

# Lab 1: Oracle DDL and User Administration

Date: 19/09/2025

## Objectives

1. Create parent and child table relationships.
  2. Verify the tables created.
  3. Create and manage non-privileged users.
- 

## Tools

- Oracle Database XE
- SQL Developer
- Docker

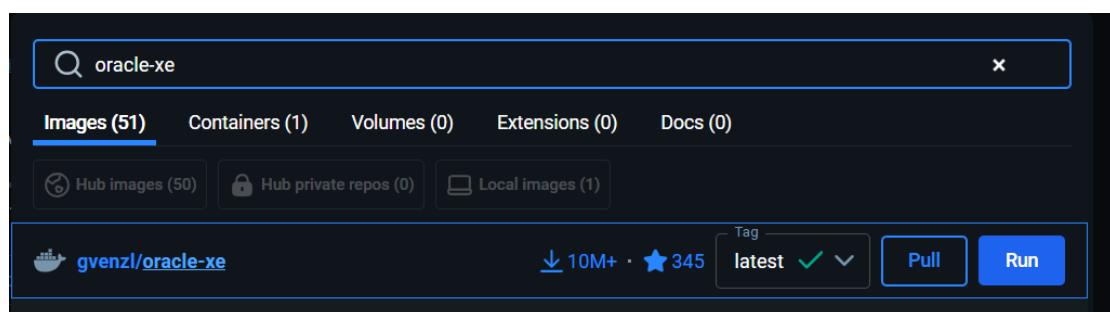
## Lab Users:

- SYS (Admin user)
  - sandesh\_csit (Lab user)
- 

## Steps

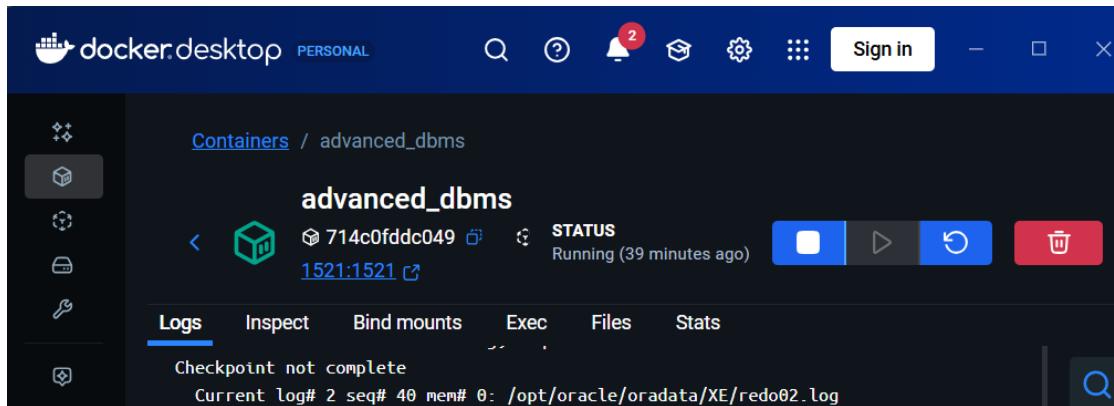
### 1. Access Docker Container

Open docker desktop and pull and run oracle



Open terminal and enter the Oracle container:

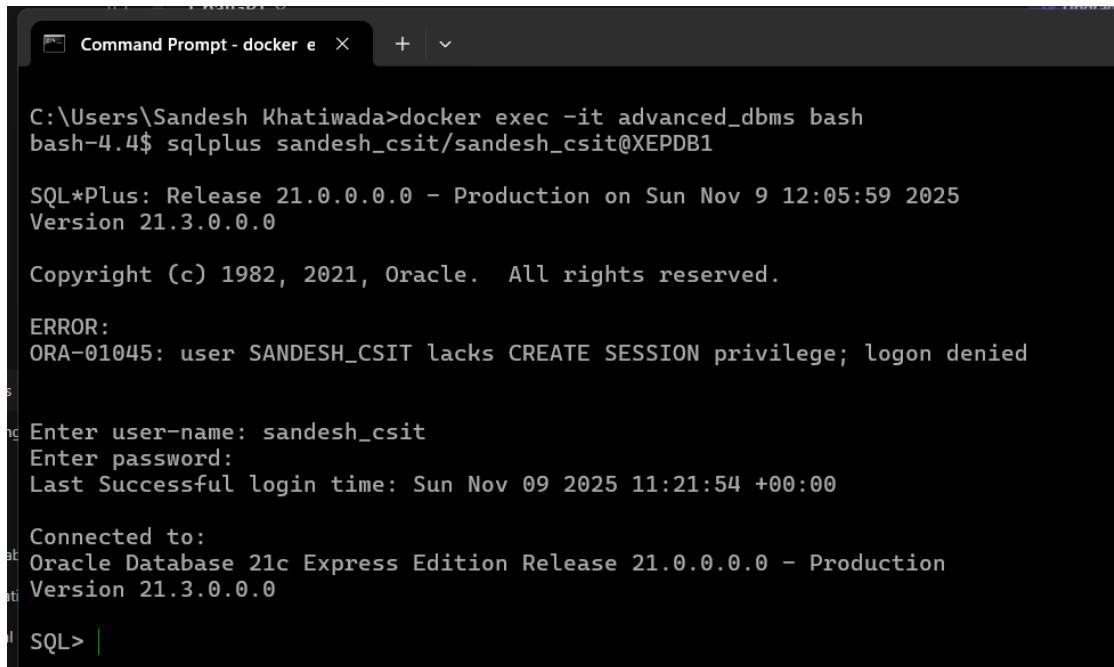
```
docker exec -it advanced_dbms bash
```



## 2. Connect as SYSDBA

Connect to Oracle using SQL\*Plus as SYSDBA:

```
sqlplus sys/oracle@XE as sysdba
```



```
C:\Users\Sandesh Khatiwada>docker exec -it advanced_dbms bash
bash-4.4$ sqlplus sandesh_csit/sandesh_csit@XEPDB1
SQL*Plus: Release 21.0.0.0.0 - Production on Sun Nov 9 12:05:59 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

ERROR:
ORA-01045: user SANDESH_CSIT lacks CREATE SESSION privilege; logon denied

Enter user-name: sandesh_csit
Enter password:
Last Successful login time: Sun Nov 09 2025 11:21:54 +00:00

Connected to:
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

## 3. Switch to the Pluggable Database

```
ALTER SESSION SET CONTAINER = XEPDB1;
```

```
SHOW CON_NAME;
```

```
-- Should display: XEPDB1
```

```
SHOW PDBS;  
-- Should display:  
-- PDB$SEED  
-- XEPDB1  
-- FREEPDB1
```

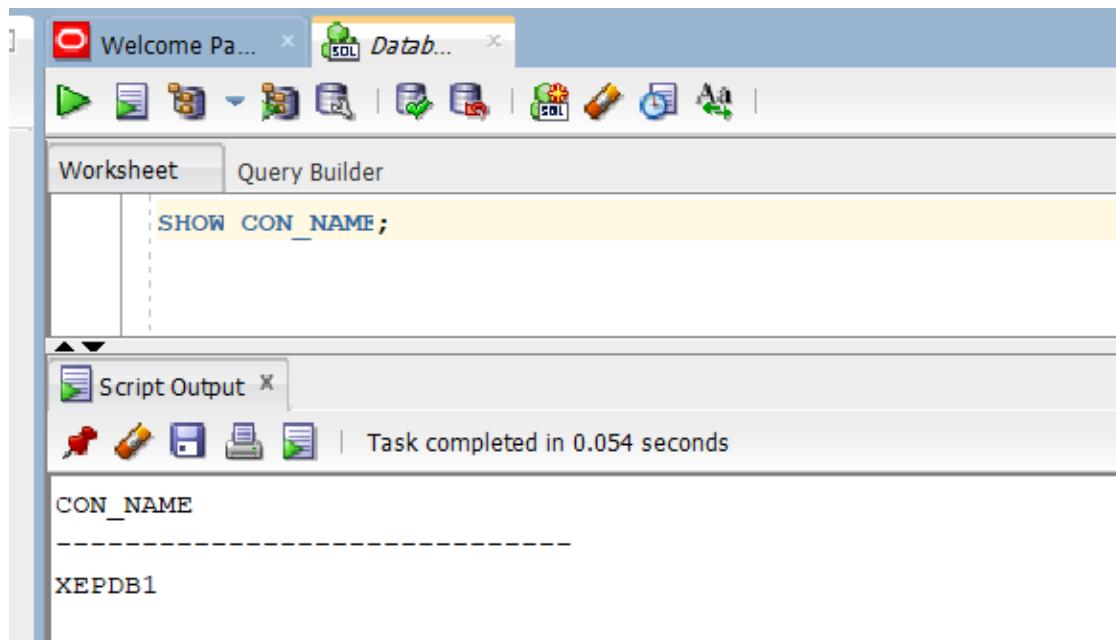
The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there is a 'Welcome Page' tab and a 'Database...' tab. Below the tabs is a toolbar with various icons. The main area has two tabs: 'Worksheet' and 'Query Builder'. The 'Worksheet' tab is selected, and its content pane contains the SQL command: 'ALTER SESSION SET CONTAINER=XEPDB1'. Below this, the 'Script Output' tab is selected, showing the results of the command. The output includes:

```
CON_NAME
-----
CDB$ROOT

  CON_ID CON_NAME          OPEN MODE RESTRICTED
  -----
        2 PDB$SEED      READ ONLY NO
        3 XEPDB1       READ WRITE NO

Session altered.
```

The output indicates that the session has been altered to container XEPDB1, and it lists the current sessions in the CDB\$ROOT container, showing PDB\$SEED and XEPDB1.



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a 'Welcome Pa...' tab and a 'Database...' tab. The main area is a 'Worksheet' tab where the SQL command 'SHOW CON\_NAME;' is entered. Below the worksheet, the 'Script Output' tab displays the result: 'CON\_NAME' followed by a dashed line and 'XEPDB1'. The status bar at the bottom indicates 'Task completed in 0.054 seconds'.

```
SHOW CON_NAME;
```

CON\_NAME

XEPDB1

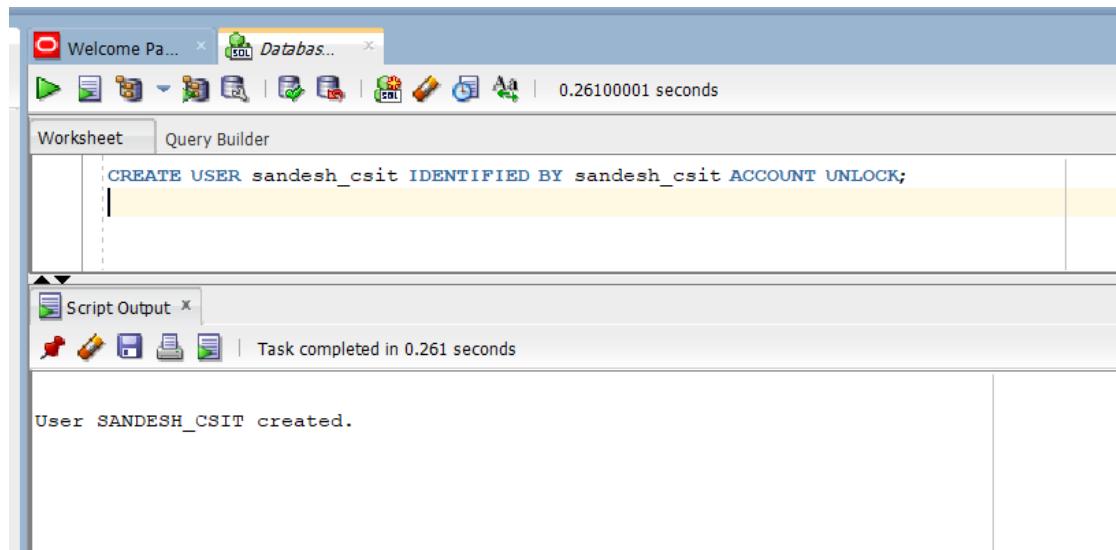
#### 4. Create Lab User

Create a new user sandesh\_csit with password sandesh\_csit:

```
CREATE USER sandesh_csit IDENTIFIED BY sandesh_csit ACCOUNT UNLOCK;
```

Grant necessary privileges:

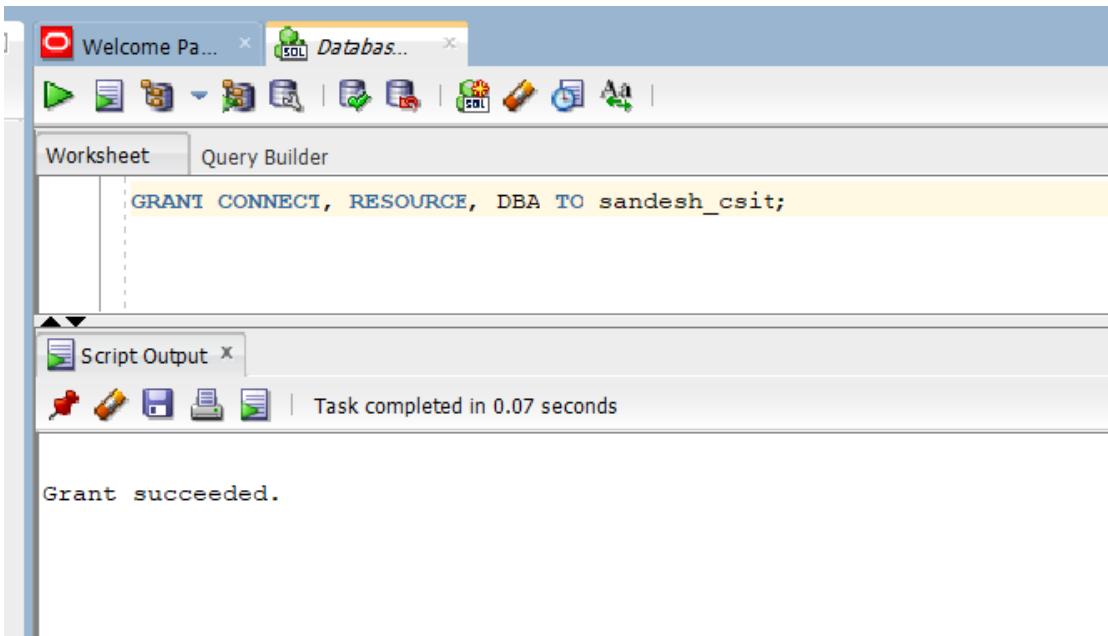
```
GRANT CONNECT, RESOURCE, DBA TO sandesh_csit;
```



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a 'Welcome Pa...' tab and a 'Database...' tab. The main area is a 'Worksheet' tab where the SQL command 'CREATE USER sandesh\_csit IDENTIFIED BY sandesh\_csit ACCOUNT UNLOCK;' is entered. Below the worksheet, the 'Script Output' tab displays the result: 'User SANDESH\_CSIT created.' The status bar at the bottom indicates 'Task completed in 0.261 seconds'.

```
CREATE USER sandesh_csit IDENTIFIED BY sandesh_csit ACCOUNT UNLOCK;
```

User SANDESH\_CSIT created.



The screenshot shows the Oracle SQL Developer interface. In the top-left window, a query is being run:

```
GRANT CONNECT, RESOURCE, DBA TO sandesh_csit;
```

In the bottom-right window, the output is displayed:

```
Grant succeeded.
```

## 5. Connect as Lab User

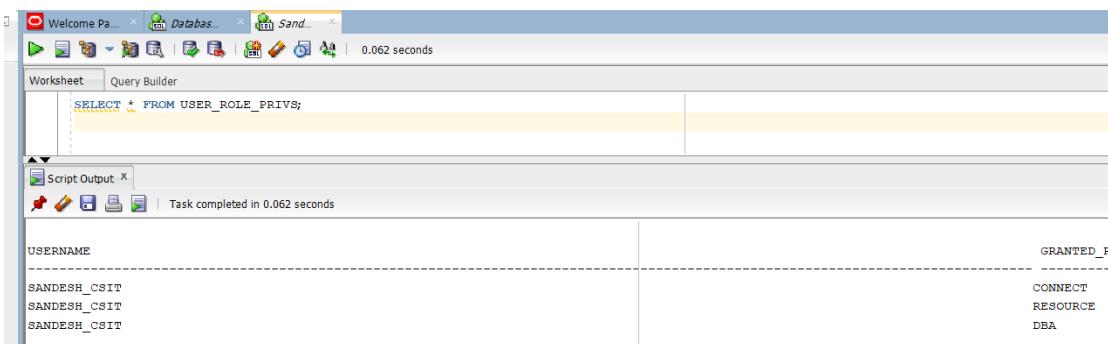
Open a new session as sandesh\_csit:

```
sqlplus sandesh_csit/sandesh_csit@XEPDB1
```

```
SHOW USER;
```

You should see:

USER is "SANDESH\_CSIT"



The screenshot shows the Oracle SQL Developer interface. A query is run in the top-left window:

```
SELECT * FROM USER_ROLE_PRIVS;
```

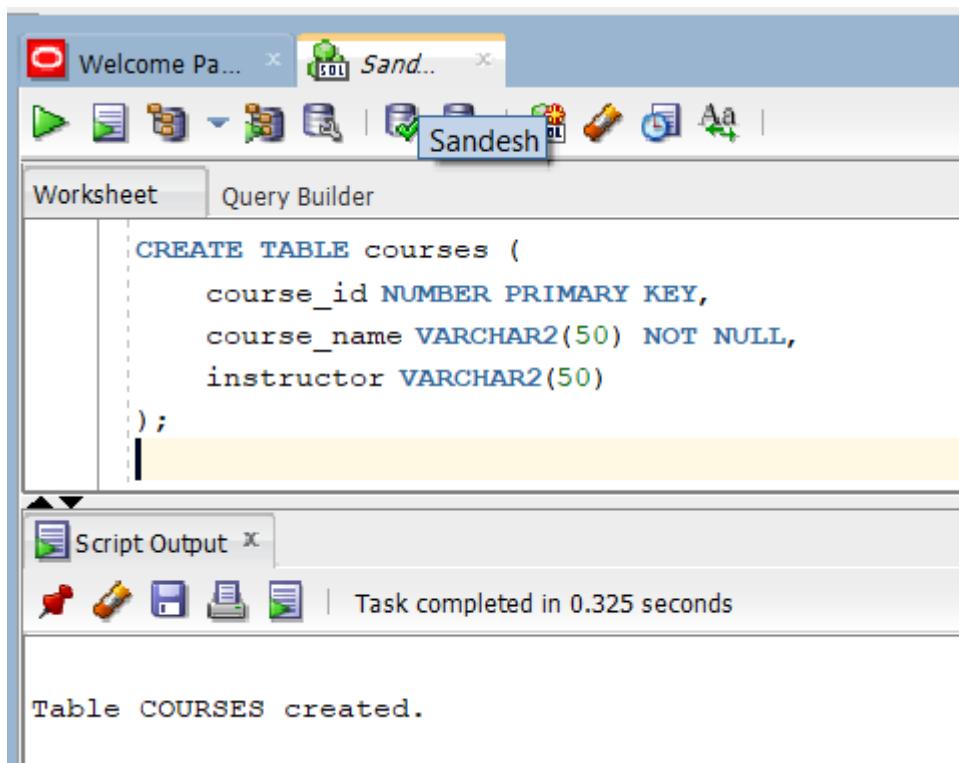
The output is shown in the bottom-right window:

USERNAME	GRANTED_R
SANDESH_CSIT	CONNECT
SANDESH_CSIT	RESOURCE
SANDESH_CSIT	DBA

## 6. Create Parent and Child Tables

### Parent Table: COURSES

```
CREATE TABLE courses (
    course_id NUMBER PRIMARY KEY,
    course_name VARCHAR2(50) NOT NULL,
    instructor VARCHAR2(50)
);
```



The screenshot shows the Oracle SQL Developer interface. The top bar has tabs for 'Welcome Pa...' and 'Sand...'. The main window is titled 'Worksheet' and contains the SQL code for creating the 'courses' table. Below the worksheet is a 'Script Output' window showing the successful creation of the table.

```
CREATE TABLE courses (
    course_id NUMBER PRIMARY KEY,
    course_name VARCHAR2(50) NOT NULL,
    instructor VARCHAR2(50)
);
```

Script Output

```
Table COURSES created.
```

## Child Table: STUDENTS

```
CREATE TABLE students (
    student_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(30),
    last_name VARCHAR2(30),
    course_id NUMBER,
    CONSTRAINT fk_course
        FOREIGN KEY (course_id)
        REFERENCES courses(course_id)
);
```

The screenshot shows the Oracle SQL Developer interface. The top bar has tabs for 'Welcome Pa...' and 'Sand...'. The main window has tabs for 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The code area contains the SQL script for creating the STUDENTS table. Below the code is a 'Script Output' window showing the execution results.

```
--child table students
CREATE TABLE students (
    student_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(30),
    last_name VARCHAR2(30),
    course_id NUMBER,
    CONSTRAINT fk_course
        FOREIGN KEY (course_id)
        REFERENCES courses(course_id)
);
```

Script Output

Task completed in 0.089 seconds

Table STUDENTS created.

## 7. Modify Tables (ALTER) and Delete Tables

Alter Parent Table: Add a new column credits to courses

ALTER TABLE courses

ADD credits NUMBER(2);

The screenshot shows the Oracle SQL Developer interface. The top bar has tabs for 'Welcome Pa...' and 'Sand...'. Below the bar are various icons. The main area has two tabs: 'Worksheet' (selected) and 'Query Builder'. In the 'Worksheet' tab, there is a code editor containing the following SQL statement:

```
--alter parent's table
ALTER TABLE courses
ADD credits NUMBER(2);
```

Below the code editor is a 'Script Output' window. It shows the message 'Task completed in 0.243 seconds' and the output of the executed query:

```
Table COURSES altered.
```

Alter Child Table: Add a new column email to students

ALTER TABLE students

ADD email VARCHAR2(50);

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'Welcome Pa...' and 'Sand...'. Below the tabs is a toolbar with various icons. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The worksheet pane contains the following SQL code:

```
--Alter Parent Table: Add a new column credits to courses
ALTER TABLE courses
ADD credits NUMBER(2);
```

Below the worksheet is a 'Script Output' pane with the following message:

```
Table STUDENTS altered.
```

Modify Column Type: Change email length

```
ALTER TABLE students
MODIFY email VARCHAR2(100);
```

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'Welcome Pa...' and 'Sand...'. Below the tabs is a toolbar with various icons. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The worksheet pane contains the following SQL code:

```
--Modify Column Type: Change email length
ALTER TABLE students
MODIFY email VARCHAR2(100);
```

Below the worksheet is a 'Script Output' pane with the following message:

```
Table STUDENTS altered.
```

At the bottom of the interface, there is another 'Script Output' pane with the same message:

```
Table STUDENTS altered.
```

Drop Column: Remove email column if not needed

ALTER TABLE students

DROP COLUMN email;

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Welcome Pa...' and 'Sand...'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
--Modify Column Type: Change email length
ALTER TABLE students
MODIFY email VARCHAR2(100);
```

Below the workspace is a 'Script Output' window showing the results of the execution:

```
Table STUDENTS altered.
```

Delete (Drop) Tables:

DROP TABLE students;

DROP TABLE courses;

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Welcome Pa...' and 'Sand...'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
--Delete (Drop) Tables:
DROP TABLE students;
DROP TABLE courses;
```

Below the workspace is a 'Script Output' window showing the results of the execution:

```
Table STUDENTS dropped.

