- SQL IS A NON - PROCEDURAL LANGUAGE.
- PL/SQL IS A PROCEDURAL LANGUAGE

<u>**PL/ SQL :**</u>

PROCEDURAL LEARNING EXTENSION OF SQL.

BASICALLY PL/SQL IS A BLOCK STRUCTURED PROGRAMMING LANGUAGE.

WHEN WE ARE SUBMITTING ANY PL/SQL BLOCK INTO ORACLE SERVER THEN ALL THE PROCEDURAL STATEMENTS ARE EXECUTED WITHIN THE PL/SQL ENGINE AND ALSO ALL SQL STATEMENTS ARE SEPARATELY EXECUTED BY USING SQL ENGINE.

## BLOCK STRUCTURE

**DECLARE [OPTIONAL]**
→ VARIABLE DECLARATION, CURSORS, USER DEFINED EXCEPTIONS

**BEGIN [MANDATORY]**
→ DML, TCL
→ SELECT … INTO CLAUSE
→ CONDITIONAL , CONTROL STMNTS;

**EXCEPTION [OPTIONAL]**
→ HANDLING RUNTIME ERRORS

**END;** [MANDATORY]

PL/SQL HAS 2 TYPES OF BLOCKS:

1. ANONYMOUS BLOCK
2. NAMED BLOCK

**ANONYMOUS BLOCK :**

```
EX: DECLARE
        —---
        BEGIN
        —--
        END;
```

- THESE BLOCKS DOES NOT HAVE ANY NAME.
- THESE BLOCKS ARE NOT STORED IN DB.
- THESE BLOCKS ARE NOT ALLOWED TO CALL IN CLIENT APPLICATIONS.

**NAMED BLOCK:**

**EX:** PROCEDURES, FUNCTIONS;

- THESE BLOCKS HAVING A NAME.
- THESE BLOCKS ARE AUTOMATICALLY STORED IN DB.
- THESE BLOCKS ARE ALLOWED TO CALL IN CLIENT APPS.

<u>DECLARING A VARIABLE:</u>

<u>SYNTAX</u>: VARIABLE_NAME DATA_TYPE(SIZE);

<u>EX:</u>

```
DECLARE
A NUMBER(20);
B VARCHAR(30);
```

<u>STORING OR ASSIGNING A VALUE TO A VARIABLE:</u>

USING ASSIGNMENT OPERATOR (:=) WE CAN STORE A VALUE IN VARIABLE.

<u>SYNTAX:</u>

```
VARIABLE_NAME := VALUE;
```

<u>EX:</u>

```
A := 10;
```

B := ‘VIKAS’;

<u>DISPLAY A MESSAGE:</u>

<u>SYNTAX:</u>

DBMS_OUTPUT.PUT_LINE(‘MESSAGE’);

OR

DBMS_OUTPUT.PUT_LINE(VARIABLE_NAME);

DBMS_OUTPUT ⇒ PACKAGE_NAME
PUT_LINE ⇒ PROCEDURE_NAME

EX:

**SET SERVEROUTPUT ON**

BEGIN
DBMS_OUTPUT.PUT_LINE(‘VIKAS’);
END;
/

EX:

DECLARE

```
A NUMBER(20);
B NUMBER(20);
C NUMBER(20);

BEGIN
A:= 20;
B := 30;
C := A+B;
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

**<u>SELECT ….. INTO CLAUSE</u>**

- IT IS USED TO RETRIEVE THE DATA FROM TABLE & STORE IT INTO PL/SQL VARIABLE.
- IT ALWAYS RETURN SINGLE RECORD OR SINGLE VALUE AT A TIME.

**SYNTAX:**

```
SELECT COL_NAME1, COL_NAME2,...
INTO VAR_NAME1,VAR_NAME2,....
FROM TABLE_NAME
[WHERE <FILTER_CONDITION>];
```

(WHERE MUST RETURN A SINGLE VALUE)

- SELECT … INTO CLAUSE IS USED IN EXECUTABLE SECTION OF PL/SQL BLOCK.

1] WRITE A PL/SQL PROGRAM TO DISPLAY DESIGNATION & SALARY OF MILLER.S

