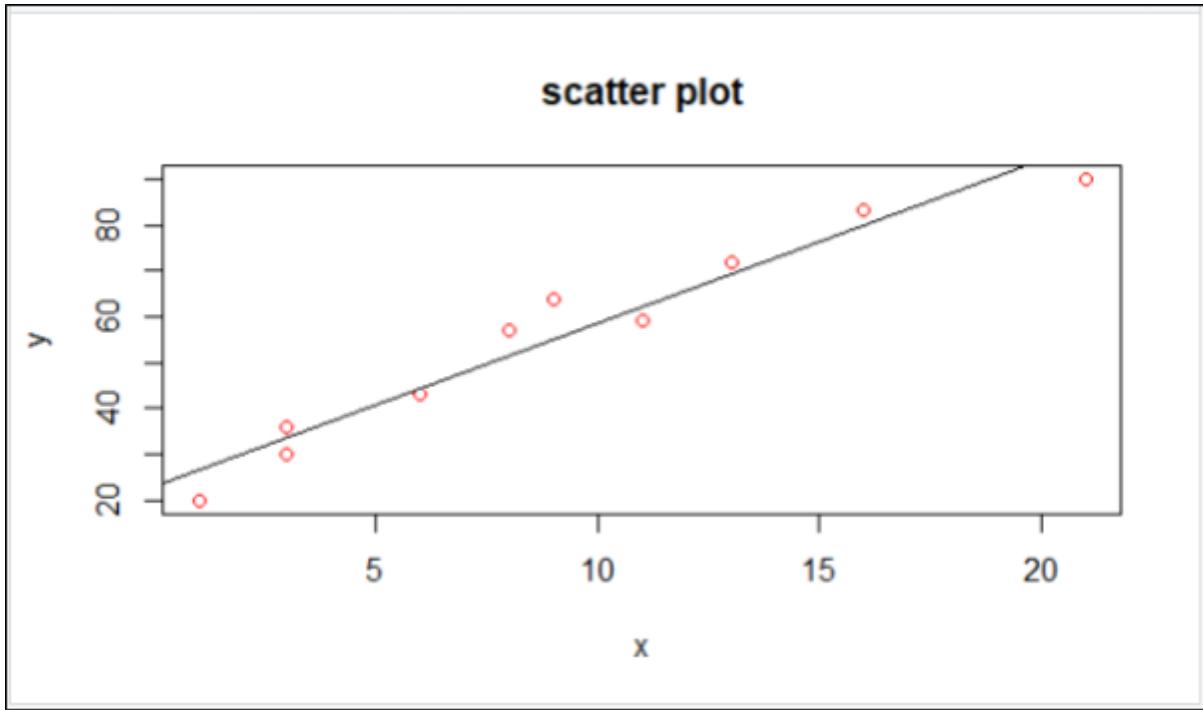
call:
lm(formula = y ~ x)

