Introduction to SQL

- SQL is a standard language for accessing and manipulating databases

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

What Can SQL Do?

- SQL can execute queries against a database.
- SQL can retrieve data from a database.
- SQL can insert records in a database.
- SQL can update records in a database.
- SQL can delete records from a database.
- SQL can create new databases.
- SQL can create new tables in a database.
- SQL can create stored procedures in a database.
- SQL can create views in a database.
- SQL can set permissions on tables, procedures, and views.

Features of SQL

- SQL is an ANSI and ISO standard computer language for creating and manipulating databases.
- SQL allows the user to create, update, delete, and retrieve data from a database.
- SQL is very simple and easy to learn.
- SQL works with database programs like DB2, Oracle, MS Access, Sybase, MS SQL Server etc.

Advantages of SQL

High Speed:

- SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.
- Well Defined Standards Exist: SQL databases use long-established standards, which are being adopted by ANSI & ISO. NonSQL databases do not adhere to any clear standard.

No Coding Required:

- Using standard SQL it is easier to manage database systems without having to write a substantial amount of code.

Emergence of ORDBMS:

- Previously SQL databases were synonymous with relational databases.
- With the emergence of Object Oriented DBMS, object storage capabilities are extended to relational databases.

Disadvantages of SQL

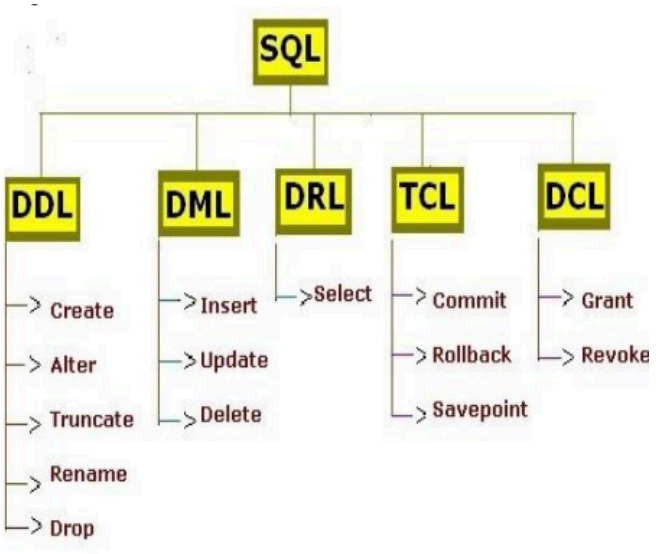
Difficulty in Interfacing:

- Interfacing an SQL database is more complex than adding a few lines of code.

More Features Implemented in Proprietary way:

- Although SQL databases conform to ANSI & ISO standards, some databases go for proprietary extensions to standard SQL to ensure vendor lock-in.

SQL COMPONENTS



SQL - Structured Query Language
DDL - Data Definition Language
DML - Data Manipulation Language
DRL - Data Retrieval Language
TCL - Transaction Control Language
DCL - Data Control Language

DDL – Data Definition Language

Naming Rules

Table names and column names:

- Must begin with a letter
- Must be 1–30 characters long
- Must contain only A–Z, a–z, 0–9, _, $, and #
- Must not duplicate the name of another object owned by the same user
- Must not be an Oracle server reserved word

CREATE TABLE Statement

• You must have:

- CREATE TABLE privilege
- A storage area

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

• You specify:

- Table name
- Column name, column data type, and column size

Creating Tables

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
Table created.
```

Confirm table creation

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

Data Types

Data Type	Description
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p,s)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Including Constraints

• Constraints enforce rules at the table level.

• Constraints prevent the deletion of a table if there are dependencies.

• The following constraint types are valid:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

Constraint Guidelines

• You can name a constraint, or the Oracle server generates a name by using the SYS_Cn format.

• Create a constraint at either of the following times:

- At the same time as the table is created
- After the table has been created

• Define a constraint at the column or table level.

• View a constraint in the data dictionary.

Defining Constraints

- **Syntax**

```
CREATE TABLE [schema.]table
      (column datatype [DEFAULT expr]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

- **Column-level Constraint**

```
column [CONSTRAINT constraint_name] constraint_type,
```

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

1

- **Table-level Constraint**

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

2

NOT NULL Constraint

- Ensures that null values are not permitted for the column.
- Absence of NOT NULL constraint means any row can contain a null value for this column.
- Can be placed only at column level.

UNIQUE Constraint

Defined at either Table-level or Column-level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - ➔ References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - ➔ Calls to SYSDATE, UID, USER, and USERENV functions
  - ➔ Queries that refer to other values in other rows

```
..., salary  NUMBER(2)
     CONSTRAINT emp_salary_min
                CHECK (salary > 0),...
```

Primary Key

- In Oracle, a primary key is a single field or combination of fields that uniquely defines a record.
- None of the fields that are part of the primary key can contain a null value.
- A table can have only one primary key.
- Note:
  - ➔ In Oracle, a primary key cannot contain more than 32 columns.
  - ➔ A primary key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Foreign Key

- A foreign key is a way to enforce referential integrity within your Oracle database.
- A foreign key means that values in one table must also appear in another table.
- The referenced table is called the parent table while the table with the foreign key is called the child table.
- The foreign key in the child table will generally reference a primary key in the parent table.
- A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.
- Note:
  - ➔ Foreign Key can be a NULL.
  - ➔ We can restrict NULL values in foreign key column by including NOT NULL constraint.

