DBMS

Department of Computer Science & Engineering

B. Tech (CSE)

UNIT-2

Relational Algebra & Calculus:

1. Preliminaries:

A query language is a language in which user requests to retrieve some information from the database. The query languages are considered as higher level languages than programming languages.

Query languages are of two types, Procedural Language Non-Procedural Language

1. In procedural language, the user has to describe the specific procedure to retrieve the information from the database. Example: The Relational Algebra is a procedural language.

2. In non-procedural language, the user retrieves the information from the database without describing the specific procedure to retrieve it. Example: The Tuple Relational Calculus and the Domain Relational Calculus are non-procedural languages.

EF.Codd's 12 Rules:

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Relation Data Model:

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple – A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema – A relation schema describes the relation name (table name), attributes, and their names.

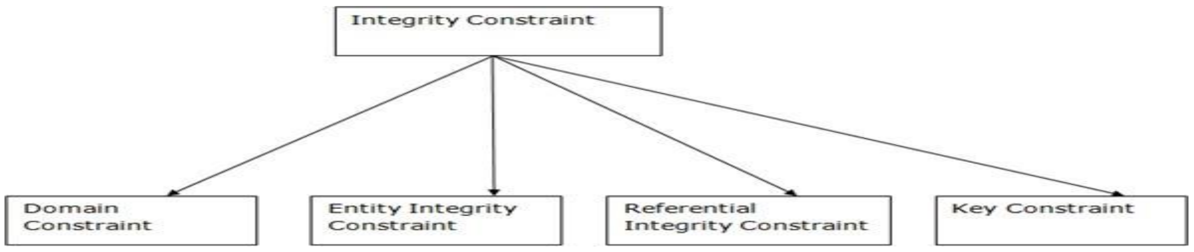
Relation key – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain – Every attribute has some pre-defined value scope, known as attribute domain.

Types of Integrity Constraint :

- Integrity constraints are set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

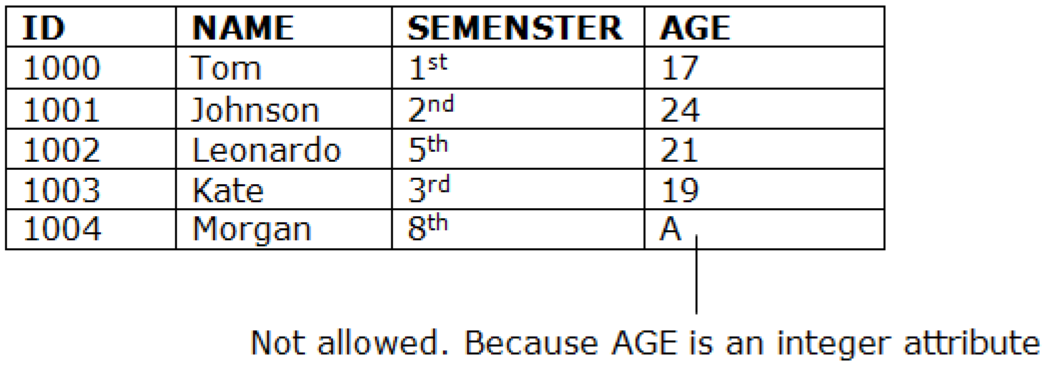
Types of Integrity Constraint



1. Domain constraints

    - Domain constraints can be defined as the definition of a valid set of values for an attribute.
    - The data type of domain includes string, character, integer, time, date, currency etc. The value of attribute must be available in the corresponding domain.