Residuals:
    Min      1Q      Median      3Q      Max 
-7.4960 -3.6463  0.3724  3.0945  8.9537 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 23.2090    3.2862   7.062 0.000106 ***
x            3.5375    0.3016  11.728 2.55e-06 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.714 on 8 degrees of freedom
Multiple R-squared:  0.945,    Adjusted R-squared:  0.9382 
F-statistic: 137.5 on 1 and 8 DF,  p-value: 2.553e-06
```

□ abline(model)



□ Predicting values manually $y=a+bx$

✓ $x10 <- \text{model\$coefficients[[1]]} + \text{model\$coefficients[[2]]}*10$
x10

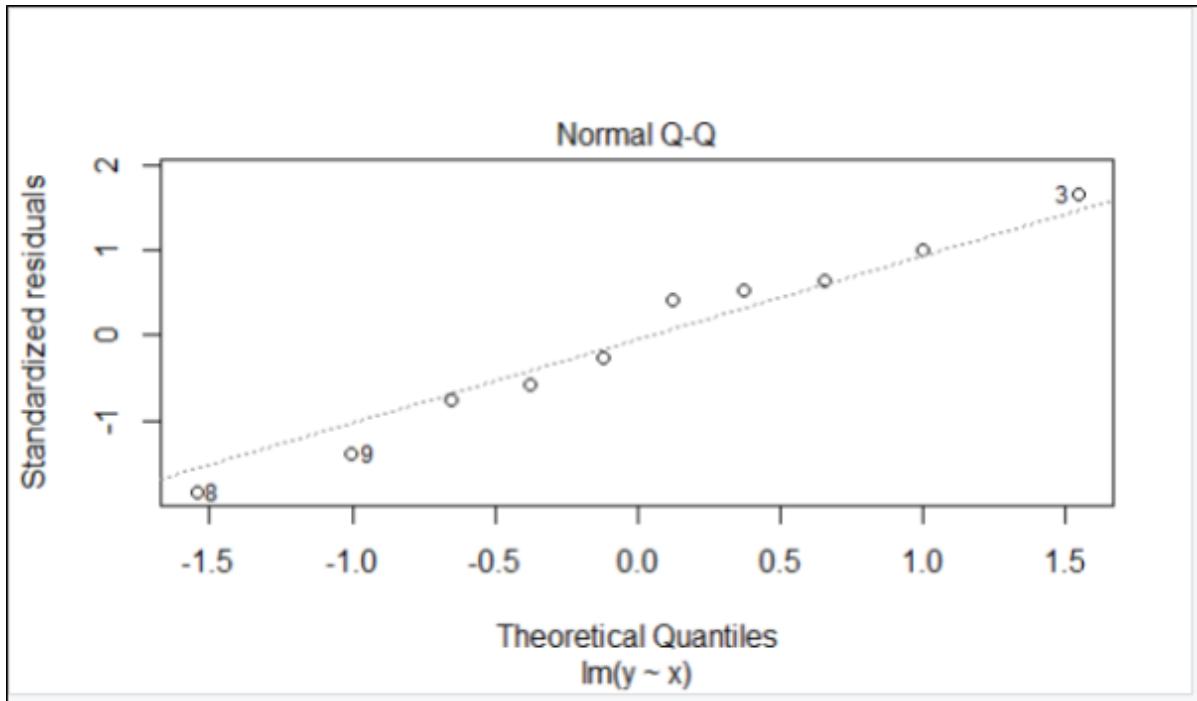
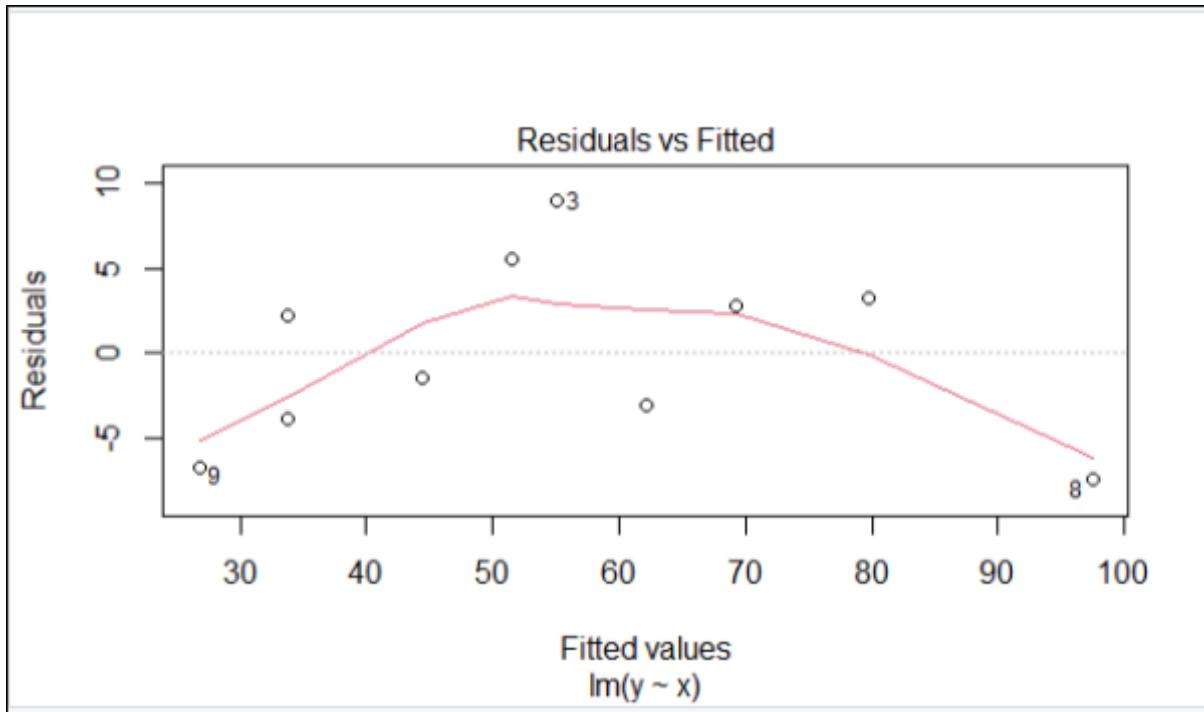
```
> #predicting values manually y=a+bx
> x10<-model$coefficients[[1]]+model$coefficients[[2]]*10
> x10
[1] 58.58373
```

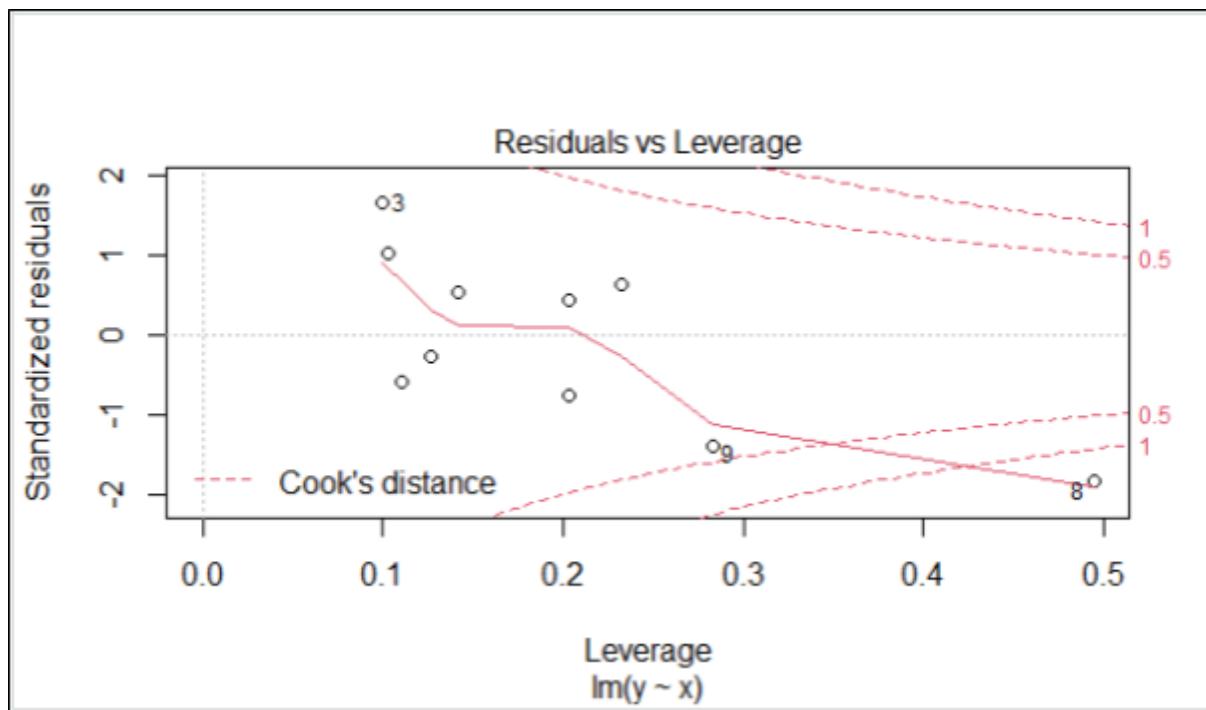
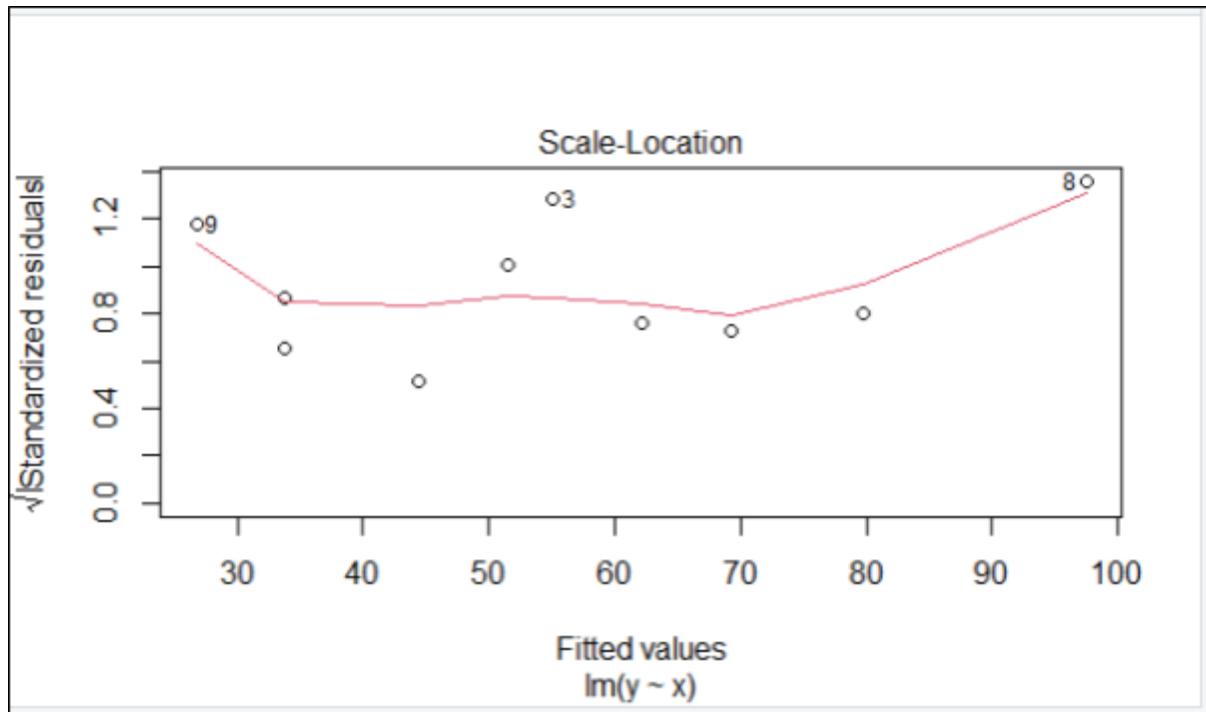
□ Using Predict()

✓ a<-data.frame(x=10)
pred<-predict(model,a) pred

