Laboratory Enterprise Database Systems I	
Laboratory 01	**Database Instance Administration** ***Inspecting instance parameters and monitoring Oracle processes***

Laboratory preparation

Requirements
Windows workstation or VM running Oracle Database 21c XE
SQL*Plus and SQLcl or SQL Developer available
Credentials for the SYS and SYSTEM accounts
Oracle XE with the XEPDB1 pluggable database

Introduction and theoretical background

Purpose. This introductory laboratory familiarizes students with the administration interfaces of an Oracle Database system running on Microsoft Windows. Students learn how to establish a database connection and identify the database, instance, storage files, initialization parameters, memory structures, and background processes. The tasks focus on reading and interpreting actual values rather than changing configuration. Each SQL statement may be copied and executed directly in a supported Oracle client.

1. Oracle Database in an enterprise environment

Oracle Database is a relational database management system (DBMS) designed to store and process data for multiple applications and users. Alongside SQL execution, it provides access control, transaction handling, concurrency control, crash recovery, storage management, backup facilities, and performance monitoring. The word *enterprise* describes capabilities needed when a database operates continuously, serves many connections, and must provide controlled access to shared resources.

In this laboratory the server is **Oracle Database 21c Express Edition (XE)** installed on Windows. XE is subject to resource and usage limits, but it exposes the administrative views needed to examine the core Oracle architecture. The examples refer to the local host and to port `1521`. When Oracle runs on another Windows computer, substitute that host's address. Paths returned by the database depend on its actual installation.

A typical workflow begins with a client connection, continues with a `SELECT` statement, and ends with examination of the returned data. The same SQL statement can be executed in SQL*Plus, SQLcl, or SQL Developer. Their appearance differs, but the database engine processes the SQL.

2. Database, instance, and connection

The term **database** primarily refers to the persistent physical files used to store the data. An **instance** consists of memory areas and Oracle execution mechanisms responsible for opening and operating those files. This distinction is important: database files remain on disk after an instance has been shut down, whereas instance memory is released. On startup Oracle reads initialization parameters, allocates memory, and starts or activates the required background operations.

An authenticated **session** is the logical context in which a user runs statements. Several sessions can use the same instance at once. An Oracle **service name** identifies a database service for connection purposes. It is not necessarily identical to the database name or instance name.

Three different concepts: database = persistent files; instance = memory and execution mechanisms; session = an authenticated connection context. Use `V$DATABASE`, `V$INSTANCE`, and `V$SESSION` to inspect different parts of this environment.

3. Multitenant architecture: CDB and PDB

Oracle Database 21c uses a multitenant architecture. A **container database (CDB)** contains the root container `CDB$ROOT`, the template container `PDB$SEED`, and one or more **pluggable databases (PDBs)**. The example environment includes `XEPDB1`. A connection to the `XE` service commonly reaches the root container, while `XEPDB1` commonly refers to the pluggable database service. The actual container is determined by the service configuration and should always be verified.

A PDB can contain application data, its own objects, and local users. Administrative views may produce different rows depending on the current container and privileges. For example, the list of users or the set of data files in `XEPDB1` may differ from the corresponding results in `CDB$ROOT`. For this reason, before comparing outputs record *which container* your session is using.

```
SELECT SYS_CONTEXT('USERENV','CON_NAME') AS CONTAINER
FROM DUAL;

SHOW CON_NAME
```

4. Physical structure: file types and tablespaces

Data files store data blocks used by segments, such as tables and indexes. **Tablespaces** are logical storage units that organize these segments and reference one or more physical files. **Temporary files** provide working space for operations such as disk-based sorting. **Control files** record essential information about database structure and file identities. **Online redo log files** store change records used for recovery after failures.

The *size* of a file, the *free space* inside a tablespace, and the *status* of a redo log group describe different things. Students should read the column headers before interpreting values. File paths in Oracle's administration views are Windows paths because the database server is running on Windows, even if a client connects from elsewhere.

Diagnostic **trace files** help administrators inspect errors and internal activities. In recent Oracle versions, diagnostic paths can be queried from `V$DIAG_INFO`; older initialization parameters may remain visible for compatibility but do not always provide the most useful path.

5. Oracle memory: SGA, PGA and buffers

The **System Global Area (SGA)** is memory shared by activities belonging to an Oracle instance. Important components include the *Database Buffer Cache* (cached database blocks), the *Shared Pool* (shared information for executing statements), and the *Redo Log Buffer* (change records waiting to be written to redo logs). Oracle exposes aggregate and detailed SGA information through views such as `V$SGA`, `V$SGAINFO`, and `V$SGASTAT`.

