Oracle SQL Notes & Advanced Concepts

Compiled Notes

Contents

1 Introduction to SELECT

Think of SELECT as a function that goes through each row, performs the things/calculations instructed, and then prints/outputs the values.

2 String Functions

2.1 SUBSTR

Uses 1-based indexing. Basic Description: Extracts a portion of a string.

Syntax

SUBSTR(string, start_position, [length])

Standard Usage:

```
-- Start at position 2, take 3 characters
SELECT SUBSTR('Oracle', 2, 3) FROM DUAL;
-- Result: 'rac'
```

Advanced / Hidden Usage

Negative Start Position: If the start position is negative, Oracle counts backward from the end of the string.

```
-- Start 2 characters from the end, take all remaining
SELECT SUBSTR('Oracle', -2) FROM DUAL;
-- Result: 'le'
```

To get the last character of a string, put `-1` there.
Omitting Length: If you leave out the length argument, it extracts everything from the start position to the very end.

```
SELECT SUBSTR('Oracle', 3) FROM DUAL;
-- Result: 'acle'
```

2.2 LENGTH

Basic Description: Returns the number of characters.

Syntax

LENGTH(string)

Standard Usage:

```
SELECT LENGTH('Database') FROM DUAL;
-- Result: 8
```

Trap / Warning

Fixed-Width Char Traps: If you have a column defined as `CHAR(10)` and you store the word "Cat", Oracle pads it with spaces to fill 10 characters. `LENGTH` will return 10, not 3.
NULL Returns NULL: `LENGTH(NULL)` returns `NULL`, not 0. This is a common bug source

in PL/SQL logic.

2.3 INSTR

Basic Description: Finds the numeric position of a substring.

Syntax

INSTR(string, substring, [start_position], [nth_appearance])

Standard Usage:

```
SELECT INSTR('Banana', 'a') FROM DUAL;
-- Result: 2 (Finds the first 'a')
```

Advanced / Hidden Usage

Finding the Nth Occurrence: You can specify which match you want. This is incredibly useful for parsing data.

```
-- Find the 2nd occurrence of 'a', starting search from position 1
SELECT INSTR('Banana', 'a', 1, 2) FROM DUAL;
-- Result: 4
```

Negative Start Position (Backwards Search): If you provide a negative start position, Oracle jumps to the end of the string and searches backward for the character.

```
-- Find the first 'a', but scanning backwards from the end
SELECT INSTR('Banana', 'a', -1) FROM DUAL;
-- Result: 6 (The last 'a')
```

Check Existence: It returns 0 if the string is not found, which is often used in boolean logic (e.g., `WHERE INSTR(col, 'xyz') > 0`).

2.4 LPAD / RPAD

Basic Description: Pads the left or right side of a string to reach a specific total length.

Syntax

LPAD(string, total_length, [pad_string])

Standard Usage:

```
SELECT LPAD('123', 5, '0') FROM DUAL;
-- Result: '00123' (Great for formatting IDs)
```

Trap / Warning

Truncation Behavior: This is the most common mistake. If the total length you request is smaller than the original string, Oracle cuts off (truncates) the string to fit that length.

```
SELECT RPAD('Hello World', 5, '*') FROM DUAL;
-- Result: 'Hello' (It did not pad; it deleted!)
```

Advanced / Hidden Usage

Visual Charts: You can use RPAD to create simple bar charts in the console.

```
-- Assume salary is 5000. Div by 1000 = 5.
SELECT RPAD('#', 5, '#') FROM DUAL;
-- Result: '#####'
```

2.5 TRIM

Basic Description: Removes whitespace from the ends of a string.

Syntax

TRIM([[LEADING | TRAILING | BOTH] trim_character FROM] string)

Standard Usage:

```
SELECT TRIM(' Hello ') FROM DUAL;
-- Result: 'Hello'
```

Advanced / Hidden Usage

Removing Specific Characters: You aren't limited to spaces. You can trim other characters, but only one specific character at a time.

```
-- Remove '0' from the start (Leading)
SELECT TRIM(LEADING '0' FROM '000123') FROM DUAL;
-- Result: '123'
```

LTRIM/RTRIM Difference: The standard `TRIM` can only remove a literal single character. If you want to remove a set of characters (like "remove all x, y, and z's from the left"), you must use `LTRIM` or `RTRIM`.

```
-- TRIM cannot do this, but LTRIM can:
SELECT LTRIM('<>Hello', '<>') FROM DUAL;
-- Result: 'Hello'
```

2.6 REPLACE

Basic Description: Replaces all occurrences of a text with another text.

