YENEPOYA
(DEEMED TO BE UNIVERSITY)
Recognised under Sec 3(A) of the UGC Act 1956
Accredited by NAAC with 'A' Grade

Structured Query Language

Unit 4

sindhusandesh@yenepoya.edu.in

Structured Query Language

- **Introduction to SQL:** Data Definition Commands, Data Manipulation Commands, Select queries, Advanced Data Definition Commands, Advanced Select queries, Virtual Tables, Joining Database Tables.
- **Advanced SQL:** Relational Set Operators, SQL Join Operators, Subqueries and correlated queries, SQL Functions, Oracle Sequences, and Procedural SQL.

About SQL

- **Oracle** is a relational database technology developed by Oracle.
- **SQL** is a database computer language designed for the retrieval and management of data in a relational database.
- SQL is a standard language for accessing and manipulating databases.
- **SQL** stands for **Structured Query Language**.

About SQL

- PLSQL stands for "Procedural Language extensions to SQL", and is an extension of SQL that is used in Oracle.
- PLSQL is closely integrated into the SQL language, yet it adds programming constructs that are not native to SQL.

- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System.
- All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Key Aspects

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

A Brief History of SQL

- **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** − Structured Query Language appeared.
- **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

Applications of SQL

  - SQL is one of the most widely used query language over the databases.
- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

- 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

MySQL

- MySQL is an open source SQL database, which is developed by a Swedish company – MySQL AB.
- MySQL is pronounced as "my ess-que-ell," in contrast with SQL, pronounced "sequel."
- MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.
- MySQL has free and paid versions, depending on its usage (non- commercial/commercial) and features.
- MySQL comes with a very fast, multi-threaded, multi-user and robust SQL database server.

History

- Development of MySQL by Michael Widenius & David Axmark beginning in 1994.
- First internal release on 23rd May 1995.
- Windows Version was released on the 8th January 1998 for Windows 95 and NT.
- Version 3.23: beta from June 2000, production release January 2001.
- Version 4.0: beta from August 2002, production release March 2003 (unions).
- Version 4.1: beta from June 2004, production release October 2004.
- Version 5.0: beta from March 2005, production release October 2005.
- Sun Microsystems acquired MySQL AB on the 26th February 2008.
- Version 5.1: production release 27th November 2008.

Features

- High Performance.
- High Availability.
- Scalability and Flexibility Run anything.
- Robust Transactional Support.
- Web and Data Warehouse Strengths.
- Strong Data Protection.
- Comprehensive Application Development.
- Management Ease.
- Open Source Freedom and 24 x 7 Support.
- Lowest Total Cost of Ownership.

MS SQL Server

- MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are −
- T-SQL
- ANSI SQL

History

- 1987 - Sybase releases SQL Server for UNIX.
- 1988 - Microsoft, Sybase, and Aston-Tate port SQL Server to OS/2.
- 1989 - Microsoft, Sybase, and Aston-Tate release SQL Server 1.0 for OS/2.
- 1990 - SQL Server 1.1 is released with support for Windows 3.0 clients.
- Aston - Tate drops out of SQL Server development.
- 2000 - Microsoft releases SQL Server 2000.
- 2001 - Microsoft releases XML for SQL Server Web Release 1 (download).
- 2002 - Microsoft releases SQLXML 2.0 (renamed from XML for SQL Server).

- 2002 - Microsoft releases SQLXML 3.0.
- 2005 - Microsoft releases SQL Server 2005 on November 7th, 2005.

Features

- High Performance
- High Availability
- Database mirroring
- Database snapshots
- CLR integration
- Service Broker
- DDL triggers
- Ranking functions
- Row version-based isolation levels
- XML integration
- TRY...CATCH
- Database Mail

ORACLE

- It is a very large multi-user based database management system.
- Oracle is a relational database management system developed by 'Oracle Corporation'.
- Oracle works to efficiently manage its resources, a database of information among the multiple clients requesting and sending data in the network.
- It is an excellent database server choice for client/server computing.
- Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

History

- Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009).
- 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work.
- 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first
  SQL database. The company changed its name to Relational Software Inc. (RSI).
- 1981 - RSI started developing tools for Oracle.
- 1982 - RSI was renamed to Oracle Corporation.
- 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms.
- 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 2007 - Oracle released Oracle11g. The new version focused on better partitioning, easy migration, etc.

Features

- Concurrency
- Read Consistency
- Locking Mechanisms
- Quiesce Database
- Portability
- Self-managing database
- SQL*Plus
- ASM
- Scheduler
- Resource Manager
- Data Warehousing
- Materialized views
- Bitmap indexes
- Table compression
- Parallel Execution
- Analytic SQL
- Data mining
- Partitioning

MS ACCESS

- This is one of the most popular Microsoft products. Microsoft Access is an entry-level database management software.
- MS Access database is not only inexpensive but also a powerful database for small-scale projects.
- MS Access uses the Jet database engine, which utilizes a specific SQL language dialect (sometimes referred to as Jet SQL).
- MS Access comes with the professional edition of MS Office package. MS Access has easyto-use intuitive

graphical interface.

History

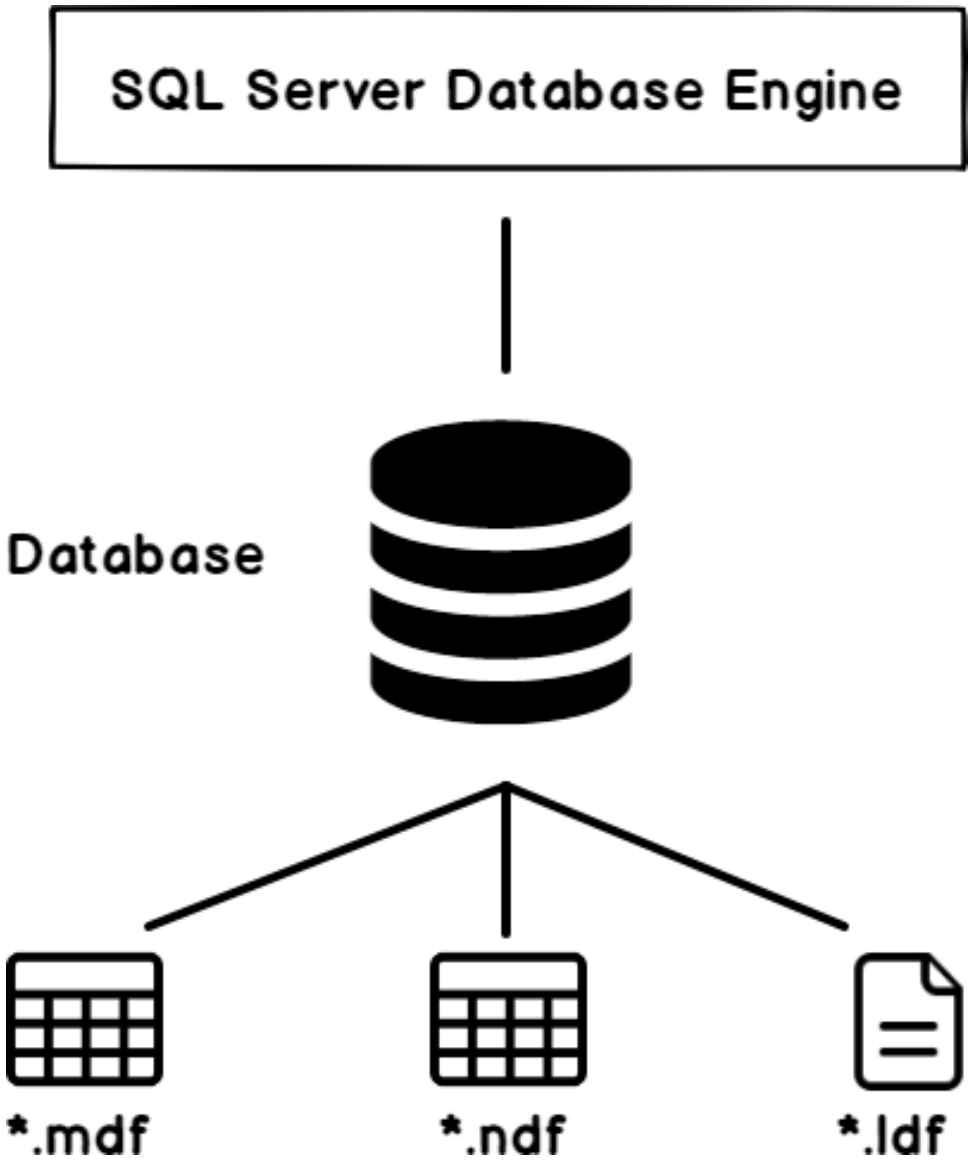
- 1992 - Access version 1.0 was released.
- 1993 - Access 1.1 released to improve compatibility with inclusion the Access Basic programming language.
- The most significant transition was from Access 97 to Access 2000.
- 2007 - Access 2007, a new database format was introduced ACCDB which supports complex data types such as multi valued and attachment fields.

