

UNIT 1

Introduction to an Oracle Database

(12 Hours) — Database Administration (CACS405)

*Note: These notes are prepared for study purposes and **may contain errors or omissions**. Syntax, file paths and screen names differ between Oracle versions and operating systems (examples here follow Oracle 19c on Linux). Please verify important points with your textbook, class notes and the official Oracle documentation, and test commands on a practice database, never on a production system.*

1.1 Overview of Oracle Database

Oracle Database is a widely used **Relational Database Management System (RDBMS)** from Oracle Corporation. It stores data in tables (rows and columns), lets users work with it through **SQL** (Structured Query Language) and PL/SQL, and provides security, concurrency control, backup and recovery, and high availability features for enterprise use. Common releases are 12c, 18c, 19c (a long-term-support release), 21c and 23ai. This course follows the 19c style of administration.

A **Database Administrator (DBA)** is the person responsible for installing, configuring, securing, monitoring, tuning and protecting the database so that it is always available, fast and safe.

1.2 Overview of the Oracle Database Architecture

The Oracle server consists of two main parts: the **instance** and the **database**. The **instance** is the set of memory structures and background processes running on the server. The **database** is the set of physical files stored on disk. An instance opens exactly one database at a time (in a single-instance setup), and users reach data only through the instance.

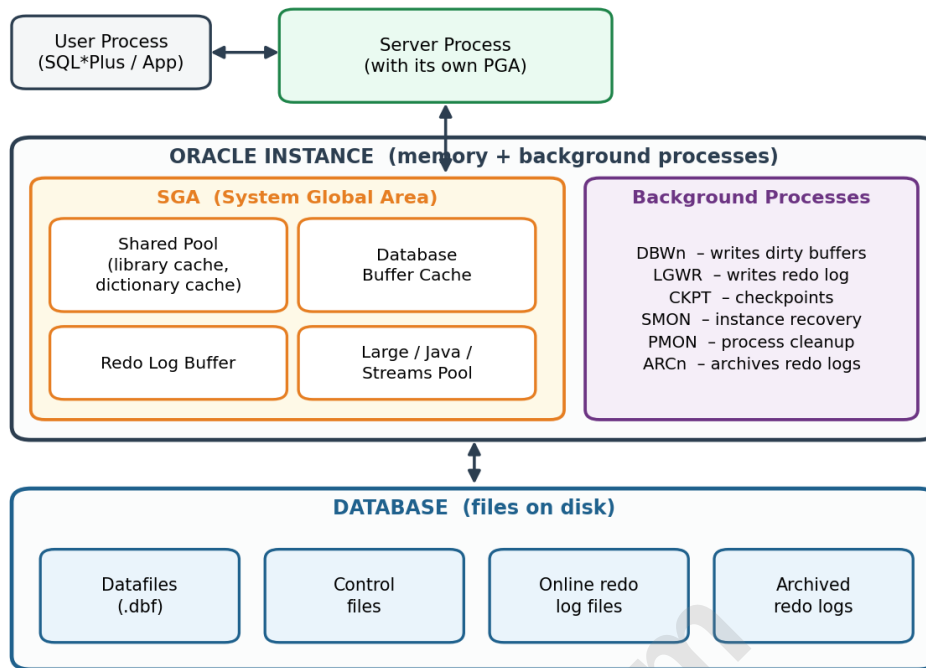


Figure 1.1: Oracle instance (memory + processes) and database (files) — simplified

1.2.1 Memory Structures

System Global Area (SGA) is a shared memory region allocated when the instance starts and shared by all server and background processes. **Program Global Area (PGA)** is private memory belonging to one server process (holding sort areas, session variables, cursor state).

SGA component	Purpose
Database Buffer Cache	Holds copies of data blocks read from datafiles. Frequently used blocks stay in memory so disk reads are avoided.
Shared Pool	Contains the library cache (parsed SQL and PL/SQL, execution plans) and the data dictionary cache (table, column and privilege information).
Redo Log Buffer	A small circular buffer that temporarily holds redo entries (change records) before LGWR writes them to the online redo log.
Large Pool	Optional area for large memory allocations such as shared-server session memory, RMAN backup and parallel execution buffers.
Java Pool / Streams Pool	Memory for Java code in the database and for Oracle Streams / GoldenGate style processing.