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
       CONSTRAINT     emp_employee_id   PRIMARY KEY
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
       CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
       CONSTRAINT     emp_email_nn      NOT NULL
       CONSTRAINT     emp_email_uk      UNIQUE
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
       CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
       CONSTRAINT     emp_job_nn        NOT NULL
    , salary         NUMBER(8,2)
       CONSTRAINT     emp_salary_ck     CHECK (salary>0)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
       CONSTRAINT     emp_dept_fk       REFERENCES
          departments (department_id));
```

Primary Key Vs Foreign Key

Primary key	Foreign key
Primary key uniquely identify a record in the table.	Foreign key is a field in the table that is primary key in another table.
Primary Key can't accept null values.	Foreign key can accept multiple null value.
By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.	Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
We can have only one Primary key in a table.	We can have more than one foreign key in a table.
Example : CREATE TABLE DEPARTMENTS (DEPTNO NUMBER(3) CONSTRAINT PK_ID_NOT_VALID PRIMARY KEY);	Example : CREATE TABLE EMPLOYEES (DEPTNO NUMBER(3), FOREIGN KEY(DEPTNO) REFERENCES DEPARTMENTS);

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You cannot roll back the DROP TABLE statement.
- But we can recover it by using flashback if it is not permanently dropped.

Syntax:
DROP TABLE table_name [PURGE];

Example:
DROP TABLE employees [PURGE];

If we skip purge keyword with dropping then dropped table is stored in recycle bin. That we can recover by using flashback query.

FLASHBACK TABLE employees TO BEFORE DROP;

We can clear the recycle bin using purge recycle bin command

- PURGE RECYCLEBIN;

ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

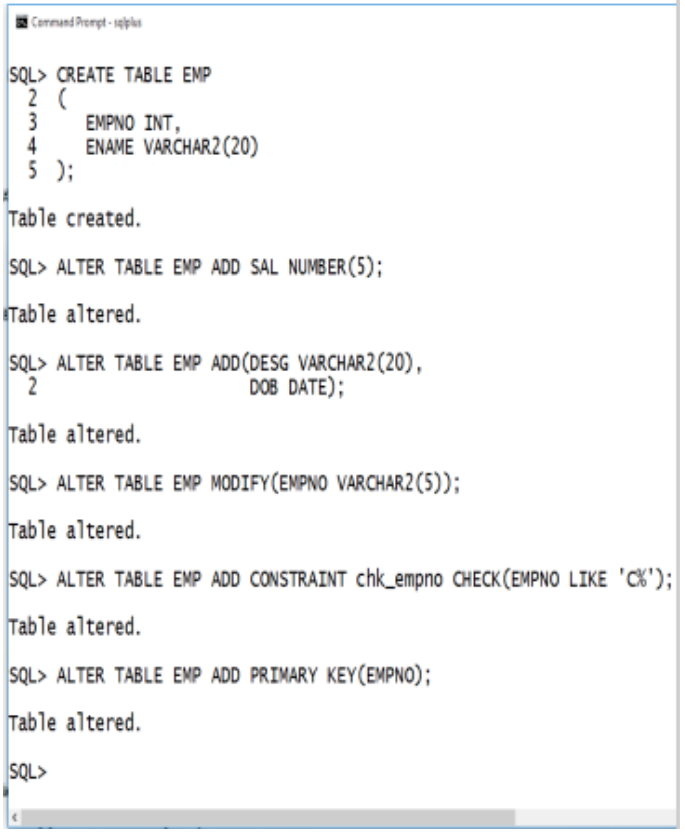
Use the ALTER TABLE statement to add, modify, or drop columns.

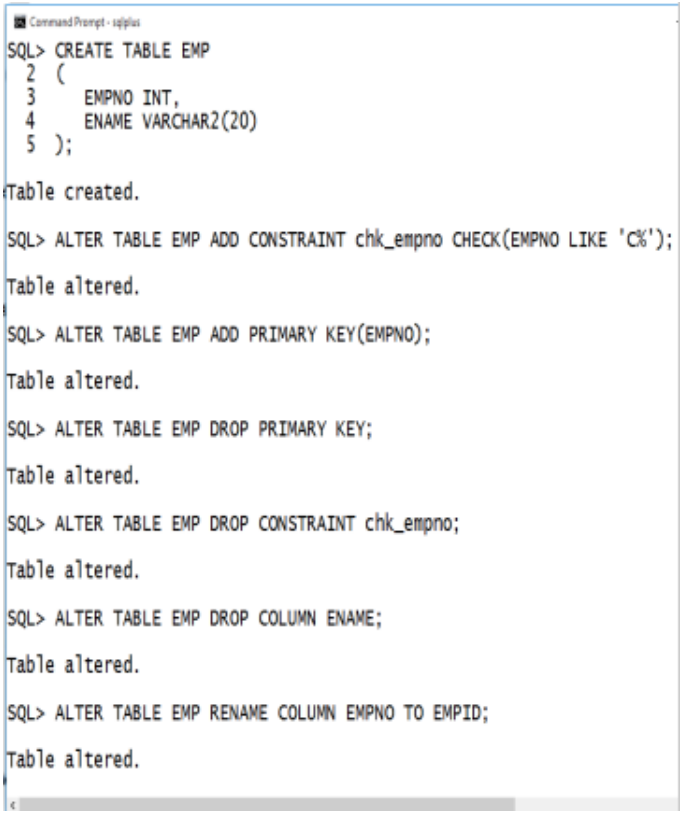
```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP        (column);
```

- ALTER TABLE tab_name ADD (COL1 DATATYPE [CONSTRAINT], COL2 DATATYPE [CONSTRAINT]);
- ALTER TABLE tab_name DROP COLUMN col_name; ✓ ALTER TABLE tab_name MODIFY(col_name DATATYPE [CONSTRAINT]);
- ALTER TABLE tab_name ADD [CONSTRAINT cons_name] cons_type;
- ALTER TABLE tab_name DROP CONSTRAINT cons_name;
- ALTER TABLE tab_name RENAME COLUMN old_name TO new_name;
- ALTER TABLE tab_name RENAME TO new_name;





TRUNCATE TABLE

- The SQL TRUNCATE TABLE statement is used to remove all records from a table.
- It performs the same function as a DELETE statement without a WHERE clause.
- NOTE:
  - ➔ If you truncate a table, the TRUNCATE TABLE statement cannot be rolled back
- The syntax for the TRUNCATE TABLE statement in SQL is:

```
TRUNCATE TABLE table_name;
```

TRUNCATE	DELETE
TRUNCATE is faster than DELETE	DELETE is slower than TRUNCATE
It is DDL statement	It is DML statement
TRUNCATE doesn't fire trigger	DELETE fire triggers
TRUNCATE doesn't support WHERE clause	DELETE support WHERE clause
ROLLBACK is not possible with TRUNCATE because it is auto commit	ROLLBACK is possible with DELETE
TRUNCATE reset identity column in table	DELETE doesn't reset anything
TRUNCATE doesn't do logging	DELETE can be used for LOG on row basis
Syntax TRUNCATE table table_name	**Syntax** DELETE FROM table_name [Where condition]
Example TRUNCATE TABLE EMP;	**Example** DELETE FROM EMP;

Data Control Language (DCL)

- DCL commands are used to enforce database security in a multiple user database environment.
- Two types of DCL commands are o GRANT and REVOKE.
- Only Database Administrator's or owners of the database object can provide/remove privileges on a database object.

  GRANT privilege_name
  ON object_name
  TO {user_name |PUBLIC |role_name}
  [WITH GRANT OPTION];

System Level Privileges	Table Level Privileges
System privileges are normally granted by a DBA to users.	Object privileges means privileges on objects such as tables, views, synonyms, procedure. These are granted by owner of the object.
This privileges allows a user to manage database and server.	This privileges allows a user to perform certain action upon certain database objects.
List of privileges: • CREATE USER • CREATE TABLE • CREATE SESSION	List of privileges: • SELECT • INSERT • UPDATE, DELETE • EXECUTE
To get information about system level privileges: SELECT * FROM USER_SYS_PRIVS SELECT * FROM ROLE_SYS_PRIVS	To get information about object level privileges SELECT OWNER,TABLE_NAME,PRIVILEGE FROM USER_TAB_PRIVS
Syntax GRANT privileges TO username;	**Syntax** GRANT privileges ON object TO username;
Example GRANT CREATE SESSION, CREATE PROCEDURE TO PARAG;	**Example** GRANT SELECT, INSERT ON EMPLOYEE TO PARAG;

Data Manipulation Language (DML)

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A transaction consists of a collection of DML statements that form a logical unit of work.
- Add new rows to a table by using the INSERT statement:

```
INSERT INTO  table [(column [, column...])]
VALUES       (value [, value...]);
```

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the INSERT clause.

```
INSERT INTO departments(department_id,
      department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

- Enclose character and date values in single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO  departments (department_id,
                          department_name    )
VALUES       (30, 'Purchasing');
1 row created.
```

- Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO  departments
VALUES       (100, 'Finance', NULL, NULL);
1 row created.
```

Copying Rows from another Table

- Write your INSERT statement with a subquery:
- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows created.
```

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause:

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

DELETE Statement

- You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]   table
[WHERE          condition];
```

Specific rows are deleted if you specify the WHERE clause:

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

All rows in the table are deleted if you omit the WHERE clause

```
DELETE FROM  copy_emp;
22 rows deleted.
```

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- **Syntax:**
  - TRUNCATE TABLE table_name;
- **Example:**
  - TRUNCATE TABLE copy_emp;

Database Transactions

- Begin when the first DML SQL statement is executed.
- End with one of the following events:
  - o A COMMIT or ROLLBACK statement is issued.
  - o A DDL or DCL statement executes (automatic commit).
  - o The user exits iSQL*Plus. o The system crashes.

Transaction Control Statement (TCL)

- There are following commands used to control transactions:
  - o COMMIT: to save the changes.
  - o ROLLBACK: to undo the changes.
  - o SAVEPOINT: creates points within groups of transactions in which to ROLLBACK
- Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

COMMIT Command:

- COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
- COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.
- The syntax for COMMIT command is as follows:

```
COMMIT;
```

The ROLLBACK Command:

- ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.
- The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
- The syntax for ROLLBACK command is as follows:

```
ROLLBACK;
```

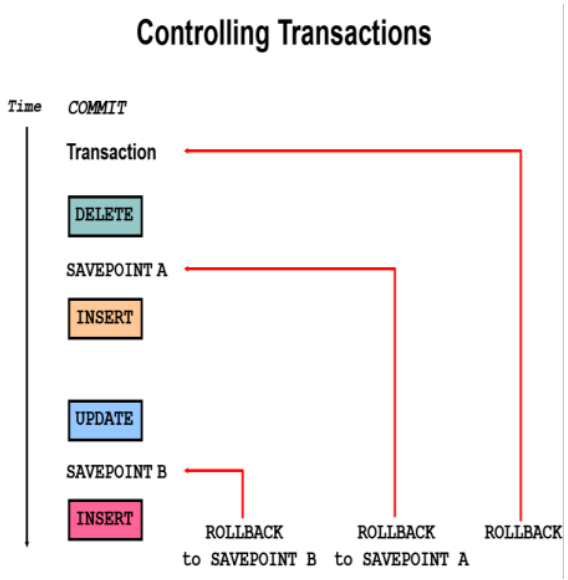
SAVEPOINT Command:

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
- The syntax for SAVEPOINT command is as follows:

```
SAVEPOINT SAVEPOINT_NAME;
```

- This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions.
- The syntax for rolling back to a SAVEPOINT is as follows:

```
ROLLBACK TO SAVEPOINT_NAME;
```



BASIC SQL SELECT STATEMENT

```
SELECT   *|{[DISTINCT] column|expression [alias],...}
FROM     table;
```

- SELECT identifies what columns
- FROM identifies which table
- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

Arithmetic Expressions

- Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Expression

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

Operator Precedence

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```
(1)

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```
(2)

Defining NULL Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as a zero or a blank space.
- Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

Defining a Column Alias

A column alias:

- Renames a column heading • Is useful with calculations.
- Immediately follows the column name (There can also be the optional AS keyword between the column name and alias.)
- Requires double quotation marks if it contains spaces or special characters or if it is case-sensitive

Concatenation Operator

A concatenation operator:

- Links columns or character strings to other columns.
- Is represented by two vertical bars (||).
- Creates a resultant column that is a character expression.

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```

Literal Character Strings

- A literal is a character, a number, or a date that is included in the SELECT statement.
- Date and character literal values must be enclosed by single quotation marks.
- Each character string is output once for each row returned.

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

Duplicate Rows

- The default display of queries is all rows, including duplicate rows.

```
SELECT department_id
FROM   employees;
```
1

```
SELECT DISTINCT department_id
FROM   employees;
```
2

Displaying Table Structure

- Use the iSQL*Plus DESCRIBE command to display the structure of a table:
- DESC[RIBE] tablename

Restricting and Sorting Data

- Where is used to Limit the rows that are retrieved by a query.
- Order By is used to Sort the rows that are retrieved by a query.
- **Restrict the rows that are returned by using the WHERE clause**

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table
[WHERE condition(s)];
```

- **The WHERE clause follows FROM clause.**

Using WHERE clause

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

Character Strings and Dates

- Character strings and date values are enclosed by single quotation marks.
- Character values are case-sensitive, and date values are format-sensitive.
- The default date format is DD-MON-RR.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Shukla' ;
```

Comparison Conditions

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Using Comparing Conditions

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

Using the BETWEEN Condition

- **Use the BETWEEN condition to display rows based on range of values:**

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Lower limit Upper limit

Using the IN Condition

- **Use the IN membership condition to test for values in a list:**

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

Using the LIKE Condition

- Use the **LIKE** condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
- **% denotes zero or many characters**
- **_ denotes one character**

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%' ;
```

- You can combine pattern-matching characters:

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

- You can use the ESCAPE identifier to search for the actual % and _ symbols.

SELECT DISTINCT JOB_ID FROM EMPLOYEES WHERE JOB_ID LIKE 'SA_%' ESCAPE '\'

Using the NULL Conditions

- Test for nulls with the IS NULL operator.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

Logical Conditions

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the following condition is false

Using AND Operator

- **AND** requires both conditions to be **TRUE**.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%' ;
```

Using OR Operator

- **OR** requires either condition to be **TRUE**.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

Using the NOT Operator

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

Using the ORDER BY Clause

- Sort retrieved rows with the **ORDER BY** clause:
- **ASC:** ascending order, default
- **DESC:** descending order
- The **ORDER BY** clause comes last in the **SELECT** statement.