```
> #using Predict()
> a<-data.frame(x=10)
> a
   x
1 10
> pred<-predict(model,a)
> pred
   1
58.58373
```

□ plot(model)





Practical No. 08

Aim: Implementation and analysis of Classification algorithms like Naive Bayesian, K-Nearest Neighbour.

□ **Naive Bayesian**

✓ View(iris)

▲	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa

✓ ir=iris

train=ir[1:100,] train

```
> ir=iris
> train=ir[1:100,]
> train
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          5.1        3.5         1.4        0.2    setosa
2          4.9        3.0         1.4        0.2    setosa
3          4.7        3.2         1.3        0.2    setosa
4          4.6        3.1         1.5        0.2    setosa
5          5.0        3.6         1.4        0.2    setosa
6          5.4        3.9         1.7        0.4    setosa
7          4.6        3.4         1.4        0.3    setosa
8          5.0        3.4         1.5        0.2    setosa
9          4.4        2.9         1.4        0.2    setosa
10         4.9        3.1         1.5        0.1    setosa
11         5.4        3.7         1.5        0.2    setosa
12         4.8        3.4         1.6        0.2    setosa
```

✓ test=ir[101:150,]

test

```
> test=ir[101:150,]
> test
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
101         6.3        3.3         6.0        2.5 virginica
102         5.8        2.7         5.1        1.9 virginica
103         7.1        3.0         5.9        2.1 virginica
104         6.3        2.9         5.6        1.8 virginica
105         6.5        3.0         5.8        2.2 virginica
106         7.6        3.0         6.6        2.1 virginica
107         4.9        2.5         4.5        1.7 virginica
108         7.3        2.9         6.3        1.8 virginica
109         6.7        2.5         5.8        1.8 virginica
110         7.2        3.6         6.1        2.5 virginica
111         6.5        3.2         5.1        2.0 virginica
112         6.4        2.7         5.3        1.9 virginica
113         6.8        3.0         5.5        2.1 virginica
114         5.7        2.5         5.0        2.0 virginica
115         5.8        2.8         5.1        2.4 virginica
```

✓ model=naiveBayes(Species ~.,data=train)

```
> model=naiveBayes(Species ~.,data=train)
> model

Naive Bayes classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
  setosa versicolor virginica
  0.5      0.5      0.0

Conditional probabilities:
  Sepal.Length
Y      [,1]      [,2]
setosa 5.006 0.3524897
versicolor 5.936 0.5161711
virginica NA       NA

  Sepal.Width
Y      [,1]      [,2]
setosa 3.428 0.3790644
versicolor 2.770 0.3137983
virginica NA       NA
```

model

✓ test\$Species

```
> test$Species
[1] virginica virginica virginica virginica virginica virginica virginica
[9] virginica virginica virginica virginica virginica virginica virginica virginica
[17] virginica virginica virginica virginica virginica virginica virginica virginica
[25] virginica virginica virginica virginica virginica virginica virginica virginica
[33] virginica virginica virginica virginica virginica virginica virginica virginica
[41] virginica virginica virginica virginica virginica virginica virginica virginica
[49] virginica virginica
Levels: setosa versicolor virginica
```

✓ train\$Species

```
> train$Species
[1] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[8] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[15] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[22] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[29] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[36] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[43] setosa    setosa    setosa    setosa    setosa    setosa    setosa
[50] setosa    versicolor versicolor versicolor versicolor versicolor
[57] versicolor versicolor versicolor versicolor versicolor versicolor
[64] versicolor versicolor versicolor versicolor versicolor versicolor
[71] versicolor versicolor versicolor versicolor versicolor versicolor
[78] versicolor versicolor versicolor versicolor versicolor versicolor
[85] versicolor versicolor versicolor versicolor versicolor versicolor
[92] versicolor versicolor versicolor versicolor versicolor versicolor
[99] versicolor versicolor
Levels: setosa versicolor virginica
```

✓ pred=predict(model,test)

```
table(pred)
table(test$Species)
table(train$Species)
```

```
> pred=predict(model,test)
> table(pred)
pred
setosa versicolor virginica
  0      50      0
> table(test$Species)

setosa versicolor virginica
  0      0      50
> table(train$Species)

setosa versicolor virginica
 50      50      0
```

□ Shuffle Iris File

✓ ir1=ir[sample(nrow(ir)),]

View(ir1)

▲	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
61	5.0	2.0	3.5	1.0	versicolor
118	7.7	3.8	6.7	2.2	virginica
33	5.2	4.1	1.5	0.1	setosa
34	5.5	4.2	1.4	0.2	setosa
96	5.7	3.0	4.2	1.2	versicolor
54	5.5	2.3	4.0	1.3	versicolor
145	6.7	3.3	5.7	2.5	virginica
51	7.0	3.2	4.7	1.4	versicolor
28	5.2	3.5	1.5	0.2	setosa
43	4.4	3.2	1.3	0.2	setosa
74	6.1	2.8	4.7	1.2	versicolor
132	7.9	3.8	6.4	2.0	virginica
52	6.4	3.2	4.5	1.5	versicolor
13	4.6	3.0	1.4	0.1	setosa
39	4.4	3.0	1.3	0.2	setosa
64	6.1	2.9	4.7	1.4	versicolor
104	6.3	2.9	5.6	1.8	virginica

✓ train=ir1[1:100,]

```
test=ir1[101:150,]
model=model=naiveBayes(Species ~.,data=train)
pred=predict(model,test)
table(pred)
```