The **Program Global Area (PGA)** is memory used by an Oracle process or execution context, for example for sort or hash operations and other per-process data. PGA and SGA differ in sharing model and purpose. A memory allocation currently reported by a process is not the same as an initialization parameter indicating a target or upper limit.

Some tasks compute a simplified *buffer cache hit ratio* from cumulative instance statistics. The ratio is useful for learning how statistics relate to each other, but it must not be treated as a complete performance assessment. Likewise, an estimate derived from `sort_area_size` and `hash_area_size` is not a measurement of total PGA consumption.

6. Oracle background processes and Windows

An Oracle instance performs work through execution contexts and background operations. Examples include **LGWR**, responsible for writing redo data, **DBWn**, associated with writing dirty database blocks, **SMON**, which performs instance recovery and maintenance tasks, and **PMON**, associated with process monitoring

and cleanup coordination. The actual set varies between versions and configurations, so the names should be confirmed in the returned data.

On Windows, many Oracle activities are implemented as threads within operating-system processes rather than as separate executables with matching names. Consequently, a row in V$PROCESS or V$BGPROCESS need not represent a separately named Windows process. The PADDR and ADDR values may help relate background descriptors to process records. Resource limits for sessions and processes can be examined in V$RESOURCE_LIMIT.

7. Initialization parameters and parameter files

Oracle reads initialization parameters during startup. Parameters control such things as process capacity, session capacity, memory configuration, file locations, and operation of database components. The two parameter-file formats are **PFILE**, a text file, and **SPFILE**, a server parameter file managed by Oracle. An SPFILE must not be edited manually in a text editor.

The parameter views serve related but different purposes. V$PARAMETER describes values effective for a session. V$PARAMETER2 expands parameters with multiple values. V$SYSTEM_PARAMETER describes instance-wide effective values, while V$SPPARAMETER reports parameter entries held in the SPFILE. An SPFILE entry can differ from an effective value if a configuration change has not yet taken effect.

8. Data dictionary and performance views

The **data dictionary** contains metadata: definitions and properties of users, tablespaces, objects, and other database structures. Administrative views beginning with DBA_ commonly expose database-wide metadata to sufficiently privileged accounts. Dynamic performance views beginning with V$ expose information about a running instance or its runtime activity.

A view's output is meaningful only together with its column definitions. For example, a numeric VALUE could represent a number of bytes, a count of operations, or a time quantity in microseconds. Do not assume that all numeric columns use the same unit.

9. SYS, SYSTEM and database privileges

Oracle provides two built-in administrative accounts used in this laboratory. SYS is a highly privileged account, conventionally used for administrative operations with the SYSDBA privilege. SYSTEM is a built-in administrative account that can be used through an ordinary connection with the client role setting Default. The role setting in SQL Developer changes how administrative authentication is requested.

Using a privileged account does not mean every exercise needs to modify the database. All exercises in this introductory laboratory are read-only. A client must know the account, a valid password, and where applicable the host, port, and service. The password should be entered at a client prompt, not typed into saved scripts.

10. Reading results and verifying a live environment

The data collected during the laboratory describe the database *as it is running*. Process counts, memory usage, account statuses, and some performance statistics can vary with configuration and workload. Therefore, a value copied from documentation or another workstation is not evidence of the student's own query result.

At the end of the session, the instructor will ask students to display selected values directly from their connected Oracle instance. During the check, the student must identify the current connection and container, run the indicated SQL statement, and point to the relevant column and value. No written report is required.

> **Operating rule:** Run only the read-only statements shown in this handout. Do not create or remove database users, change parameters, or edit database files. Use the actual values observed in your own instance for the end-of-class check.

*Connection guide: SQL*Plus, SQLcl and SQL Developer*

The following examples reproduce the detailed login workflows illustrated in the accompanying screenshots. They cover both authentication at program startup and reconnecting within an active session. Usernames, passwords, connection roles, and Oracle service names must be entered in the correct fields.

