



## **LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION**

ISO 9001-2008 CERTIFIED

S.N. 341, Next to New English School, Govt. Colony,  
Bandra(East), Mumbai – 400051,  
Tel: 91-22-26570892/3181

Date: 16<sup>th</sup> April, 2021

## CERTIFICATE

This is to Certify that Mr./Ms.

LEKHA    MANGESH    NAIK    Roll No.202130

is a student of MCA of 1<sup>st</sup> year Semester-I, has Completed Successfully full-semester Practical/Assignments of subject Advanced Database Management System for the academic year 2020 – 21.

Subject In-Charge

Director

External Examiner

## INDEX

Sr. No.	Title	Remark
1.	Implementation of Data Partitioning through Range and List Partitioning	
2.	Implementation of Analytical Queries like Roll_Up, CUBE, First, Last, Lead, Lag, Row_number, Rank and Dense Rank	
3.	Implementation of ORDBMS using ADT (Abstract Data Type), Reference	
4.	Implementation of ETL Transformation with Pentaho	
5.	Setting working directory, getting current working directory, listing contents of current working directory	
6.	Implementation of Data pre-processing techniques like, <ul style="list-style-type: none"> <li>• Naming and Renaming variables, adding a new variable.</li> <li>• Dealing with missing data.</li> <li>• Dealing with categorical data.</li> <li>• Data reduction using sub-setting.</li> <li>• Naming and Renaming variables, adding a new variable.</li> </ul>	
7.	Implementation and analysis of Linear regression through graphical methods	
8.	Implementation and analysis of Classification algorithms like Naive Bayesian, K-Nearest Neighbor	
9.	Implementation and analysis of Apriori Algorithm using Market Basket Analysis	
10.	Implementation and Analysis of Clustering Algorithms like K-Means, Agglomerative	

## 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')));

#### Output:

```

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
-----      -----
  2015          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';

**Output:**

```
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-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
-----          -----
NUM_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
-----          -----
NUM_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-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA           0 |                |
| SALES                                                                                         | SALES_Q1_2007  |
| TO_DATE(' 2007-03-31 00:00:00', 'SYYYYY-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', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA           0 |                |
| SALES                                                                                         | SALES_Q3_2006  |
| TO_DATE(' 2006-09-30 00:00:00', 'SYYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA             |                |
| TABLE_NAME                                                                                    | PARTITION_NAME |
| HIGH_VALUE                                                                                    |                |
| NUM_ROWS                                                                                      |                |
| SALES                                                                                         | SALES_Q4_2006  |
| TO_DATE(' 2006-12-31 00:00:00', 'SYYYYY-MM-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;

**Output:**

```
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'));

**Output:**

```

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:**

<b>SQL&gt; select * from Student partition(stu_divA);</b>		
<b>STUDENT_ID</b>	<b>STUDENT_NAME</b>	<b>STUDENT_D</b>
221	j	15-SEP-58
321	a	19-AUG-92
421	h	10-JUL-91
<b>SQL&gt; select * from Student partition(stu_divB);</b>		
<b>STUDENT_ID</b>	<b>STUDENT_NAME</b>	<b>STUDENT_D</b>
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_DIUB
'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_DIUB
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_DIVB
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_DIVB
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','Oficer',130245,to\_date('19-4-1978','dd-mm-yyyy'),37520,102);  
insert into Employee values(2893,'Daniel','Oficer',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	Oficer	130245	19-APR-78	37520	102
2893	Daniel	Oficer	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:**

<b>SQL&gt; select DeptNo, sum(Salary)</b>	
<b>2 as Total from Employee</b>	
<b>3 group by DeptNo</b>	
<b>4 order by DeptNo;</b>	
<b>DEPTNO</b>	<b>TOTAL</b>
-----	-----
<b>102</b>	<b>48898</b>
<b>203</b>	<b>79488</b>
<b>304</b>	<b>237589</b>

- ✓ With RollUp:

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

**Output:**

<b>SQL&gt; select DeptNo, sum(Salary)</b>	
<b>2 as Total from Employee</b>	
<b>3 group by rollup(DeptNo)</b>	
<b>4 order by DeptNo;</b>	
<b>DEPTNO</b>	<b>TOTAL</b>
-----	-----
<b>102</b>	<b>48898</b>
<b>203</b>	<b>79488</b>
<b>304</b>	<b>237589</b>
	<b>365959</b>

- ✓ Multiple Column without RollUp:

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

**Output:**

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 RollUp:

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

**Output:**

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
		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:

<b>SQL&gt; select DeptNo, sum(Salary)</b>
<b>2 as Total from Employee</b>
<b>3 group by DeptNo</b>
<b>4 order by DeptNo;</b>
<b>DEPTNO</b>
-----
<b>102</b>
<b>203</b>
<b>304</b>
<b>TOTAL</b>
-----
<b>48890</b>
<b>79480</b>
<b>237589</b>

- ✓ With CUBE:

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

Output:

<b>SQL&gt; select DeptNo, sum(Salary)</b>
<b>2 as Total from Employee</b>
<b>3 group by cube(DeptNo)</b>
<b>4 order by DeptNo;</b>
<b>DEPTNO</b>
-----
<b>102</b>
<b>203</b>
<b>304</b>
<b>TOTAL</b>
-----
<b>48890</b>
<b>79480</b>
<b>237589</b>
<b>365959</b>

✓ Multiple Column without Cube:

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

Output:

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:

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  
from Employee;

**Output:**

```
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;

**Output:**

```
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)  
ROW\_NUM from Employee;

**Output:**

```
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;
```

**Output:**

```
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;
```

**Output:**

```
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;

Output:

```
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;

Output:

```
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;

**Output:**

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

Table created.

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

1 row created.

SQL> desc Person;
Name          Null?    Type
-----        -----   -----
PNAME
```

- 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'));
```

### Output:

```
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;
```

### Output:

```
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()
-----
      9
```

## Practical No. 04

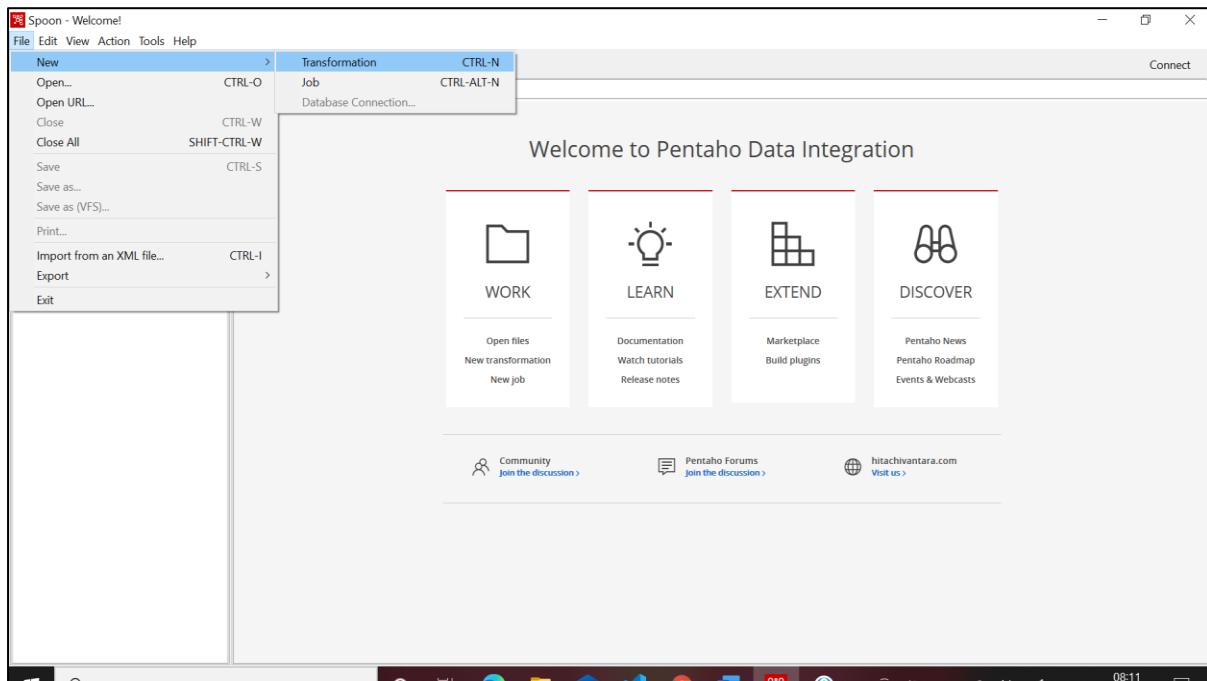
### Aim: Implementation of ETL Transformation with Pentaho.

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

#### Steps:

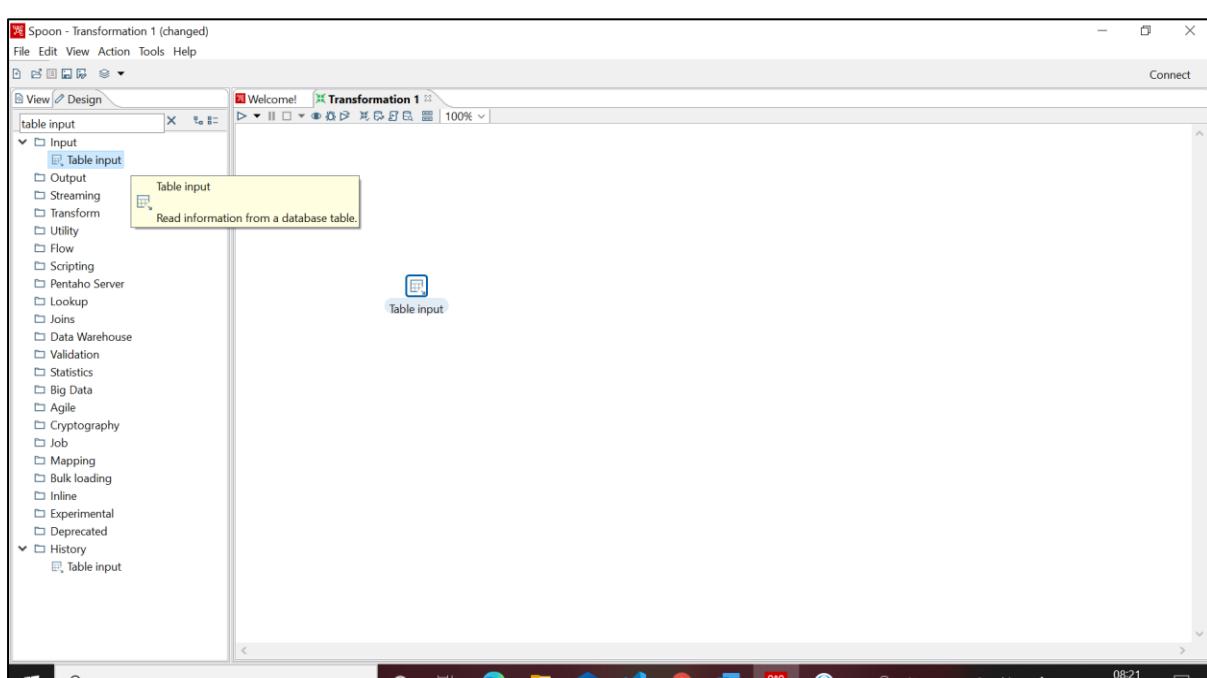
- ✓ Creating a New Transformation:

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



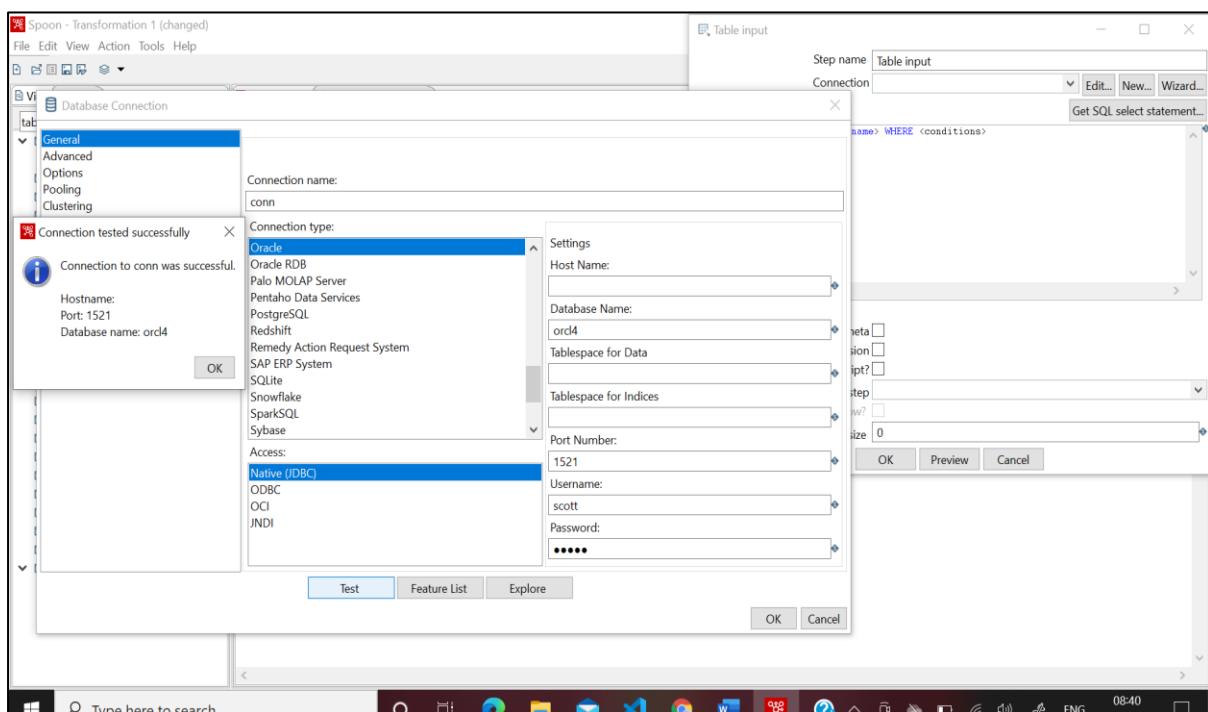
- ✓ Inserting **Table Input** into the Transformation:

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



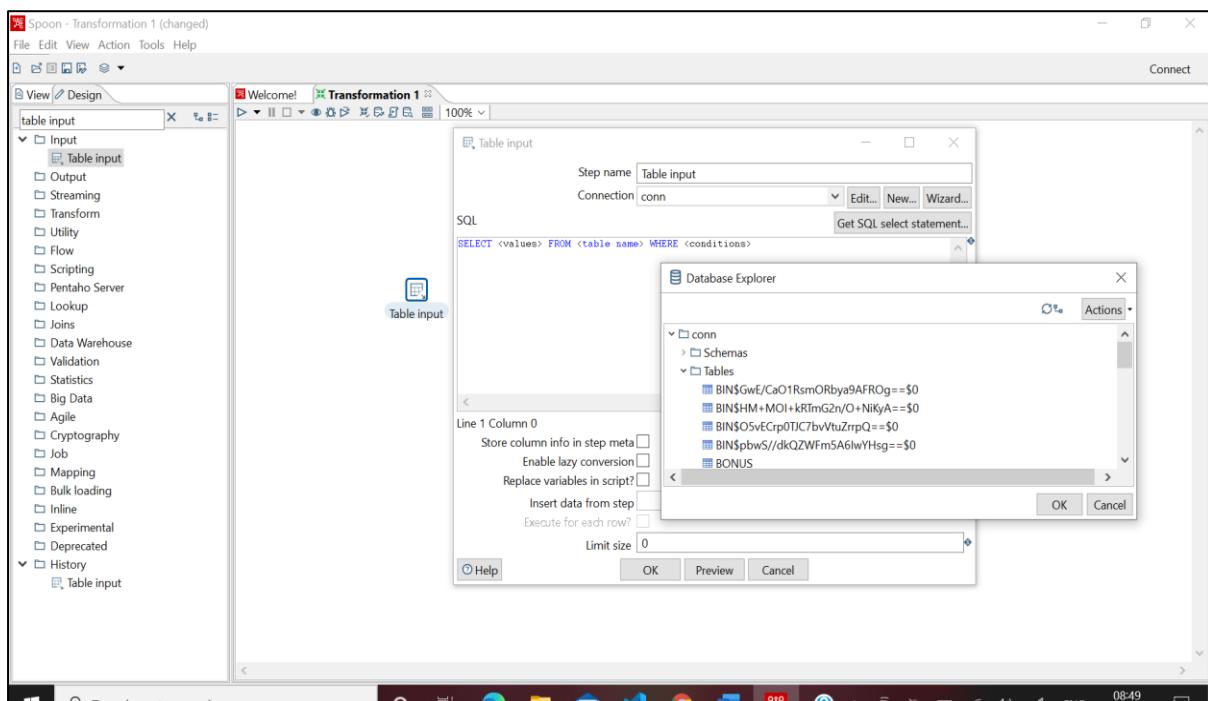
✓ Establishing Database Connection:

Double Click Table Input -> New -> Enter following respective Details in *General*:  
 Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)*  
 Database Name: *ordc4* Port Number: *1521* Username: *scott* Password: *tiger*  
 Click Test -> OK -> OK



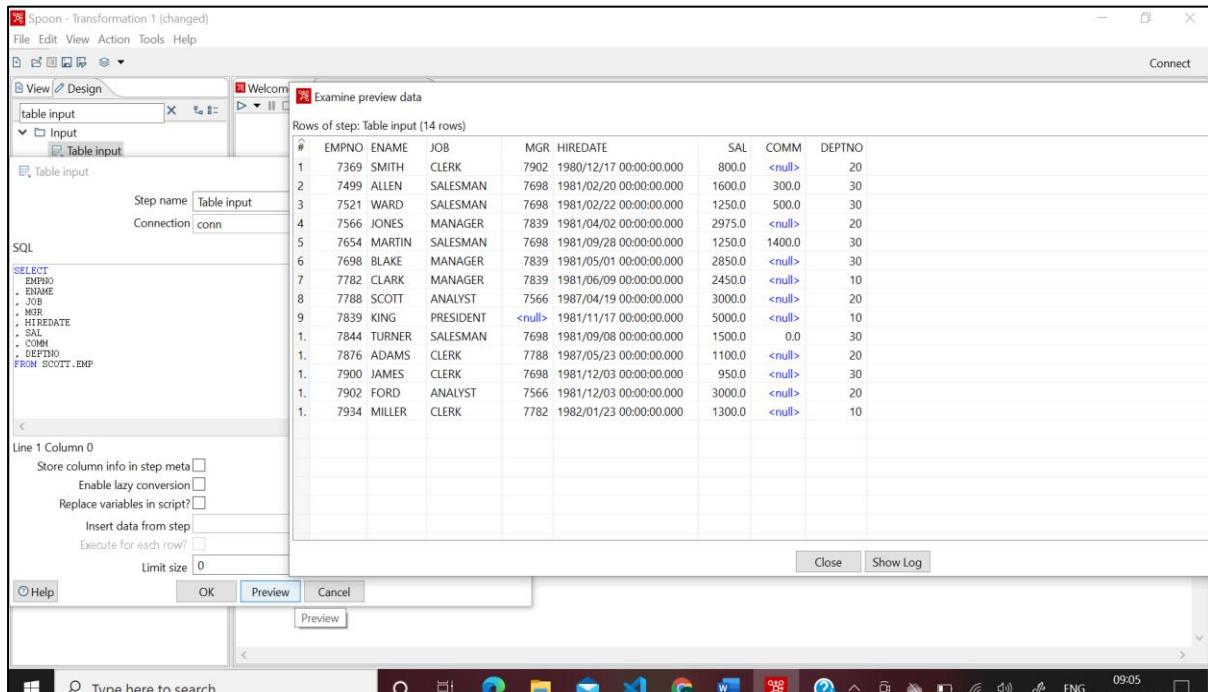
✓ Getting Source Table:

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



✓ View Input Table:

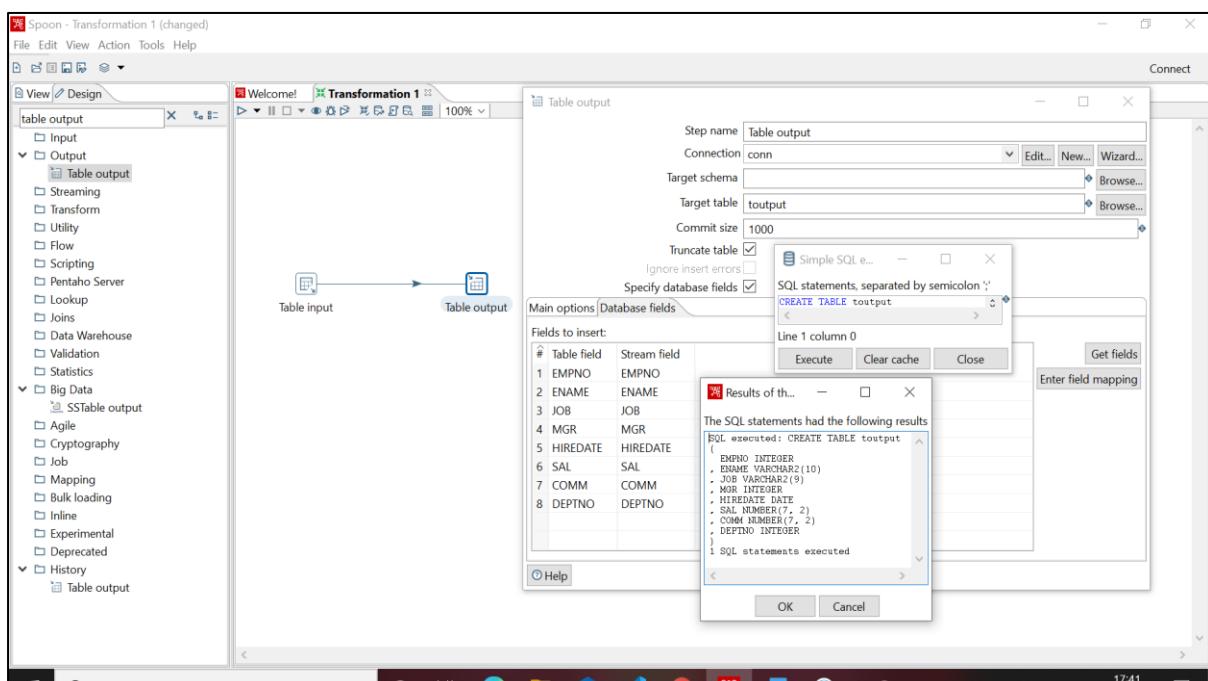
Table Input -> Preview -> OK -> Close -> OK



✓ Storing to Output Table:

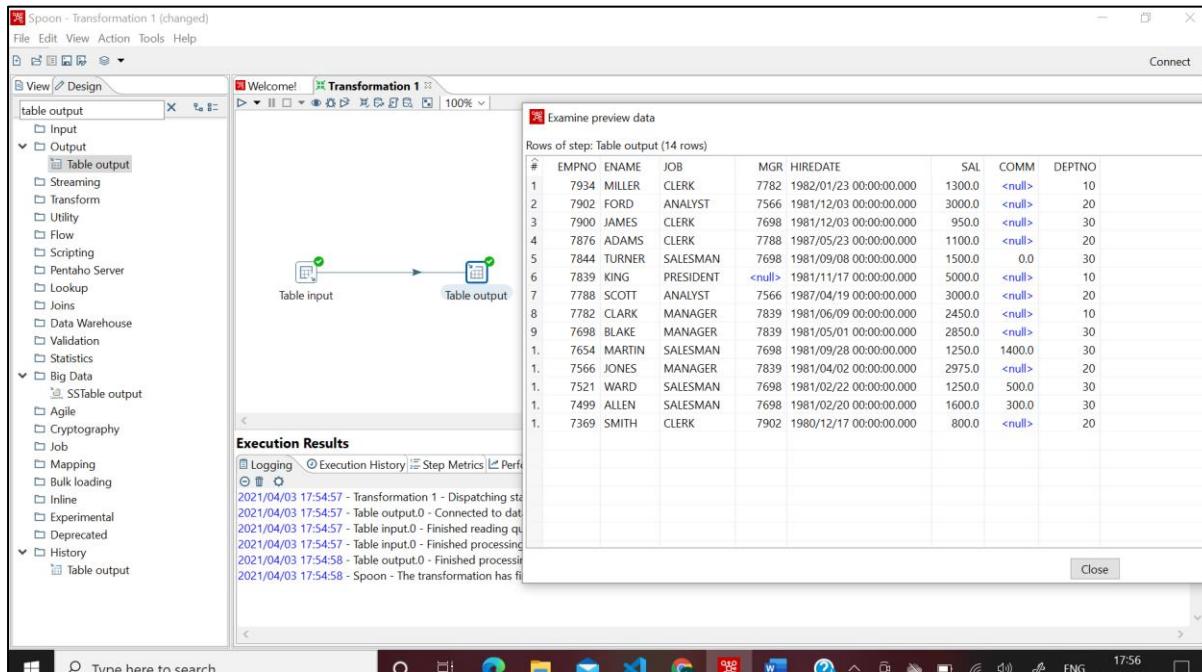
Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Input Table)

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;

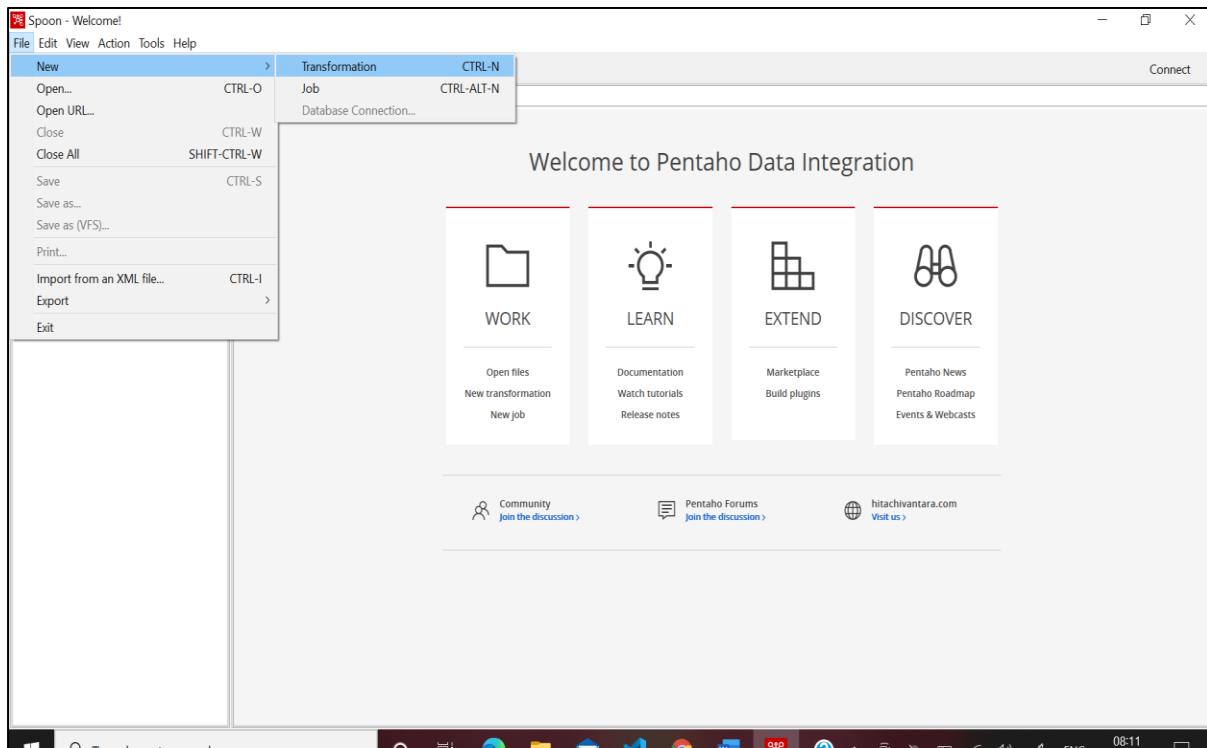
Oracle SQL*Plus						
SQL> select * from toutput;						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
<b>DEPTNO</b>						
7369	SMITH	CLERK	7902	17-DEC-80	800	
20						
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30						
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
<b>DEPTNO</b>						
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
<b>DEPTNO</b>						
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:

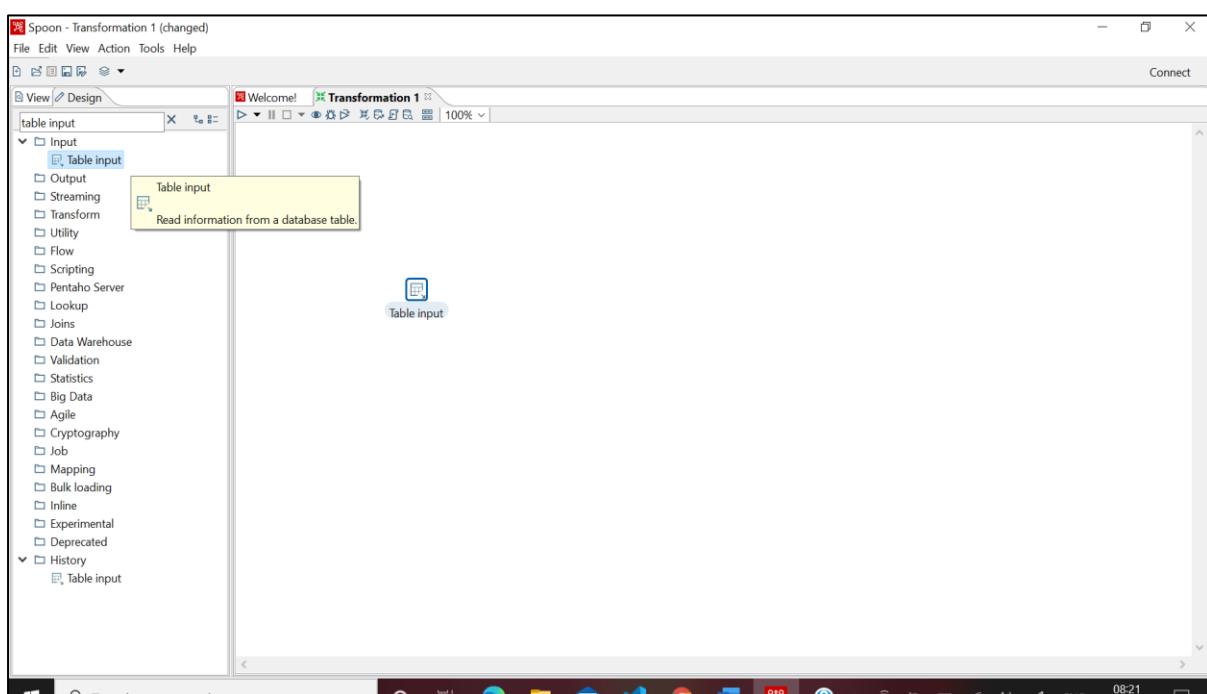
- ✓ Creating a New Transformation:

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



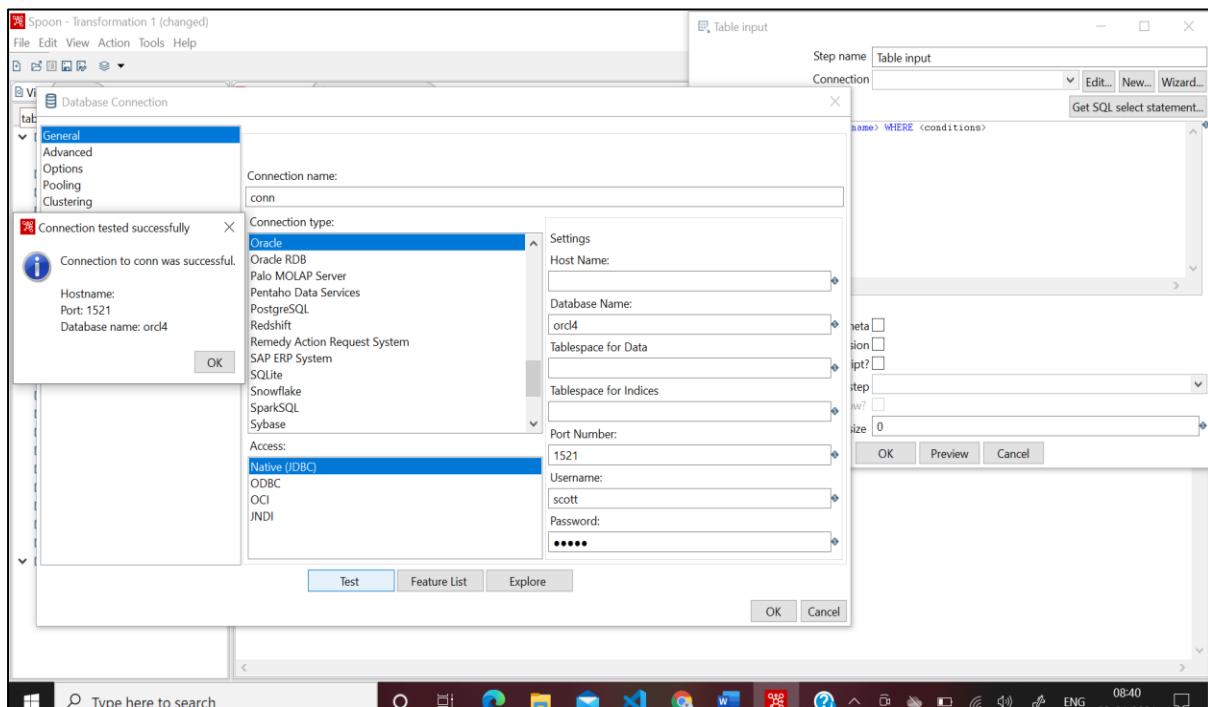
- ✓ Inserting **Table Input** into the Transformation:

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



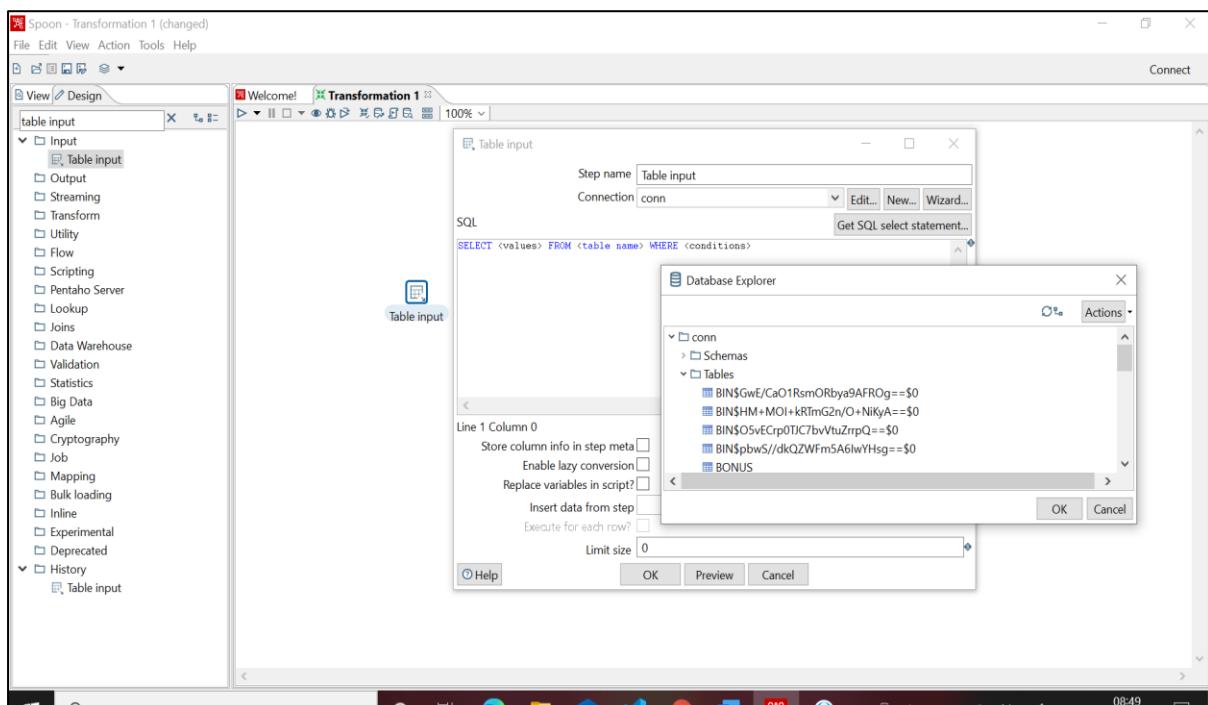
✓ Establishing Database Connection:

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



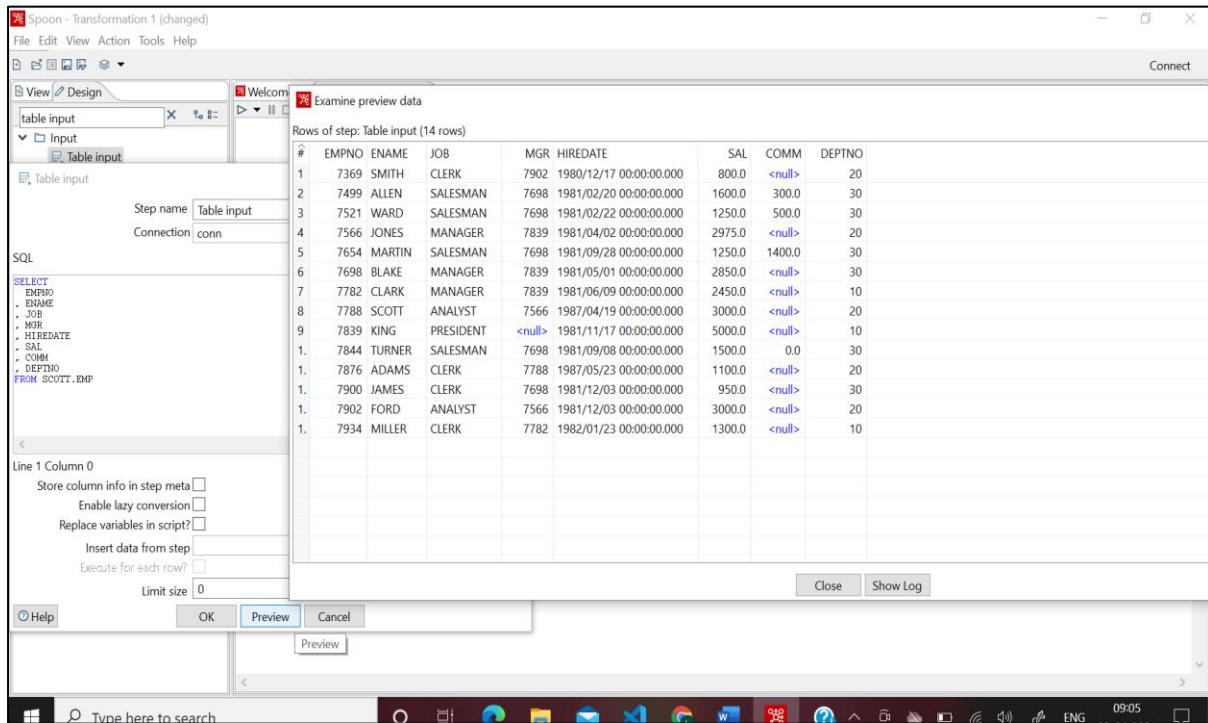
✓ Getting Source Table:

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



✓ View Input Table:

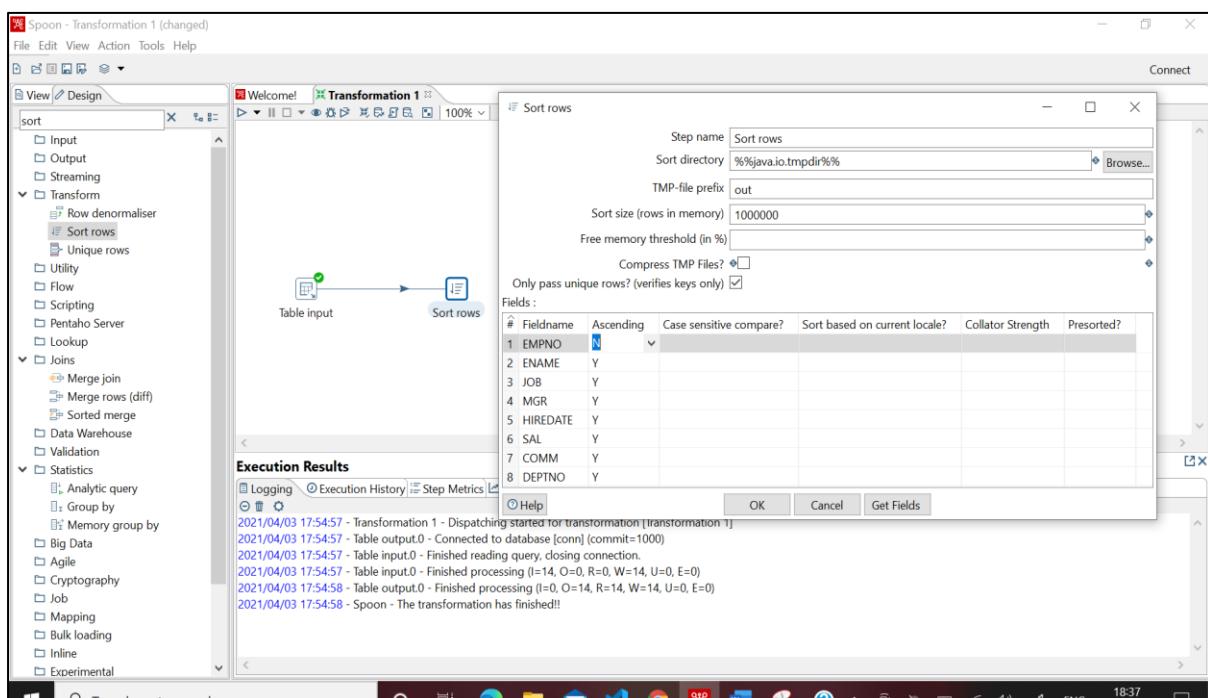
Table Input -> Preview -> OK -> Close -> OK



✓ Sorting Operation:

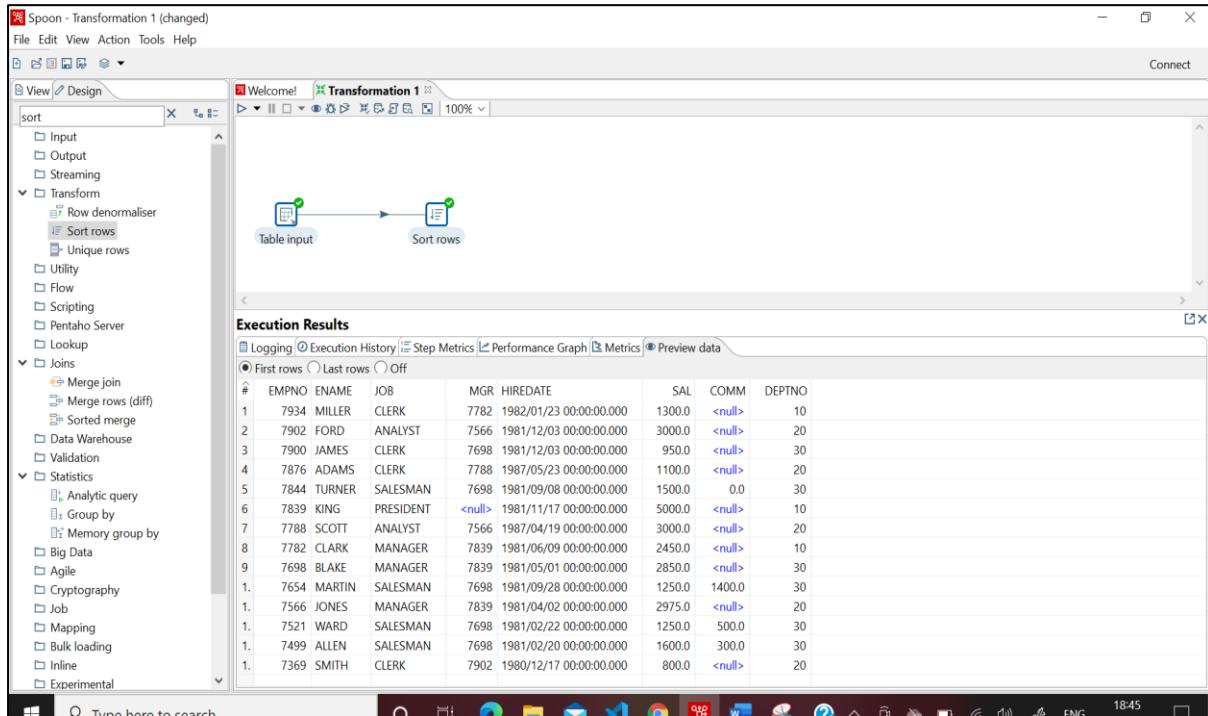
Design -> Transform -> Sort Rows (Drag & Drop in Transformation) -> Create Hop Connection with Sort Rows (Click on Output Connector arrow in Input Table)

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



✓ Debug Transformation:

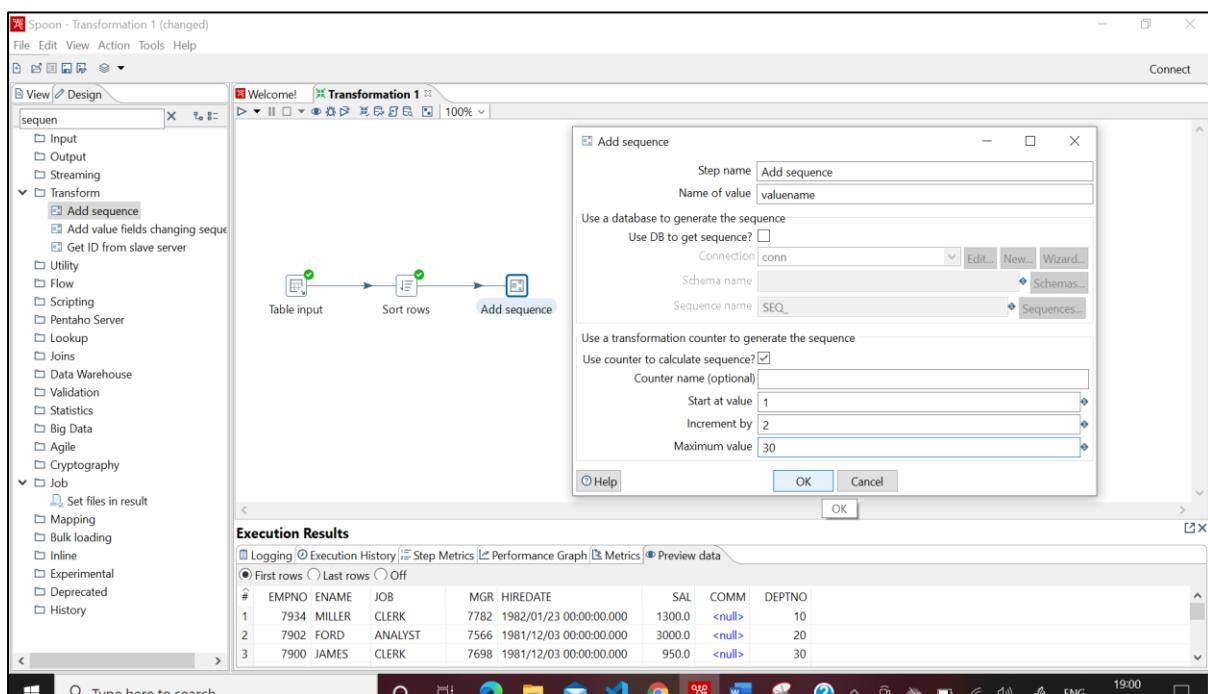
Click Sort Rows -> Debug -> Quick Launch -> Yes



✓ Adding Sequence:

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

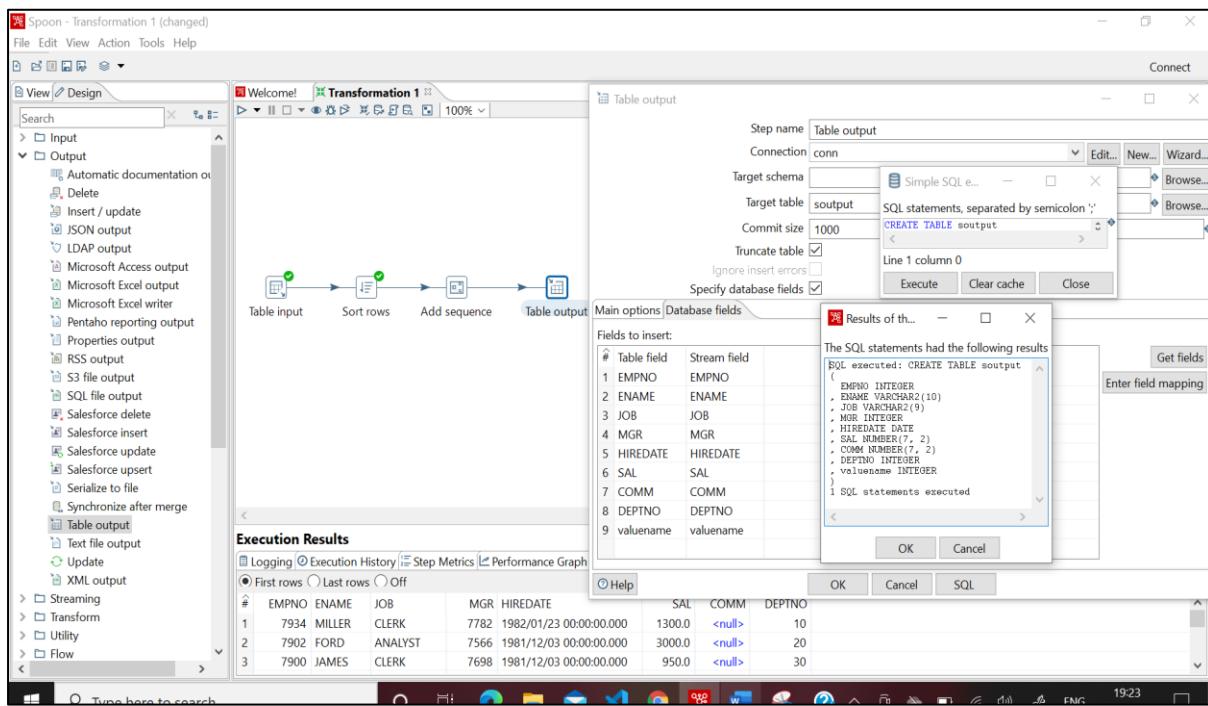
Double Click Add Sequence -> Start at value: 1 -> Increment by: 2 -> Maximum value: 30 -> OK



✓ Storing to **Output Table**:

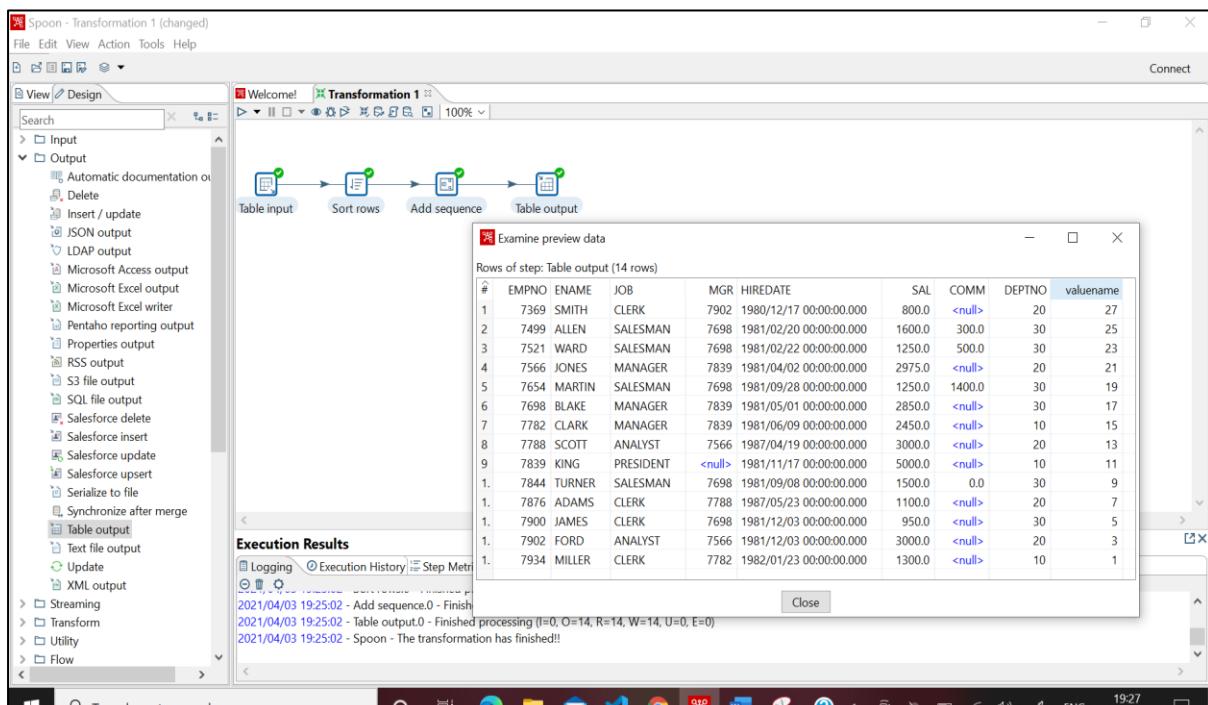
Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Add Sequence)

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

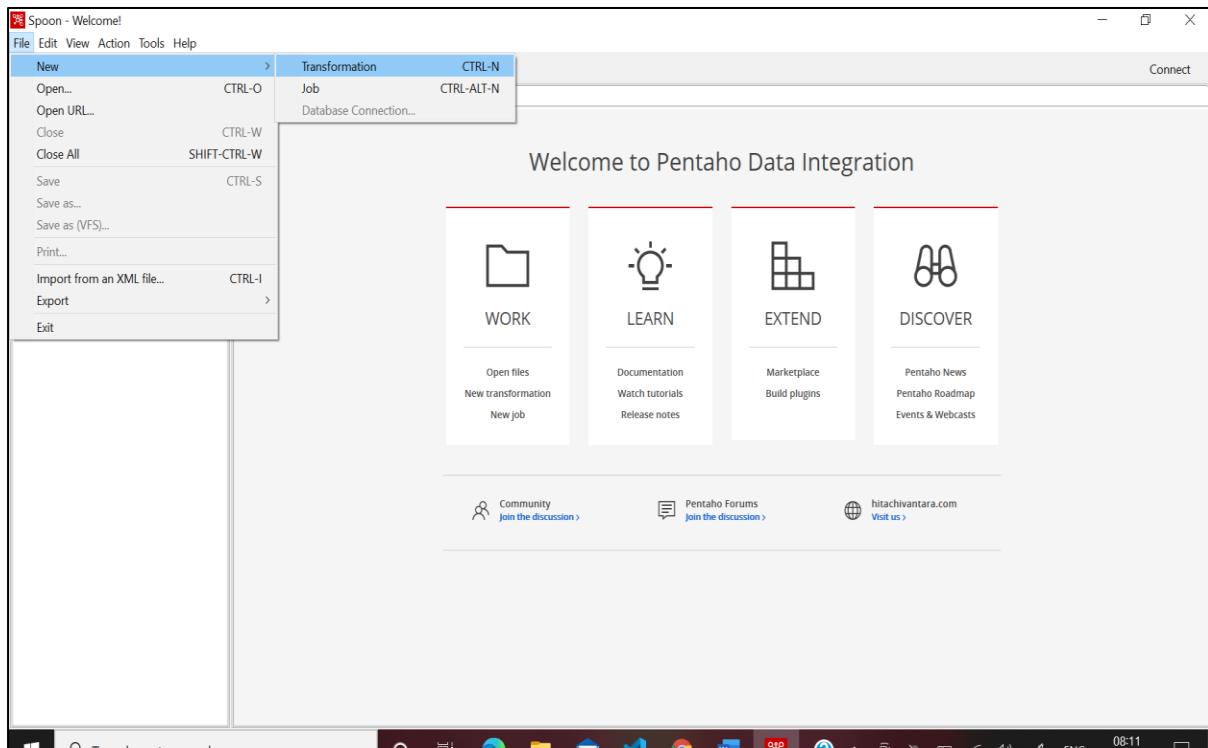


- Implementation of Calculator Operation.

### Steps:

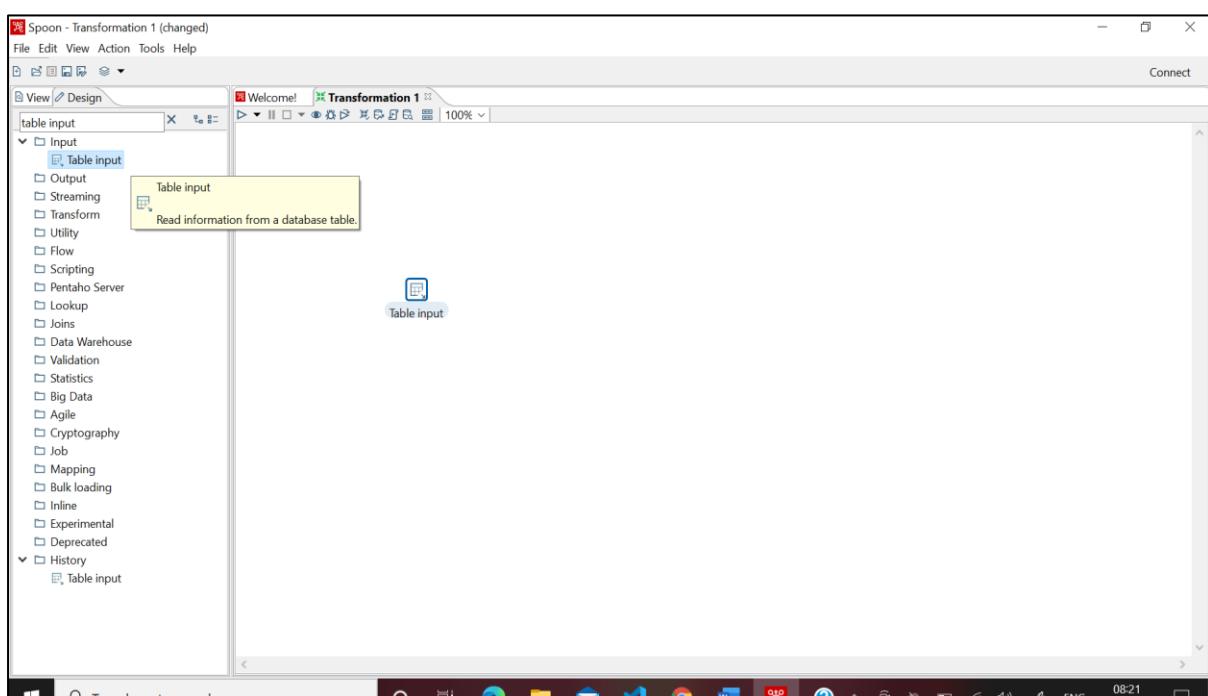
- ✓ Creating a New Transformation:

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



