ATTRIBUTES

1. KEY ATTRIBUTES / CANDIDATE KEY
2. NON-KEY ATTRIBUTE
3. PRIME-KEY ATTRIBUTE
4. NON-PRIME KEY ATTRIBUTE
5. COMPOSITE KEY ATTRIBUTE
6. SUPERKEY ATTRIBUTE
7. FOREIGN KEY ATTRIBUTE

KEY ATTRIBUTES/ CANDIDATE KEY

An attribute which is used to identify a record uniquely from the table is called Key Attribute.

NON-KEY ATTRIBUTE

All the attributes except Key Attributes are referred as Non-Key Attributes.

PRIME KEY ATTRIBUTES

Among the Key Attributes, an attribute is chosen to be the main attribute to identify the record uniquely from the table.

NON-PRIME KEY ATTRIBUTE

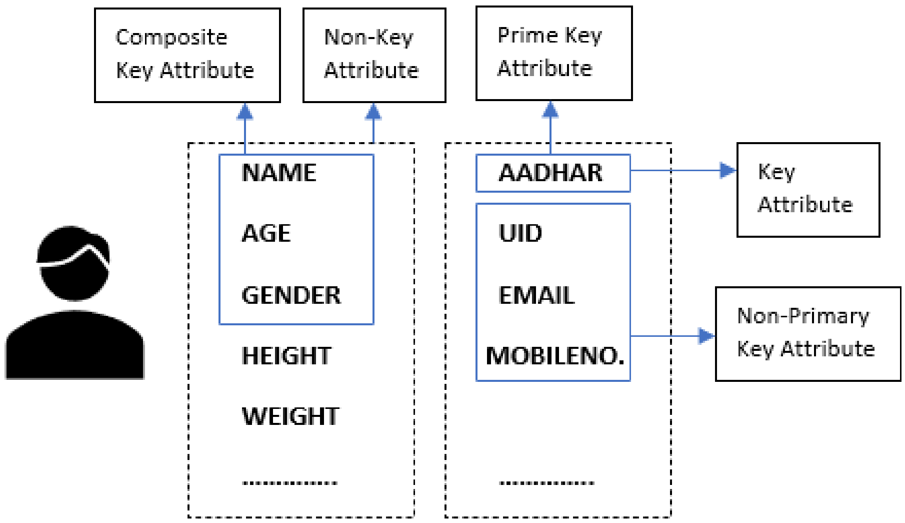
All the key attributes except Prime-Key Attribute are referred as Non-Prime Key Attribute.

COMPOSITE KEY ATTRIBUTE

It is a combination of two or more Non-Key Attributes, which are used to identify the record uniquely from the table.

SUPER KEY ATTRIBUTE

It is the set of all the key attributes.



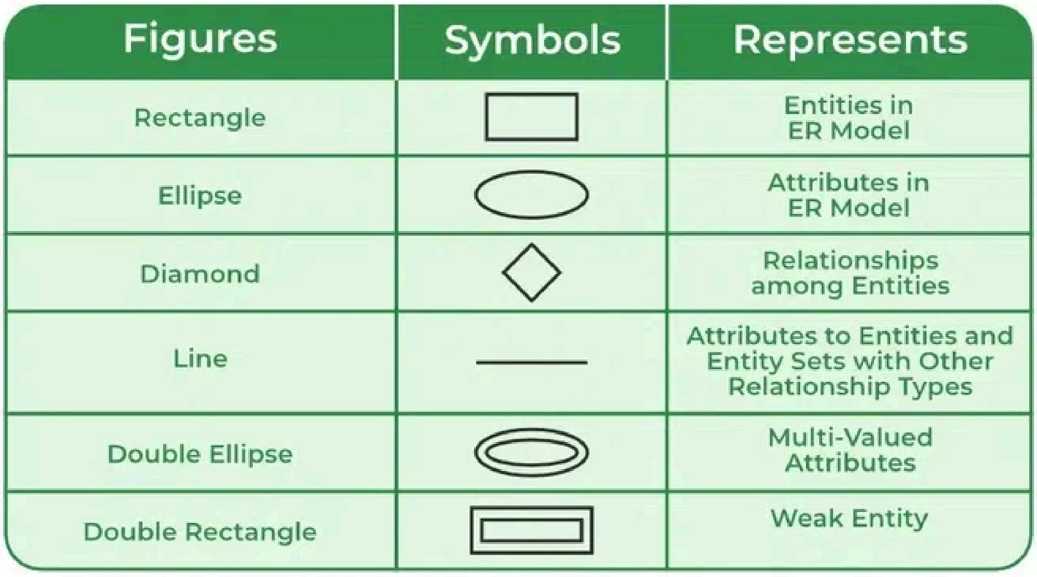
ER DIAGRAM:

