

UNIT 1

Introduction of RDBMS

(10 Hours) — Database Programming (CACS453)

1. Overview of the Oracle Database Architecture

An Oracle database is a collection of data treated as a unit, whose purpose is to store and retrieve related information. The Oracle architecture is broadly divided into two parts: the Instance (a set of memory structures and background processes that run in main memory) and the Database (the physical set of files stored on disk that hold actual data). Together, the instance and database allow multiple users to access the same data concurrently while ensuring consistency, security, and recoverability.

1.1 Oracle Instance

The instance is created every time the database is started, and it is destroyed when the database is shut down. It consists of two major components:

- System Global Area (SGA) — a shared memory region allocated when the instance starts, used by all server and background processes.
- Background Processes — a set of processes that perform maintenance tasks such as writing data, managing recovery, and coordinating user requests.

SGA Component	Purpose
Database Buffer Cache	Caches data blocks read from datafiles, reducing disk I/O
Redo Log Buffer	Temporarily stores redo entries describing changes to the database
Shared Pool	Caches parsed SQL statements, PL/SQL code, and data dictionary information
Large Pool (optional)	Used for large memory allocations such as backup and recovery operations
Java Pool	Supports Java execution within the database

Background Process	Purpose
DBWn (Database Writer)	Writes modified (dirty) buffers from the buffer cache to datafiles
LGWR (Log Writer)	Writes redo log entries from the redo log buffer to the online redo log files

Background Process	Purpose
CKPT (Checkpoint)	Signals DBWn and updates datafile headers at checkpoints
SMON (System Monitor)	Performs instance recovery at startup and cleans up temporary segments
PMON (Process Monitor)	Cleans up after failed user processes and releases locked resources
ARCn (Archiver, optional)	Copies filled online redo logs to archival storage

1.2 Oracle Database (Physical Storage)

The physical database consists of files stored on disk:

- Datafiles — store the actual data of tables, indexes, and other schema objects.
- Control File — a small binary file recording the physical structure of the database, required to mount and open the database.
- Online Redo Log Files — record all changes made to the database, used for instance recovery.
- Parameter File (PFILE/SPFILE) — contains configuration parameters used to start the instance.
- Password File — authenticates users performing privileged administrative tasks.

Note: The Oracle instance and database are logically distinct: an instance can, in principle, mount a different database, though in practice each instance is normally associated with exactly one database.

1.3 Logical Storage Structure

Internally, Oracle organizes storage in a hierarchy that separates the logical view of data from the physical files on disk:

Logical Unit	Description
Tablespace	Logical storage container that groups related datafiles
Segment	Storage allocated for a specific database object (e.g. a table or index)
Extent	A contiguous set of data blocks allocated to a segment at one time
Data Block	The smallest unit of storage that Oracle reads or writes

2. Familiar with SQL*Plus

SQL*Plus is a command-line interface provided by Oracle that allows users to enter and execute SQL statements, PL/SQL blocks, and SQL*Plus-specific commands. It connects to an Oracle database, sends statements to the server for processing, and formats and displays the results. SQL*Plus is widely used for interactive queries, running scripts, generating reports, and performing administrative tasks.

2.1 Starting SQL*Plus

```
$ sqlplus username/password@database

SQL*Plus: Release 19.0.0.0.0 — Production
Connected to:
Oracle Database 19c Enterprise Edition

SQL>
```

Once connected, the SQL> prompt appears, indicating that SQL*Plus is ready to accept commands. Two broad categories of input are accepted at this prompt:

- SQL and PL/SQL statements — sent to the Oracle server for execution (e.g. SELECT, INSERT, CREATE).
- SQL*Plus commands — processed by SQL*Plus itself and never sent to the server (e.g. DESCRIBE, SET, SPOOL).

2.2 The SQL Buffer

SQL*Plus maintains an area of memory called the SQL buffer, which holds the most recently entered SQL statement. Because the buffer holds only one statement at a time, several editing commands (LIST, APPEND, CHANGE, INPUT, DEL) exist specifically to modify the buffer's contents without retyping the entire statement — these are covered in detail in the next section.

3. SQL*Plus Commands

SQL*Plus provides a set of built-in commands (sometimes called "editing commands") used to display object structures, edit the SQL buffer, and run script files. Unlike SQL statements, these commands do not require a terminating semicolon.

3.1 DESCRIBE

Displays the column definitions (name, nullability, and data type) of a table, view, or the parameters of a stored procedure/function.