```
> train=ir1[1:100,]
> test=ir1[101:150,]
> model=model=naiveBayes(Species ~.,data=train)
> pred=predict(model,test)
> table(pred)
pred
  setosa versicolor virginica
    18       16        16
```

✓ table(train\$Species)

```
table(test$Species)
```

```
> table(train$Species)
  setosa versicolor virginica
    32       34        34
> table(test$Species)
  setosa versicolor virginica
    18       16        16
```

□ K-Nearest Neighbour

✓ `table(iris$Species)`

`str(iris$Species)`

`head(iris)`

```
> table(iris$Species)
  setosa versicolor virginica
    50      50      50
> str(iris$Species)
Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 ...
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

✓ `ir1=ir[sample(nrow(ir)),]`

`head(ir1)`

```
> #Shuffle data
> ir1=ir[sample(nrow(ir)),]
> #check Shuffling
> head(ir1)
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
72          6.1         2.8          4.0         1.3 versicolor
40          5.1         3.4          1.5         0.2   setosa
75          6.4         2.9          4.3         1.3 versicolor
32          5.4         3.4          1.5         0.4   setosa
141         6.7         3.1          5.6         2.4 virginica
122         5.6         2.8          4.9         2.0 virginica
```

✓ `normalize<-function(x){`

```
  return((x-min(x)/(max(x)-min(x))))
}
```

```
iris_n<-as.data.frame(lapply(ir1[,c(1,2,3,4)],normalize))
str(iris_n)
```

```
> normalize<-function(x){
+   return((x-min(x)/(max(x)-min(x))))
+ }
> #normalize iris data
> iris_n<-as.data.frame(lapply(ir1[,c(1,2,3,4)],normalize))
> str(iris_n)
'data.frame': 150 obs. of 4 variables:
 $ Sepal.Length: num  4.91 3.91 5.21 4.21 5.51 ...
 $ Sepal.Width : num  1.97 2.57 2.07 2.57 2.27 ...
 $ Petal.Length: num  3.83 1.33 4.13 1.33 5.43 ...
 $ Petal.Width : num  1.258 0.158 1.258 0.358 2.358 ...
```

✓ iris_train<-iris_n[1:129,]
 iris_test<-iris_n[130:150,]
 iris_train_target<-iris[1:129,5]
 iris_test_target<-iris[130:150,5]
 iris_train_target

```
> iris_train<-iris_n[1:129,]
> iris_test<-iris_n[130:150,]
> iris_train_target<-iris[1:129,5]
> iris_test_target<-iris[130:150,5]
> iris_train_target
 [1] setosa   setosa   setosa   setosa   setosa   setosa   setosa
 [8] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[15] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[22] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[29] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[36] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[43] setosa   setosa   setosa   setosa   setosa   setosa   setosa
[50] setosa   versicolor versicolor versicolor versicolor versicolor
[57] versicolor versicolor versicolor versicolor versicolor versicolor
[64] versicolor versicolor versicolor versicolor versicolor versicolor
[71] versicolor versicolor versicolor versicolor versicolor versicolor
[78] versicolor versicolor versicolor versicolor versicolor versicolor
[85] versicolor versicolor versicolor versicolor versicolor versicolor
[92] versicolor versicolor versicolor versicolor versicolor versicolor
[99] versicolor versicolor virginica  virginica  virginica  virginica
[106] virginica virginica virginica  virginica  virginica  virginica
[113] virginica virginica virginica  virginica  virginica  virginica
[120] virginica virginica virginica  virginica  virginica  virginica
[127] virginica virginica virginica
Levels: setosa versicolor virginica
```

✓ dim(iris_train)
 dim(iris_test)
 model<-knn(iris_train,iris_test,cl=iris_train_target,k=13) model
 table(iris_test_target,model)

```
> dim(iris_train)
[1] 129  4
> dim(iris_test)
[1] 21  4
> model<-knn(iris_train,iris_test,cl=iris_train_target,k=13)
> model
 [1] versicolor versicolor setosa      setosa      setosa      setosa      versicolor
 [8] virginica   setosa      setosa      setosa      setosa      versicolor versicolor
[15] versicolor  setosa      versicolor setosa      versicolor versicolor setosa
Levels: setosa versicolor virginica
> table(iris_test_target,model)
            model
iris_test_target setosa versicolor virginica
                0       0       0
setosa          0       0       0
versicolor      11      9       1
virginica        1
```

Practical No. 09

Aim: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.

✓ mba_data<-read.csv("data_apriori.csv")

View(mba_data)

Customer_Id	Products
1	bread
2	butter
3	eggs
4	milk
5	bread
6	butter
7	eggs
8	milk

□ class(mba_data)

```
> class(mba_data)
[1] "data.frame"
```

□ **Method 1: Convert Data Frame into Transactions**

✓ t<-as(mba_data,"transactions")

```
class(t)
inspect(t)
```

```
> class(t)
[1] "transactions"
attr(,"package")
[1] "arules"
> inspect(t)
  items                               transactionID
[1] {Customer_Id=[1,6],Products=bread}      1
[2] {Customer_Id=[1,6],Products=butter}     2
[3] {Customer_Id=[1,6],Products=eggs}       3
[4] {Customer_Id=[1,6],Products=milk}        4
[5] {Customer_Id=[1,6],Products=bread}      5
[6] {Customer_Id=[1,6],Products=butter}     6
[7] {Customer_Id=[1,6],Products=eggs}        7
[8] {Customer_Id=[1,6],Products=milk}        8
[9] {Customer_Id=[1,6],Products=soda}       9
[10] {Customer_Id=[1,6],Products=beer}      10
[11] {Customer_Id=[1,6],Products=bread}     11
[12] {Customer_Id=[1,6],Products=cheese}    12
[13] {Customer_Id=[1,6],Products=chips}     13
```

□ Method 2: Convert Data Frame into Transactions

- ✓ `trans<-split(mba_data$Customer_Id, mba_data$Products, "transactions")`
trans

```
> trans<-split(mba_data$Customer_Id, mba_data$Products, "transactions")
> trans
$banana
[1] 7 11 13

$beer
[1] 2 5 8 13 15

$bread
[1] 1 4 2 3 6 9 8 10 12 13 14 15

$buns
[1] 5 8

$butter
[1] 1 4 3 6 9 10 12 13 14

$cheese
[1] 2 8 13 15

$chips
[1] 2 5 8 11 15

$chocolate
[1] 6 7 8 10
```