*Example 1. SQL*Plus: initial login and reconnecting*

Step 1. Start SQL*Plus on Windows (for example, from the Start menu). At the `Enter user-name` prompt, supply the account, service, and administrative privilege. Enter the password only after the separate password prompt appears. SYS or SYSTEM account password is system

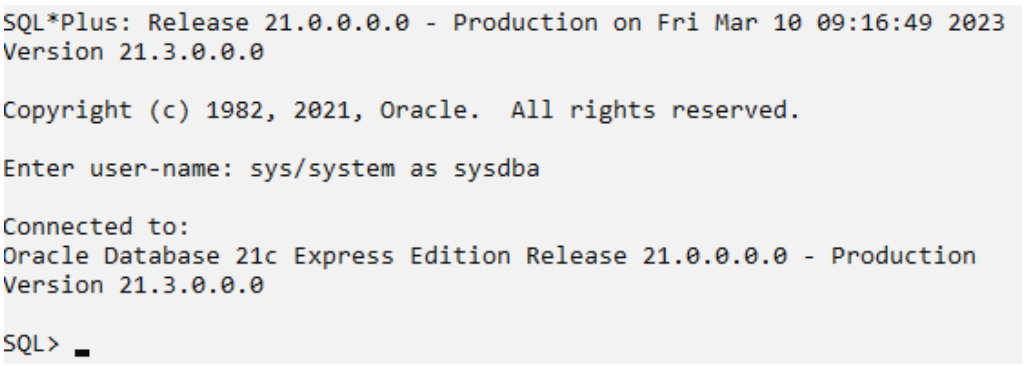
Simplified login:

```
Enter user-name: sys as sysdba
Enter password: [SYS account password]
```

Full login:

```
Enter user-name: SYS/system@localhost:1521/XE AS SYSDBA
```

After successful authentication, SQL*Plus displays information about the database version and the prompt `SQL>`. The screenshot below shows a connected SQL*Plus console.

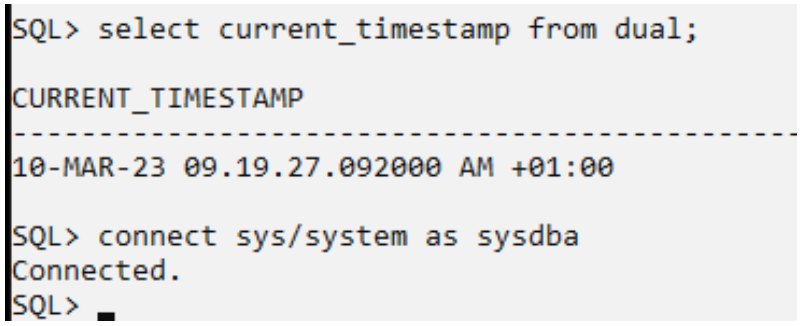


*SQL*Plus: initial authentication and the connected console (source screenshot).*

Step 2. Once the console is open, run a simple test query. To switch to the SYSTEM account, execute the full `CONNECT` statement. SQL*Plus requests the new account password.

```
SELECT CURRENT_TIMESTAMP FROM DUAL;

CONNECT SYSTEM/system@localhost:1521/XEPDB1
```



*SQL*Plus: executing a query and reconnecting from an open session.*

Step 3. Verify the logged-in database user and the current container, rather than relying on the saved connection name.

```
SELECT USER FROM DUAL;
SELECT SYS_CONTEXT('USERENV','CON_NAME') FROM DUAL;
```

> Older examples sometimes place a password immediately after a slash in a connection string. While this syntax may be accepted, it can expose passwords in process listings or command history. Use the password prompt in the laboratory.

Example 2. SQLcl: initial login and reconnecting

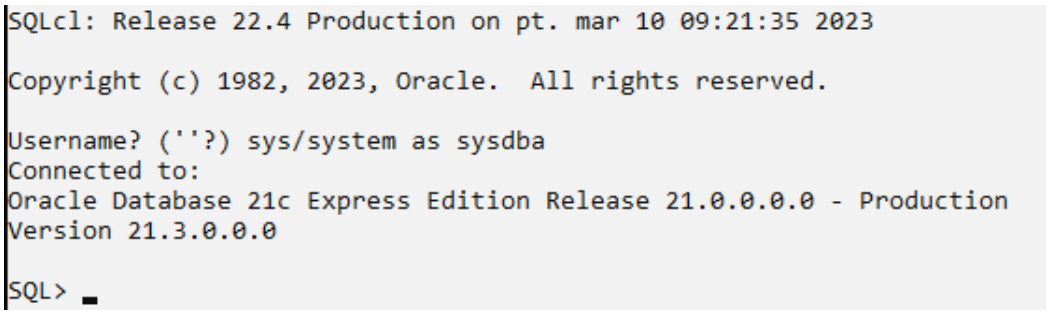
Step 1. Start SQLcl by executing `sqlcl.exe` on the desktop. At its `Username?` prompt, enter the user, service, and privilege required for the initial administrative connection.