Syntax & Example

DESCRIBE table_name; (or abbreviated: DESC table_name;)

SQL> DESC employees;

Sample Output

Name	Null?	Type
EMP_ID	NOT NULL	NUMBER(6)
EMP_NAME	NOT NULL	VARCHAR2(30)
HIRE_DATE		DATE
SALARY		NUMBER(8,2)

3.2 LIST

Displays the current contents of the SQL buffer. The abbreviation L may be used. LIST n displays a specific line and makes it the current line, ready for editing.

```
SQL> LIST
1 SELECT emp_id, emp_name
2* FROM employees
```

Note: The asterisk () marks the current line — the line that subsequent editing commands such as APPEND, CHANGE, and DEL will act upon.*

3.3 APPEND

Adds text to the end of the current line in the buffer. Abbreviation: A.

```
SQL> LIST 2
2* FROM employees
SQL> APPEND WHERE salary > 50000
2* FROM employees WHERE salary > 50000
```

3.4 CHANGE

Replaces the first occurrence of one string with another on the current line. Syntax: C/old/new. Abbreviation: C.

```
SQL> LIST 2
2* FROM employees WHERE salary > 50000
SQL> C/50000/75000
2* FROM employees WHERE salary > 75000
```

3.5 INPUT

Inserts one or more new lines after the current line in the buffer. Abbreviation: I. Typing INPUT alone switches to a mode where each subsequent line typed becomes a new buffer line, until a blank line is entered.

```
SQL> LIST 2
2* FROM employees WHERE salary > 75000
SQL> INPUT ORDER BY emp_name
SQL> LIST
1 SELECT emp_id, emp_name
2 FROM employees WHERE salary > 75000
3* ORDER BY emp_name
```

3.6 DEL

Deletes the current line from the buffer. Variants include DEL n (delete a specific line) and DEL n m (delete a range of lines).

```
SQL> LIST 3
3* ORDER BY emp_name
SQL> DEL
SQL> LIST
1 SELECT emp_id, emp_name
2* FROM employees WHERE salary > 75000
```

3.7 CLEAR BUFFER

Removes all lines from the SQL buffer, leaving it completely empty. This is useful before typing a fresh statement to avoid accidentally combining it with leftover buffer content.

```
SQL> CLEAR BUFFER
buffer cleared
```

3.8 Using Script Files

Rather than typing statements interactively, sequences of SQL and SQL*Plus commands can be saved to a text file (conventionally with a .sql extension) and executed as a unit. This is essential for repeatable tasks such as creating schemas or running reports.

Command	Purpose
SAVE filename	Saves the current buffer contents to a script file
EDIT filename	Opens a script file in the default text editor
GET filename	Loads a script file into the SQL buffer without running it
START filename (or @filename)	Runs the commands in a script file
SPOOL filename	Redirects SQL*Plus output to a file until SPOOL OFF is issued

Example — Creating and Running a Script

```
SQL> SAVE create_emp.sql
```

```
SQL> START create_emp.sql; -- or: SQL> @create_emp.sql
```

4. Accepting Values at Runtime

SQL*Plus supports substitution variables, which allow a script or statement to prompt the user for input, making SQL statements reusable with different values each time they run. A substitution variable is prefixed with an ampersand (&) for single-use substitution, or a double ampersand (&&) if the value should be remembered and reused later in the same session without prompting again.

4.1 The Ampersand (&) Substitution

Syntax & Example

```
SELECT emp_name, salary
```

```
FROM employees
```

```
WHERE department_id = &dept_id;
```

When this statement runs, SQL*Plus displays a prompt such as Enter value for dept_id: and substitutes whatever the user types before sending the completed statement to the server. If the same statement is run again, SQL*Plus prompts again — the value is not remembered.

4.2 The Double Ampersand (&&) Substitution

Syntax & Example

```
SELECT * FROM employees
WHERE department_id = &&dept_id;

SELECT COUNT(*) FROM employees
WHERE department_id = &dept_id;
```

Here, &&dept_id is defined once and its value is stored for the remainder of the session, so the second statement reuses it silently, without prompting the user again.

4.3 The ACCEPT Command

The ACCEPT command explicitly prompts the user for a value, optionally validating its data type and displaying a custom prompt message. It gives finer control than a bare substitution variable.

Syntax & Example

```
ACCEPT var_name [datatype] [FORMAT format] [PROMPT text] [HIDE]

ACCEPT dept_id NUMBER PROMPT 'Enter Department ID: '

SELECT emp_name, salary FROM employees
WHERE department_id = &dept_id;
```

4.4 DEFINE and UNDEFINE

Command	Purpose
DEFINE var = value	Creates or assigns a substitution variable without prompting
DEFINE	Lists all currently defined substitution variables
UNDEFINE var	Removes a defined substitution variable

Note: Prompting behaviour and messages can also be customised with SET VERIFY OFF (suppresses the 'old/new value' echo) so that scripts run cleanly.

5. Overview of Fundamental SQL Commands

SQL statements are broadly classified into categories based on their function. The four categories relevant here are DDL, DML, DCL, and query-enhancing constructs such as Joins and Subqueries.

