Unit IV (Essential of Information Technology)

1. What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like 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.

Advantages of relational database management system

The use of an RDBMS can be beneficial to most organizations; the systematic view of raw data helps companies better understand and execute the information while enhancing the decision-making process. The use of tables to store data also improves the security of information stored in the databases. Users are able to customize access and set barriers to limit the content that is made available. This feature makes the RDBMS particularly useful to companies in which the manager decides what data is provided to employees and customers.

Furthermore, RDBMSes make it easy to add new data to the system or alter existing tables while ensuring consistency with the previously available content.

Other advantages of the RDBMS include:

Flexibility -- updating data is more efficient since the changes only need to be made in one place.

Maintenance -- database administrators can easily maintain, control and update data in the database. Backups also become easier since automation tools included in the RDBMS automate these tasks.

Data structure -- the table format used in RDBMSes is easy to understand and provides an organized and structural manner through which entries are matched by firing queries.

On the other hand, relational database management systems do not come without their disadvantages. For example, in order to implement an RDBMS, special

software must be purchased. This introduces an additional cost for execution. Once the software is obtained, the setup process can be tedious since it requires millions of lines of content to be transferred into the RDBMS tables. This process may require the additional help of a programmer or a team of data entry specialists. Special attention must be paid to the data during entry to ensure sensitive information is not placed into the wrong hands.

Some other drawbacks of the RDBMS include the character limit placed on certain fields in the tables and the inability to fully understand new forms of data -- such as complex numbers, designs and images.

Furthermore, while isolated databases can be created using an RDBMS, the process requires large chunks of information to be separated from each other. Connecting these large amounts of data to form the isolated database can be very complicated.

Uses of RDBMS

Relational database management systems are frequently used in disciplines such as manufacturing, human resources and banking. The system is also useful for airlines that need to store ticket service and passenger documentation information as well as universities maintaining student databases.

Some examples of specific systems that use RDBMS include IBM, Oracle, MySQL, Microsoft SQLServer and PostgreSQL.

What is 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.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
```

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

What is 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.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table −

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is a column?

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

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below −

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
```

```
+----+-------+
```

What is 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.

It is very important to understand that 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.

2. 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.

Following are some of the most commonly used constraints available in SQL −

- 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.
- 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.

3. 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.

4. Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Fourth Normal form
- BCNF
- Fifth Normal form

First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390

102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212

102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38

111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:
teacher_details table:

teacher_id	teacher_age
111	38
222	38

333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp

1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores

1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

5. 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

6. Data Model

Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the data elements and the relationships between the data elements. Data Models are used to show how data is stored, connected, accessed and updated in the database management system. Here, we use a set of symbols and text to represent the information so that members of the organisation can communicate and understand it. Though there are many data models being used nowadays but the Relational model is the most widely used model. Apart from the Relational model, there are many other types of data models about which we will study in details in this blog. Some of the Data Models in DBMS are:

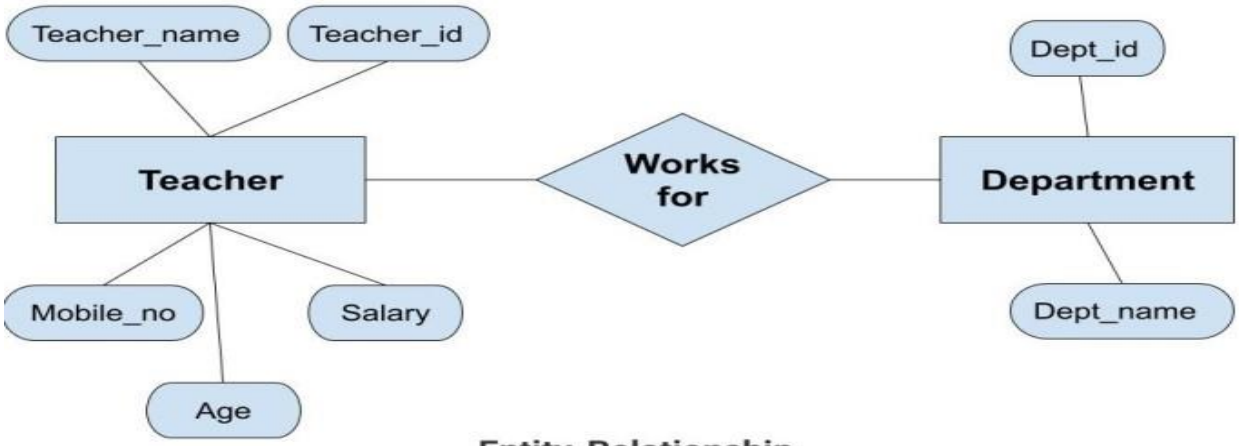
1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object-Oriented Data Model
6. Object-Relational Data Model
7. Flat Data Model
8. Semi-Structured Data Model
9. Associative Data Model
10. Context Data Model

Entity-Relationship Model

Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

- ***Entities:*** Entity is a real-world thing. It can be a person, place, or even a concept. *Example:* Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
- ***Attributes:*** An entity contains a real-world property called attribute. This is the characteristics of that attribute. *Example:* The entity teacher has the property like teacher id, salary, age, etc.
- ***Relationship:*** Relationship tells how two attributes are related. *Example:* Teacher works for a department.

Example:



Entity-Relationship Model

In the above diagram, the entities are Teacher and Department. The attributes of ***Teacher*** entity are Teacher_Name, Teacher_id, Age, Salary, Mobile_Number. The attributes of entity ***Department*** entity are Dept_id, Dept_name. The two entities are connected using the relationship. Here, each teacher works for a department.

Features of ER Model

- ***Graphical Representation for Better Understanding:*** It is very easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
- ***ER Diagram:*** ER diagram is used as a visual tool for representing the model.
- ***Database Design:*** This model helps the database designers to build the database and is widely used in database design.

Advantages of ER Model

- ***Simple:*** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
- ***Effective Communication Tool***: This model is used widely by the database designers for communicating their ideas.
- ***Easy Conversion to any Model***: This model maps well to the relational model and can be easily converted relational model by converting the ER

model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

Disadvatages of ER Model

- ***No industry standard for notation:*** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
- ***Hidden information:*** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

Relational Model

Relational Model is the most widely used model. In this model, the data is maintained in the form of a two-dimensional table. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also called *relations* in the relational model. ***Example:*** In this example, we have an Employee table.

Emp_id	Emp_name	Job_name	Salary	Mobile_no	Dep_id	Project_id
AfterA001	John	Engineer	100000	9111037890	2	99
AfterA002	Adam	Analyst	50000	9587569214	3	100
AfterA003	Kande	Manager	890000	7895212355	2	65

EMPLOYEE TABLE

Features of Relational Model

- ***Tuples***: Each row in the table is called tuple. A row contains all the information about any instance of the object. In the above example, each row has all the information about any specific individual like the first row has information about John.

- ***Attribute or field:*** Attributes are the property which defines the table or relation. The values of the attribute should be from the same domain. In the above example, we have different attributes of the *employee* like Salary, Mobile_no, etc.

Advnatages of Relational Model

- ***Simple:*** This model is more simple as compared to the network and hierarchical model.
- ***Scalable:*** This model can be easily scaled as we can add as many rows and columns we want.
- ***Structural Independence:*** We can make changes in database structure without changing the way to access the data. When we can make changes to the database structure without affecting the capability to DBMS to access the data we can say that structural independence has been achieved.

Disadvantages of Relatinal Model

- ***Hardware Overheads:*** For hiding the complexities and making things easier for the user this model requires more powerful hardware computers and data storage devices.
- ***Bad Design:*** As the relational model is very easy to design and use. So the users don't need to know how the data is stored in order to access it. This ease of design can lead to the development of a poor database which would slow down if the database grows.

But all these disadvantages are minor as compared to the advantages of the relational model. These problems can be avoided with the help of proper implementation and organisation.

