



# Oracle 11g SQL Lab Guide Part I

**This is another document from our learn Oracle SQL query series. We are committed to provide the BEST learning materials to the beginners. In the series, this document assists new bees to learn SQL Queries. The document teaches steps by steps way to write simple to complex SQL queries. The document consists of many examples to ease your learning. Join our professional training program to learn from experts.**

**Provided By: BISP**  
<http://bispsolutions.wordpress.com>  
[learnsqlquery.wordpress.com](http://learnsqlquery.wordpress.com)

**Created By: Abhishek Kumar Srivastava**  
**Reviewed By: Amit Sharma**  
**Team BISP**

<b>Table of Content.....</b>	<b>2-7</b>
<b>(A) Introduction.....</b>	<b>9-14</b>
Oracle 11g Database.....	9
Data Models.....	9
Entity Relationship Model.....	10
Relational Database.....	10
Using SQL Commands.....	10
SQL Development Environment.....	11
Oracle SQL Developer.....	12-14
Creating Database Connection.....	12-14
<b>1. Data Retrieving by using SELECT statement.....</b>	<b>15-22</b>
Creating Table.....	15
Insert Data into Table.....	15
Using SELECT statement for All Columns.....	15-16
Using SELECT statement for Selecting Specific Columns.....	16
Way of writing of SQL Statement.....	16-17
Executing SQL Statements.....	17
Column Names in a Table.....	18
Using Arithmetic Expressions in SQL.....	18-19
Operators Precedence.....	19-20
Column Aliases.....	20
Defining and Using Concatenation Operator.....	21
Defining and Using Literal Character Strings.....	21

Displaying Structure of Table.....	22
<b>2. Restricting and Sorting Data.....</b>	<b>23-29</b>
Using WHERE Clause.....	23
Character Strings and Dates.....	23
Defining and Using Comparison Operator.....	24-25
Using BETWEEN Operator as for Range Condition.....	25-26
Using LIKE Operator for Pattern Matching.....	26
Wildcard Characters.....	27
Using NULL Conditions.....	27
Defining conditions using Logical Operators.....	28-29
Using AND Operator.....	28
Using OR Operator.....	28
Using NOT Operator.....	29
<b>3. Using Single-Row Functions to Customize Output.....</b>	<b>30-34</b>
SQL Functions.....	30
Single-Row Functions.....	30
Character Functions.....	30
Using Case Conversion Function.....	31-32
Using Character Multiplication Functions.....	32
Number Functions.....	32-33
• Using ROUND Function.....	32
• Using TRUNC Function.....	33
• Using MOD Function.....	33
Working with DATE Function.....	33-34

• SYSDATE.....	34
<b>4. Using Conversion Functions and Conditional Expressions.....</b>	<b>35-73</b>
Conversion Function.....	35
• Implicit.....	35
Using TO_CHAR Function with DATE.....	35
Using Nesting Function.....	36-37
Using NVL Function.....	36
Using NVL2 Function.....	36
Using NULLIF Function.....	36
Using COALESCE Function.....	36
Conditional Expressions.....	37
• CASE Expression.....	37
• DECODE Function.....	37
<b>5. Reporting aggregated Data using Group Functions.....</b>	<b>38-42</b>
Group Functions.....	38
Using AVG and SUM functions.....	38
Using MAX and MIN functions.....	39
Using the COUNT function.....	39
Using the DISTINCT keyword.....	40
Group Functions and NULL Values.....	40
Creating Groups of Data.....	40-41
Using Group By clause on multiple columns.....	41
Using Having Clause.....	41
Nesting Group Functions.....	42

## **6. Displaying Data from Multiple Tables.....43-45**

### **Types of JOIN.....43**

- **Natural Joins.....43-44**

- Natural Join clause.....43**

- USING clause.....44**

- ON clause.....44**

- Non-equi joins.....44**

- **Outer Joins.....45**

- Left Outer.....45**

- Right Outer Join.....45**

- Full Outer Join.....45**

- **Cross Joins.....45**

## **7. Using Sub queries to solve queries.....46-48**

- Sub-query.....46**

- Single-row sub-query.....46-47**

- Using Group Functions in a sub-query.....47**

- Using HAVING clause with sub-queries.....47**

- Multiple-row sub-query.....48**

- IN Operator.....48**

- ANY Operator.....48**

- ALL operator.....48**

## **8. Using the SET Operators.....49-52**

SET Operators.....	49
Oracle server and Set operators.....	49
Using the UNION Operator.....	49-50
Using the UNION ALL Operator.....	50
Using INTERSECT Operator.....	51
Using MINUS Operator.....	51
Using ORDER BY clause in set operations.....	51-52
<b>9. Data Manipulation.....</b>	<b>53-59</b>
Data Manipulation Language.....	53
Adding New Row in a table.....	53
Inserting rows with null values.....	53
Inserting special values.....	54
Inserting specific Date Values.....	54
Creating a Script.....	54
Copying Rows from one table to another.....	55
Updating data in a table.....	55
Updating rows based on another table.....	55
Removing rows from a table.....	56
TRUNCATE Statement.....	56
Database Transaction.....	57
Implicit Transaction Processing.....	57
Explicit Transaction Processing.....	58
State of data before COMMIT or ROLLBACK.....	58
Using COMMIT statement.....	58

Using ROLLBACK statement.....	59
Statement-Level Rollback.....	59
<b>10. Using DDL statements to create and manage tables.....</b>	<b>60-63</b>
Database Objects.....	60
Naming Rules.....	60
CREATE Table Statement.....	60-61
Data Types.....	61-62
Constraints.....	62
Using NOT NULL Constraint.....	63
Using UNIQUE Constraint.....	63
<b>11. Creating Other Schema Objects.....</b>	<b>64-68</b>
Database Objects.....	64
Views.....	64
Simple and Complex views.....	64
Creating View.....	64-65
Retrieving data from a View.....	65
Modifying a View.....	65-66
Creating a Complex View.....	66-67
Rules for performing DML operations on a View.....	67
Using WITH CHECK OPTION clause.....	67
Denying DML Operations.....	68
Removing a View.....	68
Sequence.....	68

## Introduction

## Oracle 11g Database:-

**Oracle 11g database** is designed for some features, which helps to the organizations to manage everything easily and deliver high-quality services:-

DBA's can increase their productivity, reduce costs, minimize errors and maximize the quality of services by using the management automation, fault diagnostics features, etc.

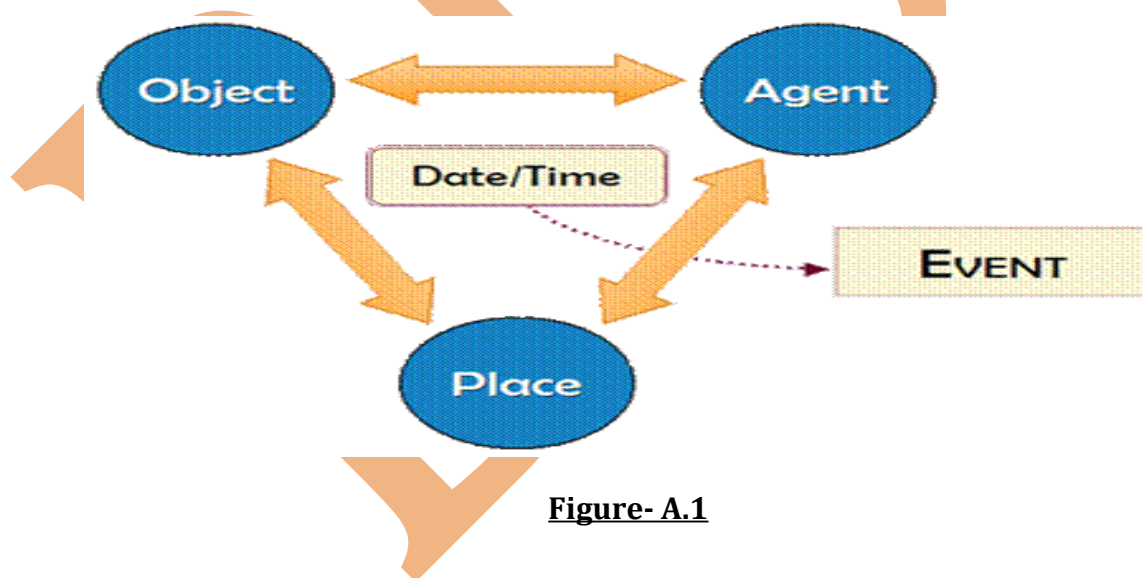
We can reduce the risk of down time and data loss by using the high availability features.

Here the performance of the database is so improved because of OLTP, secure files etc and it helps to secure the information of organization with the data encryption and masking, and complex auditing capabilities, so it's protected.

## Data Model:-

A data model is used to organize data. It captures the cardinality and referential integrity rules needed to ensure that the data is of good quality for the users.

The main aim of data models is to support the development of information systems by providing the definition and format of data.



**Figure- A.1**

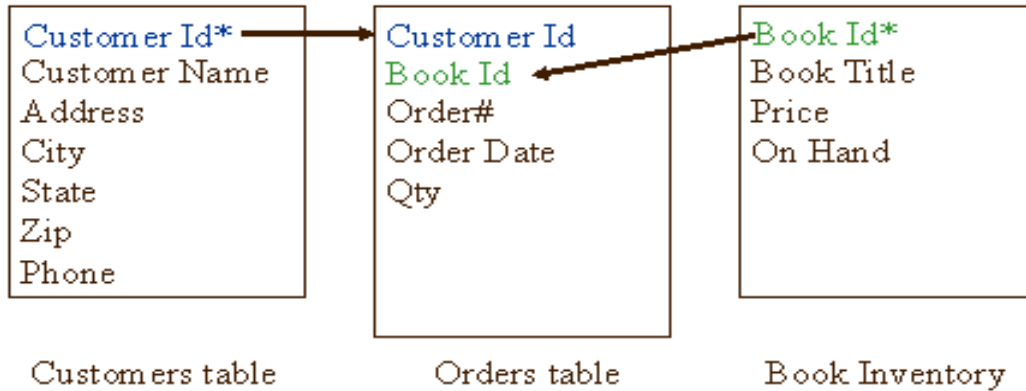
## Entity Relationship Model:-

In software [engineering](#), an **entity-relationship model (ERM)** is an abstract and conceptual representation of data.



## Relational Database:-

A **relational database** matches data by using common characteristics found within the data set. The resulting groups of data are organized and are much easier for many people to understand. It means it can contain one or many tables.



\* Denotes Primary Key

**Figure- A.2**

Diagrams created by this process are called **entity-relationship diagrams, ER diagrams, or ERDs**.

## Using SQL Command:-

SQL is a standard language for accessing databases. It is a computer language aimed to store, manipulate, and query data stored in relational databases.

SQL provides statements for variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects

## SQL Statements:

Here are the SQL Statements, which we are used for querying with database, as-

### 1. Data Manipulation Language(DML)-

Retrieve data from database, enter new rows, Changes in existing rows, and removes the rows which we no need to use from the tables in the database.

**SELECT, INSERT, UPDATE, DELETE, MERGE.**

## **2. Data Definition Language(DDL)-**

Sets up, changes and removes data structures from tables.

**CREATE, ALTER, DROP, RENAME, TRUNCATE, COMMENT.**

## **3. Data Control Language(DCL)-**

Provides or removes access rights to Oracle database and structures within it.

**GRANT, REVOKE.**

## **4. Transaction Control-**

Manages the changes made by DML statements.

**COMMIT, ROLLBACK, SAVEPOINT.**

## **SQL Development Environment:-**

For this, we used **SQL Developer** and **SQLPLUS**.

## **Oracle SQL Developer and Database Connection: -**

After installing the **Oracle 11g** on **Oracle Enterprise Linux5**, login as the **Oracle** user.

Now first, as we have to open the **SQLDEVELOPER**, so for opening this, open the **Terminal** and go to the folder where **SQLDEVELOPER** exists. As shown in the **Green** box. And run the **sqldeveloper** shell file.

Now execute this command as an **oracle** user:-

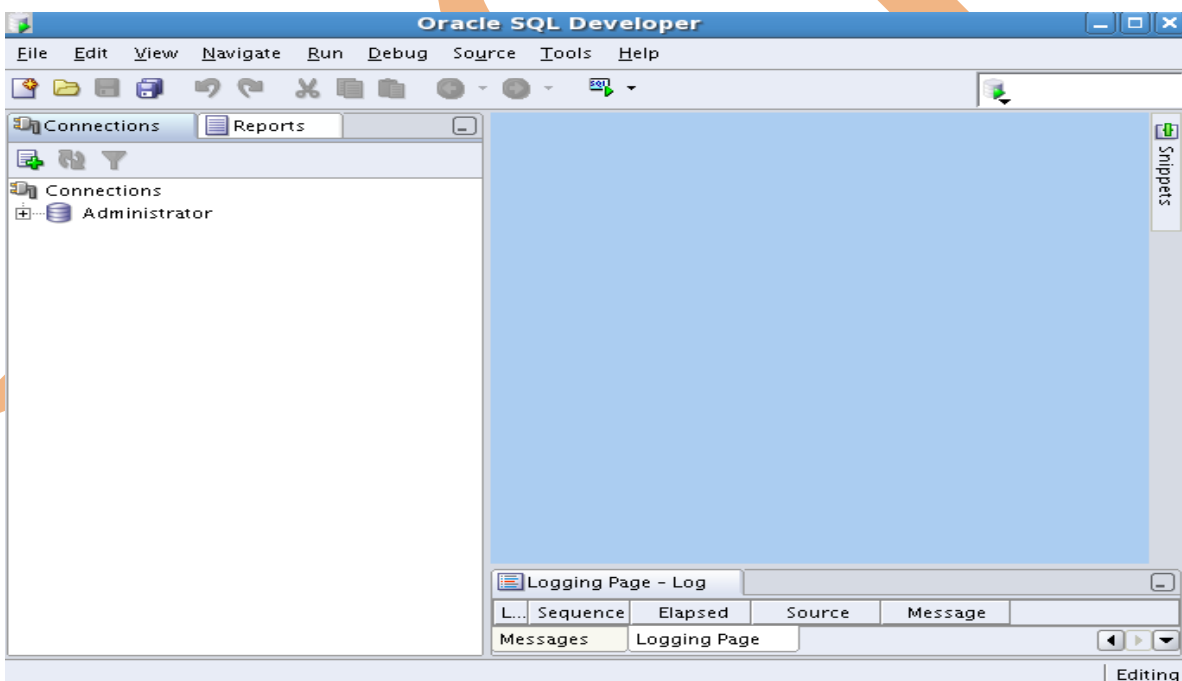
```
$ cd /u01/app/oracle/product/11.1.0/db_1/sqldeveloper
```

```
$ ./sqldeveloper.sh
```

```
[oracle@localhost ~]$ cd /u01/app/oracle/product/11.1.0/db_1/sqldeveloper  
[oracle@localhost sqldeveloper]$ ./sqldeveloper.sh
```

**Figure: A.3**

After executing this command, sqldeveloper starts to work properly.



**Figure: A.4**

Now we have to open the **SQLPLUS**. So for running this, we have to execute the following commands.

So we have to set the '**ORACLE\_SID**', '**ORACLE\_BASE**', and '**ORACLE\_HOME**'.

For setting the **ORACLE\_HOME**, we have to set the path for the **db\_home** as here **db\_1**:-

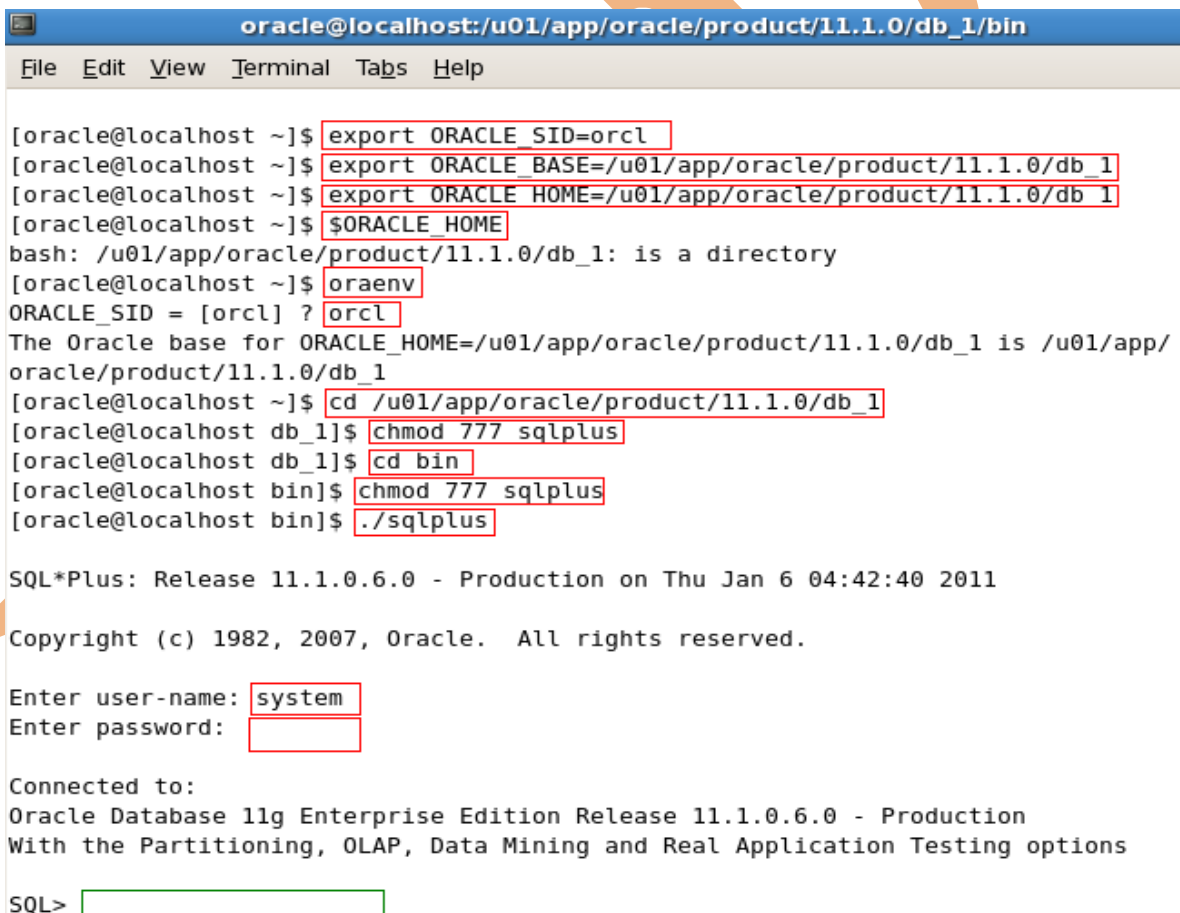
```
$ export ORACLE_SID=orcl
```

```
$ export ORACLE_BASE=/u01/app/oracle/product/11.1.0/db_1
```

```
$ export ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1
```

And then set the **Environment Variable** as 'orcl' as shown in the figure:-

```
$ oraenv
```



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help

[oracle@localhost ~]$ export ORACLE_SID=orcl
[oracle@localhost ~]$ export ORACLE_BASE=/u01/app/oracle/product/11.1.0/db_1
[oracle@localhost ~]$ export ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1
[oracle@localhost ~]$ $ORACLE_HOME
bash: /u01/app/oracle/product/11.1.0/db_1: is a directory
[oracle@localhost ~]$ oraenv
ORACLE_SID = [orcl] ? orcl
The Oracle base for ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1 is /u01/app/
oracle/product/11.1.0/db_1
[oracle@localhost ~]$ cd /u01/app/oracle/product/11.1.0/db_1
[oracle@localhost db_1]$ chmod 777 sqlplus
[oracle@localhost db_1]$ cd bin
[oracle@localhost bin]$ chmod 777 sqlplus
[oracle@localhost bin]$ ./sqlplus

SQL*Plus: Release 11.1.0.6.0 - Production on Thu Jan 6 04:42:40 2011

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Enter user-name: system
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

**Figure: A.5**

After the Environment variable, go to the **db\_home** as **db\_1** here and give the permission to the **sqlplus** and then go inside the 'bin' folder and again give the permission to **sqlplus** and execute the '**sqlplus**' shell file.

```
$ cd /u01/app/oracle/product/11.1.0/db_1
```

```
$ chmod 777 sqlplus
```

```
$ cd bin
```

```
$ chmod 777 sqlplus
```

```
$. /sqlplus
```

After executing these commands the **sqlplus** wants the **user\_name** and **password** before starts.