Syntax

REPLACE(string, search_string, [replacement_string])

Standard Usage:

```
SELECT REPLACE('Jack and Jue', 'J', 'B') FROM DUAL;
-- Result: 'Back and Bue'
```

Advanced / Hidden Usage

Deleting Text: If you omit the 3rd argument (the replacement string), Oracle defaults it to `NULL`, effectively deleting the search string.

```
SELECT REPLACE('1-2-3-4', '-') FROM DUAL;
```

```
-- Result: '1234'
```

Nested Replace is often used to clean dirty data.

3 Aggregate Functions, GROUP BY, and HAVING

These functions take many values as input and return one single value.

Function	Description	Null Behavior	Data Types	Notes
COUNT(col)	Counts non-null rows	Ignores NULLs	Any	
COUNT(*)	Counts all rows	Counts NULLs	Any	
SUM(col)	Adds values up	Ignores NULLs	Number	
AVG(col)	Averages values	Ignores NULLs	Number	
MAX(col)	Finds highest value	Ignores NULLs	Num, Date, Str	
MIN(col)	Finds lowest value	Ignores NULLs	Num, Date, Str	*Pro Tip below

Pro Tip: MIN and MAX work on Dates! `MIN(hire_date)` gives you the most senior employee (earliest date).

3.1 The "Golden Rule" of GROUP BY

This is the #1 rule that causes errors for beginners. Memorize this:

> **If a column is in the SELECT list, it MUST be in the GROUP BY clause... UNLESS it is inside an Aggregate Function.**

Trap / Warning

The Logic Error: "You can only sort by what you see." Once you use GROUP BY, the original rows are gone. They have been squashed into "piles." Therefore, your ORDER BY clause can only reference:

1. The columns you used to make the groups (GROUP BY columns).
2. The result of a calculation (Aggregate functions).

You CANNOT order by a raw column that is buried inside the pile.

Example Failure:

```
SELECT department_id, first_name, AVG(salary)
FROM employees
GROUP BY department_id;
-- ERROR: Not a GROUP BY expression
```

Why it fails: You grouped by Department. You have one "pile" for Department 90. That pile has an average salary (one number). But that pile has three employees (Steven, Neena, Lex). The database doesn't know which name to display next to the single average number.

3.2 The Mental Model: "Sub-Groups"

When you GROUP BY two columns, you are NOT making two separate sets of piles. Instead, you are asking: **"Make a pile for every UNIQUE COMBINATION of Department and Job."**

Imagine sorting the EMPLOYEES table physically:

1. **Level 1 (Department):** You toss all employees into buckets based on Dept ID (50, 90).
2. **Level 2 (Job):** You look inside the "Dept 50" bucket. You notice there are Managers (ST_MAN) and Clerks (ST_CLERK).
3. **The Action:** You separate them into smaller boxes inside that bucket.

So, your final result isn't just "How many people are in Dept 50?" It is: "How many Clerks are in Dept 50?" vs "How many Managers are in Dept 50?"

Unique Combination Visualization:

- Row 1: Dept 50, Job ST_CLERK → Group #1
- Row 2: Dept 50, Job ST_CLERK → Matches Group #1
- Row 3: Dept 50, Job ST_MAN → Does NOT match Group #1. Create Group #2.

4 ORDER BY: Advanced Tips & Tricks

Advanced / Hidden Usage

Tip 1: You can Order by an Aggregate that isn't in the SELECT list This is a weird but valid feature. You don't have to show the number to sort by it.

```
-- "List departments, sorted by how many people are in them."
-- Notice: I am NOT selecting COUNT(*) in the top line.
SELECT department_id
FROM employees
GROUP BY department_id
ORDER BY COUNT(*) DESC;
```

Advanced / Hidden Usage

Tip 2: Use Column Aliases (The Best Practice) In Oracle, `ORDER BY` is the last thing to execute. This means it knows the "nicknames" (Aliases) you created in the `SELECT` clause.

```
SELECT department_id, AVG(salary) AS avg_sal
FROM employees
GROUP BY department_id
ORDER BY avg_sal DESC;
```

Note: You generally cannot use aliases in the `GROUP BY` clause, but you can use them in `ORDER BY`.

5 DISTINCT

5.1 What does it do?

`DISTINCT` looks at the final result set of your query and removes any rows that are identical.

5.2 How to use it

Scenario A: Single Column (Most Common)

```
SELECT DISTINCT job_id FROM employees;
```

Without DISTINCT: 107 rows. With DISTINCT: 19 rows (unique job codes).

Scenario B: Multiple Columns (The "Combination" Rule) This is where students get confused. `DISTINCT` applies to the entire row, not just the first column.

```
SELECT DISTINCT department_id, manager_id FROM employees;
```

Logic: It glues `department_id` and `manager_id` together and looks for unique pairs. (Dept 50, Manager 100) is distinct from (Dept 50, Manager 120). It will keep both.