Entity Relationship Diagram – ER Diagram in DBMS

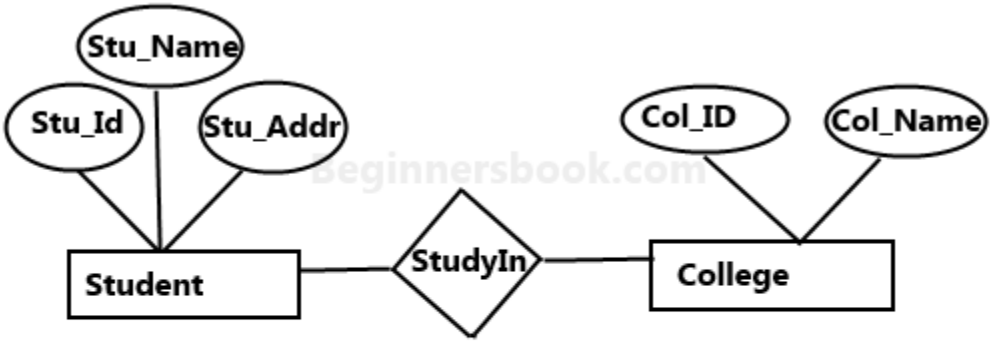
An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



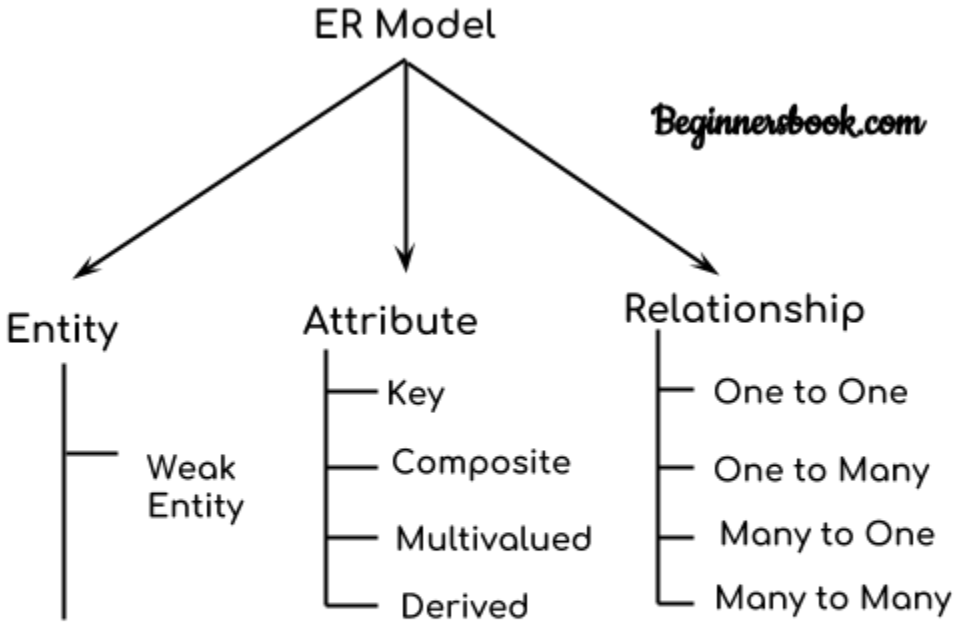
Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram



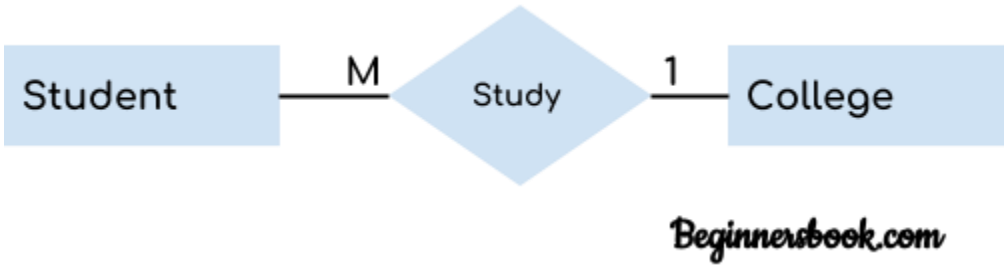
Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students

study in a single college. We will read more about relationships later, for now focus on entities.



Weak Entity:
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

2. Attribute

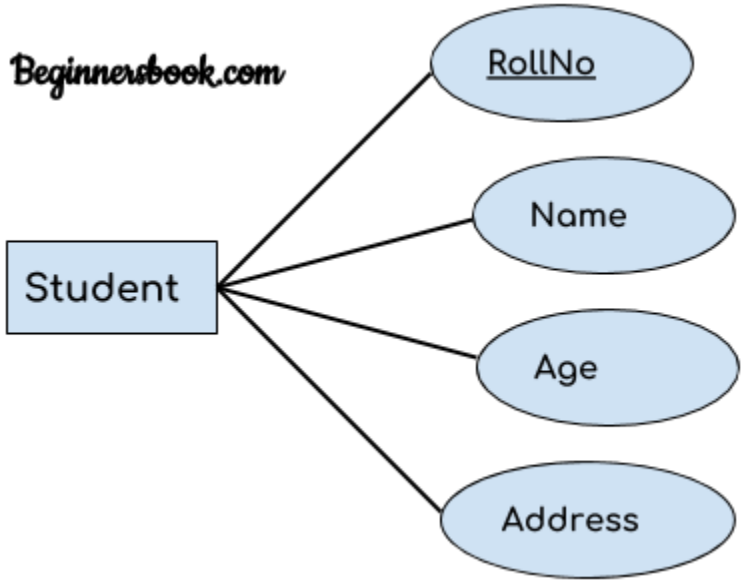
An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:

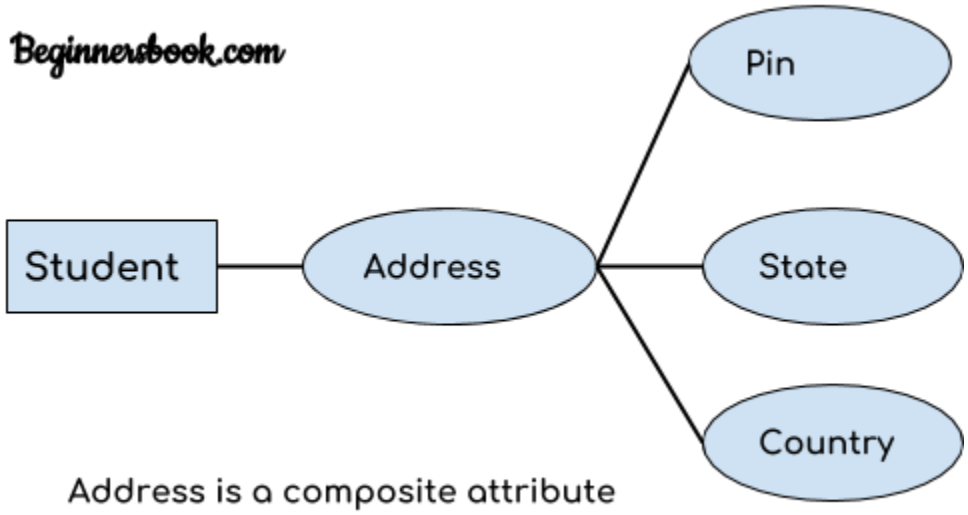
A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key

attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.



2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



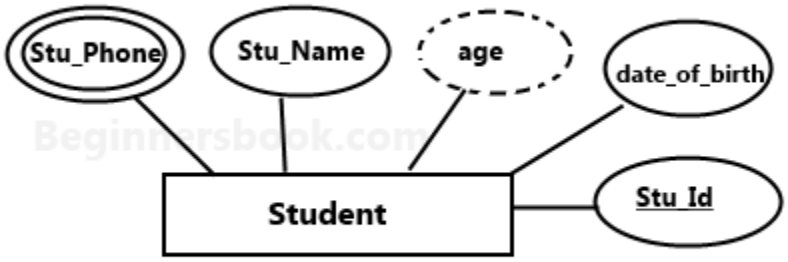
3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:



3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one

passport and a passport is given to one person.

Beginnersbook.com

2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

Beginnersbook.com

3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

Beginnersbook.com

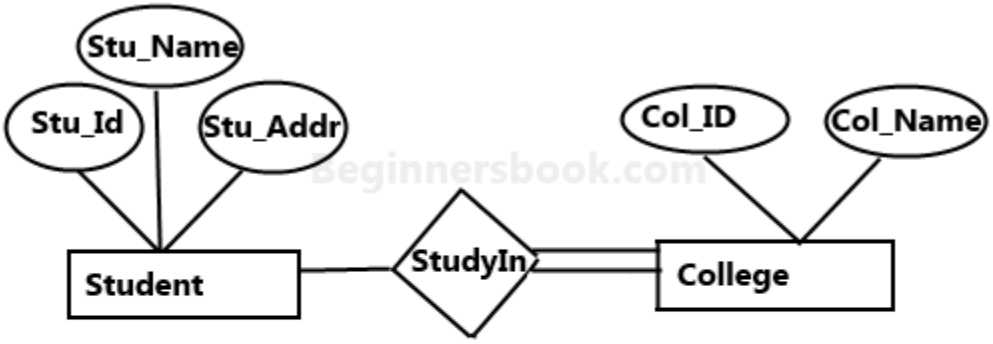
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.

Beginnersbook.com

Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

7. Converting ER Diagrams to Tables-

Before you go through this article, make sure that you have gone through the previous article on **ER Diagrams to Tables**.

After designing an **ER Diagram**,

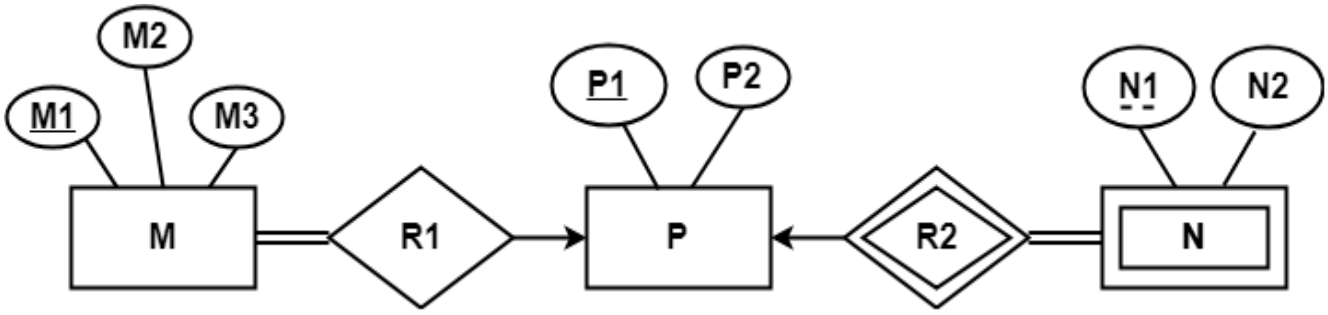
- ER diagram is converted into the tables in relational model.
- This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.
- The rules used for converting an ER diagram into the tables are already discussed.