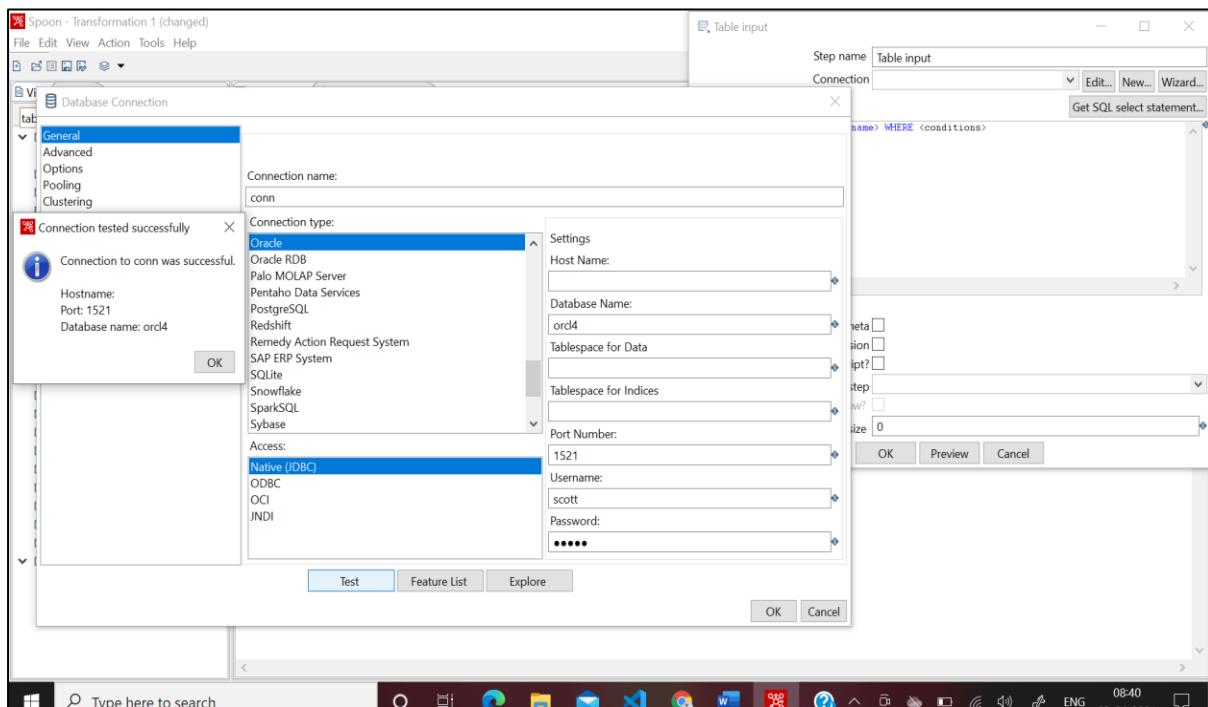
- ✓ Inserting **Table Input** into the Transformation:

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



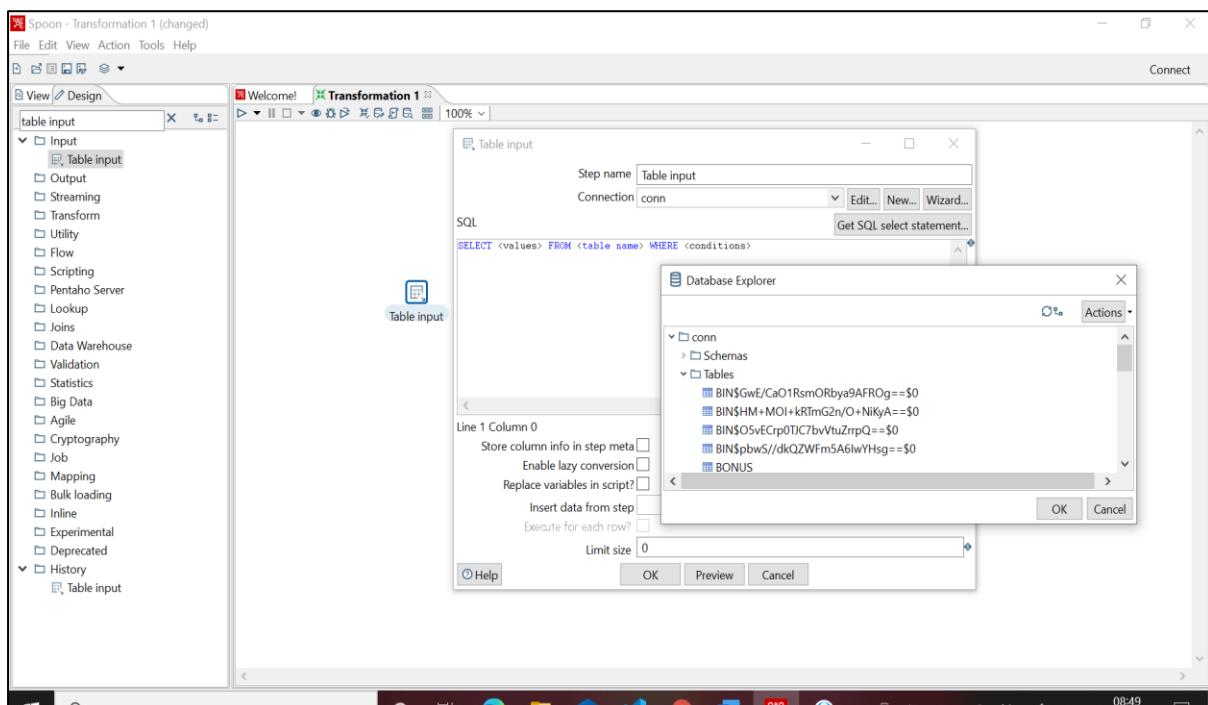
✓ Establishing Database Connection:

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



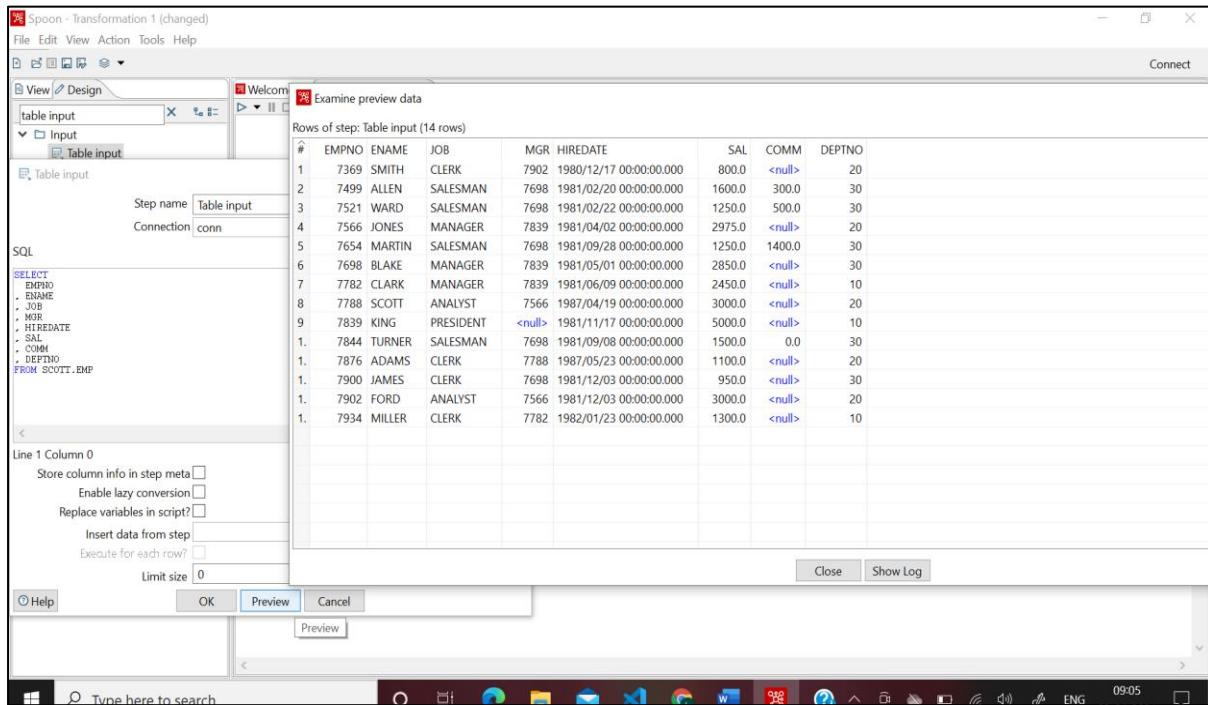
✓ Getting Source Table:

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



✓ View Input Table:

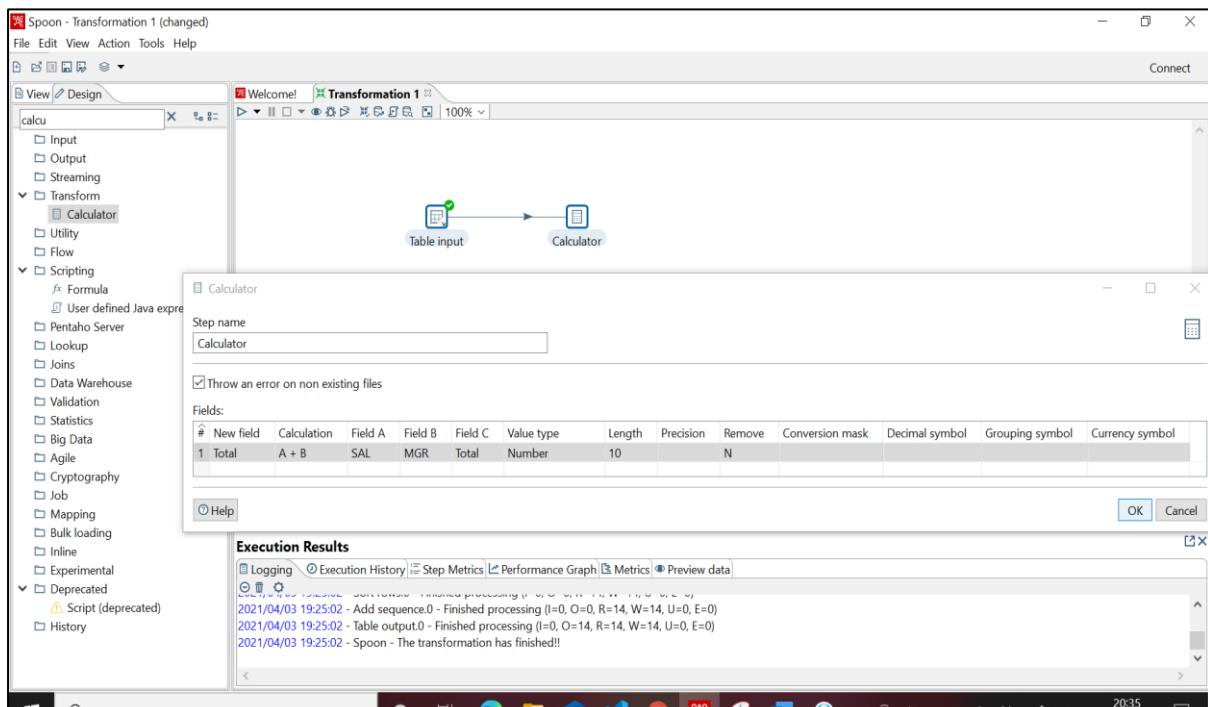
Table Input -> Preview -> OK -> Close -> OK



✓ Calculator Operation:

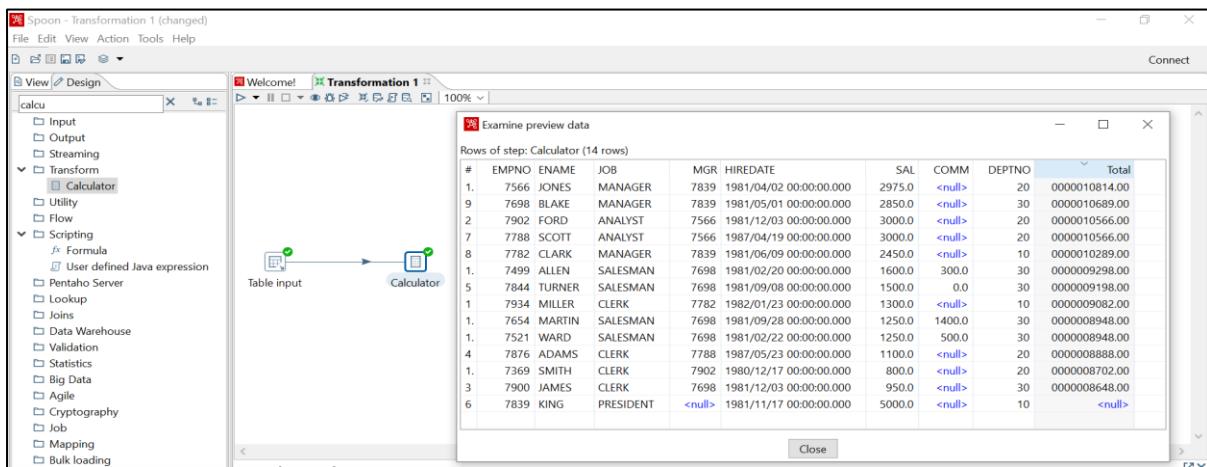
Design -> Transform -> Calculator (Drag & Drop in Transformation) -> Create Hop Connection with Calculator (Click on Output Connector arrow in Input Table)

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:**

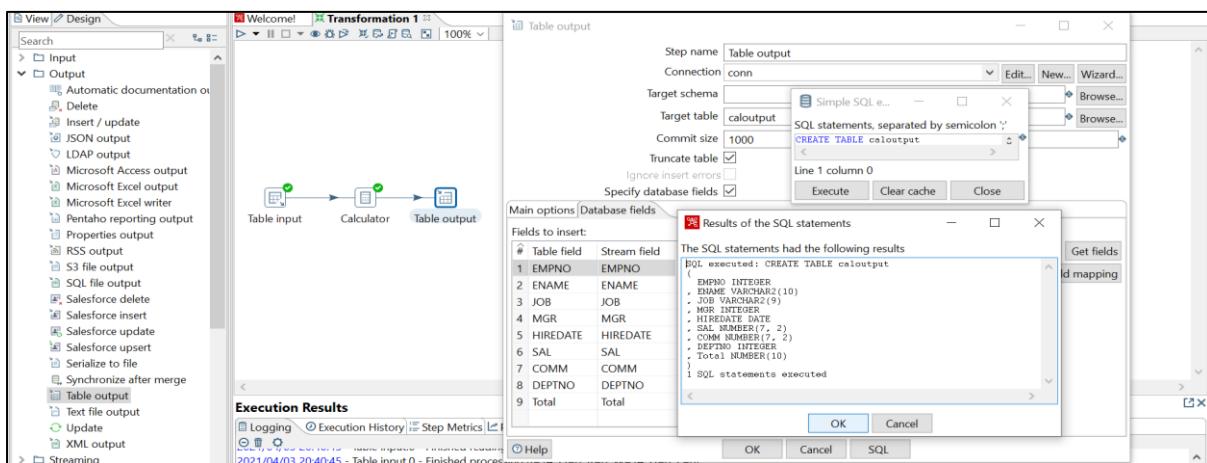
Click Calculator -> Debug -> Quick Launch -> Yes



✓ **Storing to Output Table:**

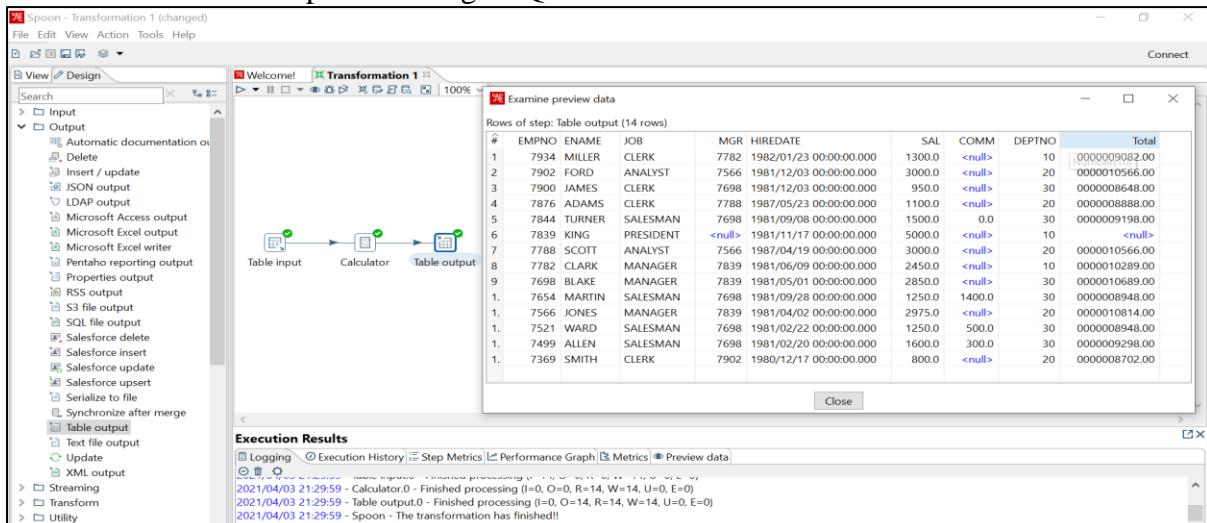
Design -> Output -> Table Output (Drag & Drop in Transformation) -> Create Hop Connection with Output Table (Click on Output Connector arrow in Calculator)

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

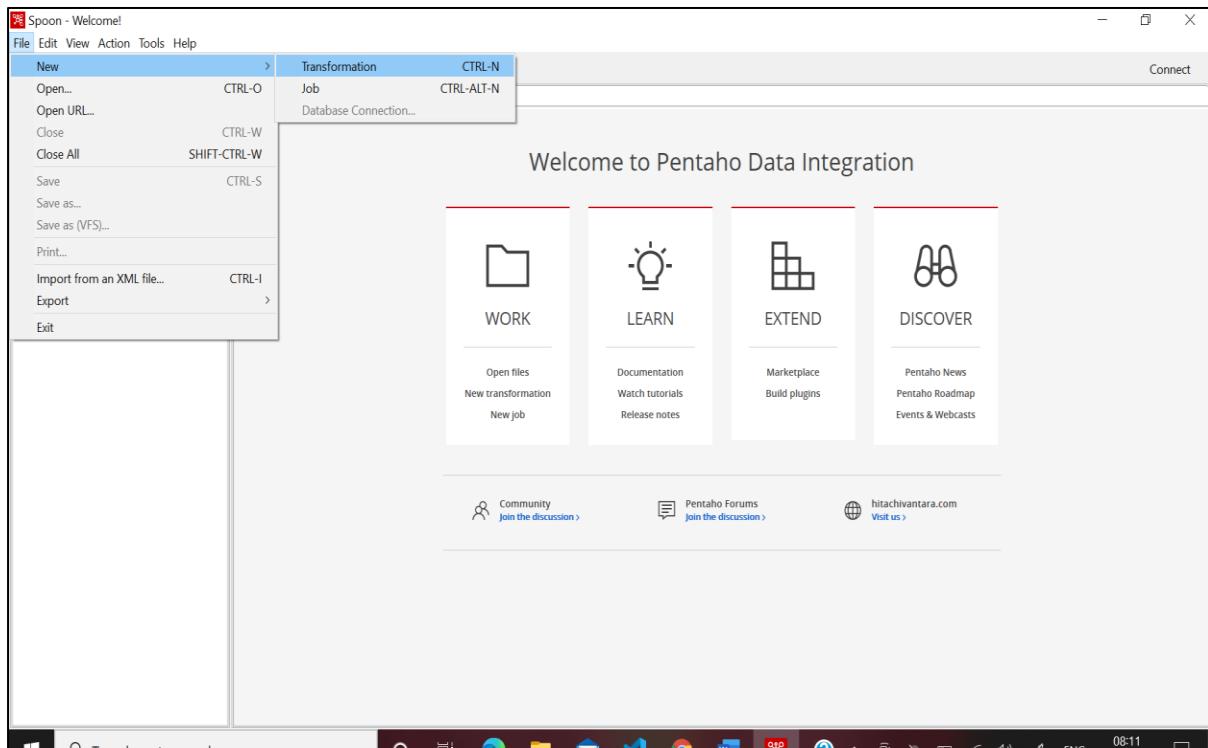


- Implementation of Concatenation Operation.

### **Steps:**

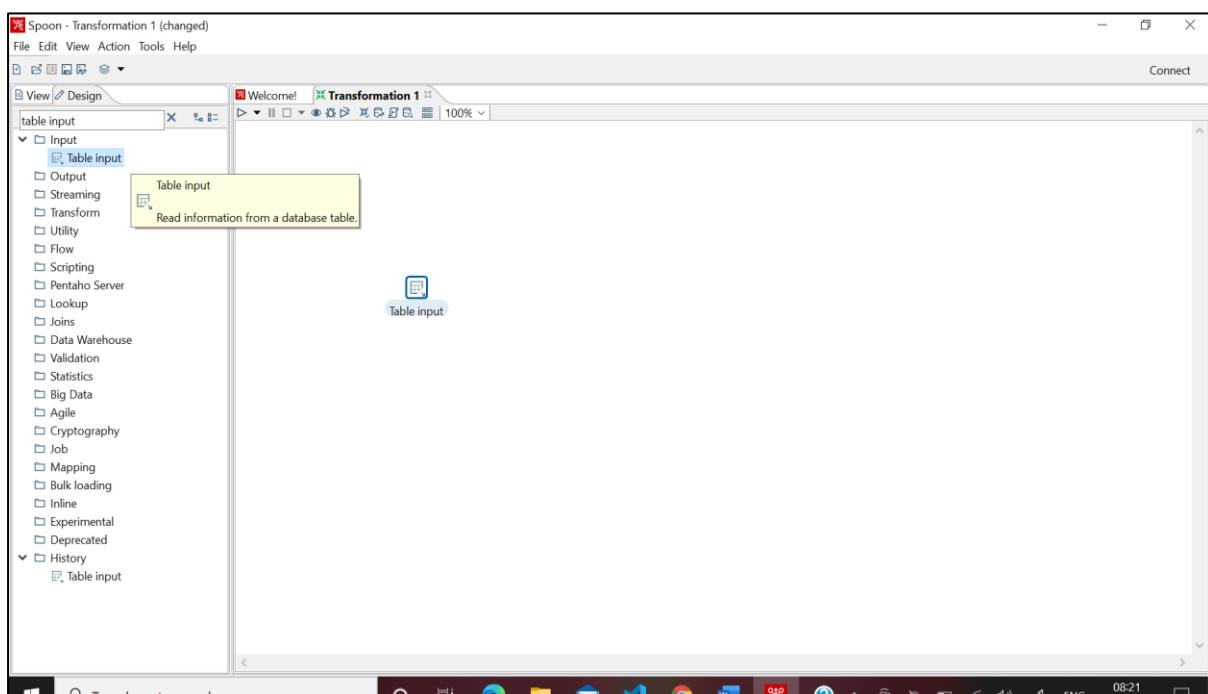
- ✓ Creating a New Transformation:

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



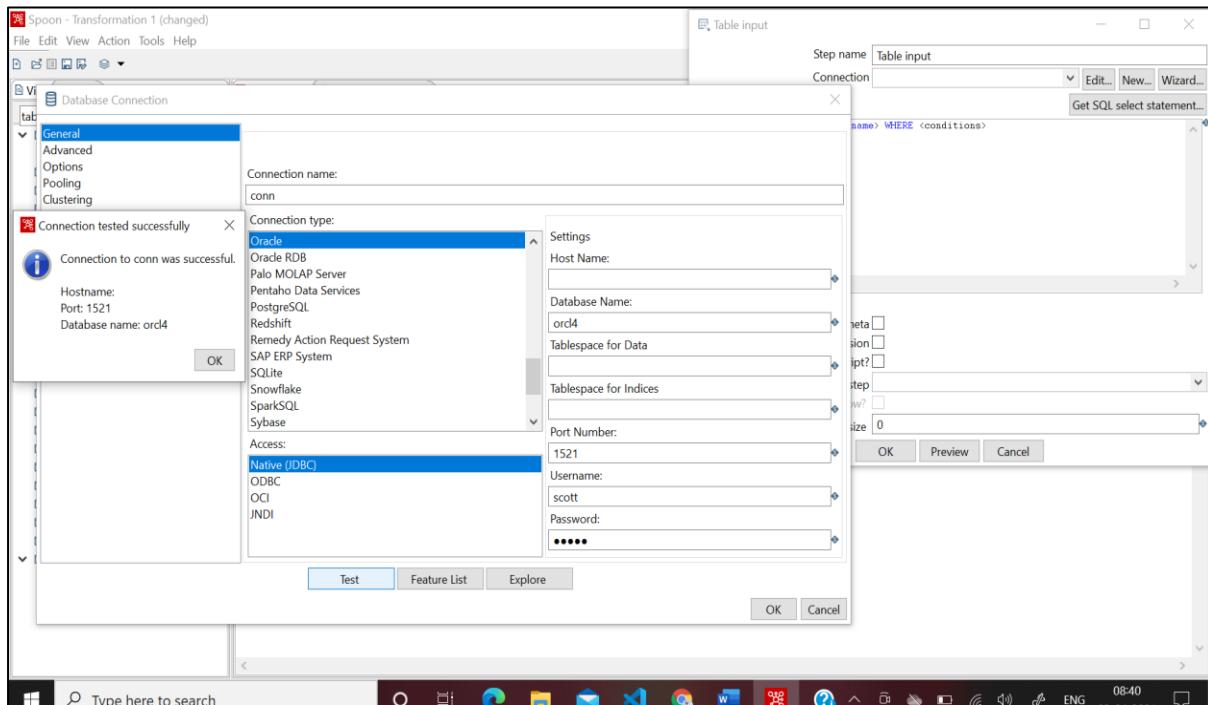
- ✓ Inserting **Table Input** into the Transformation:

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



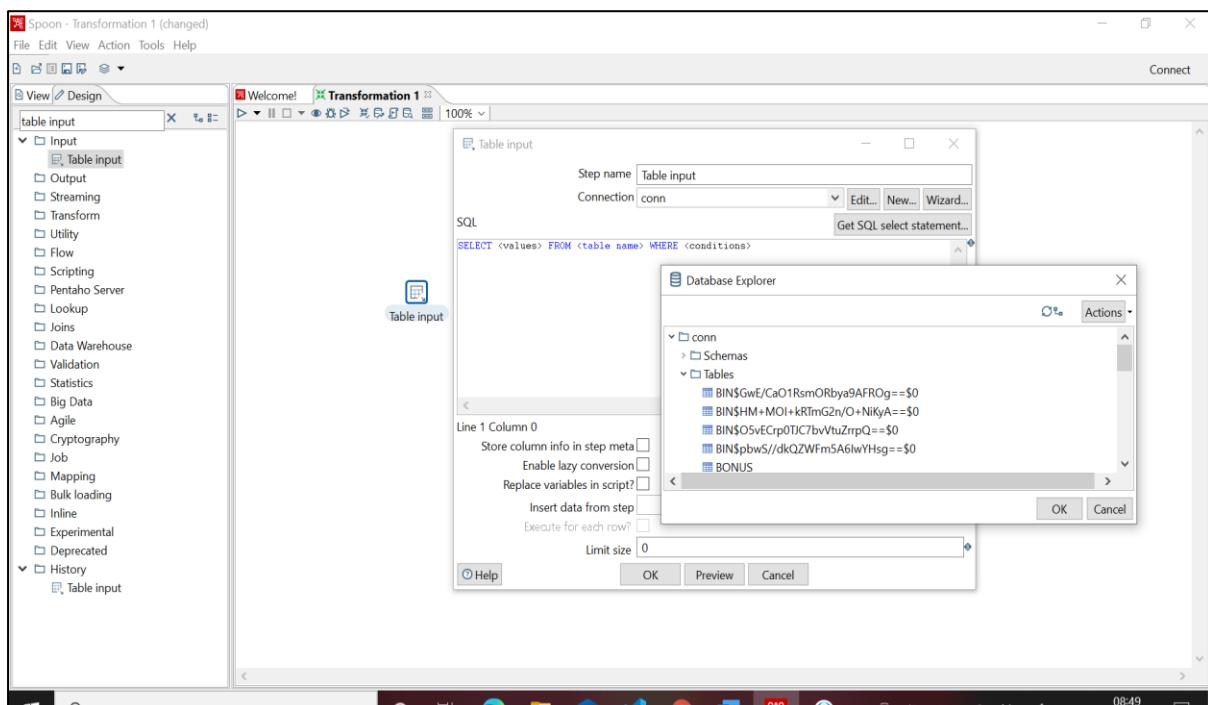
✓ Establishing Database Connection:

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



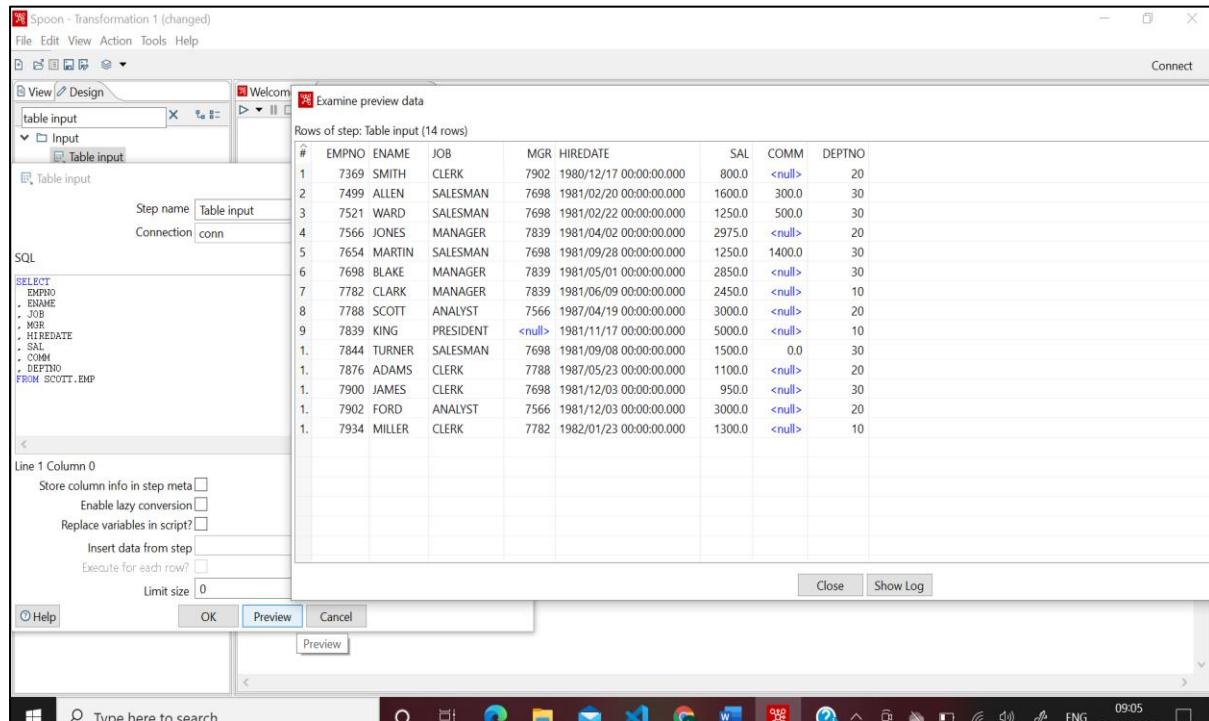
✓ Getting Source Table:

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



✓ View Input Table:

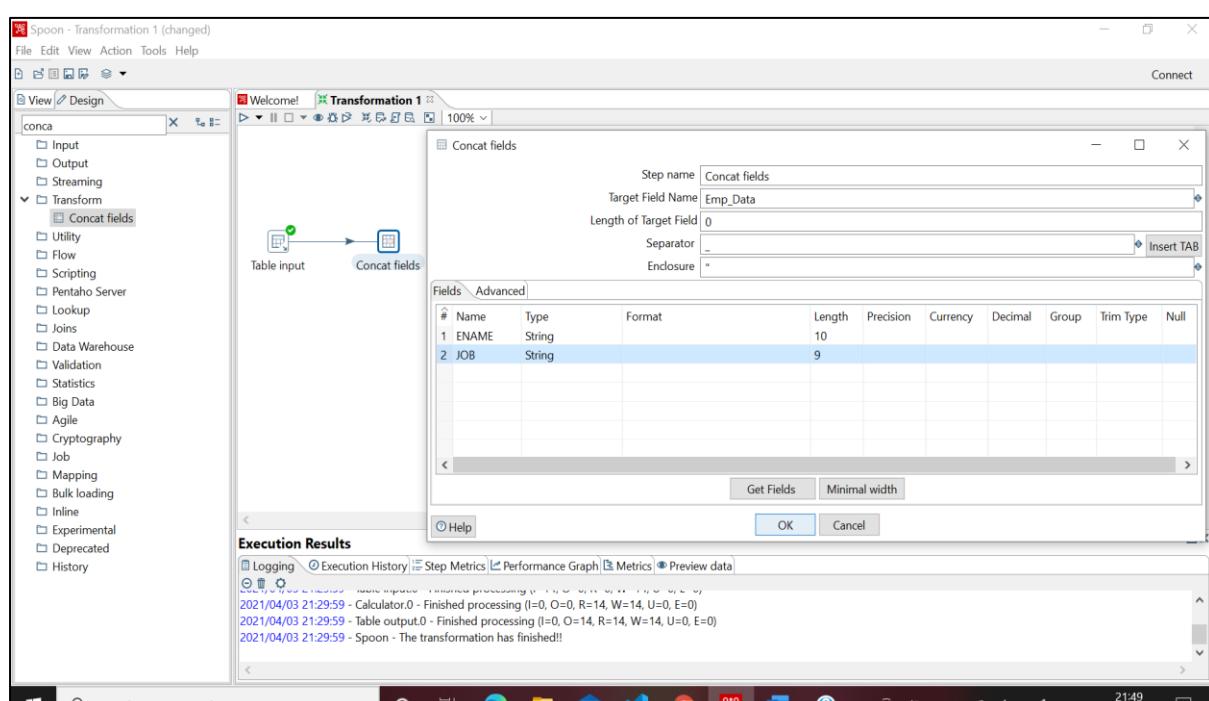
Table Input -> Preview -> OK -> Close -> OK



✓ Concatenation Operation:

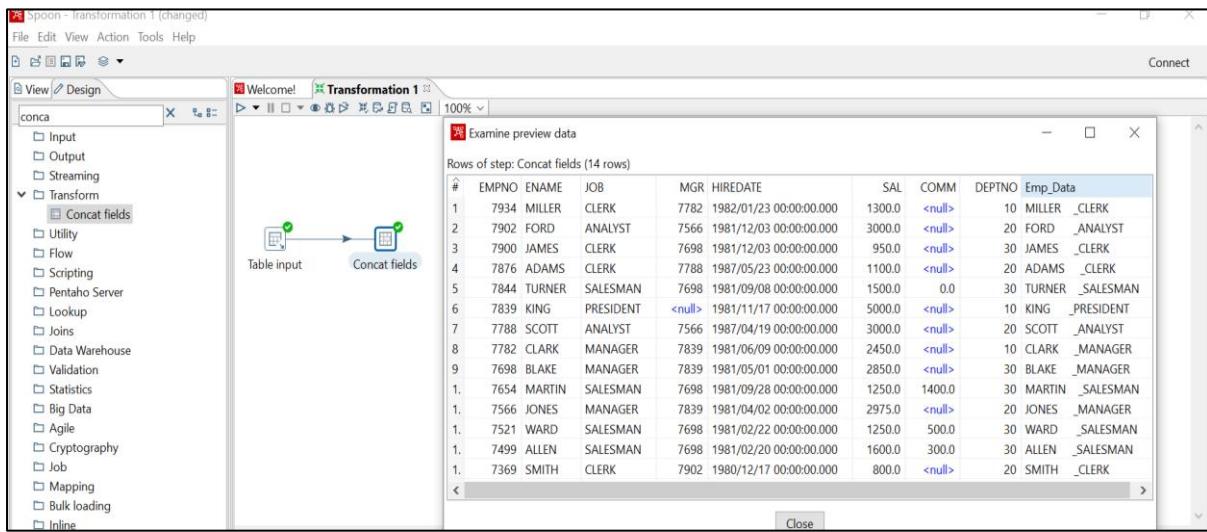
Design-> Transform-> Concat Fields (Drag & Drop in Transformation) -> Create Hop Connection with Concat Fields (Click on Output Connector arrow in Input Table)

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



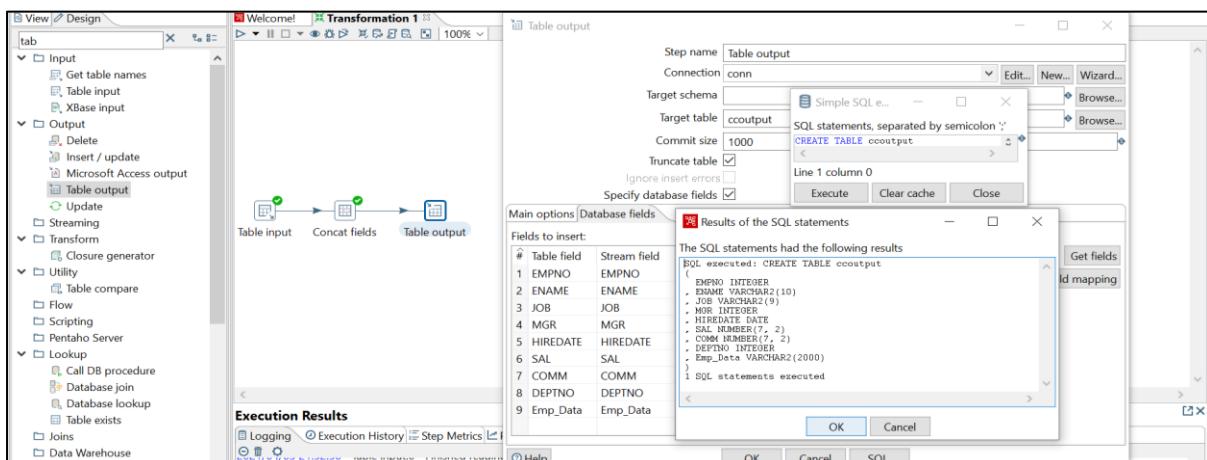
✓ **Debug Transformation:**

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



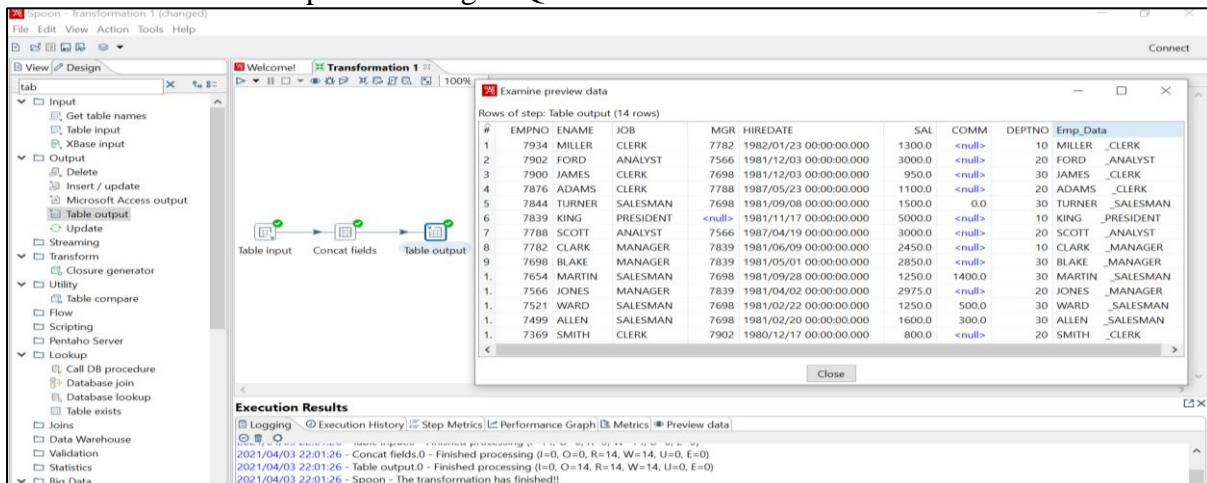
✓ **Storing to Output Table:**

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

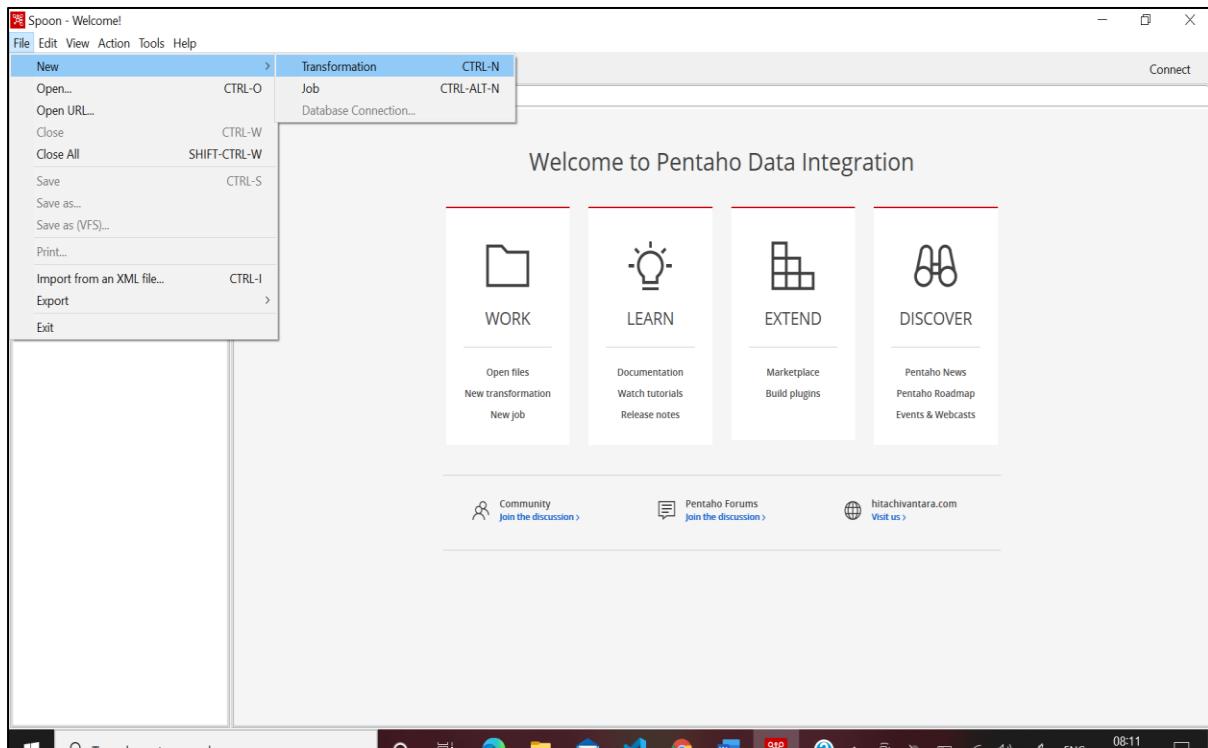


- Implementation of Splitting Operation.

### Steps:

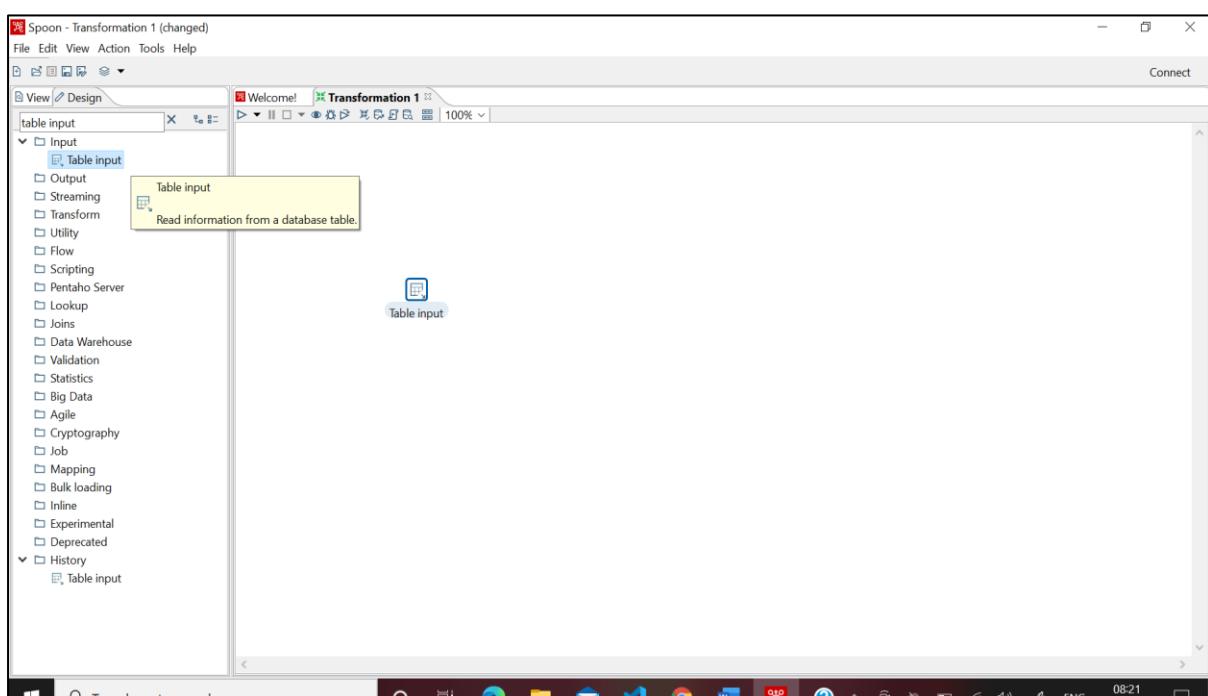
- ✓ Creating a New Transformation:

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



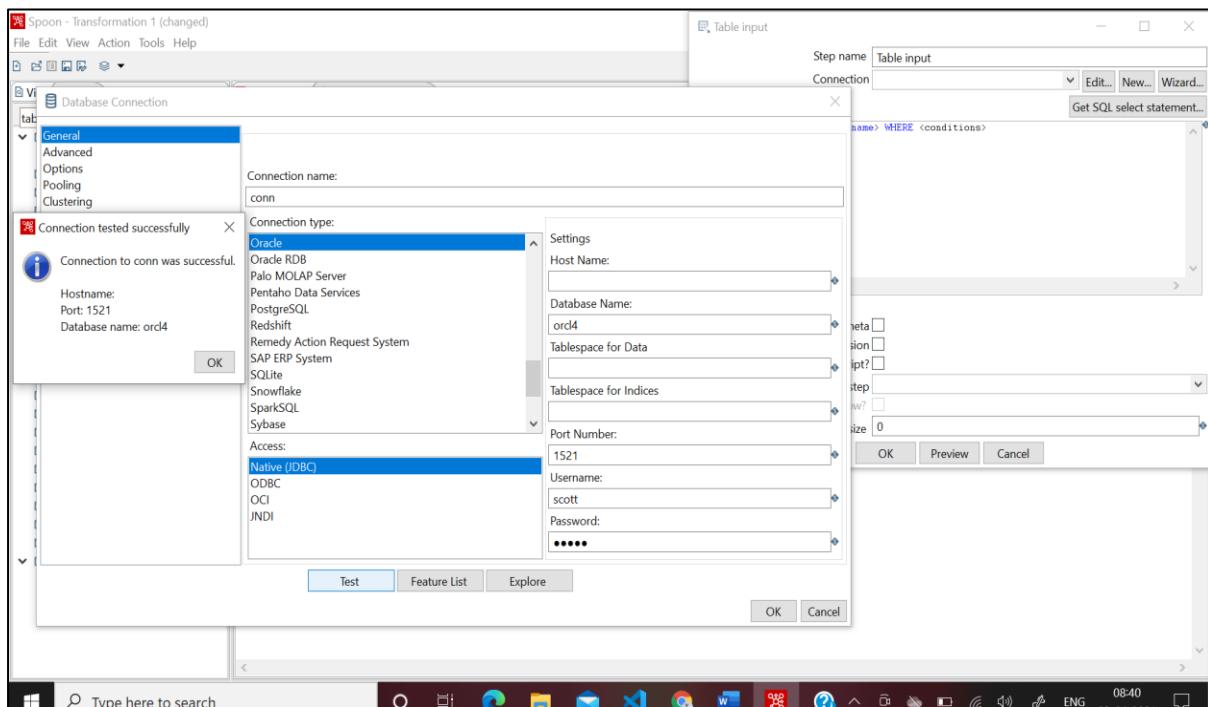
- ✓ Inserting **Table Input** into the Transformation:

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



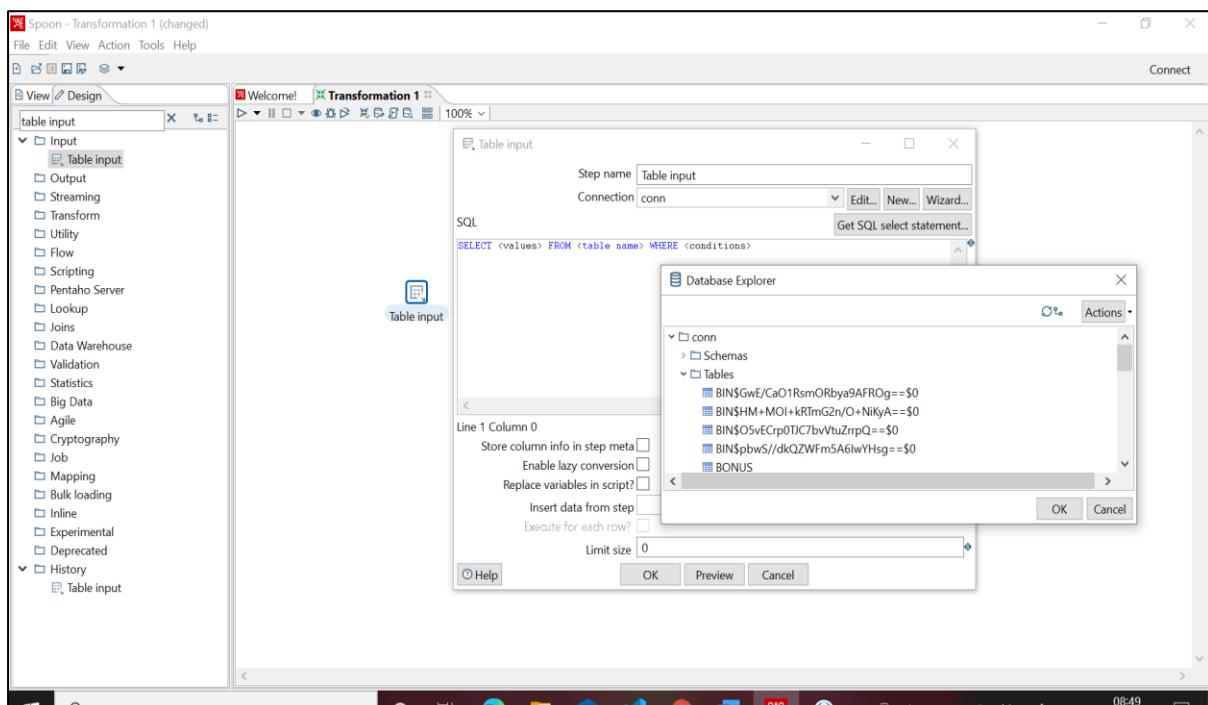
✓ Establishing Database Connection:

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



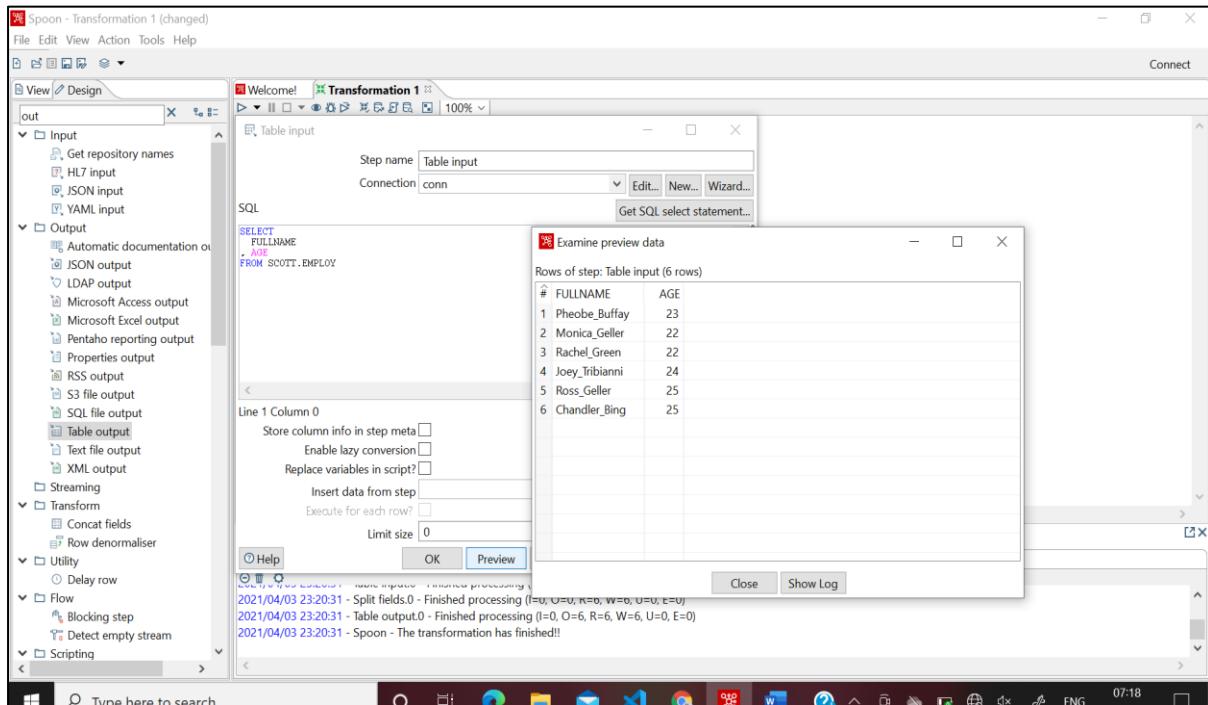
✓ Getting Source Table:

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



✓ **View Input Table:**

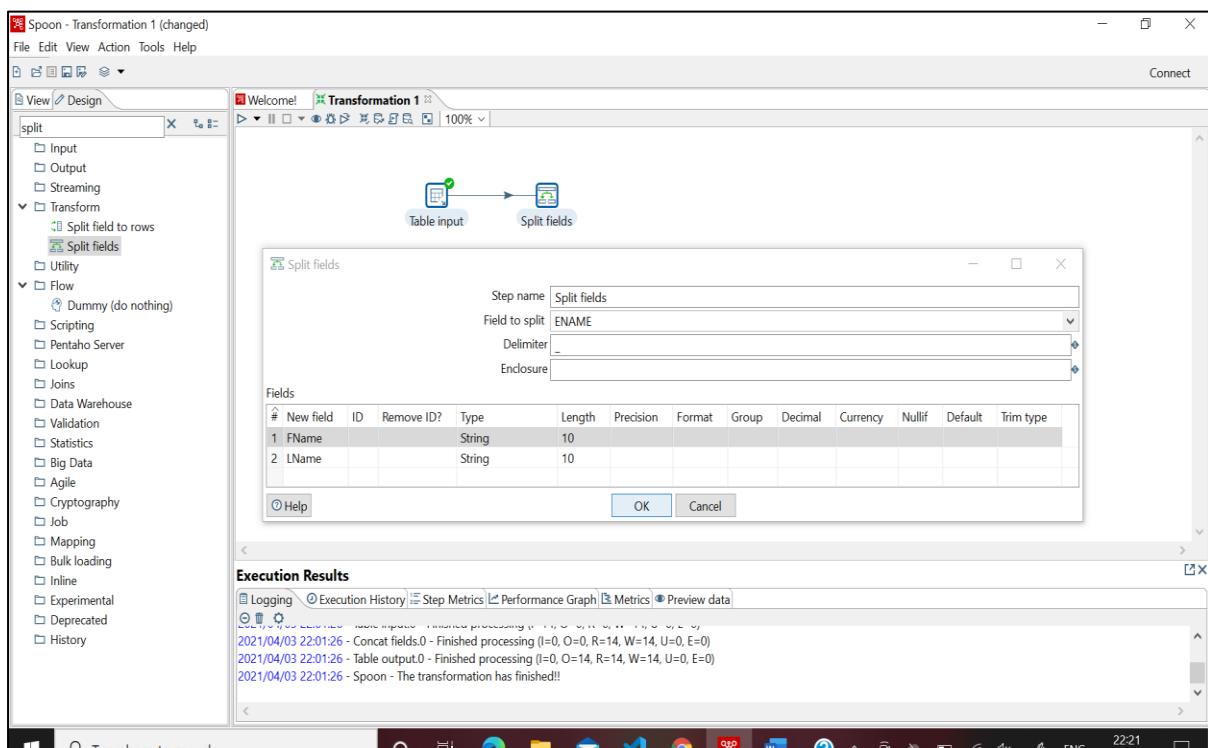
Table Input -> Preview -> OK -> Close -> OK



✓ **Splitting Operation:**

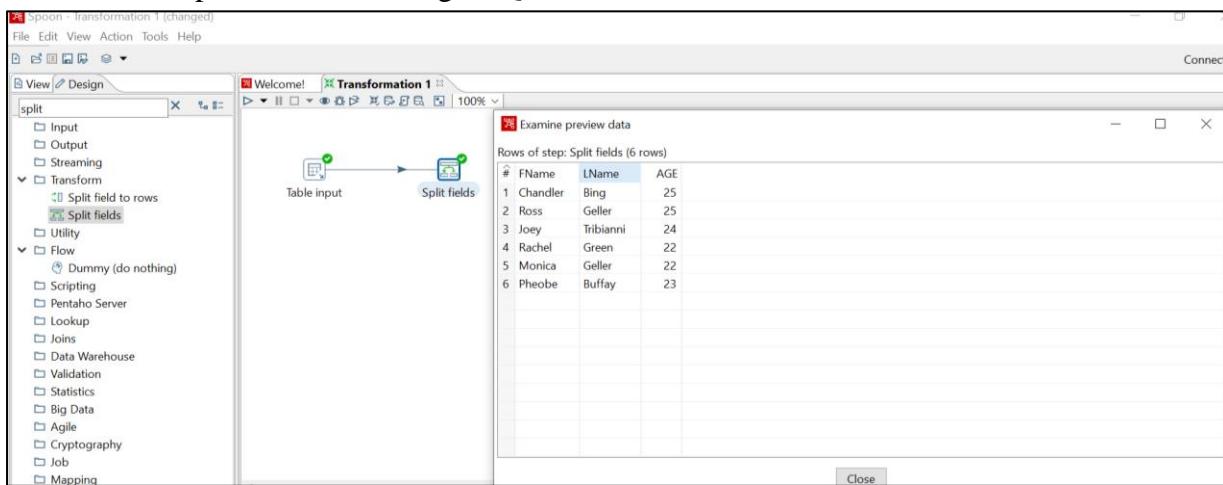
Design -> Transform -> Split Fields (Drag & Drop in Transformation) -> Create Hop Connection with Split Fields (Click on Output Connector arrow in Input Table)