So give the correct **user\_name** and **password**.

Then **sqlplus** is ready for use that is shown in the **green** box.

## Chapter- 1

1. **Creating Table:** - For creating table, we have to give the table name and the columns name and their data type and the size of the data which we can insert. Now execute the following SQL command for creating the table-

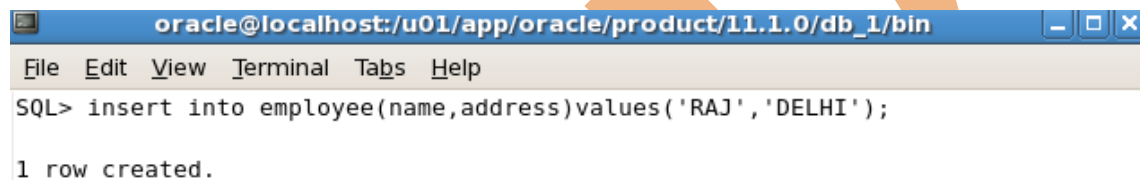


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> create table employee(name varchar(20), address varchar(20));

Table created.
```

**Figure-1.1**

2. **Insert Data into Table:** - When table will create, and then we have to insert data into the table, which will be inserting in the following columns that are specified before. So we can insert data by executing the following SQL command-



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> insert into employee(name,address)values('RAJ','DELHI');

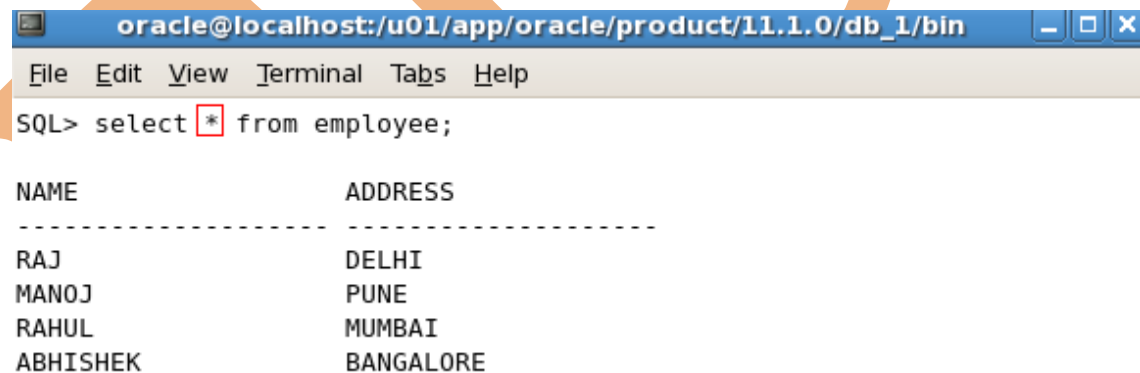
1 row created.
```

**Figure-1.2**

3. **Using SELECT Statement for All Columns:** -

**SELECT** identifies the columns to be displayed.

**FROM** identifies the table containing those Columns.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select * from employee;
```

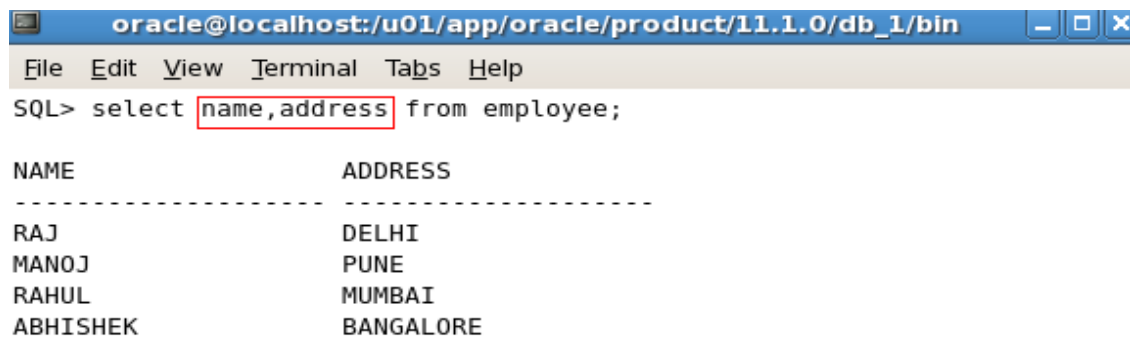
NAME	ADDRESS
RAJ	DELHI
MANOJ	PUNE
RAHUL	MUMBAI
ABHISHEK	BANGALORE

**Figure-1.3**

We can display all columns of data in a table by executing the **SELECT** statement with an **asterisk (\*)**.

In the above example, the **employee** table contains two columns **name** and **address**. The table contains **four** rows of data for each column.

We can also show all data by executing all the column names-

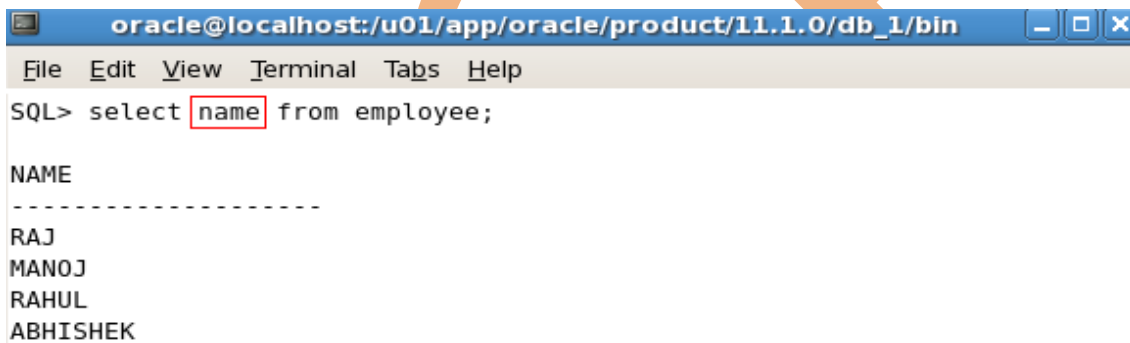


The screenshot shows a terminal window with the title bar 'oracle@localhost:/u01/app/oracle/product/11.1.0/db\_1/bin'. The menu bar includes 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The command prompt shows 'SQL> select name,address from employee;'. The output is a table with two columns: 'NAME' and 'ADDRESS'. The data rows are: RAJ (DELHI), MANOJ (PUNE), RAHUL (MUMBAI), and ABHISHEK (BANGALORE). The 'name' and 'address' columns in the query are highlighted with red boxes.

NAME	ADDRESS
RAJ	DELHI
MANOJ	PUNE
RAHUL	MUMBAI
ABHISHEK	BANGALORE

**Figure-1.4**

4. **Selecting some Specific Columns:** - We can use the **SELECT** statement to display some specific columns only.



The screenshot shows a terminal window with the title bar 'oracle@localhost:/u01/app/oracle/product/11.1.0/db\_1/bin'. The menu bar includes 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The command prompt shows 'SQL> select name from employee;'. The output is a table with one column: 'NAME'. The data rows are: RAJ, MANOJ, RAHUL, and ABHISHEK. The 'name' column in the query is highlighted with a red box.

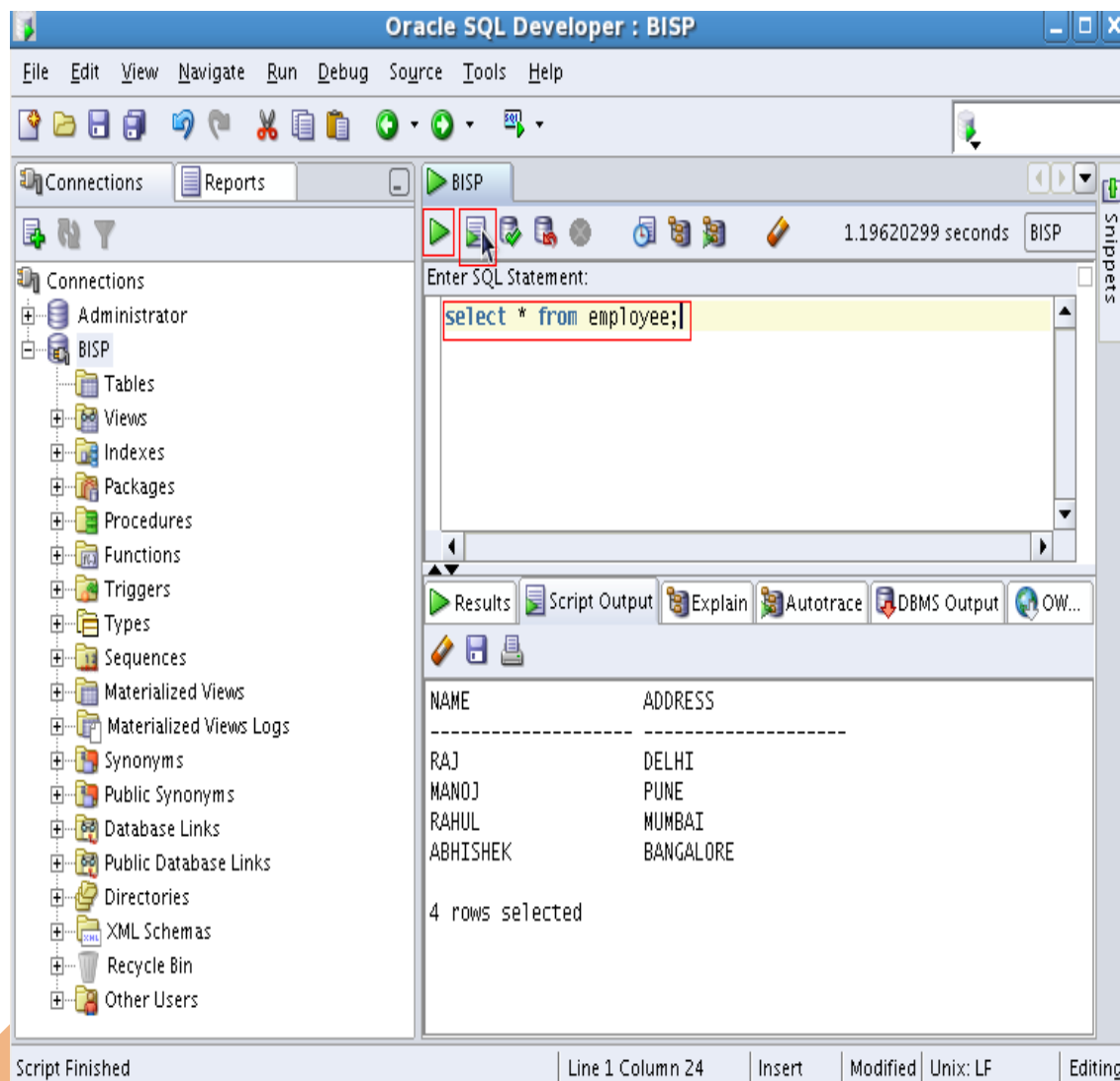
NAME
RAJ
MANOJ
RAHUL
ABHISHEK

**Figure-1.5**

5. **Ways of Writing SQL Statement:** -

- SQL Statements are not case sensitive.
- SQL Statements can be entered on more than one line.
- Keywords cannot be Split or abbreviated.
- In SQL \* Plus, we are required to end each SQL Statement with a **Semicolon (;)**. But in SQL Developer it's optional, but when we execute multiple SQL Statements then Semicolon is required.

6. **Executing SQL Statements:** -In SQL Developer, click on the **Run Script** icon or press **F5** button to run the command in SQL Worksheet. We can also click on **Execute Statement** or press **F9** button to run the command in SQL Worksheet.



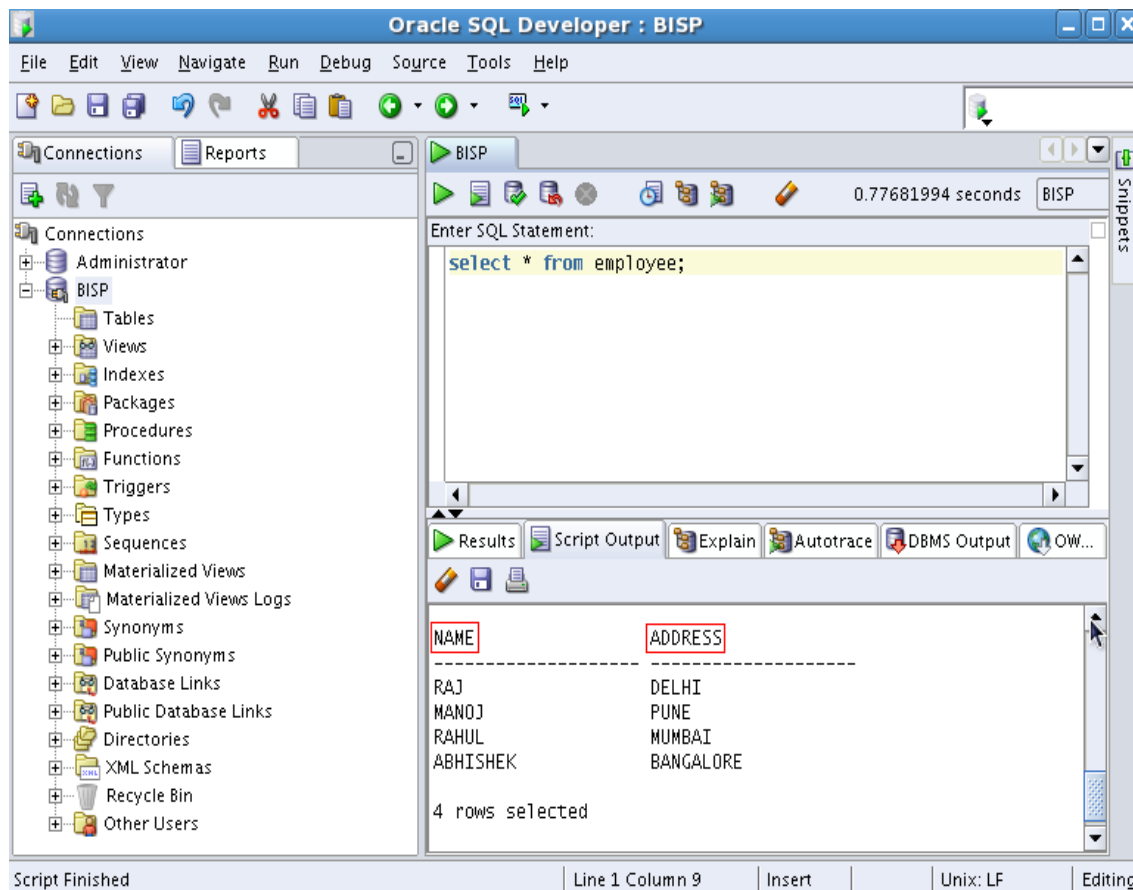
**Figure-1.6**

In SQL \* Plus, terminate the SQL Statement with a **semicolon (\*)** and press **ENTER** to run the command.

**7. Column Names in a Table:** - Column name in SQL Developer is always display in Left-aligned and in Upper case.

Column name in SQL \* Plus also is always display in Left-aligned and in Upper case.





**Figure-1.7**

**8. Using Arithmetic Expressions:** -We may need to modify the data and also have to perform some calculations. All these are possible using the arithmetic expressions. An arithmetic expression contains the Column names, Arithmetic operators and Constant numeric values. These are the arithmetic operators-

- + (Add),**
- (Subtract),**
- \*(Multiply),**
- / (Divide)**

w, we have a table and there is a **salary** column and we have to update this column.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select * from employee1;

NAME                SALARY
-----
RAJ                  5000
MANOJ                8000
RAHUL               10000
ABHISHEK            12000
```

**Figure-1.8**

Now, we will increase the salary of all employees by 500. So execute the following command-

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> UPDATE employee1 SET salary=salary+500;

4 rows updated.

SQL> commit;

Commit complete.