In this article, we will discuss practice problems based on converting ER Diagrams to Tables.

PRACTICE PROBLEMS BASED ON CONVERTING ER DIAGRAM TO TABLES-

Problem-01:

Find the minimum number of tables required for the following ER diagram in relational model-



Solution-

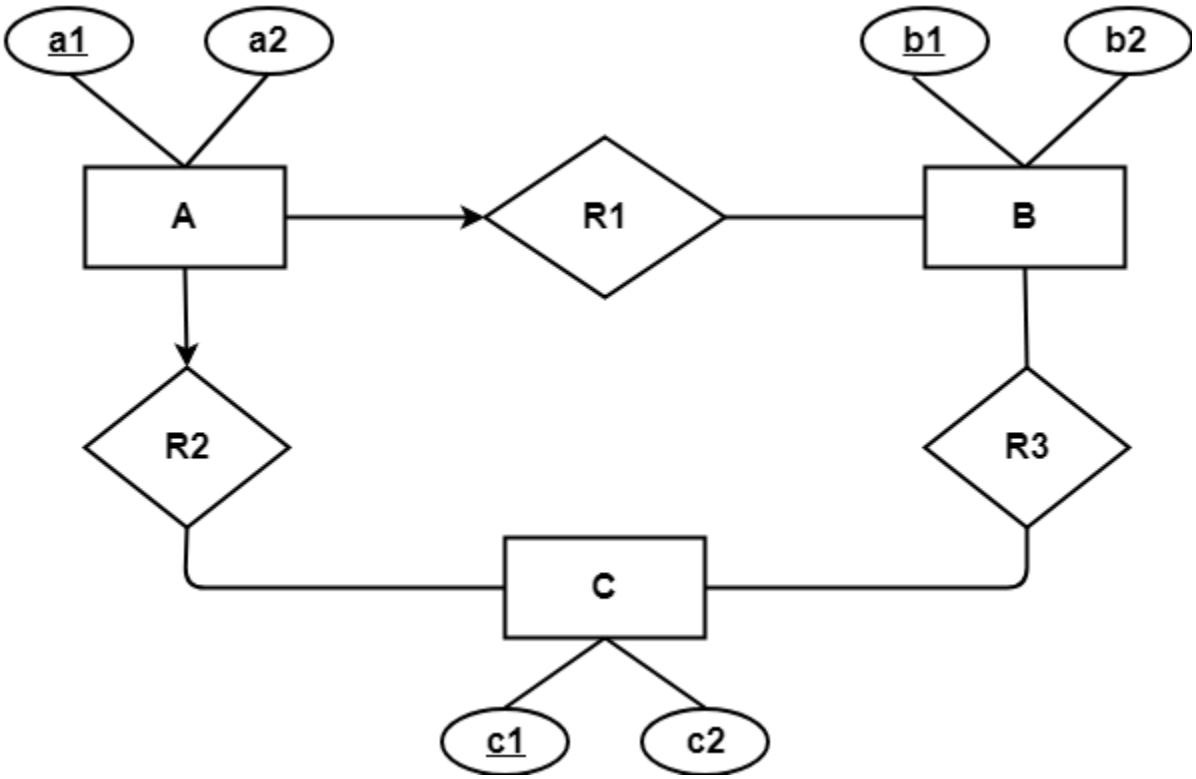
Applying the rules, minimum 3 tables will be required-

- MR1 (M1 , M2 , M3 , P1)
- P (P1 , P2)

- NR2 (P1 , N1 , N2)

Problem-02:

Find the minimum number of tables required to represent the given ER diagram in relational model-



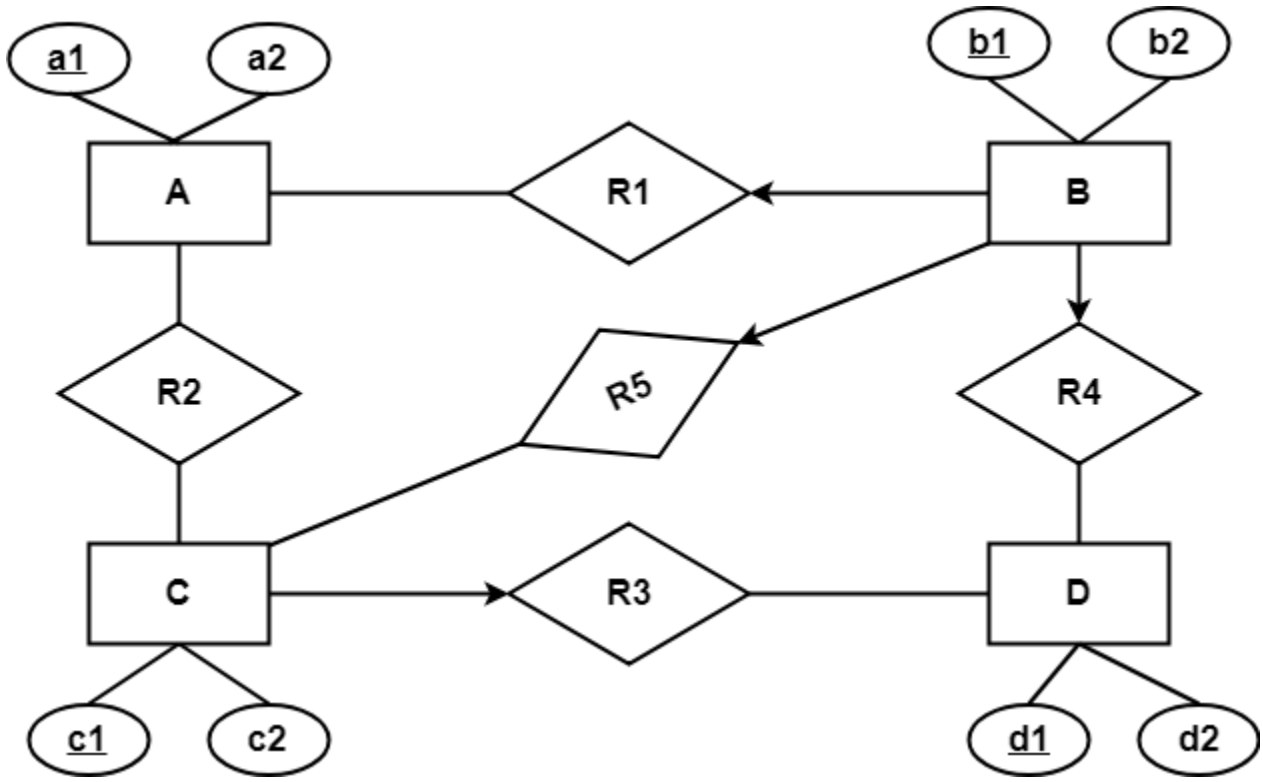
Solution-

Applying the rules, minimum 4 tables will be required-

- AR1R2 (a1 , a2 , b1 , c1)
- B (b1 , b2)
- C (c1 , c2)
- R3 (b1 , c1)

Problem-03:

Find the minimum number of tables required to represent the given ER diagram in relational model-



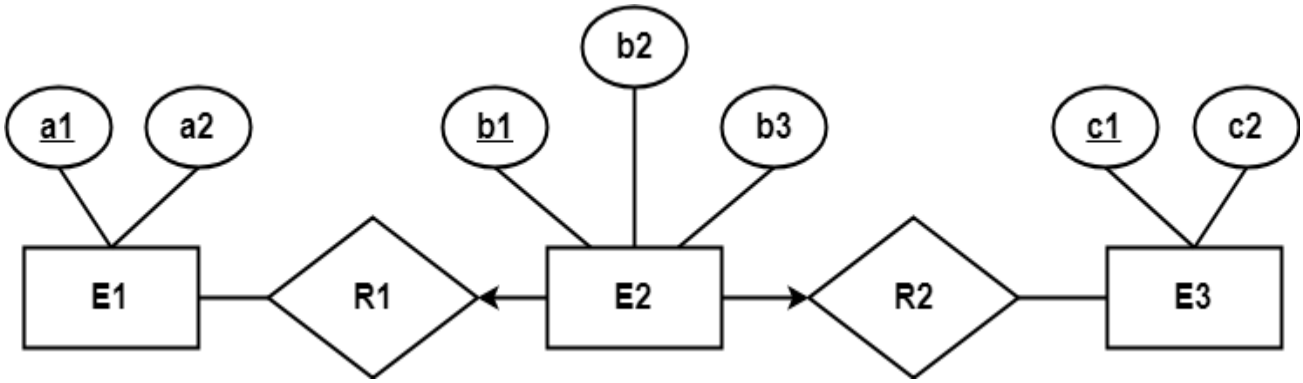
Solution-

Applying the rules, minimum 5 tables will be required-

- BR1R4R5 (b1 , b2 , a1 , c1 , d1)
- A (a1 , a2)
- R2 (a1 , c1)
- CR3 (c1 , c2 , d1)
- D (d1 , d2)