Note: SGA size is controlled by SGA_TARGET (automatic shared memory management) and PGA size by PGA_AGGREGATE_TARGET. MEMORY_TARGET (automatic memory management) can manage both together on supported platforms.

1.2.2 Process Structures

- **User process** — the client program (SQL*Plus, an application) that requests work. It does not touch database files directly.
- **Server process** — created on behalf of a user session; parses and executes SQL, reads blocks into the buffer cache and returns results.
- **Background processes** — started with the instance to perform housekeeping and I/O.

Process	Full name	Main job
DBWn	Database Writer	Writes modified (dirty) buffers from the buffer cache to the datafiles.
LGWR	Log Writer	Writes redo entries from the redo log buffer to the online redo log files (at commit, every 3 seconds, etc.).
CKPT	Checkpoint	Signals DBWn at a checkpoint and updates datafile headers and control file with checkpoint information.
SMON	System Monitor	Performs instance recovery at startup and cleans up temporary segments.
PMON	Process Monitor	Cleans up after failed user processes (releases locks and resources).
ARCn	Archiver	Copies filled online redo log groups to the archive destination (ARCHIVELOG mode).
LREG	Listener Registration	Registers instance and service information with the listener (from 12c; earlier done by PMON).
MMON / MMNL	Manageability Monitor	Collects AWR statistics and issues alerts.

1.2.3 Storage Structures

Physical structures are the files visible in the operating system: **datafiles** (actual data), **control files** (structure metadata), **online redo log files** (record of changes), **archived redo logs**, the parameter file (**SPFILE/PFILE**), the password file and diagnostic files such as the alert log. **Logical structures** are how Oracle organises space: **tablespace** → **segment** → **extent** → **data block** (covered in Unit 3).

1.2.4 How a Statement is Processed

- **Parse** — check syntax and privileges, look in the library cache for an existing plan (soft parse) or build a new one (hard parse).
- **Execute** — required blocks are read into the buffer cache; changes are made there and redo/undo is generated.
- **Fetch** — for queries, rows are returned to the user process.
- **Commit** — LGWR writes the redo to the online log; the transaction is then durable. DBWn writes the changed blocks later.

1.3 DBA Roles and Responsibilities

- Installing and upgrading Oracle software, applying patches, creating databases.
- Planning storage and capacity: tablespaces, datafiles, growth monitoring.
- Creating and managing users, roles, privileges and profiles (security).
- Backup, recovery and disaster-recovery planning; testing restores regularly.
- Monitoring performance, tuning SQL and memory, and reading the alert log.
- Managing network connectivity (listener, service names).
- Automating routine jobs, documentation, and working with developers on schema design.

Administrative accounts: **SYS** owns the data dictionary and has the highest authority; **SYSTEM** is used for administrative tasks and tools. Special administrative privileges are **SYSDBA** (full control, can start/stop and recover the database) and **SYSOPER** (start/stop and basic operations). Do not use **SYS** for routine work.

1.4 Familiar with SQL*Plus

SQL*Plus is Oracle's command-line tool for running SQL, PL/SQL and administrative commands. It can also run scripts and produce simple reports.

```

$ sqlplus / as sysdba          -- OS authentication as SYSDBA
$ sqlplus hr/hr_pwd@orclpdb1   -- user/password@service_name
$ sqlplus hr@//dbserver:1521/orclpdb1 -- Easy Connect

SQL> SHOW USER
SQL> SET LINESIZE 150
SQL> SET PAGESIZE 50
SQL> COLUMN emp_name FORMAT A20
SQL> DESCRIBE employees
SQL> SPOOL /tmp/result.txt      -- save output to a file
SQL> SPOOL OFF
SQL> @/home/oracle/scripts/report.sql -- run a script
SQL> EXIT

```

*Listing 1.1: Connecting and common SQL*Plus commands*

Command	Meaning
DESCRIBE table	Shows column names, null status and data types.
SET LINESIZE n / PAGESIZE n	Sets line width and number of lines per page of output.
COLUMN col FORMAT ...	Formats the display of a column.
SPOOL file / SPOOL OFF	Writes output to a file / stops writing.
@file or START file	Executes a script file.
SHOW USER, SHOW PARAMETER	Displays the current user or parameter values.
EDIT, /, RUN	Edit the SQL buffer; run the last statement.