SQL> select * from employee1;

NAME                SALARY
-----
RAJ                  5500
MANOJ                8500
RAHUL               10500
ABHISHEK            12500
```

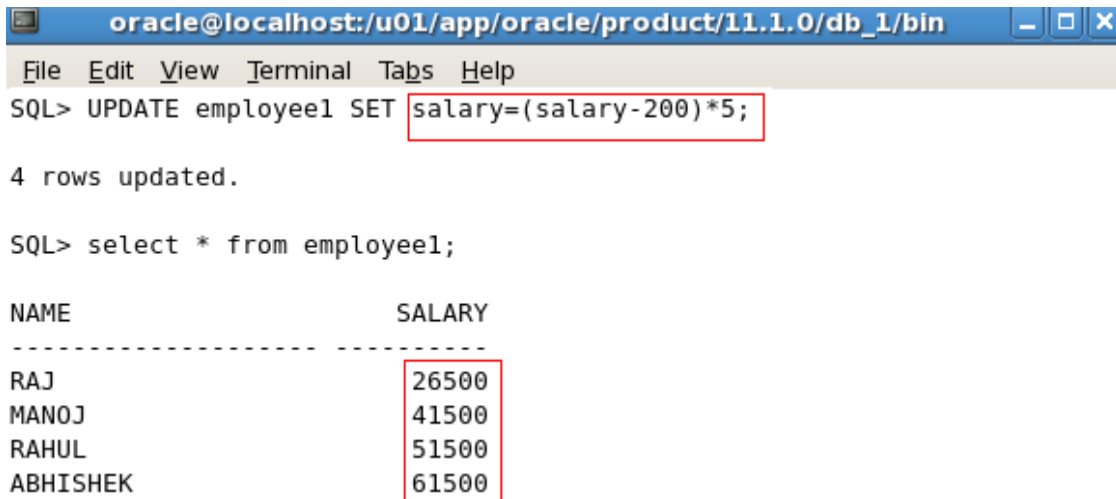
**Figure-1.9**

**9. Operators Precedence:** -If in an arithmetic expression, more than one operator is used then there is some precedence for using the operators.

So there are some rules of precedence of the arithmetic operators-

- Multiplication and Division operators occur before the Addition and Subtraction operators.
- Operators of same priority are evaluated from left to right.

- Parentheses are used to override the default precedence.



```

oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> UPDATE employee1 SET salary=(salary-200)*5;

4 rows updated.

SQL> select * from employee1;

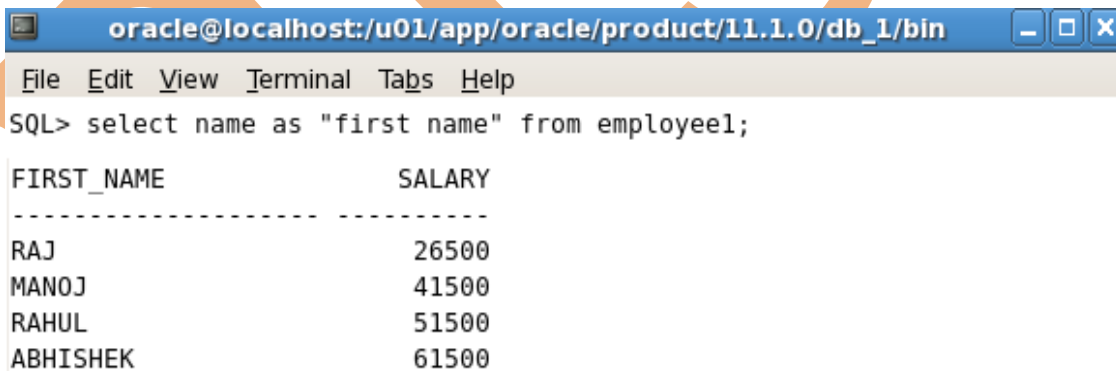
NAME                SALARY
-----
RAJ                  26500
MANOJ                41500
RAHUL                51500
ABHISHEK             61500
  
```

**Figure-1.10**

In the above example, there used the parentheses and the operators also.

- 10. Column Aliases:** -By this, we can rename the column headings. There is used **AS** keyword in between the column name and the alias.

There is used Double quotation mark (""), if alias contains **space** or **Special Characters**.



```

oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select name as "first name" from employee1;

FIRST_NAME          SALARY
-----
RAJ                  26500
MANOJ                41500
RAHUL                51500
ABHISHEK             61500
  
```

**Figure-1.11**

In the above example, there used Double quotes for alias name because there is space in between the name. And now we can see the Column name has been changed.

- 11. Concatenation Operator:** -It links Columns or character strings to other columns. It is represented by two vertical bars (||).

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select * from employee;

NAME                ADDRESS
-----
RAJ                  DELHI
MANOJ                PUNE
RAHUL                MUMBAI
ABHISHEK             BANGALORE

SQL> select (name || ' ') || address as full from employee;

FULL
-----
RAJ DELHI
MANOJ PUNE
RAHUL MUMBAI
ABHISHEK BANGALORE
```

**Figure-1.12**

12. **Literal Character String:** -A literal is a character, a number, or a date that is included in **SELECT** statement.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select (name || ' from ' ) || address as emp from employee;

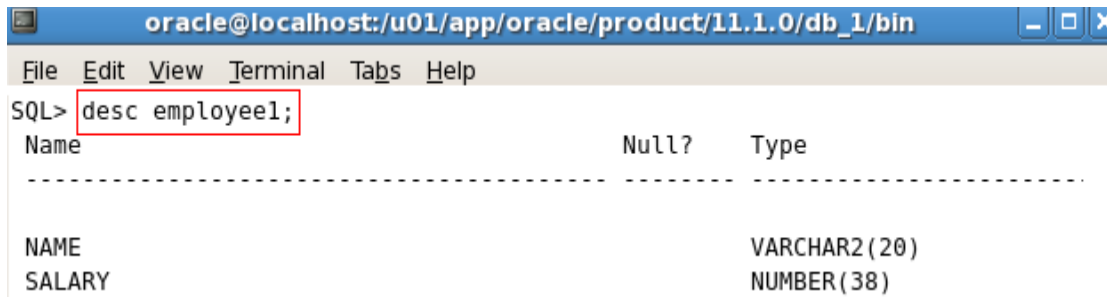
EMP
-----
RAJ from DELHI
MANOJ from PUNE
RAHUL from MUMBAI
ABHISHEK from BANGALORE
```

**Figure-1.13**

Character and Date literals must be enclosed within Single Quote marks (") whereas Number literals need not be enclosed.

Like in above example, **'from'** is as literal character string.

13. **Displaying Structure of Table:** -Use the **DESC** command to display the structure of a table.



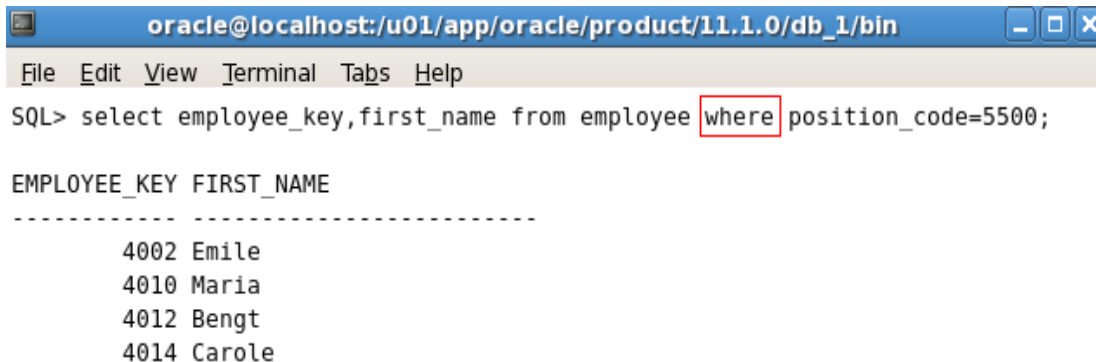
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> desc employee1;
Name                                     Null?      Type
-----
NAME                                     VARCHA2(20)
SALARY                                   NUMBER(38)
```

**Figure-1.14**

## **Chapter- 2**

1. **Using WHERE Clause:** -A **WHERE** clause contains a condition that must be met and it directly follows **FROM** clause. If condition will be true, the row meeting the condition is returned.

The **WHERE** clause can compare values in columns, literal, arithmetic expression and functions.



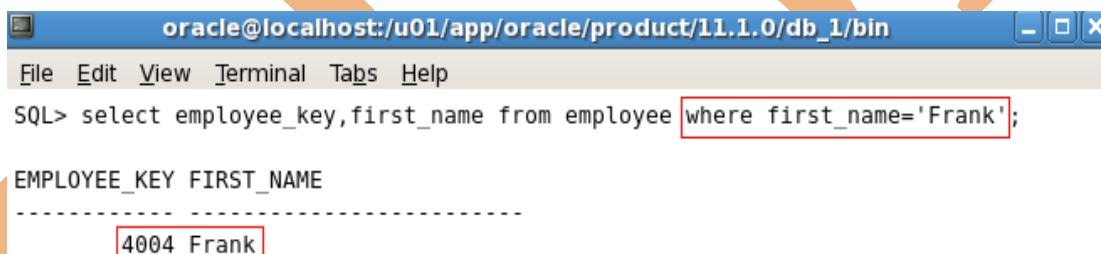
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,first_name from employee where position_code=5500;

EMPLOYEE_KEY FIRST_NAME
-----
4002 Emile
4010 Maria
4012 Bengt
4014 Carole
```

**Figure-2.1**

In the above example, we used the condition with **name** and getting the **employee\_key** in result.

2. **Character Strings and Dates:** - Character strings and Date values are enclosed with single quotation marks ("). Character values are case-sensitive and Date values are Format-sensitive.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,first_name from employee where first_name='Frank';

EMPLOYEE_KEY FIRST_NAME
-----
4004 Frank
```

**Figure-2.2**

In the above example, there is condition with character string, so we enclosed it with single quotation marks.

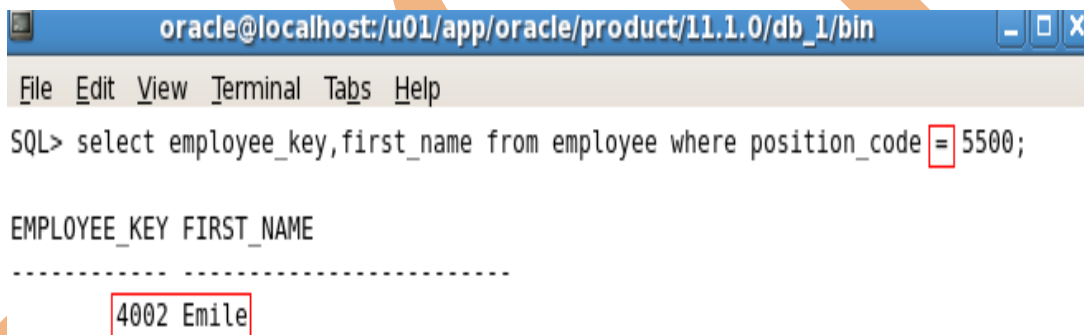
Default format for displaying the date is **DD-MM-YR**.

3. **Comparison Operators:** -Comparison operators are used in conditions that compare one expression with another expression or value. They are used with **WHERE** clause like this-

**SQL> Select expr from table\_name where expr operator value;**

Here are some operators, which are used for comparing two expressions or values-

= (Equal to),  
> (Greater than),  
>= (Greater than or equal to),  
< (Less than),  
<= (Less than or equal to),  
<> (Not equal to),  
**BETWEEN** value **AND** value (Between two values),  
**IN**(set) (Match any of list of values),  
**LIKE** (Match a character plan)  
**IS NULL** (Is a null value)

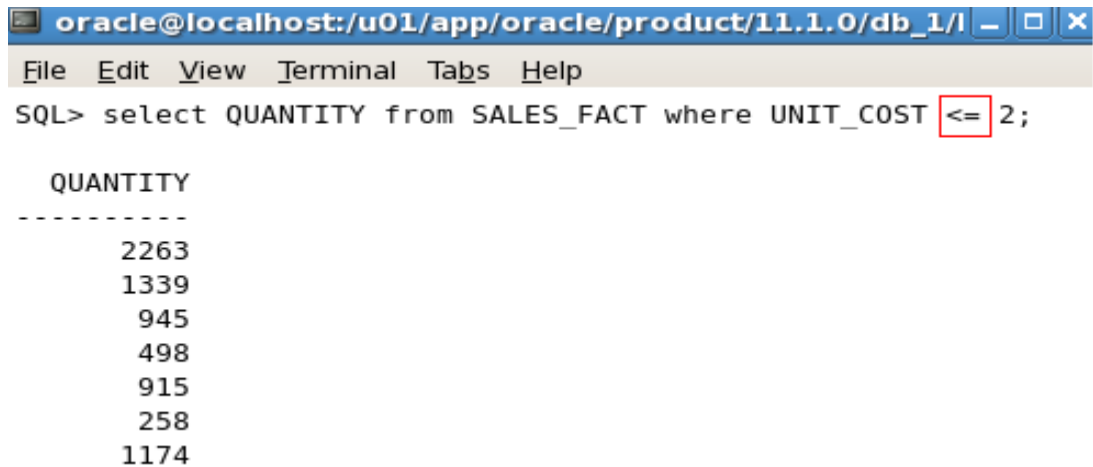


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,first_name from employee where position_code = 5500;

EMPLOYEE_KEY FIRST_NAME
-----
4002 Emile
```

**Figure-2.3**

Now, here is an example for comparing the value-

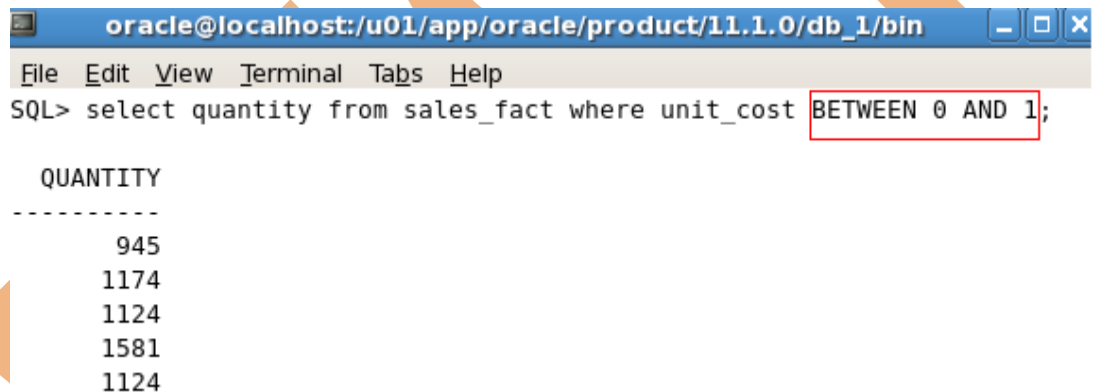


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/
File Edit View Terminal Tabs Help
SQL> select QUANTITY from SALES_FACT where UNIT_COST <= 2;

  QUANTITY
  -
2263
1339
 945
 498
 915
 258
1174
```

**Figure-2.4**

**Using BETWEEN operators for Range condition** -We can display rows based on a range of values using **BETWEEN** operators. The range that we specify will contain lower and upper limit of range.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select quantity from sales_fact where unit_cost BETWEEN 0 AND 1;

  QUANTITY
  -
 945
1174
1124
1581
1124
```

**Figure-2.5**

**Using IN Operator-** For testing for values in a specified set of values, use the IN operator. If there is used character or date, then it must be enclosed within the single quotation marks ("").

The **IN** operator can be used with any data type.

The **IN** operator is used only for logical simplicity.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,manager_code4 from employee where manager_mb4
IN ('Donald Gardner','Barbara Samuelson');

EMPLOYEE_KEY MANAGER_CODE4
-----
4025          10405
4026          10461
4027          10461
4028          10405
4029          10405
4030          10405
4014          10405
4015          10405
4016          10405
4017          10405
4018          10405
```

**Figure-2.6**

In the above example, there is used the **manager\_mb4** and on the basis of this, the **employee\_key** and **manager\_code4** will show.

4. **Using LIKE Operator:** -The **LIKE** operator can be used as a shortcut for some **BETWEEN** comparisons. We may not always know the exact value to search for. We can select rows that match a character pattern by using **LIKE** operator. There is two symbols can be used to construct the search string-

- \* % (Represents any sequence of zero or more characters)
- \* \_ (Represents any single character)

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,manager_mb4 from employee where record_start_
date LIKE '%03';

EMPLOYEE_KEY MANAGER_MB4
-----
4028 Donald Gardner
4033 ?? ??
4015 Donald Gardner
4016 Donald Gardner
4017 Donald Gardner
4018 Donald Gardner
4024 Donald Gardner
4001 Barbara Samuelson
4003 Barbara Samuelson
```

**Figure-2.7**

In the above example, the data of **employee\_key** and **manager\_mb4** will show for the year of **2003** because we used the **%03** with **LIKE** operator.

5. **Wildcard Characters:** -The % and \_ symbols can be used in any combination with literal characters for pattern matching.



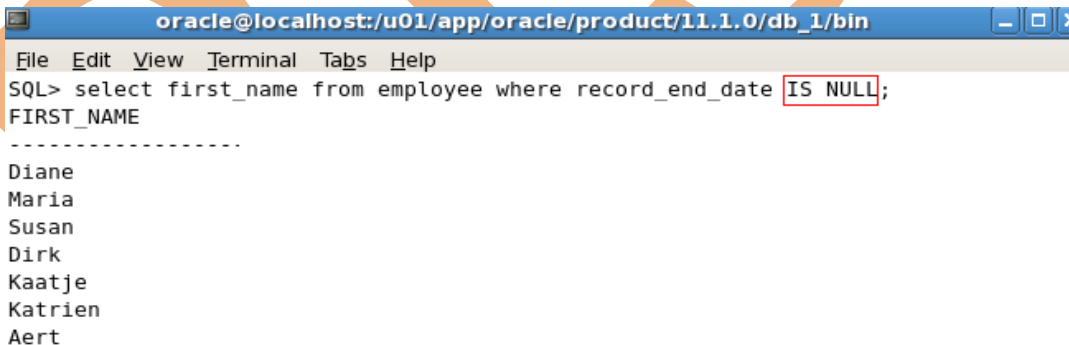
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name from employee where first_name LIKE '_a%';

FIRST_NAME
-----
Janice
Daniel
Daniel
Saburo
Masuzo
Maria
Carole
Samantha
```

**Figure-2.8**

6. **Using NULL Conditions:** -A null value means that the value is unavailable, unassigned or unknown. We cannot test with = operator because null cannot be equal or unequal to any value.