Problem-04:

Find the minimum number of tables required to represent the given ER diagram in relational model-



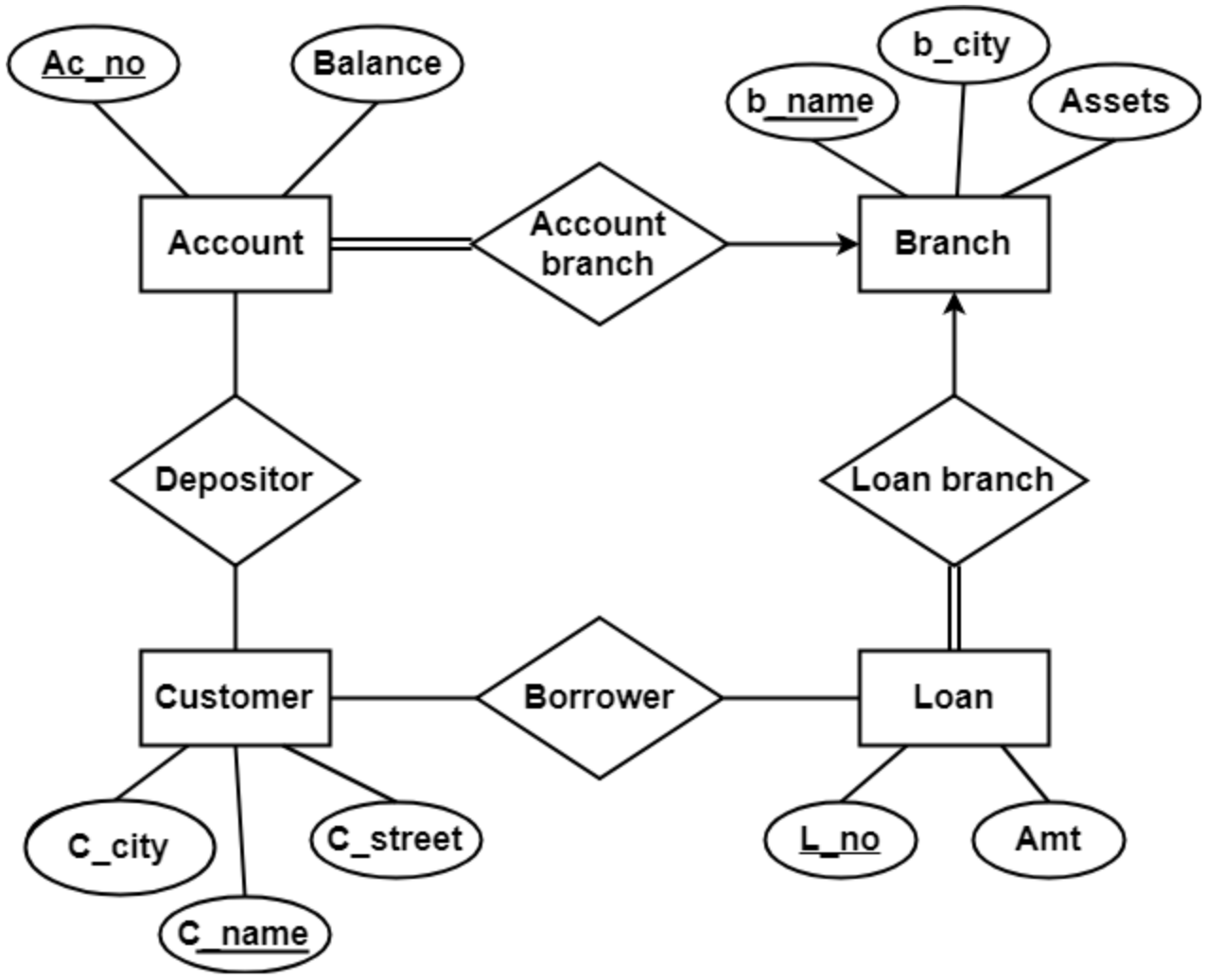
Solution-

Applying the rules, minimum 3 tables will be required-

- E1 (a1 , a2)
- E2R1R2 (b1 , b2 , a1 , c1 , b3)
- E3 (c1 , c2)

Problem-05:

Find the minimum number of tables required to represent the given ER diagram in relational model-



Solution-

Applying the rules that we have learnt, minimum 6 tables will be required-

- Account (Ac_no , Balance , b_name)
- Branch (b_name , b_city , Assets)
- Loan (L_no , Amt , b_name)
- Borrower (C_name , L_no)
- Customer (C_name , C_street , C_city)
- Depositor (C_name , Ac_no)

.

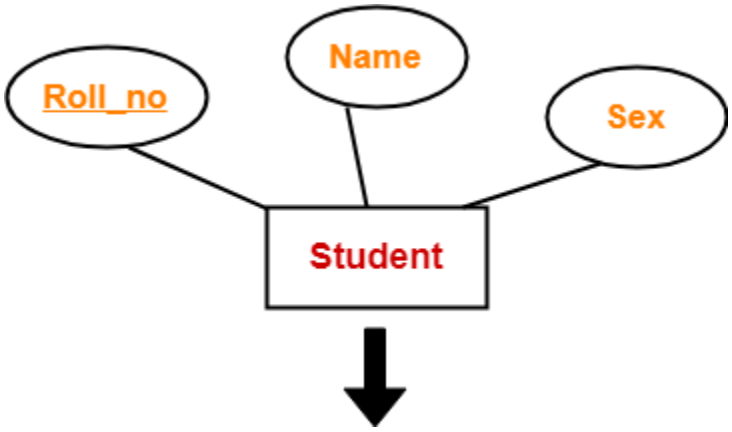
Following rules are used for converting an ER diagram into the tables-

Rule-01: For Strong Entity Set With Only Simple Attributes-

A strong entity set with only simple attributes will require only one table in relational model.

- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.

Example-



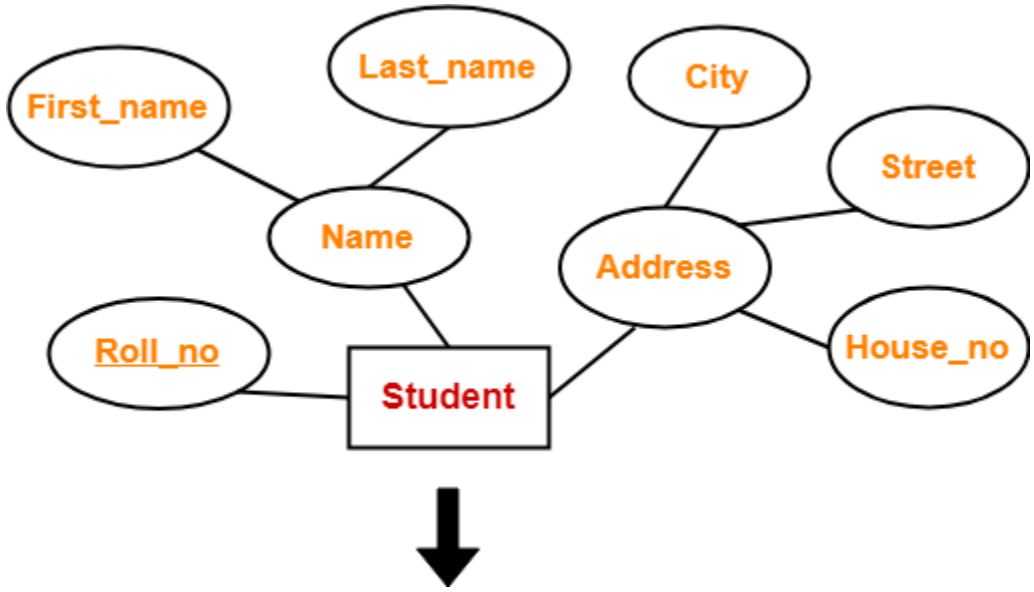
Roll_no	Name	Sex

Schema : Student (Roll_no , Name , Sex)

Rule-02: For Strong Entity Set With Composite Attributes-

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.