Simplified login:

```
Username? sys as sysdba
Password? [SYS account password]
```

Full login:

```
Enter user-name: SYS/system@localhost:1521/XE AS SYSDBA
```



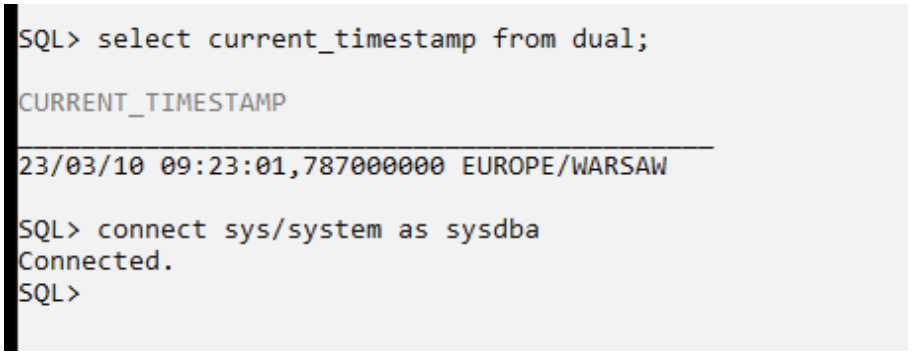
SQLcl: authentication at program startup.

Step 2. Enter a test query and switch to another user with `CONNECT` in the existing SQLcl console. Simplified login:

```
Enter user-name: sys as sysdba
Enter password: [SYS account password]
```

```
SELECT CURRENT_TIMESTAMP FROM DUAL;

CONNECT SYSTEM/system@localhost:1521/XEPDB1
SELECT USER FROM DUAL;
```

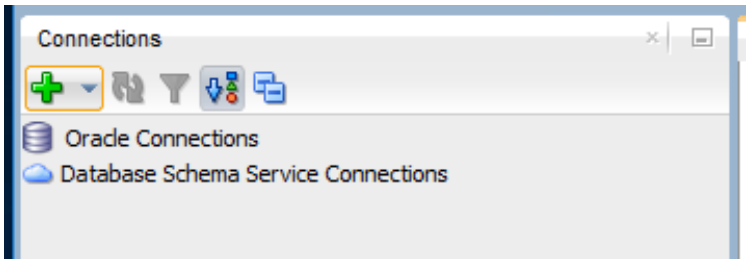


SQLcl: query execution and the CONNECT command.

Step 3. Read the connection string as follows: `SYSTEM` is the user account, `localhost` is the host, `1521` is the network port, and `XEPDB1` identifies the service. The password is supplied separately.

Example 3. SQL Developer: SYS with SYSDBA

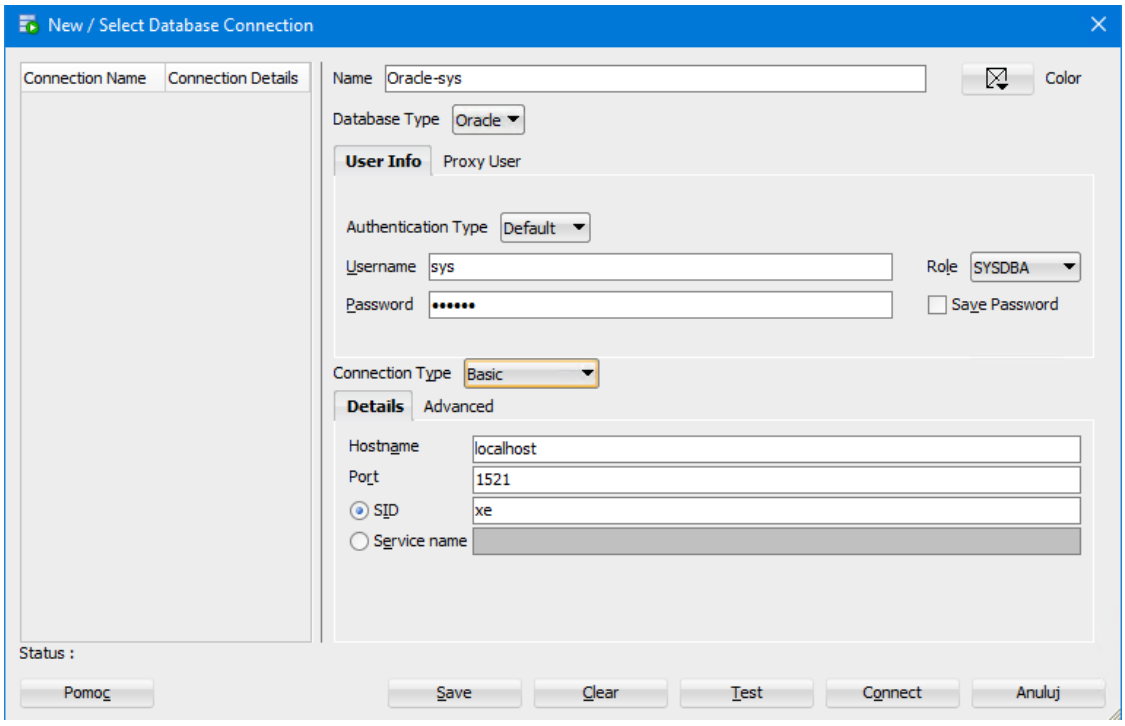
Step 1. Start SQL Developer and open the **Connections** panel. Click the green plus sign to create a new database connection.