The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

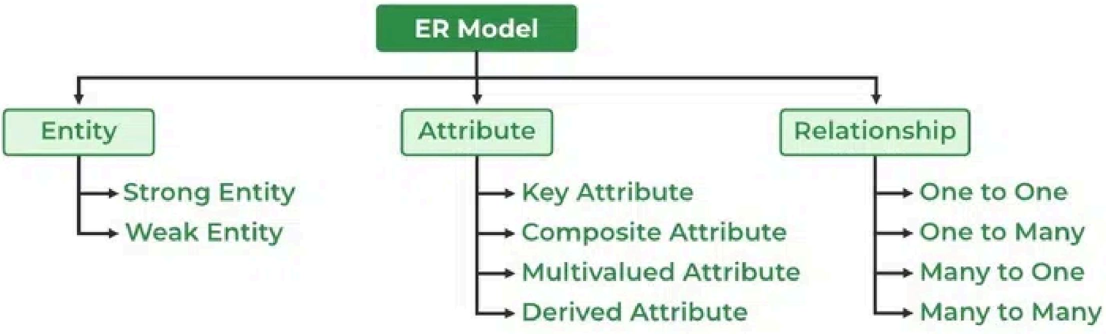
- **Rectangles:** Rectangles represent Entities in the ER Model.
- **Ellipses:** Ellipses represent Attributes in the ER Model.
- **Diamond:** Diamonds represent Relationships among Entities.
- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.
- **Double Rectangle:** Double Rectangle represents a Weak Entity.



Figures	Symbols	Represents
Rectangle	▭	Entities in ER Model
Ellipse	⬭	Attributes in ER Model
Diamond	◇	Relationships among Entities
Line	—	Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse	◎	Multi-Valued Attributes
Double Rectangle	⧈	Weak Entity

Components of ER Diagram

ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.



FUNCTIONAL DEPENDENCY:

LET US CONSIDER THE RELATION ‘R’ WITH 2 ATTRIBUTES ‘X’ & ‘Y’ RESPECTIVELY IN WHICH ATTRIBUTE ‘X’ DETERMINES ‘Y’.

IN OTHER WORDS ‘Y’ IS DEPENDENT ON ‘X’, THERE EXIST FUNCTIONAL DEPENDENCY.

R —---> {X,Y}

X —---> Y

Y IS DEPENDENT ON X.

TYPES OF FUNCTIONAL DEPENDENCY:

1. TOTAL FUNCTIONAL DEPENDENCY
2. PARTIAL FUNCTIONAL DEPENDENCY
3. TRANSITIVE FUNCTIONAL DEPENDENCY

TOTAL FUNCTIONAL DEPENDENCY

IF ALL THE ATTRIBUTES IN A RELATION IS DETERMINED BY A ATTRIBUTE WHICH IS A KEY ATTRIBUTE, THEN THERE EXIST TOTAL FUNCTIONAL DEPENDENCY.

EX:

LET US CONSIDER A RELATION R WITH 4 ATTRIBUTES A,B,C,D

IN WHICH ‘A’ IS A KEY ATTRIBUTE

R —------> {A,B,C,D}

A—--> B

A —---> C

A —---> D

THERE EXISTS TOTAL FUNCTIONAL DEPENDENCY

A —->{B,C,D}

PARTIAL FUNCTIONAL DEPENDENCY;

FOR PARTIAL FUNCTIONAL DEPENDENCY TO EXISTS THEIR MUST BE COMPOSITE KEY ATTRIBUTES.