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

Sorting

- Sorting in descending order:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC ;
```
1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```
2

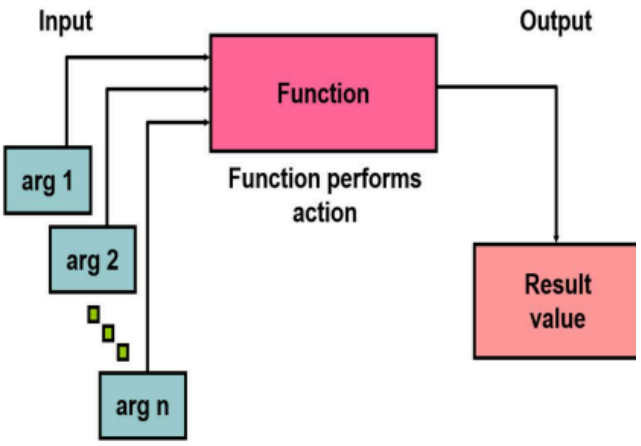
- Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```
3

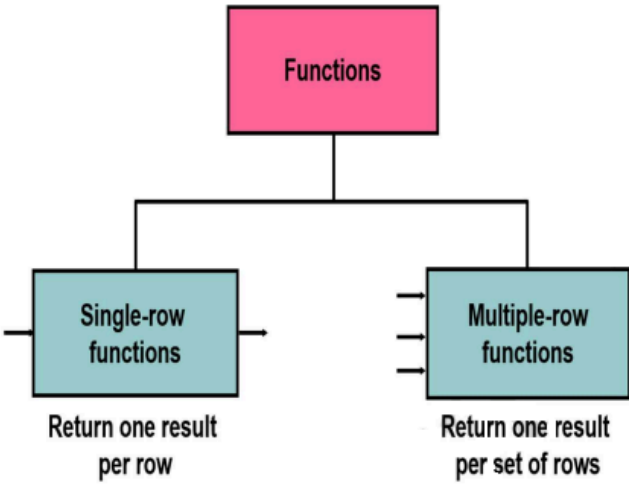
The default sort order is ascending:

- Numeric values are displayed with the lowest values first (for example, 1 to 999).
- Date values are displayed with the earliest value first (for example, 01-JAN-92 before 01-JAN-95).
- Character values are displayed in alphabetical order (for example, A first and Z last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can sort by a column that is not in the SELECT list.

SQL Functions



Two Types of SQL Function

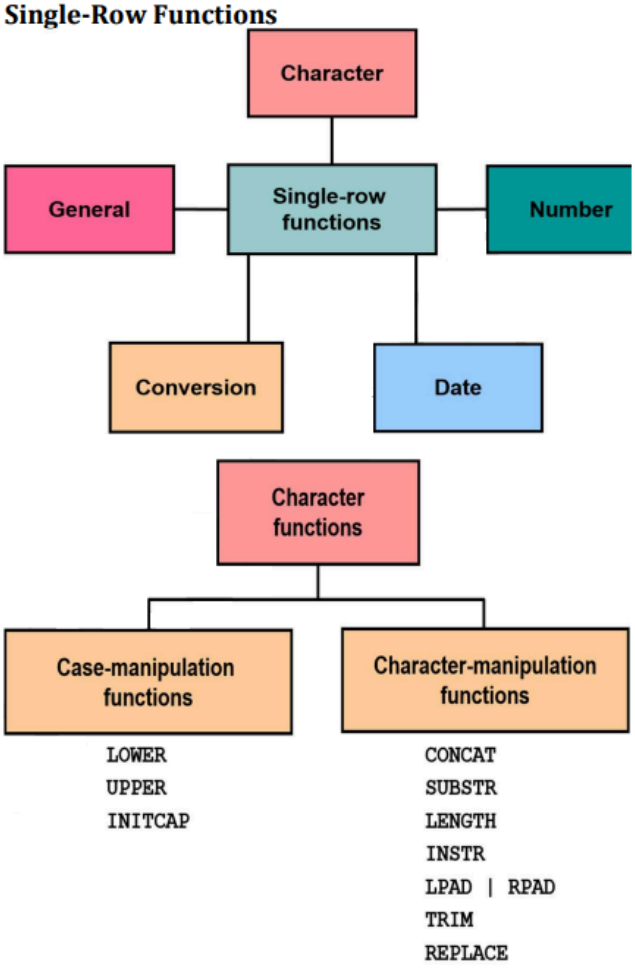


Single-Row Functions

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2,...)]
```

Single-Row Functions



Case-Manipulation Functions

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Using Case-Manipulation Functions

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'Bhatt';
no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'Bhatt';
```

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

Using Character Manipulation Function

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
       job_id, LENGTH (last_name),
       INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PatFay	MK_REP	3	2

Number Functions

- ROUND: Rounds value to specified decimal
- TRUNC: Truncates value to specified decimal
- MOD: Returns remainder of division

Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

```
SELECT ROUND(45.923,2), ROUND(45.923,0),
       ROUND(45.923,-1)
FROM   DUAL;
```

Using TRUNC function