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



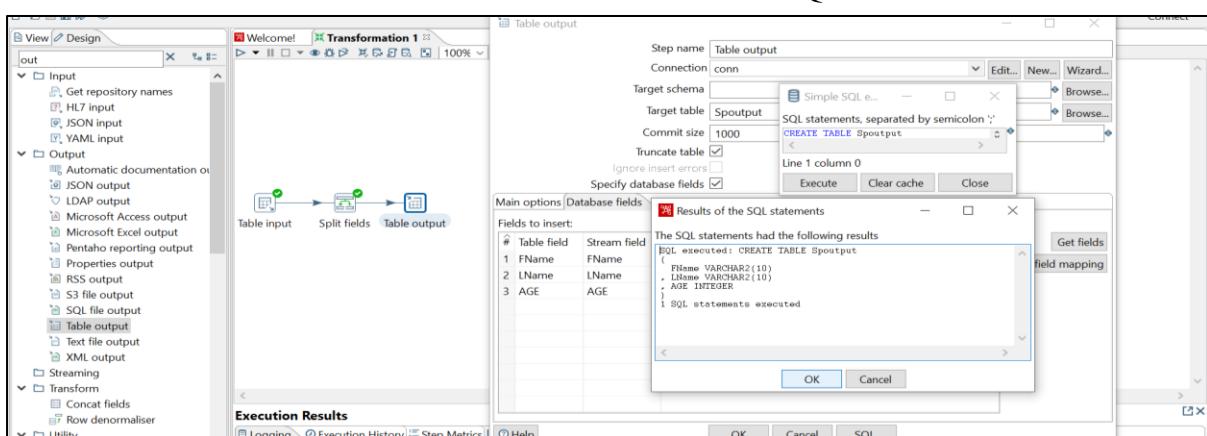
✓ Debug Transformation:

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



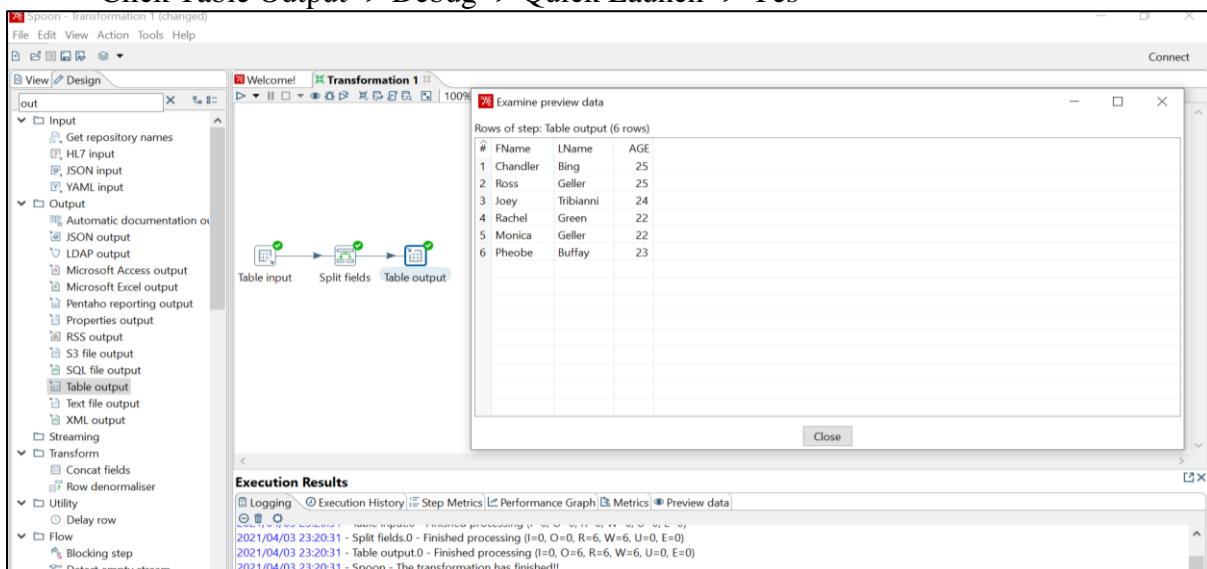
✓ Storing to Output Table:

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

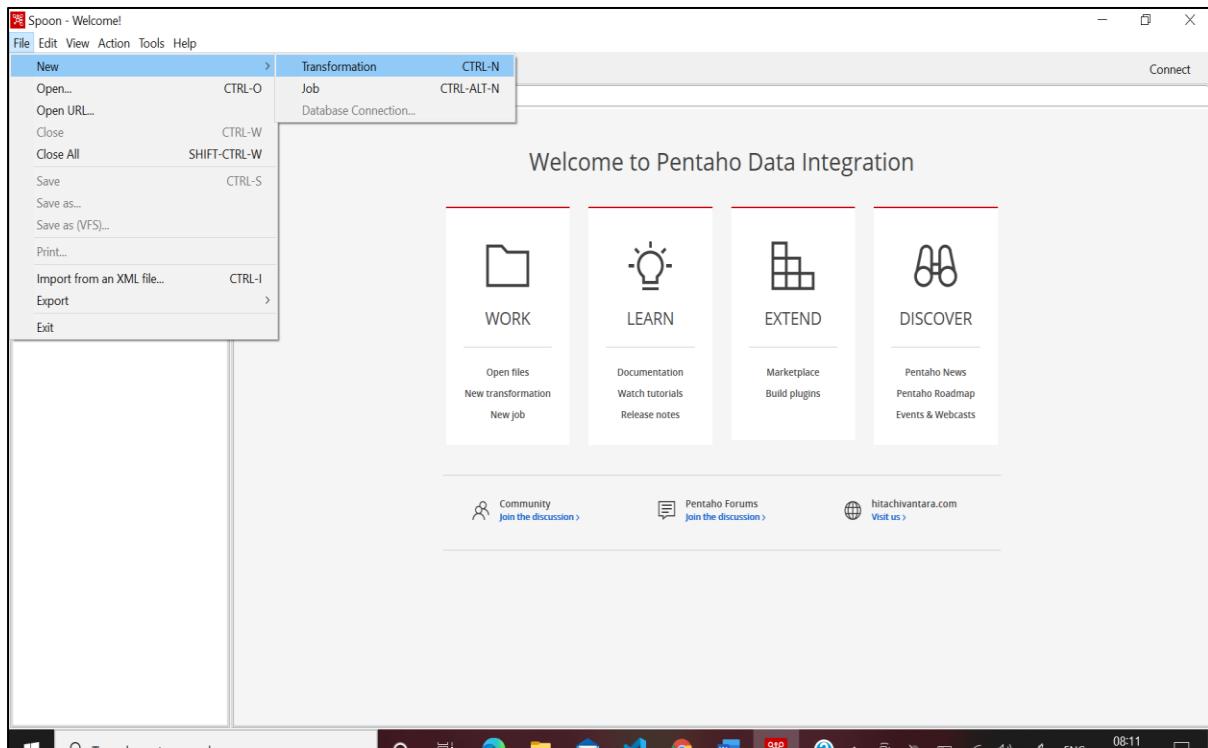


- Implementation of Number Range.

### Steps:

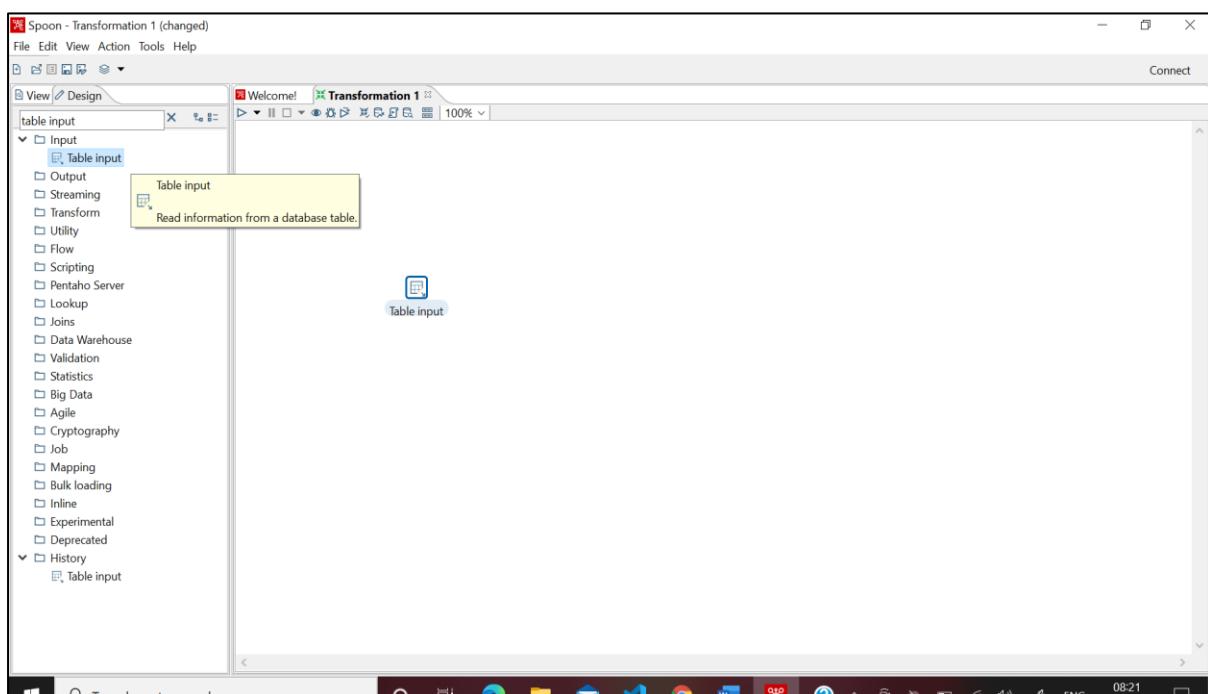
- ✓ Creating a New Transformation:

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



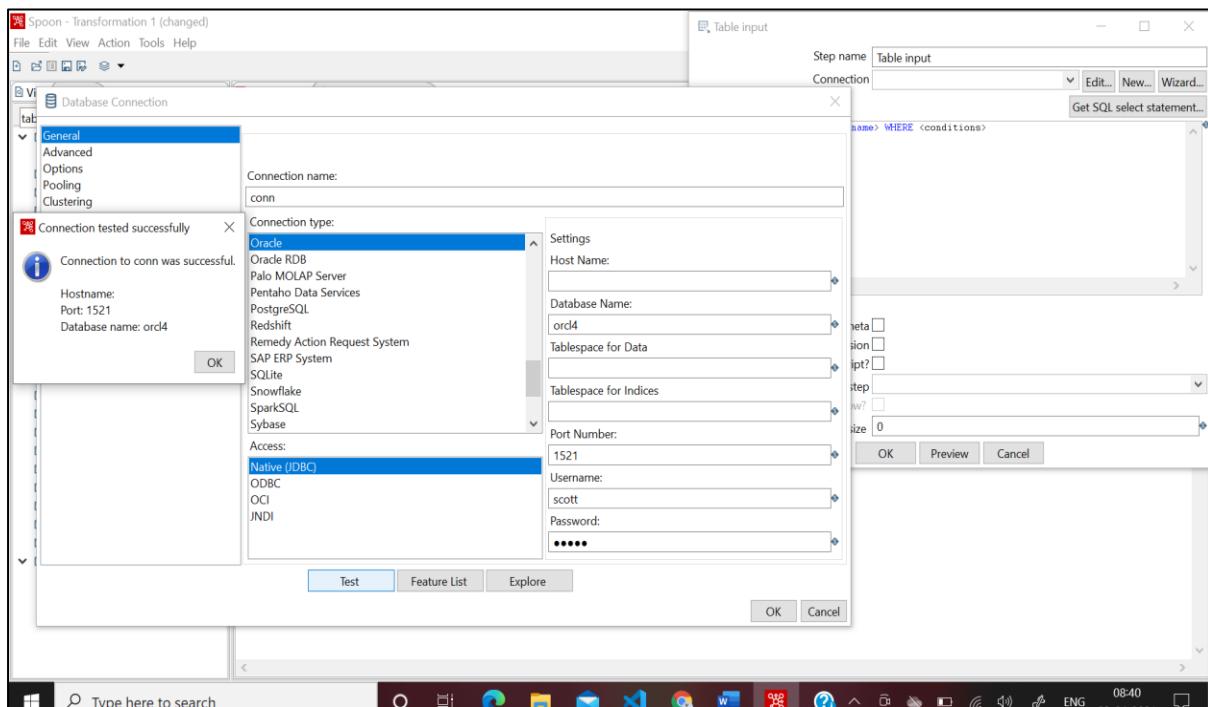
- ✓ Inserting **Table Input** into the Transformation:

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



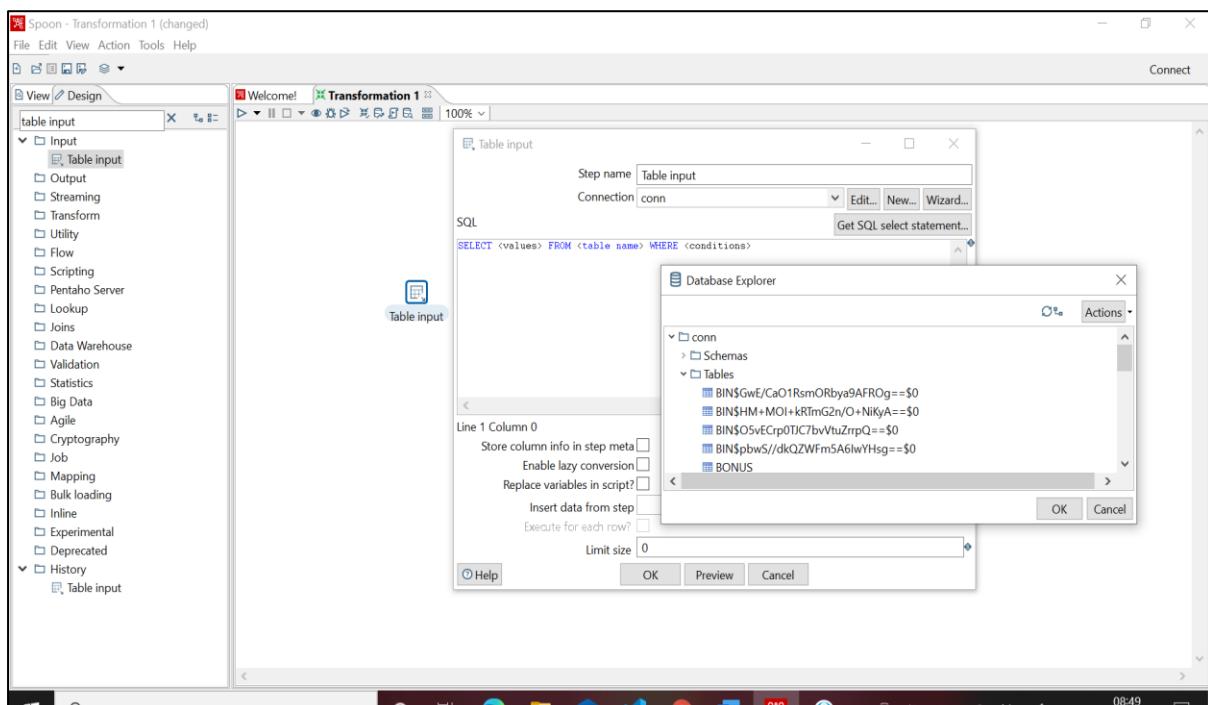
✓ Establishing Database Connection:

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



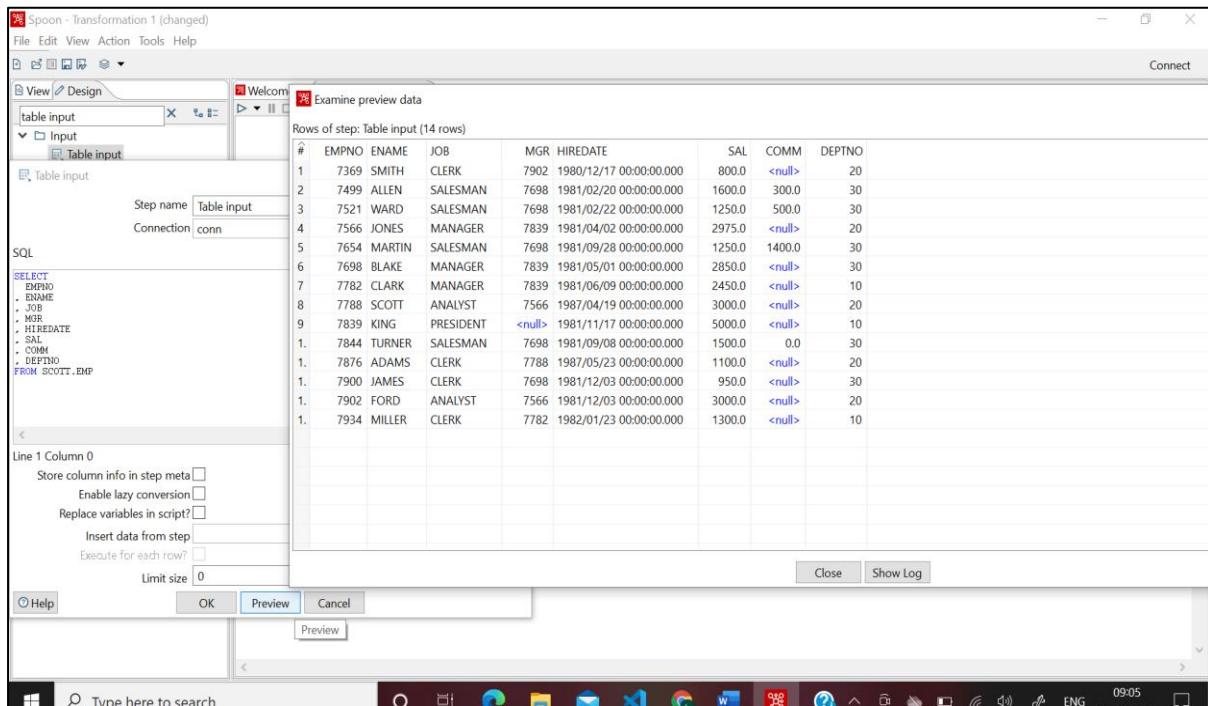
✓ Getting Source Table:

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



✓ View Input Table:

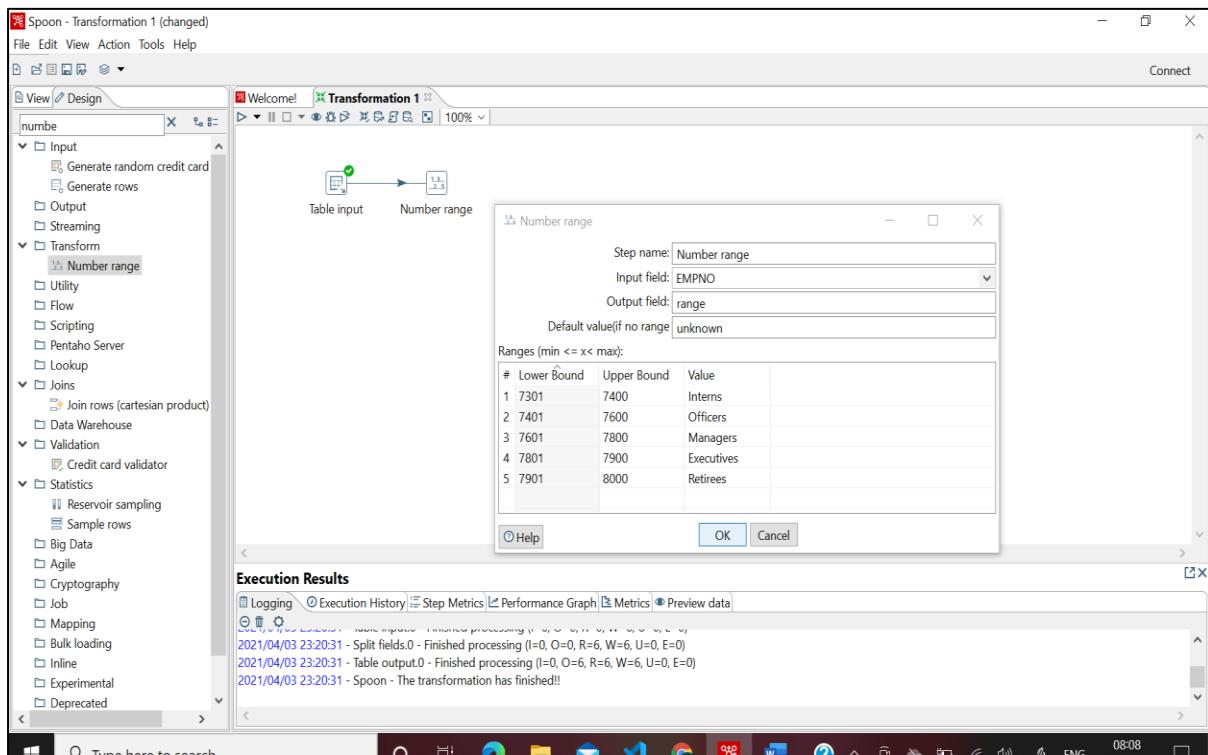
Table Input -> Preview -> OK -> Close -> OK



✓ Number Range:

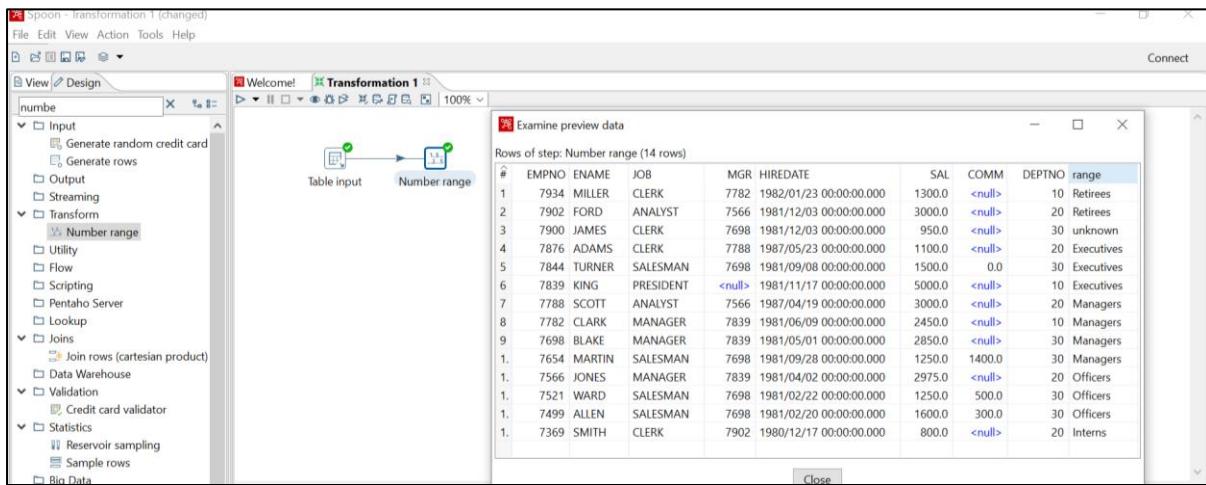
Design-> Transform-> Number Range(Drag & Drop in Transformation)->Create Hop Connection with Number Range (Click on Output Connector arrow in Input Table)

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



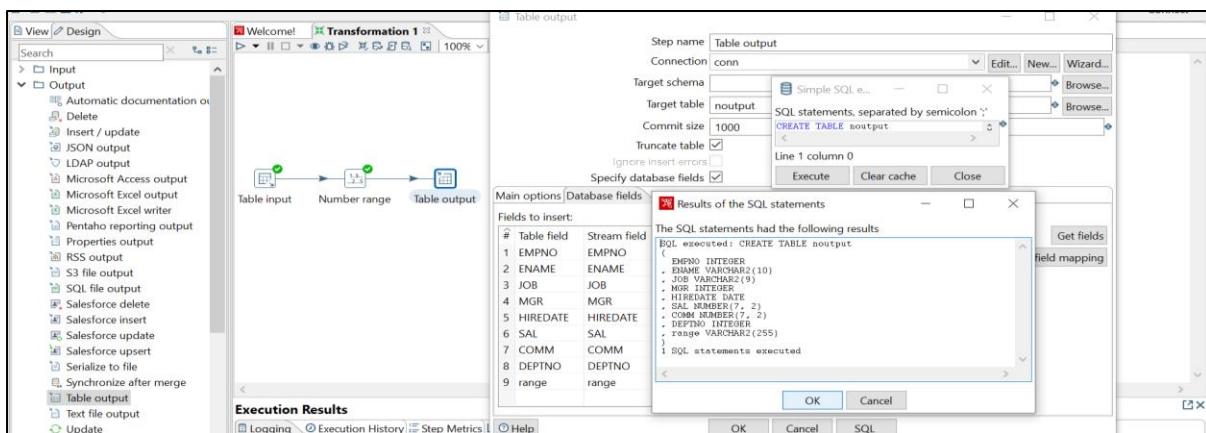
✓ Debug Transformation:

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



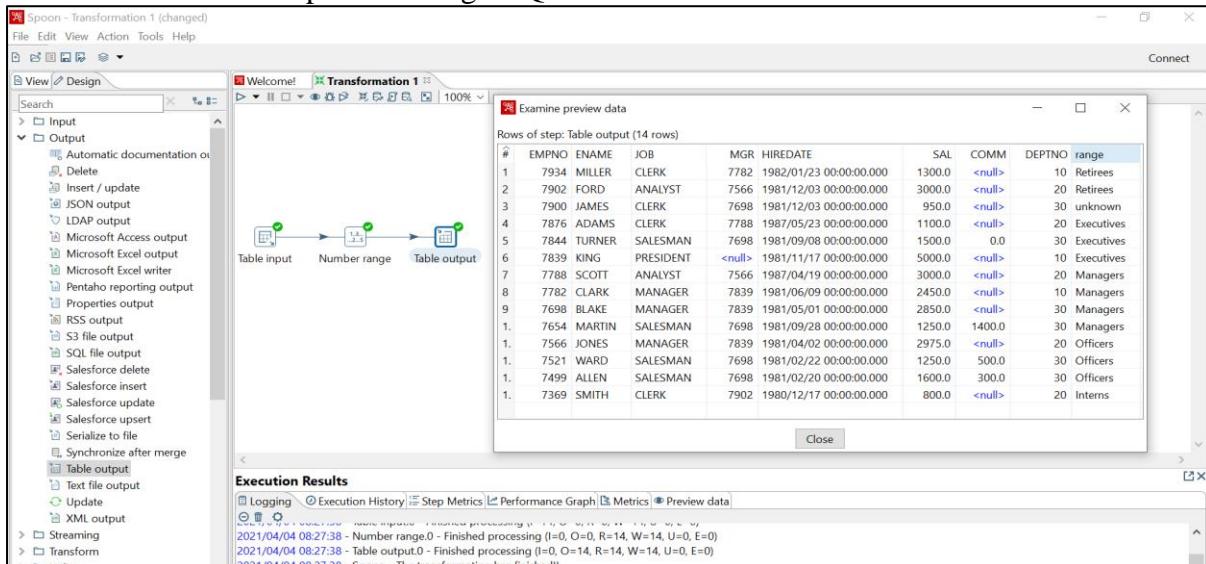
✓ Storing to Output Table:

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

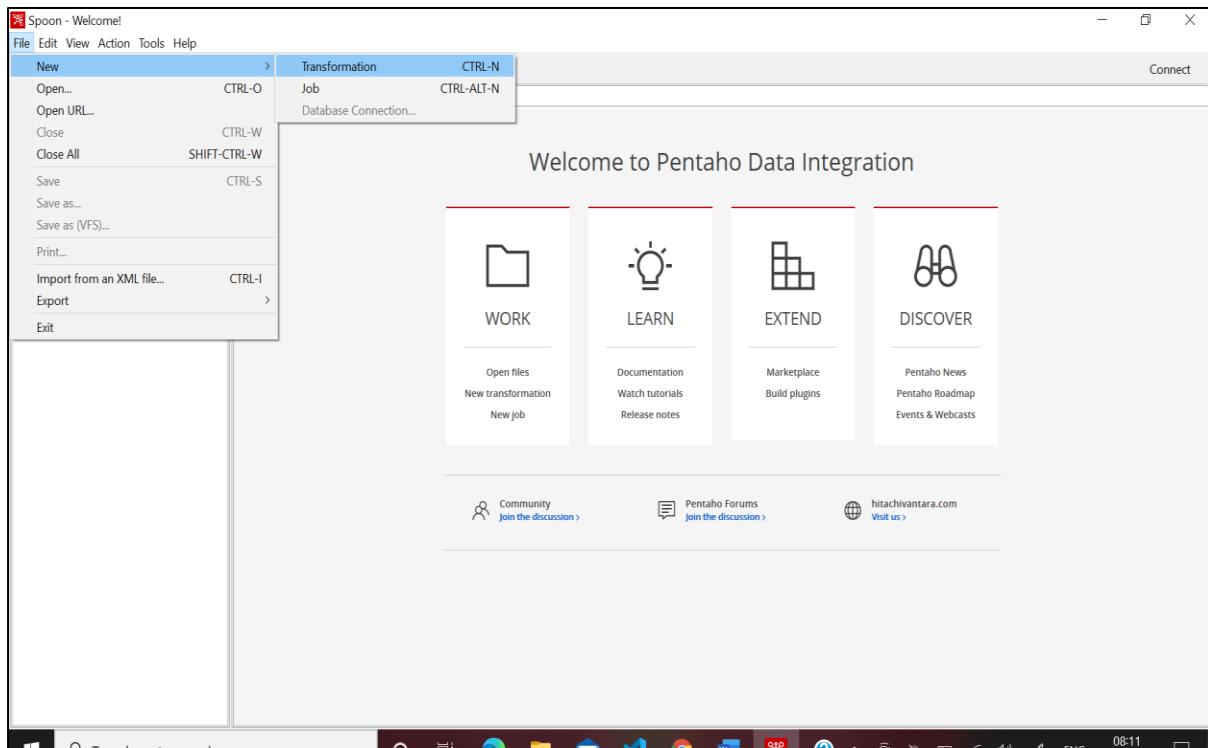


- Implementation of String Operation.

### **Steps:**

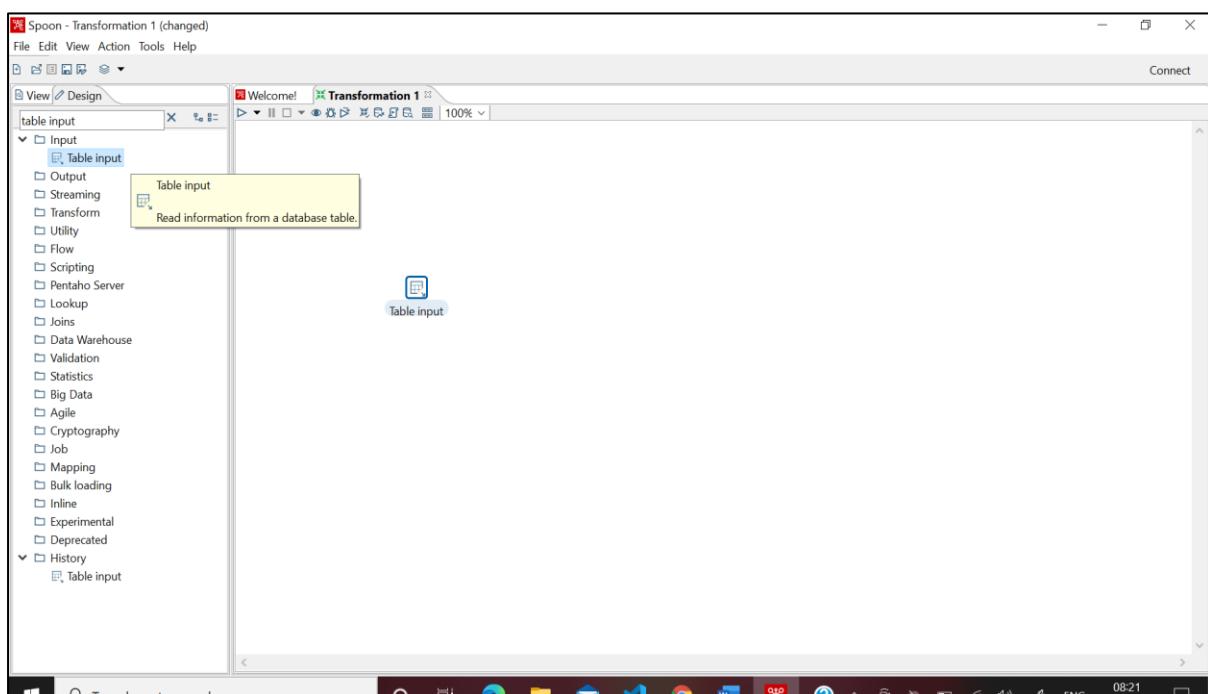
- ✓ Creating a New Transformation:

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



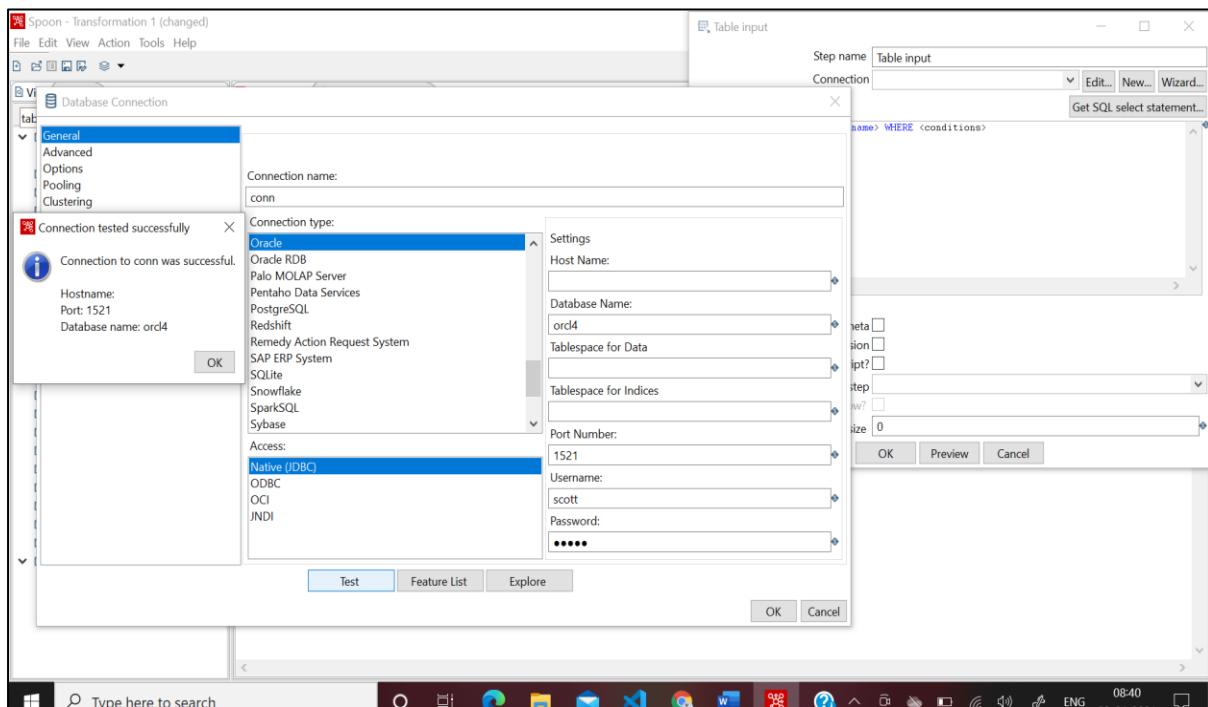
- ✓ Inserting **Table Input** into the Transformation:

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



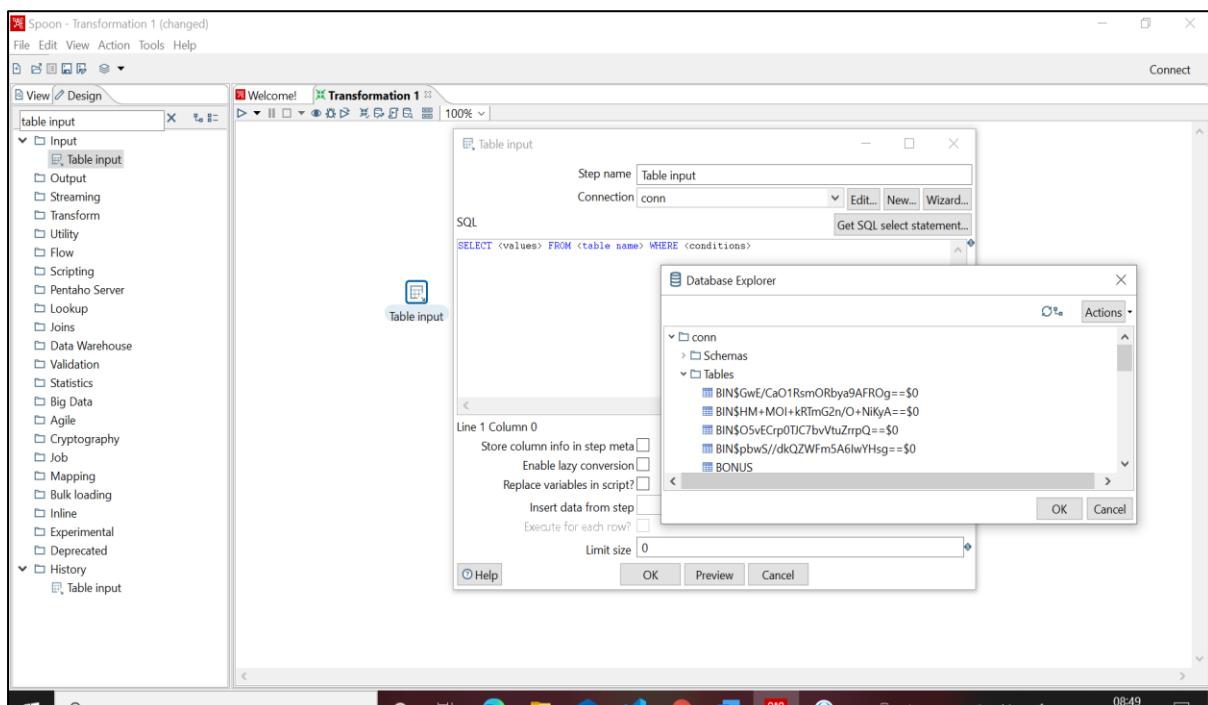
✓ Establishing Database Connection:

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



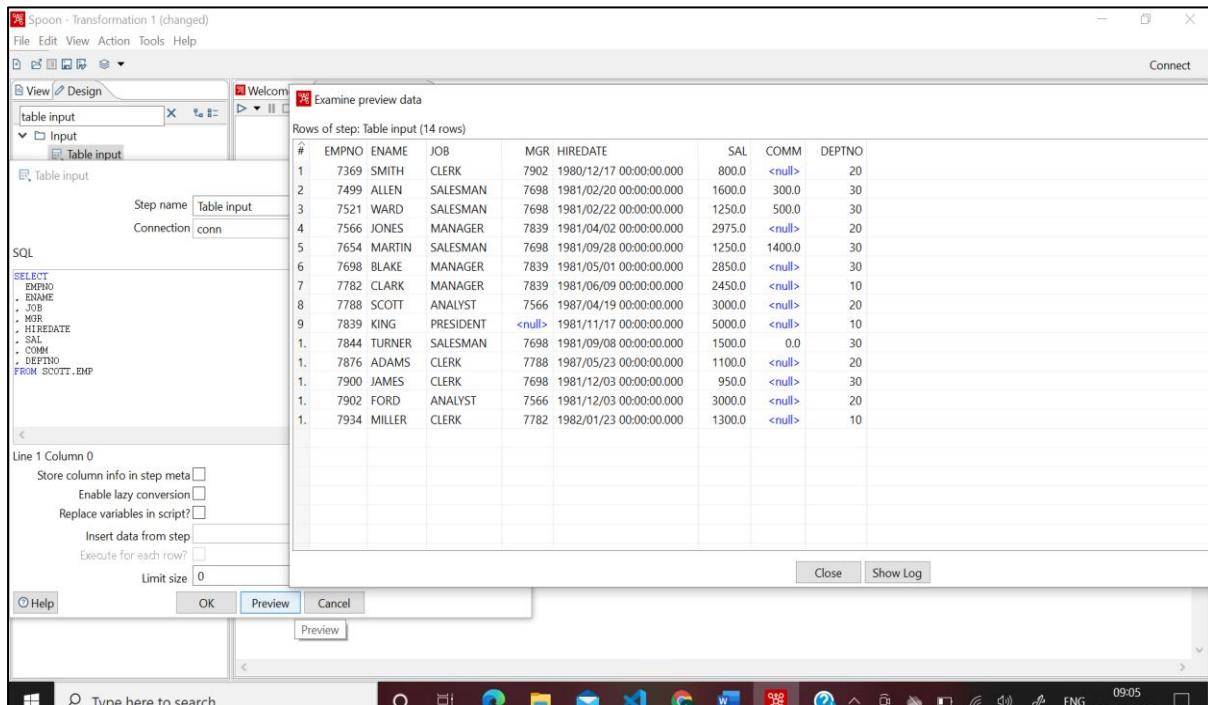
✓ Getting Source Table:

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



✓ View Input Table:

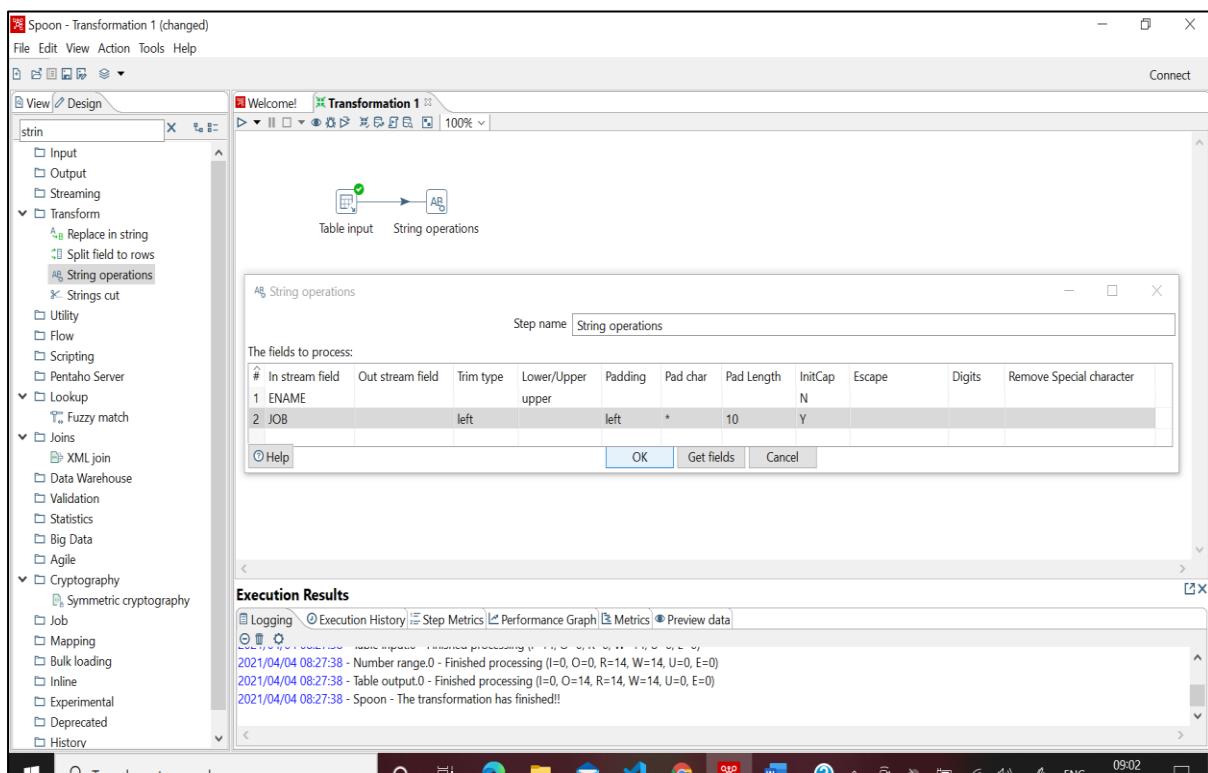
Table Input -> Preview -> OK -> Close -> OK



✓ String Operations:

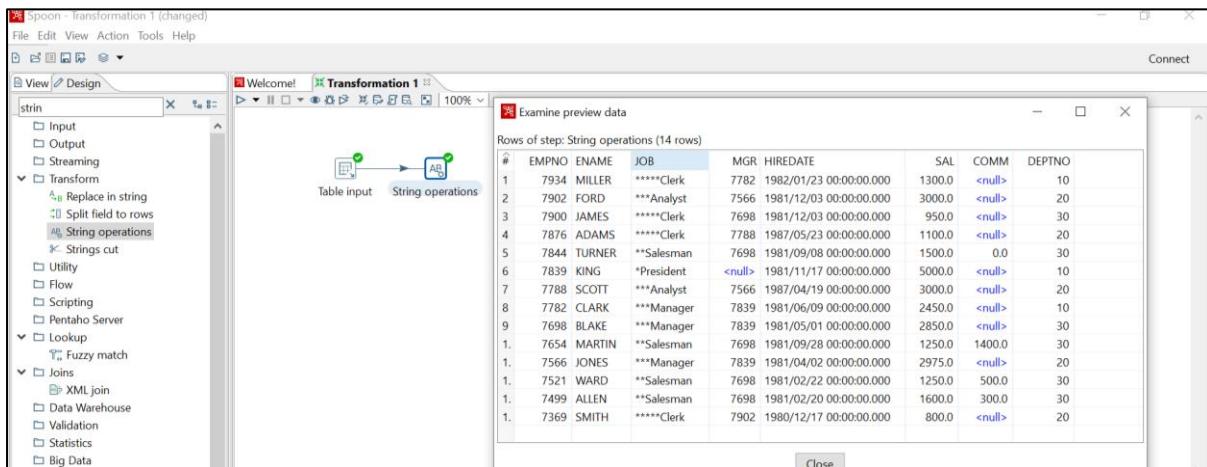
Design->Transform->String Operations(Drag / Drop in Transformation)->Create Hop Connection with String Operations (Click on Output Connector arrow in Input Table)

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



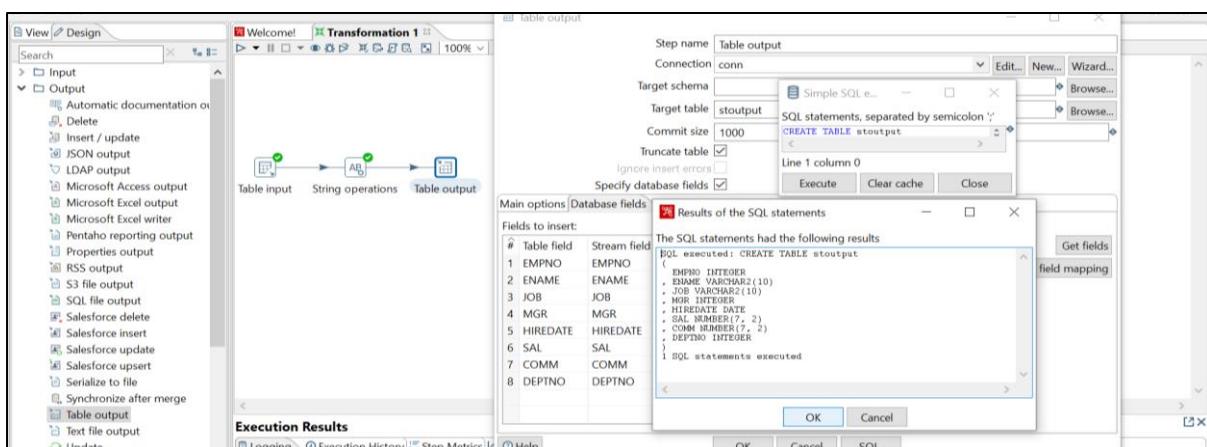
✓ Debug Transformation:

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



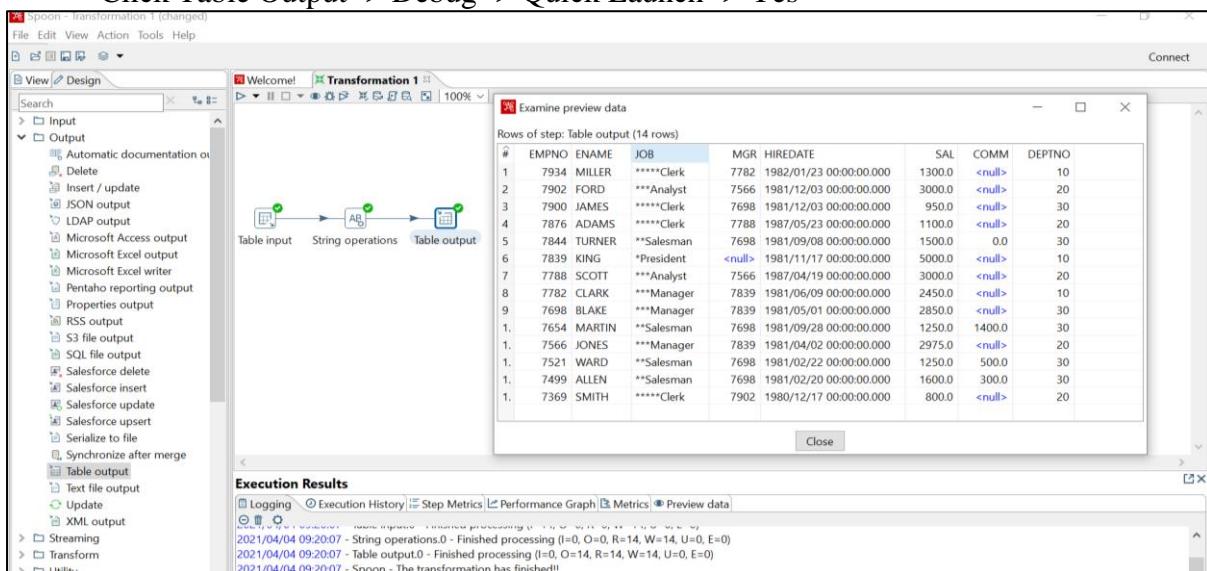
✓ Storing to Output Table:

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



✓ Debug Transformation:

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

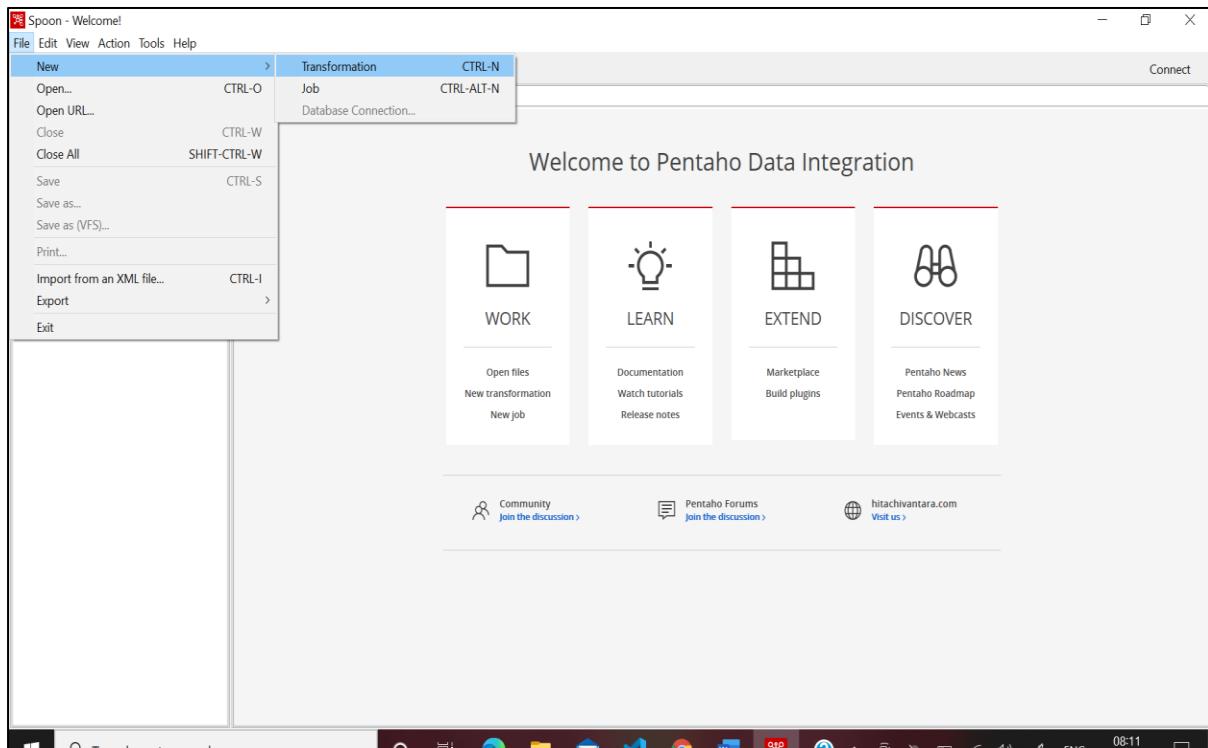


- Importing .CSV file to the target table.

### **Steps:**

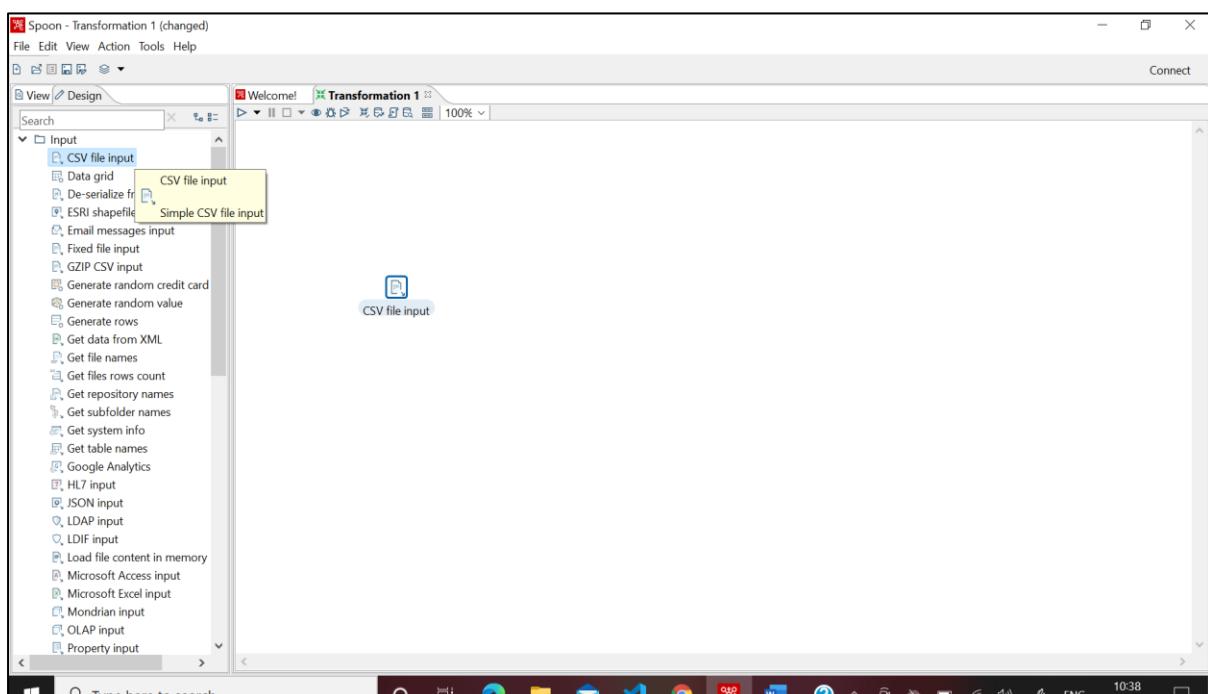
- ✓ Creating a New Transformation:

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



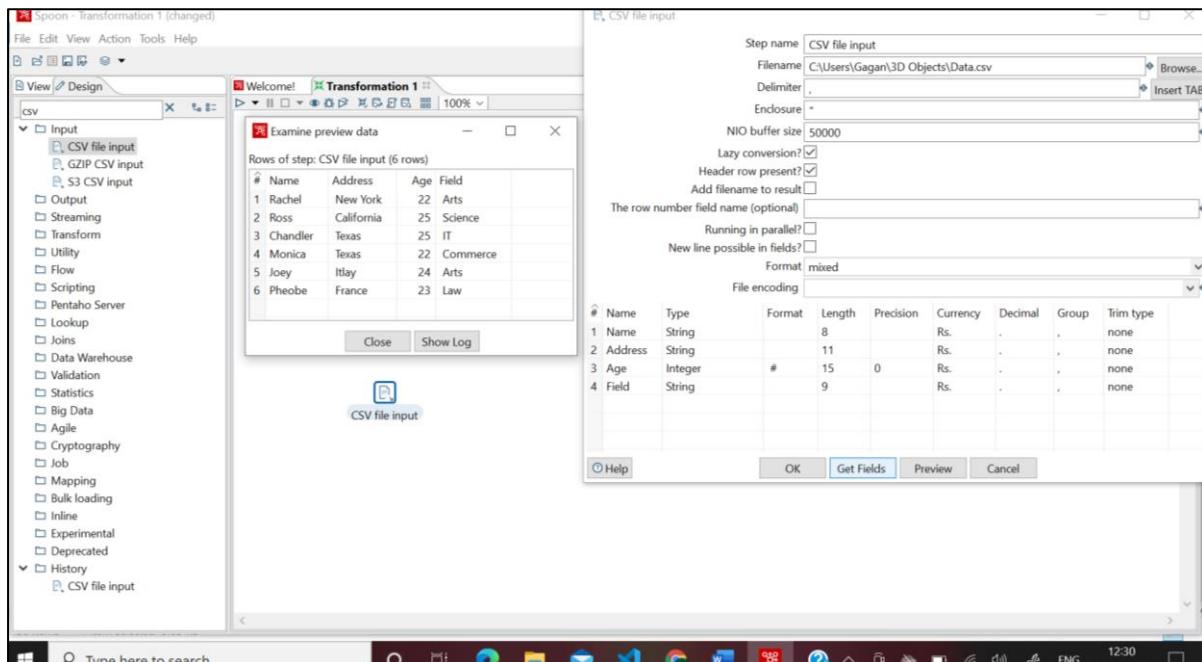
- ✓ Inserting CSV File Input into the Transformation:

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

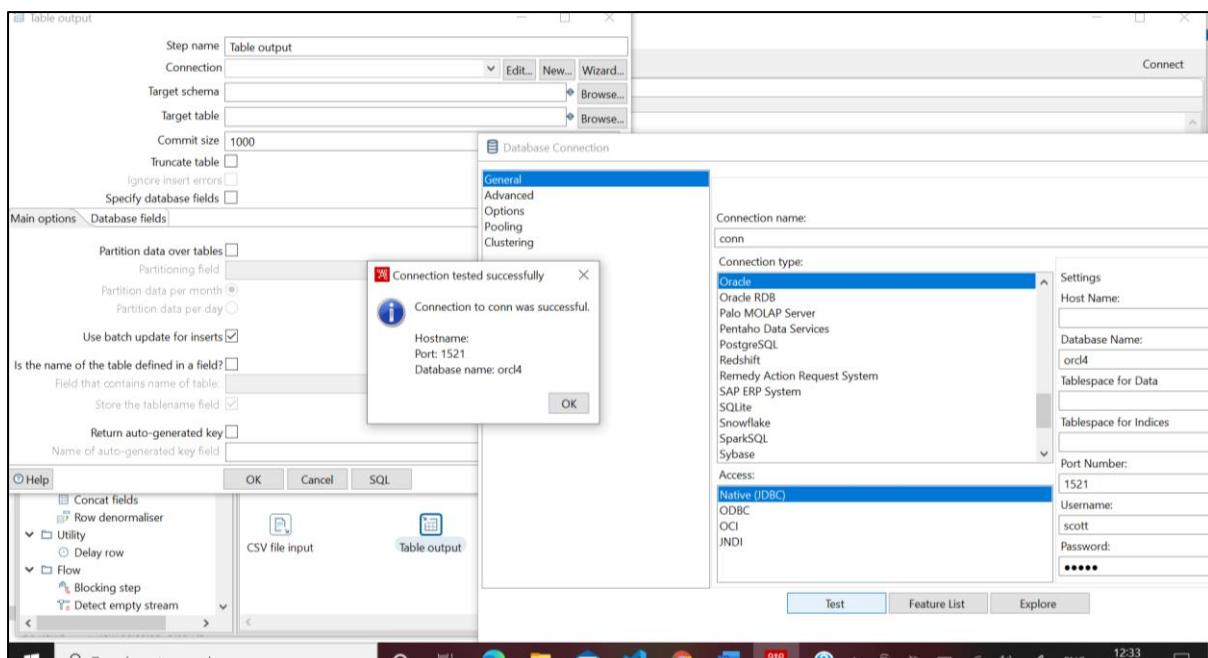
✓ Establishing Database Connection:

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

Connection name: *conn* Connection Type: *Oracle* Access: *Native (JDBC)*