Table COURSES dropped.
```

## Conclusion

In this lab, we have successfully:

- Created a non-privileged user (sandesh\_esit) with appropriate privileges.
- Created parent (courses) and child (students) tables with a foreign key relationship.
- Practiced altering tables: adding columns, modifying column types, and dropping columns.
- Practiced deleting tables using DROP TABLE.
- Reinforced basic DDL commands (CREATE, ALTER, DROP) and user administration.

# **Lab 2: Oracle DML Operations – Insert, Update, Select, Delete**

**Date:** 02/11/2025

**Database Lab**

**Objectives:**

To be able to insert a new record in a parent table.

To be able to verify the records inserted.

To be able to verify domain constraints and referential integrity.

To be able to update and delete records.

---

## **Tools Used**

Oracle Database XE

SQL Developer

Docker

---

### **1. INSERT Operations**

**General Syntax:**

```
INSERT INTO table_name (column1, column2, ...)  
VALUES (value1, value2, ...);
```

**Example:**

Insert records into parent and child tables.

Parent table: COURSES

```
INSERT INTO courses (course_id, course_name, instructor)
```

```
VALUES (101, 'Database Systems', 'Dr. Smith');
```

Child table: STUDENTS

```
INSERT INTO students (student_id, student_name, course_id)
VALUES (1, 'Alice Johnson', 101);
```

```
INSERT INTO students (student_id, student_name, course_id)
VALUES (2, 'Bob Williams', 101);
```

```
COMMIT;
```

The screenshot shows the SSMS interface with two main windows:

- Worksheet Window:** Displays the T-SQL script used to insert data into the COURSES and STUDENTS tables. The script includes comments indicating the parent table (COURSES) and child table (STUDENTS). It shows three INSERT statements: one for the COURSES table with values (101, 'Database Systems', 'Dr. Smith'), and two for the STUDENTS table with values (1, 'Alice Johnson', 101) and (2, 'Bob Williams', 101). A COMMIT; statement is also present at the end of the script.
- Script Output Window:** Displays the results of the executed script. It shows three messages: "1 row inserted." corresponding to each of the two STUDENTS insert statements, and "Commit complete." at the end.

## 2. SELECT Operations (Verification)

### General Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

### Example:

Verify that the records were inserted successfully.

Verify parent table

```
SELECT * FROM courses WHERE course_id = 101;
```

The screenshot shows the SSMS interface with two panes. The top pane is titled 'Worksheet' and contains the following text:

```
--2. Select operation
-- Verify parent table
SELECT * FROM courses WHERE course_id = 101;
```

The bottom pane is titled 'Query Result' and displays the results of the executed query:

	COURSE_ID	COURSE_NAME	INSTRUCTOR
1	101	Database Syst...	Dr. Smith

Verify child table

```
SELECT * FROM students WHERE course_id = 101;
```

SQL Workbench interface showing a query in the Worksheet tab:

```
-- Verify child table
SELECT * FROM students WHERE course_id = 101;
```

The Query Result shows two rows:

	STUDENT_ID	STUDENT_NAME	COURSE_ID
1	Alice Johnson	101	
2	Bob Williams	101	

Verify parent-child relationship

```
SELECT s.student_id, s.student_name, c.course_name, c.instructor
FROM students s
JOIN courses c ON s.course_id = c.course_id;
```

SQL Workbench interface showing a query in the Worksheet tab:

```
-- Verify parent-child relationship
SELECT s.student_id, s.student_name, c.course_name, c.instructor
FROM students s
JOIN courses c ON s.course_id = c.course_id;
```

The Query Result shows two rows:

	STUDENT_ID	STUDENT_NAME	COURSE_NAME	INSTRUCTOR
1	Alice Johnson	Database Syst...	Dr. Smith	
2	Bob Williams	Database Syst...	Dr. Smith	

### 3. Constraint Verification

#### Domain Constraint Check:

Try inserting invalid data (violating NOT NULL constraint)

```
INSERT INTO courses (course_id, course_name, instructor)
VALUES (NULL, 'Invalid Course', 'Dr. John'); -- Should fail
```

The screenshot shows the Oracle SQL Developer interface. In the 'Worksheet' tab, the following SQL code is entered:

```
--referential integrity
INSERT INTO courses (course_id, course_name, instructor)
VALUES (NULL, 'Invalid Course', 'Dr. John'); -- Should fail
```

In the 'Script Output' tab, the error message is displayed:

```
Error starting at line : 63 in command -
INSERT INTO courses (course_id, course_name, instructor)
VALUES (NULL, 'Invalid Course', 'Dr. John')
Error at Command Line : 64 Column : 9
Error report -
SQL Error: ORA-01400: cannot insert NULL into ("SANDESH_CSIT"."COURSES"."COURSE_ID")
01400. 00000 - "cannot insert NULL into (%s)"
*Cause: An attempt was made to insert NULL into previously listed objects.
*Action: These objects cannot accept NULL values.
```

### Referential Integrity Check:

Try inserting student in non-existent course

```
INSERT INTO students (student_id, student_name, course_id)
VALUES (3, 'Charlie Brown', 999); -- Should fail (no such course_id)
```

The screenshot shows the Oracle SQL Developer interface. In the 'Worksheet' tab, the following SQL code is entered:

```
--Referential Integrity Checkk
INSERT INTO students (student_id, student_name, course_id)
VALUES (3, 'Charlie Brown', 999); -- Should fail (no such course_id)
```

In the 'Script Output' tab, the error message is displayed:

```
Error starting at line : 69 in command -
INSERT INTO students (student_id, student_name, course_id)
VALUES (3, 'Charlie Brown', 999)
Error report -
ORA-02291: integrity constraint (SANDESH_CSIT.FK.Course) violated - parent key not found
```

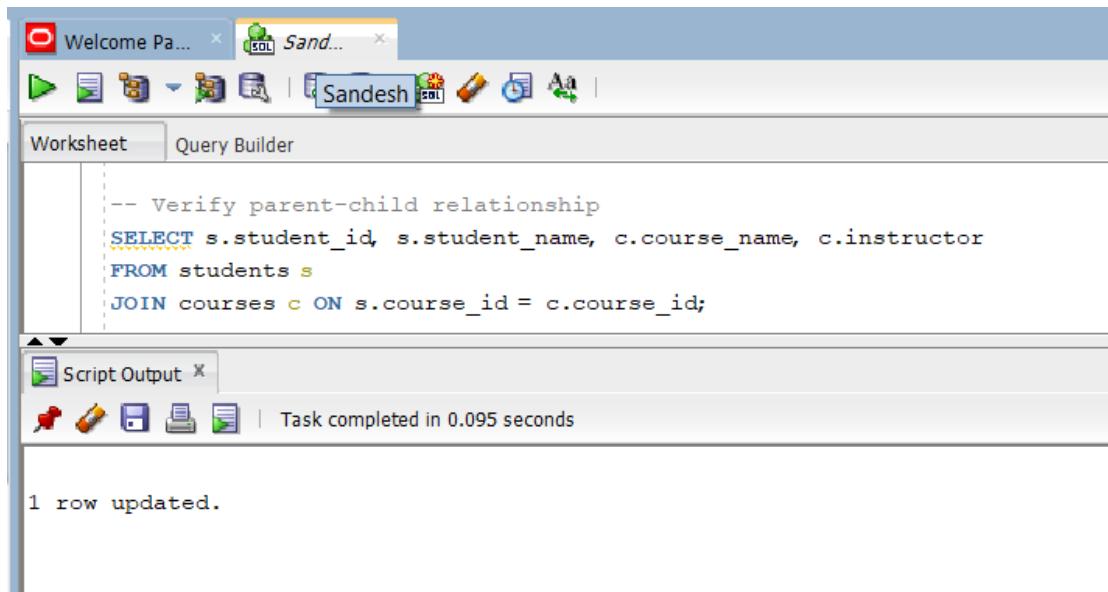
## 4. UPDATE Operations

### General Syntax:

```
UPDATE table_name  
SET column1 = new_value1, column2 = new_value2, ...  
WHERE condition;
```

### Example:

```
Update course instructor  
UPDATE courses  
SET instructor = 'Dr. Adams'  
WHERE course_id = 101;
```



The screenshot shows the SSMS interface with two windows open. The top window is titled 'Worksheet' and contains the following SQL code:

```
-- Verify parent-child relationship  
SELECT s.student_id, s.student_name, c.course_name, c.instructor  
FROM students s  
JOIN courses c ON s.course_id = c.course_id;
```

The bottom window is titled 'Script Output' and displays the result of the execution:

```
1 row updated.
```

```
Update student name  
UPDATE students  
SET student_name = 'Alice Parker'  
WHERE student_id = 1;  
  
COMMIT;
```

The screenshot shows the MySQL Workbench interface. The top window is titled 'Sandesh' and contains a SQL worksheet. The worksheet has tabs for 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The code in the worksheet is:

```
--Update student name
UPDATE students
SET student_name = 'Alice Parker'
WHERE student_id = 1;

COMMIT;
```

Below the worksheet is a 'Script Output' window. It shows the results of the query execution:

```
1 row updated.

Commit complete.
```

## 5. DELETE Operations

### General Syntax:

```
DELETE FROM table_name
WHERE condition;
```

### Example:

Delete child records first

```
DELETE FROM students
WHERE course_id = 101;
```

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'Welcome Pa...' and 'Sand...'. Below the tabs is a toolbar with various icons. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The code in the Worksheet tab is:

```
--Delete child records first
DELETE FROM students
WHERE course_id = 101;
```

In the bottom panel, there is a 'Script Output' tab with the message 'Task completed in 0.078 seconds'. Below this, the output text is:

```
2 rows deleted.
```

Then delete parent record

```
DELETE FROM courses
```

```
WHERE course_id = 101;
```

```
COMMIT;
```

## Conclusion

In this lab, we practiced **DML operations** including inserting, updating, selecting, and deleting records in related tables (**courses** and **students**). We also verified **domain constraints** and **referential integrity**, ensuring that dependent records maintain consistency when performing modifications.

-- LAB #3: SELECT STATEMENT (QUERY)

-- Student: Sandesh Khatiwada (Sandeshcsit)

-- Date: 09/11/2025

-- OBJECTIVES:

- 1. Fetch data from a single table
- 2. Fetch data conditionally [WHERE]
- 3. Use IN, BETWEEN, AND, OR, NOT operations
- 4. Apply aggregation (SUM, MIN, MAX, AVG, COUNT, STDDEV)
- 5. Apply JOINs (CROSS, INNER, OUTER)
- 6. Arrange data (ORDER BY) and fetch limited records

-- OBJECTIVE 1: FETCH DATA FROM A SINGLE TABLE

-- General Syntax:

-- SELECT \* FROM <table\_name>;

-- 1. Select all employees

SELECT \* FROM employees;

-- 2. Select all departments

SELECT \* FROM departments;

-- 3. Find employee first name, last name, and email from employees

SELECT first\_name, last\_name, email FROM employees;

-- 4. Find department name and location ID from departments

SELECT department\_name, location\_id FROM departments;

-- =====

-- OBJECTIVE 2: FETCH DATA CONDITIONALLY [WHERE]

-- =====

-- General Syntax:

```
-- SELECT * FROM <table_name> WHERE <condition>;
```

-- 5. Find all employees who work in department 60 (IT)

```
SELECT * FROM employees  
WHERE department_id = 60;
```

-- 6. Find list of all employees hired before 1st January 2006

```
SELECT * FROM employees  
WHERE hire_date < TO_DATE('01/01/2006', 'DD/MM/YYYY');
```

-- 7. Find employees with salary greater than 10000

```
SELECT * FROM employees  
WHERE salary > 10000;
```

-- =====

-- OBJECTIVE 3: IN, BETWEEN, AND, OR, NOT OPERATIONS

-- =====

-- 8. AND Operator - Find employees in IT dept (60) with salary > 5000

```
SELECT * FROM employees  
WHERE department_id = 60 AND salary > 5000;
```

-- 9. OR Operator - Find employees in department 50 OR 60

```
SELECT * FROM employees  
WHERE department_id = 50 OR department_id = 60;
```

-- 10. NOT Operator - Find employees NOT in department 90

```
SELECT * FROM employees  
WHERE NOT department_id = 90;
```

-- General Syntax for BETWEEN:

```
-- SELECT * FROM <table_name> WHERE <col_name> BETWEEN value1 AND  
value2;
```

-- 11. BETWEEN - Find employees with salary between 5000 and 10000

SELECT \* FROM employees

WHERE salary BETWEEN 5000 AND 10000;

-- 12. BETWEEN with dates - Find employees hired between 2005 and 2007

SELECT \* FROM employees

WHERE hire\_date BETWEEN TO\_DATE('01/01/2005', 'DD/MM/YYYY')

AND TO\_DATE('31/12/2007', 'DD/MM/YYYY');

-- 13. NOT BETWEEN - Find employees with salary NOT between 3000 and 7000

SELECT \* FROM employees

WHERE salary NOT BETWEEN 3000 AND 7000;

-- General Syntax for IN:

-- SELECT \* FROM <table\_name> WHERE <col\_name> IN (list\_of\_values);

-- 14. IN Operator - Find employees in departments 30, 50, or 60

SELECT \* FROM employees

WHERE department\_id IN (30, 50, 60);

-- 15. IN Operator with job titles

SELECT \* FROM employees

WHERE job\_id IN ('IT\_PROG', 'FI\_ACCOUNT', 'SA\_MAN');

-- 16. NOT IN - Find employees NOT in departments 50 or 90

SELECT \* FROM employees

WHERE department\_id NOT IN (50, 90);

-- =====

-- OBJECTIVE 4: AGGREGATION FUNCTIONS

-- =====

-- General Syntax:

```
-- SELECT SUM(<col>) FROM <table_name>;
-- SELECT MIN(<col>) FROM <table_name>;
-- SELECT MAX(<col>) FROM <table_name>;
-- SELECT AVG(<col>) FROM <table_name>;
-- SELECT COUNT(*) FROM <table_name>;
-- SELECT STDDEV(<col>) FROM <table_name>;
```

-- 17. Find total number of employees

```
SELECT COUNT(*) AS "Total Employees"
FROM employees;
```

-- 18. Find total salary (SUM)

```
SELECT SUM(salary) AS "Total Salary"
FROM employees;
```

-- 19. Find minimum salary

```
SELECT MIN(salary) AS "Minimum Salary"
FROM employees;
```

-- 20. Find maximum salary

```
SELECT MAX(salary) AS "Maximum Salary"
FROM employees;
```

-- 21. Find average salary