SQL Developer: Connections panel and the new-connection button.

Step 2. Complete the new-connection form as follows. The host and port refer to an Oracle XE installation on the local Windows computer.

Field	Value
Connection Name	`Oracle-SYS` (a freely chosen saved connection name)
Username	`SYS`
Password	The password for SYS
Role	`SYSDBA`
Connection Type	`Basic`
Hostname	`localhost` (or the Oracle server IP address)
Port	`1521`
Service name	`XE` (select *Service name* if you use PDB containers like XEPDB1)



SQL Developer: example of a SYS connection form with SYSDBA selected.

Step 3. Click **Test**. A valid connection normally displays `Status: Success`. Click **Connect** and then run:

```
SELECT USER FROM DUAL;
SELECT SYS_CONTEXT('USERENV','CON_NAME') FROM DUAL;
```

Step 4. SQL Developer can execute single SQL statements with **Run Statement (Ctrl+Enter)**. For SQL*Plus-style commands such as `SHOW PARAMETER` or `SHOW SGA`, use **Run Script (F5)**.

Administrative and standard database connections
Oracle Database provides two built-in accounts used in this laboratory:

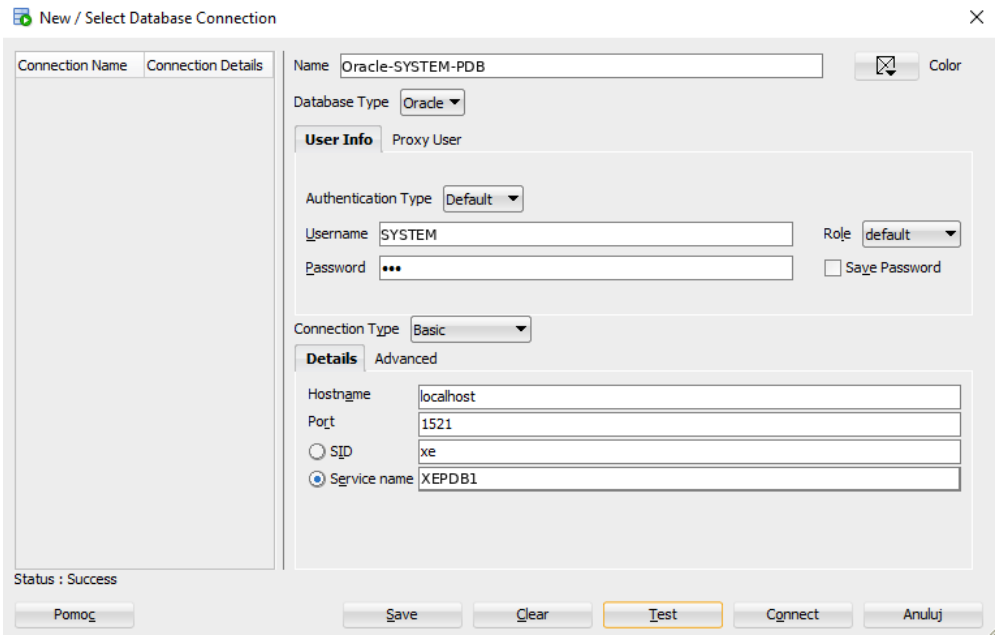
- **SYS** – the primary administrative account. In this laboratory, connections to SYS must always use the `SYSDBA` administrative privilege. In SQL Developer, select `Role: SYSDBA`.
- **SYSTEM** – a built-in database administration account used here without SYSDBA privileges. In SQL Developer, select `Role: Default`. Unlike SYS, SYSTEM does not require a SYSDBA connection. Don't use role SYSDBA in connect to this and other than SYS users.