**NULL** conditions include **IS NULL** or **IS NOT NULL** condition. The **IS NULL** condition tests for null value.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name from employee where record_end_date IS NULL;

FIRST_NAME
-----
Diane
Maria
Susan
Dirk
Kaatje
Katrien
Aert
```

**Figure-2.9**

In the above example, there is shown the **first\_name** whose **record\_end\_date** columns have **NULL** value.

7. **Logical Operators:** -A logical condition combines the result of two component conditions to produce a single result based on those conditions or inverts the result in a single condition. A row is returned if the overall result of the condition is true.

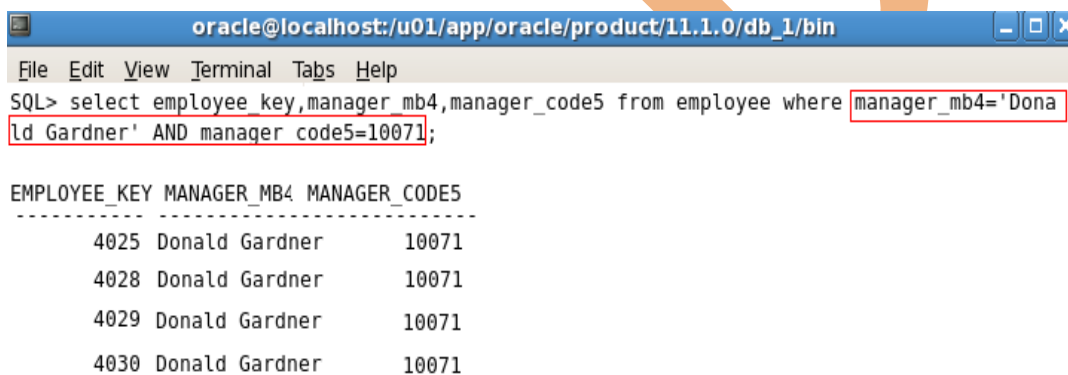
There are three logical operators available in SQL-

**AND-** Returns TRUE if both component conditions are true.

**OR-** Returns TRUE if either component condition is true.

**NOT-** Returns TRUE if the condition is false.

**AND Operator:** AND operator requires both the conditions to be true.



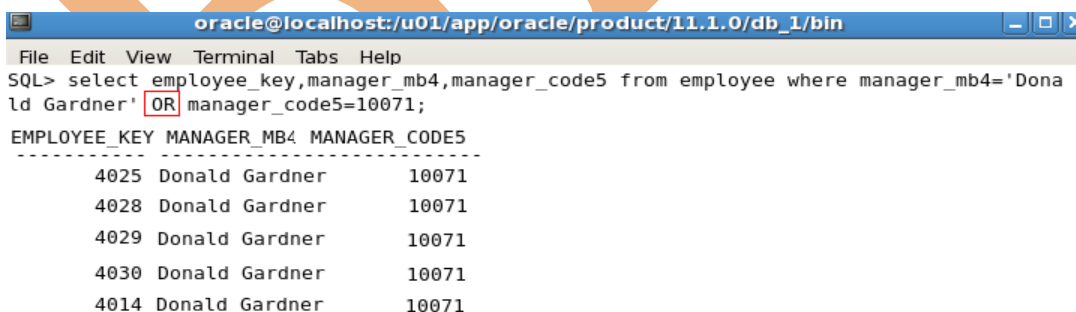
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,manager_mb4,manager_code5 from employee where manager_mb4='Donald Gardner' AND manager_code5=10071;
```

EMPLOYEE_KEY	MANAGER_MB4	MANAGER_CODE5
4025	Donald Gardner	10071
4028	Donald Gardner	10071
4029	Donald Gardner	10071
4030	Donald Gardner	10071

**Figure-2.10**

In the above example, here is checking the **manager\_mb4** and **manager\_code5** by giving condition and give the result for **employee\_key**.

**OR Operator:** OR operator requires either component condition to be true.



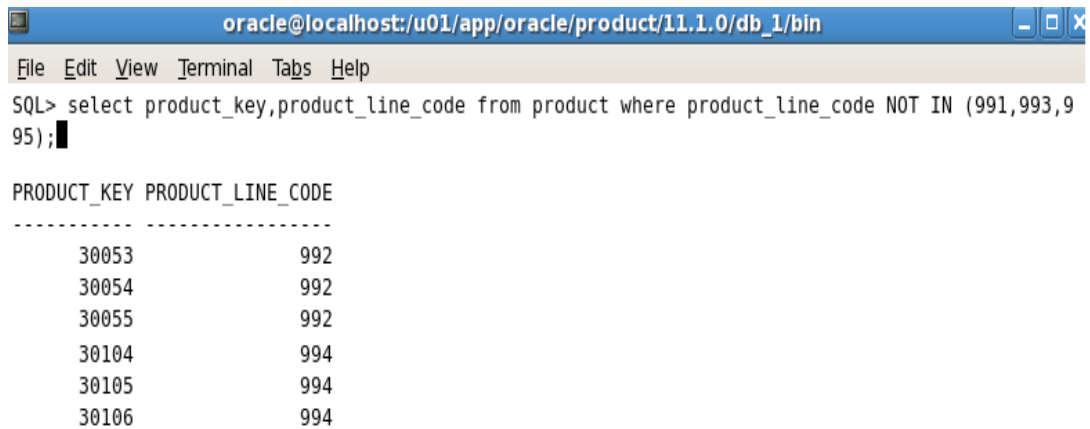
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,manager_mb4,manager_code5 from employee where manager_mb4='Donald Gardner' OR manager_code5=10071;
```

EMPLOYEE_KEY	MANAGER_MB4	MANAGER_CODE5
4025	Donald Gardner	10071
4028	Donald Gardner	10071
4029	Donald Gardner	10071
4030	Donald Gardner	10071
4014	Donald Gardner	10071

**Figure-2.11**

In the above example, whenever one condition will be true **employee\_key** will be shown.

### NOT Operator:



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select product_key,product_line_code from product where product_line_code NOT IN (991,993,995);
```

PRODUCT_KEY	PRODUCT_LINE_CODE
30053	992
30054	992
30055	992
30104	994
30105	994
30106	994

**Figure-2.12**

In the above example, all other **product\_key** is showing that is not related with the **product\_line\_code** condition.

## Chapter- 3

1. **SQL Functions:** -Functions are a very powerful feature of SQL. SQL Functions sometimes take arguments and always return a value. They can be used to do the following-

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

There are two types of SQL Functions-

**(A). Multiple-row functions-** Functions can manipulate groups of rows to give one result per group of rows. These functions are also known as **group functions**.

**(B). Single-row functions-** These functions operate on single rows only and return one result per row. There are different types of single-row functions-

- **Character Functions:** Accept character input and can return both character and number values.
- **Number Functions:** Accept numeric input and return numeric values.
- **Date Functions:** Operate on values of the **DATE** data type.

2. **Character Functions:** -Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following-

- Case-conversion functions
- Character-manipulation functions

**LOWER (col/expr)** - Converts alpha character values to lowercase

**UPPER (col/expr)** - Converts alpha character values to uppercase

**INITCAP (col/expr)** - Converts alpha character values to uppercase for first character of the word and all other in lowercase.

**CONCAT (col1/expr1, col2/expr2)** – Concatenates the first character value to the second character value.

And more are-

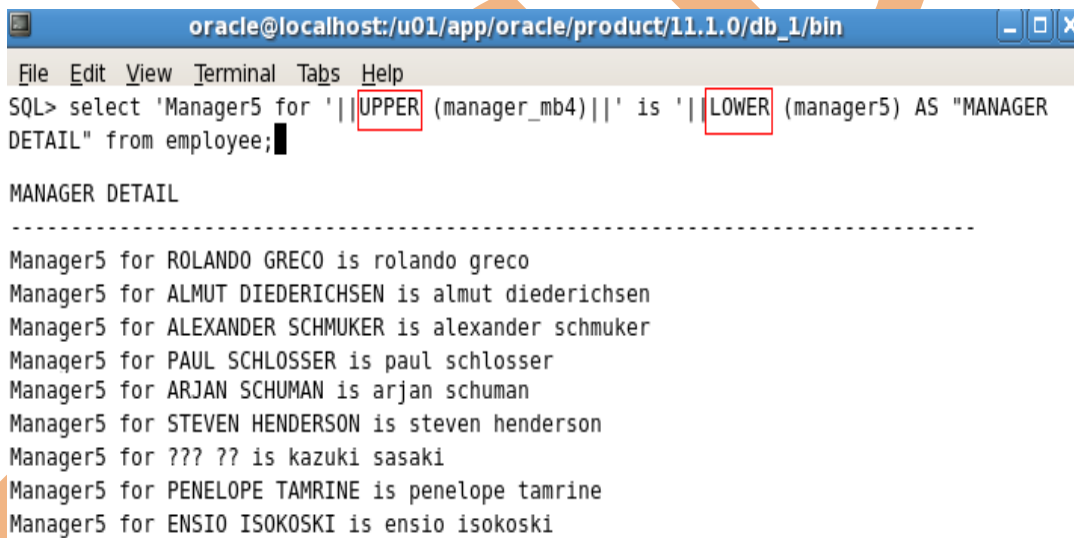
**SUBSTR, LENGTH, INSTR, LPAD, RPAD, TRIM, REPLACE.**

3. **Case conversion functions:** -These functions convert the case for character strings.

**LOWER:** Converts character strings to lowercase.

**UPPER:** Converts character strings to uppercase.

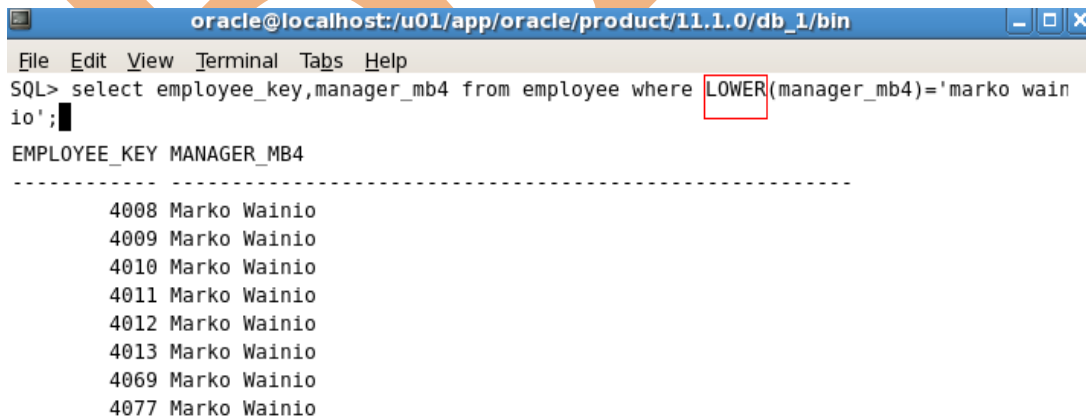
**INITCAP:** Converts first letter to uppercase and all in the lowercase.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select 'Manager5 for '||UPPER (manager_mb4)||' is '||LOWER (manager5) AS "MANAGER
DETAIL" from employee;

MANAGER DETAIL
-----
Manager5 for ROLANDO GRECO is rolando greco
Manager5 for ALMUT DIEDERICHSEN is almut diederichsen
Manager5 for ALEXANDER SCHMUKER is alexander schmuker
Manager5 for PAUL SCHLOSSER is paul schlosser
Manager5 for ARJAN SCHUMAN is arjan schuman
Manager5 for STEVEN HENDERSON is steven henderson
Manager5 for ??? ?? is kazuki sasaki
Manager5 for PENELOPE TAMRINE is penelope tamrine
Manager5 for ENSIO ISOKOSKI is ensio isokoski
```

**Figure-3.1**



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,manager_mb4 from employee where LOWER(manager_mb4)='marko wain
io';

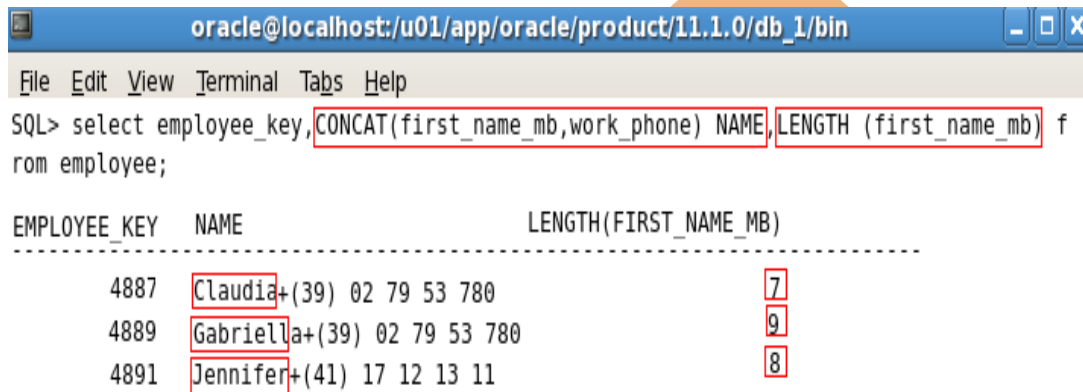
EMPLOYEE_KEY  MANAGER_MB4
-----
4008 Marko Wainio
4009 Marko Wainio
4010 Marko Wainio
4011 Marko Wainio
4012 Marko Wainio
4013 Marko Wainio
4069 Marko Wainio
4077 Marko Wainio
```

**Figure-3.2**

In the example Figure-3.2, the manager names are in upper and lower case both and we are searching by only lower case by **LOWER** function and the result will be same as in the database.

4. **Character-manipulation function:** -There are some character-manipulation functions-

**CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, REPLACE, TRIM.**



```

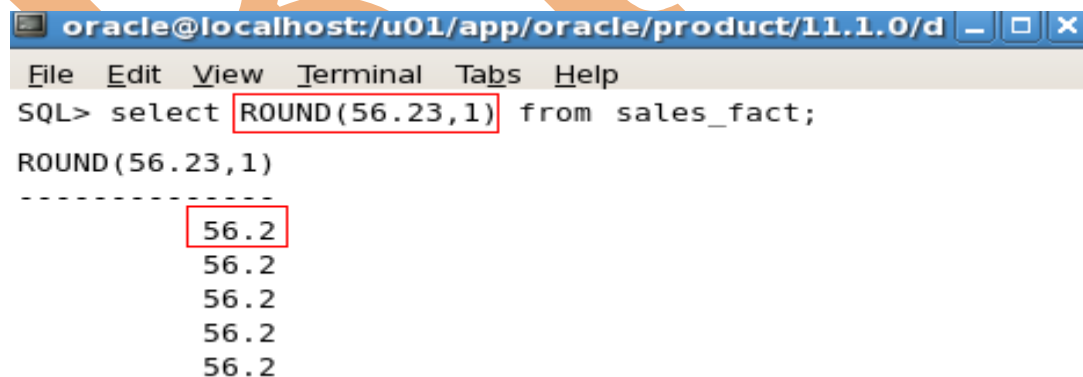
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key, CONCAT(first_name_mb,work_phone) NAME, LENGTH (first_name_mb) f
rom employee;

EMPLOYEE_KEY  NAME                                LENGTH(FIRST_NAME_MB)
-----
4887  Claudia+(39) 02 79 53 780          7
4889  Gabriella+(39) 02 79 53 780        9
4891  Jennifer+(41) 17 12 13 11          8
  
```

**Figure-3.3**

5. **Number Functions:** -Number functions accept numeric input and return numeric values. There are some number functions that are used-

- **Using ROUND Function-** It rounds the column, expression, or value to **n** decimal places. If the second argument is zero or is missing, the value is rounded to zero decimal places.



```

oracle@localhost:/u01/app/oracle/product/11.1.0/d
File Edit View Terminal Tabs Help
SQL> select ROUND(56.23,1) from sales_fact;

ROUND(56.23,1)
-----
56.2
56.2
56.2
56.2
56.2
  
```

**Figure-3.4**

- **Using TRUNC Function-** Truncates the column, expression, or value to **n** decimal places or, if **n** is omitted, **n** defaults to zero.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/d
File Edit View Terminal Tabs Help
SQL> select TRUNC(56.23,1) from sales_fact;
TRUNC(56.23,1)
-----
56.2
56.2
```

**Figure-3.5**

- **Using MOD Function-** The MOD function finds the remainder of the first argument divided by the second argument.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select manager_mb5, employee_key, MOD (employee_key, 40) from employee;
MANAGER_MB5      EMPLOYEE KEY  MOD(EMPLOYEE_KEY,40)
-----
Almut Diederichsen      4935      15
Alexander Schmuker      4936      16
```

**Figure-3.6**

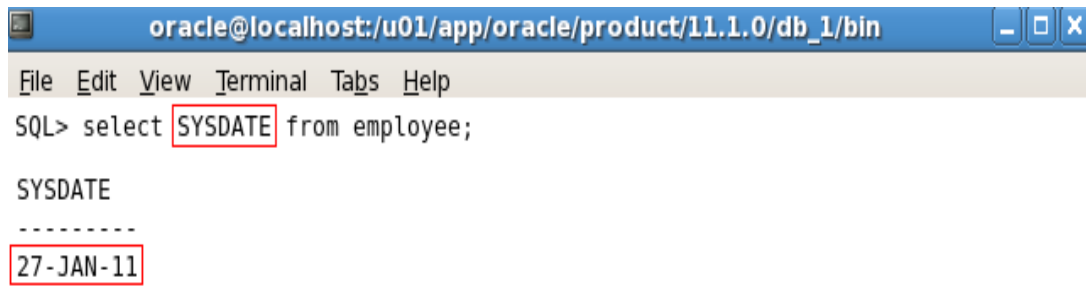
6. **DATE Functions:** - Oracle database stores dates in an internal numeric format. The default display and input format for any date is **DD-MM-RR**.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key, record_start_date from employee where record_start_d
ate = '28-JAN-03';
EMPLOYEE_KEY RECORD_ST
-----
4015 28-JAN-03
4133 28-JAN-03
4172 28-JAN-03
```

**Figure-3.7**

- **SYSDATE-** It is a function that returns DATE and TIME. It returns the current database server date and time. SYSDATE returns the current date and time set for the operating system on which the database resides.