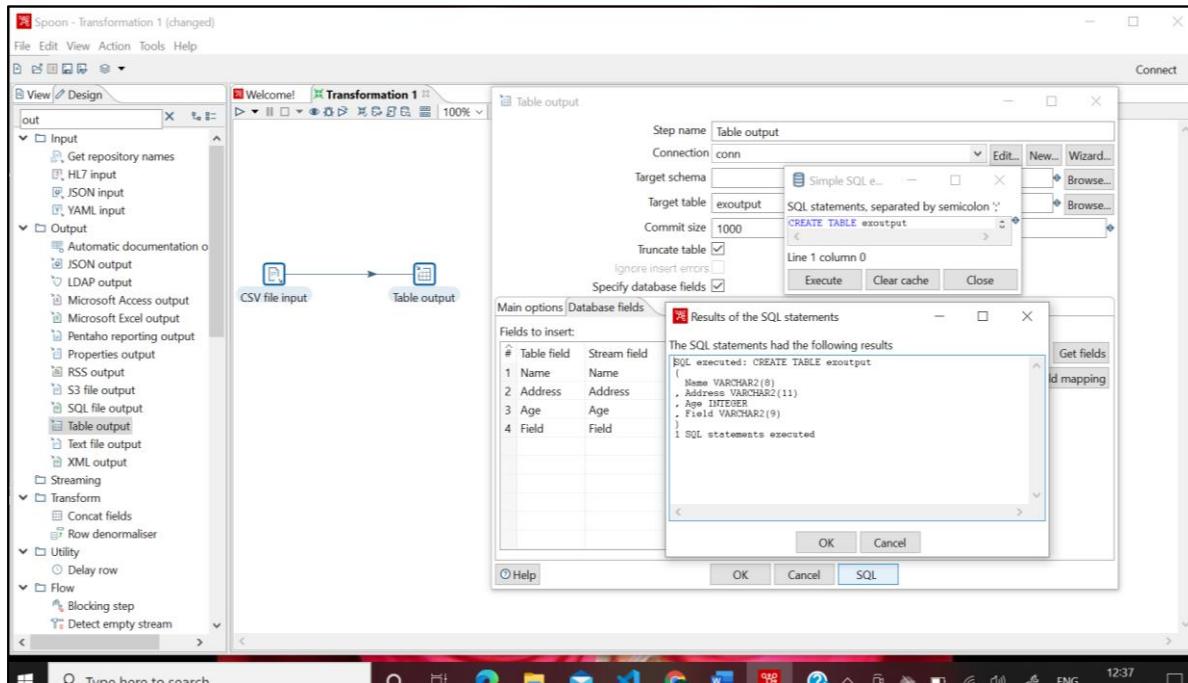
Database Name: *orcl4* Port Number: *1521* Username: *scott* Password: *tiger*

Click Test -> OK -> OK



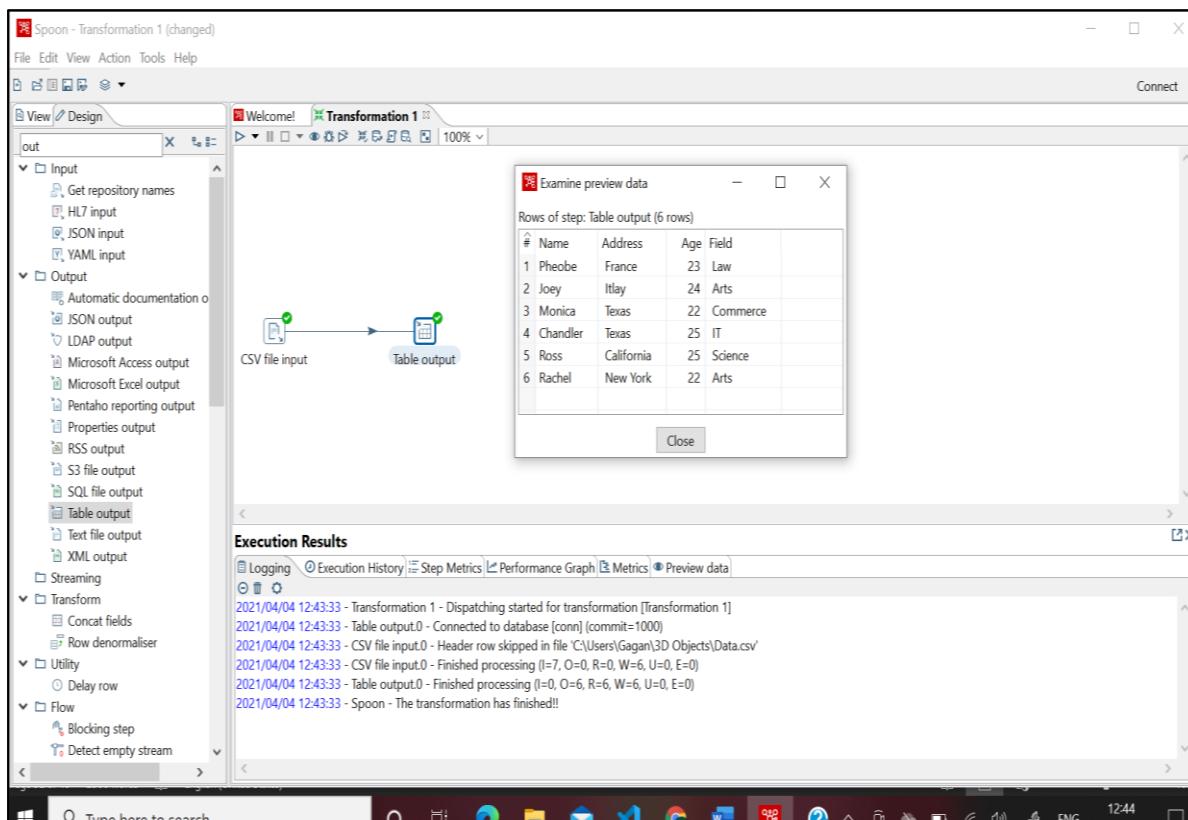
✓ Importing to Target Table:

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

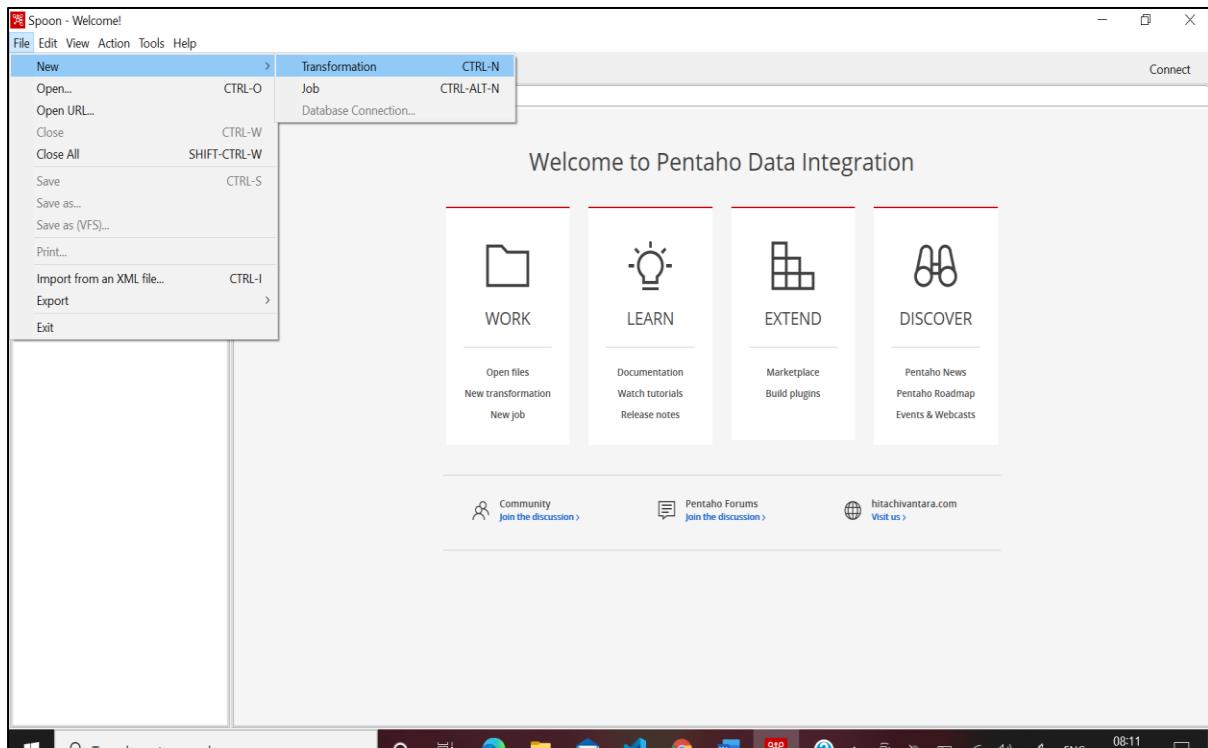


- Implement the Merge Join Transformation on Tables.

### **Steps:**

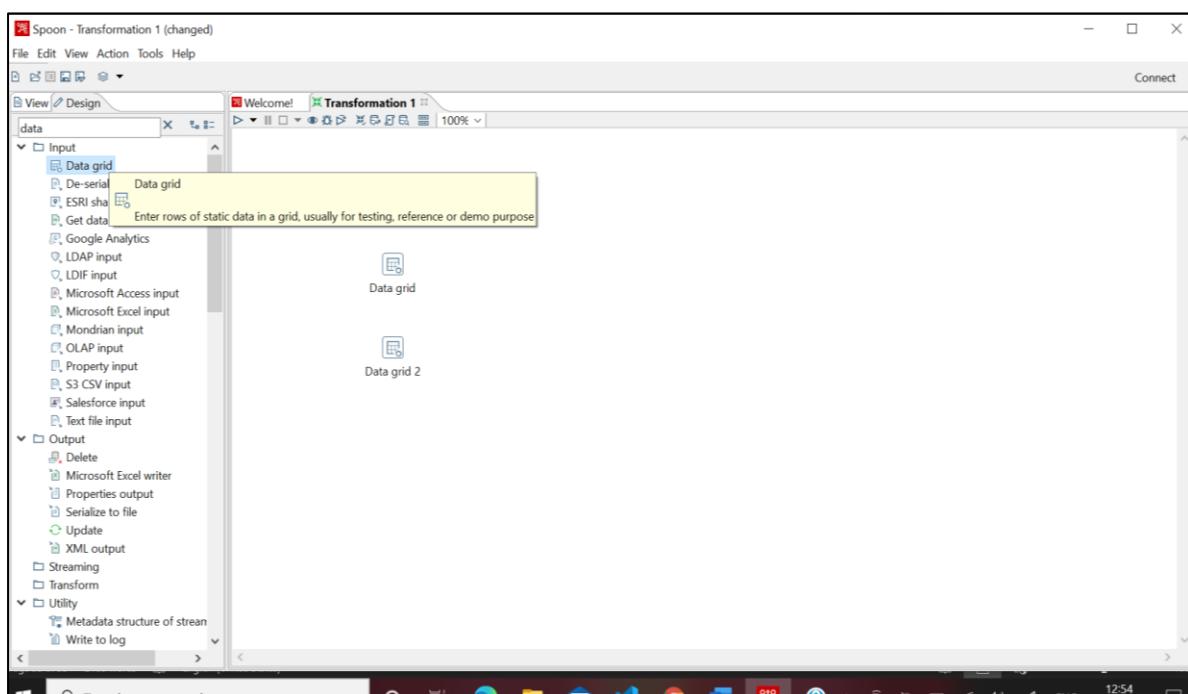
- ✓ Creating a New Transformation:

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

The top screenshot shows the configuration for the 'Employee' data grid. The 'Meta' tab displays fields: # Name, Type, Format, Length, Precision, Currency, Decimal, Group, Null if, and Set empty string? for rows 1 through 4. The 'Data' tab shows actual data for rows 1 through 4.

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Null if	Set empty string?
1	EmplID	Integer							N	N
2	EmpName	String							N	N
3	Age	Integer							N	N
4	DeptNo	Integer							N	N

The bottom screenshot shows the configuration for the 'Department' data grid. The 'Meta' tab displays fields: # Name, Type, Format, Length, Precision, Currency, Decimal, Group, Null if, and Set empty string? for rows 1 and 2. The 'Data' tab shows actual data for rows 1 through 3.

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Null if	Set empty string?
1	DeptNo	Integer							N	N
2	DeptName	String							N	N

✓ 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

The left screenshot shows the 'Employee' Sort rows step configuration. The 'Fields' section lists DeptNo as the fieldname, Ascending, Case sensitive compare? (N), Sort based on current locale? (Y), Collator Strength (0), and Presorted? (N).

Fieldname	Ascending	Case sensitive compare?	Sort based on current locale?	Collator Strength	Presorted?
DeptNo	Y	N	Y	0	N

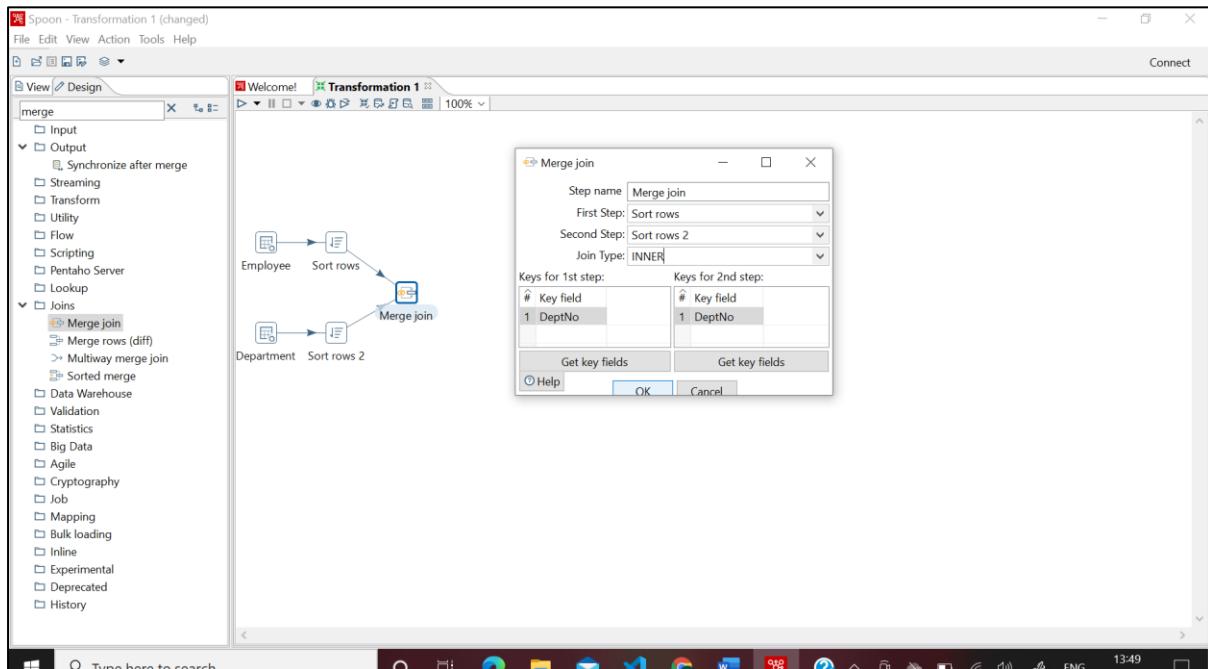
The right screenshot shows the 'Department' Sort rows step configuration, identical to the 'Employee' one.

Fieldname	Ascending	Case sensitive compare?	Sort based on current locale?	Collator Strength	Presorted?
DeptNo	Y	N	Y	0	N

- ✓ Inserting **Merge Join** into the Transformation:

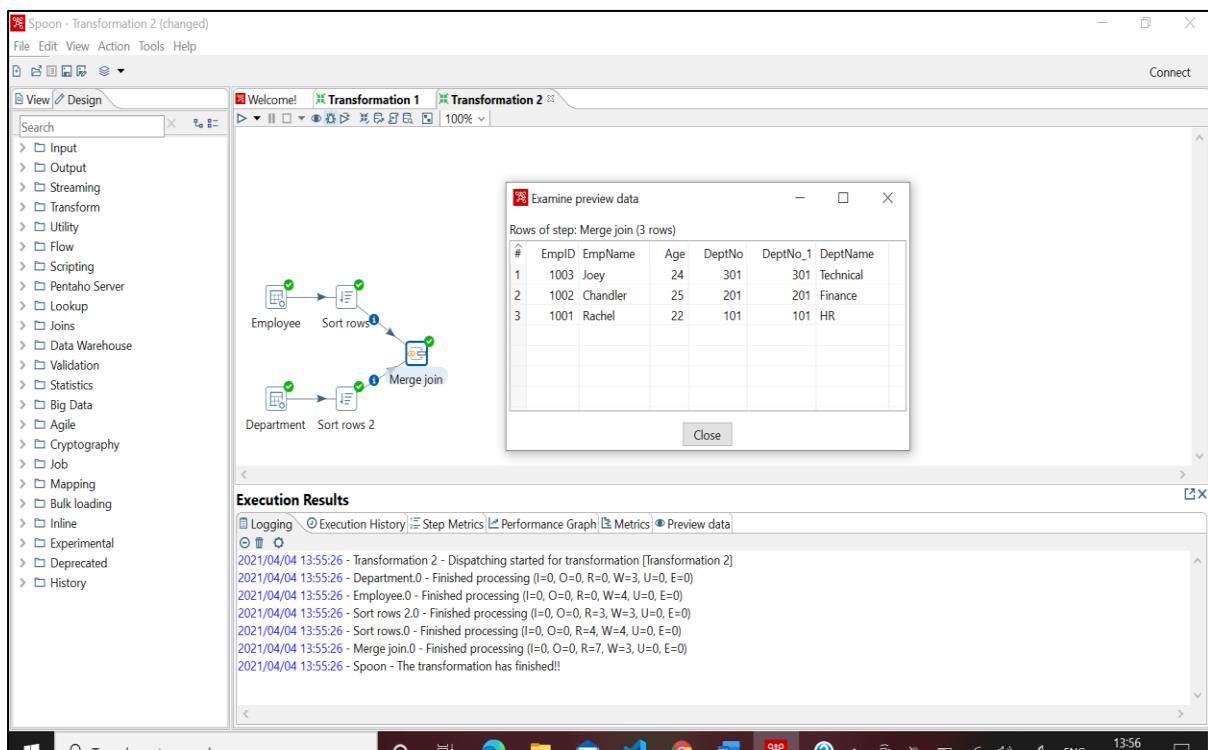
Design -> Joins -> Merge Join (Drag & Drop in Transformation) -> Create Hop Connection with Merge Join (Click on Output Connector arrow in Sort Rows)

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

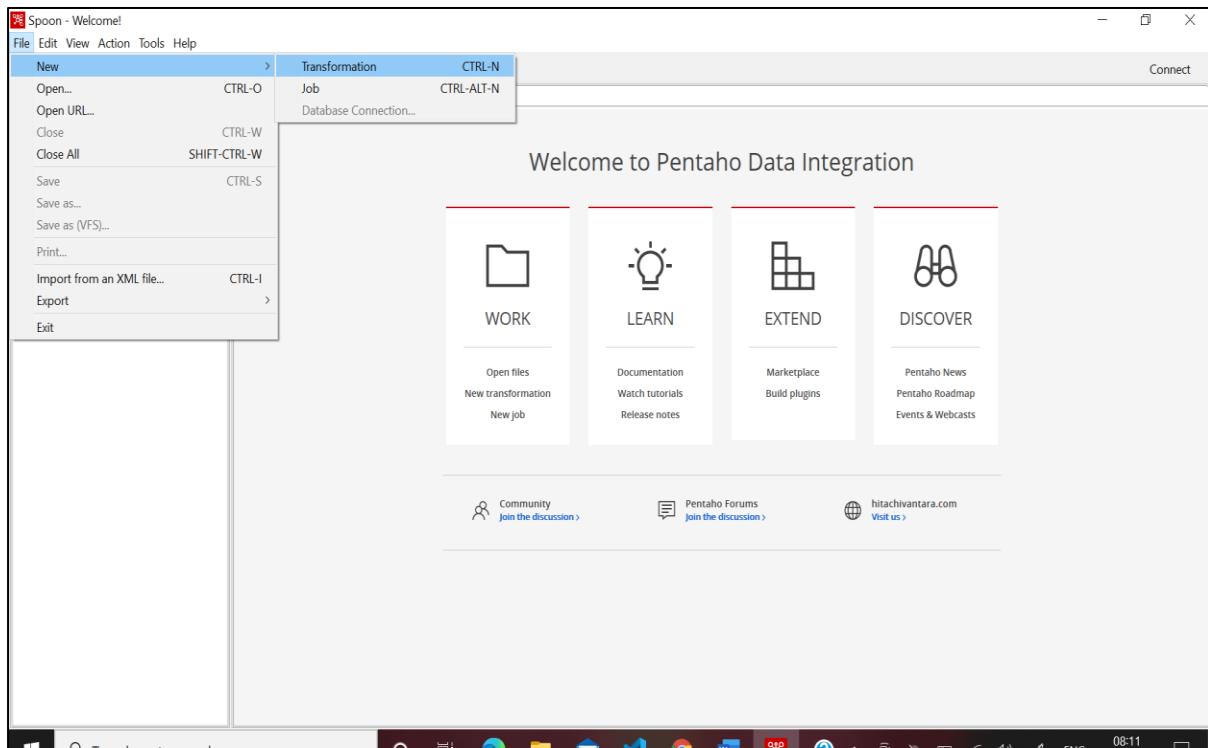


- Implement different Data Validations on Table.

### **Steps:**

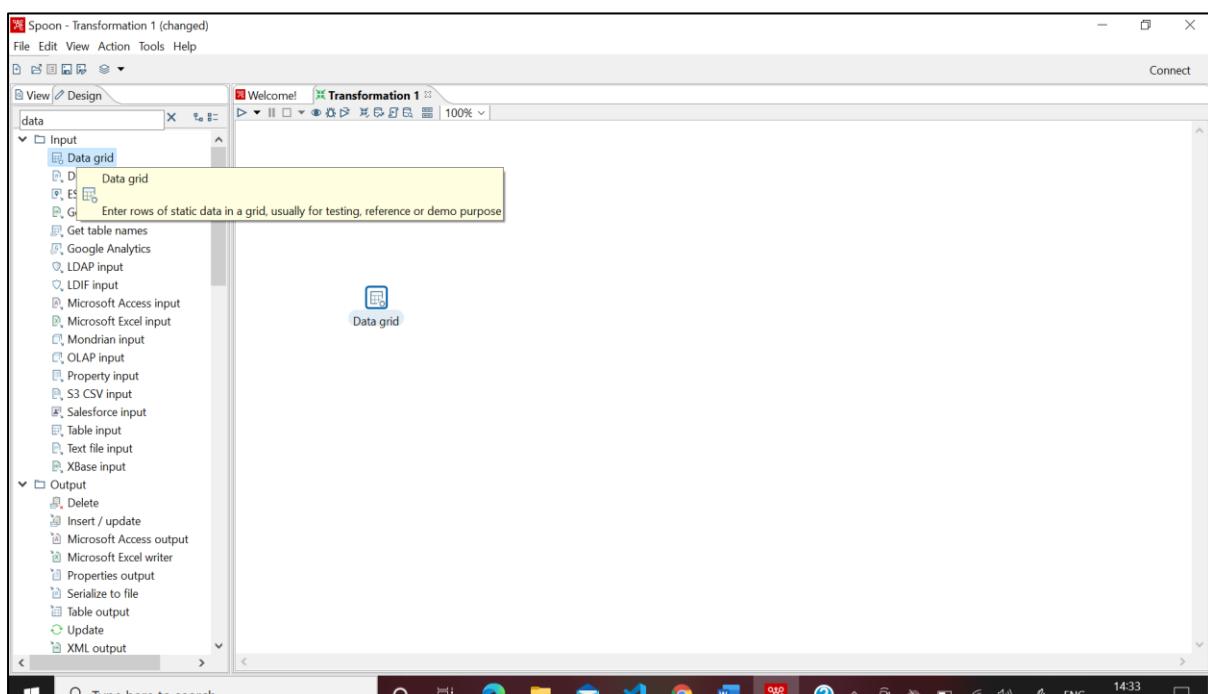
- ✓ Creating a New Transformation:

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



- ✓ Inserting Data Grid into the Transformation:

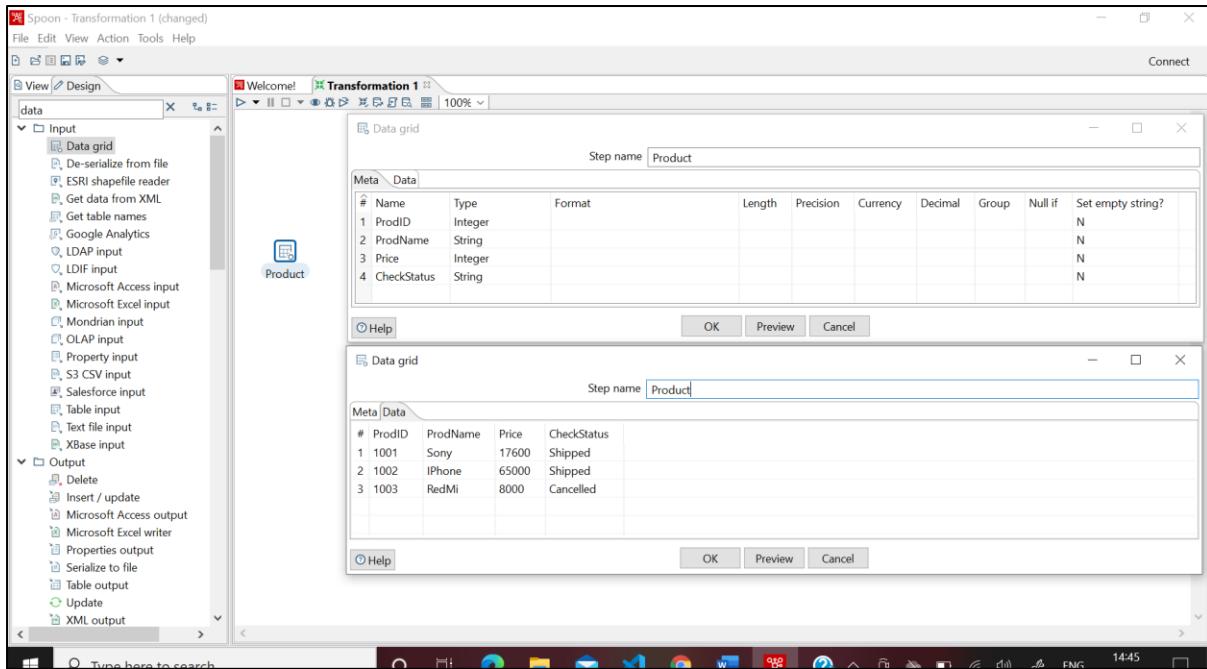
Design -> Input -> Data Grid (Drag & Drop in Transformation)



✓ Inserting Data in Data Grid:

Double Click Data Grid -> Step name : *Product* ->

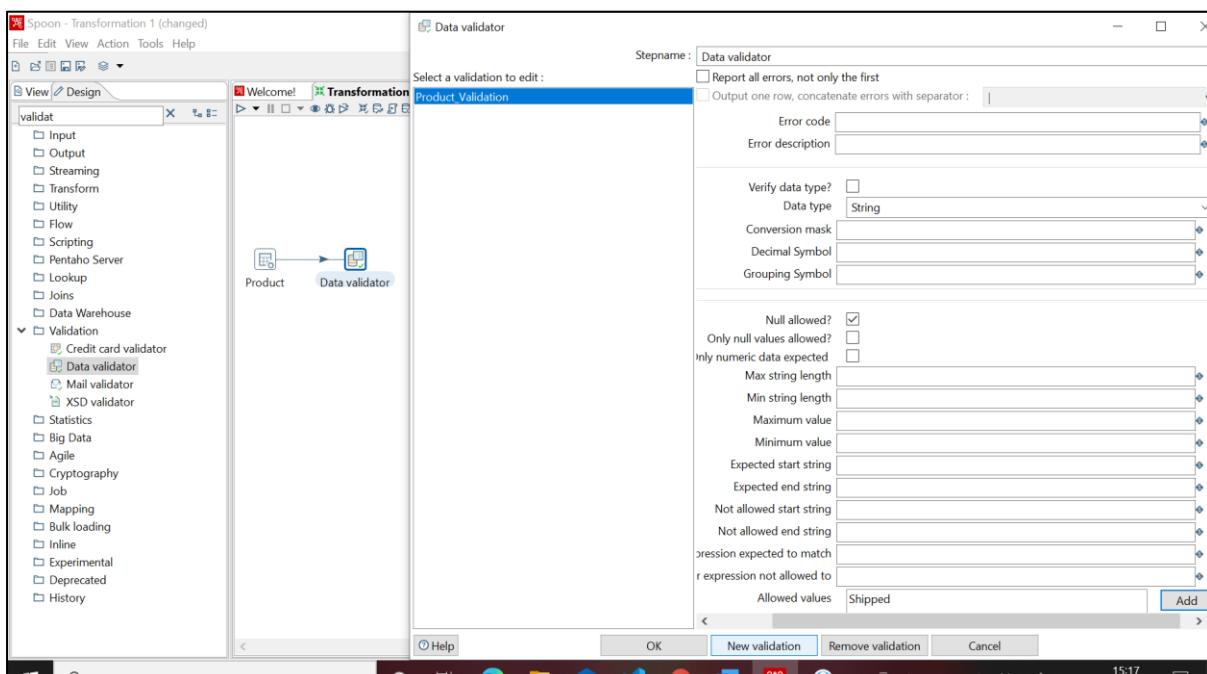
Meta: *Enter Fields Details* -> Data: *Enter actual Data* -> OK



✓ Inserting Data Validator into the Transformation:

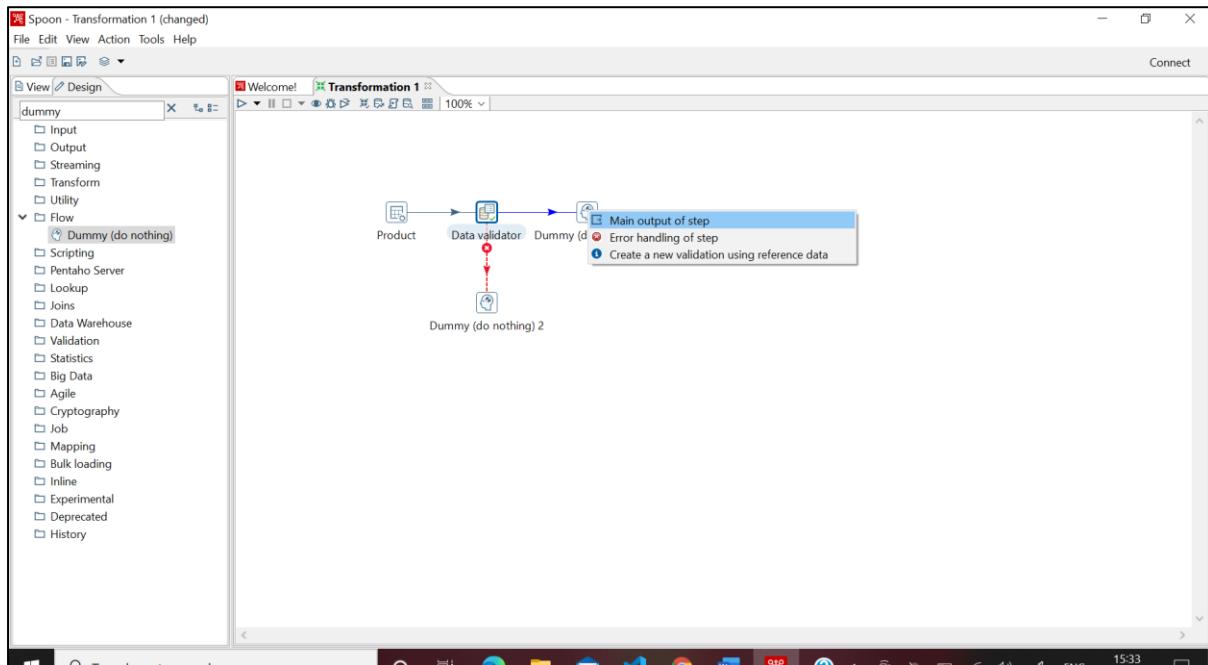
Design -> Validation -> Data Validator (Drag & Drop in Transformation) -> Create Hop Connection with Data Validator (Click on Output Connector arrow in Data Grid)

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



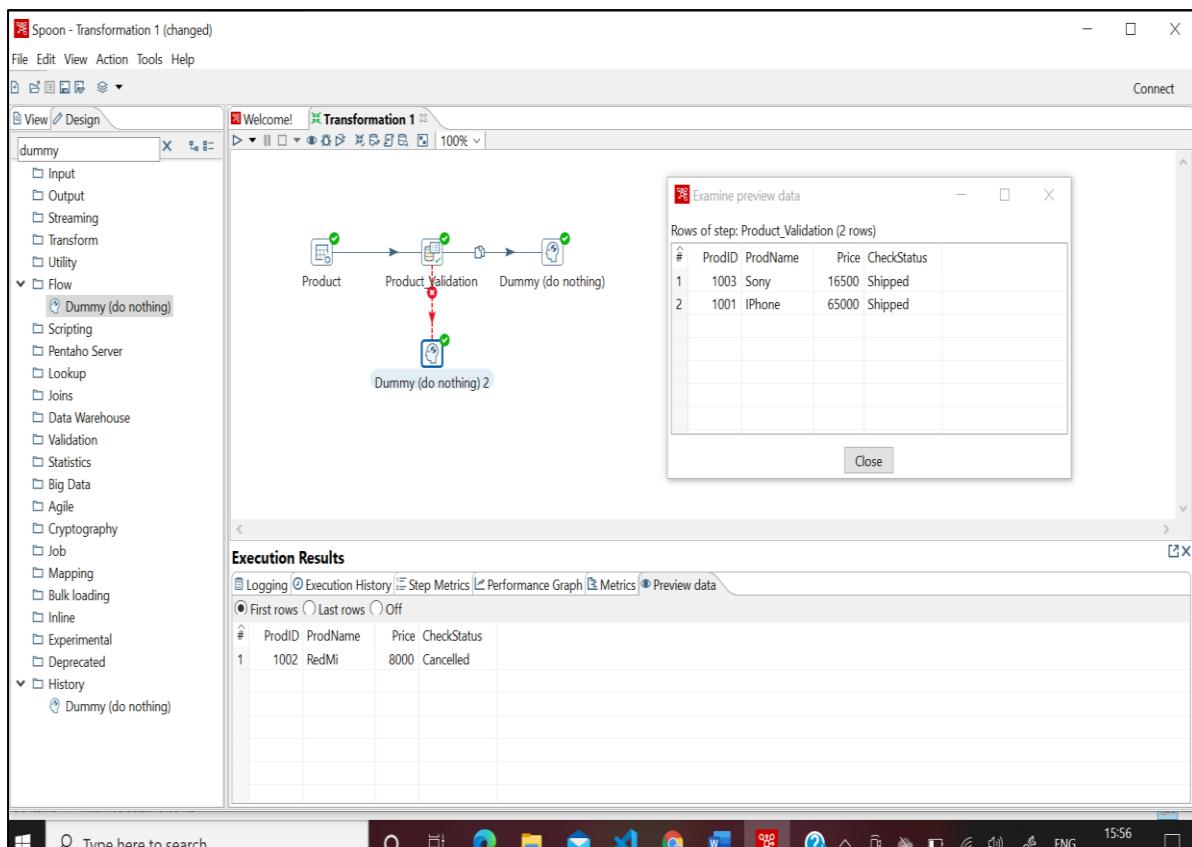
- ✓ Inserting **Dummy** (do nothing) into the Transformation:

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")
```