Important: The SYSTEM account has extensive administrative privileges and is not an ordinary application user. However, it can connect using standard database authentication without `AS SYSDBA`.

Example 4. SQL Developer: SYSTEM with Default role

Step 1. Click the plus sign again in **Connections**. Create a second saved connection, this time for SYSTEM. Leave the role set to `Default` rather than requesting SYSDBA privileges.

Field	Value
Connection Name	`Oracle-SYSTEM-PDB`
Username	`SYSTEM`
Password	The password for SYSTEM
Role	`Default`
Hostname	`localhost`
Port	`1521`
Service name	`XEPDB1`



SQL Developer: ordinary connection using the Default role, with SYSTEM and XEPDB1.

Step 2. Click **Test**, confirm that the connection works, and select **Connect**. Verify the actual login user and current container:

```
SELECT USER FROM DUAL;
SELECT SYS_CONTEXT('USERENV','CON_NAME') FROM DUAL;
```

Step 3. Compare the SYS and SYSTEM connections. Identify differences in the role and service settings. Use the returned container name to check where the SQL statement is running.

*Example 7. SQL*Plus: Standard user login*

SQL*Plus supports both initial authentication when the application starts and reconnecting after a database session has already been established.

Step 1. Initial login

Start SQL*Plus. When the application requests a username, enter the account name and connection information.

```
Enter user-name: SYSTEM
Enter password: [SYSTEM account password]
```

After successful authentication, the SQL prompt appears:

```
SQL>
```

> **Initial login:** When SQL*Plus asks for a username during startup, do not use the `CONNECT` command. Enter the username directly at the authentication prompt.

Step 2. Verify the connection

Execute the following statements:

```
SELECT USER FROM DUAL;

SELECT SYS_CONTEXT('USERENV','CON_NAME')
FROM DUAL;
```

The first statement identifies the connected user. The second identifies the current database container.

Step 3. Reconnecting during an active session

To reconnect using the existing SQL*Plus console, execute the `CONNECT` command:

```
SQL> CONNECT SYSTEM
Enter password: [SYSTEM account password]
Connected.
```

The same mechanism can be used to establish an administrative connection:

```
SQL> CONNECT SYS AS SYSDBA
Enter password: [SYS account password]
Connected.
```

> **Reconnecting:** When the SQL*Plus console is already running and displays the `SQL>` prompt, use `CONNECT` to establish a new database connection.
> **Remember:**
> - Initial login – username without CONNECT.
> - Reconnecting – CONNECT followed by the username.
> - SYS – use AS SYSDBA.
> - SYSTEM – connect without AS SYSDBA.

Example 6. SQLcl: Standard user login

SQLcl follows connection rules similar to SQL*Plus.

Step 1. Initial login

Start SQLcl using the `sql.exe` executable.

When SQLcl displays the username prompt, enter the standard database user:

```
Username? SYSTEM/system@localhost:1521/XEPDB1
```

After successful authentication, SQLcl displays the SQL prompt.

> **Initial login in SQLcl:** Do not enter the `CONNECT` command when SQLcl requests the username during startup. Only provide the account and connection details.

Step 2. Execute a test query

```
SELECT CURRENT_TIMESTAMP FROM DUAL;

SELECT USER FROM DUAL;
```

Step 3. Reconnecting in SQLcl

When the SQLcl console is already open, use the `CONNECT` command:

```
SQL> CONNECT SYSTEM/system@localhost:1521/XEPDB1
```

To switch to the administrative SYS account:

```
SQL> CONNECT SYS/system@localhost:1521/XE AS SYSDBA
```

> **Connection rules for SQLcl:**
>
> - At the initial username prompt, enter the username without CONNECT.
> - Inside an active SQLcl console, use CONNECT to establish another connection.
> - Use SYSDBA when connecting as SYS.
> - Use a standard connection for SYSTEM.
>
> The connection service determines the target container: `XE` normally identifies the root container, while `XEPDB1` identifies the pluggable database.

Example 7. Diagnosing a connection failure

If SQL Developer displays `The Network Adapter could not establish the connection`, the client could not establish a connection to the supplied network endpoint. This error alone does not identify the exact cause.



SQL Developer: example of a connection establishment error.

Check `Hostname`, `Port`, and `Service name`, and confirm that the required Oracle Windows services are running. The detailed administration of the listener belongs to a separate laboratory.