ONE OF THE ATTRIBUTE IN COMPOSITE KEY RELATION DETERMINES ANOTHER ATTRIBUTE SEP ARATELY & THIS IS KNOWN AS PARTIAL FUNCTIONAL DEPENDENCY.

EX:

LET US CONSIDER RELATION R WITH 4 ATTRIBUTES A,B,C,D

IN WHICH ‘A’ & ‘B’ ARE COMPOSITE KEY ATTRIBUTES.

R —------> {A,B,C,D}

(A,B) —--> (C,D)

B —--> C

THERE EXISTS PARTIAL FUNCTIONAL DEPENDENCY.

<u>TRANSITIVE FUNCTIONAL DEPENDENCY:</u>

IF AN ATTRIBUTE IS DETERMINED BY A NON - KEY ATTRIBUTE WHICH IN TURN IS DETERMINED BY A KEY ATTRIBUTE, THEN THERE EXISTS TRANSITIVE FUNCTIONAL DEPENDENCY.

EX:

LET US CONSIDER RELATION ‘R’ HAVING 4 ATTRIBUTES A,B,C,D

IN WHICH ‘A’ IS A KEY ATTRIBUTE.

R —----> {A,B,C,D}

A —---> B

A—---> C

A —----> D

D —---> C

THEN THERE EXIST TRANSITIVE FUNCTIONAL DEPENDENCY.

NORMALIZATION:

- IT IS PROCESS OF REDUCING A LARGER TABLE INTO SMALLER TABLE
- IN ORDER TO REMOVE THE **REDUNDANCY & ANOMALY** BY IDENTIFYING THEIR **FUNCTIONAL DEPENDENCY**.

LEVELS OF NORMAL FORM:

1. FIRST NORMAL FORM(1NF)
2. SECOND NORMAL FORM(2NF)
3. THIRD NORMAL FORM(3NF)
4. BOYCE-CODD NORMAL FORM(BCNF)

NOTE: A TABLE IS SAID TO BE NORMALIZED IF WE REDUCE THE TABLE TO 3RD NF.

1NF:

A TABLE IS SAID TO BE 1NF, IF IT SATISFIES FOLLOWING CONDITIONS.

- A TABLE/ CELL SHOULD NOT CONSIST OF MULTI-VALUES DATA.
- A TABLE SHOULD NOT HAVE DUPLICATE OR REPEATED VALUES.

R —> {EMPNO,ENAME,SAL,COMM,DEPTNO,DNAME,LOC}

STUDENT

SID	**SNAME**	**SKILLS**
101	PINKY	JAVA
102	PONKY	SQL
103	VARSHA	MT,SQL
101	PINKY	SQL
104	SANDY	JAVA,WEB

LET’S REMOVE THE MULTIPLE VALUES FROM THE CELLS.

STUDENT

SID	**SNAME**	**SKILL1**	**SKILL2**
101	PINKY	JAVA	
102	PONKY	SQL	
103	VARSHA	MT	SQL
101	PINKY	SQL	
104	SANDY	JAVA	WEB

REMOVE THE DUPLICATED VALUES :

STUDENT

SID	**SNAME**	**SKILL1**	**SKILL2**
101	PINKY	JAVA	SQL
102	PONKY	SQL	
103	VARSHA	MT	SQL
104	SANDY	JAVA	WEB

NOW THE TABLE IS IN 1NF.

2NF:

A TABLE IS SAID TO BE 2NF, IT IT SATISFIES THE FOLLOWING CONDITIONS.

- THE TABLE SHOULD BE IN 1NF
- THE TABLE SHOULD NOT HAVE PARTIAL FD.

CONSIDER THIS IS IN 1NF,

R —-> {EMPNO,ENAME,SAL,COMM,DEPTNO,DNAME,LOC}

EMPNO —> {ENAME,SAL,COMM,DEPTNO}

DEPTNO —-> {DNAME,LOC}

R1—-> {EMPNO,ENAME,SAL,COMM}

R2 —-> {DEPTNO,DNAME,LOC}

NOTE :

IF THE TABLE CONSIST OF PARTIAL FD, THEN THE ATTRIBUTES RESPONSIBLE ARE REMOVED FROM THE TABLE.