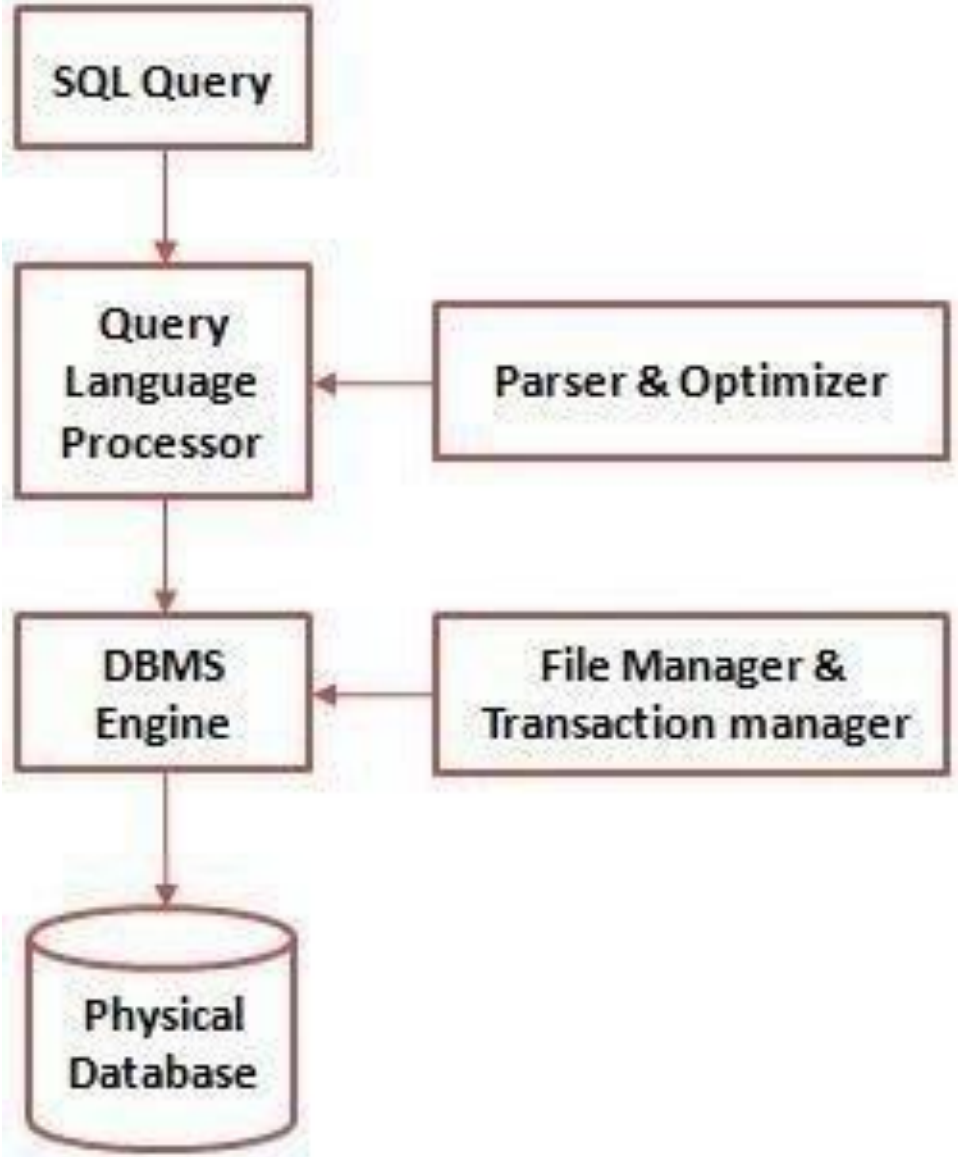
Features

- Users can create tables, queries, forms and reports and connect them together with macros.
- Option of importing and exporting the data to many formats including Excel, Outlook, ASCII, dBase,
  Paradox, FoxPro, SQL Server, Oracle, ODBC, etc.
- There is also the Jet Database format (MDB or ACCDB in Access 2007), which can contain the application and data in one file. This makes it very convenient to distribute the entire application to another user, who can run it in disconnected environments.
- Microsoft Access offers parameterized queries. These queries and Access tables can be referenced from other programs like VB6 and .NET through DAO or ADO.
- The desktop editions of Microsoft SQL Server can be used with Access as an alternative to the Jet Database Engine.
- Microsoft Access is a file server-based database. Unlike the client-server relational database management systems (RDBMS), Microsoft Access does not implement database triggers, stored procedures or transaction logging.

SQL Execution

- When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.
- There are various components included in this process.
  - Query Dispatcher
  - Optimization Engines
  - Classic Query Engine
  - SQL Query Engine, etc.
- A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.


SQL Server Database Engine
Database
*.mdf
*.ndf
*.ldf


SQL Query
Query Language Processor
Parser & Optimizer
DBMS Engine
File Manager & Transaction manager
Physical Database

Note

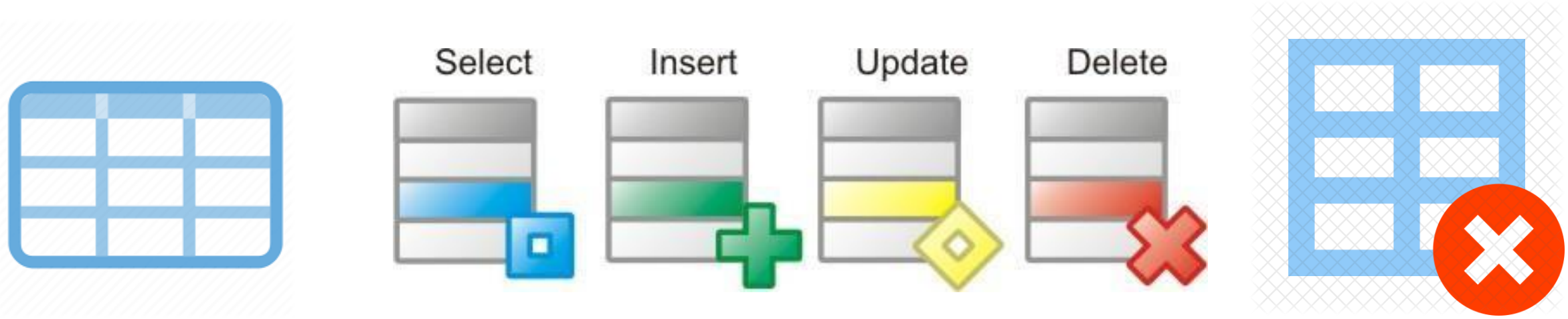
- Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

- However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

SQL Commands

- The standard SQL commands to interact with relational databases

CREATE, SELECT, INSERT, UPDATE, DELETE and DROP.



SQL Commands

Data Definition Commands, Data Manipulation Commands, Select queries

- DDL - Data Definition Language
- DML - Data Manipulation Language
- DCL - Data Control Language
- Data Control Language (DCL)

- Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- Data Manipulation Language (DML) - These SQL commands are used for storing, retrieving, modifying, and deleting data. These Data Manipulation Language commands are: SELECT, INSERT, UPDATE, and DELETE.

- Transaction Control Language (TCL) - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- Data Control Language (DCL) - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

DDL - Data Definition Language

- **CREATE**
- Creates a new table, a view of a table, or other object in the database.
- **ALTER**
- Modifies an existing database object, such as a table.
- **DROP**
- Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

- **SELECT**
- Retrieves certain records from one or more tables.
- **INSERT**
- Creates a record.
- **UPDATE**
- Modifies records.
- **DELETE**
- Deletes records.

DCL - Data Control Language

- **GRANT**
- Gives a privilege to user.
- **REVOKE**
- Takes back privileges granted from user.

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

…a Table

- The data in an RDBMS is stored in database objects which are called as **tables**.
- This table is basically a collection of related data entries and it consists of numerous columns and rows.
  - the most common and simplest form of data storage in a relational database

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXX	19

ID	NAME	AGE	ADDRESS	SALARY
101	Hardik	32	Mumbai	35000
102	Akash	25	Delhi	25000
103	Manoj	23	Bhopal	27000
104	Deepa	25	Kochin	19000
105	Komal	20	Mangalor	26000

a field

- Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
- A field is a column in a table that is designed to maintain specific information about every record in the table.

I

ADDRE

a Record or a Row

- A record is also called as a row of data is each individual entry that exists in a table.

102	Akash	25	Delhi	25000

- A record is a horizontal entity in a table.

a column

- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

ADDRESS
Mumbai
Delhi
Bhopal
Kochin

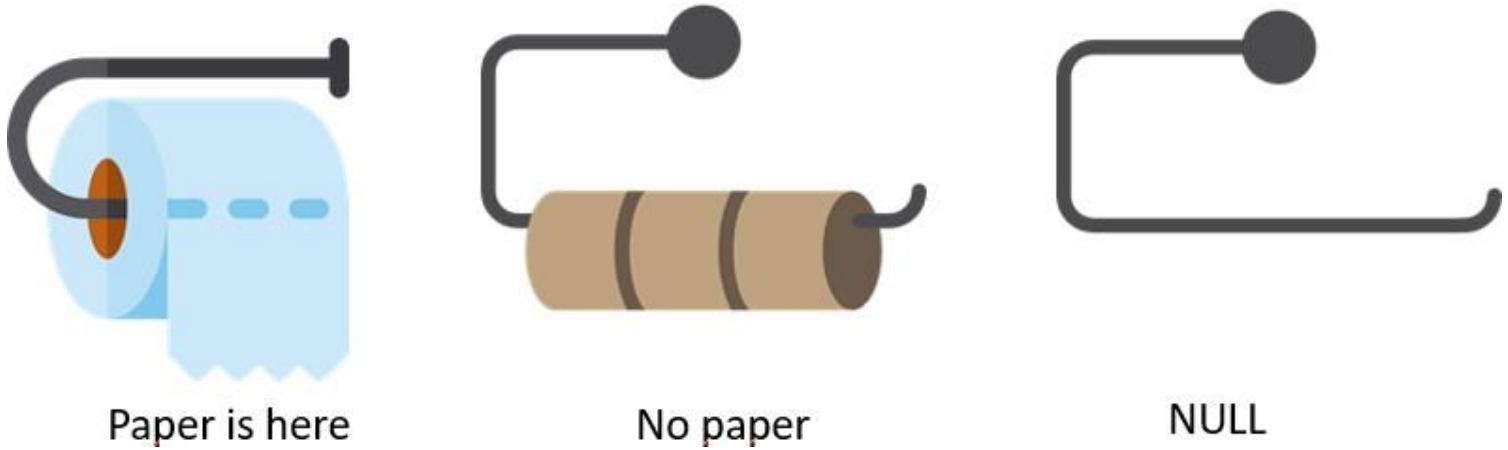
Mangalor

a NULL value

- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

- A NULL value is different than a zero value or a field that contains spaces.
- A field with a NULL value is the one that has been left blank during a record creation.

NULL vs ZERO



- NOT KNOWN
- NOT APPLICABLE

Meaning of NULL

- Value unknown (value exists but is not known)
- Value not available (exists but is purposely withheld)
- Attribute not applicable (undefined for this tuple)

- It is often not possible to determine which of the meanings is intended in a SQL Table

Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

SQL Constraints

- Constraints are the rules enforced on data columns on a table.
- These are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the database.

- Constraints can either be column level or table level.
- Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

SQL Integrity Constraints

- NOT NULL Constraint
  - Ensures that a column cannot have a NULL value.
- DEFAULT Constraint
  - Provides a default value for a column when none is specified.
- UNIQUE Constraint
  - Ensures that all the values in a column are different.

SQL Integrity Constraints

- PRIMARY Key
  - Uniquely identifies each row/record in a database table.
- FOREIGN Key
  - Uniquely identifies a row/record in any another database table.
- CHECK Constraint
  - The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX
  - Used to create and retrieve data from the database very quickly.