Example:

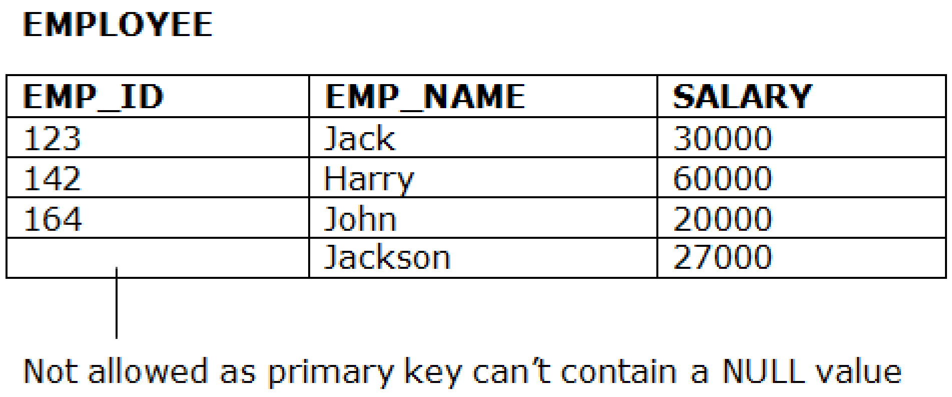


ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

    - The entity integrity constraint states that primary key value can't be null.
    - This is because the primary key value is used to identify individual rows in a relation and if the primary key has null value then we can't identify those rows.
    - A table can contain null value other than primary key field.



EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

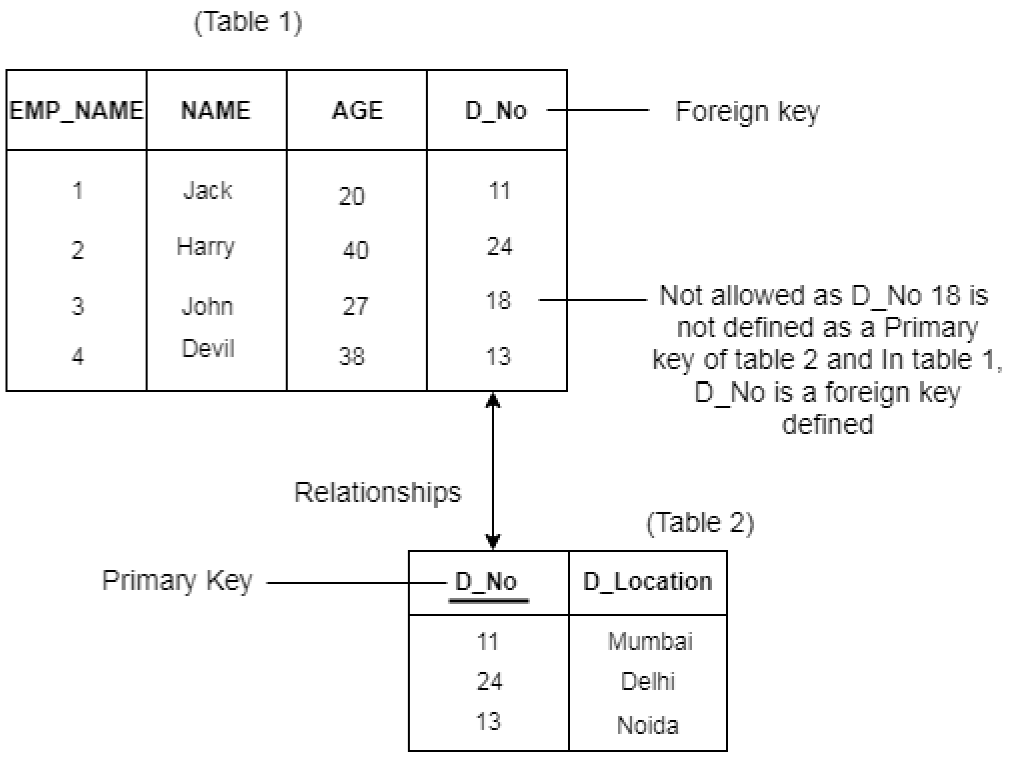
Example:

3.Referential Integrity Constraints:

Referential integrity constraint is specified between two tables.

- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2 then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:

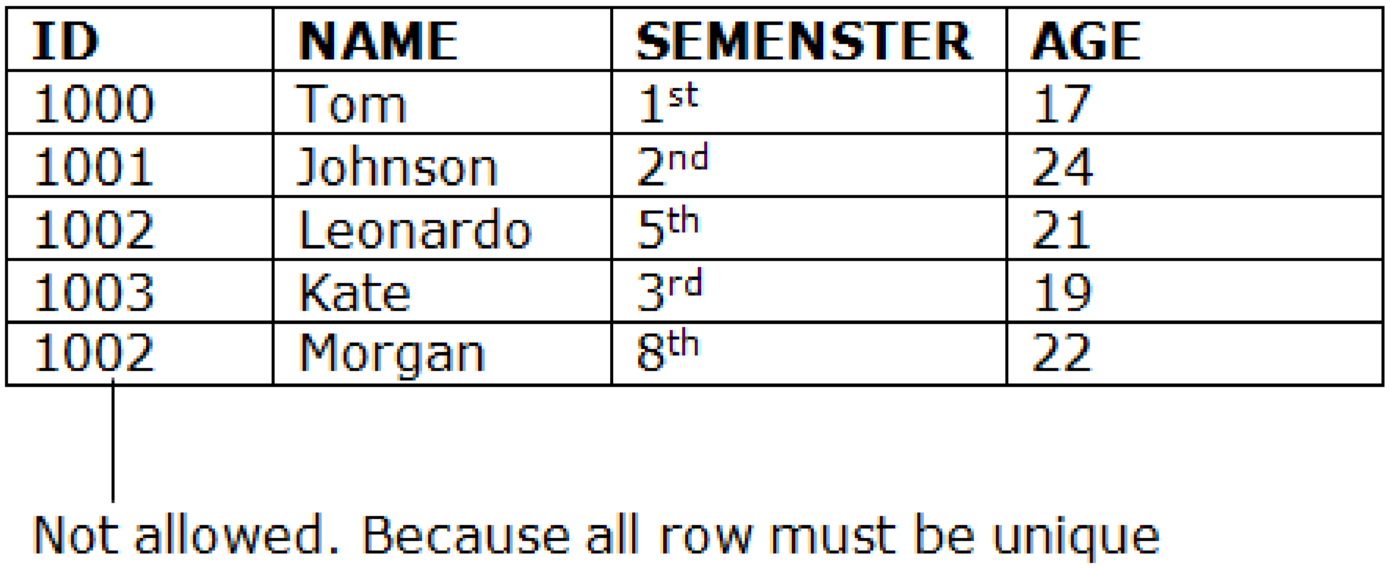

(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

3. Key constraints

- Keys are the entity set that is used to uniquely identify an entity within its entity set.
- An entity set can have multiple key but out of which one key will be primary key. A primary key can contain only unique and null value in the relational table.



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Example:

Relational Algebra:

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

```
σsubject = "database"(Books)
```

Output − Selects tuples from books where subject is 'database'.

```
σsubject = "database" and price = "450"(Books)
```

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

```
σsubject = "database" and price = "450" or year > "2010"(Books)
```

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}$ (r)

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

```
∏subject, author (Books)
```

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −

```
r ∪ s = { t | t ∈ r or t ∈ s}
```

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

```
∏ author (Books) ∪ ∏ author (Articles)
```

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

```
∏ author (Books) − ∏ author (Articles)
```

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

```
σauthor = 'KORTH'(Books X Articles)
```

Output − Yields a relation, which shows all the books and articles written by KORTH.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms −

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation − {T | Condition}

Returns all tuples T that satisfies a condition.

For example −

```
{ T.name |  Author(T) AND T.article = 'database' }
```

Output − Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

For example −

```
{ R| ∃T   ∈ Authors(T.article='database' AND R.name=T.name)}
```