EX:

COURSE :

PROF_ID	**SUBJECT**	**FEES**
101	PHYSICS	5K
103	LAW	6K
102	ECONOMICS	6.5K
103	POLITICS	5.5K
101	CHEMISTRY	5K
102	STATISTICS	6.5K
105	ECONOMICS	6.5K
104	PHYSICS	5K

THE ABOVE TABLE IS NOT IN 2NF, WE HAVE PARTIAL FUNCTIONAL DEPENDENCY, LETS DIVIDE THE TABLE INTO 2 PARTS.

PROF_COURSE

PROF_ID	**SUBJECT**
101	PHYSICS
101	CHEMISTRY
102	ECONOMICS
102	STATISTICS
103	POLITICS
104	PHYSICS
105	ECONOMICS

FEE_COURSE

SUBJECT	**FEE**
PHYSICS	5K
CHEMISTRY	5K
ECONOMICS	6.5K
STATISTICS	6.5K
LAW	6K
POLITICS	5.5K

NOW THE TABLE IS IN 2NF.

<u>3NF:</u>

A TABLE IS SAID TO BE 3NF, IT IT SATISFIES THE FOLLOWING CONDITIONS.

- THE TABLE SHOULD BE 2NF
- THE TABLE SHOULD NOT HAVE TRANSITIVE FD.

CONSIDER THIS IS IN 2NF,

R —-> {EMPNO,ENAME,SAL,PINCODE,STATE}

```
EMPNO —-> ENAME
          SAL
          PINCODE
               —--> STATE
```

R1 —> {EMPNO,ENAME,SAL}

R2 —-> {PINCODE,STATE}

<u>NOTE:</u>

IF THE TABLE CONSIST OF TRANSITIVE FD, THEN THE ATTRIBUTES RESPONSIBLE ARE REMOVED FROM THE TABLE.

EX:

EMP

EID	**ENAME**	**JOB**	**CITY**	**ZIPCODE**
101	ANAND	TE	MUMBAI	400004
102	POOJA	DE	HYDERABAD	500002
103	AMAN	TL	JAIPUR	302002
104	SHRADHA	BA	MUMBAI	400004
105	VINAY	DE	DELHI	110002
106	RAJU	TL	HYDERABAD	500002

EMP

EID	**ENAME**	**JOB**	**CITY**
101	ANAND	TE	MUMBAI
102	POOJA	DE	HYDERABAD
103	AMAN	TL	JAIPUR
104	SHRADHA	BA	MUMBAI
105	VINAY	DE	DELHI
106	RAJU	TL	HYDERABAD

CITY

ZIPCODE	**CITY**
400004	MUMBAI
500002	HYDERABAD
302002	JAIPUR
110002	DELHI

NOW THE TABLE IS IN 3NF.

TABLE IS NORMALIZED, BECAUSE IF WE REDUCE TABLE TO 3NF WE CAN SAY THAT TABLE IS NORMALIZED.

BCNF :

- THE TABLE SHOULD BE IN 3NF
- A RELATION IS IN BCNF IF FOR EVERY FUNCTIONAL DEPENDENCY X—>Y , X IS A SUPER KEY.

EX:

EMP

EID	**COUNTRY**	**DEPARTMENT**	**D_TYPE**	**DEPTNO**
101	JAPAN	MARKETING	M_INTERN	M_01
102	INDIA	DEVELOPMENT	D_INTERN	D_02
103	CANADA	HR	H_INTERN	H_03
104	JAPAN	SALES	S_INTERN	S_04

FUNCTIONAL DEPENDENCIES:

- EID —> COUNTRY
- DEPARTMENT —> D_TYPE
- DEPARTMENT —> DEPTNO

- EID —> DEPARTMENT

{EID, DEPARTMENT} : ARE THE KEYS (SUPER KEY)

T1

EID	**COUNTRY**
101	JAPAN
102	INDIA
103	CANADA
104	JAPAN

T2

DEPARTMENT	**D_TYPE**	**DEPTNO**
MARKETING	M_INTERN	M_01
DEVELOPMENT	D_INTERN	D_02
HR	H_INTERN	H_03
SALES	S_INTERN	S_04

T3

EID	**DEPARTMENT**
101	MARKETING
102	DEVELOPMENT
103	HR

104	SALES

LHS OF EACH **FUNCTIONAL DEPENDENCY** SHOULD BE A **SUPER KEY** , FOR THE TABLE TO BE IN **BCNF.**

VIEW

VIEWS ARE THE VIRTUAL TABLE WHICH CAN BE CREATED AND

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

SYNTAX

```
CREATE VIEW view_name

AS

SELECT Stmt;
```

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,

SO FOR EASY OF ACCESS, WE USE INDEX.

