Chapter 4: SELECT Statement Clauses and Row Limiting

Overview of SELECT Statement Clauses

This chapter covers the following key clauses:

- **WHERE**: Filters rows based on conditions
- **ORDER BY**: Sorts result rows
- **Ampersand (&) substitution**: Prompts user input in SQL*Plus
- **SQL Row Limiting Clause**: Restricts number of returned rows

> 📝 **Note**: Ampersand Substitution is tested on the exam. It's specific to SQL*Plus, not the Oracle SQL Language Reference.

ORDER BY Clause

Purpose

Sorts result rows by one or more columns or expressions.

Syntax

```sql
SELECT column1, column2 FROM table
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC];
```

Key Points

- **Always the last clause** in a SELECT statement
- Cannot modify table data, only display order
- Not usable in INSERT, UPDATE, or DELETE (except in subqueries)
- Without ORDER BY, output order is unpredictable

Sort Order: ASC vs DESC

- **ASC** (default): Lowest to highest
- **DESC**: Highest to lowest
- Applied **individually** to each column

Example:

```sql
ORDER BY ship_id ASC, project_cost DESC;
```

Expressions in ORDER BY

You can use expressions like:

```sql
ORDER BY project_cost / days;
```

Column Aliases

Usage Example:

```sql
SELECT project_cost / days AS per_day_cost
FROM projects
ORDER BY per_day_cost;
```

Alias Rules:

- Defined in SELECT list using AS
- Can contain spaces/special chars (if quoted)
- Can only be used in ORDER BY (not in WHERE, GROUP BY, etc.)
- Must be unique and scoped to the SQL statement

ORDER BY Reference Techniques

1. By Name:

```sql
ORDER BY state, city;
```

2. By Alias:

```sql
SELECT ..., project_cost/days AS "Cost Per Day"
ORDER BY "Cost Per Day";
```

3. By Position:

```sql
SELECT col1, col2, col3
ORDER BY 3;
```

- Position must refer to the **SELECT list**
- Invalid if referencing non-existent position

Combining Methods

ORDER BY can mix:

- Column names
- Aliases
- Positions

Example:

```sql
ORDER BY ship_id DESC, "The Project", 2;
```

NULLs in ORDER BY

- NULL is treated as greater than any value
- Always appears **last** when sorted ascending
- Applies to numbers, text, and dates

WHERE Clause

Core Concept

Used in SELECT, UPDATE, and DELETE to filter rows based on conditions.

Syntax Pattern

```sql
SELECT column_list
FROM table_name
WHERE condition;
```

How WHERE Works

- Filters rows by evaluating each row's condition to TRUE or FALSE
- Only rows that evaluate to TRUE are included
- Must follow the FROM clause
- Can evaluate:
  - Columns vs. literal values
  - Columns vs. expressions
  - Expressions vs. expressions

Comparison Operators

Operator	Description
=	Equal
>	Greater than
>=	Greater or equal
<	Less than
<=	Less or equal
!= or <>	Not equal

Data Type Comparison Rules

- **Numeric:** Smaller values < larger ones
- **Character:** Case-sensitive. 'A' < 'Z' < 'a'
- **Date:** Earlier date is considered smaller

> Oracle **may** auto-convert mismatched data types, but don't rely on it.

Special Comparison Conditions

LIKE

Used with character data and wildcards:

- `_` = single character
- `%` = zero or more characters

Examples:

```sql
WHERE PORT_NAME LIKE 'San____'; -- 4 underscores
WHERE PORT_NAME LIKE 'San%'; -- any length after 'San'
```

> Put the pattern **after** LIKE. Don't reverse it.

IN

Shorthand for multiple OR conditions:

```sql
WHERE COUNTRY IN ('UK', 'USA', 'Bahamas');
```

Rules:

- Works with text, numbers, dates
- List must be in parentheses
- All values should be of the same or compatible data type

```sql
WHERE COUNTRY NOT IN ('UK', 'USA'); -- Inverts the match
```

BETWEEN

Inclusive range comparison:

```sql
WHERE CAPACITY BETWEEN 3 AND 4;
-- Same as:
WHERE CAPACITY >= 3 AND CAPACITY <= 4;
```

> NOT BETWEEN is valid and means "outside of range".

IS NULL / IS NOT NULL

- Use IS NULL to check for missing (unknown) values
- `= NULL` **will never work**

```sql
WHERE CAPACITY IS NULL;
WHERE CAPACITY IS NOT NULL;
```

Boolean Logic in WHERE Clauses

AND, OR, NOT

- **AND**: All conditions must be true
- **OR**: At least one must be true
- **NOT**: Reverses result (true ⇄ false)

Expression	Result
TRUE AND TRUE	TRUE
TRUE AND FALSE	FALSE
FALSE OR TRUE	TRUE
NOT TRUE	FALSE

Parentheses for Grouping

Use them to override operator precedence:

```sql
WHERE (STYLE = 'Suite' OR STYLE = 'Stateroom')
  AND WINDOW = 'Ocean';
```

Operator Precedence

1. NOT
2. AND
3. OR

> Always use parentheses when mixing AND/OR to avoid logic errors.