```
SELECT TRUNC(45.923,2), TRUNC(45.923),
       TRUNC(45.923,-1)
FROM   DUAL;
```

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-1)
45.92	45	40

Using the MOD Function

For all employees with the job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

Working with Dates

- The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.
- The default date display format is DD-MON-RR.
  - o Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
  - o Enables you to store 20th-century dates in the 21st century in the same way.
- **SYSDATE is a function that returns:**
  - o **Date**
  - o **Time**

ARITHMETIC WITH DATES

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24. Using Arithmetic operators with Date

Using Arithmetic operators with Date

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

Operation	Result	Description
Date + Number	Date	Adds a number of days to a date
Date - Number	Date	Subtracts a number of days from a date
Date - Date	Number of days	Subtracts one date from another
Date + Number / 24	Date	Adds a number of hours to a date

Date Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

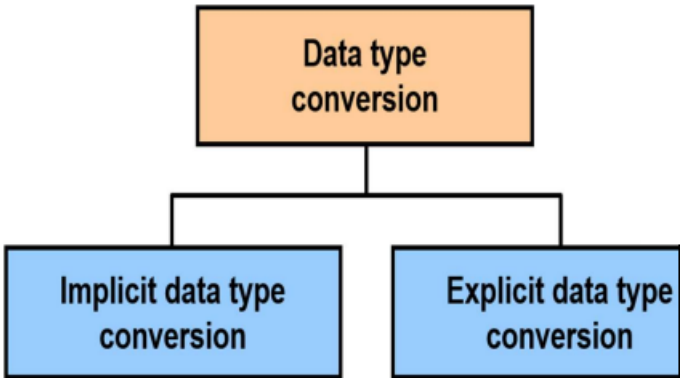
Using Date Function

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Assume SYSDATE = '25-JUL-03' :-

Function	Result
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

Conversion Function

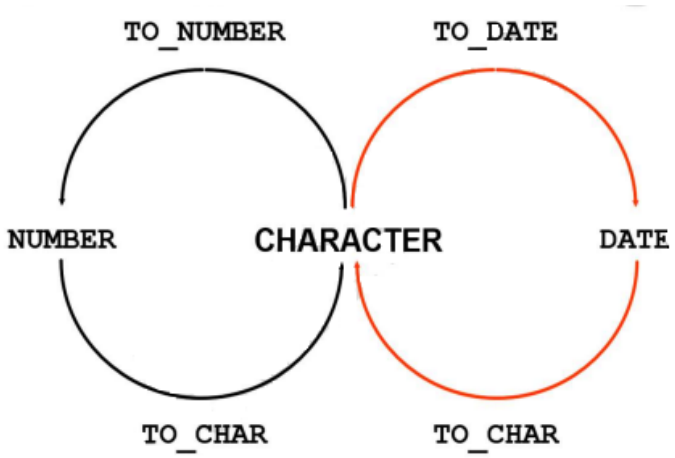


Implicit Data Type Conversion

For assignments, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

Explicit Data Type Conversion



Using the TO_CHAR Function with Dates

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed by single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros.
- Is separated from the date value by a comma.

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

Time elements format the time portion of the date

HH24:MI:SS AM	15:45:32 PM

Add character strings by enclosing them in double quotation marks

DD "of" MONTH	12 of OCTOBER

Number suffixes spell out numbers

ddspth	fourteenth

Using the TO_CHAR Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

Using the TO_CHAR Function with Numbers

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character:

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as thousands indicator

Using the TO_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

Using the TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the TO_NUMBER function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the TO_DATE function:

```
TO_DATE(char[, 'format_model'])
```

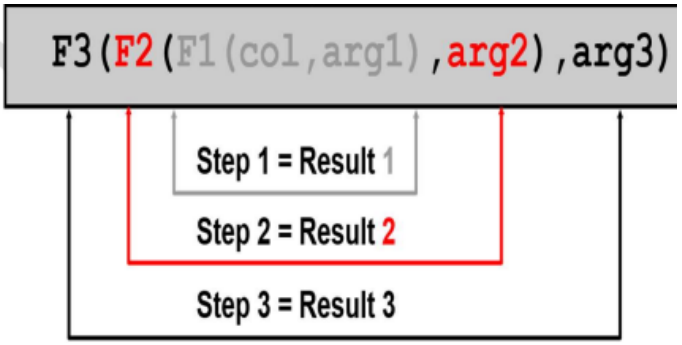
RR DATE FORMAT

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from deepest level to the least deep level.



Example

```
SELECT last name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```

General Functions

The following functions work with any data type and pertain to using nulls:

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- NULLIF (expr1, expr2)
- COALESCE (expr1, expr2, ..., exprn)

NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
- NVL(commission_pct,0)
- NVL(hire_date,'01-JAN-97')
- NVL(job_id,'No Job Yet')

NVL2 Function

- The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression.
- If the first expression is null, then the third expression is returned.

Syntax

NVL2(expr1, expr2, expr3)

In the syntax:

- expr1 is the source value or expression that may contain null
- expr2 is the value that is returned if expr1 is not null
- expr3 is the value that is returned if expr2 is null

NULLIF Function

- The NULLIF function compares two expressions.
- If they are equal, the function returns null.
- If they are not equal, the function returns the first expression.
- You cannot specify the literal NULL for the first expression.

Syntax

NULLIF (expr1, expr2)

In the syntax:

- expr1 is the source value compared to expr2.
- expr2 is the source value compared with expr1 (If it is not equal to expr1, expr1 is returned.)

COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.
- The COALESCE function returns the first non-null expression in the list.

Syntax

COALESCE (expr1, expr2, ... exprn)

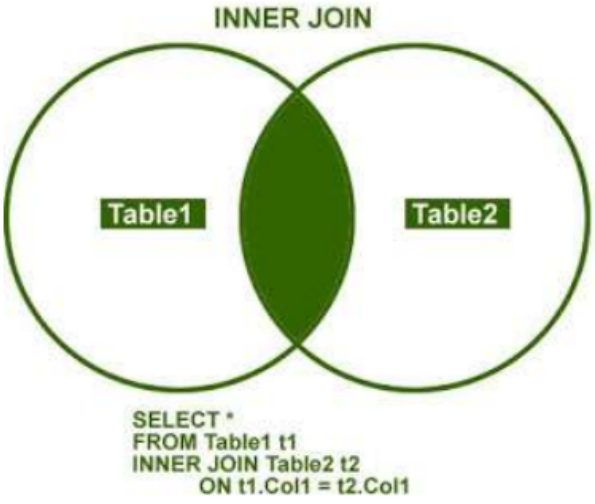
In the syntax:

- expr1 returns this expression if it is not null
- expr2 returns this expression if the first expression is null and this expression is not null
- exprn returns this expression if the preceding expressions are null

SQL JOIN

INNER JOIN

- This join returns rows when there is at least one match in both the tables.

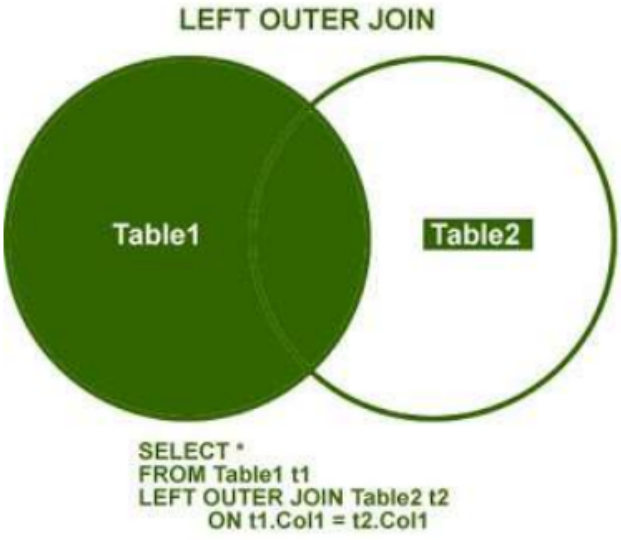


OUTER JOIN

- There are three different Outer Join methods.

