

DAY-2

SQL Indexing - In-Depth Explanation with Code Snippets

1. What is an Index in SQL?

An **index** is a database structure that improves the speed of data retrieval operations on a table. It acts like an index in a book, allowing the database to find rows quickly without scanning the entire table.

Why Use Indexes?

- **Speeds up SELECT queries** by reducing the number of rows scanned.
- **Optimizes WHERE, ORDER BY, GROUP BY operations.**
- **Prevents full table scans**, making queries more efficient.

2. Checking Existing Indexes

Before creating indexes, let's check if any exist in our table.

View all records in a table:

```
sql
CopyEdit
SELECT * FROM emp;
```

- ◆ This retrieves all rows from the emp table.

Check all indexes available in the database:

```
sql
CopyEdit
SELECT * FROM user_indexes;
```

- ◆ Returns details about indexes in the current database.

Check indexes for a specific table (e.g., emp table):

```
sql
CopyEdit
SELECT * FROM user_indexes WHERE table_name = 'EMP';
```

- ◆ Retrieves index details only for the emp table.

3. Creating Different Types of Indexes

a. Non-Unique Index


- ✓ **Purpose:** Improves search performance on frequently queried columns.
- ✓ **Allows duplicates** (i.e., multiple rows can have the same value).
- ✓ **Used for columns that appear frequently in WHERE conditions.**

```
sql
CopyEdit
CREATE INDEX emp_age_idx ON emp(age);
```

- ◆ This creates an **index on the "age" column** of the emp table.
- ◆ The database will use this index whenever a query searches for a specific age.

Example Query Before Indexing (Slow)

```
sql
CopyEdit
SELECT * FROM emp WHERE age = 30;
```

 **Without an index**, this query scans the entire table (full table scan).

Example Query After Indexing (Fast)

```
sql
CopyEdit
```

```
SELECT * FROM emp WHERE age = 30;
```

⚡ **With the index**, the query can find records quickly without scanning the entire table.

b. Unique Index

- ✓ **Purpose:** Ensures values in a column are **unique** (no duplicates allowed).
- ✓ **Similar to PRIMARY KEY** but can be applied to any column.
- ✓ **Useful for columns like employee ID, email, etc.**

sql

CopyEdit

```
CREATE UNIQUE INDEX emp_id_idx ON emp(empno);
```

- ◆ Creates a **unique index on the "empno" column**.
- ◆ Ensures that empno values are **unique** across all records.

Example: Preventing Duplicate Entries

sql

CopyEdit

```
INSERT INTO emp (empno, ename, age) VALUES (101, 'John', 30);
INSERT INTO emp (empno, ename, age) VALUES (101, 'Mike', 40); --
```

✗ Error: Duplicate empno!

✗ **Error:** The second INSERT statement fails because empno must be unique.

c. Composite Index (Multi-Column Index)

- ✓ **Purpose:** Indexes multiple columns together for better performance.
- ✓ **Useful when queries involve filtering on multiple columns.**

sql

CopyEdit

```
CREATE INDEX emp_comp_idx ON emp(age, ename);
```


- ◆ This creates an index on **both "age" and "ename"**.
- ◆ Helps optimize queries that filter **by age and name together**.

Example Query Without Index (Slow)

sql

CopyEdit

```
SELECT * FROM emp WHERE age = 30 AND ename = 'John';
```


 **Slow** because it scans the whole table.

Example Query With Composite Index (Fast)

sql

CopyEdit

```
SELECT * FROM emp WHERE age = 30 AND ename = 'John';
```

 **Faster** because it uses the emp_comp_idx index.

4. Checking Indexes After Creation

After creating indexes, we can verify them.

List all indexes again

sql

CopyEdit

```
SELECT * FROM user_indexes;
```

- ◆ Displays all indexes, including those we just created.

Check if a specific index exists

sql

CopyEdit

```
SELECT * FROM user_indexes WHERE table_name = 'EMP' AND index_name = 'EMP_AGE_IDX';
```

- ◆ Returns details of the emp_age_idx index.

5. Understanding How Indexes Work

How does an index improve query performance?