Common Pitfalls to Avoid

- **Never** use `= NULL` (always use IS NULL)

- Don't assume LIKE works with pattern on the left
- Avoid mixing data types without proper conversion
- Don't forget Boolean logic rules — order and grouping matter

What Comes Next (Covered Later)

- Subqueries in WHERE
- Set operators

Ampersand Substitution Variable

Core Concepts

- **Feature of SQL*Plus**, not SQL language itself
- Used to **pass runtime input** into a SQL script (parameterization)
- Allows flexibility in WHERE clauses, SELECT columns, table names, and more

Syntax Basics

```sql
SELECT ROOM_NUMBER, STYLE, WINDOW
FROM SHIP_CABINS
WHERE ROOM_NUMBER = &RNO;
```

- When run, prompts user: `Enter value for rno:`
- Input replaces &RNO dynamically during execution

Key Behaviors

- **Text values**: wrap substitution in quotes

  ```sql
  WHERE WINDOW = '&Window_Type'
  ```

- Alternatively, user can enter the quotes at runtime if not coded in

DEFINE / UNDEFINE

- Predefine variables:

```sql
DEFINE vWindows = Ocean
```

- Use in script:

```sql
WHERE WINDOW = '&vWindows'
```

- Remove definition:

```sql
UNDEFINE vWindows
```

SET / SHOW Commands

- Control SQL*Plus system variables:
  - `SET DEFINE ON` → Enables substitution
  - `SET DEFINE OFF` → Disables substitution
  - `SHOW DEFINE` → Shows current prefix
  - Prefix can be changed:

```sql
SET DEFINE *
```

    Now `*vVar` is used instead of `&vVar`

- `SET VERIFY OFF` hides "old/new" line display during substitution

ACCEPT and PROMPT

- Interactive scripts:

```sql
PROMPT Enter room number:
ACCEPT vRoomNumber PROMPT 'Enter a room number: '
```

- Accepts user input and stores it in vRoomNumber for use in SQL

Batch Scripts

- Save .sql file (e.g., Run.sql)
- Execute in SQL*Plus with:

```
sql

@Run
```

- Good practice: include `SET DEFINE ON` at the start of batch files

Other Notes

- Substitution variable names are alphanumeric and space-delimited
- Prefix character (&) is customizable but **must be a single character**
- Defined variable scope lasts for the **session**, unless undefined

---

SQL Row Limiting Clause (Oracle 12c+)

Used to limit rows returned from a query. It follows the ORDER BY clause (if present).

FETCH Clause

Basic Syntax:

```
sql

SELECT * FROM table
FETCH FIRST n ROWS ONLY;
```

Key Keywords (in order):

1. **FETCH** — required
2. **FIRST** or **NEXT** — either, no functional difference
3. **<number>** — optional; if omitted, defaults to 1
4. **PERCENT** — optional; rounds up to nearest full row
5. **ROW** or **ROWS** — either, same effect
6. **ONLY** — return exactly the specified number
7. **WITH TIES** — includes additional rows with same value at the cutoff point (only works with ORDER BY)

Examples:

```sql
FETCH FIRST 10 ROWS ONLY;
FETCH FIRST 50 PERCENT ROWS ONLY;
FETCH FIRST 10 ROWS WITH TIES;
```

WITH TIES

- Includes rows that are tied (equal in sort order) at the cutoff
- **Only works if** ORDER BY is used
- Without ORDER BY, WITH TIES = ONLY

OFFSET Clause

Used to skip rows before applying FETCH.

Syntax:

```sql
SELECT * FROM table
OFFSET n ROWS FETCH FIRST m ROWS ONLY;
```

Details:

- Skips n rows, then fetches m
- OFFSET must be a number (not a percentage)
- Negative OFFSET → treated as OFFSET 0

Examples:

```sql
OFFSET 5 ROWS FETCH FIRST 2 ROWS ONLY;
```

OFFSET Behavior Table

OFFSET Value	Behavior
Omitted or < 0	Treated as 0
= 0	Start at first row
> available rows	No rows returned

ORDER BY + FETCH Tips

- ORDER BY determines row order before FETCH applies
- ASC is default; use DESC for reverse
- Can order by:
  - Column names
  - Aliases
  - SELECT list positions (e.g., ORDER BY 2)
- ORDER BY always comes last in SELECT

WHERE Clause Integration

- Filters rows before FETCH applies
- Follows FROM clause
- Operators include =, !=, <, >, LIKE, IN, IS NULL
- Combines with AND, OR, NOT

Null Handling in WHERE

- **Never use:** `= NULL`
- **Use:** `IS NULL` or `IS NOT NULL`