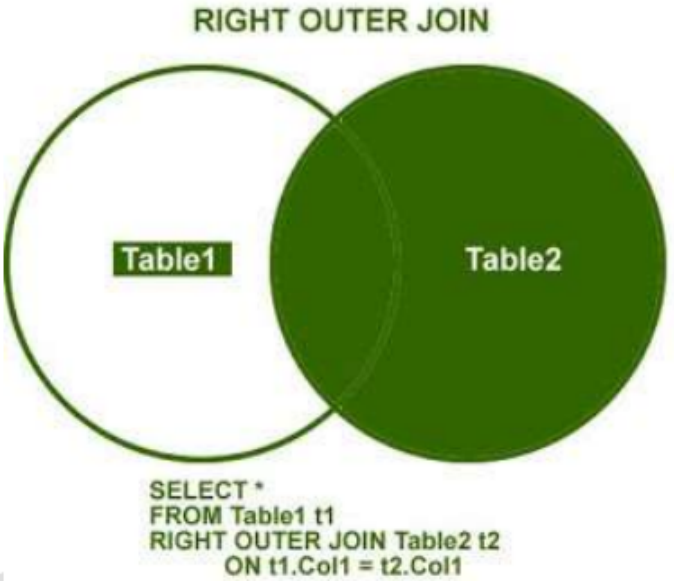
LEFT OUTER JOIN

- This join returns all the rows from the left table in conjunction with the matching rows from the right table.
- If there are no columns matching in the right table, it returns NULL values.



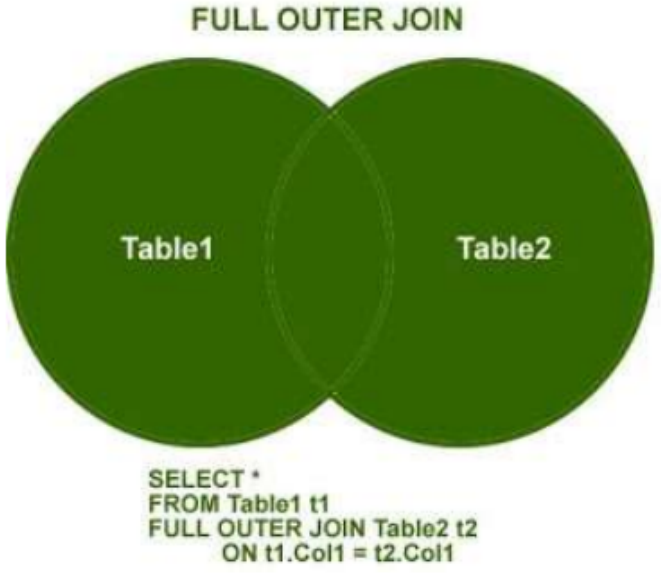
RIGHT OUTER JOIN

- This join returns all the rows from the right table in conjunction with the matching rows from the left table.
- If there are no columns matching in the left table, it returns NULL values.



FULL OUTER JOIN

- This join combines left outer join and right after join.
- It returns rows from either table when the conditions are met and returns null value when there is no match.



CROSS JOIN

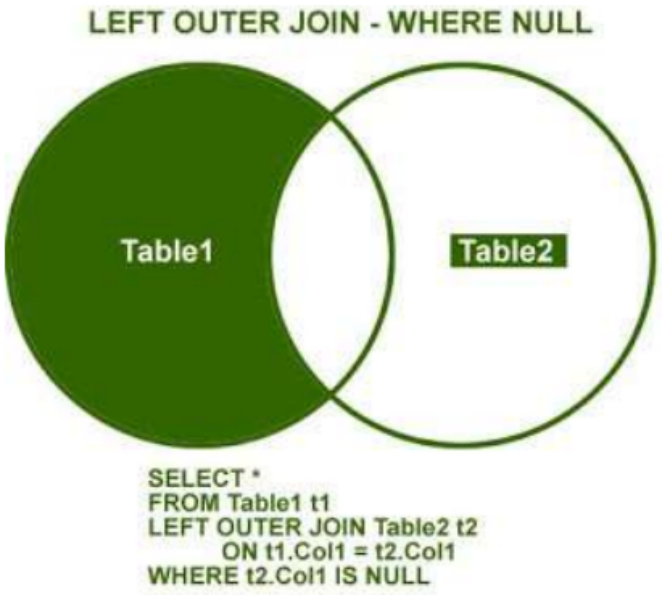
- This join is a Cartesian join that does not necessitate any condition to join.
- The result set contains records that are multiplications of record number from both the tables.

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	4	Fourth
5	5	Fifth

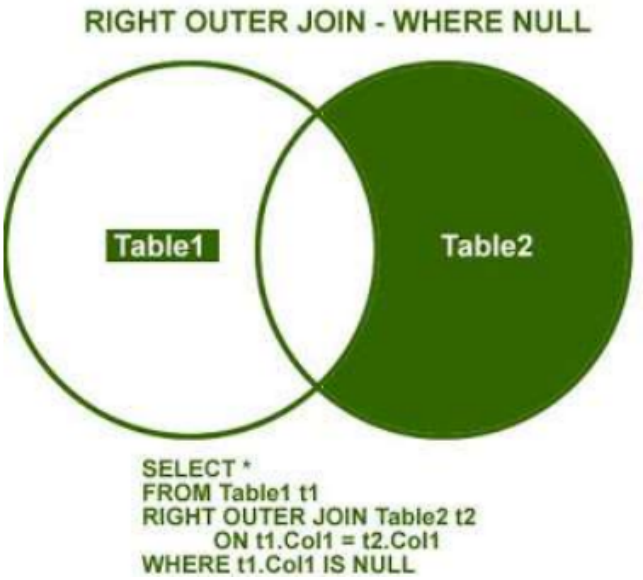
	ID	Value
1	1	First
2	2	Second
3	3	Third
4	6	Sixth
5	7	Seventh
6	8	Eighth

	ID	Value	ID	Value
1	1	First	1	First
2	1	First	2	Second
3	1	First	3	Third
4	1	First	6	Sixth
5	1	First	7	Seventh
6	1	First	8	Eighth
7	2	Second	1	First
8	2	Second	2	Second
9	2	Second	3	Third
10	2	Second	6	Sixth
11	2	Second	7	Seventh
12	2	Second	8	Eighth