```
DECLARE
A VARCHAR(20);
B NUMBER(20);
BEGIN
SELECT JOB, DEPTNO INTO A,B
FROM EMP
WHERE ENAME = 'MILLER';
DBMS_OUTPUT.PUT_LINE(A||' '||B||' ');
```

```
END;
```

2] WRITE A PL/SQL PROGRAM TO DISPLAY MAXIMUM SALARY IN EMP TABLE.

```
DECLARE
A VARCHAR(20);
BEGIN
SELECT MAX(SAL) INTO A
FROM EMP;
DBMS_OUTPUT.PUT_LINE('THE MAXIMUM SALARY IS : '||A);
END;
```

**<u>VARIABLE ATTRIBUTES (ANCHOR NOTATION) :</u>**

VARIABLE ATTRIBUTES ARE USED IN PLACE OF DATA TYPES IN VARIABLE DECLARATION. WHENEVER WE ARE USING VARIABLE ATTRIBUTES, ORACLE SERVER AUTOMATICALLY ALLOCATES MEMORY FOR THE VARIABLES BASED ON THE CORRESPONDING COLUMN DATA TYPE IN A TABLE.

PL/SQL HAVING 2 TYPES OF VARIABLE ATTRIBUTES:
1. COLUMN LEVEL ATTRIBUTE
2. ROW LEVEL ATTRIBUTE

1. **<u>COLUMN LEVEL ATTRIBUTES:</u>**

IN THIS METHODS WE ARE DEFINING ATTRIBUTES FOR INDIVIDUAL COLUMNS. COLUMN LEVEL ATTRIBUTES ARE REPRESENTED BY USING **"%TYPE".**

SYNTAX:

VAR_NAME TABLE_NAME.COL_NAME%TYPE;

EX: V_SAL EMP.SAL%TYPE;

1] WRITE A PL/SQL PROGRAM TO DISPLAY NAME & HIREDATE OF SCOTT.

```
DECLARE
V_ENAME EMP.ENAME%TYPE;
V_HIREDATE EMP.HIREDATE%TYPE;
BEGIN
SELECT ENAME, HIREDATE INTO V_ENAME, V_HIREDATE
FROM EMP
WHERE ENAME = 'SCOTT';
DBMS_OUTPUT.PUT_LINE(V_ENAME||' '||V_HIREDATE);
END;
/
```

```
O/P : SCOTT 19-APR-87
```

## 2. **ROW LEVEL ATTRIBUTE:**

IN THIS METHOD A SINGLE VARIABLE CAN REPRESENT ALL DIFFERENT DATATYPES IN A ROW WITHIN A TABLE. THE VARIABLE IS ALSO CALLED AS “RECORD TYPE VARIABLE”. IT IS REPRESENTED BY “%ROWTYPE”.

SYNTAX:

```
VAR_NAME TABLE_NAME%ROWTYPE;
```

EX:

```
I EMP%ROWTYPE;
```

1] WRITE A PL/SQL PROGRAM TO DISPLAY NAME, DESIGNATION, DEPTNO OF KING.

```
DECLARE
I EMP%ROWTYPE;
BEGIN
SELECT ENAME, JOB, DEPTNO INTO I.ENAME, I.JOB, I.DEPTNO
FROM EMP
WHERE ENAME = 'KING';
DBMS_OUTPUT.PUT_LINE(I.ENAME||' '||I.JOB||' '||I.DEPTNO);
END;
/
```

```
O/P : KING PRESIDENT 10
```

## CONDITIONAL STATEMENTS:

1. If
2. If-else
3. elsif

1] if:

**SYNTAX:**

```
If condition then
Stmts;
end if;
```

2] if-else:

SYNTAX:

```
If condition then
Stmts;
Else
Stmts;
end if;
```

2] elsif:

SYNTAX:

```
If condition 1 them
        Stmts;
elsif condition 2 then
        Stmts;
elsif condition 3 then
        Stmts;
…
else
        Stmts;
end if;
```

1] WRITE A PL/SQL PRGM IF DEPTNO IS 1O THEN DISPLAY DNAME.