*Note: SQL*Plus commands (like SET, DESCRIBE, SPOOL) do not need a semicolon, whereas SQL statements end with ; and PL/SQL blocks end with /.*

1.5 Accepting Values at Runtime

SQL*Plus lets a script ask the user for input using **substitution variables**. A variable starting with a single ampersand & prompts every time it is used, while a double ampersand && prompts once and remembers the value for the session. ACCEPT defines a variable with a custom prompt, and DEFINE / UNDEFINE create or remove variables.

```

SET VERIFY OFF                                -- do not show old/new statement text
ACCEPT v_dept NUMBER PROMPT 'Enter department id: '
SELECT emp_name, salary
FROM   employees
WHERE  dept_id = &v_dept;

-- text values need quotes
SELECT * FROM employees WHERE emp_name = '&name';

-- && remembers the value
SELECT &&col_name FROM employees ORDER BY &col_name;
UNDEFINE col_name

```

Listing 1.2: Substitution variables

Example 1.1

Write a query that asks the user for a minimum salary and lists employees earning at least that amount.

Solution

Use ACCEPT (or &min_sal) in the WHERE clause:

```

ACCEPT min_sal NUMBER PROMPT 'Minimum salary: '
SELECT emp_id, emp_name, salary FROM employees WHERE salary >=
&min_sal;

```

1.6 Overview of SQL Commands

Category	Commands	Purpose
DDL — Data Definition	CREATE, ALTER, DROP, TRUNCATE, RENAME	Define or change database objects. Auto-commit.
DML — Data Manipulation	INSERT, UPDATE, DELETE, MERGE	Change data in tables.
DQL — Query	SELECT	Retrieve data.
TCL — Transaction Control	COMMIT, ROLLBACK, SAVEPOINT	Control transactions.
DCL — Data Control	GRANT, REVOKE	Manage privileges.

The examples below use two sample tables, **departments** and **employees**.

1.6.1 DDL: Tables and Constraints

A **table** is the basic unit of data storage. **Constraints** enforce data integrity rules: NOT NULL, UNIQUE, PRIMARY KEY (unique + not null), FOREIGN KEY (references a parent table) and CHECK (a condition). DEFAULT supplies a value when none is given.

```
CREATE TABLE departments (  
  dept_id    NUMBER(4)    CONSTRAINT dept_pk PRIMARY KEY,  
  dept_name  VARCHAR2(30) NOT NULL UNIQUE,  
  location   VARCHAR2(30)  
);  
  
CREATE TABLE employees (  
  emp_id     NUMBER(6)    PRIMARY KEY,  
  emp_name   VARCHAR2(40) NOT NULL,  
  email      VARCHAR2(50) UNIQUE,  
  salary     NUMBER(8,2)  CHECK (salary > 0),  
  hire_date  DATE         DEFAULT SYSDATE,  
  dept_id    NUMBER(4)    CONSTRAINT emp_dept_fk  
                                REFERENCES departments(dept_id)  
);  
  
ALTER TABLE employees ADD (phone VARCHAR2(15));  
ALTER TABLE employees MODIFY (emp_name VARCHAR2(60));  
ALTER TABLE employees DROP COLUMN phone;  
TRUNCATE TABLE temp_data;      -- removes all rows quickly, cannot rollback  
DROP TABLE temp_data PURGE;    -- removes table completely
```

Listing 1.3: Creating and altering tables

1.6.2 DDL: Indexes

An **index** is a separate structure that lets Oracle find rows without scanning the whole table. It speeds up queries but slows DML slightly and uses space. Oracle automatically creates a unique index for each primary key and unique constraint.