NOT NULL Constraint

- By default, a column can hold NULL values.
- If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.
- A NULL is not the same as no data, rather, it represents unknown data.

```
CREATE TABLE CUSTOMERS
(
   ID   NUMBER  NOT NULL,
   NAME VARCHAR (20)  NOT NULL,
   AGE  NUMBER   NOT NULL,
   ADDRESS  CHAR (25),
   SALARY   NUMBER(18, 2),
   PRIMARY KEY (ID)
);
```

```
ALTER TABLE CUSTOMERS
        MODIFY SALARY  NUMBER (18, 2) NOT NULL;
```

DEFAULT Constraint

- The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

```
CREATE TABLE
  CUSTOMERS(
ID  INT          NOT NULL,
  NAME VARCHAR (20)    NOT
  NULL, AGE     INT      NOT
  NULL, ADDRESS      CHAR
  (25) ,
  SALARY  DECIMAL (18, 2) DEFAULT
  5000.00, PRIMARY KEY (ID)
```

```
);
```

- ALTER TABLE CUSTOMERS
- MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;

UNIQUE Constraint

- The UNIQUE Constraint prevents two records from having identical values in a column.
- In the CUSTOMERS table, for example, you might want to prevent two or more people from having an identical age

```
CREATE TABLE
  CUSTOMERS(
ID  INT          NOT NULL,
  NAME VARCHAR (20)    NOT
  NULL, AGE              INT
       UNIQUE,
  ADDRESS  CHAR (25)
  , SALARY   DECIMAL
  (18, 2), PRIMARY KEY
```

```
    (ID)
);
```

```
ALTER TABLE
  CUSTOMERS MODIFY
  AGE INT UNIQUE;
;
```

Primary Key

- A primary key is a field in a table which uniquely identifies each row/record in a database table.
- Primary keys must contain unique values. A primary key column cannot have NULL values.
- A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, **they are called a composite key.**
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of

that field(s).

```
CREATE TABLE
  CUSTOMERS(
ID  INT          NOT NULL,
  NAME VARCHAR (20)    NOT
  NULL, AGE     INT      NOT
  NULL, ADDRESS      CHAR
  (25) ,
  SALARY  DECIMAL
  (18, 2), PRIMARY KEY
```

```
    (ID)
);
```

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

```
CREATE TABLE
  CUSTOMERS(
ID  INT           NOT NULL,
  NAME VARCHAR (20)    NOT
  NULL, AGE     INT      NOT
  NULL, ADDRESS      CHAR
  (25) ,
  SALARY  DECIMAL
  (18, 2), PRIMARY KEY
```

```
    (ID, NAME)
);
```

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID,
  NAME);
```

Foreign Key

- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that

field(s).

STUDENT

studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

Foreign Keys

Relationship

Primary Keys

COURSE

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(
  CID INT          NOT NULL,
  NAME VARCHAR (20)  NOT
  NULL, AGE     INT        NOT
  NULL, ADDRESS      CHAR
  (25)
  PRIMARY KEY (CID)
);
```

ORDERS table

```
CREATE TABLE ORDERS (
  OID       INT      NOT
  NULL, DATE
            DATETIME,
  CUSTOMER_ID INT
        references CUSTOMERS(CID),
  AMOUNT
                double
  , PRIMARY KEY
  (OID)
);
```

```
ALTER TABLE ORDERS
  ADD FOREIGN KEY (Customer_ID)
                REFERENCES
                CUSTOMERS (ID);
```

```
ALTER TABLE
  ORDERS DROP
```

FOREIGN KEY;

CHECK Constraint

- The CHECK Constraint enables a condition to check the value being entered into a record.
- If the condition evaluates to false, the record violates the constraint and isn't entered the table

```
CREATE TABLE CUSTOMERS(
   ID  INT           NOT NULL,
   NAME VARCHAR (20)    NOT NULL,
   AGEINT  NOT NULL CHECK (AGE
   >= 18), ADDRESS  CHAR (25) ,
   SALARY  DECIMAL
   (18, 2), PRIMARY KEY
   (ID)
);
```

```
CREATE TABLE
    Persons ( ID int NOT
    NULL,
    LastName varchar(255) NOT
    NULL, FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

```
ALTER TABLE CUSTOMERS
  MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );

ALTER TABLE CUSTOMERS
  ADD CONSTRAINT myCheckConstraint CHECK(AGE >=
  18);
```

INDEX Constraint

- The INDEX is used to create and retrieve data from the database very quickly.
- An Index can be created by using a single or group of columns in a table.
- When the index is created, it is assigned a ROWID for each row before it sorts out the data.
- Proper indexes are good for performance in large databases, but you need to be careful while creating an index.
- A Selection of fields depends on what you are using in your SQL queries.

```
CREATE TABLE CUSTOMERS(
        ID  INT  NOT NULL,
        NAME VARCHAR (20)  NOT
        NULL, AGE                  INT  NOT
        NULL,
        ADDRESS CHAR (25) ,
        SALARY DECIMAL
        (18, 2), PRIMARY KEY
        (ID)
```

```
);
```

- To create an index on a single or multiple columns

```
CREATE INDEX index_name
    ON table_name ( column1, column2 );
```

- To create an INDEX on the AGE column, to optimize the search on customers for a specific age,

```
CREATE INDEX idx_age
    ON CUSTOMERS ( AGE );
```

```
ALTER TABLE CUSTOMERS
        DROP INDEX idx_age;
```

Data Integrity

- The following categories of data integrity exist with each RDBMS −
- **Entity Integrity −** There are no duplicate rows in a table.
- **Domain Integrity −** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity −** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity −** Enforces some specific business rules that do not fall into entity, domain or

referential integrity

DDL - Data Definition Language

- **CREATE**
- Creates a new table, a view of a table, or other object in the database.
- **ALTER**
- Modifies an existing database object, such as a table.
- **DROP**
- Deletes an entire table, a view of a table or other objects in the database.

SQL CREATE TABLE Statement

- The CREATE TABLE Statement is used to create tables to store data.
- Integrity Constraints like primary key, unique key, foreign key can be defined for the columns while creating the table.
- The integrity constraints can be defined at column level or table level.
- The implementation and the syntax of the CREATE Statements differs for different RDBMS

Syntax for the CREATE TABLE

```
CREATE TABLE table_name
(column_name1 datatype(size),
column_name2 datatype,
... column_nameN datatype
);
```

table_name - is the name of the table.
column_name1, column_name2.... - is the name of the columns datatype - is the datatype for the column like char,

date, number etc.

```
CREATE TABLE employee
( id number(5),
name char(20),
dept char(10),
age number(2),
salary
number(10),
location char(10)
);
```

- In Oracle database, the datatype for an integer column is represented as "number".
- In Sybase it is represented as "int".
- Oracle provides another way of creating a table.

```
CREATE TABLE
temp_employee AS SELECT *
FROM employee
```

SQL ALTER TABLE Statement

- The SQL ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns.
- The ALTER command is used to perform the following functions.
  - Add, drop, modify table columns
  - Add and drop constraints
  - Enable and Disable constraints

Syntax to add a column

ALTER TABLE table_name
ADD column_name
datatype;

- To add a column "experience" to the employee table, the query would be like

ALTER TABLE employee
ADD experience

number(3);

Syntax to drop a column

ALTER TABLE table_name
DROP COLUMN
column_name;

- To drop the column "location" from the employee table, the query would be like

ALTER TABLE
employee DROP

COLUMN location;

Syntax to modify a column

ALTER TABLE table_name
MODIFY column_name
datatype;

- To modify the column salary in the employee table, the query would be like

ALTER TABLE employee
MODIFY salary

number(15,2);

SQL RENAME Command

- The SQL RENAME command is used to change the name of the table or a database object.

- If you change the object's name any reference to the old name will be affected.
- You have to manually change the old name to the new name in every reference.

Syntax to rename a table

RENAME
old_table_name To
new_table_name;

- To change the name of the table employee to my_employee, the query would be like

RENAME employee
TO my_emloyee;

The SQL DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.

DROP TABLE

table_name; DROP

TABLE student;

SQL TRUNCATE TABLE

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

- TRUNCATE TABLE table_name;

- TRUNCATE TABLE student;

Data Manipulation Language

- It is a language used for selecting, inserting, deleting and updating data in a database.
- It is used to retrieve and manipulate data in a relational database.

- **DML commands are as follows,**
  1. SELECT
  2. INSERT
  3. UPDATE
  4. DELETE

SELECT COMMAND

- **SELECT command** is used to retrieve data from the database.
- This command allows database users to retrieve the specific information they desire from an operational database.
- It returns a result set of records from one or more tables.

- The most commonly used SQL command is SELECT statement.
- SQL SELECT statement is used to query or retrieve data from a table in the database.
- A query may retrieve information from specified columns or from all of the columns in the table.
- To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name.
- The whole query is called SQL SELECT Statement.

Syntax of SQL SELECT Statement

- SELECT column_list FROM table-name
- [WHERE Clause]
- [GROUP BY clause]
- [HAVING clause]
- [ORDER BY clause];

table-name is the name of the table from which the information is retrieved.
column_list includes one or more columns from which data is retrieved. The code within the brackets is optional.

SELECT Command has many optional clauses

Clause	Description
WHERE	It specifies which rows to retrieve.
GROUP BY	It is used to arrange the data into groups.
HAVING	It selects among the groups defined by the GROUP BY clause.
ORDER BY	It specifies an order in which to return the rows.

AS	It provides an alias which can be used to temporarily rename tables or columns.

- **Syntax:**
  SELECT * FROM <table_name>;

- Eg
- SELECT * FROM employee;

- SELECT * FROM
  employee where salary
  >=10,000;

- SELECT column1, column2, columnN
- FROM table_name
- WHERE [condition]

- SELECT column1, column2, columnN
- FROM table_name
- WHERE [condition1] AND [condition2]...AND [conditionN];

Group By

- The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

- SELECT column1, column2
- FROM table_name
- WHERE [conditions]
- GROUP BY column1, column2
- ORDER BY column1, column2

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

- SELECT NAME, SUM(SALARY) FROM CUSTOMERS
- GROUP BY NAME;

Order By

```
SELECT column-
list FROM
table_name
[WHERE
condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

- SELECT * FROM CUSTOMERS
- ORDER BY NAME, SALARY;

SQL - Having Clause

- The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

- SELECT ID, NAME, AGE, ADDRESS, SALARY
- FROM CUSTOMERS
- GROUP BY age
- HAVING COUNT(age) >= 2;

- IN,
- NOT IN,
- BETWEEN

LIKE Clause

- The SQL LIKE clause is used to compare a value to similar values using wildcard operators.
- There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)

- SELECT FROM table_name
- WHERE column LIKE 'XXXX%'

- or

- SELECT FROM table_name
- WHERE column LIKE '%XXXX%'

- or

- SELECT FROM table_name
- WHERE column LIKE 'XXXX_'

- or

- SELECT FROM table_name
- WHERE column LIKE '_XXXX'

- or

- SELECT FROM table_name
- WHERE column LIKE '_XXXX_'

WHERE SALARY LIKE '200%' Finds any values that start with 200.
WHERE SALARY LIKE '%200%' Finds any values that have 200 in any position.
WHERE SALARY LIKE '_00%' Finds any values that have 00 in the second and third positions.
WHERE SALARY LIKE '2_%_%' Finds any values that start with 2 and are at least 3 characters in length.
WHERE SALARY LIKE '%2' Finds any values that end with 2.
WHERE SALARY LIKE '_2%3' Finds any values that have a 2 in the second position and end with a 3.
WHERE SALARY LIKE '2_3' Finds any values in a five-digit number that start with 2 and end with 3.

TOP, LIMIT or ROWNUM Clause

- The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table
- SELECT TOP number|percent column_name(s)
- FROM table_name
- WHERE [condition]

- SELECT TOP 3 * FROM CUSTOMERS;

- SELECT * FROM CUSTOMERS
- LIMIT 3;

- SELECT * FROM CUSTOMERS
- WHERE ROWNUM <= 3;

Distinct Keyword

- The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.
- There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.