1. The database **creates a sorted structure** for indexed columns.
2. Instead of **scanning all rows**, it:
 - a. **Checks the index** for the searched value.
 - b. **Finds the Row ID (ROWID)** where the data is stored.
 - c. **Fetches data quickly.**

6. When Should You Use Indexing?

✓ Best Use Cases for Indexing

- Frequently used columns in **WHERE, ORDER BY, GROUP BY**.
- High-cardinality columns (**many unique values**, e.g., empno, email).
- Large tables with **millions of records**.

✗ When to Avoid Indexing

- Small tables (**full table scan is faster**).
- Low-cardinality columns (**few unique values**, e.g., gender with values M/F).
- Tables with frequent **INSERT, UPDATE, DELETE** (index maintenance is costly).

7. Removing an Index

If an index is no longer needed, you can remove it.

sql
CopyEdit

```
DROP INDEX emp_age_idx;
```

- ◆ Deletes the emp_age_idx index from the emp table.


8. Quick Comparison of Index Types

Index Type	Purpose	Example
Non-Unique Index	Speeds up searches but allows duplicates	CREATE INDEX emp_age_idx ON emp(age);
Unique Index	Ensures uniqueness in a column	CREATE UNIQUE INDEX emp_id_idx ON emp(empno);
Composite Index	Indexes multiple columns together	CREATE INDEX emp_comp_idx ON emp(age, ename);

9. Real-World Example


Before Indexing (Slow Query)

```
sql
CopyEdit
SELECT * FROM emp WHERE age = 30;
```

 **Slow** because it scans all rows.

After Indexing (Fast Query)

```
sql
CopyEdit
CREATE INDEX emp_age_idx ON emp(age);
SELECT * FROM emp WHERE age = 30;
```

 **Faster** because it uses the index for quick lookup.

10. Key Takeaways

- **Indexes improve SELECT performance** but **slow down data modifications**.
- **Use the right index type** based on query patterns.
- **Avoid excessive indexing** to prevent performance issues.

SQL Synonyms - Study and Revision Notes

1. What is a Synonym in SQL?

A **Synonym** in SQL is an alias or an alternative name for a database object such as a table, view, sequence, or procedure. It allows users to refer to an object without needing to know its actual schema or owner.

Why Use Synonyms?

- ✓ **Simplifies object access** – Users can refer to objects without specifying schema names.
- ✓ **Provides security abstraction** – Hides underlying object details.
- ✓ **Useful in multi-schema environments** – Makes cross-schema queries easier.
- ✓ **Ensures consistency** – Even if the underlying table moves, queries remain valid.

2. Checking Existing Synonyms

Before creating synonyms, we can check the existing ones.

```
sql
CopyEdit
SELECT * FROM user_synonyms;
```

- ◆ This retrieves all synonyms present in the database.
- ◆ Useful to check if a synonym already exists before creating a new one.

3. Creating a Synonym

Syntax:

```
sql
CopyEdit
CREATE SYNONYM synonym_name FOR original_object;
```

- ◆ This creates a synonym that acts as an alias for the original database object.

Example: Creating a Synonym for the emp Table

```
sql
CopyEdit
CREATE SYNONYM emp_tbl FOR emp;
```

- ◆ The synonym emp_tbl now refers to the emp table.
- ◆ Users can use emp_tbl instead of emp in queries.

4. Using a Synonym in Queries

Once the synonym is created, we can use it like the original table.

Retrieve All Records Using the Synonym

```
sql
CopyEdit
SELECT * FROM emp_tbl;
```

- ◆ This fetches all records from the emp table using the emp_tbl synonym.

Retrieve All Records Using the Original Table Name

```
sql
CopyEdit
SELECT * FROM emp;
```


- ◆ This fetches the same records but using the actual table name.

Retrieve All Columns Using an Alias

```
sql
CopyEdit
SELECT e.* FROM emp e;
```

- ◆ Assigns the alias e to emp and selects all columns.

Retrieve All Columns Using Only the Alias

```
sql
CopyEdit
SELECT * FROM e;
```

- ◆ If the alias e is not declared in the session, this may throw an error.

5. Updating Data Using a Synonym

Synonyms can be used for **UPDATE**, **DELETE**, and **INSERT** operations as well.

```
sql
CopyEdit
UPDATE emp_tbl
SET ename = 'Dnyanesh', age = 33
WHERE empno = 103;
```

- ◆ This updates the ename and age columns for empno = 103.
- ◆ Since emp_tbl refers to emp, this change reflects in emp.