Output − The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation −

{ a_1, a_2, a_3, ..., a_n | P (a_1, a_2, a_3, ... ,a_n)}

Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

For example −

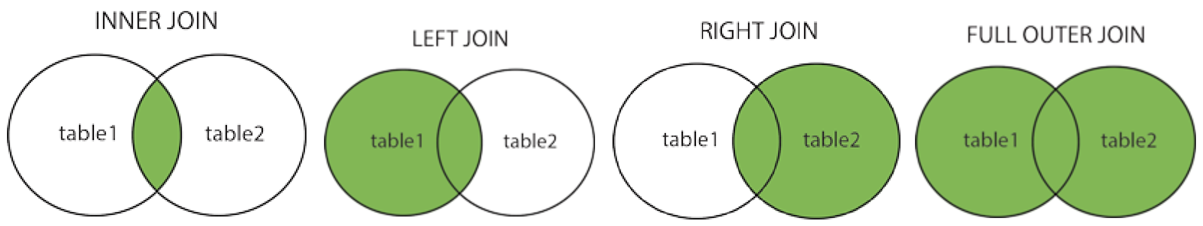
```
{< article, page, subject > |  ∈ KORTH ∧ subject = 'database'}
```

Output − Yields Article, Page, and Subject from the relation KORTH, where subject is database.

Join With types:

Different Types of JOINs(INNER) JOIN: Returns records that have matching values in both tables

- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table.
- RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table.
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table.



Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	AlfredsFutterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID,                    Customers.CustomerName
```

FROM Orders
INNER JOIN Customers ON Orders.CustomerID =

Customers.CustomerID; SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName,                                    Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID      =
                                              Orders.CustomerID ORDER
BY Customers.CustomerName;
```

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName,                                    Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON
Customers.CustomerID=Orders.CustomerID ORDER BY
Customers.CustomerName;
```

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have have placed:

Example