Example-



Roll_no	First_name	Last_name	House_no	Street	City

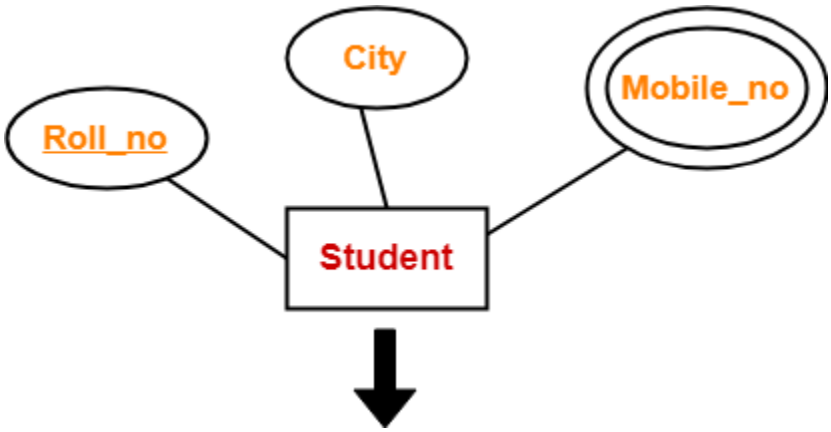
Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

Rule-03: For Strong Entity Set With Multi Valued Attributes-

A strong entity set with any number of multi valued attributes will require two tables in relational model.

- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multi valued attributes.

Example-



Roll_no	City

Roll_no	Mobile_no

Rule-04: Translating Relationship Set into a Table-

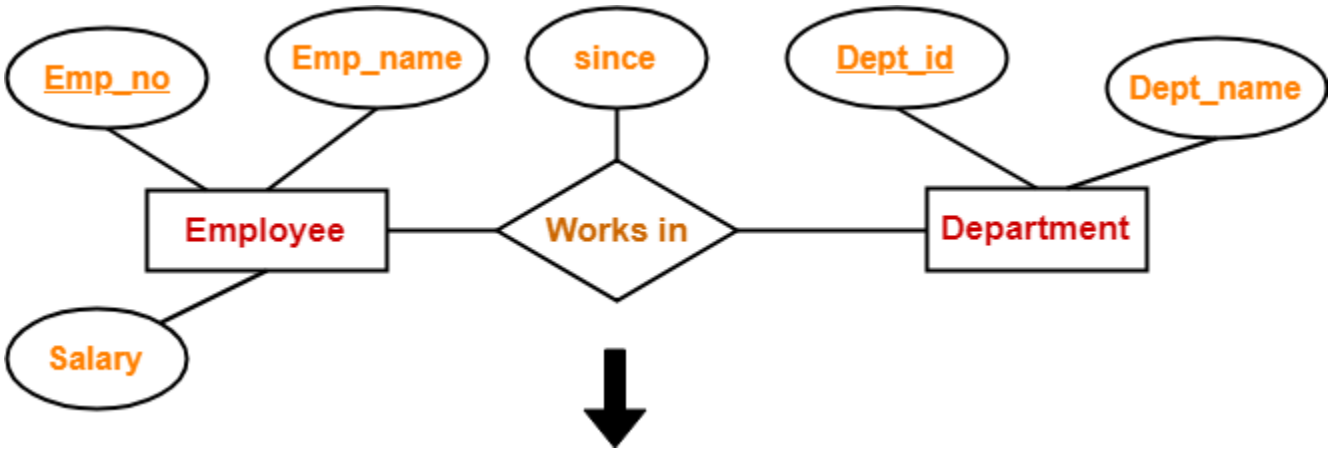
A relationship set will require one table in the relational model.

Attributes of the table are-

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.

Set of non-descriptive attributes will be the primary key.

Example-



Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

NOTE-

If we consider the overall ER diagram, three tables will be required in relational model-

- One table for the entity set “Employee”
- One table for the entity set “Department”
- One table for the relationship set “Works in”

Rule-05: For Binary Relationships With Cardinality Ratios-

The following four cases are possible-

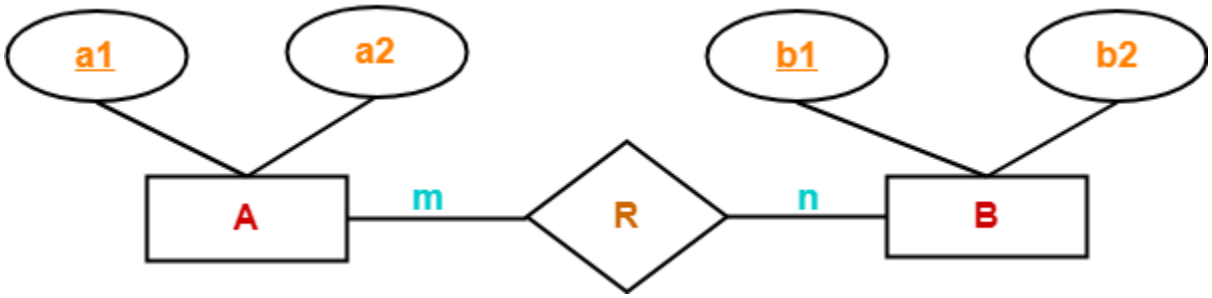
Case-01: Binary relationship with cardinality ratio m:n

Case-02: Binary relationship with cardinality ratio 1:n

Case-03: Binary relationship with cardinality ratio m:1

Case-04: Binary relationship with cardinality ratio 1:1

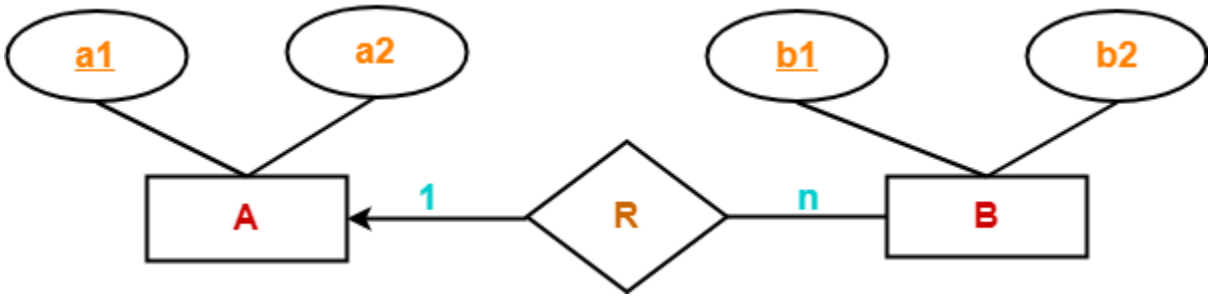
Case-01: For Binary Relationship With Cardinality Ratio m:n



Here, three tables will be required-

1. A (a1 , a2)
2. R (a1 , b1)
3. B (b1 , b2)

Case-02: For Binary Relationship With Cardinality Ratio 1:n

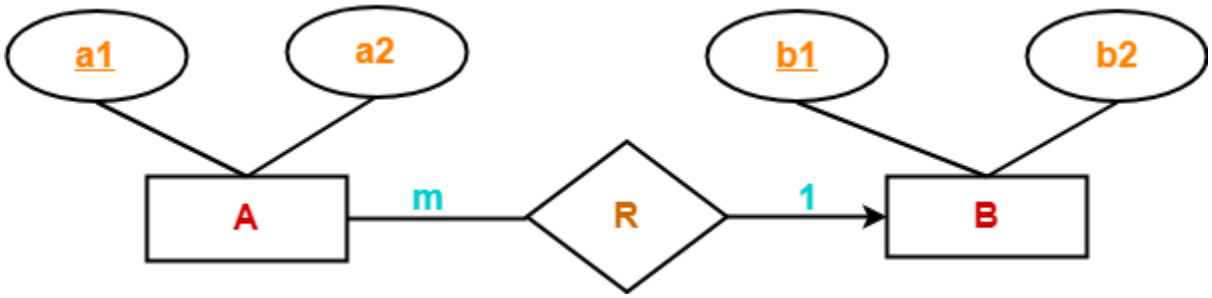


Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

NOTE- Here, combined table will be drawn for the entity set B and relationship set R.

Case-03: For Binary Relationship With Cardinality Ratio m:1

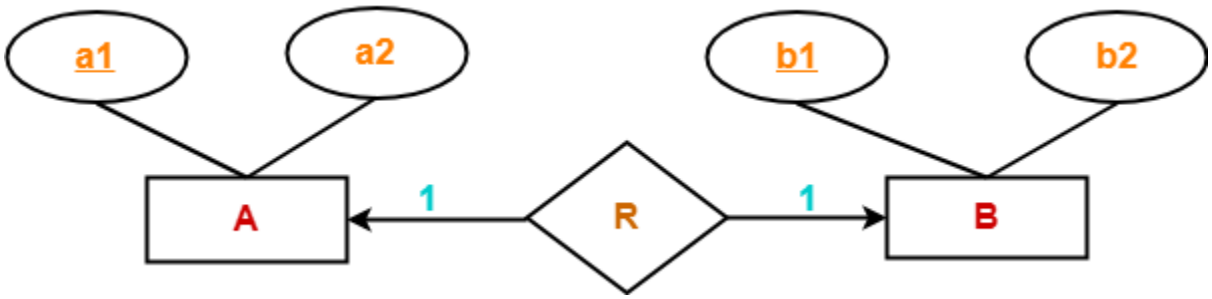


Here, two tables will be required-

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

NOTE- Here, combined table will be drawn for the entity set A and relationship set R.