- ✓ `class(trans)`

```
> class(trans)
[1] "list"
```

- ✓ `trans<-split(mba_data$Products, mba_data$Customer_Id, "transactions")`
`head(trans)`

```
> trans<-split(mba_data$Products, mba_data$Customer_Id, "transactions")
> head(trans)
$`1`
[1] "bread" "butter" "eggs"   "milk"

$`2`
[1] "beer"   "bread"  "cheese" "chips"  "mayo"   "soda"

$`3`
[1] "bread"  "butter" "eggs"   "milk"   "oranges"

$`4`
[1] "bread"  "butter" "eggs"   "milk"   "soda"

$`5`
[1] "buns"   "chips"  "beer"   "mustard" "pickels" "soda"

$`6`
[1] "bread"  "butter" "chocolate" "eggs"   "milk"
```

✓ rules<-apriori(trans,parameter =
list(support=0.5,confidence=0.9,maxlen=3,minlen=2))

```
> rules<-apriori(trans,parameter = list(support=0.5,confidence=0.9,maxlen=3,minlen=2))
Apriori

Parameter specification:
confidence minval smax arem aval originalSupport maxtime support minlen maxlen
0.9      0.1    1 none FALSE           TRUE      5     0.5     2     3
target   ext
rules    TRUE

Algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE    2    TRUE

Absolute minimum support count: 7

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[15 item(s), 15 transaction(s)] done [0.00s].
sorting and recoding items ... [4 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [11 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

✓ rules

```
> rules
set of 11 rules
```

✓ inspect(head(rules,n=3, by="lift"))
summary(rules)

```
> inspect(head(rules,n=3, by="lift"))
   lhs          rhs    support  confidence coverage lift    count
[1] {eggs}      => {milk} 0.6000000 1        0.6000000 1.666667 9
[2] {milk}      => {eggs} 0.6000000 1        0.6000000 1.666667 9
[3] {butter,eggs} => {milk} 0.5333333 1        0.5333333 1.666667 8
> summary(rules)
set of 11 rules

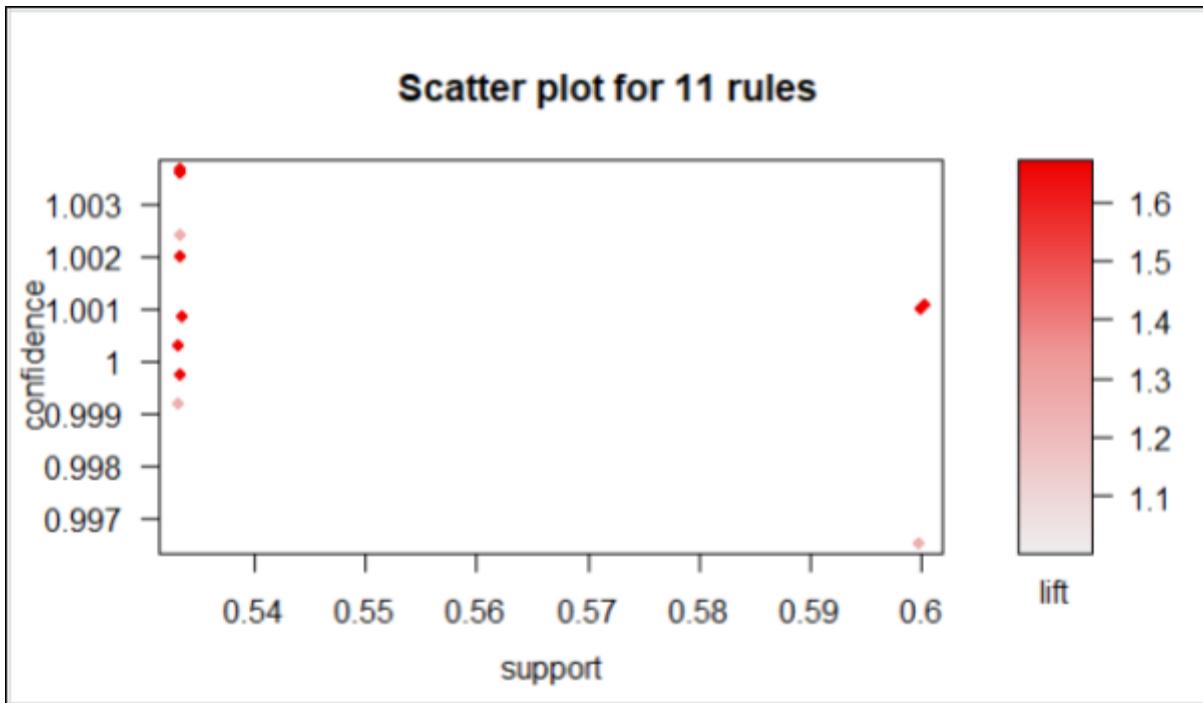
rule length distribution (lhs + rhs):sizes
2 3
3 8

   Min. 1st Qu. Median Mean 3rd Qu. Max.
2.000 2.500 3.000 2.727 3.000 3.000

summary of quality measures:
   support    confidence    coverage    lift    count
Min. :0.5333  Min. :1       Min. :0.5333  Min. :1.250  Min. :8.000
1st Qu.:0.5333 1st Qu.:1       1st Qu.:0.5333 1st Qu.:1.458  1st Qu.:8.000
Median :0.5333  Median :1       Median :0.5333  Median :1.667  Median :8.000
Mean   :0.5515  Mean   :1       Mean   :0.5515  Mean   :1.553  Mean   :8.273
3rd Qu.:0.5667 3rd Qu.:1       3rd Qu.:0.5667 3rd Qu.:1.667  3rd Qu.:8.500
Max.   :0.6000  Max.   :1       Max.   :0.6000  Max.   :1.667  Max.   :9.000

mining info:
  data ntransactions support confidence
trans      15            0.5        0.9
```

✓ plot(rules)



✓ data("Groceries")
inspect(head(Groceries,3))