LEFT JOIN – WHERE NULL

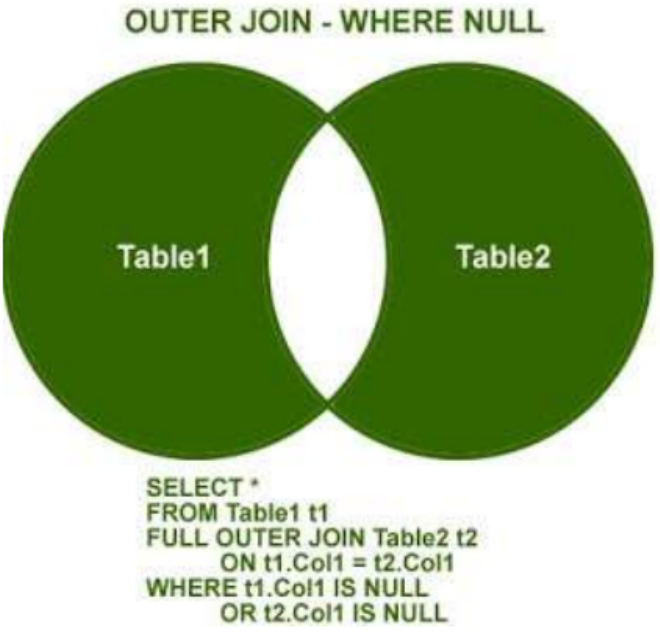


RIGHT JOIN – WHERE NULL



NOT INNER JOIN

- Remember, the term Not Inner Join does not exist in database terminology.
- However, when full Outer Join is used along with WHERE condition, as explained in the above two examples, it will give you an exclusive result to Inner Join.
- This join will give all the results that were not present in Inner Join.



SELF JOIN – USING ON CLAUSE

- **SELF JOIN** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Example - 1

SELECT e.last_name emp, m.last_name mgr FROM employees e JOIN employees m ON (e.manager_id = m.employee_id);

Example - 2

SELECT b.NAME, b.SALARY, a.SALARY FROM CUSTOMERS a, CUSTOMERS b WHERE a.SALARY < b.SALARY and a.NAME = 'RAM';

Retrieving Records with Non-Equijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.
- Do not use aliases on columns that are identified in the USING clause and listed elsewhere in the SQL statement.

Using Table Aliases

- Use table aliases to simplify queries.
- Use table aliases to improve performance.

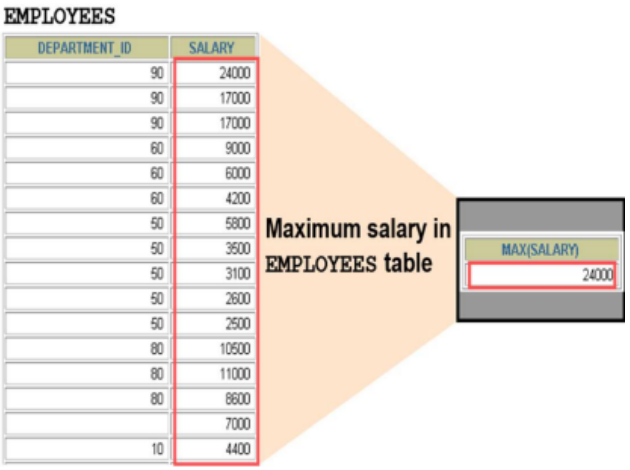
```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

GROUP FUNCTIONS

What Are Group Functions?

- Group functions operate on sets of rows to give one result per group.



Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM

Group Function Syntax

```
SELECT     [column,] group_function(column), ...
FROM       table
[WHERE     condition]
[GROUP BY  column]
[ORDER BY  column];
```

Using the AVG and SUM Functions

- You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

Using the MIN and MAX Functions

- You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM       employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

- If you need a last hired person you can use MAX, and if you need a first hired person you can use MIN function.

Using the COUNT Function

- COUNT(*) returns the number of rows in a table.

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

- COUNT(expr) returns the number of rows with non-null values for the expr

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

Using the DISTINCT Keyword

- COUNT(DISTINCT expr) returns the number of distinct non-null values of the expr.
- To display the number of distinct department values in the EMPLOYEES table.

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

Group Functions and Null Values

- Group functions ignore null values in the column

```
SELECT AVG(commission_pct)
FROM   employees;
```

- The NVL function forces group functions to include null values.

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

Creating Groups of Data

EMPLOYEES

DEPARTMENT_ID	SALARY	
10	4400	4400
20	13000	9500
20	6000	
50	5800	
50	3500	3500
50	3100	
50	2500	
50	2600	
60	9000	6400
60	6000	
60	4200	
80	10500	10033
80	8600	
80	11000	
90	24000	
90	17000	

Average salary in EMPLOYEES table for each department

DEPARTMENT_ID	AVG(SALARY
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT    column, group_function(column)
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

- You can divide rows in a table into smaller groups by using the GROUP BY clause.

Using the GROUP BY Clause

- All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

- The GROUP BY column does not have to be in the SELECT list.