INDEX ARE OF 2 TYPES:

1. CLUSTERED

2. NON-CLUSTERED

CREATE INDEX Syntax

```
CREATE INDEX index_name

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

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;
```

POINTS TO REMEMBER ABOUT INDEXES:

- 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.

PROCEDURE/STORED PROCEDURE

What is a stored procedure in SQL?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

CURSOR IN SQL:

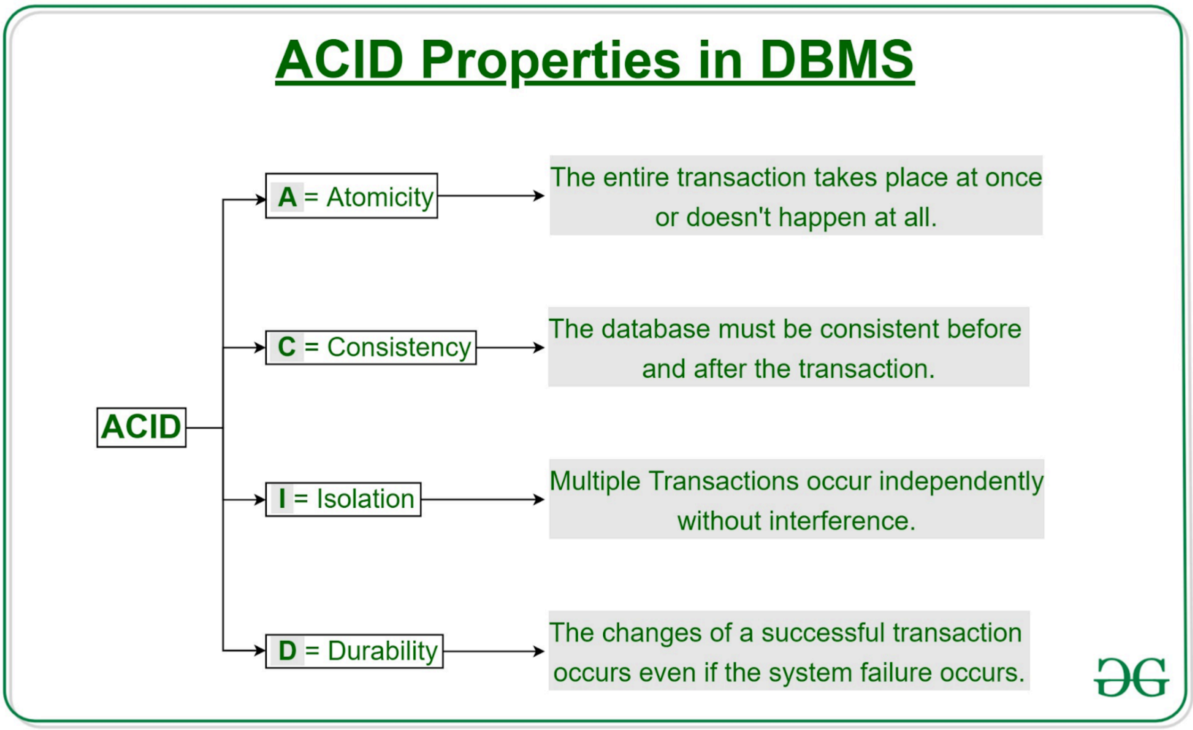
Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on the Table by the User. Cursors are used to store Database Tables.

There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:** Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors:** Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

ACID PROPERTY IN SQL :

A **transaction** is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



ACID Properties in DBMS

ACID

A = Atomicity: The entire transaction takes place at once or doesn't happen at all.

C = Consistency: The database must be consistent before and after the transaction.

I = Isolation: Multiple Transactions occur independently without interference.

D = Durability: The changes of a successful transaction occurs even if the system failure occurs.