Part I. Inspecting Oracle DBMS parameters

The following tasks reproduce the supplied laboratory sequence. Execute the provided SQL and observe the actual values returned by your Oracle instance. For SQL Developer, use Run Statement when running SHOW commands.

1. Start SQL*Plus from the Windows Start menu and connect to the Oracle Database instance using the SYSTEM account.

```
CONNECT SYSTEM
```

   Hint: Enter the SYSTEM password only when the client requests it.

2. Examine SQLcl as an alternative to SQL*Plus. Determine whether this command-line client is available on your computer.

   Hint: If SQLcl is already installed, there is no need to download it again.

3. Locate the directory containing SQLcl, open its bin subdirectory, and start the sql.exe executable.

```
sql.exe
```

   Hint: The executable may also be available from a terminal or a desktop shortcut.

4. In SQLcl, connect as SYS using the correct host, port, service, and SYSDBA administrative privilege. Read the actual computer name or IP address from your workstation.

```
CONNECT SYS@//localhost:1521/XE AS SYSDBA
```

   Hint: The client asks for the password separately. Use the XE service for the administrative SYS connection.

5. Open a graphical Oracle client, for example SQL Developer. Create a connection and locate the SQL worksheet used to execute statements.

   Hint: In SQL Developer, use Run Script (F5) for SHOW commands.

6. Open a new SQL*Plus, SQLcl, or SQL Developer session. Connect as SYS with SYSDBA privileges and verify the connection status.

```
CONNECT SYS/system@localhost:1521/XE AS SYSDBA
```

   Hint: SYSDBA is the administrative connection privilege used here with SYS.

7. Determine the database name using the two methods shown below. Compare the output format of the two commands.

```
SHOW PARAMETER db_name;

SELECT name FROM V$DATABASE;
```

   Live check: Point to the database name in your result.

8. Identify the Oracle instance name and the account currently used by your SQL session.

```
SELECT instance FROM V$THREAD;
SELECT USER FROM DUAL;
```

Live check: Point to the instance name and current user in your result.

9. Identify the physical data files. Find their Windows paths and inspect file size, block size, and status in the returned views.

```
SELECT * FROM DBA_DATA_FILES;
SELECT * FROM V$DATAFILE;
```

Live check: Point to the one actual data-file path in your result.

10. Locate the temporary files. Inspect their paths, file sizes, and statuses. Compare the information returned by the two views.

```
SELECT * FROM DBA_TEMP_FILES;
SELECT * FROM V$TEMPFILE;
```

11. Find the control file locations in the Windows file system.

```
SELECT * FROM V$CONTROLFILE;
```

12. Find the online redo log member files on the Windows machine. Identify the redo log group to which each file belongs.

```
SELECT * FROM V$LOGFILE;
SELECT GROUP#, TYPE, MEMBER FROM V$LOGFILE;
```

13. Check the size and current status of each online redo log group.

```
SELECT GROUP#, BYTES/1024/1024 AS MB, STATUS FROM V$LOG;
```

Live check: Point to the redo log group number and its STATUS in your result.

14. Find the diagnostic directory used for background-process trace files. Check which of the two methods returns the relevant directory in Oracle 21c.

```
SHOW PARAMETER background_dump_dest;

SELECT value FROM V$DIAG_INFO WHERE name = 'Diag Trace';
```

Hint: In Oracle 21c, V$DIAG_INFO can identify the diagnostic trace directory.

15. Explore the data dictionary views available to your account. Identify five recognizable view names and their purposes.

```
SELECT * FROM DICTIONARY;
SELECT * FROM DICT_COLUMNS;
```

16. Display details of the currently running Oracle instance. Identify at least three meaningful attributes of the instance.

```
SELECT * FROM V$INSTANCE;
```

Live check: Point to the INSTANCE_NAME and STATUS in your result.

17. Examine the initialization parameters currently in effect for your session. Compare the two views and locate sessions, processes, spfile, and two other parameters.

```
SELECT * FROM V$PARAMETER;
SELECT * FROM V$PARAMETER2;
```

Live check: Try to point to the value of the processes parameter in your result.