```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select SYSDATE from employee;

SYSDATE
-----
27-JAN-11
```

**Figure-3.8**

## **Chapter-4**

### **1. Conversion Functions: -**

**Implicit-** In the expressions, the Oracle server can automatically convert the following-

**VARCHAR2 or CHAR to NUMBER**

**VARCHAR2 or CHAR to DATE**

**NUMBER to VARCHAR2 or CHAR**

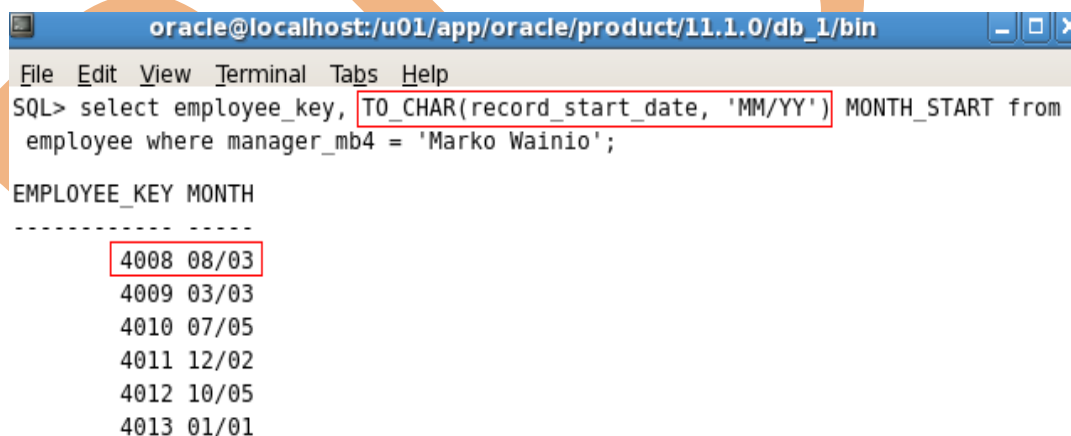
**DATE to VARCHAR2 or CHAR**

**CHAR to NUMBER** conversions succeed only if the character string represents a valid number.

2. **Using TO\_CHAR Function with DATE:** - TO\_CHAR converts a datetime data type to a value of VARCHAR2 data type in the format specified by format model. A format model is a character literal that describes the format of datetime stored in a character string.

The format model must be enclosed with single quotation marks and is case-sensitive.

The format model can include any valid date format element. But be sure to separate the date value from the format model with a comma.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key, TO_CHAR(record_start_date, 'MM/YY') MONTH_START from
employee where manager_mb4 = 'Marko Wainio';
EMPLOYEE_KEY MONTH
-----
4008 08/03
4009 03/03
4010 07/05
4011 12/02
4012 10/05
4013 01/01
```

**Figure-4.1**

3. **Using Nesting Functions:** -Nested functions are evaluated from the innermost level to the outermost level.

There is some function that works with any data type and pertain to the use of null values in the expression list-

- **Using NVL Function**-This function converts the null value to an actual value.

**NVL (expr1, expr2)**

**Expr1** is the source value that may contain a null.

**Expr2** is the target value for converting the null.

We can convert any data type, but the return value is always the same as the data type of **expr1**.

Data types that can be used are **date**, **character**, and **number**.

- **Using NVL2 Function**-This function examines the first expression. If first expression is not null< then the NVL2 function returns the second function. If first function is null, then the third expression is returned.

**NVL2 (expr1, expr2, expr3)**

**Expr1** is the source value that may contain a null.

**Expr2** is a value that is returned if expr1 is not null.

**Expr3** is a value that is returned is expr1 is null.

- **Using NULLIF Function**-It compares two expressions. If they are equal, function returns a null. If they are not equal, then function returns expr1. However we cannot specify the literal NULL for first expression.

**NULLIF (expr1, expr2)**

- **Using COALESCE Function**- COALESCE function can take multiple alternate values over the NVL function. If first expression is not null, this function returns that expression, otherwise it does same for the remaining expressions. All expressions must be of the same data type.

**COALESCE (expr1, expr2.....exprn)**

**Expr1** returns this expression if it is not null.

**Expr2** returns this expression if the first expression is null and this expression is not null.

**Exprn** returns this expression if the preceding expressions are null.

4. **Conditional Expressions:** -The two methods that are used to implement conditional processing (IF-THEN-ELSE) in a SQL statement are **CASE** expression and **DECODE** function. The CASE expression compiles with ANSI SQL. The DECODE function is specific to Oracle Syntax.
- **CASE Expression**-It facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement without having to invoke procedures.

```
CASE expr WHEN comparison_expr1 THEN return_expr1
          [WHEN comparison_expr2 THEN return_expr2
          [WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
```

**END**

- **DECODE Function**-It facilitates conditional inquiries by doing the work of a CASE expression or IF-THEN-ELSE statement.

The DECODE function decodes expression after comparing it to each **search** value. If the expression is the same as **search**, result is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

```
DECODE (col/expression, search1, result1
        [, search2, result2]
        [, default])
```

## Chapter-5

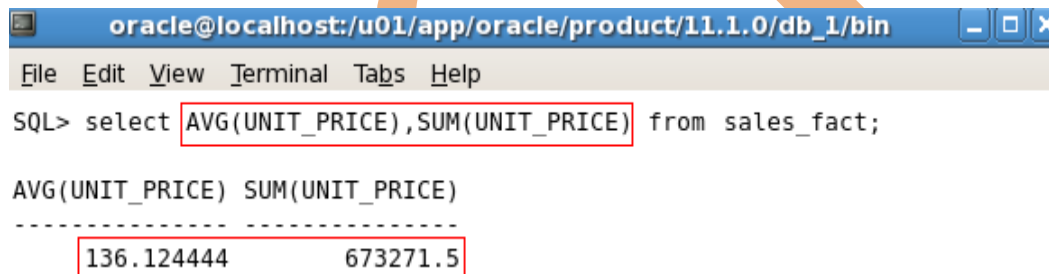
1. **Group Functions:** -Group functions operate on sets of rows to give one result per group. These sets may comprise the entire table or the table splits into groups. There are some types of group functions and that functions accepts an argument-

**AVG** (Average values of n, ignoring null values)

<b>COUNT</b>	(Number of rows)
<b>MAX</b>	(Maximum value of expr, ignoring null values)
<b>MIN</b>	(Minimum value of expr, ignoring null values)
<b>STDDEV</b>	(Standard deviation of n, ignoring null values)
<b>SUM</b>	(Sum of n values, ignoring null values)
<b>VARIANCE</b>	(Variance of n, ignoring null values)

The Group functions are placed after the SELECT keyword and we may have multiple group functions separated by commas.

2. **Using AVG and SUM functions:** -We can use AVG and SUM functions for numeric data.



```

oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select AVG(UNIT_PRICE),SUM(UNIT_PRICE) from sales_fact;

AVG(UNIT_PRICE) SUM(UNIT_PRICE)
-----
136.12444      673271.5
  
```

**Figure-5.1**

In the above example, there is average and sum of the unit\_price.

3. **Using MAX and MIN Functions:** -We can use MIN and MAX functions for numeric, character, and date data types.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select MAX(BIRTH_DATE),MIN(BIRTH_DATE) from employee;

MAX(BIRTH MIN(BIRTH
-----
19-NOV-49 25-JAN-50
```

**Figure-5.2**

In the above example, there is maximum and minimum of the birth-date of the employee.

4. **Using COUNT Function:** -It returns the number of rows in a table including the duplicate rows.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select COUNT(*) from employee;

COUNT(*)
-----
972

SQL> select COUNT(TERMINATION_DATE) from employee;

COUNT(TERMINATION_DATE)
-----
135
```

**Figure-5.3**

In the above example, there is total of 972 rows as count all the rows **count (\*)** but when we count the termination-date, there are only 135 rows, because it don't counts the rows that have null value.

5. **Using DISTINCT Keyword:** -It suppresses the counting of any duplicate values in a column.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select COUNT(DISTINCT manager2) from employee;

COUNT(DISTINCTMANAGER2)
-----
8
```

**Figure-5.4**

In the above example, the total number of manager2 is showing as there are duplicate names of same name also.

- 6. Group Functions and Null Values:** -Group functions ignore null values in the column whereas the NVL function forces Group functions to include null values.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select AVG(NVL(CURRENT_MONTH,0)) from period_time;

AVG(NVL(CURRENT_MONTH,0))
-----
6.50511945
```

**Figure-5.5**

In the above example, that rows also counted for getting the average, which have null.

- 7. Creating Groups of Data:** -At times, we need to divide the table of information into smaller groups. This can be done by using the **GROUP BY** clause.

So it is used in conjunction with the aggregate functions to group the result-set by one or more columns.

The Group By column does not have to be in the SELECT list.

**Group\_by\_expression** specifies columns determine the basis for grouping rows.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select organization_key,SUM(UNIT_COST) from sales_fact GROUP BY organization_key;

ORGANIZATION_KEY SUM(UNIT_COST)
-----
11101             16018.66
11107              9171.49
11106              6222.89
11158              9171.49
11167              8905.65
```

**Figure-5.6**

Sometimes we need to see results for groups within groups.

8. **Using Group By clause on Multiple Columns:** -Sometimes we need to see results for groups within groups.

```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select organization_key,employee_key,SUM(UNIT_COST) from sales_fact GROUP BY organization_key,employee_key;

ORGANIZATION_KEY EMPLOYEE_KEY SUM(UNIT_COST)
-----
11157             4004          5482.7
11158             4014           34.9
11158             4015          5681.21
11167             4082          5721.87
11161             4158          2319.09
```

**Figure-5.7**

9. **Using Having Clause:** -We use the Having Clause to restrict the groups in the same way that we use the **where** clause to restrict the rows that we select.

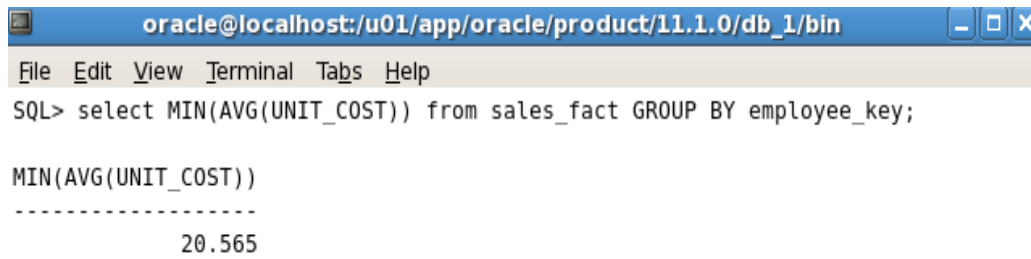
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,MIN(UNIT_COST) from sales_fact GROUP BY employee_key HAVING MIN(UNIT_COST)<2.50;

EMPLOYEE_KEY MIN(UNIT_COST)
-----
4016          .85
4018           1
4089          .85
4141          1.83
4020          1.83
4032          1.83
```

**Figure-5.8**



- 10. Nesting Group Functions:** -Group functions can be nested to a depth of two functions.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select MIN(AVG(UNIT_COST)) from sales_fact GROUP BY employee_key;

MIN(AVG(UNIT_COST))
-----
                20.565
```

**Figure-5.9**

1. **Types of JOIN:** -To join tables, we can use join syntax.

**(a) Natural Joins-**

- NATURAL JOIN clause
- USING clause
- ON clause

**(b) Outer Joins-**

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

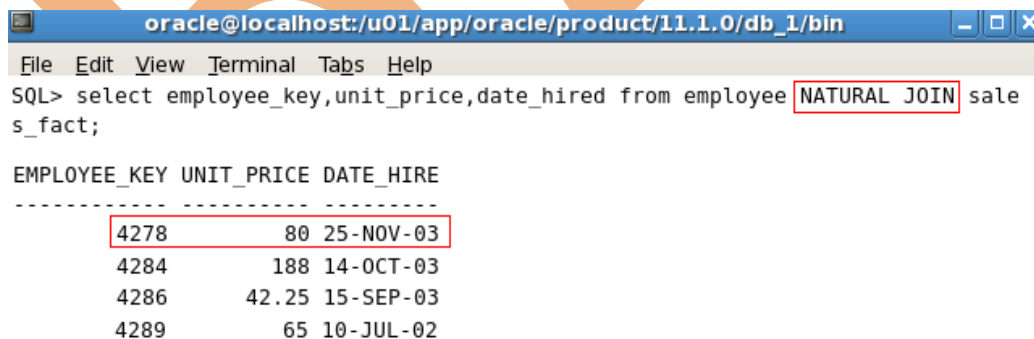
**(c) Cross Joins-**

**(a) Natural Joins-**

**Creating Natural Joins clause:**

The NATURAL JOIN clause is based on all columns in the two tables that have the same name. It selects rows from the two tables that have equal values in all matched column. If columns having the same names have different data types, an error is returned.

We can join tables automatically based on the columns in two tables that have matching data types and names. We do this by using NATURAL JOIN keyword.



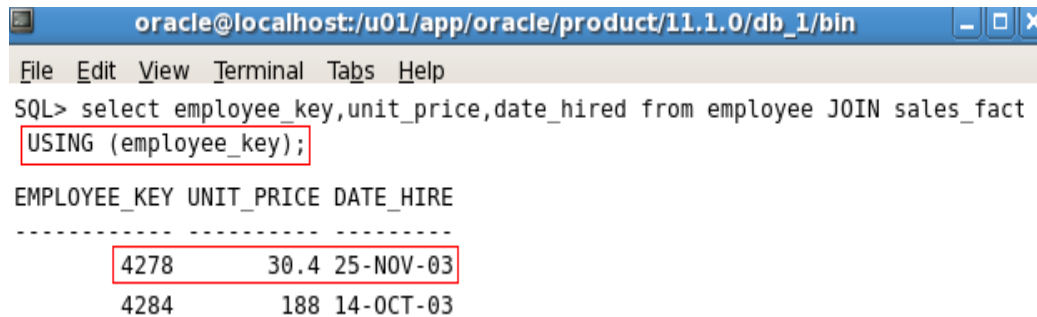
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,unit_price,date_hired from employee NATURAL JOIN sale
s_fact;

EMPLOYEE_KEY UNIT_PRICE DATE_HIRE
-----
4278          80 25-NOV-03
4284          188 14-OCT-03
4286         42.25 15-SEP-03
4289          65 10-JUL-02
```

**Figure-6.1**

**Creating joins with USING clause clause:**

If several columns have same names but data types do not match, natural join can be applied using the USING clause to specify the columns that should be used for an equijoin. USING clause is used to match only one column when more than one column matches.

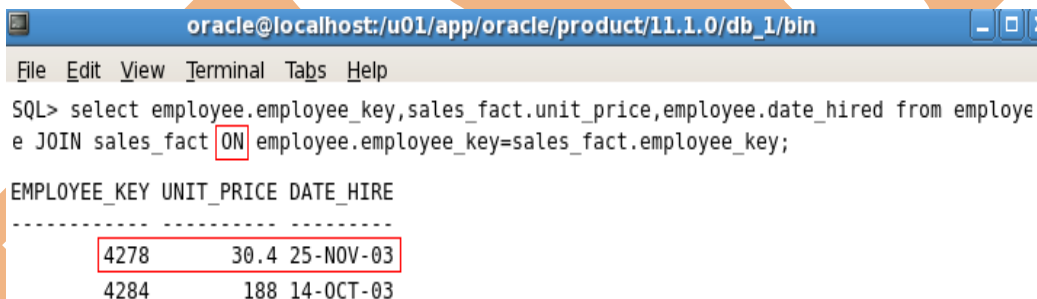


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,unit_price,date_hired from employee JOIN sales_fact
      USING (employee_key);
EMPLOYEE_KEY UNIT_PRICE DATE_HIRE
-----
4278          30.4 25-NOV-03
4284          188 14-OCT-03
```

**Figure-6.2**

### Creating joins with ON clause:

Use the ON clause to specify arbitrary conditions or specify columns to join. The join condition is separated from other search conditions.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee.employee_key,sales_fact.unit_price,employee.date_hired from employee
      e JOIN sales_fact ON employee.employee_key=sales_fact.employee_key;
EMPLOYEE_KEY UNIT_PRICE DATE_HIRE
-----
4278          30.4 25-NOV-03
4284          188 14-OCT-03
```

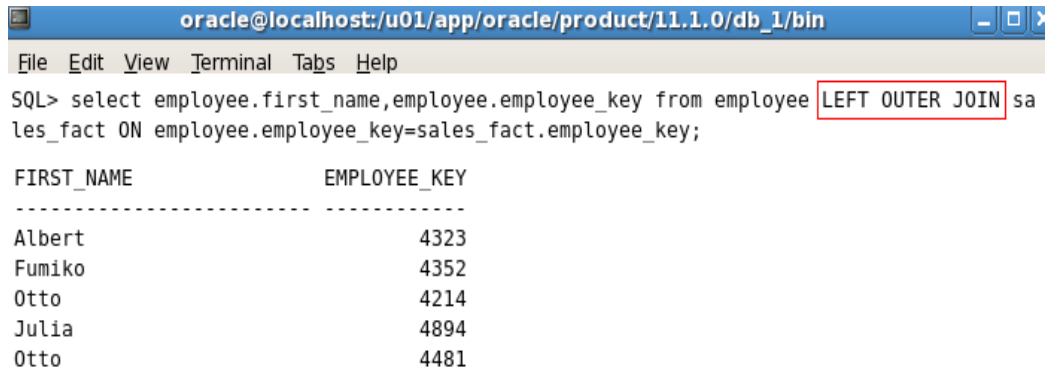
**Figure-6.3**

**Self-Join using ON clause:** The ON clause can also be used to join columns that have different names, within the same table or in a different table.

**Nonequijoins:** -A nonequijoin is a join condition containing something other than an equality operator.

**(b) Outer Joins-**If a row does not satisfy a join condition, the row does not appear in the query result.

**Left Outer Join:** Joining tables with the NATURAL JOIN, USING, or ON clauses results in an inner join.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee.first_name,employee.employee_key from employee LEFT OUTER JOIN sales_fact ON employee.employee_key=sales_fact.employee_key;

FIRST_NAME          EMPLOYEE_KEY
-----
Albert              4323
Fumiko              4352
Otto                4214
Julia               4894
Otto                4481
```

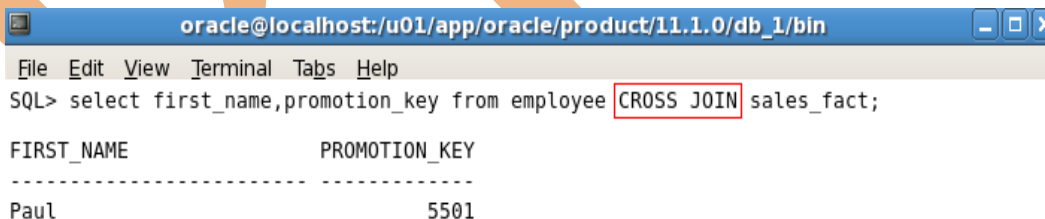
**Figure-6.4**

**Right Outer Join:** Any unmatched rows are not displayed in the output. To return the unmatched rows, we can use an outer join.

**Full Outer Join:** It retrieves all rows of all the tables, even if there is no match in the tables.

(c) **Cross Joins**-When a join condition is invalid or omitted completely, the result is a **Cartesian product**, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

The **Cross Join** clause produces the cross-product of two tables. This is also called a Cartesian product between the two tables.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name,promotion_key from employee CROSS JOIN sales_fact;

FIRST_NAME          PROMOTION_KEY
-----
Paul                5501
```

**Figure-6.5**

## **Chapter-7**

1. **Sub-Query:** -A sub-query is a SELECT statement that is embedded in the clause of another SELECT statement. We can build powerful statements out of simple ones by

using sub-queries. We can place the sub-query in a number of SQL clauses, including the following-

**WHERE** clause

**HAVING** clause

**FROM** clause

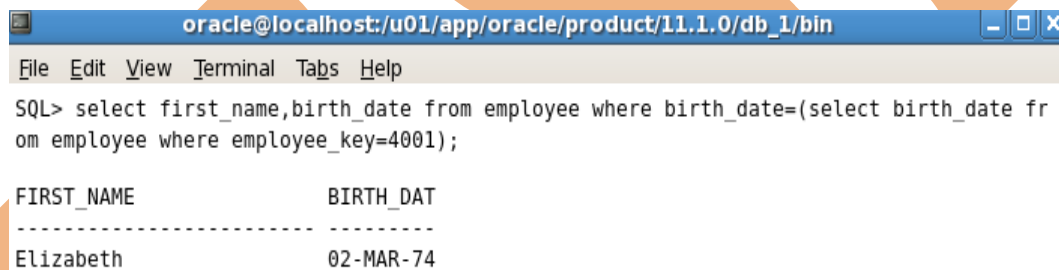
Operators can be used for the comparison conditions.