Verify the Update

```
sql
CopyEdit
SELECT * FROM emp;
```

- ◆ This query checks whether the updates are applied to the original table.

6. Checking Indexes and Synonyms

After making changes, we can verify existing indexes and synonyms.

Check Indexes on the Table

```
sql
CopyEdit
SELECT * FROM user_indexes;
```

- ◆ Displays all indexes in the database, including those for emp.

Check Synonyms Again

```
sql
CopyEdit
SELECT * FROM user_synonyms;
```

- ◆ Confirms that the synonym emp_tbl exists.

7. Dropping the Table and Synonym

If the original table is dropped, the synonym remains but becomes **invalid**.

```
sql
CopyEdit
DROP TABLE emp;
```

- ◆ Removes the emp table.
- ◆ The emp_tbl synonym now points to a non-existing object.

Drop the Synonym

```
sql
CopyEdit
DROP SYNONYM emp_tbl;
```

- ◆ Deletes the emp_tbl synonym from the database.

8. Key Takeaways

Concept	Explanation
Synonym	Alternative name for a database object (table, view, sequence, etc.)
Why Use It?	Simplifies access, improves security, and ensures consistency
Creation	CREATE SYNONYM emp_tbl FOR emp;
Usage	Can be used in SELECT, INSERT, UPDATE, and DELETE queries
Dropping	If the original table is dropped, the synonym becomes invalid

- ◆ **Best Practice:** Use synonyms when dealing with objects across different schemas.
- ◆ **Limitation:** A synonym **does not store data**, it only references an existing object.

Study Notes on SELECT Statement in SQL:

The SELECT statement is used to query data from a database, allowing you to retrieve specific rows and columns based on conditions. Let's go through each snippet you provided in detail.

1. Simplest SELECT Query (No filter on rows and columns)

```
sql
CopyEdit
SELECT * FROM SCOTT.EMP;
```

- **SELECT *:** This selects all columns in the table.
- **FROM SCOTT.EMP:** Specifies the source table (EMP) within the schema (SCOTT).
- The result will display all rows and columns of the EMP table.

2. Filter Rows using WHERE Clause

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE DEPTNO=30;
```

- WHERE DEPTNO=30: Filters the rows where the DEPTNO (Department Number) is 30.
- It returns all columns for employees who belong to department 30.

3. Filter Specific Columns

sql

CopyEdit

```
SELECT EMPNO, ENAME, JOB, SAL FROM SCOTT.EMP;
```

- SELECT EMPNO, ENAME, JOB, SAL: Instead of using * (which selects all columns), you specify individual columns to retrieve (EMPNO, ENAME, JOB, SAL).
- The result will show only these specific columns for all employees.

4. Filter Rows and Columns Together

sql

CopyEdit

```
SELECT EMPNO, ENAME, JOB, SAL FROM SCOTT.EMP WHERE DEPTNO=10;
```

- Filters both rows (by department number 10) and columns (EMPNO, ENAME, JOB, SAL).

5. Where vs Select

- WHERE: Used to filter **rows** based on a condition. It's called **selection**.
- SELECT: Used to filter **columns** you want in the output. Using * selects all columns, but you can specify others for **projection**.

6. Example: Employees Earning More Than 2000

sql

CopyEdit

```
SELECT EMPNO, ENAME, SAL FROM SCOTT.EMP WHERE SAL > 2000;
```

- Filters rows where the SAL (salary) is greater than 2000 and retrieves the specified columns.

7. List of Managers

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE JOB='MANAGER';
```

- This selects all employees whose job title is MANAGER.