```
> data("Groceries")
> inspect(head(Groceries,3))
  items
[1] {citrus fruit,semi-finished bread,margarine,ready soups}
[2] {tropical fruit,yogurt,coffee}
[3] {whole milk}
```

✓ View(Groceries)

● Groceries	S4 [9835 x 169] (arules::transacti	S4 object of class transactions
● data	S4 [169 x 9835] (Matrix::ngCMatrix	S4 object of class ngCMatrix
● itemInfo	list [169 x 3] (S3: data.frame)	A data.frame with 169 rows and 3 columns
itemsetInfo	list [0 x 0] (S3: data.frame)	A data.frame with 0 rows and 0 columns

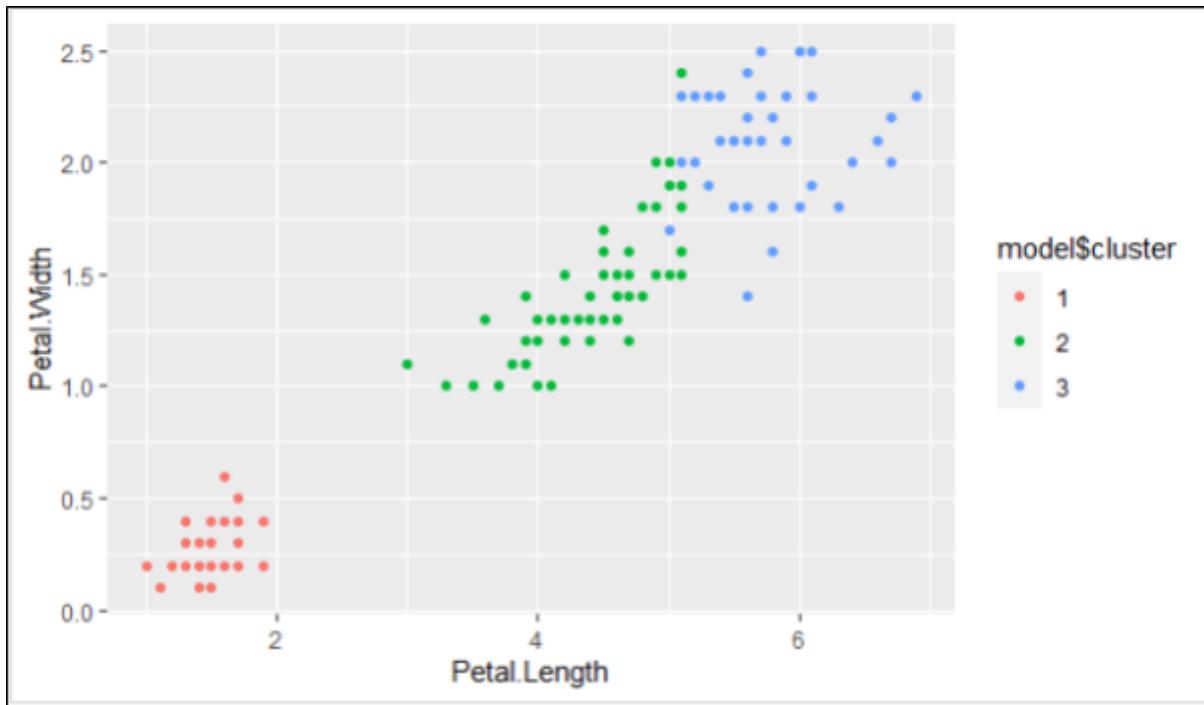
✓ rules<-apriori(Groceries,parameter =
list(support=0.01,confidence=0.09,maxlen=3,minlen=2))

```
> rules<-apriori(Groceries,parameter = list(support=0.01,confidence=0.09,maxlen=3,minle  
n=2))  
Apriori  
  
Parameter specification:  
confidence minval smax arem  aval originalSupport maxtime support minlen maxlen  
0.09      0.1    1 none FALSE           TRUE      5     0.01      2      3  
target  ext  
rules TRUE  
  
Algorithmic control:  
filter tree heap memopt load sort verbose  
0.1 TRUE TRUE FALSE TRUE      2      TRUE  
  
Absolute minimum support count: 98  
  
set item appearances ...[0 item(s)] done [0.00s].  
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
sorting and recoding items ... [88 item(s)] done [0.00s].  
creating transaction tree ... done [0.00s].  
checking subsets of size 1 2 3 done [0.00s].  
writing ... [445 rule(s)] done [0.00s].  
creating 54 object ... done [0.00s].
```

✓ rules

```
|> rules  
set of 445 rules
```


✓ model\$cluster<-as.factor(model\$cluster)
ggplot(iris,aes(Petal.Length,Petal.Width,color=model\$cluster))+geom_point()



□ Agglomerative

✓ df<-USArrests

View(df)

	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7
Connecticut	3.3	110	77	11.1
Delaware	5.9	238	72	15.8
Florida	15.4	335	80	31.9
Georgia	17.4	211	60	25.8
Hawaii	5.3	46	83	20.2
Idaho	2.6	120	54	14.2

□ Pre-processing

✓ #Remove NA Values

```
df<-na.omit(df)
```

```
df
```

```
> df<-na.omit(df)
> df
      Murder Assault UrbanPop Rape
Alabama    13.2     236      58 21.2
Alaska     10.0     263      48 44.5
Arizona      8.1     294      80 31.0
Arkansas     8.8     190      50 19.5
california   9.0     276      91 40.6
Colorado     7.9     204      78 38.7
Connecticut   3.3     110      77 11.1
Delaware      5.9     238      72 15.8
Florida      15.4     335      80 31.9
Georgia      17.4     211      60 25.8
Hawaii        5.3      46      83 20.2
Idaho         2.6     120      54 14.2
Illinois     10.4     249      83 24.0
Indiana       7.2     113      65 21.0
Iowa          2.2      56      57 11.3
Kansas         6.0     115      66 18.0
Kentucky       9.7     109      52 16.3
Louisiana     15.4     249      66 22.2
Maine          2.1      83      51  7.8
Maryland       11.3     300      67 27.8
Massachusetts  4.4     149      85 16.3
Michigan       12.1     255      74 35.1
Minnesota      2.7      72      66 14.9
Mississippi    16.1     259      44 17.1
Missouri        9.0     178      70 28.2
Montana         6.0     109      53 16.4
Nebraska        4.3     102      62 16.5
Nevada          12.2     252      81 46.0
New Hampshire   2.1      57      56  9.5
New Jersey      7.4     159      89 18.8
```

✓ #scale (normalizing or Standadizing)