The sub-query is often referred to as nested SELECT, sub-SELECT, or inner-SELECT statement.

Always enclose the sub-query within the parentheses and place on the right side of the comparison condition.

2. **Single-row sub-query:** -Queries that return only one row from the inner SELECT statement. It uses single-row comparison operators-

**=, <, <=, >, >=, <>**



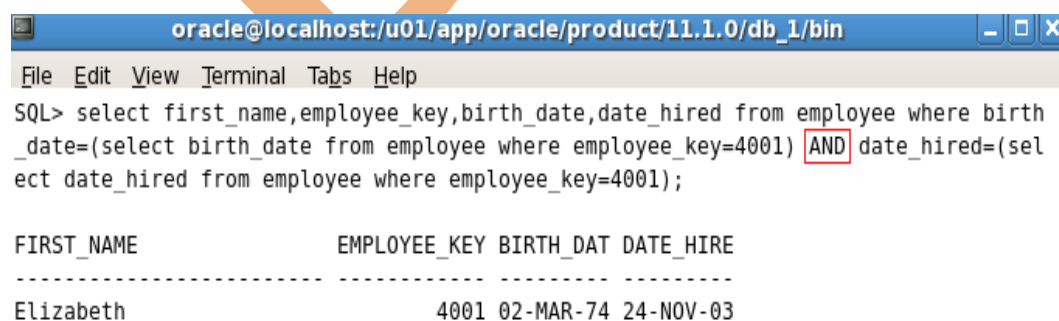
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name,birth_date from employee where birth_date=(select birth_date fr
om employee where employee_key=4001);
```

FIRST_NAME	BIRTH_DAT
Elizabeth	02-MAR-74

**Figure-7.1**

In the above example, there is only one sub-query comparison condition.

A SELECT statement can be considered as a query block.



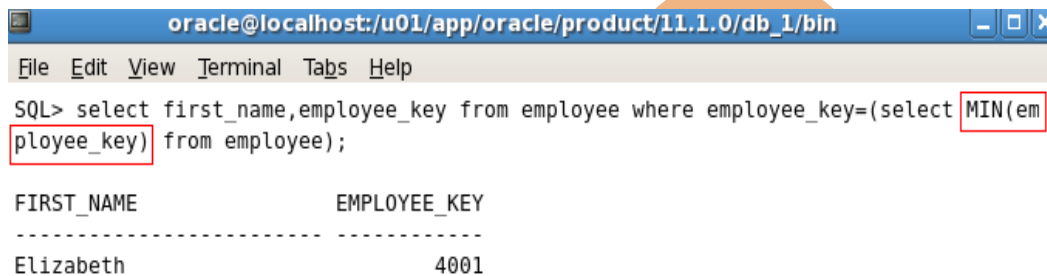
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name,employee_key,birth_date,date_hired from employee where birth
_date=(select birth_date from employee where employee_key=4001) AND date_hired=(sel
ect date_hired from employee where employee_key=4001);
```

FIRST_NAME	EMPLOYEE_KEY	BIRTH_DAT	DATE_HIRE
Elizabeth	4001	02-MAR-74	24-NOV-03

**Figure-7.2**

In the above example, there is two sub-query comparison conditions. So when both of the condition will be true, then query will be executed. Because all queries return single values, so this SQL statement called single-row sub-query.

3. **Using Group Functions in a sub-query:** -We can display data from a main query by using a group function in a sub-query to return a single row.

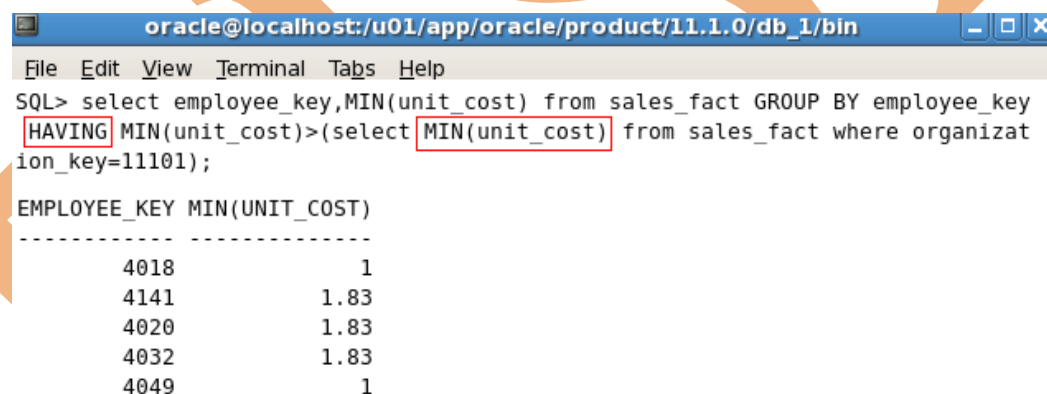


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select first_name,employee_key from employee where employee_key=(select MIN(em
ployee_key) from employee);
```

FIRST_NAME	EMPLOYEE_KEY
Elizabeth	4001

**Figure-7.3**

4. **Using HAVING clause with sub-queries:** -The Oracle server executes the sub-query and the results are returned into the **HAVING** clause of the main query.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,MIN(unit_cost) from sales_fact GROUP BY employee_key
HAVING MIN(unit_cost)>(select MIN(unit_cost) from sales_fact where organizat
ion_key=11101);
```

EMPLOYEE_KEY	MIN(UNIT_COST)
4018	1
4141	1.83
4020	1.83
4032	1.83
4049	1

**Figure-7.4**

When sub-queries execute, then some result shows but when there is not any data searched, then it returns **0 rows selected**.

5. **Multiple rows sub-query:** -Sub-queries that return more than one row are called multiple-row sub-query. We use multiple-row operator, with a multiple-row sub-query-

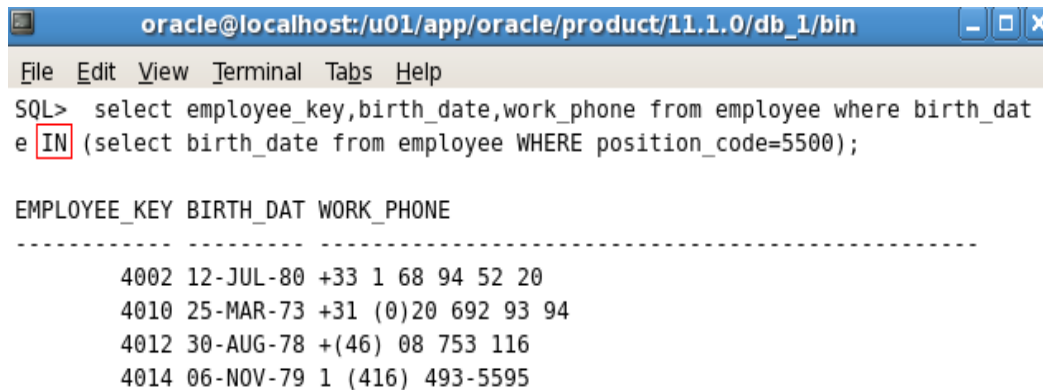
**IN** Equal to any number in list

**ANY** Compares a value to each value in a list or returned by a query.

**ALL** Compares a value to every value in a list or returned by a query.

**ALL** and **ANY** must be preceded by =, !=, >, <, >=, <=.

### Using IN Operator-



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key,birth_date,work_phone from employee where birth_date
IN (select birth_date from employee WHERE position_code=5500);

EMPLOYEE_KEY BIRTH_DAT WORK_PHONE
-----
4002 12-JUL-80 +33 1 68 94 52 20
4010 25-MAR-73 +31 (0)20 692 93 94
4012 30-AUG-78 +(46) 08 753 116
4014 06-NOV-79 1 (416) 493-5595
```

**Figure-7.5**

**Using ANY Operator**-The ANY operator compares a value to each value returned by a sub-query.

**<ALL** Means less than the minimum

**>ALL** Means more than the maximum

**Using ALL Operator**-The ALL operator compares a value to every value returned by a sub-query.

The **NOT** operator can be used with IN, ANY and All operators.

## Chapter-8

1. **SET Operators:** -Set operators combine the results of two or more component queries into one result. Queries containing set operators are called **compound queries**.

**UNION** Rows from both queries after eliminating duplications.

**UNION ALL** Rows from both queries, including all duplications.

**INTERSECT** Rows that are common to both queries.

## MINUS

Rows in first query that are not present in second query.

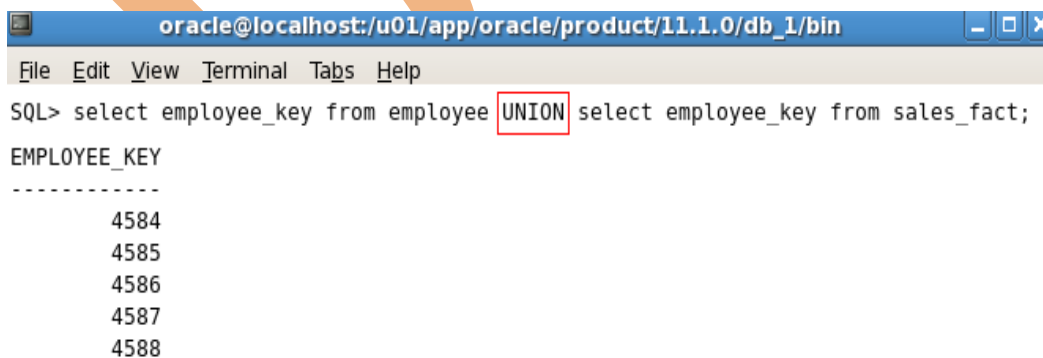
All set operators have equal precedence. If SQL statement contains multiple set operators, it evaluates from **top of left** to **bottom of right**. We should use parentheses to specify the order of evaluation explicitly in queries that use the INTERSECT operator with other set operators.

Set operators can be used in sub-queries.

2. **Oracle Server and Set Operators:** -When a query uses set operators, the oracle server eliminates duplicate rows automatically except in the case of the UNION ALL operator. If component queries select numeric data, then the data type of the return values is determined by numeric precedence. If all queries select values of the NUMBER type, then the return values have the NUMBER data type.
3. **Using the UNION Operator:** -The UNION Operator returns all rows that are selected by either query. We can use the UNION Operator to return all rows from multiple tables and eliminate any duplicate rows.

### Some Guidelines-

- Number of columns being selected must be the same.
- Data types of columns must be in the same data type group.
- The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The output is always sorted in ascending order of the columns of the SELECT clause.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key from employee UNION select employee_key from sales_fact;
EMPLOYEE_KEY
-----
4584
4585
4586
4587
4588
```

**Figure-8.1**



In the above example, there is displayed all the employee\_key of both table **employee** and **sales\_fact**. And all the duplicate records will eliminate because of UNION operator.

4. **Using UNION ALL Operator:** -The UNION ALL operator returns all rows from multiple queries. Everything will be same like UNION operator, but by the UNION ALL operator, all duplicate records will also show and record will not be sorted.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key from employee UNION ALL select employee_key from sales_fa
ct;

EMPLOYEE_KEY
-----
4239
4239
4241
4241
4241
4277
4277
4245
4245
```

**Figure-8.2**

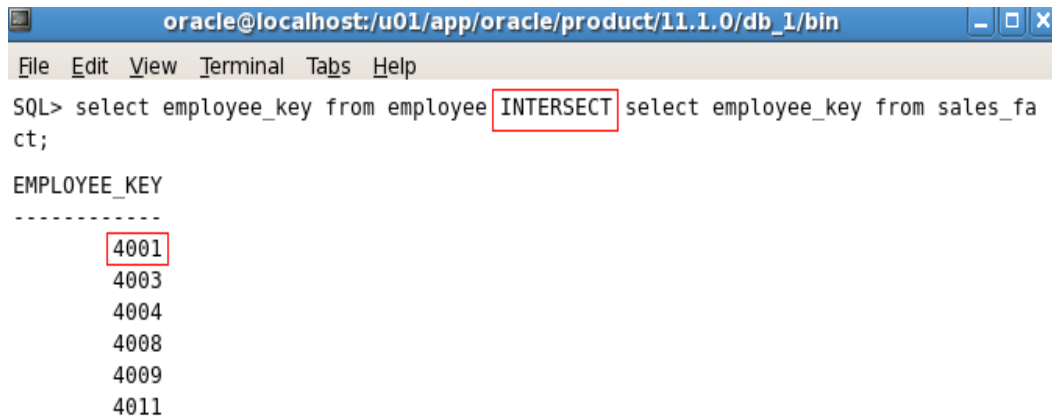
In the above example, the UNION ALL operator will display all the records of employee\_key of both the table **employee** and **sales\_fact** with unsorted and duplicate records.

5. **Using INTERSECT Operator:** -The INTERSECT operator returns rows that are common to multiple queries.

#### **Some Guidelines-**

- The number of columns and the data types of columns being selected by the SELECT statements in queries must be identical in all the SELECT statements used in the query.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT operator does not ignore NULL values.

- The names of the columns need not be identical.



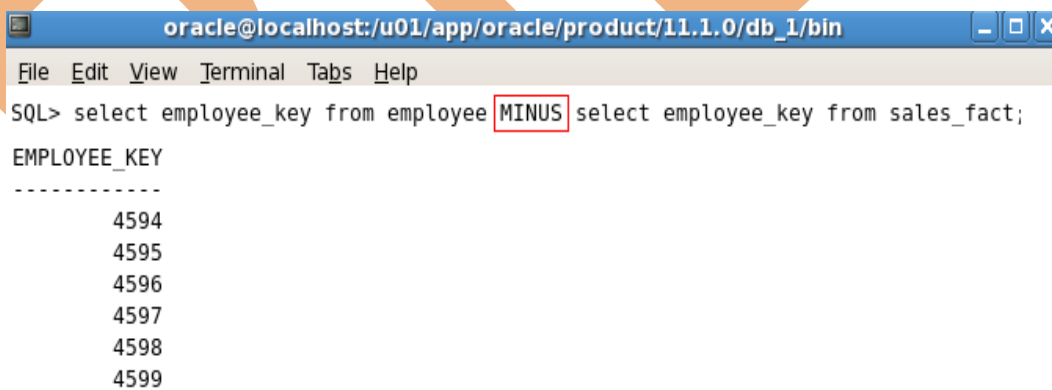
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key from employee INTERSECT select employee_key from sales_fa
ct;
EMPLOYEE_KEY
-----
4001
4003
4004
4008
4009
4011
```

**Figure-8.3**

In the above example, the INTERSECT will display those records of employee\_key that will be common in the table **employee** and **sales\_fact**.

6. **Using MINUS Operator:** -The MINUS operator returns all the distinct rows selected by first query, but not present in second query result set.

The number of columns must be same and the data types of the columns being selected by SELECT statements in the queries must belong to the same data type group in all the SELECT statements used in the query. The names of the columns need not be identical.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select employee_key from employee MINUS select employee_key from sales_fact;
EMPLOYEE_KEY
-----
4594
4595
4596
4597
4598
4599
```

**Figure-8.4**

In the above example, the employee\_key that presents only in **employee** table and not in the **sales\_fact** table.

7. **Using ORDER BY clause in set operations:** -The order by clause can appear only once at the end of compound query.

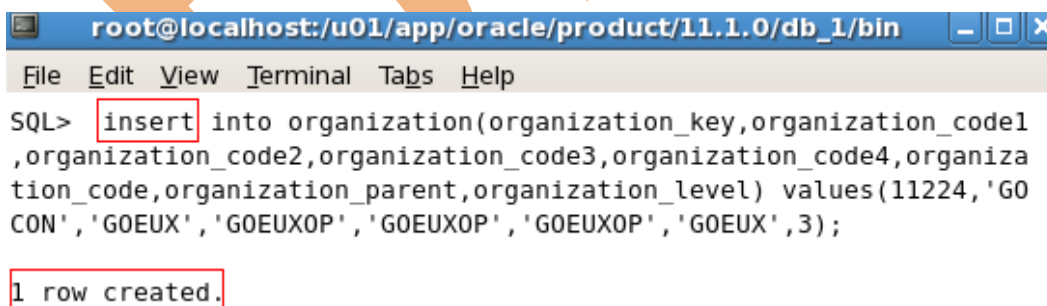
Compound queries cannot have individual ORDER BY clauses.

ORDER BY clause recognizes only the columns of the first SELECT query, not the second SELECT query.

By default, the first column of the first SELECT query is used to sort the output in an ascending order.

## Chapter-9

1. **Data Manipulation Language (DML):** -DML is core part of SQL. When we want to add, update, or delete data in the database, then we execute a DML statement. A collection of DML statements that form a logical unit of work is called a **transaction**.
2. **Add a new Row in a table:** -We can add a new row in a table by **INSERT** statement.