```
SELECT AVG(salary) AS "Average Salary"
FROM employees;
```

-- 22. Find standard deviation of salaries

```
SELECT STDDEV(salary) AS "Salary Std Deviation"
FROM employees;
```

-- 23. Find total salary by department (GROUP BY)

```
SELECT
    department_id,
```

```
COUNT(*) AS "Employee Count",
SUM(salary) AS "Total Salary",
AVG(salary) AS "Average Salary"
FROM employees
GROUP BY department_id
ORDER BY department_id;
```

```
-- =====
-- OBJECTIVE 5: JOINS (CROSS, INNER, OUTER)
-- =====
```

```
-- CROSS JOIN - General Syntax:
```

```
-- SELECT * FROM table1, table2;
```

```
-- 24. Cross Join - Cartesian product
```

```
SELECT * FROM employees, departments;
```

```
-- CROSS JOIN with WHERE (Old Style Join):
```

```
-- SELECT * FROM table1, table2 WHERE <condition>;
```

```
-- 25. Cross Join with WHERE clause
```

```
SELECT
```

```
    e.employee_id,
    e.first_name,
    e.last_name,
    d.department_name
```

```
FROM employees e, departments d
```

```
WHERE e.department_id = d.department_id;
```

```
-- INNER JOIN - General Syntax:
```

```
-- SELECT * FROM table1 t1
```

```
-- INNER JOIN table2 t2
```

```
-- ON t1.column = t2.column;
```

-- 26. Inner Join - Employees with department names

SELECT

```
e.employee_id,  
e.first_name,  
e.last_name,  
e.salary,  
d.department_name
```

FROM employees e

INNER JOIN departments d

ON e.department\_id = d.department\_id;

-- 27. Inner Join - Employees with job titles

SELECT

```
e.employee_id,  
e.first_name || ' ' || e.last_name AS "Full Name",  
j.job_title,  
e.salary
```

FROM employees e

INNER JOIN jobs j

ON e.job\_id = j.job\_id;

-- 28. Multiple INNER JOINs - Employees with department and location

SELECT

```
e.employee_id,  
e.first_name || ' ' || e.last_name AS "Employee Name",  
d.department_name,  
l.city,  
c.country_name
```

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

INNER JOIN locations l ON d.location\_id = l.location\_id

INNER JOIN countries c ON l.country\_id = c.country\_id;

-- LEFT OUTER JOIN - General Syntax:

```
-- SELECT * FROM table1 t1
-- LEFT OUTER JOIN table2 t2
-- ON t1.column = t2.column;

-- 29. Left Outer Join - All departments with employee count
SELECT
    d.department_id,
    d.department_name,
    COUNT(e.employee_id) AS "Employee Count"
FROM departments d
LEFT OUTER JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
ORDER BY d.department_id;
```

```
-- =====
-- COMBINATION OF JOIN AND AGGREGATION
-- =====
```

```
-- General Syntax:
-- SELECT t1.col, SUM(t2.col) FROM table1 t1
-- INNER JOIN table2 t2 ON t1.id = t2.id
-- GROUP BY t1.col;
```

```
-- 30. Total salary by department with department names
SELECT
    d.department_id,
    d.department_name,
    COUNT(e.employee_id) AS "Employee Count",
    SUM(e.salary) AS "Total Salary"
FROM departments d
INNER JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name;
```

```
-- 31. Average salary by job title
SELECT
    j.job_title,
    COUNT(e.employee_id) AS "Number of Employees",
    AVG(e.salary) AS "Average Salary"
FROM jobs j
INNER JOIN employees e
ON j.job_id = e.job_id
GROUP BY j.job_title;
```

```
-- =====
-- OBJECTIVE 6: ORDER BY AND LIMIT (ROWNUM)
-- =====
```

```
-- General Syntax for ORDER BY:
-- SELECT * FROM <table_name> ORDER BY <col> ASC;
-- SELECT * FROM <table_name> ORDER BY <col> DESC;
```

```
-- 32. Employees ordered by salary (ascending)
SELECT employee_id, first_name, last_name, salary
FROM employees
ORDER BY salary ASC;
```

```
-- 33. Employees ordered by salary (descending)
SELECT employee_id, first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

```
-- 34. Employees ordered by hire date (newest first)
SELECT employee_id, first_name, last_name, hire_date
FROM employees
ORDER BY hire_date DESC;
```

```
-- General Syntax for ROWNUM (LIMIT):
```

```
-- WHERE ROWNUM <= n;
```

```
-- 35. Top 5 highest paid employees using ROWNUM
```

```
SELECT * FROM (
    SELECT employee_id, first_name, last_name, salary
    FROM employees
    ORDER BY salary DESC
)
WHERE ROWNUM <= 5;
```

```
-- 36. Top 3 most recent hires
```

```
SELECT * FROM (
    SELECT employee_id, first_name, last_name, hire_date
    FROM employees
    ORDER BY hire_date DESC
)
WHERE ROWNUM <= 3;
```

```
-- 37. Combine JOIN, Aggregation, ORDER BY, and ROWNUM
```

```
-- Top 5 departments by total salary
```

```
SELECT * FROM (
    SELECT
        d.department_name,
        SUM(e.salary) AS "Total Salary"
    FROM departments d
    INNER JOIN employees e
    ON d.department_id = e.department_id
    GROUP BY d.department_name
    ORDER BY SUM(e.salary) DESC
)
WHERE ROWNUM <= 5;
```

```
-- =====
```

```
-- ADDITIONAL IMPORTANT QUERIES
```

```
-- =====
```

```
-- 38. Self Join - Employees with their managers
```

```
SELECT
```

```
    e.employee_id,  
    e.first_name || ' ' || e.last_name AS "Employee",  
    m.first_name || ' ' || m.last_name AS "Manager"
```

```
FROM employees e
```

```
LEFT OUTER JOIN employees m
```

```
ON e.manager_id = m.employee_id;
```

```
-- 39. Subquery - Employees earning more than average salary
```

```
SELECT employee_id, first_name, last_name, salary
```

```
FROM employees
```

```
WHERE salary > (SELECT AVG(salary) FROM employees)
```

```
ORDER BY salary DESC;
```

```
-- 40. HAVING Clause - Departments with more than 5 employees
```

```
SELECT
```

```
    d.department_name,  
    COUNT(e.employee_id) AS "Employee Count"
```

```
FROM departments d
```

```
INNER JOIN employees e
```

```
ON d.department_id = e.department_id
```

```
GROUP BY d.department_name
```

```
HAVING COUNT(e.employee_id) > 5
```

```
ORDER BY COUNT(e.employee_id) DESC;
```

Welcome Pa... Sand... Worksheet Query Builder

```
-- 1. Select all employees
SELECT * FROM employees;
```

Query Result SQL Fetched 50 rows in 0.085 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
6	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100

Welcome Pa... Sand... Worksheet Query Builder

```
-- 2. Select all departments
SELECT * FROM departments;
```

Query Result SQL Fetched 50 rows in 0.024 seconds

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	MANAGER_ID
1	10 Administration	1700	(null)
2	20 Marketing	1800	(null)
3	30 Purchasing	1700	114
4	40 Human Resources	2400	(null)
5	50 Shipping	1500	120
6	60 IT	1400	102
7	70 Public Relations	2700	(null)
8	80 Sales	2500	145
9	90 Executive	1700	100
10	100 Finance	1700	101

Welcome Pa... Sand... Worksheet Query Builder

```
-- 3. Find employee first name, last name, and email from employees
SELECT first_name, last_name, email FROM employees;
```

Query Result x | Query Result 1 x | Query Result 2 x

SQL | Fetched 50 rows in 0.015 seconds

	FIRST_NAME	LAST_NAME	EMAIL
1	Steven	King	SKING
2	Neena	Kochhar	NKOCHHAR
3	Lex	De Haan	LDEHAAN
4	Alexander	Hunold	AHUNOLD
5	Bruce	Ernst	BERNST
6	David	Austin	DAUSTIN
7	Valli	Pataballa	VPATABAL
8	Diana	Lorentz	DLORENTZ
9	Nancy	Greenberg	NGREENBE

Welcome Pa... Sand... Worksheet Query Builder

```
-- 4. Find department name and location ID from departments
SELECT department_name, location_id FROM departments;
```

Query Result x | Query Result 1 x | Query Result 2 x | Query Result 3 x

SQL | Fetched 50 rows in 0.009 seconds

	DEPARTMENT_NAME	LOCATION_ID
1	Administration	1700
2	Marketing	1800
3	Purchasing	1700
4	Human Resources	2400
5	Shipping	1500

Welcome Pa... Sand... | Worksheet | Query Builder

```
-- 5. Find all employees who work in department 60 (IT)
SELECT * FROM employees
WHERE department_id = 60;
```

Query Result 4 | SQL | All Rows Fetched: 5 in 0.033 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	103 Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
2	104 Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
3	105 David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
4	106 Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
5	107 Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60

Welcome Pa... Sand... | Worksheet | Query Builder

```
-- 6. Find list of all employees hired before 1st January 2006
SELECT * FROM employees
WHERE hire_date < TO_DATE('01/01/2006', 'DD/MM/YYYY');
```

Query Result | SQL | All Rows Fetched: 28 in 0.039 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100 Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
2	101 Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
3	102 Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
4	105 David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
5	108 Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
6	109 Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
7	110 John	Chen	JCHEN	515.124.4269	20-SEP-05	FI_ACCOUNT	8200	(null)	108	100
8	111 Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
9	114 Den	Raphaelly	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	11000	(null)	100	30
10	115 Alexander	Khoo	AKHOO	515.127.4562	18-MAY-03	PU_CLERK	3100	(null)	114	30

Welcome Pa... Sand... | Worksheet | Query Builder

```
-- 7. Find employees with salary greater than 10000
SELECT * FROM employees
WHERE salary > 10000;
```

Query Result 1 | SQL | All Rows Fetched: 10 in 0.012 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100 Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
2	101 Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
3	102 Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
4	108 Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
5	114 Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	11000	(null)	100	30
6	145 John	Russell	JRUSSEL	011.44.1344.429...	01-OCT-04	SA_MAN	14000	0.4	100	80
7	146 Karen	Partners	KPARTNER	011.44.1344.467...	05-JAN-05	SA_MAN	13500	0.3	100	80
8	147 Alberto	Errazuriz	AERRAZUR	011.44.1344.429...	10-MAR-05	SA_MAN	12000	0.3	100	80
9	148 Gerald	Cambrault	GCAMBRAU	011.44.1344.619...	15-OCT-07	SA_MAN	11000	0.3	100	80
10	149 Eleni	Slotkey	EZLOTKEY	011.44.1344.429...	29-JAN-08	SA_MAN	10500	0.2	100	80

Welcome Pa... Sand... | Worksheet | Query Builder

```
-- 8. AND Operator - Find employees in IT dept (60) with salary > 5000
SELECT * FROM employees
WHERE department_id = 60 AND salary > 5000;
```

Query Result 1 | Query Result 2 | SQL | All Rows Fetched: 2 in 0.025 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	103 Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
2	104 Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60

Worksheet | Query Builder

```
-- 9. OR Operator - Find employees in department 50 OR 60
SELECT * FROM employees
WHERE department_id = 50 OR department_id = 60;
```

Query Result 3 | SQL | All Rows Fetched: 30 in 0.02 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-04	ST_MAN	8000	(null)	100	50
2	121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-05	ST_MAN	8200	(null)	100	50
3	122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-03	ST_MAN	7900	(null)	100	50
4	123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-05	ST_MAN	6500	(null)	100	50
5	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800	(null)	100	50
6	125	Julia	Nayer	JNAYER	650.124.1214	16-JUL-05	ST_CLERK	3200	(null)	120	50
7	126	Irene	Mikkilineni	IMIKKILINENI	650.124.1224	28-SEP-06	ST_CLERK	2700	(null)	120	50
8	127	James	Landry	JLANDRY	650.124.1334	14-JAN-07	ST_CLERK	2400	(null)	120	50
9	128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-08	ST_CLERK	2200	(null)	120	50
10	129	Laura	Bisso	LBISSET	650.124.5234	20-AUG-05	ST_CLERK	3300	(null)	121	50

Worksheet | Query Builder

```
-- 10. NOT Operator - Find employees NOT in department 90
SELECT * FROM employees
WHERE NOT department_id = 90;
```

Query Result 3 | Query Result 4 | SQL | All Rows Fetched: 47 in 0.016 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
2	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
3	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
4	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
5	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
6	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
7	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
8	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
9	111	Ismael	Sciarrra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
10	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100

Worksheet | Query Builder

```
-- 11. BETWEEN - Find employees with salary between 5000 and 10000
SELECT * FROM employees
WHERE salary BETWEEN 5000 AND 10000;
```

Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6 | SQL | All Rows Fetched: 12 in 0.006 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
2	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
3	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
4	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
5	111	Ismael	Sciarrra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
6	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100
7	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	6900	(null)	108	100
8	120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-04	ST_MAN	8000	(null)	100	50
9	121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-05	ST_MAN	8200	(null)	100	50
10	122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-03	ST_MAN	7900	(null)	100	50
11	123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-05	ST_MAN	6500	(null)	100	50
12	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800	(null)	100	50

-- 12. BETWEEN with dates - Find employees hired between 2005 and 2007

```
SELECT * FROM employees
WHERE hire_date BETWEEN TO_DATE('01/01/2005', 'DD/MM/YYYY')
    AND TO_DATE('31/12/2007', 'DD/MM/YYYY');
```

Query Result

All Rows Fetched: 35 in 0.025 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	101	Neena	Kochhar	NKOCHHAR 515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
2	103	Alexander	Hunold	AHUNOLD 590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
3	104	Bruce	Ernst	BERNST 590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
4	105	David	Austin	DAUSTIN 590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
5	106	Valli	Fataballa	VFATABAL 590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ 590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
7	110	John	Chen	JCHEN 515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
8	111	Ismael	Sciarra	ISCIARRA 515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
9	112	Jose Manuel	Urman	JMURMAN 515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100
10	113	Luis	Popp	LPOPP 515.124.4567	07-DEC-07	FI_ACCOUNT	6900	(null)	108	100

-- 13. NOT BETWEEN - Find employees with salary NOT between 3000 and 7000

```
SELECT * FROM employees
WHERE salary NOT BETWEEN 3000 AND 7000;
```

Query Result 1

All Rows Fetched: 35 in 0.015 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING 515.123.4567	17-JUN-03	AD PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR 515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN 515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD 590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
5	108	Nancy	Greenberg	NGREENBE 515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
6	109	Daniel	Faviet	DFAVIET 515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
7	110	John	Chen	JCHEN 515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
8	111	Ismael	Sciarra	ISCIARRA 515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
9	112	Jose Manuel	Urman	JMURMAN 515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100
10	114	Den	Raphaely	DRAPEHAL 515.127.4561	07-DEC-02	PU_MAN	11000	(null)	100	30

-- 14. IN Operator - Find employees in departments 30, 50, or 60

```
SELECT * FROM employees
WHERE department_id IN (30, 50, 60);
```

Query Result 1

Query Result 2

All Rows Fetched: 36 in 0.011 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	114	Den	Raphaely	DRAPEHAL 515.127.4561	07-DEC-02	PU_MAN	11000	(null)	100	30
2	115	Alexander	Rhoo	AKHOO 515.127.4562	18-MAY-03	PU_CLERK	3100	(null)	114	30
3	116	Shelli	Baida	SBAIDA 515.127.4563	24-DEC-05	PU_CLERK	2900	(null)	114	30
4	117	Sigal	Tobias	STOBIA 515.127.4564	24-JUL-05	PU_CLERK	2800	(null)	114	30
5	118	Guy	Himuro	GHIMURO 515.127.4565	15-NOV-06	PU_CLERK	2600	(null)	114	30
6	119	Karen	Colmenares	KCOLMENA 515.127.4566	10-AUG-07	PU_CLERK	2500	(null)	114	30
7	120	Matthew	Weiss	MWEISS 650.123.1234	18-JUL-04	ST_MAN	8000	(null)	100	50
8	121	Adam	Fripp	AFRIPP 650.123.2234	10-APR-05	ST_MAN	8200	(null)	100	50
9	122	Payam	Raufling	PKAUFLIN 650.123.3234	01-MAY-03	ST_MAN	7900	(null)	100	50
10	123	Shanta	Vollman	SVOLLMAN 650.123.4234	10-OCT-05	ST_MAN	6500	(null)	100	50

Worksheet | Query Builder

```
-- 15. IN Operator with job titles
SELECT * FROM employees
WHERE job_id IN ('IT_PROG', 'FI_ACCOUNT', 'SA_MAN');
```

Query Result 1 | Query Result 2 | Query Result 3 |

SQL | All Rows Fetched: 15 in 0.024 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	109	Daniel	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100	
2	110	John	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100	
3	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
4	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100
5	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	6900	(null)	108	100
6	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
7	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
8	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
9	106	Valli	Patataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
10	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60

Worksheet | Query Builder

```
-- 16. NOT IN - Find employees NOT in departments 50 or 90
SELECT * FROM employees
WHERE department_id NOT IN (50, 90);
```

Query Result 4 |

SQL | All Rows Fetched: 22 in 0.015 seconds

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
2	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
3	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
4	106	Valli	Patataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
5	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
6	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
7	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
8	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
9	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
10	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100

Worksheet | Query Builder

```
-- 17. Find total number of employees
SELECT COUNT(*) AS "Total Employees"
FROM employees;
```

Query Result |

SQL | All Rows Fetched: 1 in 0.008 seconds

Total Employees	
1	50

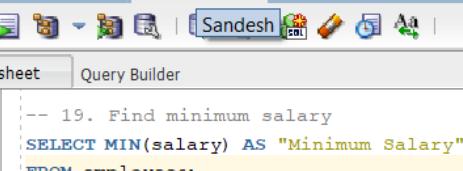
Worksheet | Query Builder

```
-- 18. Find total salary (SUM)
SELECT SUM(salary) AS "Total Salary"
FROM employees;
```

Query Result | Query Result 1 |

SQL | All Rows Fetched: 1 in 0.006 seconds

Total Salary	
1	316408



-- 19. Find minimum salary

```
SELECT MIN(salary) AS "Minimum Salary"  
FROM employees;
```

The screenshot shows the SQL Server Management Studio interface. The title bar says "Welcome Pa... Sandesh". The toolbar includes icons for Run, Save, Print, and others. The main window has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The query results pane shows the following output:

```
-- 20. Find maximum salary
SELECT MAX(salary) AS "Maximum Salary"
FROM employees;
```

The results pane shows a single row with the value 24000.

-- 21. Find average salary

```
SELECT AVG(salary) AS "Average Salary"  
FROM employees;
```

Query Result

	Average Salary
1	6328.16

-- 22. Find standard deviation of salaries  
SELECT STDDEV(salary) AS "Salary Std Deviation"  
FROM employees;

Query Result x    Query Result 1 x

SQL | All Rows Fetched: 1 in 0.006 seconds

Salary Std Deviation
1:811.872253720213734451475359169446480...

Welcome Pa... Sand...

Worksheet Query Builder

```
-- 25. Cross Join with WHERE clause
SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | Fetched 50 rows in 0.024 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
1	114	Den	Raphaely	Purchasing
2	115	Alexander	Khoo	Purchasing
3	116	Shelli	Baida	Purchasing
4	117	Sigal	Tobias	Purchasing
5	118	Guy	Himuro	Purchasing
6	119	Karen	Colmenares	Purchasing
7	120	Matthew	Weiss	Shipping
8	121	Adam	Fripp	Shipping
9	122	Payam	Kaufling	Shipping
10	123	Shanta	Vollman	Shipping

Welcome Pa... Sand... Sandesh

Worksheet Query Builder

```
-- 26. Inner Join - Employees with department names
SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    e.salary,
    d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

Query Result X

SQL | Fetched 50 rows in 0.011 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPARTMENT_NAME
1	114	Den	Raphaely	11000	Purchasing
2	115	Alexander	Khoo	3100	Purchasing
3	116	Shelli	Baida	2900	Purchasing
4	117	Sigal	Tobias	2800	Purchasing
5	118	Guy	Himuro	2600	Purchasing
6	119	Karen	Colmenares	2500	Purchasing
7	120	Matthew	Weiss	8000	Shipping
8	121	Adam	Fripp	8200	Shipping
9	122	Payam	Kaufling	7900	Shipping
10	123	Shanta	Vollman	6500	Shipping

Welcome Pa... x Sand... x

Worksheet Query Builder

```
-- 27. Inner Join - Employees with job titles
SELECT
    e.employee_id,
    e.first_name || ' ' || e.last_name AS "Full Name",
    j.job_title,
    e.salary
FROM employees e
INNER JOIN jobs j
ON e.job_id = j.job_id;
```

Query Result1 x

SQL | Fetched 50 rows in 0.03 seconds

	EMPLOYEE_ID	Full Name	JOB_TITLE	SALARY
1	100	Steven King	President	24000
2	101	Neena Kochhar	Vice President	17000
3	102	Lex De Haan	Vice President	17000
4	108	Nancy Greenberg	Finance Manager	12008
5	109	Daniel Faviet	Accountant	9000
6	110	John Chen	Accountant	8200
7	111	Ismael Sciarra	Accountant	7700
8	112	Jose Manuel Ur...	Accountant	7800
9	113	Luis Popp	Accountant	6900
10	145	John Russell	Sales Manager	14000

Welcome Pa... Sand... |

Worksheet Query Builder

```
-- 28. Multiple INNER JOINS - Employees with department and location
SELECT
    e.employee_id,
    e.first_name || ' ' || e.last_name AS "Employee Name",
    d.department_name,
    l.city,
    c.country_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
INNER JOIN locations l ON d.location_id = l.location_id
INNER JOIN countries c ON l.country_id = c.country_id;
```

Query Result X

SQL | Fetched 50 rows in 0.039 seconds

	EMPLOYEE_ID	Employee Name	DEPARTMENT_NAME	CITY	COUNTRY_NAME
1	103	Alexander Hunold	IT	London	United King...
2	104	Bruce Ernst	IT	London	United King...
3	105	David Austin	IT	London	United King...
4	106	Valli Pataballa	IT	London	United King...
5	107	Diana Lorentz	IT	London	United King...
6	120	Matthew Weiss	Shipping	Oxford	United King...
7	121	Adam Fripp	Shipping	Oxford	United King...
8	122	Payam Kaufling	Shipping	Oxford	United King...
9	123	Shanta Vollman	Shipping	Oxford	United King...
10	124	Kevin Mourgos	Shipping	Oxford	United King...

Welcome Pa... Sand... Worksheet Query Builder

```
-- 29. Left Outer Join - All departments with employee count
SELECT
    d.department_id,
    d.department_name,
    COUNT(e.employee_id) AS "Employee Count"
FROM departments d
LEFT OUTER JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
ORDER BY d.department_id;
```

Query Result 1 X

SQL | Fetched 50 rows in 0.05 seconds

DEPARTMENT_ID	DEPARTMENT_NAME	Employee Count
1	10 Administration	0
2	20 Marketing	0
3	30 Purchasing	6
4	40 Human Resources	0
5	50 Shipping	25
6	60 IT	5
7	70 Public Relations	0
8	80 Sales	5
9	90 Executive	3
10	100 Finance	6

Welcome Pa... x SQL Sand... x

Worksheet Query Builder

```
-- 30. Total salary by department with department names
SELECT
    d.department_id,
    d.department_name,
    COUNT(e.employee_id) AS "Employee Count",
    SUM(e.salary) AS "Total Salary"
FROM departments d
INNER JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name;
```

Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 6 in 0.032 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME	Employee Count	Total Salary
1	30	Purchasing	6	24900
2	50	Shipping	25	92100
3	60	IT	5	28800
4	80	Sales	5	61000
5	90	Executive	3	58000
6	100	Finance	6	51608

Welcome Pa... Sand...

Worksheet Query Builder

```
-- 31. Average salary by job title
SELECT
    j.job_title,
    COUNT(e.employee_id) AS "Number of Employees",
    AVG(e.salary) AS "Average Salary"
FROM jobs j
INNER JOIN employees e
ON j.job_id = e.job_id
GROUP BY j.job_title;
```

Query Result 1 X | Query Result 2 X | Query Result 3 X

SQL | All Rows Fetched: 10 in 0.017 seconds

JOB_TITLE	Number of Employees	Average Salary
President	1	24000
Vice President	2	17000
Finance Manager	1	12008
Accountant	5	7920
Sales Manager	5	12200
Purchasing Mana...	1	11000
Purchasing Clerk	5	2780
Stock Manager	5	7280
Stock Clerk	20	2785
Programmer	5	5760

Welcome Pa... Sand...

Worksheet Query Builder

```
-- 32. Employees ordered by salary (ascending)
SELECT employee_id, first_name, last_name, salary
FROM employees
ORDER BY salary ASC;
```

Query Result Fetched 50 rows in 0.009 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	132	TJ	Olson	2100
2	136	Hazel	Philtanker	2200
3	128	Steven	Markle	2200
4	135	Ki	Gee	2400
5	127	James	Landry	2400
6	119	Karen	Colmenares	2500
7	144	Peter	Vargas	2500
8	140	Joshua	Patel	2500
9	131	James	Marlow	2500
10	143	Randall	Matos	2600
11	118	Guy	Himuro	2600
12	139	John	Seo	2700

Welcome Pa... Sand...