8. Employees Who Joined After 31st Dec 1981

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE HIREDATE > '31-DEC-1981';
```

- Filters employees whose HIREDATE is after December 31, 1981.

9. Get the Current System Date

sql

CopyEdit

```
SELECT SYSDATE FROM DUAL;
```

- SYSDATE: Returns the current system date.
- DUAL: A special dummy table in Oracle used for functions like SYSDATE.

10. List of Employees in Dept 10 and 20

sql

CopyEdit

```
SELECT ENAME FROM SCOTT.EMP WHERE DEPTNO=10 OR DEPTNO=20;  
SELECT ENAME FROM SCOTT.EMP WHERE DEPTNO IN(10, 20);  
SELECT ENAME FROM SCOTT.EMP WHERE DEPTNO = ANY(10, 20);
```

- The first query uses OR to filter rows where DEPTNO is either 10 or 20.
- The second uses IN, which is more concise and clearer when checking against multiple values.
- The third uses ANY, which works in a similar way but can be used with comparison operators.

11. Handling NULL Values

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE COMM = NULL;    -- Incorrect  
SELECT * FROM SCOTT.EMP WHERE COMM IS NULL;    -- Correct
```

- COMM = NULL is invalid because NULL cannot be compared using the = operator.
- Use IS NULL to check for NULL values.

12. Employees with Non-Null Commission

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE COMM IS NOT NULL;
```

- Filters employees who have a non-null value in the COMM (commission) column.

13. Employees with Specific Job and Salary Conditions

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE JOB='ANALYST' AND SAL > 2000;
```

- Filters rows where the employee's job is ANALYST and their salary is greater than 2000.

14. Salary Range Filter

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE SAL >= 1000 AND SAL <= 2000;
```

```
SELECT * FROM SCOTT.EMP WHERE SAL BETWEEN 1000 AND 2000;
```

- Both queries filter rows where the SAL is between 1000 and 2000, inclusive.
- BETWEEN is just a shorthand for the comparison operators (>= and <=).

15. Using LIKE for Pattern Matching

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE ENAME LIKE 'A%';
```

- Finds employees whose names start with A.
- % is a wildcard that matches any sequence of characters.

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE ENAME LIKE '%E';
```

- Finds employees whose names end with E.

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE ENAME LIKE '_A%';
```

- Finds employees whose second character is A. _ matches any single character.

16. Employees with Salary Greater Than Ford and Miller

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE SAL > ALL(1300, 3000);
```

- Filters employees whose salary is greater than **both** 1300 and 3000.

sql

CopyEdit

```
SELECT * FROM SCOTT.EMP WHERE SAL > ANY(1300, 3000);
```

- Filters employees whose salary is greater than **either** 1300 or 3000.

17. Column Aliases

sql

CopyEdit

```
SELECT ENAME EMP_NAME, JOB JOB_DESCRIPTION, DEPTNO DEPARTMENT_NUMBER  
FROM SCOTT.EMP;
```

- EMP_NAME, JOB_DESCRIPTION, and DEPARTMENT_NUMBER are aliases for the ENAME, JOB, and DEPTNO columns, respectively.

sql

CopyEdit

```
SELECT ENAME AS EMP_NAME, JOB AS JOB_DESCRIPTION, DEPTNO AS  
DEPARTMENT_NUMBER FROM SCOTT.EMP;
```

- AS keyword is optional in Oracle but can be used for better readability when assigning aliases.

18. Ordering Results

sql

CopyEdit

```
SELECT DEPTNO, ENAME, JOB, SAL FROM SCOTT.EMP ORDER BY SAL ASC;
```

- Orders the result set by salary (SAL) in ascending order (ASC).

sql

CopyEdit

```
SELECT DEPTNO, ENAME, JOB, SAL FROM SCOTT.EMP ORDER BY SAL DESC;
```

- Orders the result set by salary in descending order (DESC).

sql

CopyEdit

```
SELECT DEPTNO, ENAME, JOB, SAL FROM SCOTT.EMP ORDER BY DEPTNO DESC,  
SAL DESC;
```

- Orders by department number (DEPTNO) first, and then by salary within each department.

Summary:

- **Clauses** in the SELECT statement:
 - SELECT: Specifies columns to retrieve.
 - FROM: Specifies the table to query.
 - WHERE: Filters the rows based on conditions.
 - ORDER BY: Sorts the result set.
- **Common operators for filtering:**
 - IN, ANY, ALL, BETWEEN, LIKE, comparison operators (=, <, >, <=, >=, !=, <>), IS NULL, IS NOT NULL, AND, OR.

SINGLE ROW FUNCTIONS

Study and Revision Notes on String Functions

Here's an in-depth explanation of each SQL string function, along with code snippets:

1. LENGTH()

- **Description:** Returns the length of a string in characters.
- **Example:**

sql

CopyEdit