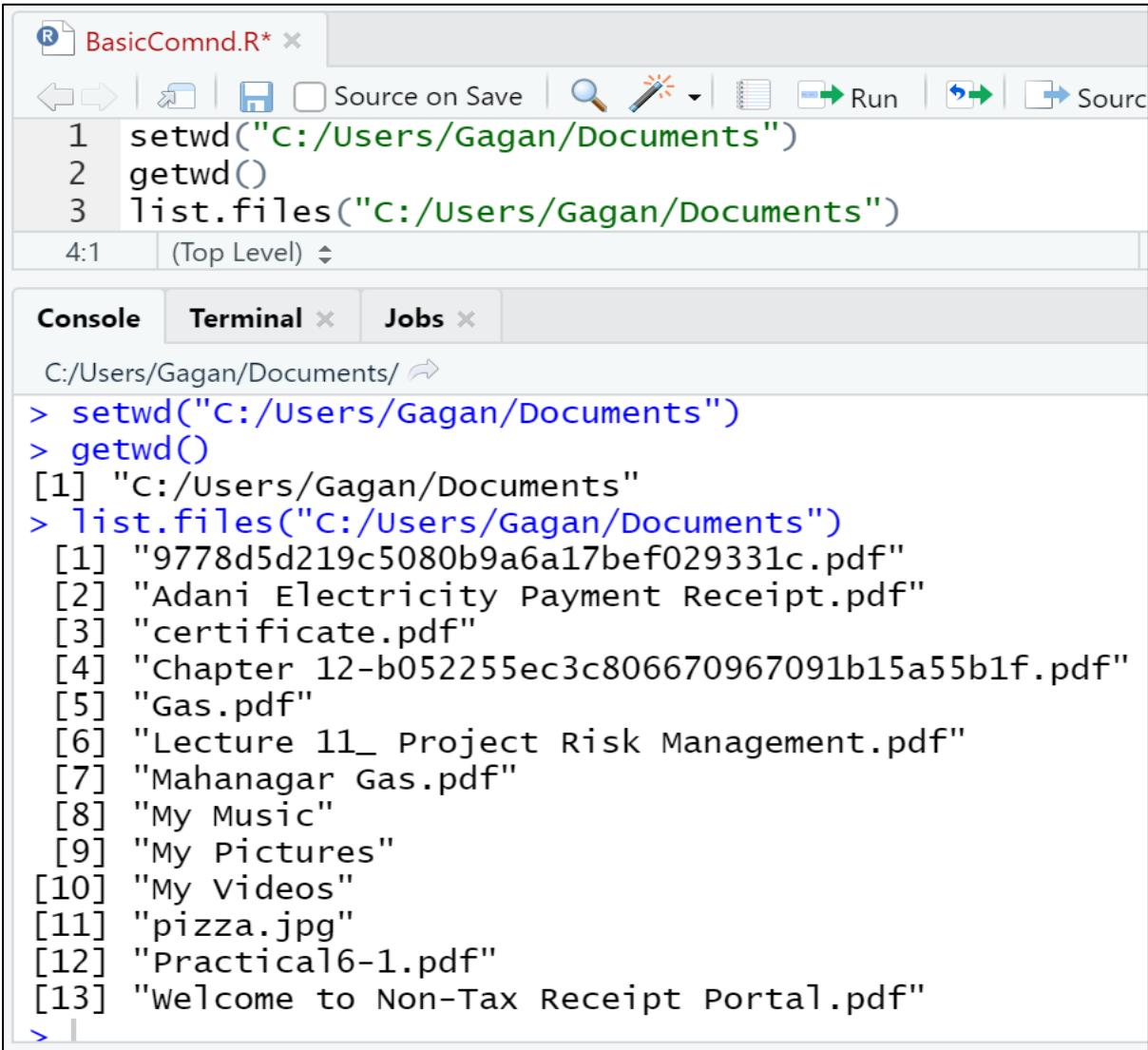
- ✓ Getting Current Working Directory

```
getwd()
```