```
root@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> insert into organization(organization_key,organization_code1
,organization_code2,organization_code3,organization_code4,organiza
tion_code,organization_parent,organization_level) values(11224,'GO
CON','GOEUX','GOEUXOP','GOEUXOP','GOEUXOP','GOEUX',3);

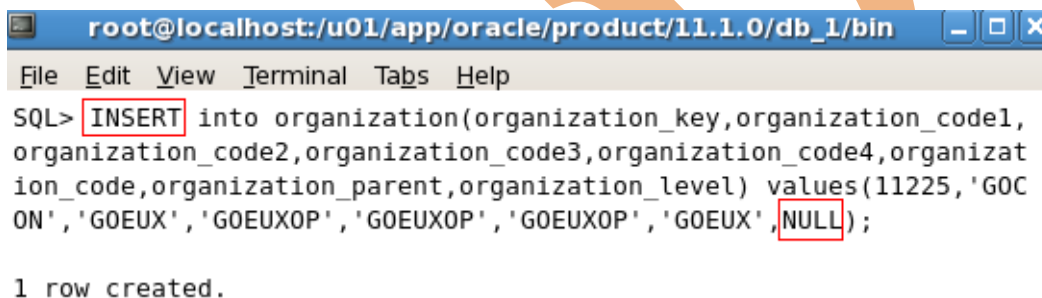
1 row created.
```

Figure-9.1

In the above example, we executed the insert statement and the result is **1 row created** that is the inserted data in the **organization** table.

With this syntax, we can insert one row at a time. All the character and date values will be enclosed within the single quotation marks and it is not recommended that we enclose numeric values within single quotation marks.

- 3. Inserting rows with Null Values:** -Be sure that we can use Null values in the targeted column by verifying the Null status with the DESCRIBE command. The **Not Null** columns have only mandatory value. Any column that is not listed explicitly obtains a null value in the new row.

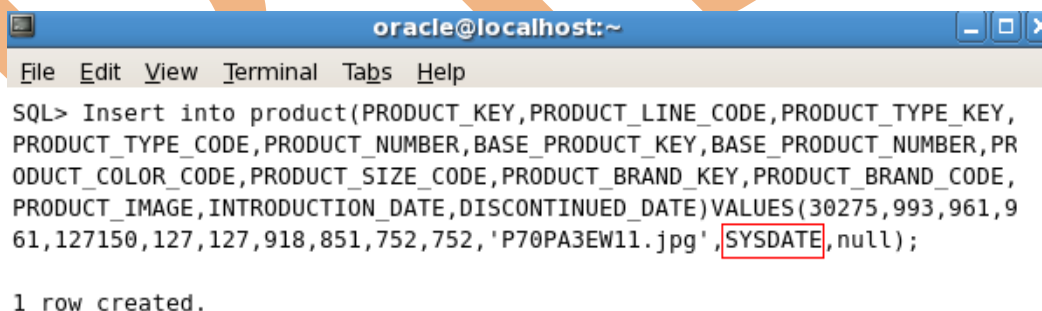


```
root@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> INSERT into organization(organization_key,organization_code1,
organization_code2,organization_code3,organization_code4,organization_code,organization_parent,organization_level) values(11225,'GOC ON','GOEUX','GOEUXOP','GOEUXOP','GOEUXOP','GOEUX',NULL);

1 row created.
```

**Figure-9.2**

- 4. Inserting Special Values:** -For inserting the special values in a table, we can use the functions.



```
oracle@localhost:~
File Edit View Terminal Tabs Help
SQL> Insert into product(PRODUCT_KEY,PRODUCT_LINE_CODE,PRODUCT_TYPE_KEY,
PRODUCT_TYPE_CODE,PRODUCT_NUMBER,BASE_PRODUCT_KEY,BASE_PRODUCT_NUMBER,PR
ODUCT_COLOR_CODE,PRODUCT_SIZE_CODE,PRODUCT_BRAND_KEY,PRODUCT_BRAND_CODE,
PRODUCT_IMAGE,INTRODUCTION_DATE,DISCONTINUED_DATE)VALUES(30275,993,961,9
61,127150,127,127,918,851,752,752,'P70PA3EW11.jpg',SYSDATE,null);

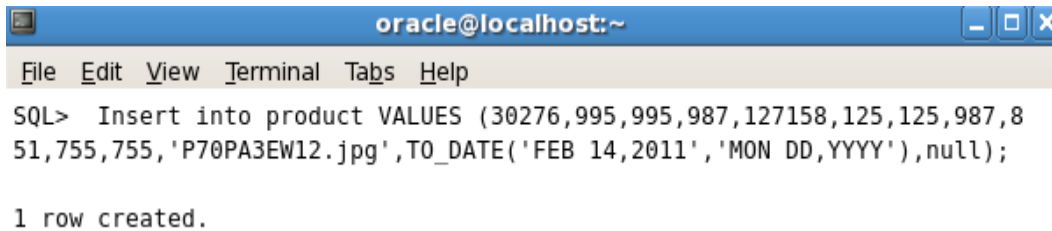
1 row created.
```

**Figure-9.3**

In the above example, we are entering data by the SYSDATE function in **introduction\_date** column of the **product** table.

- 5. Inserting specific Date Values:** -The **DD-MM-RR** format is generally used to insert a date value. With **RR** format, the system provides the correct century automatically. We may also supply the date value in the **DD-MM-YYYY** format. If a date must be

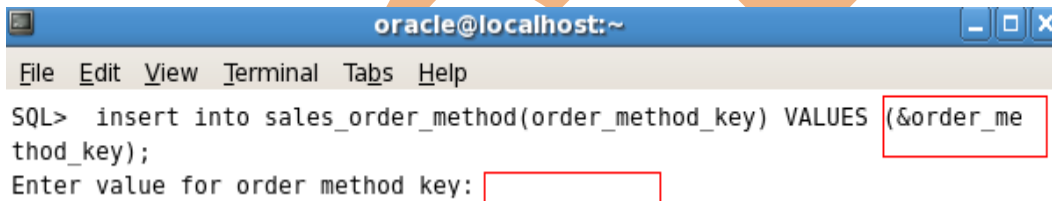
entered in a format other than the default format, we must use the **TO\_DATE** function.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> Insert into product VALUES (30276,995,995,987,127158,125,125,987,8  
51,755,755,'P70PA3EW12.jpg',TO_DATE('FEB 14,2011','MON DD,YYYY'),null);  
  
1 row created.
```

**Figure-9.4**

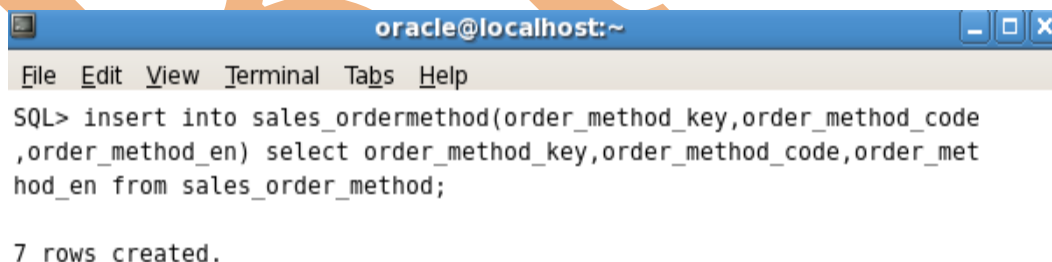
- 6. Creating a Script:** -By using the scripts, we can enter the values again and again with different values. By this we can save the time and writing the command again and again.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> insert into sales_order_method(order_method_key) VALUES (&order_me  
thod_key);  
Enter value for order method key: 
```

**Figure-9.5**

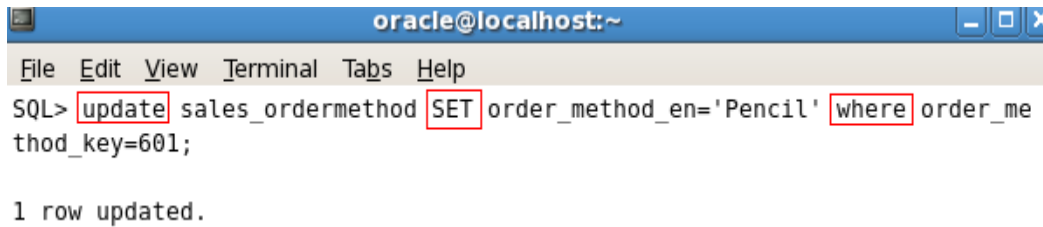
- 7. Copying Rows:** -We can use the **INSERT** statement to add rows to a table where the values are derived from existing table. SO we write the statement with sub-query and do not use the **VALUES** clause. And always match the number of columns in the INSERT clause to those in the sub-query.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> insert into sales_ordermethod(order_method_key,order_method_code  
,order_method_en) select order_method_key,order_method_code,order_met  
hod_en from sales_order_method;  
  
7 rows created.
```

**Figure-9.6**

- 8. Updating data in a table:** -We can modify the existing values in a table by using the **UPDATE** statement. Values for a specific row or rows are modified if we specify the **WHERE** clause.



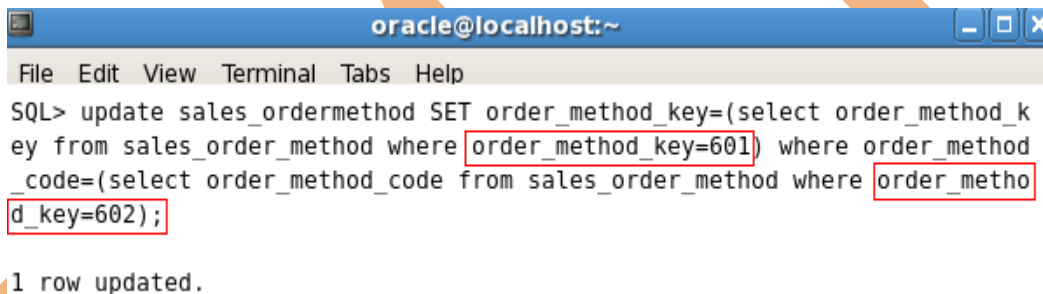
```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> update sales_ordermethod SET order_method_en='Pencil' where order_me  
thod_key=601;  
  
1 row updated.
```

**Figure-9.7**

In the above example, the value of the column **order\_method\_en** of the table **sales\_ordermethod**, whose **order\_method\_key** is **601**.

SO, the **UPDATE** statement updates the particular table and **SET** statement sets the value to the particular column. And **WHERE** statement clarify that what will be the condition for updating the values of the column of a table.

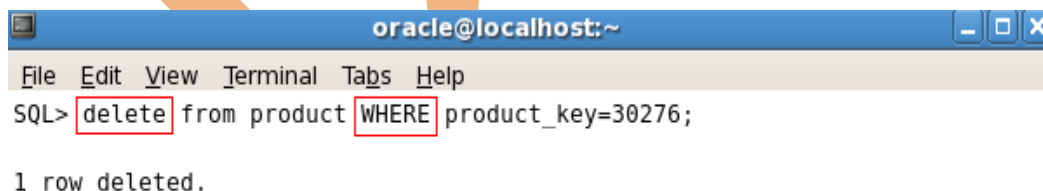
9. **Updating Rows based on another table:** -We can use sub-queries in the UPDATE statements to update values in a table.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> update sales_ordermethod SET order_method_key=(select order_method_k  
ey from sales_order_method where order_method_key=601) where order_method  
_code=(select order_method_code from sales_order_method where order_metho  
d_key=602);  
  
1 row updated.
```

**Figure-9.8**

10. **Removing rows from a table:** -We can remove existing rows from a table by using **DELETE** statement. If no rows are deleted, the message **0 rows deleted** is returned. If we specify **WHERE** clause, then specific rows are deleted.



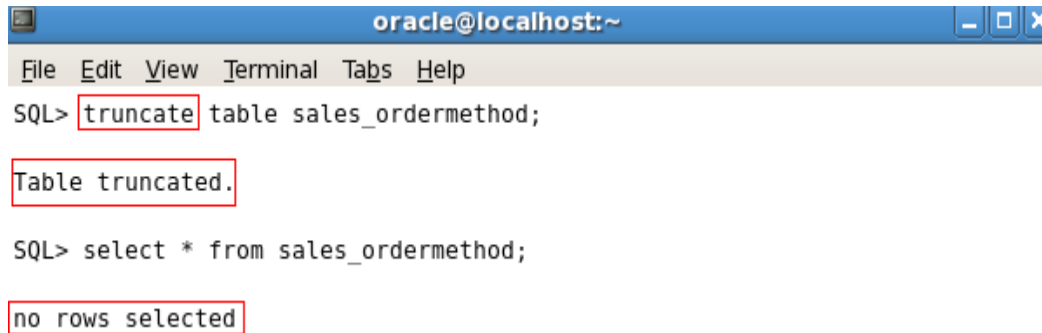
```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> delete from product WHERE product_key=30276;  
  
1 row deleted.
```

**Figure-9.9**

In the above example, the row of the product table will be removed whose **product\_key** is **30276**.

**11. Using Truncate statement:** -It removes all rows from a table and leaving the table empty and the table structure intact. The TRUNCATE statement is a data definition language statement and generates no rollback information.

Truncating a table does not fire the delete triggers of the table.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> truncate table sales_ordermethod;  
  
Table truncated.  
  
SQL> select * from sales_ordermethod;  
  
no rows selected
```

**Figure-9.10**

**12. Database Transaction:** -A database transaction consists of one of the following-

- DML statements that constitute one consistent change to the data.
- One DDL statement.
- One DCL statement.

Database transaction begins when the first DML SQL statement is executed.

End with one of the following events-

- A **COMMIT** or **ROLLBACK** statement is issued.
- A DDL or DCL statement executes.
- The user exits SQL Developer or SQL \* Plus.
- The system crashes.

After one transaction ends, the next executable statement automatically starts the next transaction.

A DDL or DCL statement is automatically committed and therefore implicitly ends a transaction.

**13. Implicit Transaction Processing:** -An automatic commit occurs in the following circumstances-

- A DDL statement is issued.
- A DCL statement is issued.
- Normal exit from SQL Developer or SQL \* Plus without COMMIT or ROLLBACK.

An automatic ROLLBACK occurs when there is an abnormal termination of SQL Developer or a system.

**Automatic Commit**-DDL or DCL statement issued and SQL Developer or SQL \* Plus exited normally, without explicitly issuing COMMIT or ROLLBACK commands.

**Automatic Rollback**-Abnormal termination of SQL Developer or SQL \* Plus or system failure.

**14. Explicit Transaction Processing:** -We can control the logic of transactions by using the COMMIT, ROLLBACK and SAVEPOINT statements.

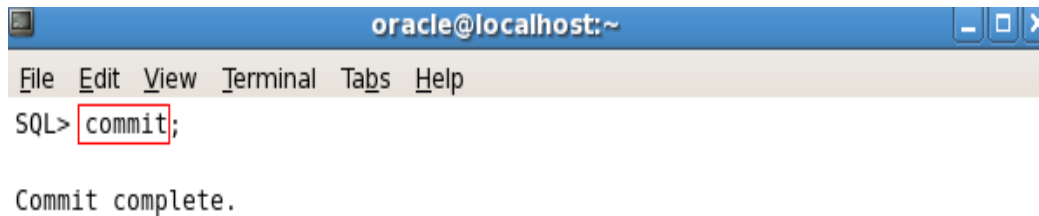
**15. State of Data before COMMIT or ROLLBACK:** -All data change made during the transaction is temporary until the transaction is committed. So these are the following states before COMMIT or ROLLBACK-

- The previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Others users cannot view the results of the data manipulation operations made by current user.
- The affected rows are locked; other users cannot change the data in affected rows.

**16. Using COMMIT Statement:** -Make all pending changes permanent by using COMMIT statement. After committing-

- Data changes are written to the database.
- The previous state of data is no longer available with normal SQL queries.
- All users can view the results of transaction.
- All savepoints are erased.
- The locks on affected rows are released.





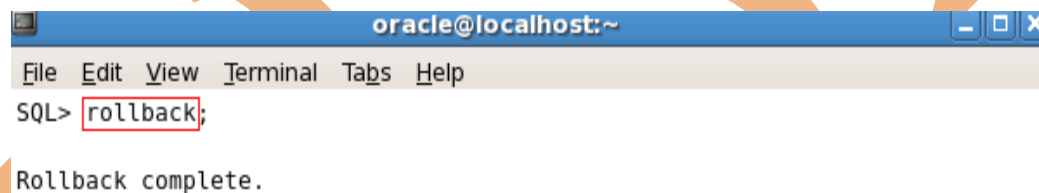
```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> commit;  
  
Commit complete.
```

**Figure-9.11**

When we execute the commit statement then all the data will be saved in the database.

**17. Using ROLLBACK Statement:** -Discards all pending changes by using the ROLLBACK statement, which results in the following-

- Data changes are undone.
- The previous state of data is restored.
- Locks on the affected rows are released.



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> rollback;  
  
Rollback complete.
```

**Figure-9.12**

**18. Statement-Level Rollback:** -A part of transaction can be discarded through an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of transaction, its effect is undone by a statement-level rollback, but changes made by previous DML statements in transaction are not discarded. They can be committed or rolled back explicitly by the user. Terminate the transactions explicitly by executing a COMMIT or ROLLBACK statement.

## Chapter-10

1. **Database Objects:** -The oracle database can contain multiple data structures.

**Table** stores the data.

**View** is the subset of data from one or more tables.

**Sequence** generates numeric values.

**Index** improves the performance of some queries.

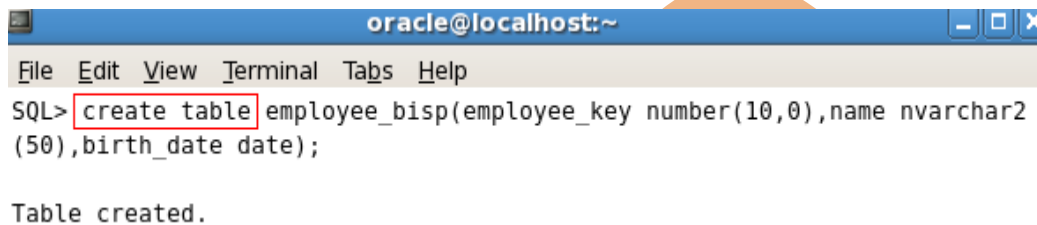
**Synonym** gives alternative name to an object.

- Tables can be created at any time, even when users are using the database.
- We do not need to specify the size of a table because the size is ultimately defined by the amount of space allocated to the database as a whole.
- Table structure can be modified online.

2. **Naming Rules:** -Names are not case-sensitive. There is some standard rules for naming the database tables and columns-

- Table names and column names must begin with a letter and must be **1-30** characters long.
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must contain only the characters **A-Z, a-z, 0-9, \_ (Underscore), \$, #**.
- Names must not be an Oracle server-reserved word.

3. **CREATE Table Statement:** -To create table, a user must have the **CREATE TABEL** privilege and a storage area in which to create objects. We create tables to store data by executing the SQL CREATE TABLE statement. The database administrator uses data control language (DCL) statements to grant privileges to users. Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed. When we define a table, we can specify that a column should be given a default value by using the DEFAULT option.

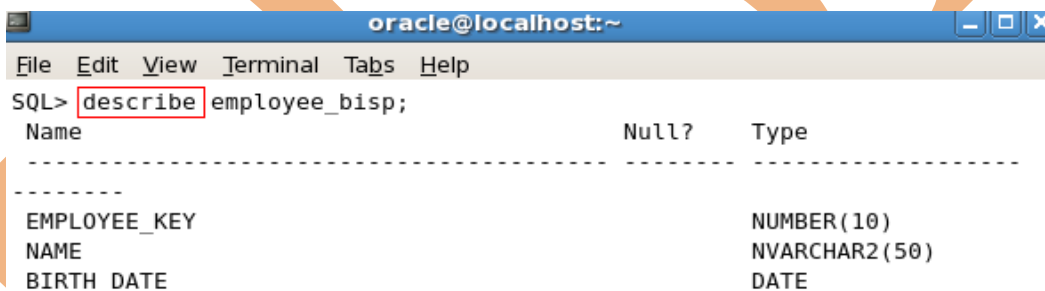