Index type	Description / use
B-tree (default)	Balanced tree; best for columns with many distinct values.
Bitmap	Uses bitmaps; good for low-cardinality columns in read-mostly (data warehouse) systems.
Unique / Composite	Unique enforces uniqueness; composite covers several columns.
Function-based	Index on an expression, e.g. UPPER(emp_name).

```
CREATE INDEX emp_name_idx ON employees(emp_name);  
CREATE UNIQUE INDEX emp_email_uk ON employees(email);  
CREATE BITMAP INDEX emp_dept_bix ON employees(dept_id);  
CREATE INDEX emp_upper_idx ON employees(UPPER(emp_name));  
DROP INDEX emp_name_idx;
```

1.6.3 DDL: Views, Synonyms and Sequences

A **view** is a stored query that behaves like a virtual table; it stores no data. Views simplify complex queries and hide columns for security. `WITH READ ONLY` blocks DML through the view and `WITH CHECK OPTION` ensures changed rows still satisfy the view's condition. A **synonym** is an alias for an object (table, view, sequence, procedure). A **public** synonym is visible to all users. A **sequence** generates unique numbers, usually for primary keys, using `NEXTVAL` and `CURRVAL`.

```
CREATE OR REPLACE VIEW emp_it AS
  SELECT emp_id, emp_name, salary FROM employees WHERE dept_id = 10
  WITH READ ONLY;

CREATE SYNONYM emp FOR hr.employees;
CREATE PUBLIC SYNONYM dept FOR hr.departments;  -- needs privilege

CREATE SEQUENCE emp_seq START WITH 1000 INCREMENT BY 1
  NOCACHE NOCYCLE;
INSERT INTO employees (emp_id, emp_name, salary, dept_id)
VALUES (emp_seq.NEXTVAL, 'Sita Sharma', 55000, 10);
SELECT emp_seq.CURRVAL FROM dual;
```

Listing 1.4: View, synonym and sequence

1.6.4 DDL: Partitioning

Partitioning divides a large table or index into smaller pieces called partitions, each stored separately but accessed as one object. Benefits: better performance through *partition pruning* (only relevant partitions are read), easier maintenance (drop or archive one partition), and improved availability. Partitioning is a separately licensed option (Enterprise Edition).

Method	How rows are placed
Range	By ranges of the partition key (e.g. dates).
List	By explicit lists of values (e.g. region).
Hash	By a hash function, spreading rows evenly.
Composite	A combination, e.g. range-hash or range-list.

```

CREATE TABLE sales (
  sale_id    NUMBER,
  sale_date  DATE,
  amount     NUMBER(10,2)
)
PARTITION BY RANGE (sale_date) (
  PARTITION p2024 VALUES LESS THAN (TO_DATE('01-JAN-2025','DD-MON-YYYY')),
  PARTITION p2025 VALUES LESS THAN (TO_DATE('01-JAN-2026','DD-MON-YYYY')),
  PARTITION pmax  VALUES LESS THAN (MAXVALUE)
);
SELECT * FROM sales PARTITION (p2025);

```

Listing 1.5: Range partitioned table

1.6.5 DDL: Materialized Views

A **materialized view (MV)** stores the result of a query physically, so expensive joins and aggregates need not be recomputed each time. It must be **refreshed** to stay current. Refresh methods: **COMPLETE** (re-run the whole query), **FAST** (apply only changes, needs a *materialized view log* on the base tables) and **FORCE** (fast if possible, otherwise complete). Refresh timing: **ON DEMAND** (manual) or **ON COMMIT**.

```

CREATE MATERIALIZED VIEW mv_dept_salary
  BUILD IMMEDIATE
  REFRESH COMPLETE ON DEMAND
AS
  SELECT dept_id, SUM(salary) total_sal, COUNT(*) emp_count
  FROM employees GROUP BY dept_id;

EXEC DBMS_MVIEW.REFRESH('MV_DEPT_SALARY', 'C');  -- C = complete

-- for fast refresh a log is needed on the base table
CREATE MATERIALIZED VIEW LOG ON employees WITH PRIMARY KEY;

```

Listing 1.6: Materialized view

Feature	View	Materialized view
Stores data?	No — query only	Yes — result stored on disk
Data currency	Always current	As current as the last refresh
Speed for heavy queries	Recomputes each time	Fast (precomputed)