- SELECT DISTINCT column1, column2,. columnN
- FROM table_name
- WHERE [condition]

- SELECT DISTINCT SALARY FROM CUSTOMERS
- ORDER BY SALARY;

Update

- The SQL **UPDATE** Query is used to modify the existing records in a table.
- You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

- UPDATE table_name
- SET column1 = value1, column2 = value2...., columnN = valueN
- WHERE [condition];

- UPDATE CUSTOMERS
- SET ADDRESS = 'Pune'
- WHERE ID = 6;

- UPDATE CUSTOMERS
- SET ADDRESS = 'Pune', SALARY = 1000.00;

SQL - INSERT Query

- The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

- INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
- VALUES (value1, value2, value3,...valueN);

- INSERT INTO TABLE_NAME
  VALUES
  (value1,value2,value3,...valueN)
  ;

- INSERT INTO CUSTOMERS
- VALUES (7, 'Muffy', 24, 'Indore', 10000.00);

Populate one table using another table

```
INSERT INTO first_table_name [(column1, column2, ...
  columnN)] SELECT column1, column2, ...columnN
  FROM second_table_name
  [WHERE condition];
```

DELETE Query

- The SQL DELETE Query is used to delete the existing records from a table.
- You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

- DELETE FROM table_name
- WHERE [condition];

- DELETE FROM CUSTOMERS
- WHERE ID = 6;

- DELETE FROM CUSTOMERS;

SQL Functions

- SQL Aggregate Functions
- SQL aggregate functions return a single value, **calculated from values in a column.**
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Functions

- SQL Scalar functions
- SQL scalar functions return a single value, based on the input value.
- UPPER() - Converts a field to upper case
- LOWER() - Converts a field to lower case
- LENGTH() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- FORMAT() - Formats how a field is to be displayed

- SELECT UPPER(last_name) "Uppercase"
- FROM employees;

Selecting from the DUAL Table

- DUAL is a table automatically created by Oracle Database along with the data dictionary. DUAL is in the schema of the user SYS but is accessible by the name DUAL to all users.
- It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement.
- Because DUAL has only one row, the constant is returned only once. Alternatively, you can select a constant, pseudocolumn, or expression from any table, but the value will be returned as many times as there are rows in the table.

Date Functions

- CURRENT_DATE
- DAY(date)

SQL - Using Joins

- The SQL **Joins** clause is used to combine records from two or more tables in a database.
- A JOIN is a means for combining fields from two tables by using values common to each.

```
SELECT ID, NAME, AGE,
 AMOUNT FROM
 CUSTOMERS, ORDERS
 WHERE    CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Types of Join

- INNER JOIN − returns rows when there is a match in both tables.
- LEFT JOIN − returns all rows from the left table, even if there are no matches in the
  right table.
- RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN − returns rows when there is a match in one of the tables.
- 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.
- CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two

or more joined tables.

INNER JOINS

- The most important and frequently used of the joins is the **INNER JOIN**.
- They are also referred to as an **EQUIJOIN**.
- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join- predicate.
- The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.
- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

```
SELECT table1.column1,
table2.column2... FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

EMP

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEPT

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana
13	Accounts	Bhavish

select empid, empname, deptname from emp
inner join dept
on **emp.deptno=dept.deptno**

EMPID	EMPNAME	DEPTNAME
12345	Akash	Sales
12346	Akshay	Sales
12547	Bhavana	HR
12548	Bhavya	Accounts
12568	Bhavish	Accounts
12346	Adarsh	Sales
12346	Anand	Sales

- Question: List the manager of each employee…..

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana
13	Accounts	Bhavish

```
select empname, mgrname from emp,dept
where emp.deptno=dept.deptno;
```

```
select empname, mgrname
from emp
inner join
dept
on emp.deptno=dept.deptno;
```

EMP

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEPT2

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana

```
select empid, empname,
deptname
from emp
inner join dept2
on emp.deptno=dept2.deptno
```

EMPID	EMPNAME	DEPTNAME
12345	Akash	Sales
12346	Akshay	Sales
12547	Bhavana	HR
12346	Adarsh	Sales
12346	Anand	Sales

LEFT JOINS

- The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

- SELECT table1.column1, table2.column2...
- FROM table1
- LEFT JOIN table2
- ON table1.common_field = table2.common_field;

EMP

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEP

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana

```
select empid, empname, deptname
from emp
left join dept2
on emp.deptno=dept2.deptno
```

EMPID	EMPNAME	DEPTNAME
12346	Anand	Sales
12346	Adarsh	Sales
12346	Akshay	Sales
12345	Akash	Sales
12547	Bhavana	HR
12568	Bhavish	-
12548	Bhavya	-

RIGHT JOINS

- The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

```
SELECT table1.column1,
table2.column2... FROM table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEPT3

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana
14	Accounts	Bhavish

```
select empid, empname,
deptname from emp
right join dept3
on emp.deptno=dept3.deptno
```

EMPID	EMPNAME	DEPTNAME
12345	Akash	Sales
12346	Akshay	Sales
12547	Bhavana	HR
12346	Adarsh	Sales
12346	Anand	Sales
-	-	Accounts

FULL JOINS

- The SQL **FULL JOIN** combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

```
SELECT table1.column1,
table2.column2... FROM table1
FULL JOIN table2
ON table1.common_field = table2.common_field;
```

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akash	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	18500
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400

DEPTNO	DEPTNAME	MGRNAME
10	Sales	Akshay
11	HR	Bhavana
14	Accounts	Bhavish

```
select empid, empname, deptname
from emp
full join dept3
on emp.deptno=dept3.deptno
```

EMPID	EMPNAME	DEPTNAME
12346	Anand	Sales
12346	Adarsh	Sales
12346	Akshay	Sales
12345	Akash	Sales
12547	Bhavana	HR
12568	Bhavish	-
12548	Bhavya	-
-	-	Accounts

SELF JOINS

- The SQL **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.

- SELECT a.column_name, b.column_name...
- FROM table1 a, table1 b
- WHERE a.common_field = b.common_field;

EMPLOYEEID	NAME	MANAGERID
101	Mary	102
102	Ravi	NULL
103	Raj	102
104	Pete	103
105	Prasad	103
106	Ben	103

EMPLOYE

- List the names of employees and their Managers
- **NAME** **MANAGER NAME**
- MARY RAVI
- RAVI -
- RAJ RAVI
- PETE RAJ
- PRASAD RAJ
- BEN RAJ

ENAME	ENAME
RAJ	RAVI
RAVI	RAVI
Mary	RAVI
BEN	RAJ
PRASAD	RAJ
PETER	RAJ

- CREATE TABLE MANAGER
- AS SELECT * FROM EMP;

    - ANOTHER TABLE CALLED MANAGER WILL BE CREATED!!!!

EMPLOYEEID	NAME	MANAGERID
101	Mary	102
102	Ravi	NULL
103	Raj	102
104	Pete	103
105	Prasad	103
106	Ben	103

EMPLOYEEID	NAME	MANAGERID
101	Mary	102
102	Ravi	NULL
103	Raj	102
104	Pete	103
105	Prasad	103
106	Ben	103

EMP

EMPLOYEEID	NAME	MANAGERID
101	Mary	102
102	Ravi	NULL
103	Raj	102
104	Pete	103
105	Prasad	103
106	Ben	103

MANAGER

- select emp.ename,manager.ename
- from employee emp,employee manager
- where emp.mgrid=manager.eid

ENAME	ENAME
RAJ	RAVI
RAVI	RAVI
Mary	RAVI
BEN	RAJ
PRASAD	RAJ
PETER	RAJ

CARTESIAN or CROSS JOINS

- The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

- SELECT table1.column1, table2.column2...
- FROMtable1, table2 [, table3]

Company

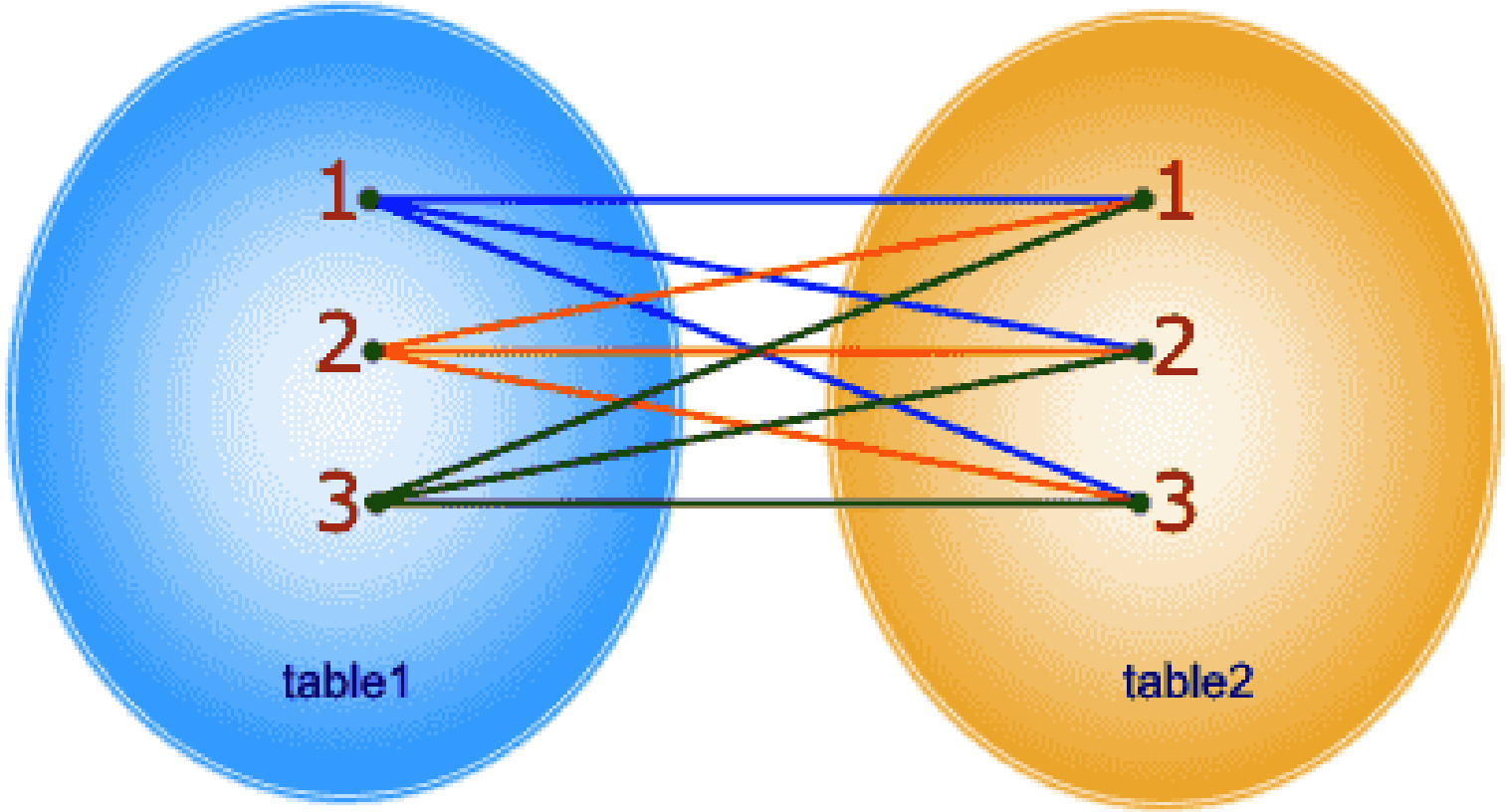
COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

Foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

SQL Cross Join

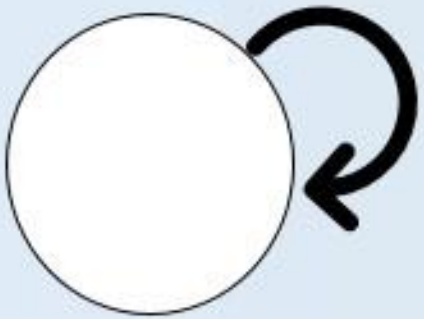
SELECT * FROM table1 CROSS JOIN table2;



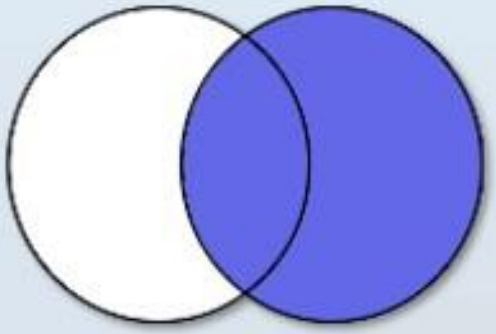
In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.

Joins in SQL

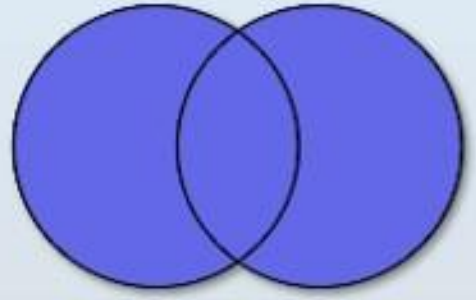
Self Join



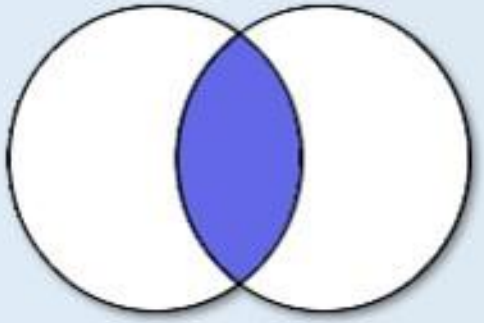
Right Join



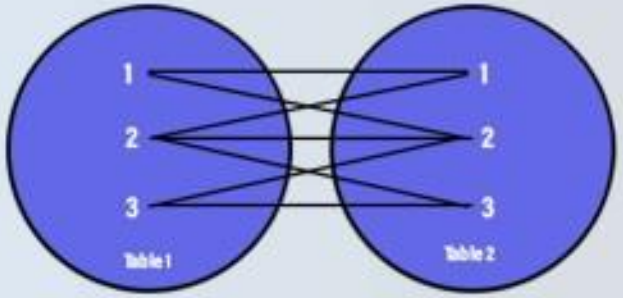
Full Join



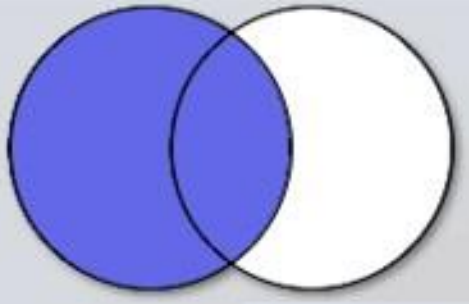
Inner Join



Cross Join



Left Join



Views (Virtual Tables) in SQL

- A view is nothing more than a SQL statement that is stored in the database with an associated name.
- A view is actually a composition of a table in the form of a predefined SQL query.
- A view can contain all rows of a table or select rows from a table.
- A view can be created from one or many tables which depends on the written SQL query to create a view.

- Database Administrator and Database Users will face two challenges: writing complex SQL queries and **securing database access.**
- Sometimes SQL queries become more complicated due to the use of multiple joins, subqueries, and GROUP BY in a single query.
- To simplify such queries, you can use some proxy over the original table.
- Also, Sometimes from the security side, the database administrator wants to restrict direct access to the database.

- Views, which are a type of virtual tables allow users to do the following −
- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

- Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

CREATE VIEW *view_name* **AS**

SELECT column1, column2.....

FROM table_name

WHERE [condition];

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name,
age FROM
        CUSTOME
RS;
```