```
d<-scale(df)
```

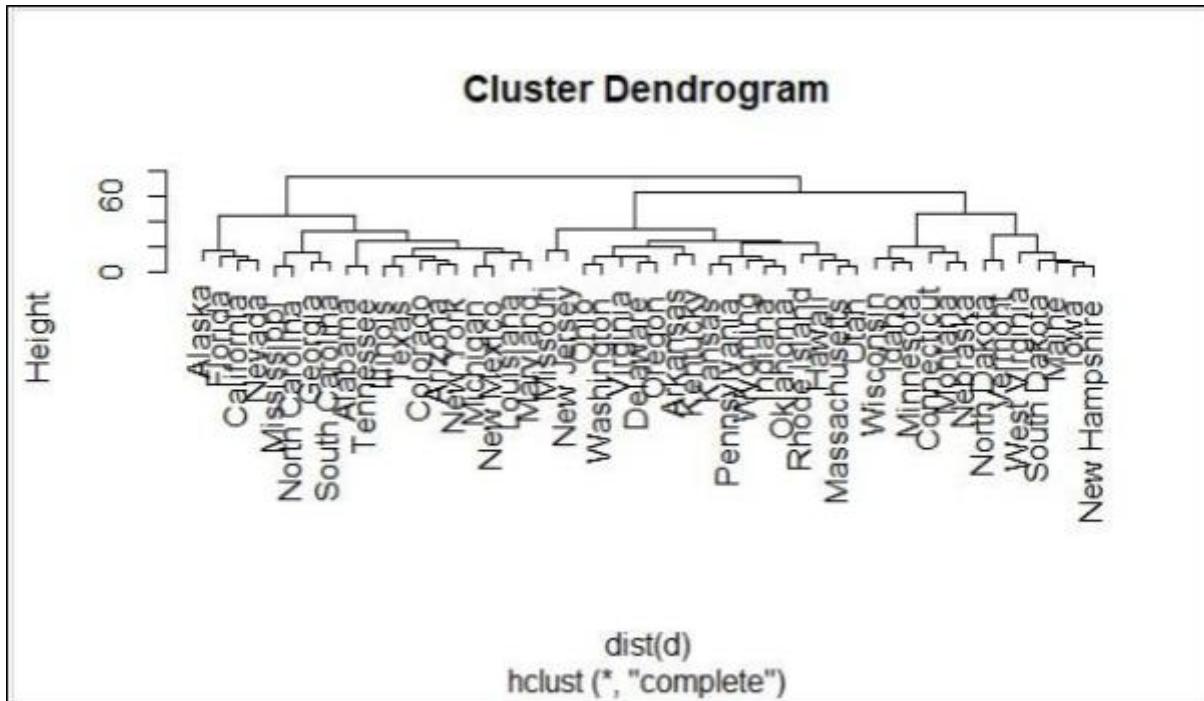
```
head(d)
```

```
> #scale (normalizing or Standadizing)
> d<-scale(df)
> head(d)
      Murder Assault UrbanPop Rape
Alabama  1.24256408  0.7828393 -0.5209066 -0.003416473
Alaska   0.50786248  1.1068225 -1.2117642  2.484202941
Arizona  0.07163341  1.4788032  0.9989801  1.042878388
Arkansas 0.23234938  0.2308680 -1.0735927 -0.184916602
California 0.27826823  1.2628144  1.7589234  2.067820292
Florida   0.00577166  0.2000000  0.0000000  1.000000000
```

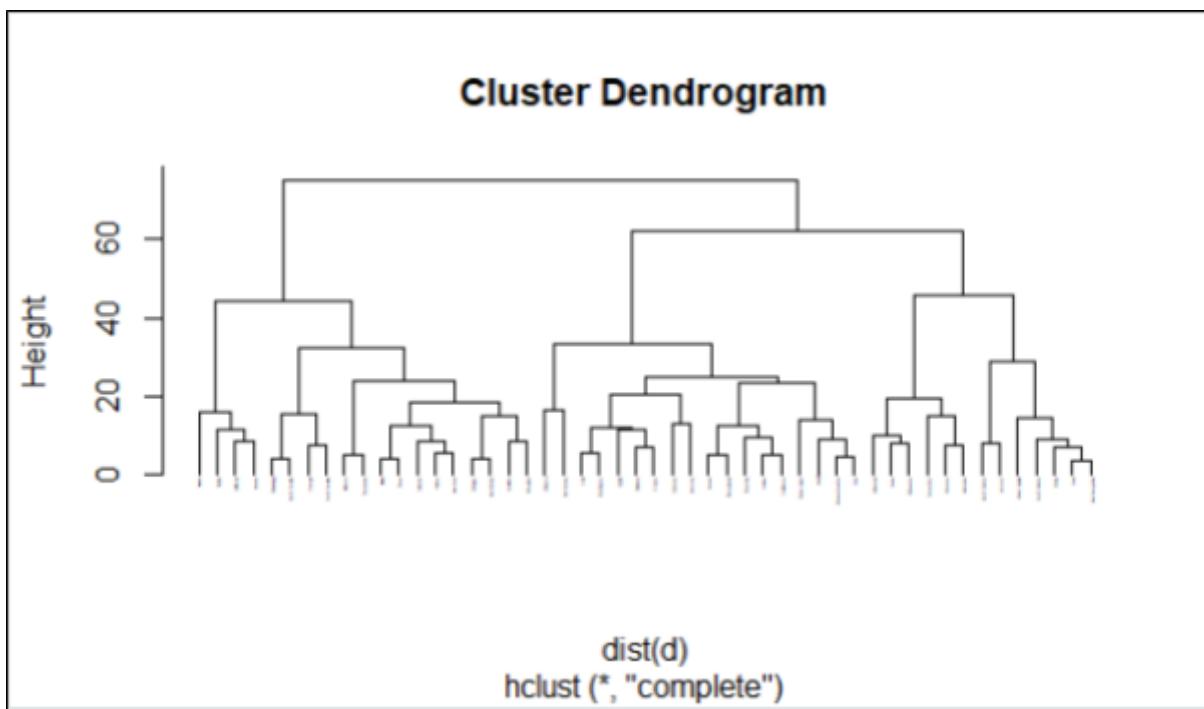
✓ `d<-dist(d,method = "euclidean")`