Trap / Warning

Rule #1: You cannot selectively "Distinct" one column WRONG:

```
SELECT first_name, DISTINCT job_id FROM employees;
-- Impossible in one SELECT.
```

Advanced / Hidden Usage

Rule #3: DISTINCT Inside Aggregates (The "Count Unique" Trick) This is a superpower. Sometimes you don't want to see the list of unique items, you just want to know how many there are.

```
-- "How many different departments actually have people in them?"
SELECT COUNT(DISTINCT department_id) FROM employees;
-- Result: 11
```

Compare to `COUNT(department_id)`, which returns 106.

Rule #2: NULLs are treated as "One Value". `SELECT DISTINCT commission_pct` will return one single row containing NULL. It treats NULL as a distinct category.

Rule #4: DISTINCT vs. GROUP BY Use `DISTINCT` when you want to see unique values (display). Use `GROUP BY` when you want to calculate something (SUM, AVG) for each unique value.

6 Single Quotes vs Double Quotes

You are correct that Single Quotes are for data and Double Quotes are for aliases/identifiers. However, "Case Sensitivity" works differently for data vs. identifiers.

6.1 Single Quotes (Data Values) = Strict Case Sensitivity

When you use single quotes, you are telling Oracle: "This is a specific piece of text data." Oracle compares bytes exactly. **The Rule:** `'Apple'` $\neq$ `'apple'`.

```
SELECT * FROM users WHERE name = 'smith'; -- Returns 0 rows (No Match)
SELECT * FROM users WHERE name = 'Smith'; -- Returns 1 row (Match)
```

6.2 Double Quotes (Identifiers) = The "Auto-Uppercase" Trap

A. Without Quotes (The Default) $\rightarrow$ Case INSENSITIVE If you don't use double quotes, Oracle automatically converts everything you type into UPPERCASE before running it. Type: `CREATE TABLE my_table` $\rightarrow$ Oracle sees: `CREATE TABLE MY_TABLE`. Result: Table stored as `MY_TABLE`.

B. With Double Quotes $\rightarrow$ Strict Case Preservation If you use double quotes, you force Oracle to skip the auto-uppercase step. Type: `SELECT * FROM "my_table";` Oracle sees: `SELECT`

`* FROM my_table` (lowercase). Result: ERROR (Table does not exist) because it is looking for lowercase `my_table`, but it is stored as `MY_TABLE`.

7 SET Operations

UNION, UNION ALL, INTERSECT, MINUS

7.1 Structural Constraints

1. **Column Count Parity:** Both queries must select the exact same number of columns.
2. **Data Type Compatibility:** Corresponding columns must be of the same data type group (Number vs Number, Char vs Varchar).

7.2 Result Set Behavior

- **Header Naming:** Column names in the final result come from the **first** SELECT statement.
- **Duplicate Handling:** UNION, INTERSECT, MINUS eliminate duplicates. **UNION ALL** preserves duplicates (and is faster).
- **NULL Handling:** Set operators treat NULLs as identical (unlike standard SQL comparisons).

7.3 Sorting and Ordering

- **Global Sort:** The `ORDER BY` clause can appear only once at the very end.
- **Reference Scope:** Must reference columns by names in the first query or by position number (e.g., `ORDER BY 3`).

Example: All-in-One Syntax

```
SELECT employee_id, job_id, salary FROM employees WHERE department_id = 90
UNION ALL
SELECT employee_id, job_id, 0 FROM retired_employees
MINUS
SELECT employee_id, job_id, salary FROM blacklisted_employees
ORDER BY 3 DESC; -- Sorts final result by 3rd col (Salary)
```

8 Practice Problems & Solutions

Problem 1: Vowel Count & Logic

Display Last Name and a calculated column `VOWEL_COUNT`. Filter to show only employees whose Last Name contains exactly 2 instances of 'a' and ends with a consonant.

```
SELECT LAST_NAME,
 (LENGTH(LAST_NAME) - LENGTH(TRANSLATE(LOWER(LAST_NAME), '@aeiou', '@'))) AS
    VOWEL_COUNT
FROM EMPLOYEES
WHERE
 -- Logic: Replace 'a' with empty string to reduce length by exactly 2?
 (LENGTH(LAST_NAME) - LENGTH(REPLACE(LOWER(LAST_NAME), 'a', ''))) = 2
```

```
 AND
 SUBSTR(LOWER(LAST_NAME), -1) NOT IN ('a', 'e', 'i', 'o', 'u')
ORDER BY VOWEL_COUNT DESC;
```

Concept - TRANSLATE vs REPLACE: TRANSLATE works on individual characters. REPLACE works on strings/blocks. We use `TRANSLATE(str, '@aeiou', '@')` to replace all vowels with '@' (except the dummy char) or remove them if they don't map. Note: You cannot put NULL as the 3rd parameter of TRANSLATE, unlike REPLACE.