```
SELECT Orders.OrderID,        Employees.LastName,        Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID  =
                                           Employees.EmployeeID
ORDER BY Orders.OrderID
```

SQL

- SQL stands for Structure Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like: MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a numbers of ways, using English-like statements.

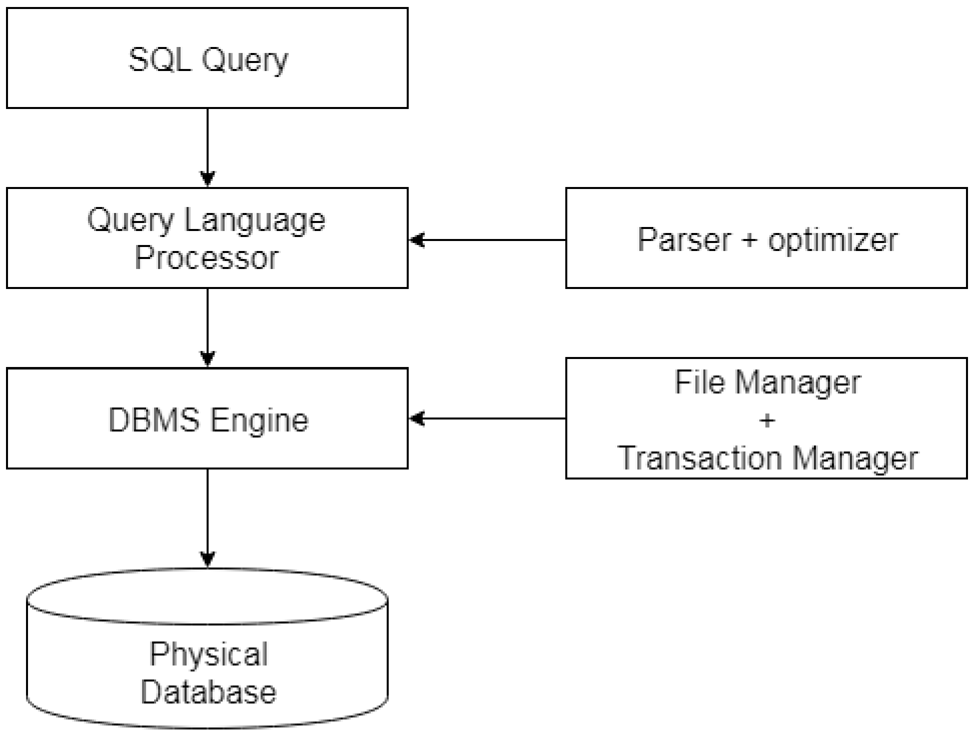
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS then the system figure out the best way to carry out the request and SQL engine determines that how to interpret the task.
- In the process, various components are included. These components are optimization Engine, Query engine, Query dispatcher, classic etc.
- All the non-SQL queries are handled by the classic query engine but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.

- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that is being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some of mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

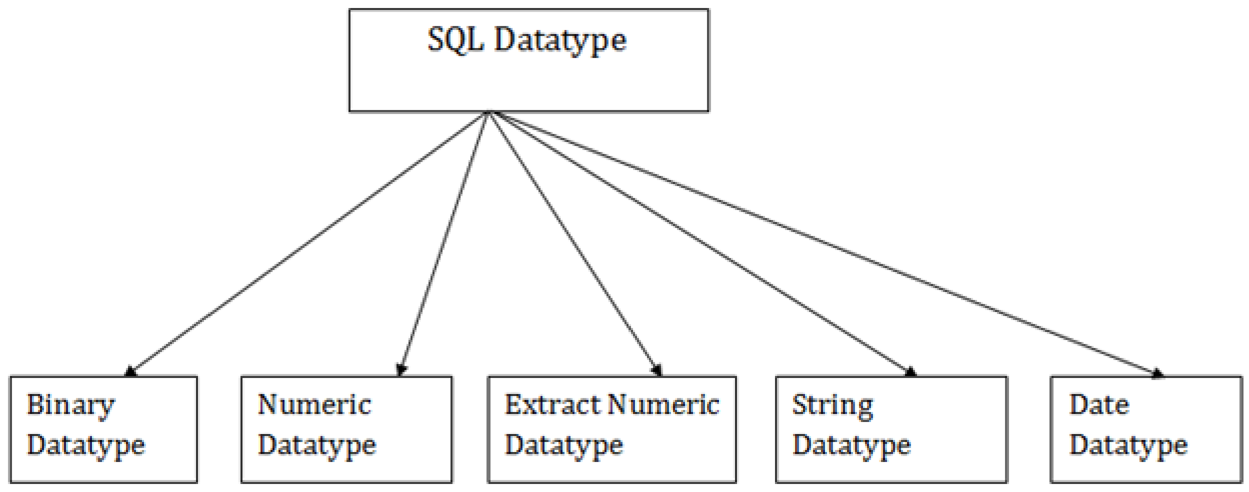
Multiple data view

Using the SQL language, the users can make different views of database structure.

SQL Datatype:

- SQL Data type is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Datatype of SQL:



1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
Binary	It has maximum length of 8000 bytes. It contains fixed-length binary data.