```
SELECT ename, LENGTH(ename) FROM scott.emp;
```

- This query retrieves the length of the ename (employee name) for each employee from the EMP table.

2. CONCAT()

- **Description:** Concatenates (joins) two strings together.
- **Example:**

sql

CopyEdit

```
SELECT ename, job, CONCAT(ename, job) FROM scott.emp;
```

- This query combines the employee's name (ename) and job (job).

Error Example:

sql

CopyEdit

```
SELECT ename, job, CONCAT(ename, ' ', job) FROM scott.emp;
```

- This results in an error because CONCAT() only accepts two parameters at a time.

- **Correct Approach:**

sql

CopyEdit

```
SELECT ename, job, CONCAT(ename, CONCAT(' ', job)) FROM scott.emp;
```

- This correctly concatenates ename with job, separated by a space.

Alternatively:

sql

CopyEdit

```
SELECT ename, job, ename || ' is a ' || job AS Emp_info FROM  
scott.emp;
```

- This uses ||, the concatenation operator, to join strings with custom text.

3. UPPER()

- **Description:** Converts a string to uppercase.
- **Example:**

sql

CopyEdit

```
SELECT UPPER('deloitte') FROM dual;
```

- Converts the string 'deloitte' to 'DELOITTE'.

4. LOWER()

- **Description:** Converts a string to lowercase.
- **Example:**

sql

CopyEdit

```
SELECT LOWER('DELOITTE') FROM dual;
```

- Converts the string 'DELOITTE' to 'deloitte'.

5. INITCAP()

- **Description:** Converts the first letter of each word in the string to uppercase and the rest to lowercase.
- **Example:**

sql

CopyEdit

```
SELECT INITCAP('deloitte') FROM dual;
```

- Converts the string 'deloitte' to 'Deloitte'.

6. LTRIM()

- **Description:** Removes leading spaces (spaces from the left side) from a string.
- **Example:**

sql

CopyEdit

```
SELECT LTRIM('    Welcome to Deloitte    ') FROM dual;
```

- Removes the spaces at the beginning of ' Welcome to Deloitte ', resulting in 'Welcome to Deloitte '.

7. RTRIM()

- **Description:** Removes trailing spaces (spaces from the right side) from a string.
- **Example:**

sql

CopyEdit

```
SELECT RTRIM('    Welcome to Deloitte    ') FROM dual;
```

- Removes the spaces at the end, resulting in ' Welcome to Deloitte'.

8. TRIM()

- **Description:** Removes both leading and trailing spaces from a string.
- **Example:**

sql

CopyEdit

```
SELECT TRIM('    Welcome to Deloitte    ') FROM dual;
```

- Removes spaces from both sides, resulting in 'Welcome to Deloitte'.

Error Example:

sql

CopyEdit

```
SELECT TRIM('#####Welcome to Deloitte#####', '#') FROM dual;
```

- This results in an error because the TRIM() function in Oracle only works with spaces. You need to use LTRIM() and RTRIM() for specific characters.

9. LPAD() and RPAD()

- **Description:** These functions pad the string with a specified character until the string reaches a certain length.
- **LPAD():** Pads the string on the left (beginning) side.

sql

CopyEdit

```
SELECT ename, LPAD(ename, 10, '#') FROM scott.emp;
```

- Pads ename to a length of 10, adding # characters to the left.

- **RPAD():** Pads the string on the right (end) side.

sql

CopyEdit

```
SELECT ename, RPAD(ename, 10, '#') FROM scott.emp;
```

- Pads ename to a length of 10, adding # characters to the right.

- **Handling Strings Longer Than Specified Length:**

sql

CopyEdit

```
SELECT ename, LPAD(ename, 5, '#') FROM scott.emp;
```

```
SELECT ename, RPAD(ename, 5, '#') FROM scott.emp;
```

- If the original string exceeds the specified length, it will not be truncated but will remain intact.

10. REPLACE()

- **Description:** Replaces all occurrences of a specified substring within a string with another substring.
- **Example:**

sql

CopyEdit

```
SELECT ename, REPLACE(ename, 'A', 'B') FROM scott.emp;
```

- Replaces every occurrence of 'A' with 'B' in the employee names.

11. SUBSTR()

- **Description:** Extracts a substring from a given string.
- **Example:**

sql

CopyEdit

```
SELECT ename, SUBSTR(ename, 1, 3) FROM scott.emp;
```

- Extracts the first 3 characters of ename.

sql

CopyEdit

```
SELECT ename, SUBSTR(ename, 2, 5) FROM scott.emp;
```

- Extracts 5 characters starting from the 2nd position in ename.

12. INSTR()

- **Description:** Returns the position of the first occurrence of a substring within a string.
- **Example:**

```
sql
CopyEdit
SELECT ename, INSTR(ename, 'A') FROM scott.emp;
```

- Returns the position of the first occurrence of 'A' in ename.