18. Find the startup parameter file (SPFILE). Determine how many entries in V$SPPARAMETER have a non-NULL value.

```
SHOW PARAMETER spfile

SELECT COUNT(*) FROM V$SPPARAMETER
WHERE value IS NOT NULL;
```

Hint: SPFILE is a binary Oracle-managed file. ***Do not modify*** *it with a text editor.*

19. Inspect the initialization parameter values stored in the SPFILE. Compare this view with V$PARAMETER and look for one or two differing entries.

```
SELECT * FROM V$SPPARAMETER;
```

20. Display the initialization parameters in effect for the instance as a whole. Compare the information with V*PARAMETERandV*SPPARAMETER.

```
SELECT * FROM V$SYSTEM_PARAMETER;
```

21. Inspect users in the current container. Determine how many accounts have the exact statuses OPEN and LOCKED. Identify one account with a status other than OPEN, if present.

```
SELECT * FROM DBA_USERS;

SELECT ACCOUNT_STATUS, COUNT(*) AS Digit FROM DBA_USERS
WHERE ACCOUNT_STATUS IN ('OPEN', 'LOCKED')
GROUP BY ACCOUNT_STATUS;
```

Hint: Account statuses can include compound labels, such as EXPIRED & LOCKED, which are distinct from the exact label LOCKED.

Live check: Point to the exact ACCOUNT_STATUS of one account in your result.

22. Inspect free space in the tablespaces, broken down by extents. Count how many rows refer to the USERS tablespace.

```
SELECT * FROM DBA_FREE_SPACE;

SELECT COUNT(*) FROM DBA_FREE_SPACE
WHERE TABLESPACE_NAME = 'USERS';
```

23. Examine the System Global Area (SGA). Determine whether the view and the client command present identical information.

```
SELECT * FROM V$SGA;
SHOW SGA
```

Live check: Point to the total SGA in your result.

24. Display detailed SGA memory information. Determine which view shows component-level summaries and which contains more granular allocation records.

```
SELECT * FROM V$SGAINFO;
SELECT * FROM V$SGASTAT;
```

25. Use the parameter views to estimate sorting and hashing memory for a specified number of sessions, allowing an additional 2 MB per session. Express the result in MB.

```
SHOW PARAMETER area_size

SELECT &&hwm * (2097152 + p1.value + p2.value)
       AS pga_size,
       &&hwm * (2097152 + p1.value + p2.value)
       / (1024*1024) AS pga_mb
FROM V$PARAMETER p1, V$PARAMETER p2
WHERE p1.name = 'sort_area_size'
  AND p2.name = 'hash_area_size';
```

*Hint: The parameter hwm is a session count entered when SQLcl or SQL*Plus requests a value. This formula is only a simplified estimate, not the measured total PGA usage. Use 1024 in both windows pop-ups*

26. Estimate the Database Buffer Cache hit ratio using the listed instance statistics. Read the calculated value and its unit.

```
SELECT (1 - pr.value / NULLIF(dbg.value + cg.value,0)) * 100 AS "Cache Hit Ratio [%]"
FROM V$SYSSTAT pr, V$SYSSTAT dbg, V$SYSSTAT cg
WHERE pr.name = 'physical reads'
  AND dbg.name = 'db block gets'
  AND cg.name = 'consistent gets';
```

Hint: This is a simplified ratio derived from SGA buffer-cache statistics, not a PGA metric.

27. Determine the size of the redo log buffer. Convert its size to megabytes.

```
SHOW PARAMETER log_buffer;

SELECT name, value FROM V$PARAMETER WHERE name = 'log_buffer';
```

Live check: Point to the LOG_BUFFER value in your result.

28. Display instance statistics whose names contain the word buffer. Identify two counters and read their values.

```
SELECT name, value FROM V$SYSSTAT WHERE name LIKE '%buffer%';
```

End-of-class verification: live results

No report is submitted. At the end of the session, keep your Oracle client open. The instructor will select a few of the following checks and ask you to execute a query and point to the result on screen. The values must come from your own current Oracle connection, not from a screenshot or a sample answer.

Ref.	Where to look	Value to show on your screen
A	I–7, I–8	Database NAME, instance name, and current USER
B	I–9, I–10	One file NAME and its Windows directory
C	I–13	One redo GROUP# together with STATUS and size (MB)
D	I–16	INSTANCE_NAME, STATUS and STARTUP_TIME
E	I–17	Actual processes and sessions parameter values
F	I–21	Username and full ACCOUNT_STATUS of one non-OPEN account
G	I–23, I–27	SGA total and redo LOG_BUFFER size

How the check works. First show the current account and container using the short query below. Then run the query associated with the check selected by the instructor. Indicate both the column name and the returned value. The instructor may ask you to rerun a query or switch to `XEPDB1` to confirm the result.

```
SELECT USER AS USERNAME, SYS_CONTEXT('USERENV','CON_NAME') AS CONTAINER FROM DUAL;
```