Varbinary	It has maximum length of 8000 bytes. It contains variable-length binary data.
Image	It has maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Datatype :

The subtypes are given below:

Data type	From	To	Description
Float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.

Real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
Int	It is used to specify an integer value.
Smallint	It is used to specify small integer value.
Bit	It has the number of bit to store.
Decimal	It specifies a numeric values that can have a decimal number.
Numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
Char	It has maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
Varchar	It has maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

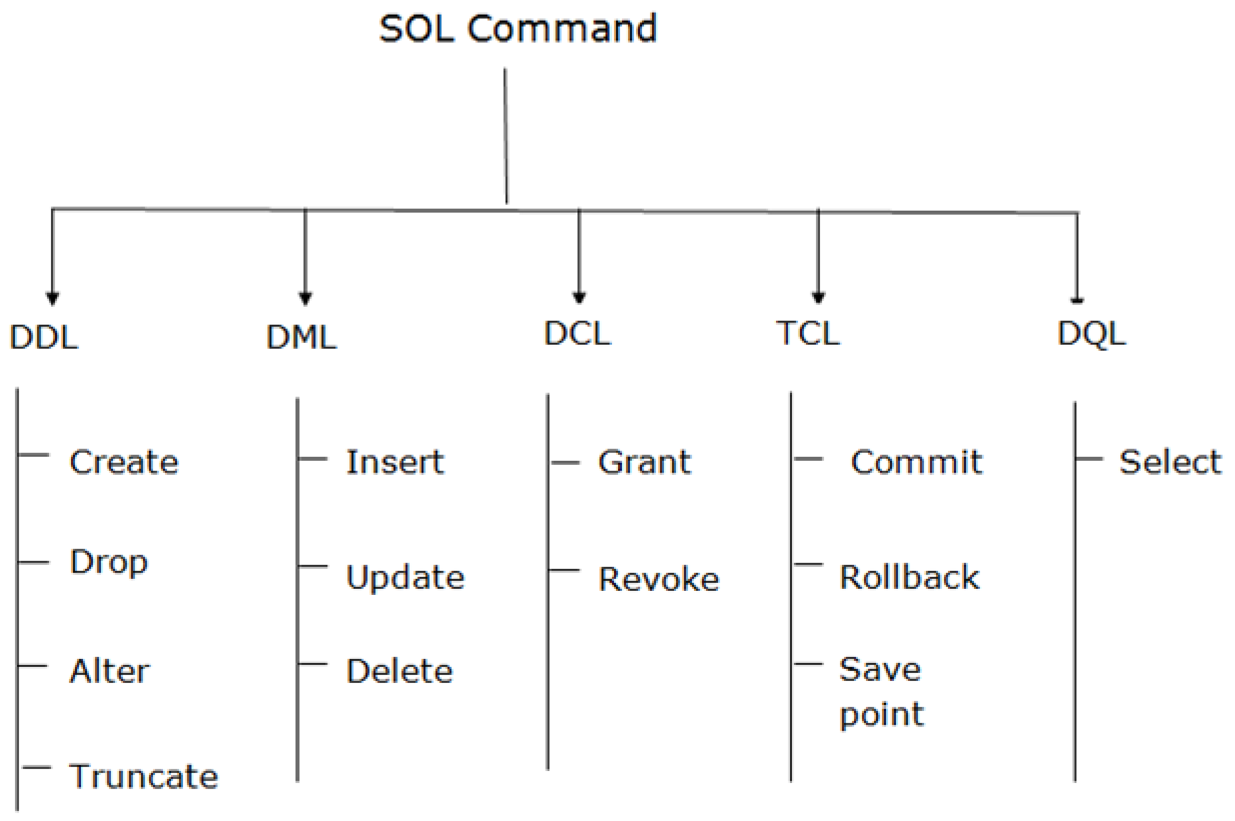
The subtypes are given below:

Datatype	Description
Date	It is used to store year, month and days value.
Time	It is used to store hour, minute and second values.
Timestamp	It stores year, month, day, hour, minute and second value.

SQL command

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions and queries of data.
- SQL can perform various tasks like: create table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Command:



1. Data definition language (DDL)

- DDL changes the structure of the table like: creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

1. DROP TABLE ;

Example

1. DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add new column in table

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in table