- SELECT * FROM CUSTOMERS_VIEW;

Updating a View

- A view can be updated under certain conditions which are given below −
- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

```
UPDATE
CUSTOMERS_VIEW
SET AGE = 35
WHERE name = 'Ramesh';
```

Inserting Rows into a View

- Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Deleting Rows into a View

- Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
- DELETE FROM CUSTOMERS_VIEW
- WHERE age = 22;

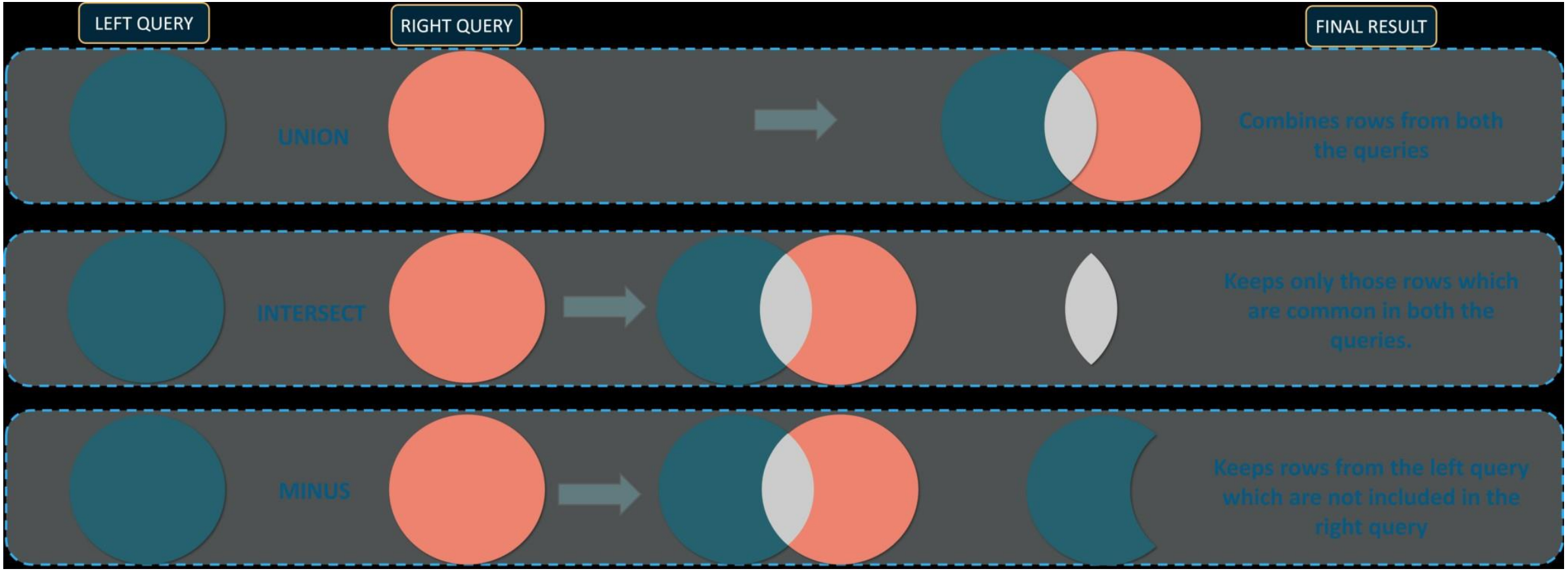
Dropping Views

- DROP VIEW view_name;

- DROP VIEW CUSTOMERS_VIEW;

Relational Set Operators in DBMS

- DBMS supports relational set operators as well. The major relational set operators are union, intersection and set difference.
- All of these can be implemented in DBMS using different queries.


LEFT QUERY
RIGHT QUERY
FINAL RESULT
UNION
Combines rows from both the queries
INTERSECT
Keeps only those rows which are common in both the queries.
MINUS
Keeps rows from the left query which are not included in the right query

Union

- Union combines two different results obtained by a query into a single result in the form of a table.
- However, the results should be similar if union is to be applied on them.
- Union removes all duplicates, if any from the data and only displays distinct values.
- If duplicate values are required in the resultant data, then UNION ALL is used.

- Select Student_Name from Art_Students
- UNION
- Select Student_Name from Dance_Students

- The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

- To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order
- But they need not have to be in the same length.

Syntax of UNION

SELECT column1 [,
column2] FROM table1 [,
table2] [WHERE
condition]

UNION

SELECT column1 [,
column2] FROM table1 [,
table2] [WHERE
condition]

```
SQL> SELECT    ID, NAME, AMOUNT, DATE
  FROM
  CUSTOMERS
  LEFT JOIN
  ORDERS
  ON CUSTOMERS.ID =

ORDERS.CUSTOMER_ID UNION

  SELECT   ID, NAME,
  AMOUNT, DATE FROM
  CUSTOMERS
```

```
RIGHT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

The UNION ALL Clause

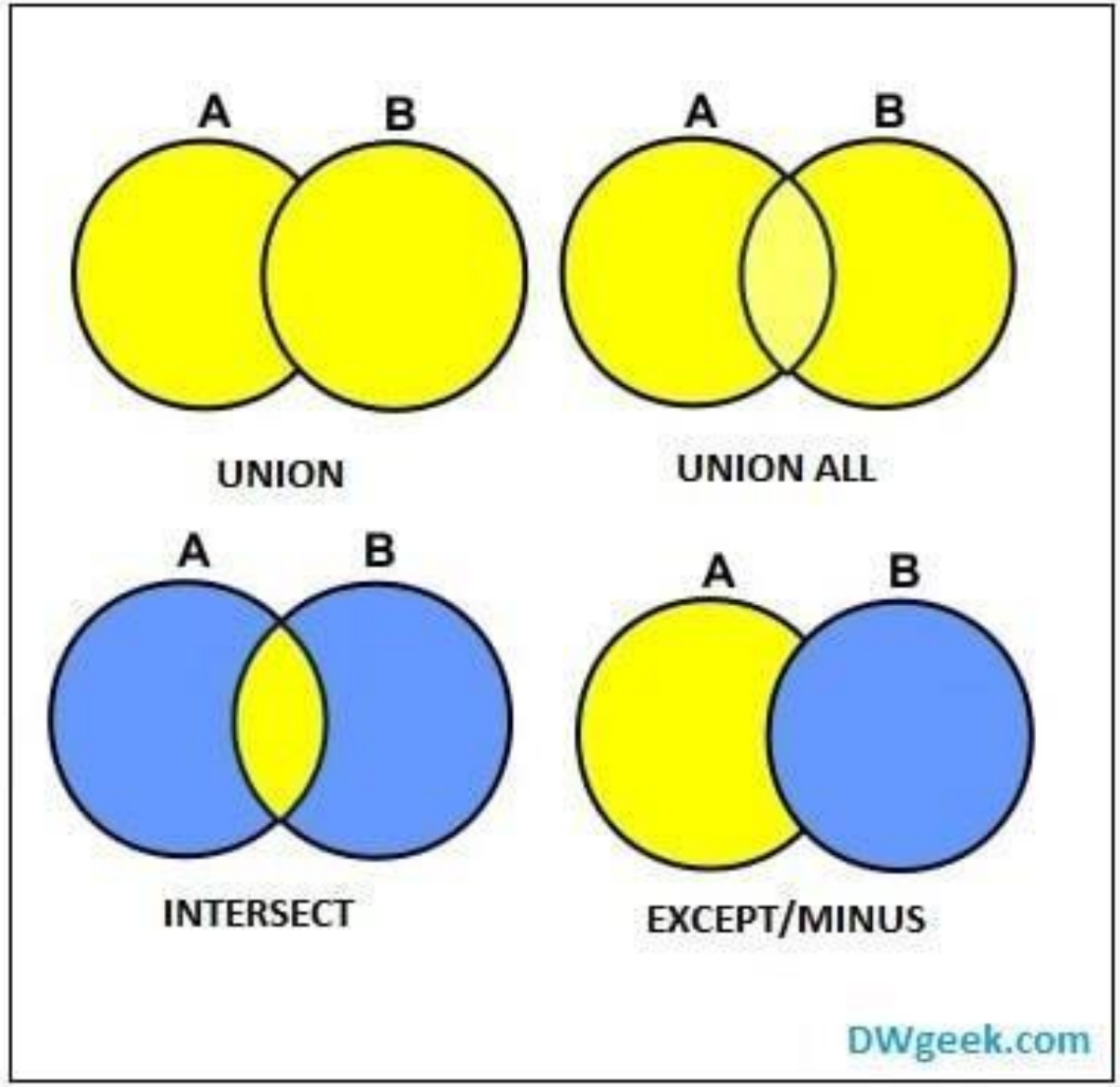
- The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

- The same rules that apply to the UNION clause will apply to the UNION ALL operator.

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

UNION ALL

SELECT column1 [,
column2] FROM table1 [,
table2] [WHERE
condition]



Intersection

- The intersection operator gives the common data values between the two data sets that are intersected.
- The two data sets that are intersected should be similar for the intersection operator to work.
- Intersection also removes all duplicates before displaying the result.

- Select Student_Name from Art_Students
- INTERSECT
- Select Student_Name from Dance_Students

Set difference

- The set difference operators takes the two sets and returns the values that are in the first set but not the second set.

- Select Student_Name from Art_Students
- MINUS
- Select Student_Name from Dance_Students

NAME
Akashkrishna
Akshay
Bhavana
Bhavya
Bhavish
Adarsh
Anand
Manoj

MINUS

FIRSTNAME
Adarsh
Arathi
Akhay
Anand

NAME
Akashkrishna
Akshay
Bhavana
Bhavish
Bhavya
Manoj

SQL - Sub Queries

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the

operators like =, <, >, >=, <=, IN, BETWEEN, etc.

- SB_Accounts : Details of SB Account Customers & their Balance in SB Account
- Loan_Account : Details of Customers who have taken loan
- I want to know…
- The Saving Bank Balance of customers whose **loan is due.**