Category	Full Form	Purpose	Example Commands
DDL	Data Definition Language	Defines/modifies database structure	CREATE, ALTER, DROP, TRUNCATE
DML	Data Manipulation Language	Manipulates data within tables	INSERT, UPDATE, DELETE, SELECT
DCL	Data Control Language	Controls access/permissions	GRANT, REVOKE
TCL	Transaction Control Language	Manages transactions	COMMIT, ROLLBACK, SAVEPOINT

5.1 Data Definition Language (DDL)

DDL statements define and modify the structure of database objects such as tables, views, and indexes. DDL statements are auto-committed — each one is finalized immediately and cannot be rolled back.

CREATE — Example

```
CREATE TABLE employees (
  emp_id NUMBER(6) PRIMARY KEY,
  emp_name VARCHAR2(30) NOT NULL,
  hire_date DATE,
  salary NUMBER(8,2)
);
```

ALTER — Example

```
ALTER TABLE employees ADD (department_id NUMBER(4));
ALTER TABLE employees MODIFY (salary NUMBER(10,2));
```

DROP & TRUNCATE — Example

```
DROP TABLE employees; -- removes table structure and data
TRUNCATE TABLE employees; -- removes all rows, keeps structure
```

5.2 Data Manipulation Language (DML)

DML statements manipulate the data stored within tables. Unlike DDL, DML changes are not permanent until a COMMIT is issued, and can be undone with ROLLBACK.

INSERT — Example

```
INSERT INTO employees (emp_id, emp_name, hire_date, salary)
VALUES (101, 'Anita Rai', DATE '2024-01-15', 55000);
```

UPDATE — Example

```
UPDATE employees
SET salary = salary * 1.10
WHERE department_id = 20;
```

DELETE — Example

```
DELETE FROM employees
WHERE hire_date < DATE '2015-01-01';
```

SELECT — Example

```
SELECT emp_name, salary
FROM employees
WHERE salary > 50000
ORDER BY salary DESC;
```

5.3 Data Control Language (DCL)

DCL statements manage user privileges — granting or revoking the ability to perform specific actions on database objects.

GRANT & REVOKE — Example

```
GRANT SELECT, INSERT ON employees TO hr_clerk;
REVOKE INSERT ON employees FROM hr_clerk;
```

5.4 Join

A join combines rows from two or more tables based on a related column, allowing queries to retrieve data that is spread across multiple normalized tables.

Join Type	Description
INNER JOIN	Returns only rows with matching values in both tables

Join Type	Description
LEFT OUTER JOIN	Returns all rows from the left table, matched rows from the right (NULL if no match)
RIGHT OUTER JOIN	Returns all rows from the right table, matched rows from the left (NULL if no match)
FULL OUTER JOIN	Returns all rows from both tables, with NULLs where no match exists
SELF JOIN	Joins a table to itself, typically using table aliases

INNER JOIN — Example

```
SELECT e.emp_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

LEFT OUTER JOIN — Example

```
SELECT e.emp_name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;
```

5.5 Subquery

A subquery (or inner query) is a `SELECT` statement nested inside another SQL statement. It is evaluated first, and its result is used by the outer (main) query. Subqueries may appear in the `WHERE`, `FROM`, or `SELECT` clause and can return a single value, a single column of values, or an entire table of results.

Single-Row Subquery — Example

```
SELECT emp_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

The inner query (`SELECT AVG(salary) FROM employees`) computes the average salary once; the outer query then returns employees earning above that average.

Multi-Row Subquery with IN — Example

```
SELECT emp_name, department_id
FROM employees
WHERE department_id IN
  (SELECT department_id FROM departments WHERE location_id = 1700);
```

Correlated Subquery — Example

```
SELECT e.emp_name, e.salary, e.department_id
FROM employees e
WHERE e.salary > (
  SELECT AVG(salary) FROM employees
  WHERE department_id = e.department_id
);
```

Note: A correlated subquery references a column from the outer query, so it is re-evaluated once for every row processed by the outer query — unlike an ordinary subquery, which executes only once.

Key Takeaways

- ✓ An Oracle instance (SGA + background processes) works together with the physical database (datafiles, control file, redo logs) to manage data.
- ✓ SQL*Plus is the interactive command-line tool for sending SQL/PL-SQL statements and running SQL*Plus-specific commands.
- ✓ Buffer-editing commands — DESCRIBE, LIST, APPEND, CHANGE, INPUT, DEL, CLEAR BUFFER — let you inspect and modify the current statement without retyping it.
- ✓ Script files (.sql) combined with SAVE, GET, START/@ and SPOOL make SQL work repeatable and shareable.
- ✓ Substitution variables (& and &&) and the ACCEPT command let scripts prompt users for values at runtime.
- ✓ DDL, DML, DCL and TCL organize SQL statements by function: defining structure, manipulating data, controlling access, and managing transactions.
- ✓ Joins combine related rows across tables; subqueries nest one query inside another to compute intermediate results.

— End of Unit 1 —