```
1. ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

EXAMPLE

```
1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: it is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML are not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);
```

Or

```
1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, .... valueN);
```

For example:

```
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of column in the table.

Syntax:

```
1. UPDATE table_name SET [column_name1= value1,...column_nameN = va
   lueN] [WHERE CONDITION]
```

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
1. DELETE FROM table_name [WHERE condition];
```

For example:

```
1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

```
1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER
   ;
```

b. Revoke: It is used to take back permissions from the user.

Example

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

4. Transaction Control Language

TCL commands can only use with DML commands like: INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
COMMIT;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

1. SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

Consider the following relation. Worker (Worker_Id, First_Name, Last_Name, Salary, Joining_Date, Department), Bonus (Worker_Ref_Id, Bonus_Date, Bonus_Amount), Title (Worker_Ref_Id, Worker_Title, Affected_From). The Primary key is Rollno, Isbn. Write the query in Relational algebra of the following:

(a) Write An SQL Query That Fetches The Unique Values Of DEPARTMENT From Worker Table And Prints Its Length.

```
Select distinct length(DEPARTMENT) from Worker;
```

(b) Write An SQL Query To Print Details For Workers With The First Name As "Vipul" And "Satish" From Worker Table.

```
Select * from Worker where FIRST_NAME in ('Vipul','Satish');
```

(c) Write An SQL Query To Print Details Of The Workers Who Have Joined In Feb’2014.

```
Select * from Worker where year(JOINING_DATE) = 2014 and
month(JOINING_DATE) = 2;
```

(d) Write An SQL Query To Fetch The No. Of Workers For Each Department In The Descending Order.

```
SELECT DEPARTMENT, count(WORKER_ID) No_Of_Workers
FROM worker
GROUP BY DEPARTMENT
ORDER BY No_Of_Workers DESC;
```

(e) Write An SQL Query To Fetch Duplicate Records Having Matching Data In Some Fields Of A Table.

```
SELECT WORKER_TITLE, AFFECTED_FROM, COUNT(*)
FROM Title
GROUP BY WORKER_TITLE, AFFECTED_FROM
HAVING COUNT(*) > 1;
```

(f) Write An SQL Query To Fetch The Departments That Have Less Than Five People In It.

```
SELECT DEPARTMENT, COUNT(WORKER_ID) as 'Number of Workers' FROM Worker GROUP
BY DEPARTMENT HAVING COUNT(WORKER_ID) < 5;
```

Characteristics Of SQL & five aggregate Function:

Solution. Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures and views.

The following are the most commonly used SQL aggregate functions:

- AVG – calculates the average of a set of values.
- COUNT – counts rows in a specified table or view.
- MIN – gets the minimum value in a set of values.
- MAX – gets the maximum value in a set of values.

- SUM – calculates the sum of values.

SQL aggregate function

examples COUNT function

example

To get the number of the products in the products table, you use the COUNT function as follows:

```
SELECTCOUNT(*)FROM products;
```

AVG function example

To calculate the average units in stock of the products, you use the AVG function as follows:

```
SELECT
AVG(unitsinstock)
FROMproducts;
```

MIN function example

To get the minimum units in stock of products in the products table, you use the MIN function as follows:

```
SELECTMIN(unitsinstock)FROMproducts;
```

MAX function example

To get the maximum units in stock of products in the products table, you use the MAX function as shown in the following query:

```
SELECT

MAX(unitsinstock)

FROM products;
```

Domain and Tuple Relational Calculus:

Solution. Relational Calculus in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in

two forms: **Tuple Relational**

Calculus (TRC) Domain

Relational Calculus (DRC)

<u>**TupleRelationalCalculus (RC):**</u>

In tuple relational calculus, we

work on filtering tuples based

on the given condition.

condition. Syntax: { T |

Condition }

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.

We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.

A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for

data, we do the following: Relation(T), where T is our tuple variable.

For example if our table is Student, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particluar attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table Student, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table Student, with age greater than 17, then, for T being our tuple variable,

T.name | Student(T) AND T.age > 17

Domain Relational Calculus (DRC)

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

For example,

∈
{< name, age > | Student age > 17}
∧

Again, the above query will return the names and ages of the students in the table Student who are older than 17.

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables

- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

1. **CREATE** [OR REPLACE] **TRIGGER** trigger_name
2. {BEFORE | **AFTER** | **INSTEAD OF** }
3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
4. [**OF** col_name]
5. **ON** table_name
6. [REFERENCING OLD **AS** o NEW **AS** n]
7. [**FOR** EACH ROW]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. EXCEPTION
14. Exception-handling-statements
15. **END**;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.

- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000

4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

Trigger created.

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

Old salary: 20000

New salary: 25000

```
Salary difference: 5000

Old salary: 22000

New salary: 27000

Salary difference: 5000

Old salary: 24000

New salary: 29000

Salary difference: 5000

Old salary: 26000

New salary: 31000

Salary difference: 5000

Old salary: 28000

New salary: 33000

Salary difference: 5000

Old salary: 30000

New salary: 35000

Salary difference: 5000

6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

```
Old salary: 25000

New salary: 30000
```

```
Salary difference: 5000

Old salary: 27000

New salary: 32000

Salary difference: 5000

Old salary: 29000

New salary: 34000

Salary difference: 5000

Old salary: 31000

New salary: 36000

Salary difference: 5000

Old salary: 33000

New salary: 38000

Salary difference: 5000

Old salary: 35000

New salary: 40000

Salary difference: 5000

6 customers updated
```

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Cursors

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.
There are two types of cursors in PL/SQL :

1. Implicit cursors.
2. Explicit cursors.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Cursor Attributes	
Name	**Description**
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.
%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records fetched from

	cursor at that point in time.
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise.