```
Select Balance from SB_Accounts
Where Custumerid IN (Select customerid
                     from Loan_Account
                     Where
                     loanstatus='Due');
```

EMP

EMPID	EMPNAME	DEPTNO	SALARY
12345	Akashkrishna	10	25200
12346	Akshay	10	30450
12547	Bhavana	11	28000
12548	Bhavya	13	20350
12568	Bhavish	13	19500
12346	Adarsh	10	26250
12346	Anand	10	29400
1	Manoj	-	-

LOAN

EMPID	LOAN_AMT
12346	5000
12345	15000
12548	7000
14569	40000

- List the employee details who have taken loan

```
Select * from emp1
where empid in( select empid from loan)
```

```
Select * from emp1
where empid in(select empid from loan where loan_amt >5000)
```

Rules

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement

```
SELECT column_name [,
column_name ] FROM  table1 [,
table2 ]
WHERE column_name  OPERATOR
   (SELECT column_name [,
   column_name ] FROM table1 [,
   table2 ]
   [WHERE])
```

```
SELECT *
 FROM CUSTOMERS
 WHERE ID IN
 (SELECT ID
      FROM CUSTOMERS
      WHERE SALARY >
      4500) ;
```

Subqueries with the INSERT Statement

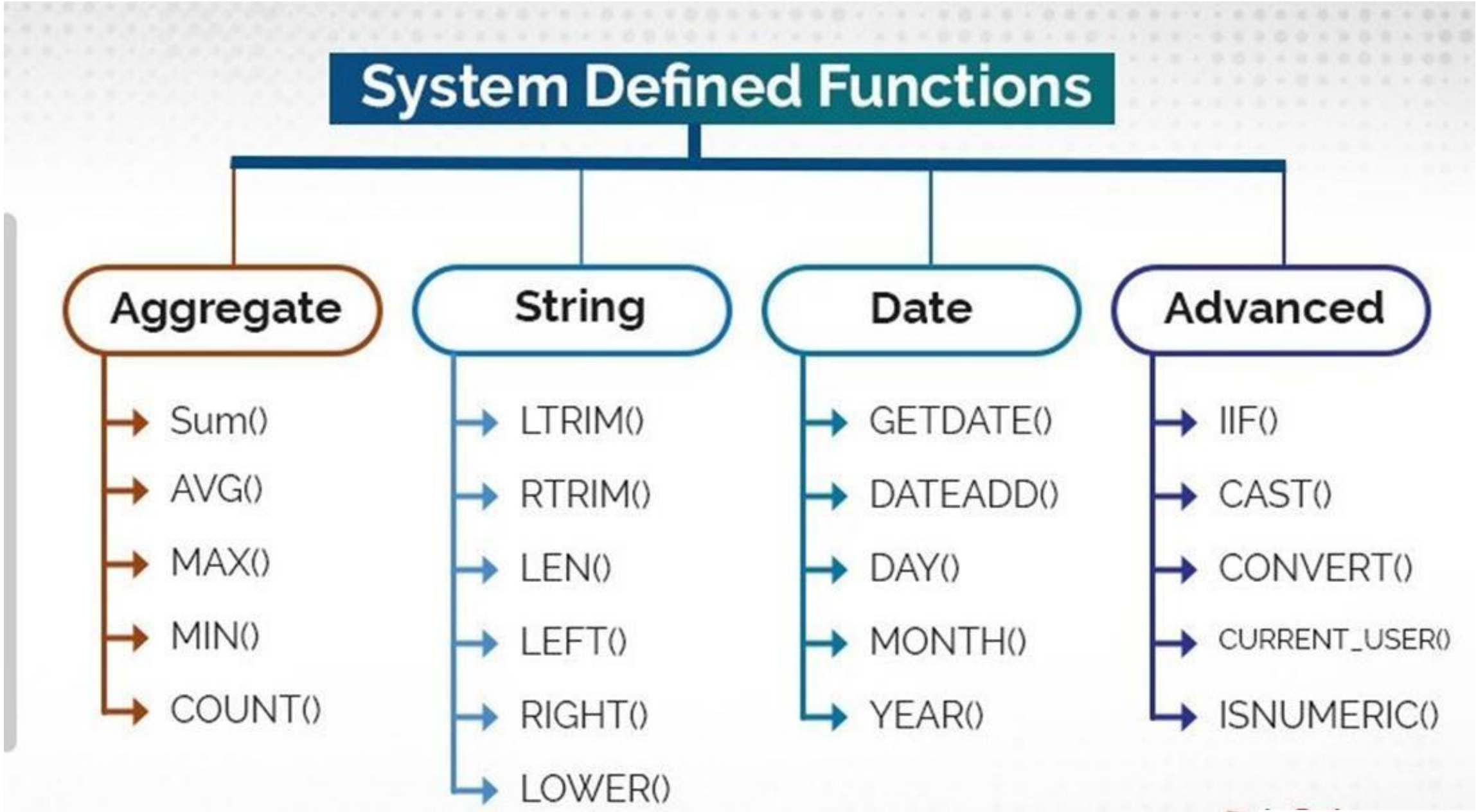
- The INSERT statement uses the data returned from the subquery to insert into another table.
- The selected data in the subquery can be modified with any of the character, date or number functions.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
  SELECT [ *|column1 [, column2 ]
  FROM table1 [, table2 ]
  [ WHERE VALUE OPERATOR ]
```

```
INSERT INTO CUSTOMERS_BKP
  SELECT * FROM
  CUSTOMERS WHERE ID
  IN (SELECT ID FROM
  CUSTOMERS) ;

UPDATE CUSTOMERS
  SET SALARY = SALARY * 0.25
  WHERE AGE IN (SELECT AGE FROM
    CUSTOMERS_BKP WHERE AGE >= 27 );

 DELETE FROM CUSTOMERS
  WHERE AGE IN (SELECT AGE FROM
    CUSTOMERS_BKP WHERE AGE >= 27 );
```



Sequences (Autonumber)

- A sequence is an object in Oracle that is used to generate a number sequence.
- This can be useful when you need to create a unique number to act as a primary key.

```
CREATE SEQUENCE
  sequence_name MINVALUE
  value
  MAXVALUE value
  START WITH value
  INCREMENT BY
  value CACHE value;
```

```
CREATE SEQUENCE
  supplier_seq MINVALUE 1
  MAXVALUE 9999999999999999999999999
  START WITH 1
  INCREMENT
  BY 1
  CACHE 20;
```

This would create a sequence object called supplier_seq. The first sequence number that it would use is 1 and each subsequent number would increment by 1 (ie: 2,3,4,...}. It will cache up to 20 values for performance.

If you omit the MAXVALUE option, your sequence will automatically default to:

MAXVALUE 9999999999999999999999999

- To retrieve the next value in the sequence order, you need to use *nextval*.
- supplier_seq.NEXTVAL;

```
INSERT INTO suppliers
(supplier_id,
supplier_name) VALUES
(supplier_seq.NEXTVAL, 'Kraft Foods');
```

Drop Sequence

- DROP SEQUENCE sequence_name;
- DROP SEQUENCE supplier_seq;

```
CREATE SEQUENCE
  c_seq MINVALUE 1
  MAXVALUE 99
  START WITH 1
  INCREMENT BY 1
  CACHE 20;
```

- INSERT INTO emp
- (empid, empname)
- VALUES
- (c_seq.NEXTVAL, 'Manoj');

Procedural SQL : PL/SQL

- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Features of PL/SQL

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.

- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object- oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

Basic Syntax

- Declarations
- This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
- Executable Commands
- This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
- Exception Handling
- This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in

the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

```
DECLARE
   message varchar2(20):= 'Hello,
World!'; BEGIN
   dbms_output.put_line(messag
e); END;
/
```

The end; line signals the end of the PL/SQL block. To run the code from the SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code.