- ✓ Listing Contents of Current Working Directory

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

#### Output:



The screenshot shows the RStudio interface with the following details:

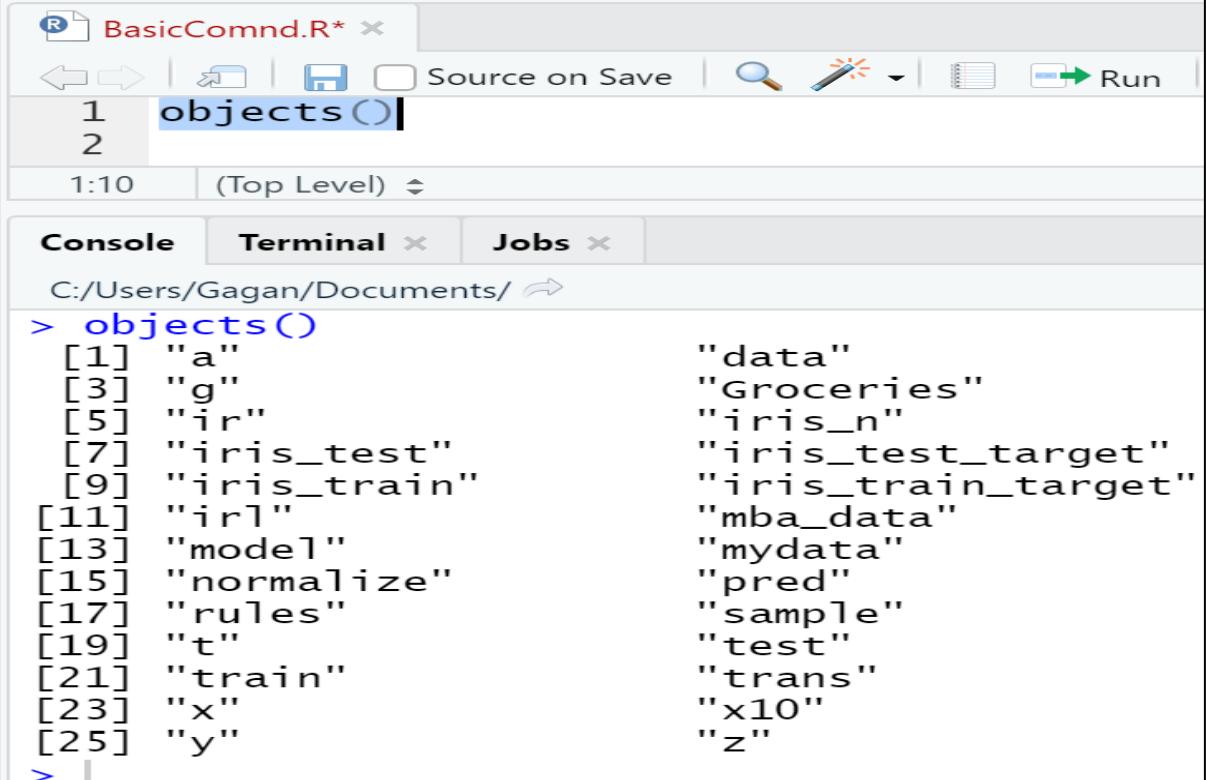
- Code Editor:** The file is named "BasicComnd.R\*". The code contains three lines of R code: `setwd("C:/Users/Gagan/Documents")`, `getwd()`, and `list.files("C:/Users/Gagan/Documents")`.
- Console Output:**
  - Shows the current working directory as `C:/Users/Gagan/Documents`.
  - Execution of the R code results in the following output:
    - `> setwd("C:/Users/Gagan/Documents")`
    - `> getwd()`
    - `[1] "C:/Users/Gagan/Documents"`
    - `> list.files("C:/Users/Gagan/Documents")`
    - Output list:
      - [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] "Practical6-1.pdf"
      - [13] "Welcome to Non-Tax Receipt Portal.pdf"

- 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 it, the code editor shows two lines of R code: "1 objects()" and "2". The status bar indicates "1:10 (Top Level)". The main area is the "Console" tab, which displays the output of the "objects()" command. The output lists numerous objects in the environment, including "a", "g", "ir", "iris\_test", "iris\_train", "irl", "model", "normalize", "rules", "t", "train", "x", "y", and various datasets like "data", "Groceries", "iris\_n", "iris\_test\_target", "iris\_train\_target", "mba\_data", "mydata", "pred", "sample", "test", "trans", "x10", and "z".

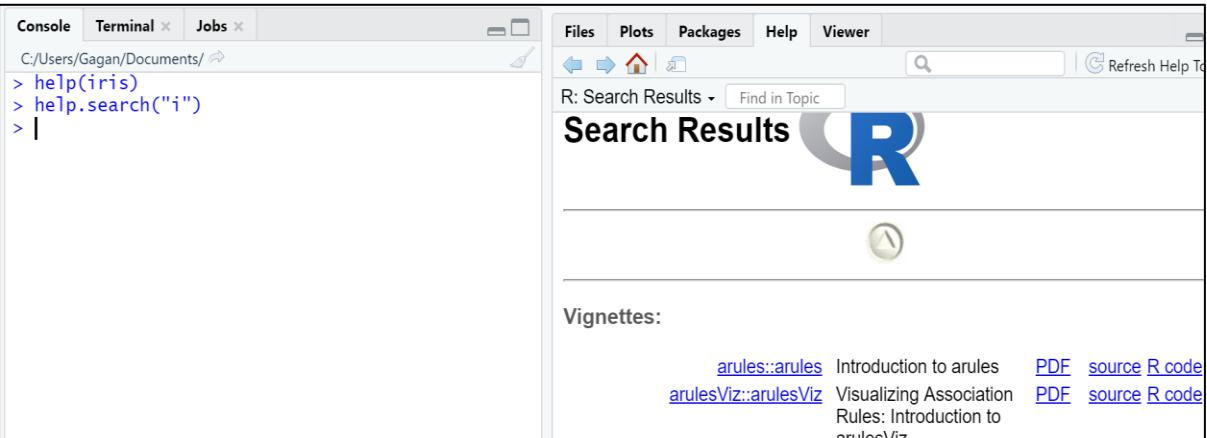
```
> objects()
[1] "a"                      "data"
[3] "g"                      "Groceries"
[5] "ir"                     "iris_n"
[7] "iris_test"               "iris_test_target"
[9] "iris_train"              "iris_train_target"
[11] "irl"                    "mba_data"
[13] "model"                  "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 left pane, the "Console" tab shows the commands `help(iris)` and `help.search("i")`. The right pane displays the "Search Results" for "Search Results". It features a large blue "R" logo at the top. Below it, there's a section titled "Vignettes:" with links to "arules::arules" and "arulesViz::arulesViz". Each link has a "PDF" and "source R code" option.

Files Plots Packages Help Viewer

R: Search Results Find in Topic

## Search Results

Vignettes:

<a href="#">arules::arules</a>	Introduction to arules	<a href="#">PDF</a>	<a href="#">source R code</a>
<a href="#">arulesViz::arulesViz</a>	Visualizing Association Rules: Introduction to arulesViz	<a href="#">PDF</a>	<a href="#">source R code</a>

- 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:**

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

4:1 (Top Level) R Script

Console Terminal x Jobs x
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 MB)
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:**

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

Console Terminal x Jobs x
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 corner, there's a file icon and the text "BasicComnd.R\*". Below the editor area, there are tabs for "Console", "Terminal", and "Jobs". The "Console" tab is active, showing the path "C:/Users/Gagan/Documents/" followed by the command line input and output:  
 > 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 corner, there's a file icon and the text "BasicComnd.R\*". Below the editor area, there are tabs for "Console", "Terminal", and "Jobs". The "Console" tab is active, showing the path "C:/Users/Gagan/Documents/" followed by the command line input and output:  
 > 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:**

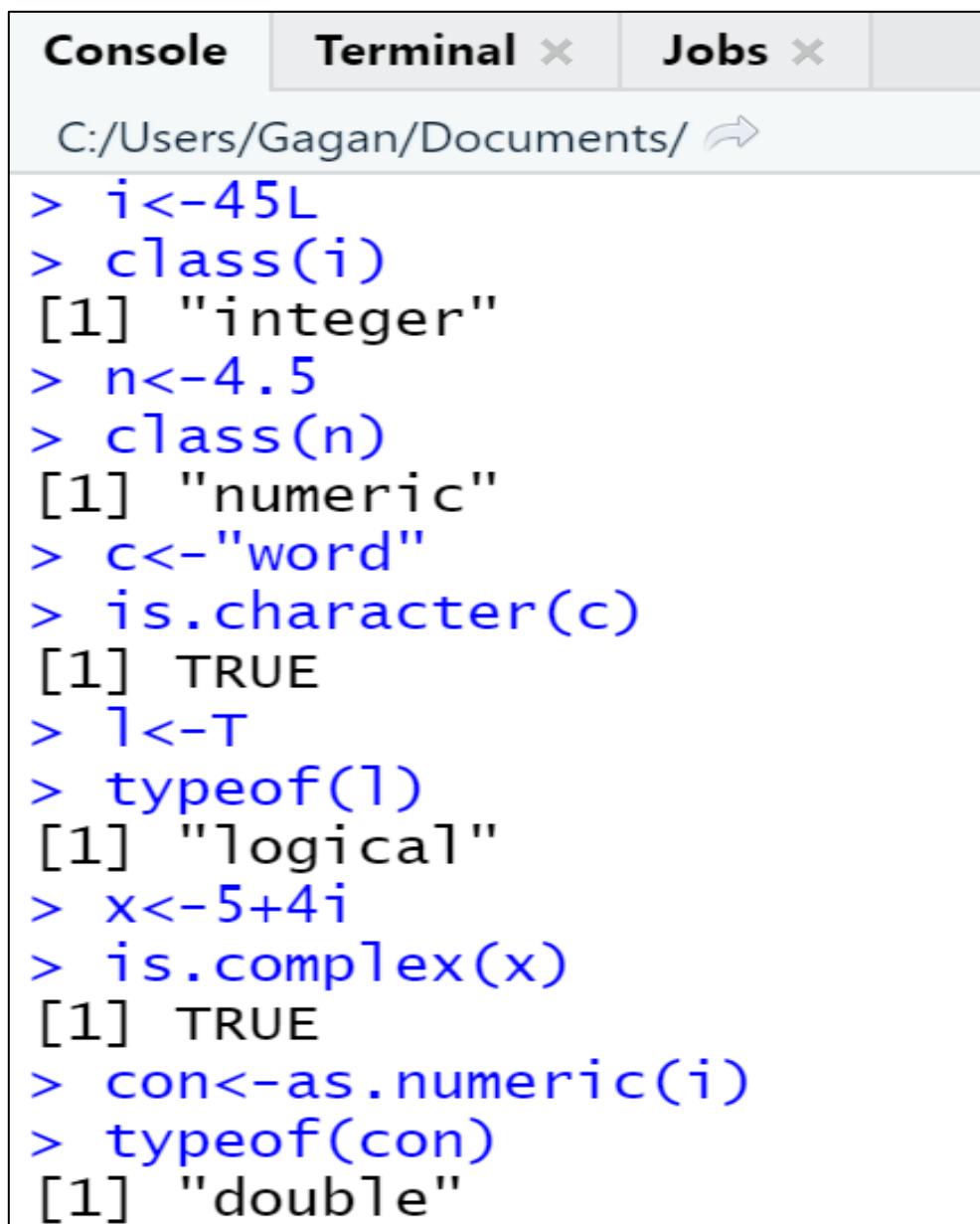
```
C:/Users/Gagan/Documents/ ➔
> 45+28
[1] 73
> s<-45-28
> s
[1] 17
> 4*5
[1] 20
> 8/2
[1] 4
> sqrt(64)
[1] 8
> 8^2
[1] 64
> log(2)
[1] 0.6931472
> exp(8)
[1] 2980.958
> factorial(5)
[1] 120
```

- 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	l<-T	#Logical
class(i)	#Checking Type	typeof(l)	#CheckingType
n<-4.5	#Numeric	x<-5+4i	#Complex
c<-"word"	#Character	con<-as.numeric(i)	#Converting Type
is.character(c)	#Checking Type		

### Output:



The screenshot shows the RStudio interface with the 'Console' tab selected. The working directory is 'C:/Users/Gagan/Documents/'. The console window displays the following R session:

```

> 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
m2    56   67   78
```

- ✓ 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")
  print(list)
  print(list[2])
```

#Creating List  
#Accessing List  
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] "apple"      "cherry"    "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)
```

	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  Italy
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
1                                         PISTON RINGS
2 CAPACITY CONTROL VALVE ASSEMBLY (FOR 5H*
  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    Raj  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  Astronaut
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 5H*	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  
head(mtcars,5)

```
> 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  
colnames(my\_data)

```
> 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        HR
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
```

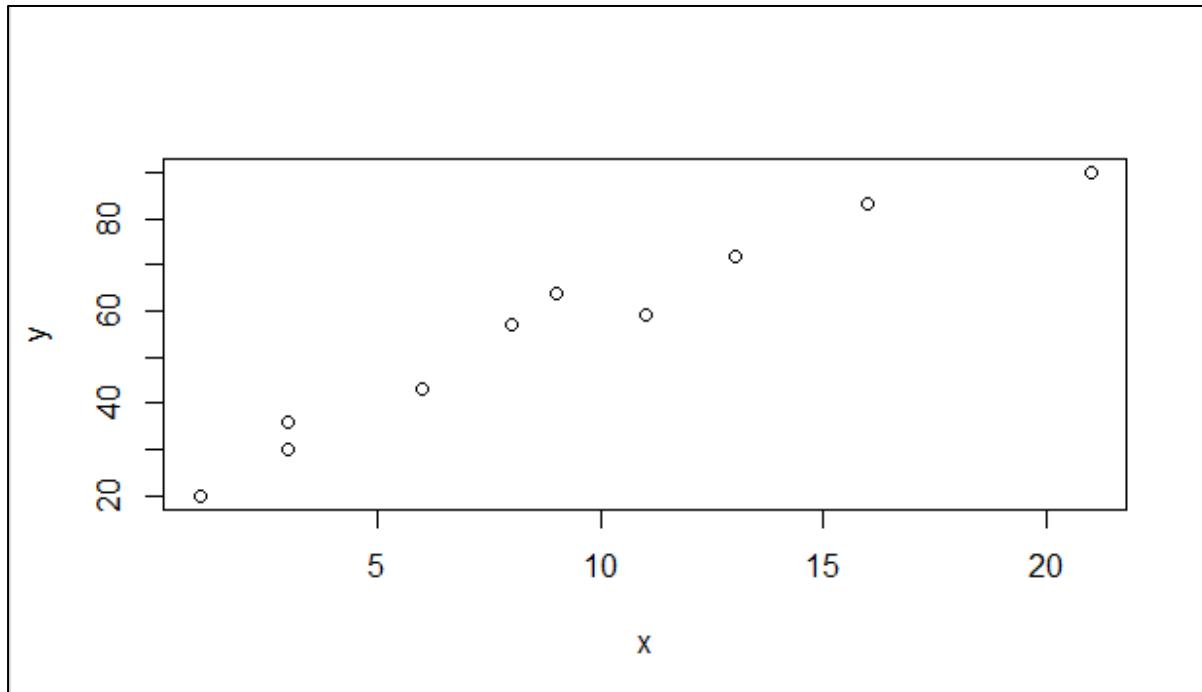
- ✓ `Na.omit(x)`

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

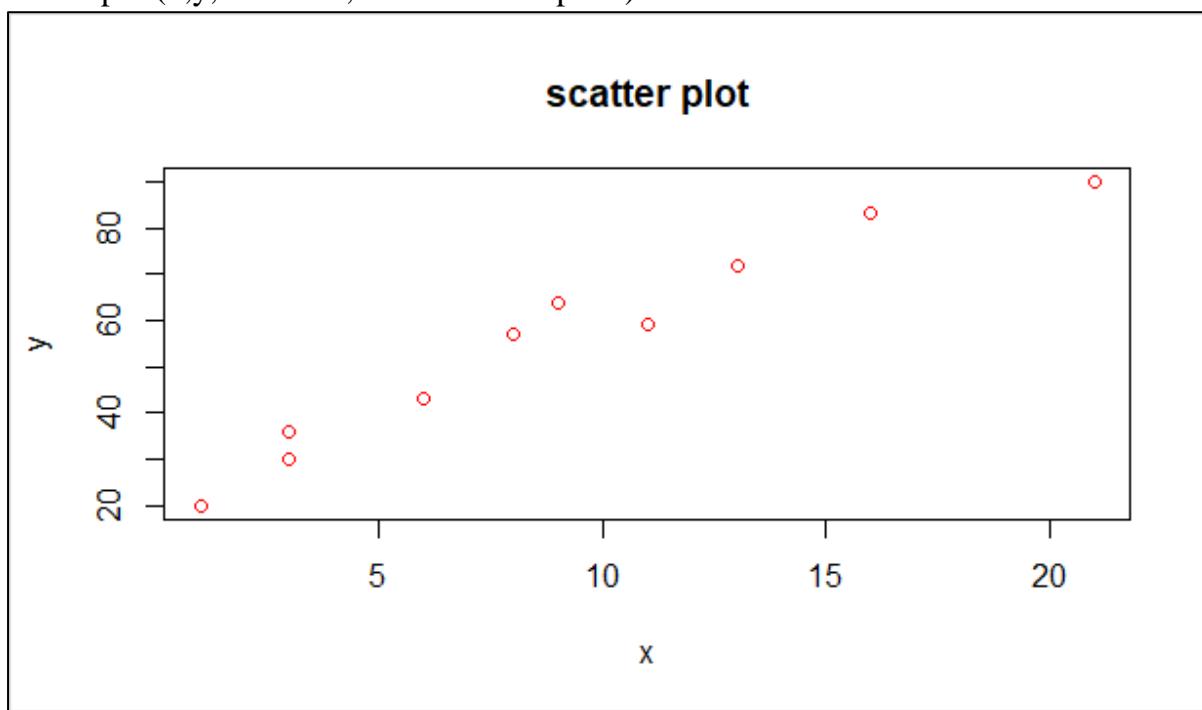
### Practical No. 07

**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")`



## ➤ Linear Regression Model

- ✓ model=lm(y~x)
- model

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

call:
lm(formula = y ~ x)

Coefficients:
(Intercept)           x
23.209             3.537
```

- ✓ 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" "call"

$class
[1] "lm"

> coef(model)
(Intercept)           x
23.208972            3.537476
> residuals(model)
     1      2      3      4      5      6      7      8
-3.821399  5.491223  8.953748  2.803845  2.178601 -1.433826 -3.121204 -7.495960
     9     10
-6.746447  3.191418
```