Implicit cursors :

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.
When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit statements are created to process these statements.
Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.
For example, When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected. When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.
Consider the PL/SQL Stock that uses implicit cursor attributes as shown below:

Example:

```
DECLARE

   Eid  number(3);

 BEGIN

   UPDATE emp set eid=&eid where salary=&salary; eid:=sql%rowcount;

   IF SQL%found then

        dbms_output.put_line('success');

   ELSE
```

```
        dbms_output.put_line ( ' not' ) ;
    END IF;
        dbms_output.put_line( 'rowcount'||eid);
 END;
 /
```

Output:

Run SQL Command Line

```
SQL>START D://c.sql
success
PL/SQL Procedure successfully completed.
```

Explicit Cursors :

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.
An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

Syntax:

```
CURSOR cursor_name IS select_statement;
```

cursor _name -A suitable name for the cursor.
Select_statement - A select query which returns multiple rows.

How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

1. **DECLARE** the cursor in the declaration section

2. **OPEN** the cursor in the Execution Section.
3. **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
4. **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

Declaring a Cursor in the Declaration Section:

Syntax:

CURSOR CURSORNAME IS SELECT.......;

How to access an Explicit Cursor?

These are the three steps in accessing the cursor.
Open the cursor :

Syntax:

OPEN cursor_name;

Fetch the records in the cursor one at a time :

Syntax:

FETCH cursor_name INTO record_name;

OR

FETCH cursor_name INTO variable_list;

Close the cursor :

Syntax:

CLOSE cursor__name;

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

Points to remember while fetching a row:

1. We can fetch the rows in a cursor to a PL/SQL Record or a list of variables created in the PL/SQL Block.
2. If you are fetching a cursor to a PL/SQL Record, the record should have the same structure as the cursor.
3. If you are fetching a cursor to a list of variables, the variables should be listed in the same order in the fetch statement as the columns are present in the cursor.

General Form of using an explicit cursor is:

Syntax:

```
DECLARE
  variables;
  records;
  create a cursor;
 BEGIN
  OPEN cursor;
  FETCH cursor;
  process the records;
  CLOSE cursor;
 END;
```

Example:

```
DECLARE

  CURSOR er IS select eid,name from emp order by name ;

  id emp.eid%type;

  ename emp.name%type;

 BEGIN

  OPEN er;

  Loop

   FETCH er into id,ename;

   Exit when er%notfound;

   dbms_output.put_line (id || ename);

  end loop;

    close er;

 END;

 /
```

Output:

Run SQL Command Line

```
SQL>start D://ec.sql
1||parimal
2||preet
PL/SQL Procedure successfully completed.
```

Formated by [Tuotiral Blocks]()

Cursor For Loop :

Oracle provides another loop-statements to control loops specifically for cursors. This statements is a variation of the basic FOR loop , and it is known as cursor for loops.

Syntax:

```
FOR VARIABLE IN CURSORNAME

LOOP

  <EXECUTE COMMANDS>

END LOOP
```

Parameterized Cursors :

Oracle allows to pass parameters to cursors that can be used to provide condition with WHERE clause.If parameters are passed to curso, that cursor is called a **parameterized cursors.**

Syntax:

```
CURSOR CURSORNAME (VARIABLENAME DATATYPES ) IS SELECT.........
```

And, parameters canbe passed to cursor while opening it using syntax-

Syntax:

```
OPEN CURSORNAME (VALUE / VARIABLE / EXPRESSION );
```