PL/SQL - Data Types

- **Numeric**
- Numeric values on which arithmetic operations are performed.
- **Character**
- Alphanumeric values that represent single characters or strings of characters.
- **Boolean**
- Logical values on which logical operations are performed.
- **Datetime**
- Dates and times.

```
DECLARE
   num1 INTEGER;
   num2 REAL;
   num3 DOUBLE PRECISION;
BEGIN
   null;
END;
/
```

PL/SQL - Variables

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in PL/SQL has a specific data type, which determines the size and the layout of the variable's memory; the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

```
pi CONSTANT double precision := 3.1415;
```

PL/SQL - Operators

- Arithmetic operators
- Relational operators
- Comparison operators
- Logical operators
- String operators

+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give 5
*	Multiplies both operands	A * B will give 50
/	Divides numerator by de-numerator	A / B will give 2
**	Exponentiation operator, raises one operand to the power of other	A ** B will give 100000

=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A = B) is not true.
!= < > ~ =	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
> =	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

Comparison Operators

LIKE	The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern and FALSE if it does not.	If 'Zara Ali' like 'Z% A_i' returns a Boolean true, whereas, 'Nuha Ali' like 'Z% A_i' returns a Boolean false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a and x <= b.	If x = 10 then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x between 11 and 20 returns false.
IN	The IN operator tests set membership. x IN (set) means that x is equal to any member of set.	If x = 'm' then, x in ('a', 'b', 'c') returns Boolean false but x in ('m', 'n', 'o') returns Boolean true.
IS NULL	The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values always yield NULL.	If x = 'm', then 'x is null' returns Boolean false.

Logical Operators

Operator	Description	Examples
and	Called the logical AND operator. If both the operands are true then condition becomes true.	(A and B) is false.
or	Called the logical OR Operator. If any of the two operands is true then condition becomes true.	(A or B) is true.

not	Called the logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make it false.	not (A and B) is true.

PL/SQL - Conditions

S.No	Statement & Description
1	IF - THEN statementThe **IF statement** associates a condition with a sequence of statements enclosed by the keywords **THEN** and **END IF**. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.
2	IF-THEN-ELSE statement**IF statement** adds the keyword **ELSE** followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.
3	IF-THEN-ELSIF statementIt allows you to choose between several alternatives.

4	Case statementLike the IF statement, the **CASE statement** selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

- IF <condition> THEN

  ……

  ……

 …… END IF

- IF <condition> THEN

  ……

  ……

…… ELSE

  ……

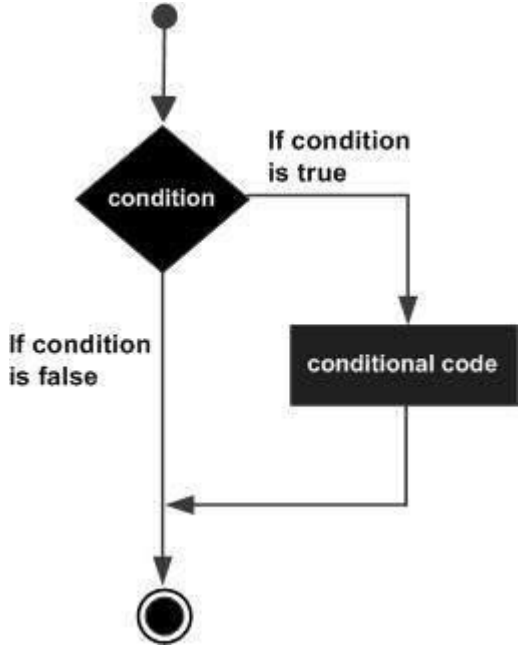
  ……

…

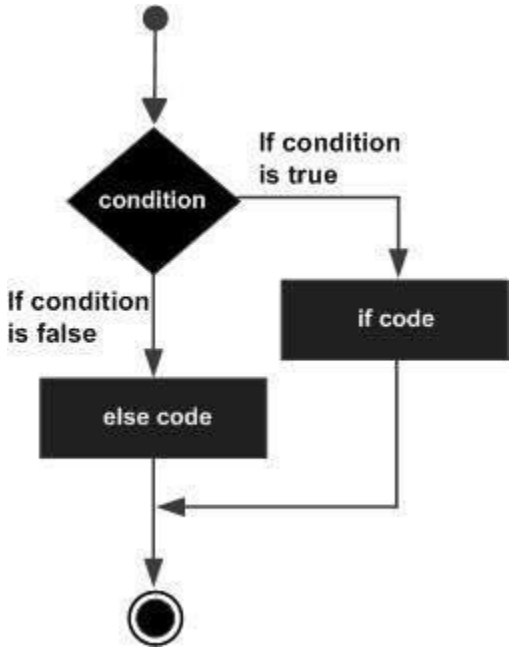
END

IF

```
DECLARE
    a number(2) := 10;
  BEGIN
    a:=
    10;
   -- check the boolean condition using if
    statement IF( a < 20 ) THEN
      -- if condition is true then print the following
      dbms_output.put_line('a is less than 20 ' );
  END IF;
    dbms_output.put_line('value of a is : ' || a);
  END;
  /
```



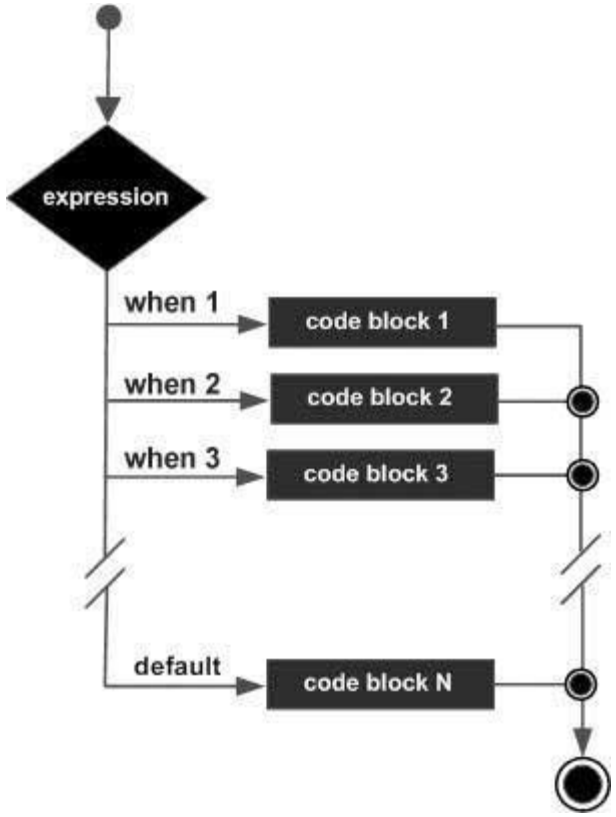
```
DECLARE
   a number(3) := 100;
BEGIN
   -- check the boolean condition using if statement
   IF( a < 20 ) THEN
      -- if condition is true then print the following
      dbms_output.put_line('a is less than 20 ' );
   ELSE
      dbms_output.put_line('a is not less than 20 ' );
   END IF;
   dbms_output.put_line('value of a is : ' || a);
END;
/
```



```
DECLARE
  a number(3) :=
100; BEGIN
  IF ( a = 10 ) THEN
    dbms_output.put_line('Value of a is 10' );
  ELSIF ( a = 20 ) THEN
    dbms_output.put_line('Value of a is 20' );
  ELSIF ( a = 30 ) THEN
    dbms_output.put_line('Value of a is
  30' ); ELSE
     dbms_output.put_line('None of the values is matching');
  END IF;
  dbms_output.put_line('Exact value of a is: '|| a
); END;
/
```

CASE Statement

- CASE selector
- WHEN 'value1' THEN S1;
- WHEN 'value2' THEN S2;
- WHEN 'value3' THEN S3;
- ...
- ELSE Sn; -- default case
- END CASE;



```
DECLARE
  grade char(1) := 'A';
BEGIN
  CASE grade
    when 'A' then
    dbms_output.put_line('Excellent'); when 'B'
    then dbms_output.put_line('Very good'); when
    'C' then dbms_output.put_line('Well done');
    when 'D' then dbms_output.put_line('You
    passed');
    when 'F' then dbms_output.put_line('Better try
    again'); else dbms_output.put_line('No such grade');
  END CASE;
END;
```

/

PL/SQL - Loops

1	PL/SQL Basic LOOPIn this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.
2	PL/SQL WHILE LOOPRepeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	PL/SQL FOR LOOPExecute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
4	Nested loops in PL/SQLYou can use one or more loop inside any another basic loop, while, or for

	loop.

- LOOP
- ……..
- ……
- IF(condition) THEN
- EXIT
- END IF
- END LOOP

Loop

- LOOP
- Sequence of statements;
- END LOOP;

```
DECLARE
  x number :=
10; BEGIN
  LOOP
    dbms_output.put_line(x
    ); x := x + 10;
    IF x > 50 THEN
      exit;
     END
      IF;
     END
    LOOP;
  -- after exit, control resumes here
  dbms_output.put_line('After Exit x is: ' || x);
END;
```

/

```
DECLARE
  x number := 10;
BEGIN
  LOOP
    dbms_output.put_line(x);
    x := x + 10;
    exit WHEN x > 50;
  END LOOP;
  -- after exit, control resumes here
  dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

WHILE LOOP Statement

- **WHILE condition** LOOP
- sequence_of_statements
- END LOOP;

```
DECLARE
   a number(2) := 10;
BEGIN
   WHILE a < 20 LOOP
      dbms_output.put_line('value of a: ' || a);
      a := a + 1;
   END LOOP;
END;
/
```

FOR LOOP

- FOR counter IN initial_value .. final_value LOOP
- sequence_of_statements;
- END LOOP;

The initial step is executed first, and only once. This step allows you to declare and initialize any loop control variables.

Next, the condition, i.e., initial_value .. final_value is evaluated. If it is TRUE, the body of the loop is executed. If it is FALSE, the body of the loop does not execute and the flow of control jumps to the next statement just after the for loop.

After the body of the for loop executes, the value of the counter variable is increased or decreased.

The condition is now evaluated again. If it is TRUE, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the

FOR LOOP