- ✓ summary(model)

```
> summary(model)

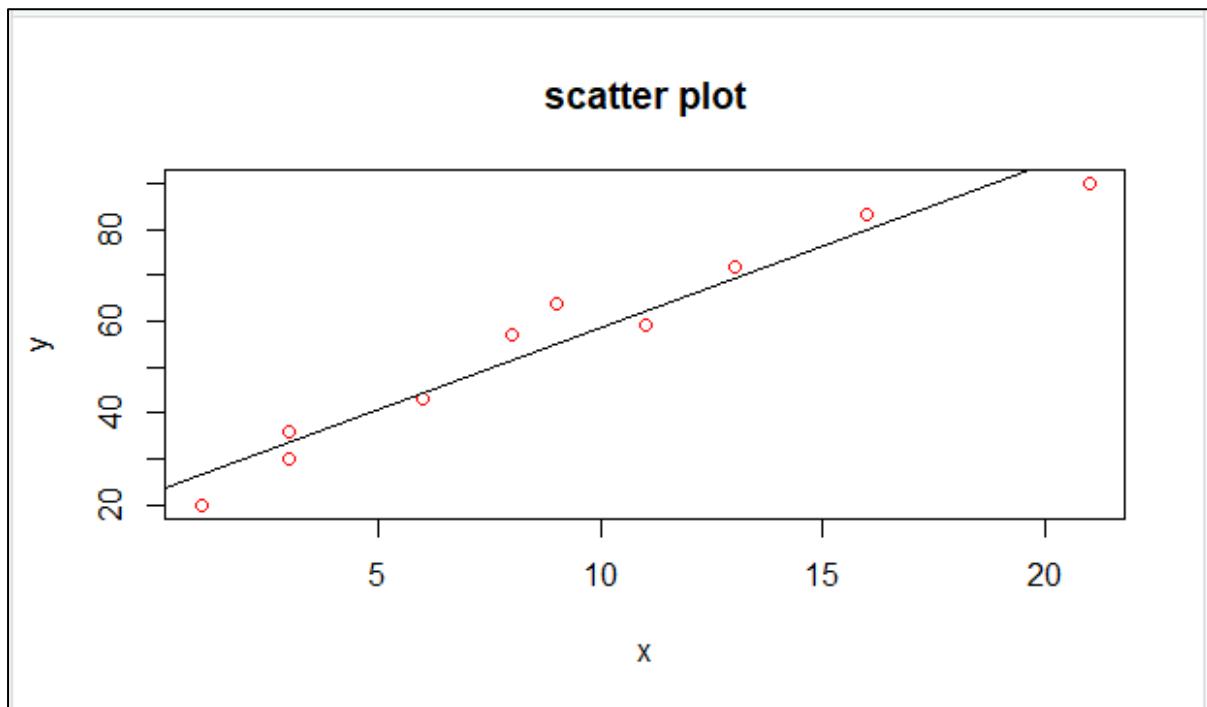
call:
lm(formula = y ~ x)

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  $y=a+bx$

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

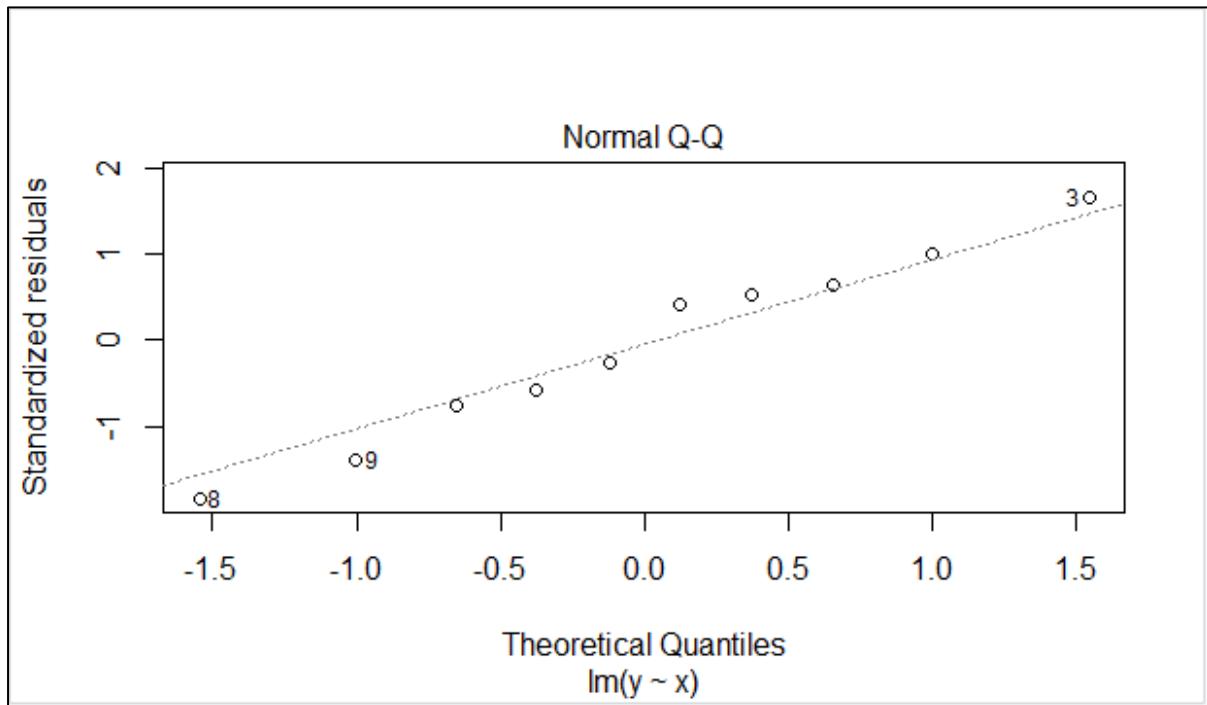
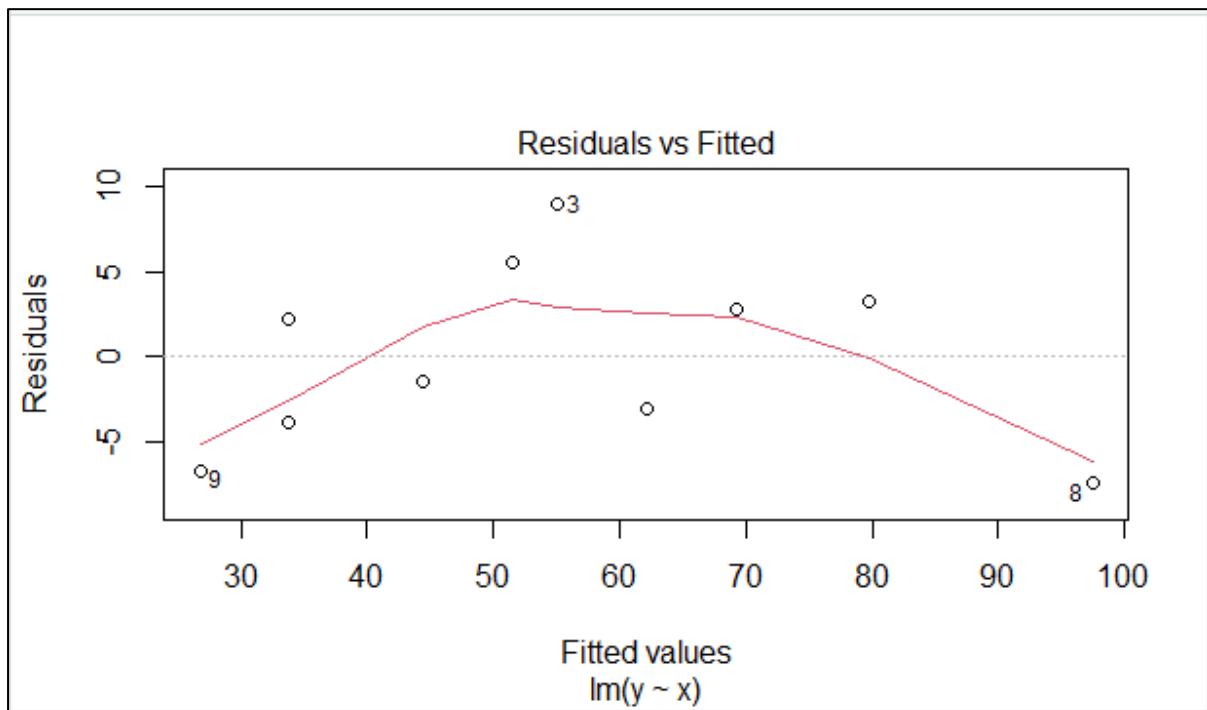
```
> #predicting values manually y=a+bx
> x10<-model$coefficients[[1]]+model$coefficients[[2]]*10
> x10
[1] 58.58373
```

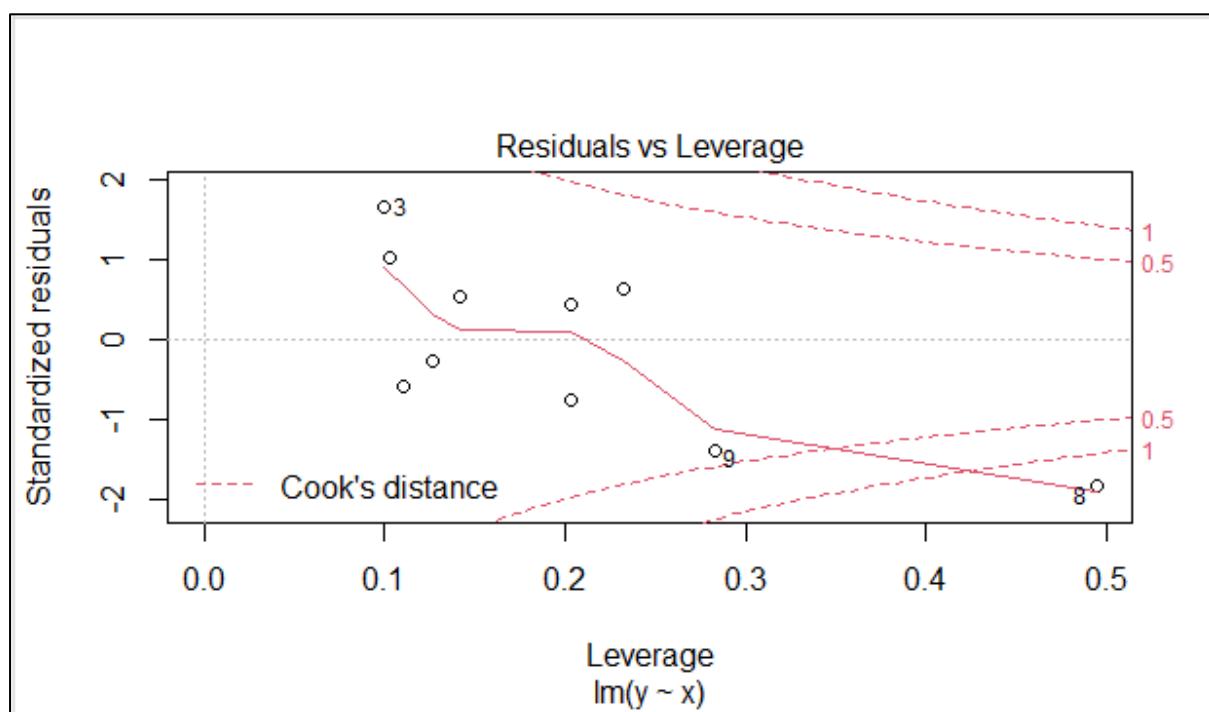
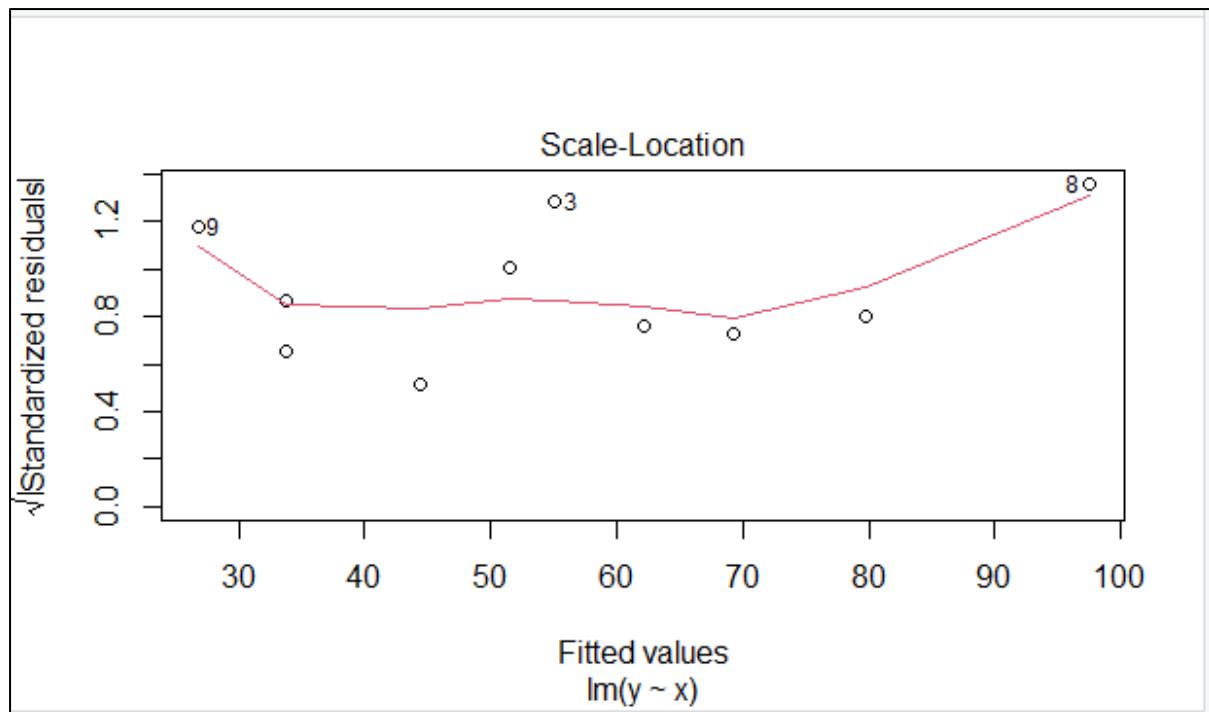
➤ Using Predict ()

✓  $a <- \text{data.frame}(x=10)$   
 $a$   
 $\text{pred} <- \text{predict}(\text{model}, a)$   
 $\text{pred}$

```
> #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

```
> 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
```

- ✓ 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
[33] virginica virginica virginica virginica virginica virginica virginica
[41] 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)  

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	4.8	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=naiveBayes(Species ~.,data=train)  
pred=predict(model,test)  
table(pred)`

```
> train=ir1[1:100,]
> test=ir1[101:150,]
> 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 virginica
[106] virginica virginica virginica virginica virginica virginica virginica
[113] virginica virginica virginica virginica virginica virginica virginica
[120] virginica 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)
iris_test_target setosa versicolor virginica
  setosa        0        0        0
  versicolor    0        0        0
  virginica    11       9        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 54 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

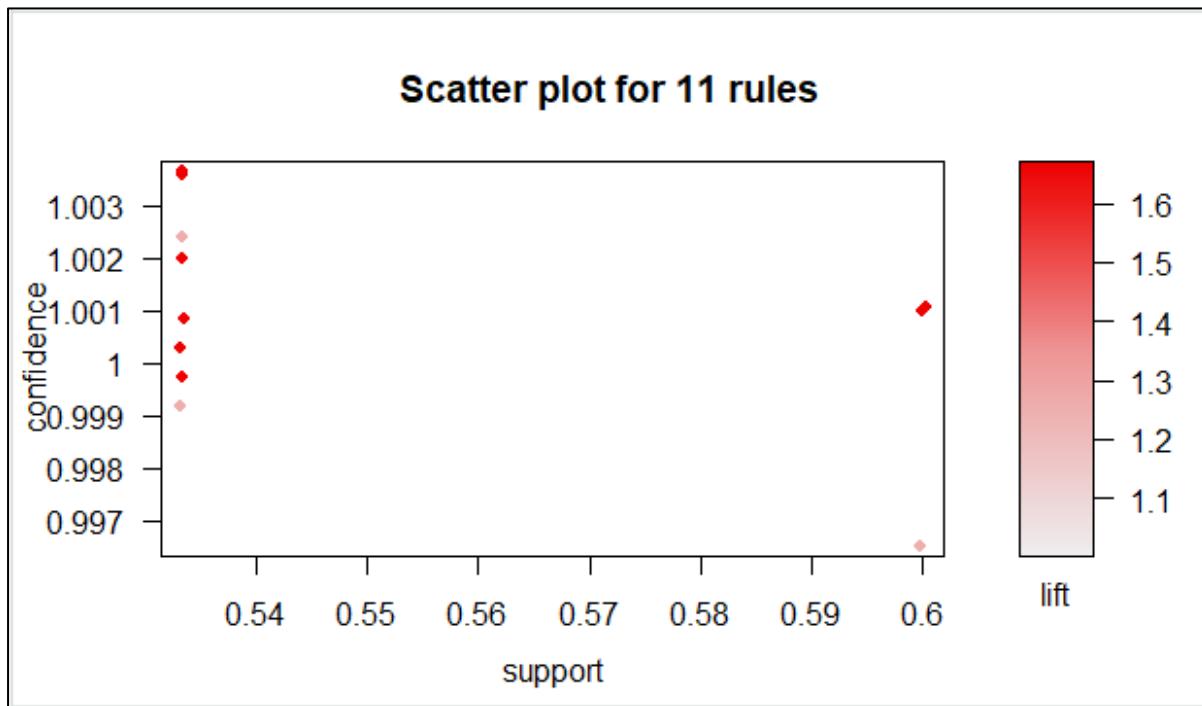
rule length distribution (lhs + rhs):sizes
2 3
3 8

   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,minlen=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 S4 object ... done [0.00s].
```

- ✓ rules

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

## **Practical No. 10**

**Aim:** Implementation and Analysis of Clustering Algorithms like K-Means, Agglomerative.

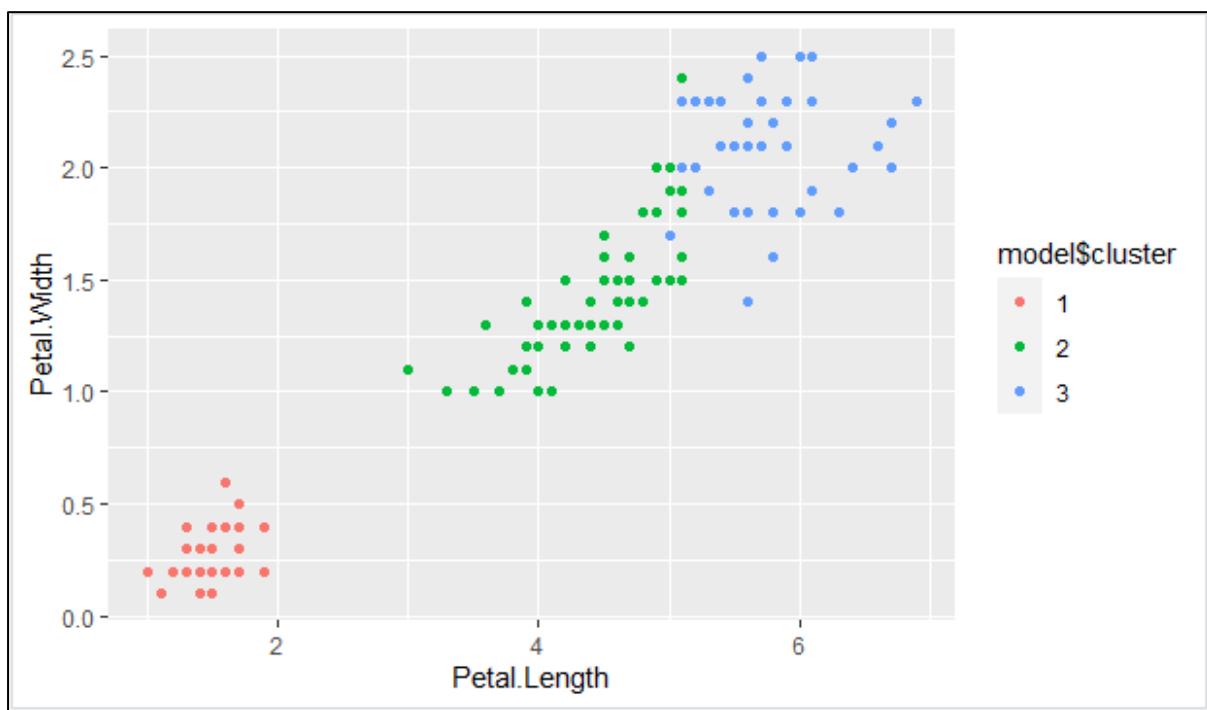
## ➤ K-Means

- ```
✓ mydata<-select(iris,c(1,2,3,4))
  model<-kmeans(mydata,3)
  model
```

- ✓ model\$size  
table(model\$cluster,iris\$Species)

```
> model$size  
[1] 50 62 38  
> table(model$cluster,iris$species)  
  
setosa versicolor virginica  
1      50          0          0  
2      0           48         14  
3      0           2          36
```

- ✓ 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
```

|               | 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 |

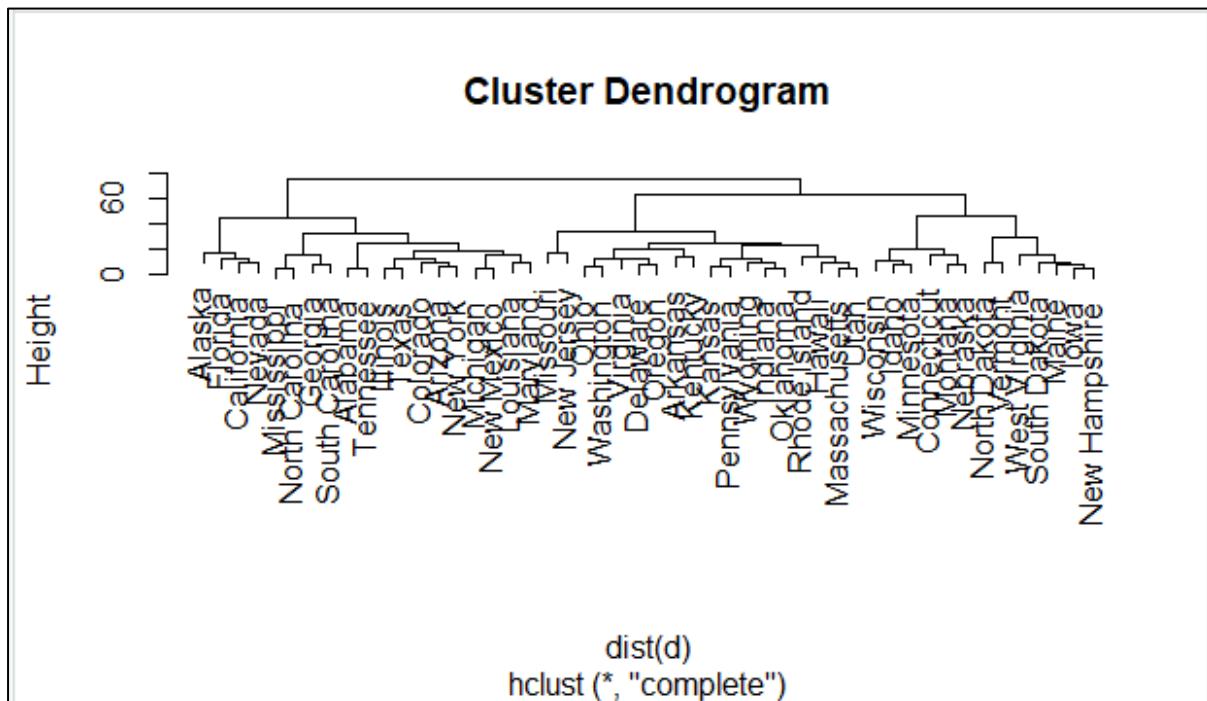
- ✓ #scale (normalizing or Standardizing)

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

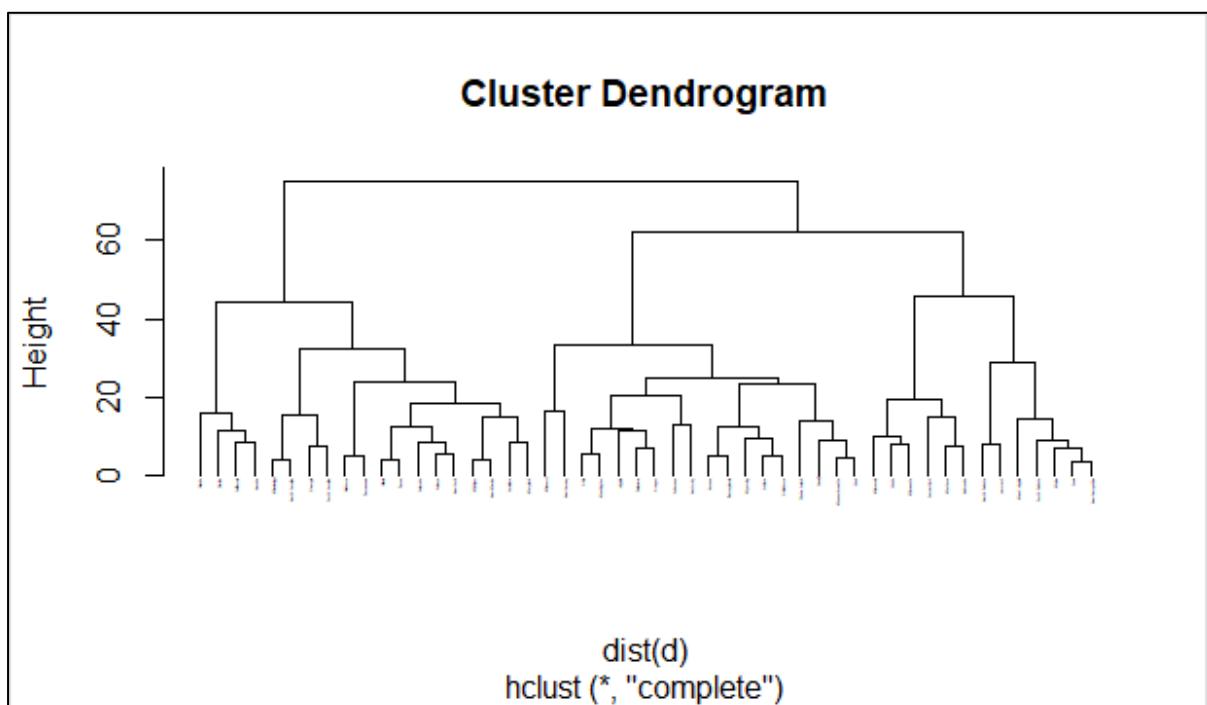
```
head(d)
```

|            | 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  |
| Colorado   | 0.02571456 | 0.3988593 | 0.8608085  | 1.864967207  |

```
✓ d<-dist(d,method = "euclidean")
  hc<-hclust(d,method = "complete")
  plot(hc)
```

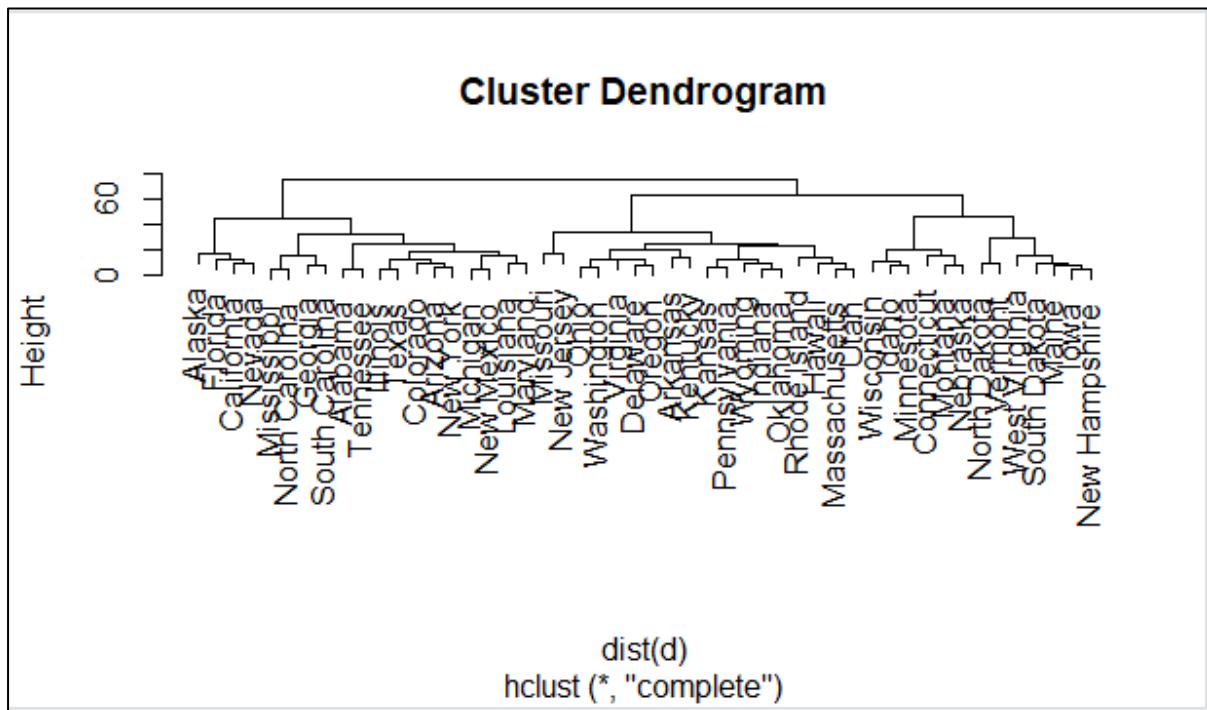


✓ `plot(hc,cex=0.1,hang=-1)`



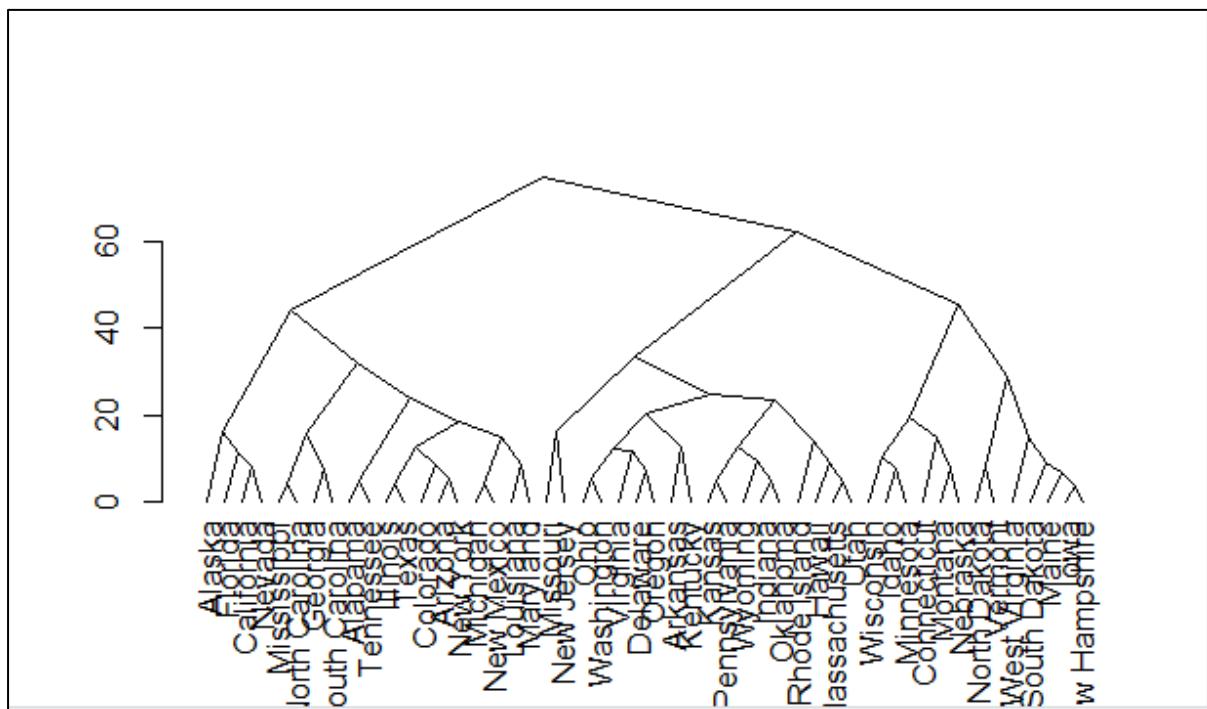
✓ `hc<-hclust(dist(d))`

`plot(hc)`

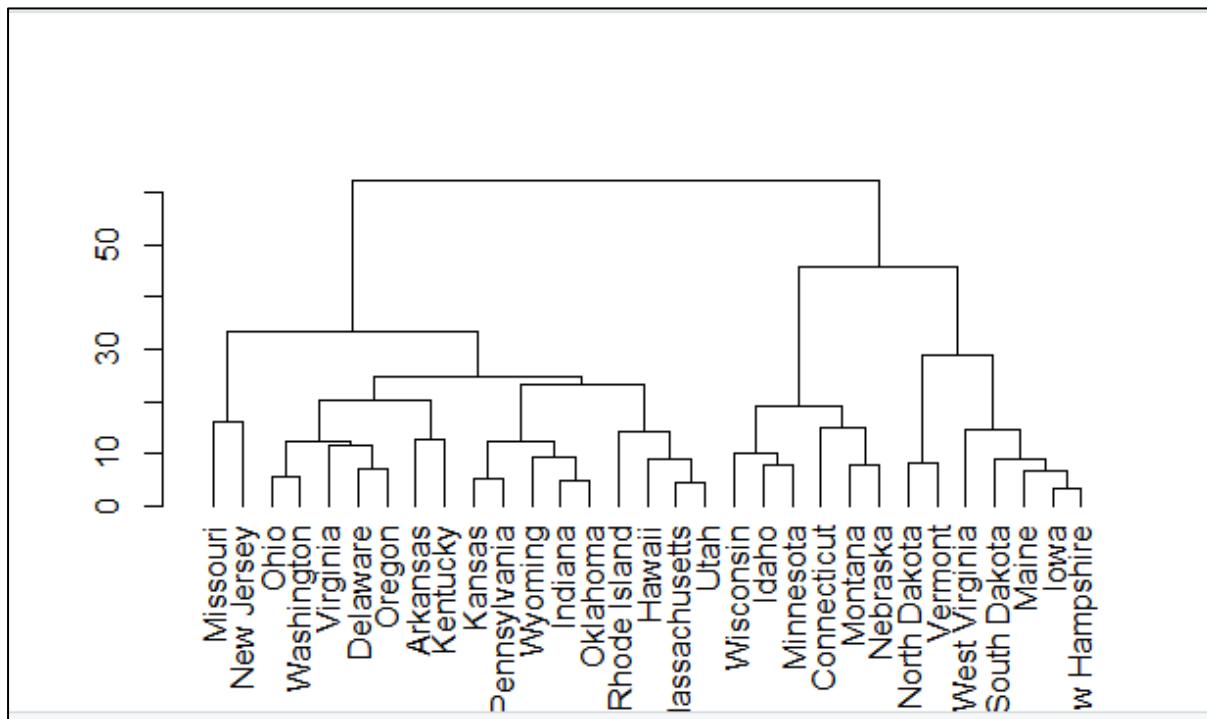


✓ `hcd=as.dendrogram(hc)`

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



✓ plot(cut(hcd,h=75)\$upper)



✓ plot(cut(hcd,h=75)\$lower[[2]])