Worksheet Query Builder

```
-- 33. Employees ordered by salary (descending)
SELECT employee_id, first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

Query Result 1 X

SQL | Fetched 50 rows in 0.016 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	100	Steven	King	24000
2	101	Neena	Kochhar	17000
3	102	Lex	De Haan	17000
4	145	John	Russell	14000
5	146	Karen	Partners	13500
6	108	Nancy	Greenberg	12008
7	147	Alberto	Errazuriz	12000
8	148	Gerald	Cambrault	11000
9	114	Den	Raphaely	11000
10	149	Eleni	Zlotkey	10500

Welcome Pa... Sand... | Sandesh

Worksheet Query Builder

```
-- 34. Employees ordered by hire date (newest first)
SELECT employee_id, first_name, last_name, hire_date
FROM employees
ORDER BY hire_date DESC;
```

Query Result 1 × Query Result 2 ×

SQL | Fetched 50 rows in 0.025 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	HIRE_DATE
1	128	Steven	Markle	08-MAR-08
2	136	Hazel	Philtanker	06-FEB-08
3	149	Eleni	Zlotkey	29-JAN-08
4	135	Ki	Gee	12-DEC-07
5	113	Luis	Popp	07-DEC-07
6	124	Kevin	Mourgos	16-NOV-07
7	148	Gerald	Cambrault	15-OCT-07
8	119	Karen	Colmenares	10-AUG-07
9	104	Bruce	Ernst	21-MAY-07
10	132	TJ	Olson	10-APR-07

Welcome Pa... x Sand... x

Worksheet Query Builder

```
-- 35. Top 5 highest paid employees using ROWNUM
SELECT * FROM (
    SELECT employee_id, first_name, last_name, salary
    FROM employees
    ORDER BY salary DESC
)
WHERE ROWNUM <= 5;
```

Query Result 3 x

All Rows Fetched: 5 in 0.012 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	100	Steven	King	24000
2	101	Neena	Kochhar	17000
3	102	Lex	De Haan	17000
4	145	John	Russell	14000
5	146	Karen	Partners	13500

Welcome Pa... Sand... Sandesh

Worksheet Query Builder

```
-- 36. Top 3 most recent hires
SELECT * FROM (
    SELECT employee_id, first_name, last_name, hire_date
    FROM employees
    ORDER BY hire_date DESC
)
WHERE ROWNUM <= 3;
```

Query Result 4 X

SQL | All Rows Fetched: 3 in 0.017 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	HIRE_DATE
1	128	Steven	Markle	08-MAR-08
2	136	Hazel	Philtanker	06-FEB-08
3	149	Eleni	Zlotkey	29-JAN-08

Welcome Pa... x Sand... x

Worksheet Query Builder

```
-- 37. Combine JOIN, Aggregation, ORDER BY, and ROWNUM
-- Top 5 departments by total salary
SELECT * FROM (
    SELECT
        d.department_name,
        SUM(e.salary) AS "Total Salary"
    FROM departments d
    INNER JOIN employees e
    ON d.department_id = e.department_id
    GROUP BY d.department_name
    ORDER BY SUM(e.salary) DESC
)
WHERE ROWNUM <= 5;
```

Query Result 4 x Query Result 5 x

SQL | All Rows Fetched: 5 in 0.024 seconds

	DEPARTMENT_NAME	Total Salary
1	Shipping	92100
2	Sales	61000
3	Executive	58000
4	Finance	51608
5	IT	28800

## Lab 4:

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are two tabs: '4. View....' and 'Sandesh...'. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
-- 1. Simple view of all employees
CREATE VIEW vw_all_employees AS
SELECT employee_id, first_name, last_name, salary, department_id, job_id
FROM employees;
```

Below the worksheet is a 'Script Output' window showing the result of the execution:

```
View VW_ALL_EMPLOYEES created.
```

The screenshot shows the Oracle SQL Developer interface again. The top tab bar has tabs for '4. View....' and 'Sandesh...'. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
-- 2. Fetch employees with salary > 10000 from the view
SELECT * FROM vw_all_employees
WHERE salary > 10000;
```

Below the worksheet is a 'Query Result' window showing the output of the query:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPARTMENT_ID	JOB_ID
1	100	Steven	King	24000	90 AD_PRES
2	101	Neena	Kochhar	17000	90 AD_VP
3	102	Lex	De Haan	17000	90 AD_VP
4	108	Nancy	Greenberg	12008	100 FI_MGR
5	114	Den	Raphaely	11000	30 PU_MAN
6	145	John	Russell	14000	80 SA_MAN
7	146	Karen	Partners	13500	80 SA_MAN
8	147	Alberto	Errazuriz	12000	80 SA_MAN
9	148	Gerald	Cambrault	11000	80 SA_MAN
10	149	Eleni	Zlotkey	10500	80 SA_MAN

SQl 4. View.... x Sandesh... x

Sandesh~1

Worksheet Query Builder

```
-- 3. Join view with jobs table to get job titles
SELECT v.employee_id, v.first_name || ' ' || v.last_name AS full_name, j.job_title, v.salary
FROM vw_all_employees v
INNER JOIN jobs j ON v.job_id = j.job_id;
```

Query Result x

SQL | Fetched 50 rows in 0.03 seconds

	EMPLOYEE_ID	FULL_NAME	JOB_TITLE	SALARY
1	100	Steven King	President	24000
2	101	Neena Kochhar	Vice President	17000
3	102	Lex De Haan	Vice President	17000
4	109	Daniel Faviet	Accountant	9000
5	110	John Chen	Accountant	8200
6	111	Ismael Sciarra	Accountant	7700
7	112	Jose Manuel Ur...	Accountant	7800
8	113	Luis Popp	Accountant	6900
9	108	Nancy Greenberg	Finance Manager	12008

SQl 4. View.... x Sandesh... x

Sandesh~1

Worksheet Query Builder

```
-- 4. Create a materialized view for high salary employees
CREATE MATERIALIZED VIEW mv_high_salary_employees AS
SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE salary > 15000;
```

Script Output x

Task completed in 0.707 seconds

Materialized view MV\_HIGH\_SALARY\_EMPLOYEES created.

SQl 4. View.... x Sandesh... x

Sandesh~1

Worksheet Query Builder

```
-- 5. Query materialized view
SELECT * FROM mv_high_salary_employees
ORDER BY salary DESC;
```

Query Result x

SQL | All Rows Fetched: 3 in 0.03 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	100	Steven	King	24000
2	102	Lex	De Haan	17000
3	101	Neena	Kochhar	17000

## LAB 5: PL/SQL

Student: Sandesh Khatiwada

Date: 11/11/2025

### OBJECTIVES:

1. Understand basic PL/SQL structure
2. Perform simple computations and decision-making
3. Create and execute functions and procedures
4. Use loops in PL/SQL programs

```
-- =====  
-- GENERAL SYNTAX (REFERENCE)  
-- =====  
-- DECLARE  
--   <variables>;  
-- BEGIN  
--   <statements>;  
-- EXCEPTION  
--   <exception-handling>;  
-- END;  
-- /
```

### OBJECTIVE 1: BASIC PL/SQL BLOCK

```
SET SERVEROUTPUT ON;
```

#### 1. Hello World in PL/SQL

```
BEGIN  
  DBMS_OUTPUT.PUT_LINE('Hello world');  
END;  
/
```

## OBJECTIVE 2: SIMPLE COMPUTATION & CONDITIONS

### 2.1 Calculate area of a circle

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
    pi CONSTANT NUMBER := 3.14;
```

```
    radius NUMBER := &r;
```

```
    area NUMBER;
```

```
BEGIN
```

```
    area := pi * radius * radius;
```

```
    DBMS_OUTPUT.PUT_LINE('The area is: ' || area);
```

```
END;
```

```
/
```

### 2.2 Eligibility check using IF-ELSE

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
    age NUMBER := &age;
```

```
BEGIN
```

```
    IF age > 22 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Student is eligible for admission.');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('Student is NOT eligible for admission.');
```

```
    END IF;
```

```
END;
```

```
/
```

## OBJECTIVE 3: FUNCTIONS IN PL/SQL

3 Function: Total salary by department

```
CREATE OR REPLACE FUNCTION get_total_salary_by_dept
(
    p_dept_id IN NUMBER
)
RETURN NUMBER
AS
    v_total_salary NUMBER;
BEGIN
    SELECT SUM(salary + NVL(commission_pct,0) * salary)
    INTO v_total_salary
    FROM employees
    WHERE department_id = p_dept_id;

    RETURN NVL(v_total_salary, 0);
END;
/

SET SERVEROUTPUT ON;

DECLARE
    v_total NUMBER;
BEGIN
    v_total := get_total_salary_by_dept(80);
    DBMS_OUTPUT.PUT_LINE('Total Salary for Dept 80: ' || v_total);
END;
/
```

```
-- =====
-- OBJECTIVE 4: STORED PROCEDURES
-- =====

CREATE OR REPLACE PROCEDURE get_max_salary(
    v_max_salary OUT employees.salary%TYPE,
    dept_id IN employees.department_id%TYPE
)
AS
BEGIN
    SELECT MAX(salary)
    INTO v_max_salary
    FROM employees
    WHERE department_id = dept_id;

    DBMS_OUTPUT.PUT_LINE('Maximum Salary: ' || v_max_salary);
END get_max_salary;
/

SET SERVEROUTPUT ON;

DECLARE
    v_max employees.salary%TYPE;
BEGIN
    get_max_salary(v_max, 80);
    DBMS_OUTPUT.PUT_LINE('Returned Max Salary: ' || v_max);
END;
/

```

## OBJECTIVE 5: LOOPS IN PL/SQL

```
-- Print even multiples of 5 less than 100
SET SERVEROUTPUT ON;
```

```
DECLARE
    i NUMBER := 1;
    prod NUMBER;
BEGIN
    LOOP
        prod := i * 5;

        EXIT WHEN prod >= 100;

        IF MOD(prod, 2) = 0 THEN
            DBMS_OUTPUT.PUT_LINE(prod);
        END IF;

        i := i + 1;
    END LOOP;
END;
/
-- =====
-- CONCLUSION
-- =====
-- PL/SQL allows procedural logic inside SQL environments.
-- Variables, loops, conditions, functions, and procedures make Oracle more powerful.
-- This lab demonstrated practical use cases of PL/SQL blocks, functions, procedures,
-- and loops.
```

Sandesh~2

Worksheet    Query Builder

```
--1. To be able to understand and implement PL/SQL
--hello world in PL/SQL

SET SERVEROUTPUT ON;

BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello world');
END;
/
```

Script Output X

Task completed in 0.107 seconds

Hello world

PL/SQL procedure successfully completed.

Sandesh... x 5.PL\_SQL... x

Sandesh HR | Worksheet | Query Builder |

Worksheet    Query Builder

```
--2.1 To calculate area of a circle
SET SERVEROUTPUT ON;

DECLARE
    pi CONSTANT NUMBER := 3.14;
    radius NUMBER := &r;
    area NUMBER;
BEGIN
    area := pi * radius * radius;
    DBMS_OUTPUT.PUT_LINE('The area is: '|| area);
END;
/

```

Script Output x

Task completed in 3.704 seconds

```
old:DECLARE
    pi CONSTANT NUMBER := 3.14;
    radius NUMBER := &r;
    area NUMBER;
BEGIN
    area := pi * radius * radius;
    DBMS_OUTPUT.PUT_LINE('The area is: ' || area);
END;

new:DECLARE
    pi CONSTANT NUMBER := 3.14;
    radius NUMBER := 5;
    area NUMBER;
BEGIN
    area := pi * radius * radius;
    DBMS_OUTPUT.PUT_LINE('The area is: ' || area);
END;
The area is: 78.5

PL/SQL procedure successfully completed.
```

Sandesh... x 5.PL\_SQL... x

Sandesh HR | Worksheet | Query Builder |

```
-- 2.2. Use IF-ELSE to check admission eligibility based on age
SET SERVEROUTPUT ON;

DECLARE
    age NUMBER := &age;
BEGIN
    IF age > 22 THEN
        DBMS_OUTPUT.PUT_LINE('Student is eligible for admission.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Student is NOT eligible for admission.');
    END IF;
END;
/

```

Script Output x | Task completed in 2.366 seconds

```
old:DECLARE
    age NUMBER := &age;
BEGIN
    IF age > 22 THEN
        DBMS_OUTPUT.PUT_LINE('Student is eligible for admission.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Student is NOT eligible for admission.');
    END IF;
END;
Student is eligible for admission.