1.6.6 DML and Transaction Control

```
INSERT INTO departments VALUES (10, 'IT', 'Kathmandu');
INSERT INTO employees (emp_id, emp_name, salary, dept_id)
VALUES (1, 'Ram Thapa', 60000, 10);

UPDATE employees SET salary = salary * 1.10 WHERE dept_id = 10;
DELETE FROM employees WHERE emp_id = 1;

SAVEPOINT before_bonus;
UPDATE employees SET salary = salary + 5000;
ROLLBACK TO before_bonus;      -- undo only the update
COMMIT;                        -- make changes permanent

MERGE INTO emp_backup b
USING employees e ON (b.emp_id = e.emp_id)
WHEN MATCHED THEN UPDATE SET b.salary = e.salary
WHEN NOT MATCHED THEN INSERT (emp_id, emp_name, salary)
VALUES (e.emp_id, e.emp_name, e.salary);
```

Listing 1.7: DML and transactions

1.6.7 Joins

A **join** combines rows from two or more tables based on a related column.

Join type	Result
Inner join	Only rows that match in both tables.
Left outer join	All rows from the left table plus matches from the right (NULLs where none).
Right outer join	All rows from the right table plus matches from the left.
Full outer join	All rows from both tables.
Cross join	Cartesian product (every row with every row).
Self join	A table joined with itself (e.g. employee and manager).

```

-- inner join (ANSI syntax)
SELECT e.emp_name, d.dept_name
FROM   employees e JOIN departments d ON e.dept_id = d.dept_id;

-- left outer join: employees even without a department
SELECT e.emp_name, d.dept_name
FROM   employees e LEFT OUTER JOIN departments d
      ON e.dept_id = d.dept_id;

-- Oracle traditional syntax (outer join marker (+))
SELECT e.emp_name, d.dept_name
FROM   employees e, departments d
WHERE  e.dept_id = d.dept_id(+);

-- self join
SELECT w.emp_name worker, m.emp_name manager
FROM   employees w JOIN employees m ON w.manager_id = m.emp_id;

```

Listing 1.8: Join examples

1.6.8 Subqueries

A **subquery** is a query nested inside another statement. A **single-row** subquery returns one row and uses operators such as = or >. A **multiple-row** subquery uses IN, ANY or ALL. A **correlated** subquery refers to a column of the outer query and runs once per outer row. EXISTS tests whether a subquery returns any row. A subquery in the FROM clause is called an **inline view**.

```

-- single-row subquery
SELECT emp_name FROM employees
WHERE  salary > (SELECT AVG(salary) FROM employees);

-- multiple-row subquery
SELECT emp_name FROM employees
WHERE  dept_id IN (SELECT dept_id FROM departments
                  WHERE location = 'Kathmandu');

-- correlated subquery: earn more than their own department average
SELECT e.emp_name FROM employees e
WHERE  e.salary > (SELECT AVG(salary) FROM employees
                  WHERE dept_id = e.dept_id);

-- EXISTS
SELECT d.dept_name FROM departments d
WHERE  EXISTS (SELECT 1 FROM employees e WHERE e.dept_id = d.dept_id);

-- inline view: top 3 salaries (12c+)
SELECT * FROM (SELECT emp_name, salary FROM employees ORDER BY salary DESC)
FETCH FIRST 3 ROWS ONLY;

```

Listing 1.9: Subquery examples

Example 1.2

List the names of departments that have no employees.

Solution

```
SELECT dept_name FROM departments d WHERE NOT EXISTS (SELECT 1 FROM
employees e WHERE e.dept_id = d.dept_id);
```

The correlated subquery checks each department; departments with no matching employee row are returned.

Key Takeaways

- ✓ Oracle = **instance** (SGA + background processes) + **database** (datafiles, control files, redo logs).
- ✓ SGA is shared; PGA is private to each server process. Key processes: DBWn, LGWR, CKPT, SMON, PMON, ARCn.
- ✓ DBA duties: install, secure, monitor, back up/recover, tune and automate. SYS and SYSTEM are administrative accounts.
- ✓ SQL*Plus: &var prompts each time, &&var remembers; ACCEPT gives a custom prompt.
- ✓ DDL covers tables, constraints, indexes, views, synonyms, sequences, partitions and materialized views; DDL auto-commits.
- ✓ Joins combine tables; subqueries can be single-row, multi-row, correlated or inline views.

Reminder: these notes may contain errors — cross-check with the official syllabus, your textbook and Oracle documentation.

— End of Unit 1 —