Case-04: For Binary Relationship With Cardinality Ratio 1:1



Here, two tables will be required. Either combine 'R' with 'A' or 'B'

Way-01:

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

Way-02:

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Thumb Rules to Remember

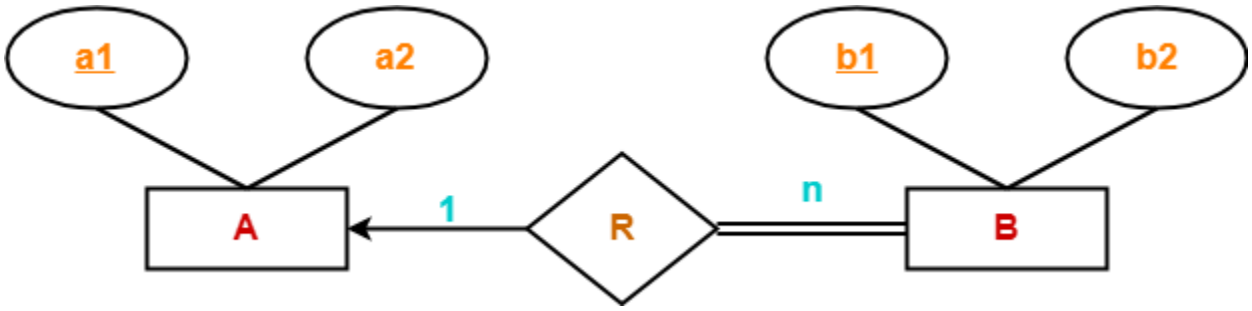
While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-

- For binary relationship with cardinality ration m : n , separate and individual tables will be drawn for each entity set and relationship.
- For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember "many side will consume the relationship" i.e. a combined table will be drawn for many side entity set and relationship set.
- For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets.

Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints-

- Cardinality constraints will be implemented as discussed in Rule-05.
- Because of the total participation constraint, foreign key acquires **NOT NULL** constraint i.e. now foreign key can not be null.

Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side-



Because cardinality ratio = 1 : n , so we will combine the entity set B and relationship set R.

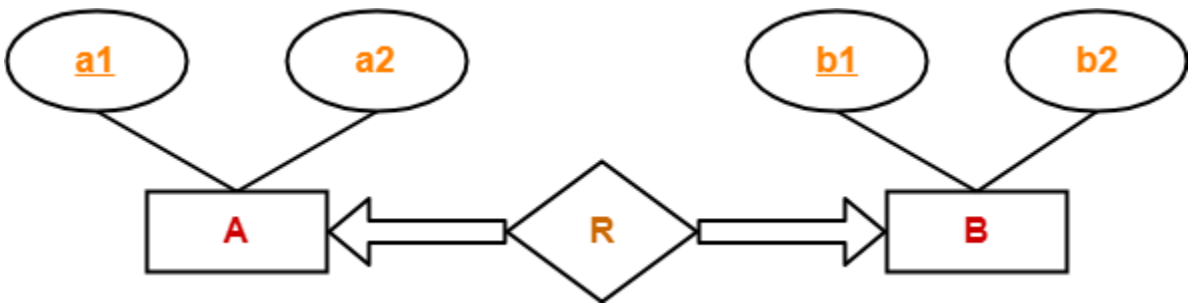
Then, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can't be null now.

Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides-

If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.

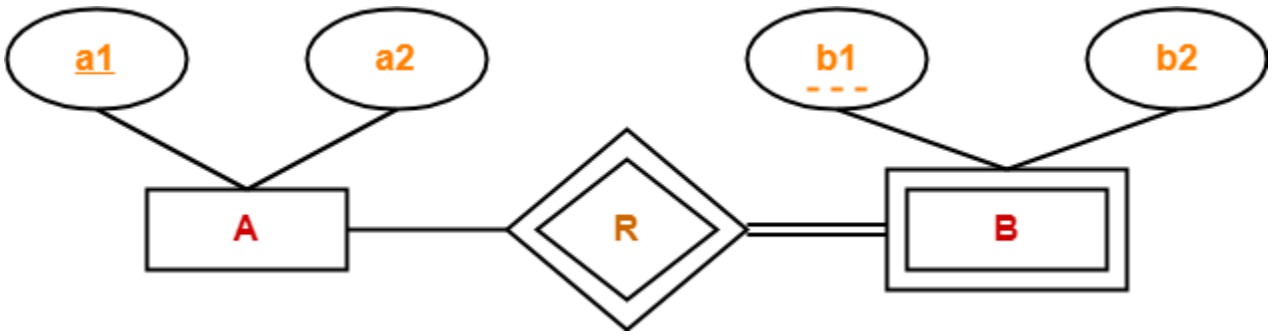


Here, Only one table is required.

- ARB (a1 , a2 , b1 , b2)

Rule-07: For Binary Relationship With Weak Entity Set-

Weak entity set always appears in association with identifying relationship with total participation constraint.



Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

SQL: SQL consist of following four types of sub-languages

- **Data Definition Language(DDL)** – Consists of commands which are used to define the database.
- **Data Manipulation Language(DML)** – Consists of commands which are used to manipulate the data present in the database.
- **Data Control Language(DCL)** – Consists of commands which deal with the user permissions and controls of the database system.
- **Transaction Control Language(TCL)** – Consist of commands which deal with the transaction of the database.

Apart from the above commands, the following topics will also be covered in this article:

- Comments in SQL
- Different Types Of Keys In Database
- Constraints Used In Database

- Nested Queries
- Joins
- Set Operations
- Dates & Auto Increment
- Views
- Stored Procedures
- Triggers

In this article on SQL Commands, I am going to consider the below database as an example, to show you how to write commands.

EmployeeID	EmployeeName	Emergency ContactName	PhoneNumber	Address	City	Country
01	Shanaya	Abhinay	9898765612	Oberoi Street 23	Mumbai	India
02	Anay	Soumya	9432156783	Marathalli House No 23	Delhi	India
03	Preeti	Rohan	9764234519	Queens Road 45	Bangalore	India
04	Vihaan	Akriti	9966442211	Brigade Road Block 4	Hyderabad	India
05	Manasa	Shourya	9543176246	Mayo Road 23	Kolkata	India

So, let's get started now!

Comments in SQL

There are two ways in which you can comment in SQL, i.e. either the Single-Line Comments or the Multi-Line Comments.

Single-Line Comments

The single line comment starts with two hyphens (–). So, any text mentioned after (–), till the end of a single line will be ignored by the compiler.

Example:

```
1                                    --Select all:
```

```
2                                   SELECT * FROM Employee_Info;
```

Multi-Line Comments

The Multi-line comments start with **/*** and end with ***/**. So, any text mentioned between /* and */ will be ignored by the compiler.

Example:

```
1                                   /*Select all the columns
2                                      of all the records
3                                   from the Employee_Info table:*/
4                                      SELECT * FROM Students;
```

SQL Commands: Data Definition Language Commands (DDL)

This section of the article will give you an insight into the commands through which you can define your database. The commands are as follows:

-
- CREATE
- DROP
- TRUNCATE
- ALTER
- BACKUP DATABASE

CREATE

This statement is used to create a table or a database.

The 'CREATE DATABASE' Statement

As the name suggests, this statement is used to create a database.

Syntax

```
CREATE DATABASE DatabaseName;
```

Example

```
1                                   CREATE DATABASE Employee;
```

The 'CREATE TABLE' Statement

This statement is used to create a table.

Syntax

```
CREATE TABLE TableName (
Column1 datatype,
Column2 datatype,
```

```
Column3 datatype,
....

ColumnN datatype
);
```

Example

```
CREATE TABLE Employee_Info
(
EmployeeID int,
EmployeeName varchar(255),
Emergency ContactName varchar(255),
PhoneNumber int,
Address varchar(255),
City varchar(255),
Country varchar(255)
);
```

You can also create a table using another table. Refer the below sytax and example:

The 'CREATE TABLE AS' Statement

Syntax

```
CREATE TABLE NewTableName AS
SELECT Column1, column2,..., ColumnN
FROM ExistingTableName
WHERE ....;
```

Example

```
CREATE TABLE ExampleTable AS
SELECT EmployeeName, PhoneNumber
FROM Employee_Info;
```

DROP

This statement is used to drop an existing table or a database.

The 'DROP DATABASE' Statement

This statement is used to drop an existing database. When you use this statement, complete information present in the database will be lost.

Syntax

```
DROP DATABASE DatabaseName;
```

Example

```
1                                   DROP DATABASE Employee;
```

The 'DROP TABLE' Statement

This statement is used to drop an existing table. When you use this statement, complete information present in the table will be lost.

Syntax

```
DROP TABLE TableName;
```

Example

```
1                                   DROP Table Employee_Info;
```

TRUNCATE

This command is used to delete the information present in the table but does not delete the table. So, once you use this command, your information will be lost, but not the table.

Syntax

```
TRUNCATE TABLE TableName;
```

Example

```
1                                   TRUNCATE Table Employee_Info;
```

ALTER

This command is used to delete, modify or add constraints or columns in an existing table.

The 'ALTER TABLE' Statement

This statement is used to add, delete, modify columns in an existing table.

The 'ALTER TABLE' Statement with ADD/DROP COLUMN

You can use the ALTER TABLE statement with ADD/DROP Column command according to your need. If you wish to add a column, then you will use the ADD command, and if you wish to delete a column, then you will use the DROP COLUMN command.