```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> create table employee_bisp(employee_key number(10,0),name nvarchar2  
(50),birth_date date);  
  
Table created.
```

**Figure-10.1**

In the above example, table is created, now we have to check that table is created or not. So execute the following statement-

**DESCRIBE Table\_Name**



```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
SQL> describe employee_bisp;  
  
Name                               Null?    Type  
-----  
EMPLOYEE_KEY                       NUMBER(10)  
NAME                               NVARCHAR2(50)  
BIRTH_DATE                         DATE
```

**Figure-10.2**

By this statement, we can view the structure of the table and can confirm that table is created.

4. **Data Types:** -When we identify a column for a table, we need to provide a data type for the column. There are different types of data types-

**VARCHAR2 (size)** Variable-length character data

**CHAR (size)** Fixed-length character data

**NUMBER (p,s)** Variable-length numeric data

<b>DATE</b>	Date and time values
<b>LONG</b>	Variable-length character data
<b>CLOB</b>	Character data
<b>RAW &amp; LONG RAW</b>	Raw binary data
<b>BLOB</b>	Binary data
<b>BFILE</b>	Binary data stored in an external file
<b>ROWID</b>	A base-64 number system represents unique address of a row.

#### Some Guideline-

- A LONG column is not copied when a table is created using a sub-query.
- A LONG column cannot be included in a GROUP BY or ORDER BY clause.
- Only one LONG column can be used per table.
- No constraint can be defined on a LONG column.

#### There are some Date-time data types-

<b>TIMESTAMP</b>	Date with fractional seconds
<b>INTERVAL YEAR TO MONTH</b>	Stored as an interval of years and months
<b>INTERVAL DAY TO SECOND</b>	Stored as an interval of days, hours, minutes and seconds.

5. **Using Constraints:** -The oracle server uses constraints to prevent invalid data entry into tables. We can use the constraints to do the following-

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table.
- Prevent the deletion of a table if there are dependencies from other tables.

There are following types of constraints-

<b>NOT NULL</b>	Specifies that the column cannot contain the null value
<b>UNIQUE</b>	Specifies a column or a combination of columns whose values must be unique for all rows in the table

<b>PRIMARY KEY</b>	Uniquely identifies each row of the table
<b>FOREIGN KEY</b>	Establishes and enforces a referential integrity between column and a column of the referenced table such that values in one table match values in another table.
<b>CHECK</b>	Specifies a condition that must be true.

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also be temporarily disabled.

6. **Using NOT NULL Constraint:** -The NOT NULL constraint ensures that column contains no null values. Columns without NOT NULL constraint can contain null values by default. NOT NULL constraint must be defined at the **column level**.
7. **Using UNIQUE Constraint:** -A UNIQUE key integrity constraint requires that every value in a column or a set of columns be unique that is no two rows of a table can have duplicate values in a specified column or a set of columns. The column or set of columns included in the definition of the UNIQUE KEY constraint is called the **unique key**. If the UNIQUE constraint comprises more than one column, the group of columns is called a **composite unique key**. Unique constraint enables the input of nulls unless we also define NOT NULL constraints for the same columns.

## Chapter-11

### 1. **Database Objects:** -There are several objects in a database-

<b>Table</b>	Basic unit of storage
<b>View</b>	Logically represents subsets of data from one or more tables
<b>Sequence</b>	Generates numeric values
<b>Index</b>	Improves the performance of data retrieval queries
<b>Synonym</b>	Gives alternative names to objects.

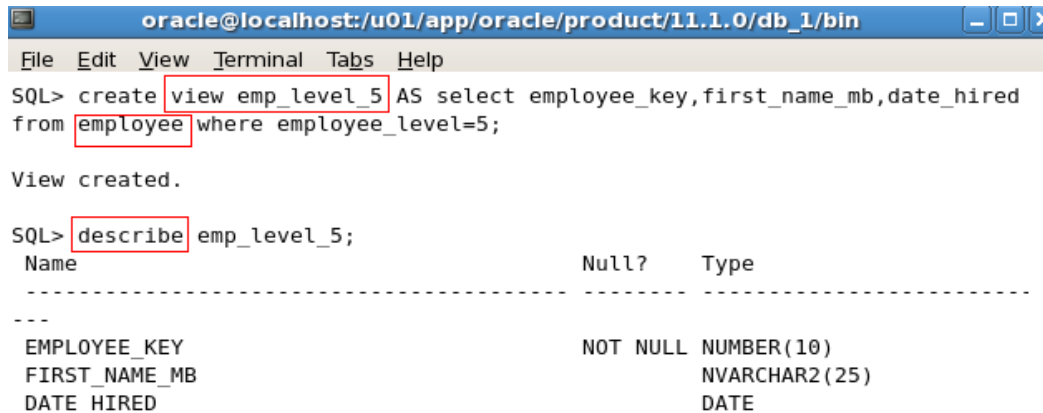
### 2. **View:** -We can present logical subsets or combinations of data by creating **views** of tables. A **view** is a logical table based on a table or another view. A view contains no data of its own, but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called **base tables**.

#### **Advantages of View-**

- Views restrict access to the data because it displays selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of user's access to data according to their particular criteria.

### 3. **Simple and Complex Views:** -Simple view derives data from only one table. It contains no functions or groups of data and can perform DML operations through the view. Whereas a complex view derives data from many tables. It contains functions or groups of data and does not always allow DML operations through the view.

4. **Creating a View:** -We can create a view by embedding a sub-query in the CREATE VIEW statement. The sub-query can contain complex SELECT syntax, including joins, groups and sub-queries.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> create view emp_level_5 AS select employee_key,first_name_mb,date_hired
from employee where employee_level=5;

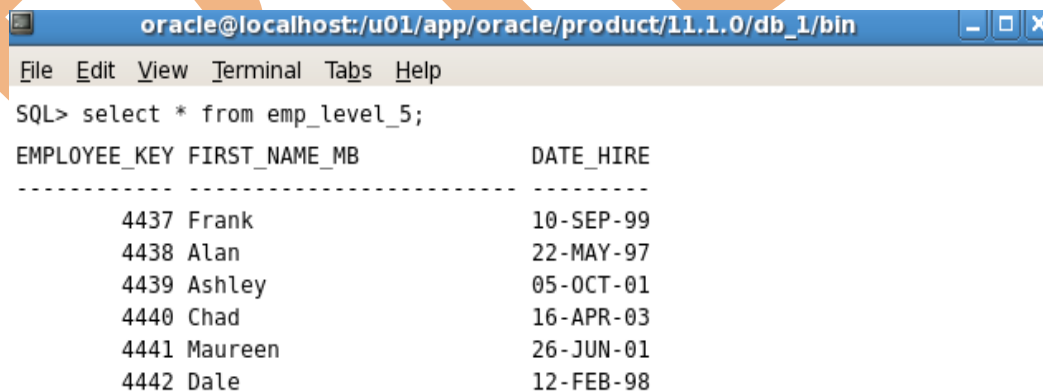
View created.

SQL> describe emp_level_5;
Name                                         Null?    Type
-----
EMPLOYEE_KEY                                NOT NULL NUMBER(10)
FIRST_NAME_MB                               NVARCHAR2(25)
DATE_HIRED                                  DATE
```

**Figure-11.1**

In the above example, we have created a view table of the **employee** table as there is selected the **employee\_key**, **first\_name\_mb**, and **date\_hired** and the view table name is **emp\_level\_5**. As there is condition, the employees who have **level 5** will be selected.

5. **Retrieving Data from a View:** -As we have created a view for a table, so now we have to retrieve the data from the view table. We can show data as how we show the data of any other table.

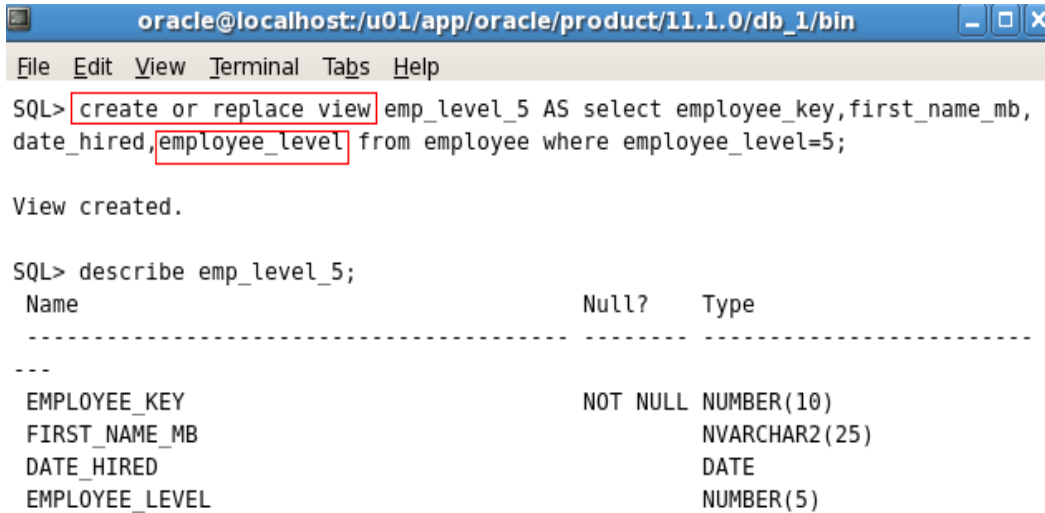


```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> select * from emp_level_5;
EMPLOYEE_KEY FIRST_NAME_MB      DATE_HIRE
-----
4437 Frank    10-SEP-99
4438 Alan     22-MAY-97
4439 Ashley   05-OCT-01
4440 Chad     16-APR-03
4441 Maureen  26-JUN-01
4442 Dale     12-FEB-98
```

**Figure-11.2**

In the above example, the employee's detail is showing whose level is 5 in the original table.

6. **Modifying a View:** -As we have created a view table and now we have to modify this table as we have to add one more column **employee\_level**. So, the following query will help to add the column and also will fetch the value of that column from the original table of that view table.



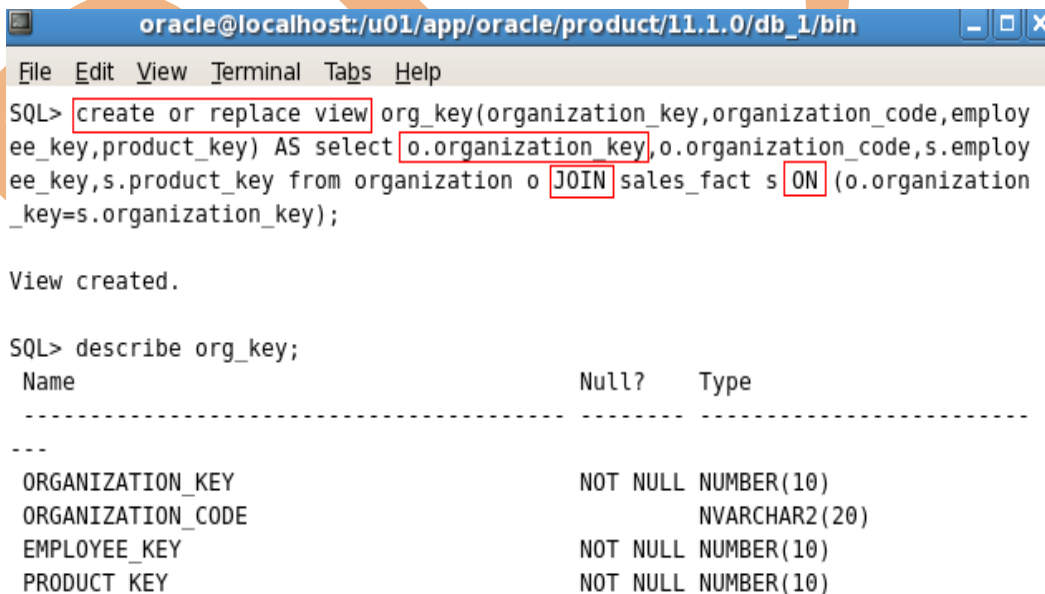
```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> create or replace view emp_level_5 AS select employee_key,first_name_mb,
date_hired,employee_level from employee where employee_level=5;

View created.

SQL> describe emp_level_5;
Name                                         Null?    Type
-----
EMPLOYEE_KEY                                NOT NULL NUMBER(10)
FIRST_NAME_MB                               NVARCHAR2(25)
DATE_HIRED                                  DATE
EMPLOYEE_LEVEL                              NUMBER(5)
```

**Figure-11.3**

7. **Creating a Complex View:** -Create a complex view that contains group functions to display values from two tables.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> create or replace view org_key(organization_key,organization_code,employ
ee_key,product_key) AS select o.organization_key,o.organization_code,s.employ
ee_key,s.product_key from organization o JOIN sales_fact s ON (o.organization
_key=s.organization_key);

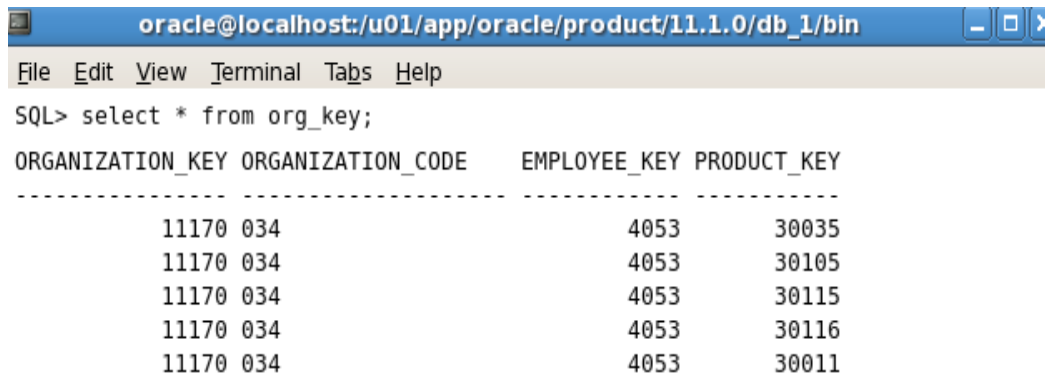
View created.

SQL> describe org_key;
Name                                         Null?    Type
-----
ORGANIZATION_KEY                                NOT NULL NUMBER(10)
ORGANIZATION_CODE                               NVARCHAR2(20)
EMPLOYEE_KEY                                    NOT NULL NUMBER(10)
PRODUCT_KEY                                    NOT NULL NUMBER(10)
```

**Figure-11.4**

In the above example, there are data from two table **organization** and **sales\_fact**.





The screenshot shows a terminal window titled 'oracle@localhost:/u01/app/oracle/product/11.1.0/db\_1/bin'. The command prompt is 'SQL> select \* from org\_key;'. The output is a table with four columns: ORGANIZATION\_KEY, ORGANIZATION\_CODE, EMPLOYEE\_KEY, and PRODUCT\_KEY. The data is as follows:

ORGANIZATION_KEY	ORGANIZATION_CODE	EMPLOYEE_KEY	PRODUCT_KEY
11170	034	4053	30035
11170	034	4053	30105
11170	034	4053	30115
11170	034	4053	30116
11170	034	4053	30011

**Figure-11.5**

In the above example, the data is retrieved from the view table **org\_key**, in which there is **organization\_key**, **organization\_code**, **employee\_key**, and **product\_key** that is fetched from the **organization** and **sales\_fact** table.

**8. Rules for performing DML operations on View:** -We can perform DML operations on data through a view if those operations follow certain rules.

(a) We can remove a row, cannot modify data, cannot add data from a view unless it contains any of the following-

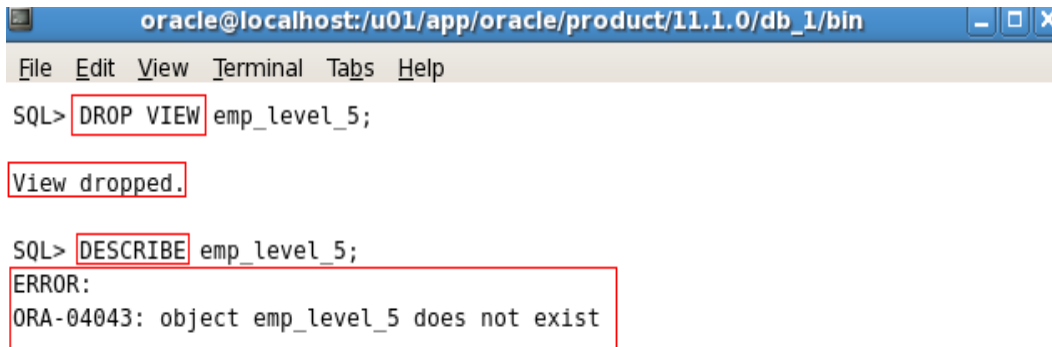
- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudo-column ROWNUM keyword
- Columns defined by expressions
- NOT NULL columns in the base tables that are not selected by the view.

**9. Using WITH CHECK OPTION Clause:** -The view can be used to protect data integrity, but the use is very limited. The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select. Therefore it enables integrity constraints and data validation checks to be enforced on data being inserted or updated. If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

**10. Denying DML Operations:** -We can ensure that no DML operations occur on our view by creating it with the WITH READ ONLY option. Any attempt to remove, insert

or remove a row using the view with a read-only constraint results in an error. And the error will be same for all these three.

- 11. Removing a View:** -We can remove a view without losing data because a view is based on underlying tables in the database. We can remove the view definition from the database by using the **DROP VIEW** statement.



```
oracle@localhost:/u01/app/oracle/product/11.1.0/db_1/bin
File Edit View Terminal Tabs Help
SQL> DROP VIEW emp_level_5;
View dropped.
SQL> DESCRIBE emp_level_5;
ERROR:
ORA-04043: object emp_level_5 does not exist
```

**Figure-11.6**

In the above example, there were a view table **emp\_level\_5** that is deleted by the **DROP** statement and for verifying that table is deleted or not just follow the **DESCRIBE** statement.

- 12. Sequence:** -A sequence is user-created database object that creates integer values. We can create sequences and then use them to generate numbers. We can define a sequence to generate unique values and use the same numbers again. A sequence is generated and incremented or decremented by an internal Oracle routine. Sequence numbers are stored and generated independent of tables. Therefore, the same sequence can be used for multiple tables.