```
DECLARE
I DEPT%ROWTYPE;
BEGIN
SELECT * INTO I
FROM DEPT
WHERE DEPTNO =&DEPTNO;
IF I.DEPTNO = 10 THEN
```

```
DBMS_OUTPUT.PUT_LINE('THE DEPT NAME IS :'|| I.DNAME);
END IF;
END;
/

Enter value for deptno: 10
old   6: WHERE DEPTNO =&DEPTNO;
new   6: WHERE DEPTNO =10;
THE DEPT NAME IS :ACCOUNTING
```

2] WRITE A PL/SQL PRGRM TO DISPALY DNAME IF DEPTNO IS 10  ELSE DISPLAY INSERT VALID DEPTNO.

```
 DECLARE
 I DEPT%ROWTYPE;
 BEGIN
 SELECT * INTO I
 FROM DEPT
 WHERE DEPTNO =&DEPTNO;
 IF I.DEPTNO = 10 THEN
 DBMS_OUTPUT.PUT_LINE('THE DEPT NAME IS :'|| I.DNAME);
 ELSE
 DBMS_OUTPUT.PUT_LINE('INSERT VALID DEPTNO');
 END IF;
 END;
/

Enter value for deptno: 10
old   6: WHERE DEPTNO =&DEPTNO;
new   6: WHERE DEPTNO =10;
THE DEPT NAME IS :ACCOUNTING

PL/SQL procedure successfully completed.

SQL> /
Enter value for deptno: 20
old   6: WHERE DEPTNO =&DEPTNO;
new   6: WHERE DEPTNO =20;
INSERT VALID DEPTNO

PL/SQL procedure successfully completed.
```

3] WRITE A PL/SQL PRGRM TO DIPLAY DNAME IF DEPTNO IS 10, LOC IF DEPTNO IS 20, DEPTNO IF IT IS 30 & ALL IF DEPTNO IS OTHER.

```
DECLARE
I DEPT%ROWTYPE;
BEGIN
SELECT * INTO I
FROM DEPT
WHERE DEPTNO =&DEPTNO;
IF I.DEPTNO = 10 THEN
DBMS_OUTPUT.PUT_LINE('THE DEPT NAME IS :'|| I.DNAME);
ELSIF I.DEPTNO = 20 THEN
DBMS_OUTPUT.PUT_LINE('THE LOC IS :'|| I.LOC);
ELSIF I.DEPTNO = 30 THEN
DBMS_OUTPUT.PUT_LINE('THE DEPTNO IS :'|| I.DEPTNO);
ELSE
DBMS_OUTPUT.PUT_LINE(I.DNAME||' '||I.LOC||' '||I.DEPTNO);
END IF;
END;
```

**CASE STATEMENT :**

ORACLE 8.0 INTRODUCED CASE STATEMENT & ALSO ORACLE 8i INTRODUCES CASE CONDITIONAL STATEMENT. THIS STATEMENT IS ALSO CALLED AS “SEARCHED CASE”.

**SYNTAX:**

```
CASE VARIABLE_NAME
WHEN VALUE 1 THEN
        STMT;
WHEN VALUE 2 THEN
        STMT;
…
WHEN VALUE N THEN
        STMT;
ELSE STMT N;
END CASE;
```

1] WRITE A PL SQL PROGRAM IF USER INSERTING THE DEPTNO 10 OR 20 0R 30 DISPLAY THE VALUE IN ALPHABETS.