Problem 2: Advanced Date Arithmetic

Filter employees: 1. Hired in a Leap Year. 2. Hired in 2nd half of month (>15th). 3. Worked > 8000 days.

```
SELECT EMPLOYEE_ID, HIRE_DATE, 'Q' || TO_CHAR(HIRE_DATE, 'Q') as hire_quarter
FROM EMPLOYEES
WHERE
 -- Leap Year Check: Year divisible by 4
 MOD(TO_NUMBER(TO_CHAR(HIRE_DATE, 'YYYY')), 4) = 0
 AND
 -- Day Check
 TO_NUMBER(TO_CHAR(HIRE_DATE, 'DD')) > 15
 AND
 -- Days Worked Check
 (SYSDATE - HIRE_DATE) > 8000
ORDER BY HIRE_DATE ASC;
```

Problem 3: Nth Occurrence of Character

Find employees where 'e' appears at least twice, and the 2nd occurrence is at index 5 or later.

```
SELECT FIRST_NAME, HIRE_DATE,
 INSTR(LOWER(FIRST_NAME), 'e', 1, 2) AS SECOND_E_POS
FROM EMPLOYEES
WHERE
 -- Find pos of 2nd 'e', start search at 1. If >= 5, it exists & is far enough.
 INSTR(LOWER(FIRST_NAME), 'e', 1, 2) >= 5
ORDER BY HIRE_DATE DESC;
```

Problem 4: Performance Score (NVL2)

Score = Salary + (10,000 extra if they have commission).

```
SELECT LAST_NAME, SALARY, COMMISSION_PCT,
 SALARY + NVL2(COMMISSION_PCT, 10000, 0) as PERFORMANCE_SCORE
FROM EMPLOYEES;
```

NVL2 Syntax: `NVL2(expr1, expr2, expr3)`. If expr1 is NOT NULL, return expr2. If NULL, return expr3.

Problem 5: Commission Charged Boolean

Count employees charged with commission (1) vs not (0).

```
SELECT NVL2(commission_pct, '1 (YES)', '0 (NO)') AS "IS_COMMISSION_CHARGED?",
 COUNT(*) AS EMPLOYEE_COUNT
```

```
FROM employees
GROUP BY NVL2(commission_pct, '1 (YES)', '0 (NO)');
```

Problem 6: Email Generation (.buet.ac.bd)

Create email: `firstname.lastname@jobid.buet.ac.bd`. No spaces, no underscores, all lowercase.

```
SELECT first_name, last_name, job_id,
 LOWER(REPLACE(REPLACE(
  (first_name || '.' || last_name || '@' || job_id || '.buet.ac.bd'),
  ' ', ''), '_', '')) as dept_mail
FROM hr.employees
WHERE instr(last_name,' ') <> 0 or instr(first_name,' ') <> 0;
```

Problem 7: Triennial Hires

Group hires into 3-year periods (e.g., 2001-2003).

```
SELECT
 ((TRUNC(TO_CHAR(HIRE_DATE, 'yyyy')/3)*3)) || '-' ||
 (((TRUNC(TO_CHAR(HIRE_DATE, 'yyyy')/3)+1)* 3)-1) AS TRIENNIAL_PERIOD,
 JOB_ID,
 COUNT(EMPLOYEE_ID) AS EMPLOYEES_HIRED
FROM HR.EMPLOYEES
GROUP BY TRUNC(TO_CHAR(HIRE_DATE, 'yyyy')/3), JOB_ID
ORDER BY TRUNC(TO_CHAR(HIRE_DATE, 'yyyy')/3) DESC;
```

Problem 8: Phone Number Parsing

Extract Country Code (before 1st dot) and Region Code (between 1st and 2nd dot).

```
SELECT PHONE_NUMBER,
 -- Country Code: 1 to index of first dot - 1
 SUBSTR(PHONE_NUMBER, 1, INSTR(PHONE_NUMBER, '.')-1) as country_code,
 -- Region Code: Start after first dot, length is diff between dot 2 and dot 1
 SUBSTR(PHONE_NUMBER,
        INSTR(PHONE_NUMBER, '.')+1,
        INSTR(PHONE_NUMBER, '.', 1, 2) - INSTR(PHONE_NUMBER, '.') - 1
 ) as REGION_CODE
FROM EMPLOYEES;
```