Syntax

```
ALTER TABLE TableName
ADD ColumnName Datatype;

ALTER TABLE TableName
DROP COLUMN ColumnName;
```

Example

```
                                    --ADD Column BloodGroup:
1
2
                                    ALTER TABLE Employee_Info
```

```
3                                   ADD BloodGroup varchar(255);
4
5                                   --DROP Column BloodGroup:
6
7                                   ALTER TABLE Employee_Info
8                                   DROP COLUMN BloodGroup ;
9
```

The 'ALTER TABLE' Statement with ALTER/MODIFY COLUMN

This statement is used to change the datatype of an existing column in a table.

Syntax

```
ALTER TABLE TableName
ALTER COLUMN ColumnName Datatype;
```

Example

```
1                   --Add a column DOB and change the data type to Date.
2
3                                   ALTER TABLE Employee_Info
4                                         ADD DOB year;
5
6                                   ALTER TABLE Employee_Info
7                                         ALTER DOB date;
```

BACKUP DATABASE

This statement is used to create a full backup of an existing database.

Syntax

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath';
```

Example

```
1                                   BACKUP DATABASE Employee
2                                 TO DISK = 'C:UsersSahitiDesktop';
```

You can also use a ***differential back up.*** This type of back up only backs up the parts of the database, which have changed since the last complete backup of the database.

Syntax

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

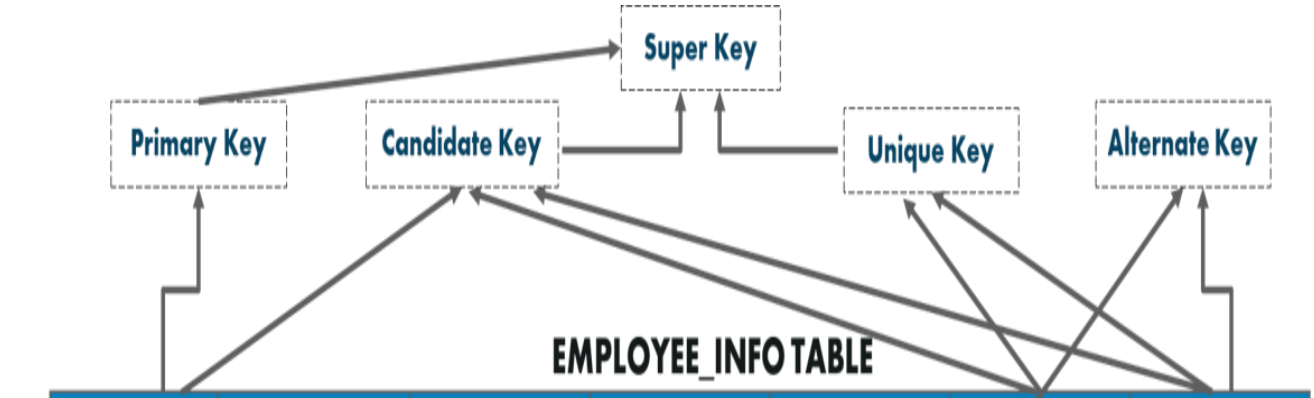
Example

```
1    BACKUP DATABASE Employee
2    TO DISK = 'C:UsersSahitiDesktop'
3    WITH DIFFERENTIAL;
```

Now that you know the data definition commands, let me take you through the various types of Keys and Constraints that you need to understand before learning how to manipulate the databases.

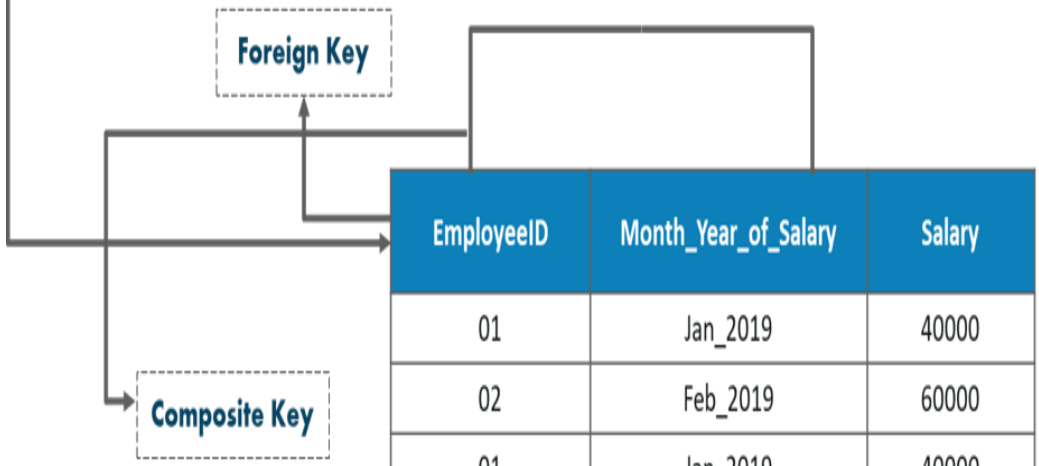
SQL Commands: Different Types Of Keys In Database

There are mainly 7 types of Keys, that can be considered in a database. I am going to consider the below tables to explain to you the various keys.



EMPLOYEE_INFO TABLE

EmployeeID	EmployeeName	Emergency ContactName	PhoneNumber	Address	Insurance Number	Pan Number
01	Shanaya	Abhinay	9898765612	Oberoi Street 23	DWJB3182	ABCKQ11213
02	Anay	Soumya	9432156783	Marathalli House No 23	WJBD8271	FEWEF34531
03	Preeti	Rohan	9764234519	Queens Road 45	SJDN8219	HWKLF21211
04	Vihaan	Akriti	9966442211	Brigade Road Block 4	XCWK9128	JAEWA92167
05	Manasa	Shourya	9543176246	Mayo Road 23	NKAS9829	WEBKQ73992



EmployeeID	Month_Year_of_Salary	Salary
01	Jan_2019	40000
02	Feb_2019	60000
01	Jan_2019	40000
02	Feb_2019	60000

EMPLOYEE_SALARY TABLE

- **Candidate Key –** A set of attributes which can uniquely identify a table can be termed as a Candidate Key. A table can have more than one candidate key, and out of the chosen candidate keys, one key can be chosen as a Primary Key. In the above example, since EmployeeID, InsuranceNumber and PanNumber can uniquely identify every tuple, they would be considered as a Candidate Key.
- **Super Key –** The set of attributes which can uniquely identify a tuple is known as Super Key. So, a candidate key, primary key, and a unique key is a superkey, but vice-versa isn't true.
- **Primary Key –** A set of attributes which are used to uniquely identify every tuple is also a primary key. In the above example, since EmployeeID, InsuranceNumber and PanNumber are candidate keys, any one of them can be chosen as a Primary Key. Here EmployeeID is chosen as the primary key.
- **Alternate Key –** Alternate Keys are the candidate keys, which are not chosen as a Primary key. From the above example, the alternate keys are PanNumber and Insurance Number.
- **Unique Key –** The unique key is similar to the primary key, but allows one NULL value in the column. Here the Insurance Number and the Pan Number can be considered as unique keys.
- **Foreign Key –** An attribute that can only take the values present as the values of some other attribute, is the foreign key to the attribute to which it refers. in the above example, the Employee_ID from the Employee_Information Table is referred to the Employee_ID from the Employee_Salary Table.
- **Composite Key –** A composite key is a combination of two or more columns that identify each tuple uniquely. Here, the Employee_ID and Month-Year_Of_Salary can be grouped together to uniquely identify every tuple in the table.

SQL Commands: Constraints Used In Database

Constraints are used in a database to specify the rules for data in a table. The following are the different types of constraints:

- NOT NULL
- UNIQUE
- CHECK
- DEFAULT
- INDEX

NOT NULL

This constraint ensures that a column cannot have a NULL value.

Example

```
1  --NOT NULL on Create Table
2
3  CREATE TABLE Employee_Info
4  (
   EmployeeID int NOT NULL,
```

```
EmployeeName varchar(255) NOT NULL,
Emergency ContactName varchar(255),
PhoneNumber int NOT NULL,
Address varchar(255),
City varchar(255),
Country varchar(255)
);


--NOT NULL on ALTER TABLE

ALTER TABLE Employee_Info
MODIFY PhoneNumber int NOT NULL;
```

UNIQUE

This constraint ensures that all the values in a column are unique.

Example

```
--UNIQUE on Create Table


CREATE TABLE Employee_Info
(
EmployeeID int NOT NULL UNIQUE,
EmployeeName varchar(255) NOT NULL,
Emergency ContactName varchar(255),
PhoneNumber int NOT NULL,
Address varchar(255),
City varchar(255),
Country varchar(255)
```

```
);

--UNIQUE on Multiple Columns

CREATE TABLE Employee_Info
(
EmployeeID int NOT NULL,
EmployeeName varchar(255) NOT NULL,
Emergency ContactName varchar(255),
PhoneNumber int NOT NULL,
Address varchar(255),
City varchar(255),
Country varchar(255),
CONSTRAINT UC_Employee_Info UNIQUE(Employee_ID, PhoneNumber)
);