PL/SQL procedure successfully completed.
```

Sandesh... | Sandesh HR | Worksheet | Query Builder

```
--4 Function
CREATE OR REPLACE FUNCTION get_total_employees_before_date
(
    p_hire_date IN DATE
)
RETURN NUMBER
AS
    v_total NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_total
    FROM employees
    WHERE hire_date < p_hire_date;

    RETURN NVL(v_total, 0);
END;
/
SET SERVEROUTPUT ON;

DECLARE
    v_total NUMBER;
BEGIN
    v_total := get_total_employees_before_date(TO_DATE('2021-01-01', 'YYYY-MM-DD'));
    DBMS_OUTPUT.PUT_LINE('Total Employees hired before 2021-01-01: '|| v_total);
END;
/
```

Script Output X | Task completed in 0.206 seconds

```
Function GET_TOTAL_EMPLOYEES_BEFORE_DATE compiled
Total Employees hired before 2021-01-01: 107
PL/SQL procedure successfully completed.
```

Sandesh... 5.PL\_SQL... Sandesh HR

Worksheet Query Builder

```
--3 Create a function
CREATE OR REPLACE FUNCTION get_total_salary_by_dept
(
    p_dept_id IN NUMBER
)
RETURN NUMBER AS
    v_total_salary NUMBER;
BEGIN
    SELECT SUM(salary + NVL(commission_pct,0)*salary)
    --SELECT SUM(salary)
    INTO v_total_salary
    FROM employees
    WHERE department_id = p_dept_id;
    RETURN NVL(v_total_salary, 0);
END;
/
SET SERVEROUTPUT ON;
DECLARE
    v_total NUMBER;
BEGIN
    v_total := get_total_salary_by_dept(80);
    DBMS_OUTPUT.PUT_LINE('Total Salary for Dept 80: '|| v_total);
END;
/
```

Script Output | Task completed in 0.277 seconds

```
Function GET_TOTAL_SALARY_BY_DEPT compiled
Total Salary for Dept 80: 382740
PL/SQL procedure successfully completed.
```

Sandesh... | Sandesh HR | A A |

Worksheet    Query Builder

```
--5 STORED pROCEDURE
CREATE OR REPLACE PROCEDURE get_max_salary(
    v_max_salary OUT employees.salary%TYPE,
    dept_id IN employees.department_id%TYPE)
AS
BEGIN
    SELECT MAX(salary)
    INTO v_max_salary
    FROM employees
    WHERE department_id=dept_id;

    DBMS_OUTPUT.PUT_LINE('Maximum Salary: '|| v_max_salary);
END get_max_salary;
/

SET SERVEROUTPUT ON;

DECLARE
    v_max employees.salary%TYPE;
BEGIN
    get_max_salary(v_max, 80);

    DBMS_OUTPUT.PUT_LINE('Returned Max Salary: '|| v_max);
END;
/

```

Script Output X | Task completed in 0.202 seconds

```
Procedure GET_MAX_SALARY compiled

Maximum Salary: 14000
Returned Max Salary: 14000

PL/SQL procedure successfully completed.
```

Sandesh... x 5.PL\_SQL... x

Sandesh HR

Worksheet Query Builder

```
--6. wap to find even multiples of 5 less than 100
SET SERVEROUTPUT ON;

DECLARE
    i NUMBER := 1;
    prod NUMBER;
BEGIN
    LOOP
        prod := i * 5;

        -- exit when product exceeds 100
        EXIT WHEN prod >= 100;

        -- print only even multiples
        IF MOD(prod, 2) = 0 THEN
            DBMS_OUTPUT.PUT_LINE(prod);
        END IF;

        i := i + 1;
    END LOOP;
END;
/

```

Script Output x

Task completed in 0.097 seconds

```
10
20
30
40
50
60
70
80
90

PL/SQL procedure successfully completed.
```

## 1. Range Partitioning

-- Create table with RANGE partitioning on sales\_date

```
CREATE TABLE sales1
```

```
(
```

```
    customer_id NUMBER,
```

```
    sales_date DATE,
```

```
    order_amount NUMBER,
```

```
    region NVARCHAR2(10)
```

```
)
```

```
PARTITION BY RANGE (sales_date)
```

```
(
```

```
    PARTITION p1 VALUES LESS THAN (TO_DATE('01-03-2015', 'DD-MM-YYYY')),
```

```
    PARTITION p2 VALUES LESS THAN (TO_DATE('01-05-2015', 'DD-MM-YYYY')),
```

```
    PARTITION p3 VALUES LESS THAN (TO_DATE('01-07-2015', 'DD-MM-YYYY')),
```

```
    PARTITION p4 VALUES LESS THAN (MAXVALUE)
```

```
);
```

The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'File', 'Edit', 'Tools', 'Help', and a 'Sand...' icon. Below the menu is a toolbar with various icons. The main area is divided into two tabs: 'Worksheet' (selected) and 'Query Builder'. The 'Worksheet' tab contains the SQL code for creating a table with range partitioning. The 'Script Output' tab at the bottom shows the message 'Table SALES1 created.' and indicates a task completed in 0.184 seconds.

```
--1 Range Partitioning
-----
-- Create table with RANGE partitioning on sales_date
CREATE TABLE sales1
(
    customer_id NUMBER,
    sales_date DATE,
    order_amount NUMBER,
    region NVARCHAR2(10)
)
PARTITION BY RANGE (sales_date)
(
    PARTITION p1 VALUES LESS THAN (TO_DATE('01-03-2015', 'DD-MM-YYYY')),
    PARTITION p2 VALUES LESS THAN (TO_DATE('01-05-2015', 'DD-MM-YYYY')),
    PARTITION p3 VALUES LESS THAN (TO_DATE('01-07-2015', 'DD-MM-YYYY')),
    PARTITION p4 VALUES LESS THAN (MAXVALUE)
);

```

```
-- Insert rows (
-- p1 (date < 01-03-2015) -> 1 row
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (1, TO_DATE('01-01-2015', 'DD-MM-YYYY'), 600, N'SOUTH');

-- p2 (>=01-03-2015 and <01-05-2015) -> 2 rows
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (2, TO_DATE('15-03-2015', 'DD-MM-YYYY'), 1200, N'NORTH');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (3, TO_DATE('30-04-2015', 'DD-MM-YYYY'), 750, N'EAST');

-- p3 (>=01-05-2015 and <01-07-2015) -> 3 rows
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (4, TO_DATE('05-05-2015', 'DD-MM-YYYY'), 300, N'WEST');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (5, TO_DATE('20-05-2015', 'DD-MM-YYYY'), 450, N'SOUTH');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (6, TO_DATE('15-06-2015', 'DD-MM-YYYY'), 900, N'NORTH');

COMMIT;
```

Sand... X

Sandesh

Worksheet | Query Builder

```
-- Insert rows (
-- p1 (date < 01-03-2015) -> 1 row
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (1, TO_DATE('01-01-2015', 'DD-MM-YYYY'), 600, N'SOUTH');

-- p2 (>=01-03-2015 and <01-05-2015) -> 2 rows
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (2, TO_DATE('15-03-2015', 'DD-MM-YYYY'), 1200, N'NORTH');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (3, TO_DATE('30-04-2015', 'DD-MM-YYYY'), 750, N'EAST');

-- p3 (>=01-05-2015 and <01-07-2015) -> 3 rows
INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (4, TO_DATE('05-05-2015', 'DD-MM-YYYY'), 300, N'WEST');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (5, TO_DATE('20-05-2015', 'DD-MM-YYYY'), 450, N'SOUTH');

INSERT INTO sales1 (customer_id, sales_date, order_amount, region)
VALUES (6, TO_DATE('15-06-2015', 'DD-MM-YYYY'), 900, N'NORTH');

COMMIT;
```

Script Output X

| Task completed in 0.439 seconds

1 row inserted.

1 row inserted.

1 row inserted.

Commit complete.

-- Verify: view all rows

SELECT \* FROM sales1 ORDER BY sales\_date;

Sand... X

Sandesh

Worksheet | Query Builder

```
-- Verify: view all rows
SELECT * FROM sales1 ORDER BY sales_date;
```

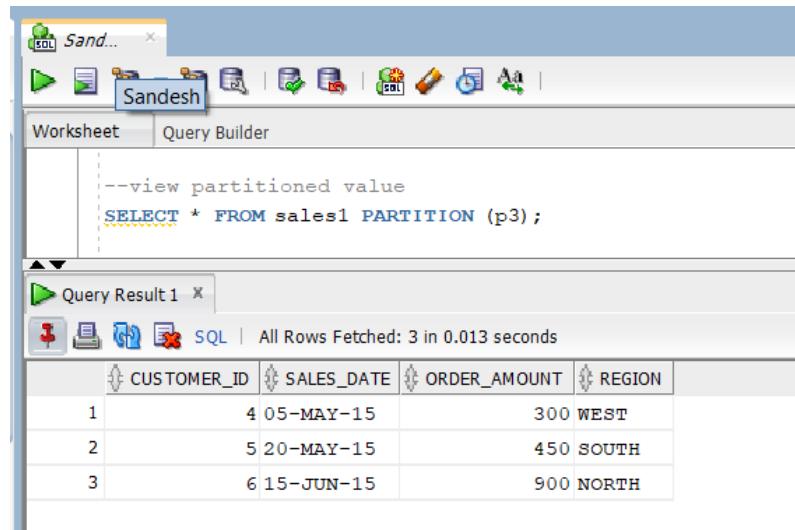
Query Result X

All Rows Fetched: 6 in 0.103 seconds

	CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1	1	01-JAN-15	600	SOUTH
2	2	15-MAR-15	1200	NORTH
3	3	30-APR-15	750	EAST
4	4	05-MAY-15	300	WEST
5	5	20-MAY-15	450	SOUTH
6	6	15-JUN-15	900	NORTH

--view partitioned value

```
SELECT * FROM sales1 PARTITION (p3);
```



The screenshot shows the Oracle SQL Developer interface. At the top, there's a toolbar with various icons. Below the toolbar, the title bar says "Sandesh". Underneath the title bar, there are two tabs: "Worksheet" and "Query Builder". The "Worksheet" tab is selected and contains the SQL query: "SELECT \* FROM sales1 PARTITION (p3);". Below the query, the results are displayed in a table titled "Query Result 1". The table has four columns: CUSTOMER\_ID, SALES\_DATE, ORDER\_AMOUNT, and REGION. The data is as follows:

CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1	4 05-MAY-15	300	WEST
2	5 20-MAY-15	450	SOUTH
3	6 15-JUN-15	900	NORTH

At the bottom of the results window, it says "All Rows Fetched: 3 in 0.013 seconds".

## 2. List partitioning

-- Create table partitioned by LIST on region (

CREATE TABLE sales2

(

customer\_id NUMBER,

sales\_date DATE,

order\_amount NUMBER,

region NVARCHAR2(10)

)

PARTITION BY LIST (region)

(

PARTITION p1 VALUES ('East'),

PARTITION p2 VALUES ('West'),

PARTITION p3 VALUES ('North'),

PARTITION p4 VALUES ('South')

);

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'Worksheet' and contains the SQL code for creating a table with list partitioning. The bottom window is titled 'Script Output' and displays the message 'Table SALES2 created.' indicating the successful execution of the command.

```
-- Create table partitioned by LIST on region (
CREATE TABLE sales2
(
    customer_id NUMBER,
    sales_date DATE,
    order_amount NUMBER,
    region NVARCHAR2(10)
)
PARTITION BY LIST (region)
(
    PARTITION p1 VALUES ('East'),
    PARTITION p2 VALUES ('West'),
    PARTITION p3 VALUES ('North'),
    PARTITION p4 VALUES ('South')
);

```

Script Output

Table SALES2 created.