```
sql
CopyEdit
SELECT ename, INSTR(ename, 'A', 3) FROM scott.emp;
```

- Starts searching for 'A' from the 3rd character of ename.

```
sql
CopyEdit
SELECT ename, INSTR(ename, 'A', 2, 2) FROM scott.emp;
```

- Finds the second occurrence of 'A' starting from the second character.

Summary of String Functions:

- **LENGTH():** Finds the length of a string.
- **CONCAT():** Concatenates two strings.
- **UPPER(), LOWER(), INITCAP():** Change case (uppercase, lowercase, and title case).
- **LTRIM(), RTRIM(), TRIM():** Remove leading/trailing spaces or characters.
- **LPAD(), RPAD():** Pad a string to the left or right.
- **REPLACE():** Replace a substring with another.
- **SUBSTR():** Extract part of a string.
- **INSTR():** Find the position of a substring.

Study and Revision Notes on General Functions

These SQL functions are used for handling null values, conditional logic, and pattern matching. Here's a detailed breakdown of each function, with examples:

1. NVL()

- **Description:** The NVL() function is used to replace NULL values with a specified replacement value. It takes two arguments:
 - If the first argument is NULL, it returns the second argument; otherwise, it returns the first argument.
- **Examples:**

```
sql
CopyEdit
SELECT NVL(10, 20) FROM dual;
```

- Since 10 is not NULL, it returns 10.

```
sql
CopyEdit
SELECT NVL(NULL, 20) FROM dual;
```

- Since the first argument is NULL, it returns 20.

```
sql
CopyEdit
SELECT sal, comm, sal + NVL(comm, 0) AS gross_sal FROM scott.emp;
```

- This query calculates the gross salary by adding sal (salary) and comm (commission). If comm is NULL, it substitutes 0 for comm.

2. NVL2()

- **Description:** The NVL2() function takes three arguments. If the first argument is not NULL, it returns the second argument; if the first argument is NULL, it returns the third argument.
- **Examples:**

```
sql
CopyEdit
SELECT NVL2(10, 'Not null', 'Null') FROM dual;
```

- Since the first argument is not NULL, it returns 'Not null'.

```
sql
CopyEdit
SELECT NVL2(NULL, 'Not null', 'Null') FROM dual;
```

- Since the first argument is NULL, it returns 'Null'.

sql

CopyEdit

```
SELECT comm, NVL2(comm, 'Eligible for Comm', 'Not eligible for  
Comm') AS comm_comment FROM scott.emp;
```

- If comm is not NULL, it returns 'Eligible for Comm'; otherwise, it returns 'Not eligible for Comm'.

3. NULLIF()

- **Description:** The NULLIF() function compares two expressions. If they are equal, it returns NULL; otherwise, it returns the first expression.
- **Examples:**

sql

CopyEdit

```
SELECT NULLIF(10, 10) FROM dual;
```

- Since the two values are equal, it returns NULL.

sql

CopyEdit

```
SELECT NULLIF(10, 20) FROM dual;
```

- Since the values are not equal, it returns 10.

4. DECODE()

- **Description:** The DECODE() function is similar to a CASE expression. It compares an expression to a list of possible values, and returns a corresponding result. If the value matches one of the conditions, it returns the corresponding result; otherwise, it returns the default value.
- **Example:**

sql

CopyEdit

```
SELECT ename, deptno, DECODE(deptno, 10, 'Works for dept 10', 'Works  
for other dept') dept_comment FROM scott.emp;
```


- This query returns a comment based on the deptno. If deptno is 10, it returns 'Works for dept 10', otherwise 'Works for other dept'.

5. CASE Statement

- **Description:** The CASE statement is used for conditional logic. It works like an IF-ELSE construct and returns different values based on conditions.
- **Examples:**
 - **Simple CASE Expression:**