```
DECLARE
V_DEPTNO NUMBER(10);
BEGIN
SELECT DEPTNO INTO V_DEPTNO
FROM DEPT
WHERE DEPTNO = &DEPTNO;
CASE V_DEPTNO
   WHEN 10 THEN
```

```
          DBMS_OUTPUT.PUT_LINE('TEN');
      WHEN 20 THEN
           DBMS_OUTPUT.PUT_LINE('TWENTY');
      WHEN 30 THEN
           DBMS_OUTPUT.PUT_LINE('THIRTY');
    ELSE
        DBMS_OUTPUT.PUT_LINE('ENTER VALID NUMBER');
    END CASE;
    END;
    /
```

**CASE CONDITIONAL STATEMENT (OR) SEARCHED CASE:**

**SYNTAX:**

```
CASE
WHEN CONDITION 1 THEN
        STMTS;
WHEN CONDITION 2 THEN
        STMTS;
..
WHEN CONDITION N THEN
        STMTS;

ELSE
        STMTS;
END CASE;
```

1] WRITE A PL SQL PROGRAM IF USER INSERTING THE DEPTNO 10 OR 20 0R 30 DISPLAY THE VALUE IN ALPHABETS.

```
DECLARE
V_DEPTNO NUMBER(10);
BEGIN
SELECT DEPTNO INTO V_DEPTNO
FROM DEPT
WHERE DEPTNO = &DEPTNO;
CASE
   WHEN V_DEPTNO=10 THEN
        DBMS_OUTPUT.PUT_LINE('TEN');
   WHEN V_DEPTNO=20 THEN
         DBMS_OUTPUT.PUT_LINE('TWENTY');
   WHEN V_DEPTNO=30 THEN
         DBMS_OUTPUT.PUT_LINE('THIRTY');
 ELSE
      DBMS_OUTPUT.PUT_LINE('ENTER VALID NUMBER');
END CASE;
END;
/
```

**PL/SQL HAS FOLLOWING 3 TYPES OF LOOPS:**

1. SIMPLE LOOP
2. WHILE LOOP
3. FOR LOOP

**SIMPLE LOOP:**

THIS LOOP IS ALSO CALLED AS “INFINITE LOOP”. HERE BODY OF THE LOOP STATEMENTS IS EXECUTED REPEATEDLY.

**SYNTAX:**

```
        LOOP
        STMTS;
        END LOOP;
```

EX:

```
BEGIN
        LOOP
                DBMS_OUTPUT.PUT_LINE(‘HI’);
        END LOOP;
END;
/
```

**NOTE:** IF WE WANT TO EXIT FROM INFINITE LOOP THEN WE ARE USING ORACLE PROVIDED PREDEFINED METHODS.

**METHOD 1:** (DEFAULT METHOD)

SYNTAX :

```
EXIT WHEN <TRUE CONDITION>;
```

EX:

```
DECLARE
A NUMBER(10) := 1;
BEGIN
LOOP
        DBMS_OUTPUT.PUT_LINE(A);
        EXIT WHEN A>=10;
        A := A+1;
END LOOP;
END;
/
```

**METHOD 2: (USING IF):**

SYNTAX:

```
IF TRUE CONDITION THEN
        EXIT;
END IF;
```

**EX:**

```
DECLARE

A NUMBER(10) := 1;

BEGIN

LOOP

 DBMS_OUTPUT.PUT_LINE(A);

 IF A >= 10 THEN

          EXIT;

END IF;

 A := A+1;

END LOOP;

END;

/
```

**<u>WHILE LOOP:</u>**

HERE BODY OF THE LOOP STATEMENTS ARE EXECUTED REPEATEDLY UNTIL CONDITION IS FALSE. IN “WHILE LOOP” WHENEVER CONDITION IS TRUE THEN ONLY LOOP BODY IS EXECUTED.

**SYNTAX:**

```
WHILE(CONDITION)
LOOP
        STMTS;
END LOOP;
```

**EX:**

```
DECLARE
A NUMBER(10) := 1;
BEGIN
WHILE A<= 10
LOOP
 DBMS_OUTPUT.PUT_LINE(A);
 A := A+1;
END LOOP;
END;
/
```

## <u>**FOR LOOP:**</u>

SYNTAX:

```
FOR INDEX_VARIABLE_NAME IN LOWERBOND..UPPERBOND
LOOP
        STMTS;
END LOOP;
```

EX:

```
DECLARE
A NUMBER(10) ;
BEGIN
FOR I IN 1..10
LOOP
 DBMS_OUTPUT.PUT_LINE(I);
END LOOP;
END;
/
```

**NOTE:**

FOR LOOP INDEX VARIABLE INTERNALLY BEHAVES LIKE AN "INTEGER" VARIABLE THAT'S WHY WHEN WE ARE USING "FOR LOOP" WE ARE NOT REQUIRED TO DECLARE VARIABLE IN DECLARE SECTION. GENERALLY PL/SQL FOR LOOP IS ALSO CALLED AS **"NUMERIC FOR LOOP".**

```
DECLARE

A NUMBER(10);

BEGIN

FOR I IN REVERSE 1..10

LOOP

 DBMS_OUTPUT.PUT_LINE(I);

 A := A+1;

END LOOP;

END;

/
```

**VIEW**

**-----**

VIEWS ARE THE VIRTUAL TABLE WHICH CAN BE CREATED AND

RE-USED WHEN EVER WE ARE DEALING WITH A PART OF A TABLE.

MySQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name
```

[WHERE *condition]*;

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

MySQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil

EX:

```
CREATE VIEW V1 AS

SELECT *

FROM EMP;
```

We can query the view above as follows

```
SELECT * FROM V1;
```

The following SQL creates a view that selects ALL the Details  in the "Emp" table who are earning more than 2000 rps.

Example

```
CREATE VIEW v2 AS

SELECT *

FROM Emp

WHERE sal > 2000
```

We can query the view above as follows:

```
SELECT * FROM v2;
```

---

MySQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name
```

```
WHERE condition;
```

The following SQL adds the condition to the v1 view:

Example

```
CREATE OR REPLACE VIEW v1 AS

SELECT ename, sal, deptno

FROM emp

WHERE deptno = 20;
```

---

## MySQL Dropping a View

A view is deleted with the DROP VIEW statement.

DROP VIEW Syntax

```
DROP VIEW view_name;
```

The following SQL drops the "v2" view:

Example

```
DROP VIEW v2;
```

**Difference between View and Table:**

Following are the differences between the view and table.

| **Basis** | **View** | **Table** |
|---|---|---|

| | | |
|---|---|---|
| **Definition** | A view is a database object that allows generating a logical subset of data from one or more tables. | A table is a database object or an entity that stores the data of a database. |
| **Dependency** | The view depends on the table. | The table is an independent data object. |
| **Database space** | The view is utilized database space when a query runs. | The table utilized database space throughout its existence. |
| **Manipulate data** | We can not add, update, or delete any data from a view. | We can easily add, update, or delete any data from a table. |
| **Recreate** | We can easily use replace option to recreate the view. | We can only create or drop the table. |

|  |  |  |
|---|---|---|
| **Aggregation of data** | Aggregate data in views. | We can not aggregate data in tables. |
| **table/view relationship** | The view contains complex multiple tables joins. | In the table, we can maintain relationships using a primary and foreign key. |

INDEX:

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

WHEN WE HAVE INDEX, IT ENHANCE (INCREASES) THE SPEED OF SEARCHING IN THE DATABASE.

i.e, THERE ARE 1000 TABLES IN DATABASE, or 1000 records in a TABLE.

SO FOR EASY OF ACCESS, WE USE INDEX.

INDEX ARE OF 2 TYPES:

1. CLUSTERED

2. NON-CLUSTERED

<u>CREATE INDEX Syntax</u>

CREATE INDEX *index_name*

ON *table_name* (*column1*, *column2*, ...);

EX:

CREATE INDEX I1

ON EMP(SAL ASC);

Creating Index using multiple Columns:

Ex:

Create Index wages
On emp(sal, comm,hiredate);

## DROP INDEX Statement

SYNTAX:

ALTER TABLE TABLE_NAME
DROP INDEX *index_name*;

OR

DROP INDEX INDEX_NAME ===> ORACLE