```
--insert into list partitioning
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (110, TO_DATE('01-FEB-2015','DD-MON-YYYY'), 200, N'East');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (111, TO_DATE('12-FEB-2015','DD-MON-YYYY'), 350, N'East');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (112, TO_DATE('25-FEB-2015','DD-MON-YYYY'), 150, N'East');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (120, TO_DATE('05-MAR-2015','DD-MON-YYYY'), 400, N'West');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (121, TO_DATE('20-MAR-2015','DD-MON-YYYY'), 250, N'West');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (130, TO_DATE('03-APR-2015','DD-MON-YYYY'), 500, N'North');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (131, TO_DATE('15-APR-2015','DD-MON-YYYY'), 275, N'North');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (132, TO_DATE('29-APR-2015','DD-MON-YYYY'), 325, N'North');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (140, TO_DATE('07-MAY-2015','DD-MON-YYYY'), 180, N'South');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (141, TO_DATE('21-MAY-2015','DD-MON-YYYY'), 420, N'South');
COMMIT;
```

Sandesh\_28936...

Worksheet    Query Builder

```
--insert into list partitioning
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (110, TO_DATE('01-FEB-2015','DD-MON-YYYY'), 200, N'East');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (111, TO_DATE('12-FEB-2015','DD-MON-YYYY'), 350, N'East');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (112, TO_DATE('25-FEB-2015','DD-MON-YYYY'), 150, N'East');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (120, TO_DATE('05-MAR-2015','DD-MON-YYYY'), 400, N'West');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (121, TO_DATE('20-MAR-2015','DD-MON-YYYY'), 250, N'West');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (130, TO_DATE('03-APR-2015','DD-MON-YYYY'), 500, N'North');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (131, TO_DATE('15-APR-2015','DD-MON-YYYY'), 275, N'North');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (132, TO_DATE('29-APR-2015','DD-MON-YYYY'), 325, N'North');

INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (140, TO_DATE('07-MAY-2015','DD-MON-YYYY'), 180, N'South');
INSERT INTO sales2 (customer_id, sales_date, order_amount, region)
VALUES (141, TO_DATE('21-MAY-2015','DD-MON-YYYY'), 420, N'South');
COMMIT;
```

Script Output X

1 row inserted.

1 row inserted.

Commit complete.

-- Verify: view all rows ordered by region/date

```
SELECT * FROM sales2 ORDER BY region, sales_date;
```

The screenshot shows a SQL Workbench interface with two panes. The top pane is titled 'Worksheet' and contains the SQL query:

```
-- Verify: view all rows ordered by region/date
SELECT * FROM sales2 ORDER BY region, sales_date;
```

The bottom pane is titled 'Query Result' and displays the results of the query. The results are presented in a table with four columns: CUSTOMER\_ID, SALES\_DATE, ORDER\_AMOUNT, and REGION. The data is as follows:

CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1	110 01-FEB-15	200	East
2	111 12-FEB-15	350	East
3	112 25-FEB-15	150	East
4	130 03-APR-15	500	North
5	131 15-APR-15	275	North
6	132 29-APR-15	325	North
7	140 07-MAY-15	180	South
8	141 21-MAY-15	420	South
9	120 05-MAR-15	400	West
10	121 20-MAR-15	250	West

--view partitioned value

```
SELECT * FROM sales2 PARTITION (p1);
```

The screenshot shows a SQL Workbench interface with two panes. The top pane is titled 'Worksheet' and contains the SQL query:

```
--view partitioned value
SELECT * FROM sales2 PARTITION (p1);
```

The bottom pane is titled 'Query Result' and displays the results of the query. The results are presented in a table with four columns: CUSTOMER\_ID, SALES\_DATE, ORDER\_AMOUNT, and REGION. The data is as follows:

CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1	110 01-FEB-15	200	East
2	111 12-FEB-15	350	East
3	112 25-FEB-15	150	East

### 3. Hash Partitioning

-- Create table with HASH partitioning and named partitions so PARTITION(...) selects work

```
CREATE TABLE sales3
```

```
(
```

```
    customer_id NUMBER,
```

```
    sales_date DATE,
```

```
    order_amount NUMBER,
```

```
    region NVARCHAR2(10)
```

```
)
```

```
PARTITION BY HASH (customer_id)
```

```
(
```

```
    PARTITION p1,
```

```
    PARTITION p2,
```

```
    PARTITION p3,
```

```
    PARTITION p4
```

```
);
```

The screenshot shows the Oracle SQL Developer interface. The top window is titled "Sandesh\_28936..." and contains a worksheet with the following SQL code:

```
---3. Hash Partition
-----
-- Create table with HASH partitioning
CREATE TABLE sales3
(
    customer_id NUMBER,
    sales_date DATE,
    order_amount NUMBER,
    region NVARCHAR2(10)
)
PARTITION BY HASH (customer_id)
(
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4
);
```

The "Query Result" tab at the bottom shows the output:

```
Table SALES3 created.
```

```
--insert values for hash partitioning
```

```
INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
```

```
VALUES (11, TO_DATE('01-FEB-2015','DD-MON-YYYY'), 1200, N'West');
```

```
INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
```

```
VALUES (12, TO_DATE('10-FEB-2015','DD-MON-YYYY'), 450, N'East');
```

```
INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
```

```
VALUES (13, TO_DATE('20-FEB-2015','DD-MON-YYYY'), 800, N'South');
```

```
INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
```

```
VALUES (14, TO_DATE('28-FEB-2015','DD-MON-YYYY'), 300, N'North');
```

```
COMMIT;
```

The screenshot shows the Oracle SQL Developer interface. The top window is titled "Sandesh\_28936..". It contains a "Worksheet" tab with the following SQL code:

```
--insert values for hash partitioning
INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
VALUES (11, TO_DATE('01-FEB-2015','DD-MON-YYYY'), 1200, N'West');

INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
VALUES (12, TO_DATE('10-FEB-2015','DD-MON-YYYY'), 450, N'East');

INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
VALUES (13, TO_DATE('20-FEB-2015','DD-MON-YYYY'), 800, N'South');

INSERT INTO sales3 (customer_id, sales_date, order_amount, region)
VALUES (14, TO_DATE('28-FEB-2015','DD-MON-YYYY'), 300, N'North');

COMMIT;
```

The bottom window is titled "Script Output". It displays the results of the executed SQL statements:

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Commit complete.
```

-- Verify: view all rows

```
SELECT * FROM sales3 ORDER BY customer_id;
```

The screenshot shows the SSMS interface with two panes. The top pane, titled 'Worksheet', contains the SQL query:-- Verify: view all rows  
SELECT \* FROM sales3 ORDER BY customer\_id;

```
The bottom pane, titled 'Query Result', displays the results in a grid format:
```

	CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1		11 01-FEB-15	1200	West
2		12 10-FEB-15	450	East
3		13 20-FEB-15	800	South
4		14 28-FEB-15	300	North

--view from partition

```
SELECT * FROM sales3 PARTITION (p2) ORDER BY customer_id;
```

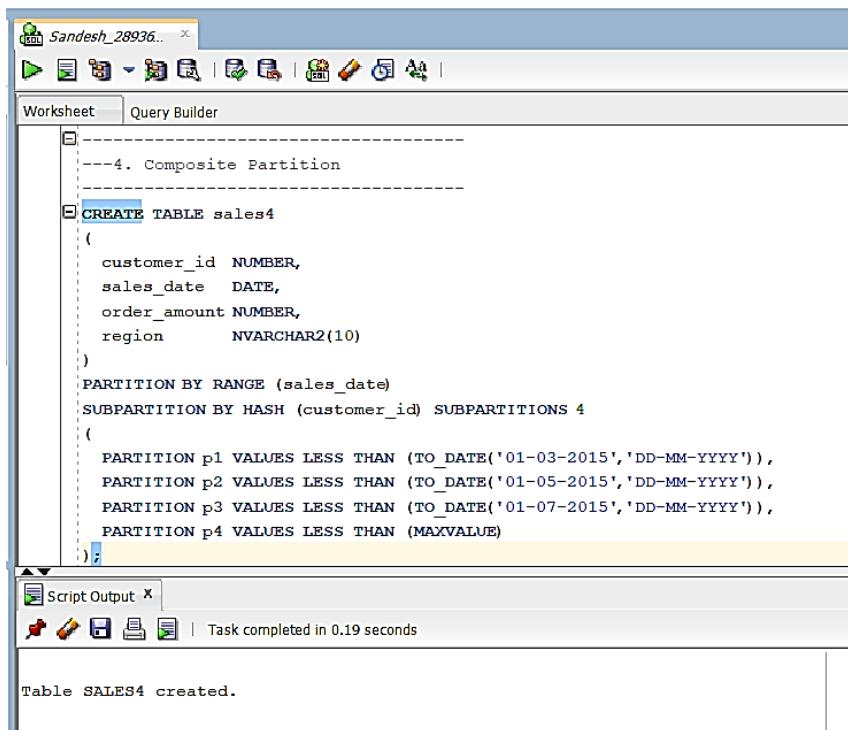
The screenshot shows the SSMS interface with two panes. The top pane, titled 'Worksheet', contains the SQL query:--view from partition  
SELECT \* FROM sales3 PARTITION (p2) ORDER BY customer\_id;

```
The bottom pane, titled 'Query Result', displays the results in a grid format:
```

	CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1		12 10-FEB-15	450	East

#### 4. Composite Partitioning

```
CREATE TABLE sales4
(
    customer_id NUMBER,
    sales_date DATE,
    order_amount NUMBER,
    region NVARCHAR2(10)
)
PARTITION BY RANGE (sales_date)
SUBPARTITION BY HASH (customer_id) SUBPARTITIONS 4
(
    PARTITION p1 VALUES LESS THAN (TO_DATE('01-03-2015','DD-MM-YYYY')),
    PARTITION p2 VALUES LESS THAN (TO_DATE('01-05-2015','DD-MM-YYYY')),
    PARTITION p3 VALUES LESS THAN (TO_DATE('01-07-2015','DD-MM-YYYY')),
    PARTITION p4 VALUES LESS THAN (MAXVALUE)
);
```



The screenshot shows the Oracle SQL Developer interface with a worksheet tab active. The code in the worksheet pane is identical to the one above. Below the worksheet, there is a script output pane which displays the message "Table SALES4 created.".

```
--4. Composite Partition
CREATE TABLE sales4
(
    customer_id NUMBER,
    sales_date DATE,
    order_amount NUMBER,
    region NVARCHAR2(10)
)
PARTITION BY RANGE (sales_date)
SUBPARTITION BY HASH (customer_id) SUBPARTITIONS 4
(
    PARTITION p1 VALUES LESS THAN (TO_DATE('01-03-2015','DD-MM-YYYY')),
    PARTITION p2 VALUES LESS THAN (TO_DATE('01-05-2015','DD-MM-YYYY')),
    PARTITION p3 VALUES LESS THAN (TO_DATE('01-07-2015','DD-MM-YYYY')),
    PARTITION p4 VALUES LESS THAN (MAXVALUE)
);
Table SALES4 created.
```

-- Inserts to composite table

```
-- p1 (date < 01-03-2015) -> 1 row
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (1, TO_DATE('15-01-2015','DD-MM-YYYY'), 500, N'East');

-- p2 (>=01-03-2015 and <01-05-2015) -> 2 rows
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (2, TO_DATE('20-03-2015','DD-MM-YYYY'), 1200, N'West');

INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (3, TO_DATE('10-04-2015','DD-MM-YYYY'), 750, N'North');

-- p3 (>=01-05-2015 and <01-07-2015) -> 2 rows
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (4, TO_DATE('05-05-2015','DD-MM-YYYY'), 300, N'South');

INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (5, TO_DATE('15-06-2015','DD-MM-YYYY'), 900, N'East');

-- p4 (MAXVALUE) -> 1 row (example from your original)
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (111, TO_DATE('02-07-2015','DD-MM-YYYY'), 2100, N'East');

COMMIT;
```

Sandesh\_28936..

Worksheet    Query Builder

```
--insert for composite table
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (1, TO_DATE('15-01-2015','DD-MM-YYYY'), 500, N'East');

INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (2, TO_DATE('20-03-2015','DD-MM-YYYY'), 1200, N'West');
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (3, TO_DATE('10-04-2015','DD-MM-YYYY'), 750, N'North');

INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (4, TO_DATE('05-05-2015','DD-MM-YYYY'), 300, N'South');
INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (5, TO_DATE('15-06-2015','DD-MM-YYYY'), 900, N'East');

INSERT INTO sales4 (customer_id, sales_date, order_amount, region)
VALUES (111, TO_DATE('02-07-2015','DD-MM-YYYY'), 2100, N'East');

COMMIT;
```

Script Output X

| Task completed in 0.367 seconds

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Commit complete.

```
-- Verify: view all rows ordered by date
SELECT * FROM sales4 ORDER BY sales_date;
```

CUSTOMER_ID	SALES_DATE	ORDER_AMOUNT	REGION
1	1 15-JAN-15	500	East
2	2 20-MAR-15	1200	West
3	3 10-APR-15	750	North
4	4 05-MAY-15	300	South
5	5 15-JUN-15	900	East
6	111 02-JUL-15	2100	East

```
-- 1) Show existing partitions (Oracle dictionary)
SELECT partition_name, high_value
FROM user_tab_partitions
WHERE table_name = 'SALES1'
ORDER BY partition_position;
```

PARTITION_NAME	HIGH_VALUE
P1	TO_DATE(' 2015-03-01 00:00:00', 'SYDDD-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA.
P2	TO_DATE(' 2015-05-01 00:00:00', 'SYDDD-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA.
P3	TO_DATE(' 2015-07-01 00:00:00', 'SYDDD-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA.
P4	MAXVALUE

```
-- 2) Drop partition p2
```

```
ALTER TABLE sales1
```

```
    DROP PARTITION p2;
```

The screenshot shows a SQL management interface with two main panes: a Worksheet pane and a Script Output pane.

**Worksheet Pane:** Displays the SQL script being run:

```
-- delete partition
ALTER TABLE sales1
    DROP PARTITION p2;
```

**Script Output Pane:** Shows the results of the executed command:

```
Table SALES1 altered.
```