```
sql
CopyEdit
SELECT ename, deptno, sal,
       CASE deptno
         WHEN 10 THEN 'Works for Dept 10'
         WHEN 20 THEN 'Works for Dept 20'
         WHEN 30 THEN 'Works for Dept 30'
         ELSE 'Works for other dept'
       END AS dept_comment
FROM scott.emp;
```

- Based on the value of deptno, this query returns a string indicating the department.

- **Searched CASE Expression:**

```
sql
CopyEdit
SELECT ename, deptno, sal,
       CASE
         WHEN sal < 1000 THEN 'Needs more money'
         WHEN sal < 2000 THEN 'Just Okay'
         WHEN sal < 3000 THEN 'Good Salary'
         ELSE 'High Salary'
       END AS salary_comment
FROM scott.emp;
```

- This query categorizes salaries into different levels based on the salary value.

6. COALESCE()

- **Description:** The COALESCE() function returns the first non-NULL value from a list of expressions. It can take multiple arguments.
- **Examples:**

sql

CopyEdit

```
SELECT comm, sal, COALESCE(comm, sal) FROM scott.emp;
```

- If comm is NULL, it will return the value of sal (salary); otherwise, it will return comm (commission).

sql

CopyEdit

```
SELECT COALESCE(NULL, NULL, NULL, 10, NULL, 20, NULL, 20, 30) FROM dual;
```

- This query returns the first non-NULL value from the list, which is 10.

7. REGEXP_LIKE()

- **Description:** The REGEXP_LIKE() function is used for pattern matching using regular expressions. It checks whether a string matches a specified regular expression pattern.
- **Examples:**

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, '^A');
```

- This returns employee names that start with the letter 'A'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, 'E$');
```

- This returns employee names that end with the letter 'E'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, '^.A');
```

- This returns employee names where the second letter is 'A'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, '.A');
```

- This returns employee names where the second-to-last character is 'A'.

8. Regular Expression Quantifiers in REGEXP_LIKE()

- **Description:** You can use quantifiers to match specific numbers of occurrences in a string.
- **Examples:**

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, 'T*');
```

- Matches names that contain zero or more occurrences of the letter 'T'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, 'T+');
```

- Matches names that contain one or more occurrences of the letter 'T'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, 'T?');
```

- Matches names that contain zero or one occurrence of the letter 'T'.

sql

CopyEdit

```
SELECT ename FROM scott.emp WHERE REGEXP_LIKE(ename, '[ABC]');
```

- Matches names that contain either the letter 'A', 'B', or 'C'.

9. REGEXP_REPLACE()

- **Description:** The REGEXP_REPLACE() function searches a string for a pattern and replaces it with a specified string.
- **Example:**

sql

CopyEdit

```
SELECT ename, REGEXP_REPLACE(ename, '[A-H]', 'Z') FROM scott.emp;
```

- This query replaces all characters from 'A' to 'H' in ename with 'Z'.

Summary of Functions:

- **NVL()**: Replaces NULL with a specified value.
- **NVL2()**: Returns one value if the first argument is not NULL and another if it is.
- **NULLIF()**: Returns NULL if two values are equal; otherwise, returns the first value.
- **DECODE()**: Provides conditional logic by matching expressions.
- **CASE**: More flexible conditional logic for handling multiple conditions.
- **COALESCE()**: Returns the first non-NULL value from a list of arguments.
- **REGEXP_LIKE()**: Used for regular expression pattern matching.
- **REGEXP_REPLACE()**: Replaces parts of a string that match a regular expression with a new value.

Study and Revision Notes on Number Functions

SQL provides various number functions to handle mathematical operations and number manipulations. Below is a breakdown of each function used in the provided code snippets:

1. MOD()

- **Description:** The MOD() function returns the remainder of the division of two numbers.
 - Syntax: MOD(dividend, divisor)
- **Examples:**

sql

CopyEdit

```
SELECT MOD(10, 2) FROM dual;
```

- This divides 10 by 2 and returns the remainder, which is 0.

sql

CopyEdit

```
SELECT MOD(2, 10) FROM dual;
```

- This divides 2 by 10 and returns the remainder, which is 2, because 2 is less than 10.

sql

CopyEdit

```
SELECT MOD(10, 3) FROM dual;
```

- This divides 10 by 3 and returns the remainder, which is 1.