```
EX: ALTER TABLE EMP
        DROP INDEX I1;
```

EX: Drop Index Salary;

For MySql

Alter Table Table_name
Drop Index Index_name;

EX:

Alter Table Emp
Drop Index salary;

**<u>POINTS TO REMEMBER ABOUT INDEXES:</u>**

- TO FACILIATE QUICK RETRIEVAL OF DATA FROM A DB WE USE INDEXES.
- AN INDEX IN SQL CONTAINS INFORMATION THAT ALLOWS YOU TO FIND SPECIFIC DATA WITHOUT SCANNING THROUGH THE ENTIRE TABLE.
- CREATE INDEXES ON COLUMNS THAT WILL BE FREQUENTLY SEARCHED AGAINST.
- AN INDEX IS A POINTER TO DATA IN A TABLE.
- AN INDEX HELPS TO SPEED UP SELECT CLAUSES AND WHERE CLAUSESE, BUT IT SLOWS DOWN DATA I/P, WITH THE UPDATE AND THE INSERT STATEMENTS.
- INDEX CAN BE CREATED OR DROPPED WITH NO EFFECT ON THE DATA.

- INDEXES ARE CREATED AUTOMATICALLY CREATED WHEN PRIMARY KEY AND UNIQUE CONSTRAINTS ARE DEFINED ON A TABLE.

**A SINGLE COLUMN INDEX:**

A SINGLE COLUMN INDEX IS CREATED BASED ON ONLY ONE COLUMN OF A TABLE.

**MULTIPLE COLUMN INDEX:**

IT IS CREATED BASED ON MULTIPLE COLUMNS OF A TABLE.

## IMPLICIT INDEXES:

THESE ARE INDEXES THAT ARE CREATED AUTOMATICALLY BY THE DB SERVER WHEN AN  OBJECT IS CREATED. INDEXES ARE AUTOMATICALLY CREATE FOR PRIMARY KEY CONSTRAINTS & UNIQUE CONSTRAINTS.

## WHEN SHOULD AVOID INDEXES?

ALTHOUGH INDEXES ARE INTENDED TO ENHANCE DB'S PERFORMANCE, THERE ARE TIMES WHEN THEY SHOULD BE AVOIDED.

FOLLOWING ARE THE CASES / SCENARIOS:

- INDEXES SHOULD NOT BE USED ON SMALL TABLES.

- TABLES HAT HAVE FREQUENT, LARGE UPDATES OR INSERT OPERATIONS.
- INDEXES SHOULD NOT BE USED ON COLUMNS THAT CONTAINS A HIGH NUMBER OF NULL VALUES.
- COLUMNS THAT ARE FREQUENTLY MANIPULATED SHOULD NOT BE INDEXED.

POINTS TO BE REMEMBER:

- TO FACILITATE QUICK RETRIEVAL OF DATA FROM A DB WE USE INDEX.
- INDEXES ON TABLE IS VERY SIMILAR TO AN INDEX THAT WE FIND IN A BOOK.
- IT HELPS TO REDUCE THE TIME TO RETRIEVE THE DATA.
- THE EXISTENCE OF THE RIGHT INDEXES, CAN IMPROVE THE PERFORMANCE OF THE QUERY.IF THERE IS NO INDEX TO HELP THE QUERY, THEN QUERY ENGINE, CHECKS EVERY ROW IN THE TABLE  FROM THE BEGINNING TO END. THIS IS CALLED AS TABLE SCAN, TABLE SCAN IS VERY BAD FOR PERFORMANCE.

KEY POINTS:

- CREATE INDEX IN COLUMNS THAT WILL BE FREQUENTLY SEARCHED AGAINST.
- AN INDEX IS A POINTER TO DATA IN A TABLE.
- AN INDEX HELPS TO SPEED UP SELECT QUERIES & WHERE CLAUSES, BUT IT SLOWS DOWN DATA I/P,WITH THE UPDATE & INSERT STATEMENTS.
- INDEXES CAN BE CREATED OR DROPPED WITH NO EFFECT ON THE DATA.