```
DECLARE
   a number(2);
BEGIN
   FOR a in 10 .. 20 LOOP
      dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
/
```

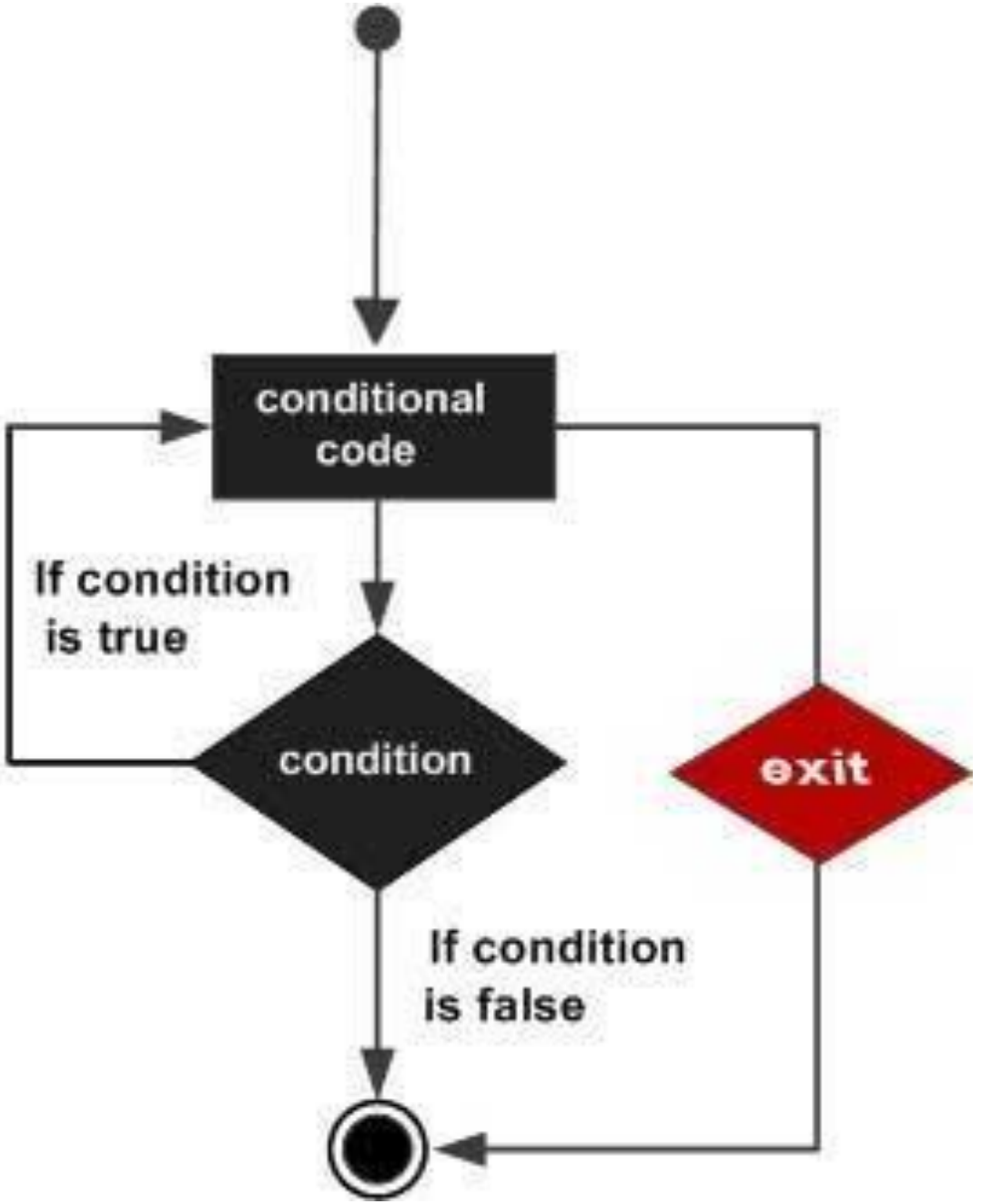
The Loop Control Statements

S.No	Control Statement & Description
1	EXIT statementThe Exit statement completes the loop and control passes to the statement immediately after the END LOOP.
2	CONTINUE statementCauses the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	GOTO statementTransfers control to the labeled statement. Though it is not advised to use the GOTO statement in your program.

EXIT Statement

- The **EXIT** statement in PL/SQL programming language has the following two usages −
- When the EXIT statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- If you are using nested loops (i.e., one loop inside another loop), the EXIT statement will stop the execution of the innermost loop and start executing the next line of code after the block.

- EXIT;



```
DECLARE
  a number(2) := 10;
BEGIN
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    IF a > 15 THEN
      -- terminate the loop using the exit
      statement EXIT;
    END IF;
  END
  LOOP;
END;
/
```

The EXIT WHEN Statement

- The **EXIT-WHEN** statement allows the condition in the WHEN clause to be evaluated. If the condition is true, the loop completes and control passes to the statement immediately after the END LOOP.
- Following are the two important aspects for the EXIT WHEN statement −
- Until the condition is true, the EXIT-WHEN statement acts like a NULL statement, except for evaluating the condition, and does not terminate the loop.
- A statement inside the loop must change the value of the condition.

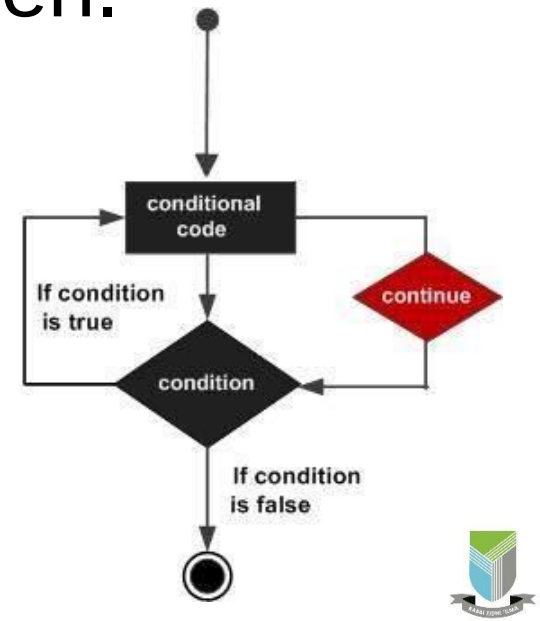
```
EXIT WHEN condition;
```

```
DECLARE
   a number(2) := 10;
BEGIN
   -- while loop execution
   WHILE a < 20 LOOP
      dbms_output.put_line ('value of a: ' || a);
      a := a + 1;
      -- terminate the loop using the exit when statement
   EXIT WHEN a > 15;
   END LOOP;
END;
/
```

CONTINUE Statement

- The **CONTINUE** statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- In other words, it forces the next iteration of the loop to take place, skipping any code in between.

CONTINUE;

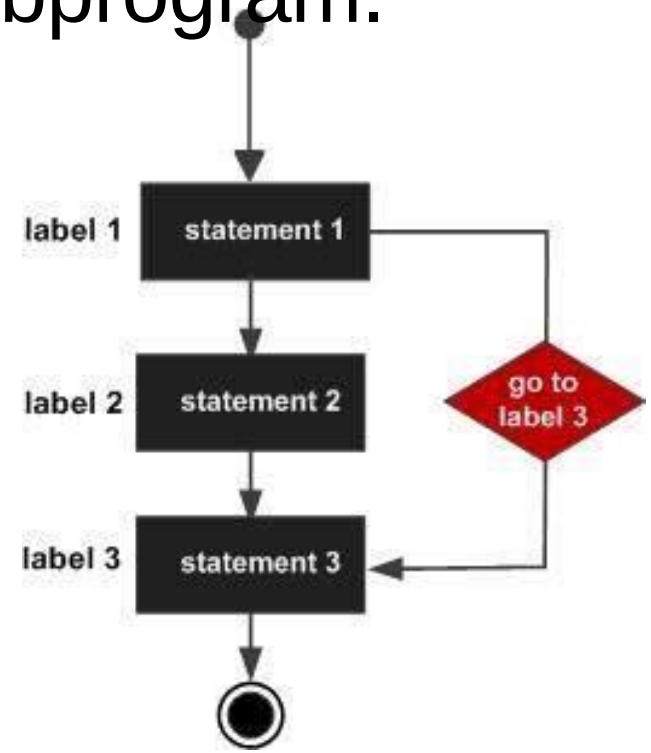


```
DECLARE
  a number(2) := 10;
BEGIN
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' ||
    a); a := a + 1;
    IF a = 15 THEN
      -- skip the loop using the CONTINUE
      statement a := a + 1;
      CONTINUE;
    END IF;
  END LOOP;
END;
/
```

GOTO Statement

- A **GOTO** statement in PL/SQL programming language provides an unconditional jump from the GOTO to a labeled statement in the same subprogram.

```
GOTO label;
..
..
<< label >>
statement;
```



```
DECLARE
  a number(2) :=
10; BEGIN
  <<loopstart>>
  -- while loop execution
  WHILE a < 20 LOOP
  dbms_output.put_line ('value of a: ' ||
    a); a := a + 1;
    IF a = 15 THEN
      a := a + 1;
      GOTO loopstart;
    END IF;
  END
LOOP;
END;
/
```

- https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/a_samps.htm

Starting and Ending a Transaction

- A transaction has a **beginning** and an **end**. A transaction starts when one of the following events take place −
- The first SQL statement is performed after connecting to the database.
- At each new SQL statement issued after a transaction is completed.

- A **COMMIT** or a **ROLLBACK** statement is issued.
- A **DDL** statement, such as **CREATE TABLE** statement, is issued; because in that case a COMMIT is automatically performed.
- A **DCL** statement, such as a **GRANT** statement, is issued; because in that case a COMMIT is automatically performed.
- User disconnects from the database.
- User exits from **SQL*PLUS** by issuing the **EXIT** command, a COMMIT is automatically performed.
- SQL*Plus terminates abnormally, a **ROLLBACK** is automatically performed.
- A **DML** statement fails; in that case a ROLLBACK is automatically performed for undoing that DML statement.

- Committing a Transaction
- A transaction is made permanent by issuing the SQL command COMMIT.

- Rolling Back Transactions
- Changes made to the database without COMMIT could be undone using the ROLLBACK command.

- Savepoints
- Savepoints are sort of markers that help in splitting a long transaction into smaller units by setting some checkpoints.
- By setting savepoints within a long transaction, you can roll back to a checkpoint if required.
- This is done by issuing the **SAVEPOINT** command.

- Automatic Transaction Control
- To execute a COMMIT automatically whenever an INSERT, UPDATE or DELETE command is executed, you can set the AUTOCOMMIT environment variable as −

- SET AUTOCOMMIT ON;
- SET AUTOCOMMIT OFF;

PL/SQL SELECT INTO – selecting one column example

```
DECLARE
  l_customer_name customers.name%TYPE;
BEGIN
  -- get name of the customer 100 and assign it to
  l_customer_name SELECT name INTO l_customer_name
  FROM customers
  WHERE customer_id = 100;
  -- show the customer name
dbms_output.put_line( v_customer_name );
END;
```

- First, declare a variable l_customer_namewhose data type anchors to the name columns of the customers table. This variable will hold the customer name.
- Second, use the SELECT INTO statement to select value from the name column and assign it to the l_customer_name variable.
- Third, show the customer name using the dbms_output.put_line procedure.
- Because the customers table has only one row with customer ID 100, the code block displayed the customer name.
- If there were no such row, the code block would fail

with an unhandled NO_DATA_FOUND exception.

PL/SQL SELECT INTO – selecting a complete row example

```
DECLARE
  r_customer
customers%ROWTYPE; BEGIN
  -- get the information of the customer
  100 SELECT * INTO r_customer
  FROM customers
  WHERE customer_id = 100;
  -- show the customer info
  dbms_output.put_line( r_customer.name || ', website: '
|| r_customer.website );
```

END;

- First, declare a record based on the row of the customers table. This record will hold the entire row of the customers table.
- Second, select the customer whose id is 100 into the r_customer record.
- Third, show the customer’s name and website.

on EMP table

```
DECLARE
  ename
emp.empname%TYPE;
BEGIN
        SELECT empname INTO ename
                                        FROM
                emp WHERE empid = 12345;
        dbms_output.put_line( ename);
END;
```

/

on EMP table

```
DECLARE
  ename
  emp.empname%TYPE; sal
  emp.salary%TYPE;
BEGIN
        SELECT empname,salary INTO ename,sal
                                        FROM
                emp WHERE empid = 12345;
        dbms_output.put_line( 'Name :'||ename);
EN      dbms_output.put_line( 'Salary :'||sal);
D;
/
```

on EMP table

```
DECLARE
  emprow
emp%ROWTYPE; BEGIN
        SELECT * INTO emprow FROM
                emp WHERE empid =
                12345;
        dbms_output.put_line( 'Name
        :'||emprow.empname); dbms_output.put_line(
        'Salary :'||emprow.salary);
END;
```

/