2. ABS()

- **Description:** The ABS() function returns the absolute value of a number, removing any negative sign.
 - Syntax: ABS(number)
- **Examples:**

sql

CopyEdit

```
SELECT ABS(10) FROM dual;
```

- This returns 10, as the absolute value of 10 is 10.

sql

CopyEdit

```
SELECT ABS(-10) FROM dual;
```

- This returns 10, as the absolute value of -10 is 10.

3. POWER()

- **Description:** The POWER() function returns the value of a number raised to the power of another number.
 - Syntax: POWER(base, exponent)
- **Example:**

sql

CopyEdit

```
SELECT POWER(10, 3) FROM dual;
```

- This calculates 10 raised to the power of 3, which is 1000.

4. ROUND()

- **Description:** The ROUND() function rounds a number to a specified number of decimal places. If the number of decimal places is not specified, it rounds to the nearest integer.

- Syntax: ROUND(number, [decimal_places])

- **Examples:**

```
sql
```

```
CopyEdit
```

```
SELECT ROUND(10.1234) FROM dual;
```

- This rounds 10.1234 to the nearest integer, resulting in 10.

```
sql
```

```
CopyEdit
```

```
SELECT ROUND(10.5234) FROM dual;
```

- This rounds 10.5234 to the nearest integer, resulting in 11.

```
sql
```

```
CopyEdit
```

```
SELECT ROUND(10.1234, 2) FROM dual;
```

- This rounds 10.1234 to two decimal places, resulting in 10.12.

```
sql
```

```
CopyEdit
```

```
SELECT ROUND(1234.1234, -2) FROM dual;
```

- This rounds 1234.1234 to the nearest hundred, resulting in 1200.

5. TRUNC()

- **Description:** The TRUNC() function truncates a number to a specified number of decimal places. Unlike ROUND(), TRUNC() does not round the number, but simply removes the digits after the decimal point (or truncates digits before the decimal point if a negative value is provided).

- Syntax: TRUNC(number, [decimal_places])

- **Examples:**

sql

CopyEdit

```
SELECT TRUNC(10.1234) FROM dual;
```

- This truncates 10.1234 to 10, removing the decimals.

sql

CopyEdit

```
SELECT TRUNC(10.9234) FROM dual;
```

- This truncates 10.9234 to 10.

sql

CopyEdit

```
SELECT TRUNC(1234.1234, 2) FROM dual;
```

- This truncates 1234.1234 to two decimal places, resulting in 1234.12.

sql

CopyEdit

```
SELECT TRUNC(1234.1234, -2) FROM dual;
```

- This truncates 1234.1234 to the nearest hundred, resulting in 1200.

6. FLOOR()

- **Description:** The FLOOR() function returns the largest integer less than or equal to the specified number. Essentially, it "rounds down" to the nearest whole number.

- Syntax: FLOOR(number)

- **Examples:**

sql

CopyEdit

```
SELECT FLOOR(10.1) FROM dual;
```

- This rounds down 10.1 to 10.

sql

CopyEdit

```
SELECT FLOOR(10.9) FROM dual;
```

- This rounds down 10.9 to 10.

7. CEIL()

- **Description:** The CEIL() function returns the smallest integer greater than or equal to the specified number. Essentially, it "rounds up" to the nearest whole number.
 - Syntax: CEIL(number)
- **Examples:**

sql

CopyEdit

```
SELECT CEIL(10.1) FROM dual;
```

- This rounds up 10.1 to 11.

sql

CopyEdit

```
SELECT CEIL(10.9) FROM dual;
```

- This rounds up 10.9 to 11.

Summary of Number Functions:

1. **MOD():** Returns the remainder of a division.
2. **ABS():** Returns the absolute value of a number.
3. **POWER():** Raises a number to the specified power.
4. **ROUND():** Rounds a number to a specified number of decimal places.
5. **TRUNC():** Truncates a number to a specified number of decimal places without rounding.
6. **FLOOR():** Returns the largest integer less than or equal to the number.
7. **CEIL():** Returns the smallest integer greater than or equal to the number.