```
hc<-hclust(d,method = "complete")
```

```
plot(hc)
```

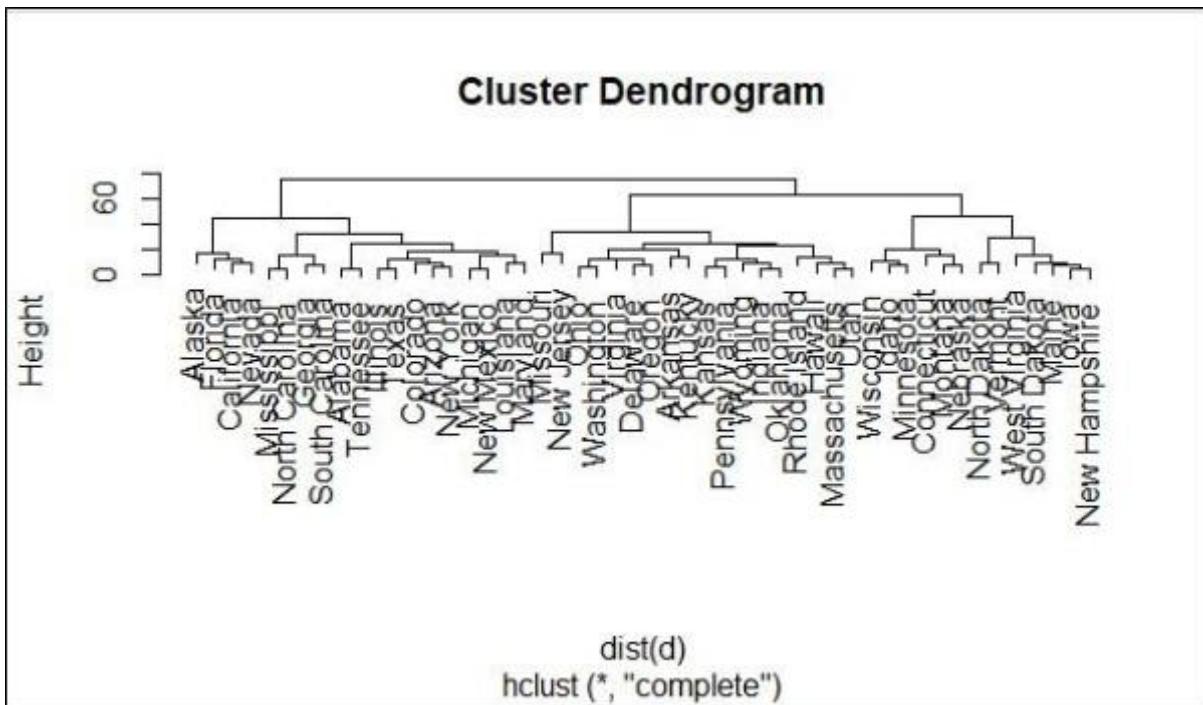


✓ `plot(hc,cex=0.1,hang=-1)`

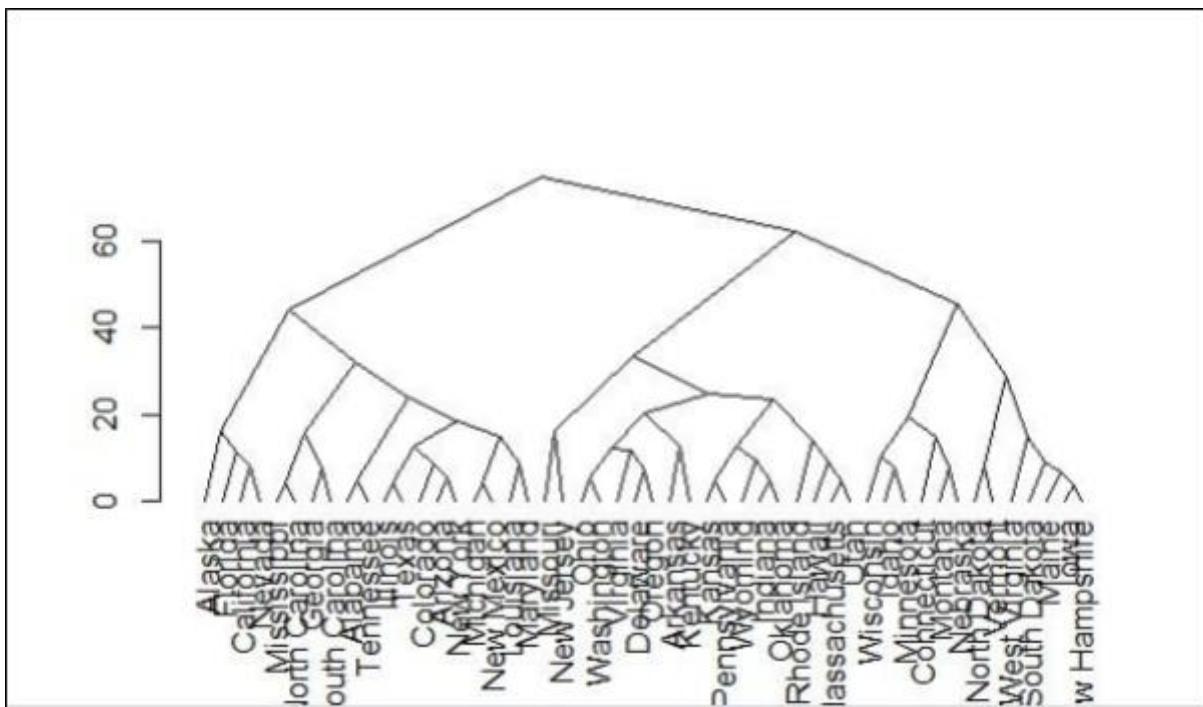


✓ `hc<-hclust(dist(d))`

`plot(hc)`

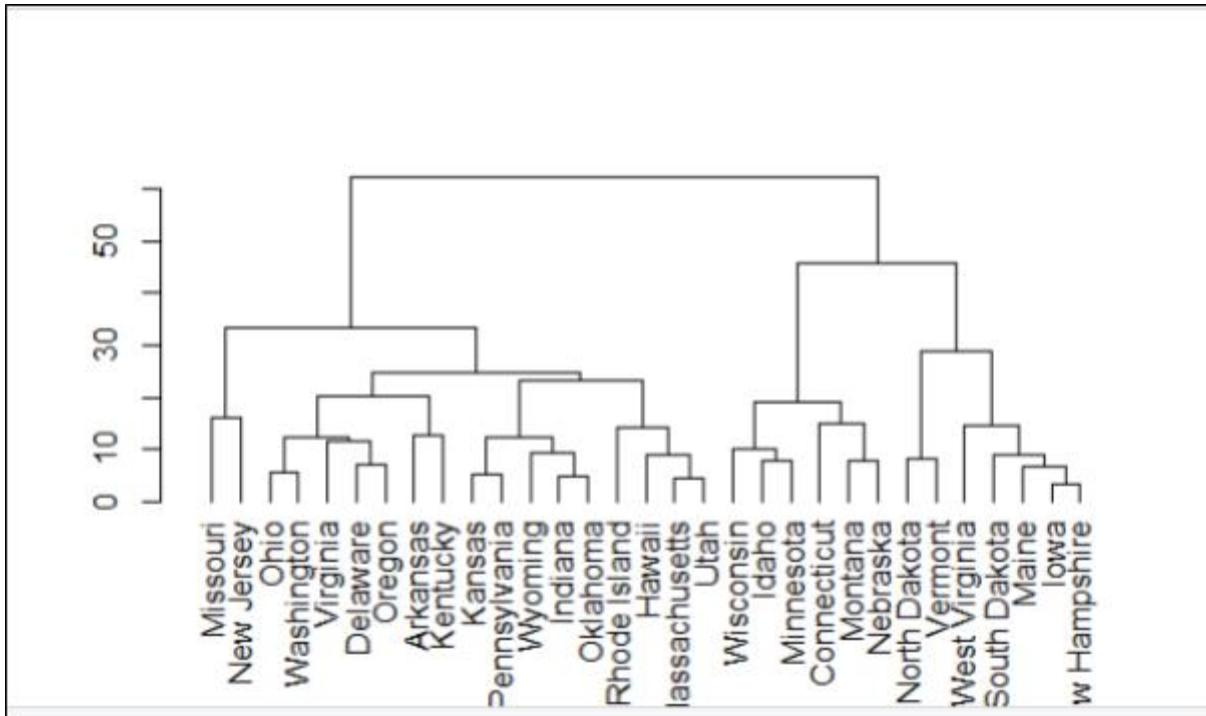


✓ `hcd=as.dendrogram(hc)`



```
plot(hcd,type="triangle")
```

✓ plot(cut(hcd,h=75)\$upper)



✓ plot(cut(hcd,h=75)\$lower[[2]])