```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```

Using the GROUP BY Clause on Multiple Columns

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;
```

Illegal Queries Using Group Functions

- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Column missing in the GROUP BY clause

- You cannot use the WHERE clause to restrict groups.
- You can use the HAVING clause to restrict groups.
- You cannot use group functions in the WHERE clause.

Restricting Group Results with HAVING clause

When you use the HAVING clause, the Oracle server restricts groups as follows:

- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

Using the HAVING Clause
Example - 1

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

Example - 2

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

Nesting Group Functions

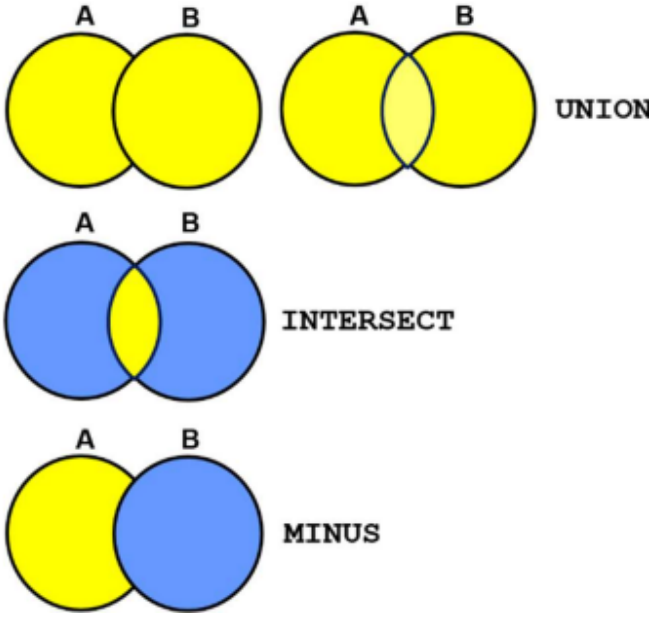
Display the maximum average salary

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

Where Vs Having

Where	Having
It is used for restricting rows from tables.	It is used for restricting rows which are Grouped.
We cannot use the aggregate function in where clause.	We can use aggregate functions in having clauses.
No need to use Group by clause when we are using Where clause.	We have to use Group by Clause When we are using Having Clause.
Where can be used to filter rows from tables only.	Having can be used to filter groups as well table rows.
Example SELECT ENAME,SALARY FROM EMP WHERE SALARY>10000	Example SELECT DEPTNO,SUM(SALARY) FROM EMP GROUP BY DEPTNO HAVING SUM(SALARY)>10000 It returns all department whose total salary > 10000

SET OPERATORS



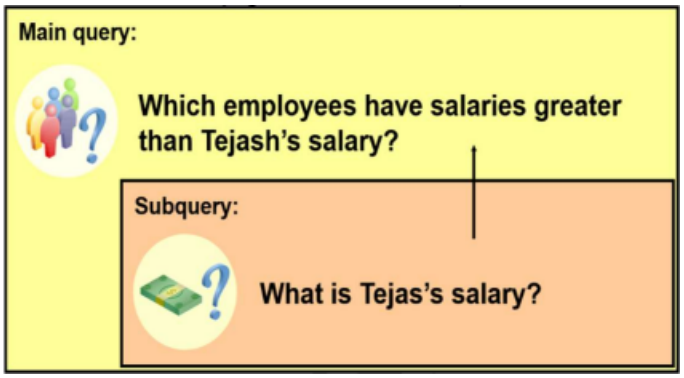
- The UNION operator returns results from both queries after eliminating duplicates.
- **SELECT EMPLOYEE_ID, JOB_ID FROM EMPLOYEES UNION SELECT EMPLOYEE_ID, JOB_ID FROM JOB_HISTORY;**
- Display the current and previous job details of all employees. Display each employee only once.
- The UNION ALL operator returns results from both queries, including all duplications.
- The INTERSECT operator returns rows that are common to both queries.
- The MINUS operator returns rows in the first query that are not present in the second query.

SET OPERATOR GUIDELINES

- The expressions in the SELECT lists must match in number and data type.
- Column names from the first query appear in the result.
- The output is sorted in ascending order by default except in UNION ALL.
- The ORDER BY clause:
  - o Can appear only at the very end of the statement
  - o Will accept the column name, aliases from the first SELECT statement, or the positional notation

SUB-QUERY

Using Subquery to solve a problem. Who has a salary greater than John's?



Syntax

```
SELECT   select_list
FROM     table
WHERE    expr operator
                (SELECT   select_list
                FROM      table);
```

- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query

Using SubQuery

```
SELECT last_name
FROM   employees
WHERE  salary >
               (SELECT salary
                FROM   employees
                WHERE  last_name = 'Tejash');
```

11000

Guidelines for using SubQuery

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

Types of SubQuery

- Single Row subquery
- Multiple Row subquery

Single Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Group Function in Subquery

```
SELECT last_name, job_id, salary
FROM   employees                    2500
WHERE  salary =
                  (SELECT MIN(salary)
                   FROM   employees);
```

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                  (SELECT   MIN(salary)
                   FROM     employees
                   GROUP BY department_id);
```

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Single-row operator with multiple-row subquery

Multiple Row Subquery

- When your inner query returns more than one row.
- Use multiple-row comparison operators.

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

- **<ANY** means less than Maximum
- **>ANY** means greater than Minimum
- **>ALL** means greater than Maximum
- **<ALL** means less than Minimum

Using ANY

```
SELECT employee_id, last_name, job_id, salary
FROM   employees          9000, 6000, 4200
WHERE  salary < ANY
                  (SELECT salary
                   FROM   employees
                   WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

Using ALL

```
SELECT employee_id, last_name, job_id, salary
FROM   employees          9000, 6000, 4200
WHERE  salary < ALL
                  (SELECT salary
                   FROM   employees
                   WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

DATABASE OBJECTS

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

View

- You can present logical subsets or combinations of data by creating views of tables.
- A view is a logical table based on a table or another view.
- A view contains no data of its own but is like a window through which data from tables can be viewed or changed.

- The tables on which a view is based are called base tables.
- The view is stored as a SELECT statement in the data dictionary.

Advantages of Views

- Views restrict access to the data because the view can display selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries.
- For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

SImple Vs Complex View

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

Creating A View

- You embed a subquery in the CREATE VIEW statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

The subquery can contain complex SELECT syntax.

- Create the EMPVU80 view, which contains details of employees in department 80:

```
CREATE VIEW  empvu80
 AS SELECT  employee_id, last_name, salary
    FROM    employees
    WHERE   department_id = 80;
View created.
```

Describe the structure of the view by using the iSQL*Plus DESCRIBE command:

```
CREATE VIEW  salvu50
 AS SELECT  employee_id ID_NUMBER, last_name NAME,
            salary*12 ANN_SALARY
    FROM    employees
    WHERE   department_id = 50;
View created.
```

Retrieving Data from a View

```
SELECT *
FROM   salvu50;
```

Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
            MAX(e.salary),AVG(e.salary)
   FROM     employees e JOIN departments d
   ON       (e.department_id = d.department_id)
   GROUP BY d.department_name;
View created.
```

Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views.
- You cannot remove a row if the view contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
- You cannot modify data in a view if it contains:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
  - Columns defined by expressions
- You cannot add data through a view if the view includes:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
  - Columns defined by expressions
  - NOT NULL columns in the base tables that are not selected by the view

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
View dropped.
```

SEQUENCES

A sequence:

- Can automatically generate unique numbers Is a shareable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

CREATE SEQUENCE Statement

Syntax

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

Creating a Sequence

- Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.
- Do not use the CYCLE option.

```
CREATE SEQUENCE dept_deptid_seq
               INCREMENT BY 10
               START WITH 120
               MAXVALUE 9999
               NOCACHE
               NOCYCLE;
Sequence created.
```

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value.
- It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Using a Sequence

- Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 row created.
```

- View the current value for the DEPT_DEPTID_SEQ sequence:

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
Sequence altered.
```

Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the DROP statement:

```
DROP SEQUENCE dept_deptid_seq;
Sequence dropped.
```