- INDEXES ARE AUTOMATICALLY CREATED WHEN PRIMARY KEY & UNIQUE CONSTRAINTS ARE DEFINED IN A TABLE.

IMPLICIT INDEX:
THERE ARE INDEXES THAT ARE AUTOMATICALLY CREATED BY THE DB SERVER WHEN AN OBJECT IS CREATED.

WHEN SHOULD INDEXES CAN BE AVOIDED:
ALTHOUGH INDEXES ARE INTENDED TO ENHANCE A DB’S PERFORMANCE, THERE ARE TIMES WHEN THEY SHOULD BE AVOIDED.

- INDEXES SHOULD NOT BE USED ON SMALL TABLES.
- TABLES THAT HAVE FREQUENT, LARGE BATCH UPDATES OR INSERT OPERATIONS.
- INDEXES SHOULD NOT BE USED ON COLUMNS THAT CONTAIN A HIGH NUMBER OF NULL VALUES.

- COLUMNS THAT ARE FREQUENTLY MANIPULATED SHOULD NOT BE INDEXED.

CLUSTERED INDEX:

- A CLUSTERED INDEX CAUSES RECORDS TO BE STORED PHYSICALLY STORED IN A SORTED OR SEQUENTIAL ORDER.
- A CLUSTERED INDEX DETERMINES THE ACTUAL ORDER IN WHICH DATA IS STORED IN THE DB. HENCE, YOU CAN CREATE ONLY ONE CLUSTERED INDEX IN A TABLE.
- UNIQUENESS OF A VALUE IN CLUSTERED INDEX IS MAINTAINED EXPLICITLY USING THE UNIQUE KEYWORD OR IMPLICITLY USING AN INTERNAME UNIQUE IDENTIFIER.
- CLUSTERED INDEX IS AS SAME AS DICTIONARY WHERE DATA IS ARRANGED BY ALPHABETICAL ORDER.
- WE CAN HAVE ONLY ONE CLUSTERED INDEX IN ONE TABLE, BUT WE CAN HAVE ONE CLUSTERED INDEX ON MULTIPLE COLUMNS & THAT TYPE OF INDEX IS CALLED AS COMPOSITE INDEX.

NON CLUSTERED INDEX:

- A NON - CLUSTERED INDEX IS AS SAME AS AN INDEX OF A BOOK.
- THE DATA IS STORED IN ONE PLACE , AND INDEX IS STORED IN ANOTHER PLACE.
- SINCE, THE NON-CLUSTERED INDEX IS STORED SEPARATELY FROM THE ACTUAL DATA, A TABLE CAN HAVE MORE THAN ONE NON - CLUSTERED INDEX.
- JUST LIKE HOW A BOOK CAN HAVE INDEX BY CHAPTERS AT THE BEGINNING AND ANOTHER INDEX BY COMMON TERMS AT THE END.

## MySQL LIMIT Clause

---

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)

FROM table_name

[WHERE condition]

LIMIT number;
```

MySQL LIMIT Examples

The following SQL statement selects the first three records from the "EMP" table:

Example

```
SELECT *
FROM EMP
LIMIT 3;
```

What if we want to select records 4 - 6 (inclusive)?

MySQL provides a way to handle this: by using OFFSET.

The SQL query below says "return only 3 records, start on record 4 (OFFSET 3)":

Example

```
SELECT * FROM Customers
LIMIT 3 OFFSET 3;
```

ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "EMP" table, where the JOB is "SALESMAN":

EX:

```
SELECT *

 FROM EMP

WHERE JOB = ‘SALESMAN’

LIMIT 3;
```