--UNIQUE on ALTER TABLE

ALTER TABLE Employee_Info
ADD UNIQUE (Employee_ID);

--To drop a UNIQUE constraint

ALTER TABLE  Employee_Info
DROP CONSTRAINT UC_Employee_Info;
```

```
36
```

CHECK

This constraint ensures that all the values in a column satisfy a specific condition.

Example

```
--CHECK Constraint on CREATE TABLE


CREATE TABLE Employee_Info
(
EmployeeID int NOT NULL,
EmployeeName varchar(255),
Emergency ContactName varchar(255),
PhoneNumber int,
Address varchar(255),
City varchar(255),
Country varchar(255) CHECK (Country=='India')
);

--CHECK Constraint on multiple columns

CREATE TABLE Employee_Info
(
EmployeeID int NOT NULL,
EmployeeName varchar(255),
Emergency ContactName varchar(255),
PhoneNumber int,
Address varchar(255),
City varchar(255),
Country varchar(255) CHECK (Country = 'India' AND Cite = 'Hyderabad')
```

```
);


--CHECK Constraint on ALTER TABLE


ALTER TABLE Employee_Info
ADD CHECK (Country=='India');

--To give a name to the CHECK Constraint

ALTER TABLE Employee_Info
ADD CONSTRAINT CheckConstraintName CHECK (Country=='India');

--To drop a CHECK Constraint

ALTER TABLE Employee_Info
DROP CONSTRAINT CheckConstraintName;
```

DEFAULT

This constraint consists of a set of default values for a column when no value is specified.

Example

```
--DEFAULT Constraint on CREATE TABLE

CREATE TABLE Employee_Info
(
EmployeeID int NOT NULL,
```

```
    EmployeeName varchar(255),
    Emergency ContactName varchar(255),
    PhoneNumber int,
    Address varchar(255),
    City varchar(255),
    Country varchar(255) DEFAULT 'India'
    );

--DEFAULT Constraint on ALTER TABLE

    ALTER TABLE Employee_Info
    ADD CONSTRAINT defau_Country
    DEFAULT 'India' FOR Country;

    --To drop the Default Constraint

    ALTER TABLE Employee_Info
    ALTER COLUMN Country DROP DEFAULT;
```

INDEX

This constraint is used to create indexes in the table, through which you can create and retrieve data from the database very quickly.

Syntax

```
--Create an Index where duplicate values are allowed
CREATE INDEX IndexName
ON TableName (Column1, Column2, ...ColumnN);

--Create an Index where duplicate values are not allowed
CREATE UNIQUE INDEX IndexName
ON TableName (Column1, Column2, ...ColumnN);
```

Example

```
CREATE INDEX idex_EmployeeName
ON Persons (EmployeeName);


--To delete an index in a table


DROP INDEX Employee_Info.idex_EmployeeName;
```

Now, let us look into the next part of this article i.e. DML Commands.

SQL Commands: Data Manipulation Language Commands (DML)

This section of the article will give you an insight into the commands through which you can manipulate the database. The commands are as follows:

- USE
- INSERT INTO
- UPDATE
- DELETE
- SELECT

Apart from these commands, there are also other manipulative operators/functions such as:

- Operators
- Aggregate Functions
- NULL Functions
- Aliases & Case Statement

USE

The USE statement is used to select the database on which you want to perform operations.

Syntax

```
USE DatabaseName;
```

Example

```
USE Employee;
```

INSERT INTO

This statement is used to insert new records into the table.

Syntax

```
INSERT INTO TableName (Column1, Column2, Column3, ...,ColumnN)
VALUES (value1, value2, value3, ...);

--If you don't want to mention the column names then use the below
syntax

INSERT INTO TableName
VALUES (Value1, Value2, Value3, ...);
```

Example

```
1          INSERT INTO Employee_Info(EmployeeID, EmployeeName, Emergency ContactName, Pho
2                VALUES ('06', 'Sanjana','Jagannath', '9921321141', 'Camel Street House N
3
4                                                    INSERT INTO Employee_Info
5                        VALUES ('07', 'Sayantini','Praveen', '9934567654', 'Nice Road 21'
```

UPDATE

This statement is used to modify the records already present in the table.

Syntax

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2, ...
WHERE Condition;
```

Example

```
1                                    UPDATE Employee_Info
2                        SET EmployeeName = 'Aahana', City= 'Ahmedabad'
3                                    WHERE EmployeeID = 1;
```

DELETE

This statement is used to delete the existing records in a table.

Syntax

```
DELETE FROM TableName WHERE Condition;
```

Example

```
1                                    DELETE FROM Employee_Info
2                                  WHERE EmployeeName='Preeti';
```

SELECT

This statement is used to select data from a database and the data returned is stored in a result table, called the **result-set**.

Syntax

```
SELECT Column1, Column2, ...ColumN
FROM TableName;

--(*) is used to select all from the table
SELECT * FROM table_name;

-- To select the number of records to return use:
SELECT TOP 3 * FROM TableName;
```

Example

```
1                                SELECT EmployeeID, EmployeeName
2                                     FROM Employee_Info;
3
4                           --(*) is used to select all from the table
5                                   SELECT * FROM Employee_Info;
6
7                        -- To select the number of records to return use:
8                                 SELECT TOP 3 * FROM Employee_Info;
```

Apart from just using the SELECT keyword individually, you can use the following keywords with the SELECT statement:

-
- DISTINCT
- ORDER BY
- GROUP BY
- HAVING Clause
- INTO

The 'SELECT DISTINCT' Statement

This statement is used to return only different values.

Syntax

```
SELECT DISTINCT Column1, Column2, ...ColumnN
FROM TableName;
```

Example

```
SELECT DISTINCT PhoneNumber FROM Employee_Info;
```

The 'ORDER BY' Statement

The 'ORDER BY' statement is used to sort the required results in ascending or descending order. The results are sorted in ascending order by default. Yet, if you wish to get the required results in descending order, you have to use the **DESC** keyword.

Syntax

```
SELECT Column1, Column2, ...ColumnN
FROM TableName
ORDER BY Column1, Column2, ... ASC|DESC;
```

Example

```
-- Select all employees from the 'Employee_Info' table s
SELECT * FROM Employee_Info
ORDER BY EmergencyContactNam


-- Select all employees from the 'Employee_Info' table sorted by E
SELECT * FROM Employee_Info
ORDER BY EmergencyContactName D


-- Select all employees from the 'Employee_Info' table sorted by
SELECT * FROM Employee_Info
ORDER BY EmergencyContactName, Empl


/* Select all employees from the 'Employee_Info' table sorted by EmergencyContactNam
                                                                */
SELECT * FROM Employee_Info
ORDER BY EmergencyContactName ASC, Emplo
```

The 'GROUP BY' Statement

This 'GROUP BY' statement is used with the aggregate functions to group the result-set by one or more columns.

Syntax

```
SELECT Column1, Column2,..., ColumnN
```

```
FROM TableName
WHERE Condition
GROUP BY ColumnName(s)
ORDER BY ColumnName(s);
```

Example

```
1                    -- To list the number of employees from each city.
2
3                          SELECT COUNT(EmployeeID), City
4                               FROM Employee_Info
5                                 GROUP BY City;
```

The 'HAVING' Clause

The 'HAVING' clause is used in SQL because the **WHERE keyword** cannot be used everywhere.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE Condition
GROUP BY ColumnName(s)
HAVING Condition
ORDER BY ColumnName(s);
```

Example

```
1  /*&amp;nbsp;&amp;nbsp;To list the number of employees in each city. The employees should be s
                                                          who have more than 5 employees:
2
3                                                       SELECT COUNT(EmployeeID), Cit
4                                                            FROM Employee_Info
5                                                              GROUP BY City
6                                                       HAVING COUNT(EmployeeID) > 2
7                                                     ORDER BY COUNT(EmployeeID) DES
```

The 'SELECT INTO' Statement

The 'SELECT INTO' statement is used to copy data from one table to another.

Syntax

```
SELECT *
INTO NewTable [IN ExternalDB]
FROM OldTable
```

```
WHERE Condition;
```

Example

```
-- To create a backup of database 'Employee'
SELECT * INTO EmployeeBackup
FROM Employee;

--To select only few columns from Employee
SELECT EmployeeName, PhoneNumber INTO EmployeeContactDetails
FROM Employee;

SELECT * INTO BlrEmployee
FROM Employee
WHERE City = 'Bangalore';
```

Now, as I mentioned before, let us move onto our next section in this article on SQL Commands, i.e. the Operators.

Operators in SQL

The different set of operators available in SQL are as follows:

Let us look into each one of them, one by one.

Arithmetic Operators

Operator	Description
%	Modulous [A % B]
/	Division [A / B]
*	Multiplication [A * B]
–	Subtraction [A – B]
+	Addition [A + B]

Bitwise Operators

Operator	Description
^	Bitwise Exclusive OR (XOR) [A ^ B]
|	Bitwise OR [A | B]
&	Bitwise AND [A & B]

Comparison Operators

Operator	Description
< >	Not Equal to [A < > B]
<=	Less than or equal to [A <= B]
>=	Greater than or equal to [A >= B]
<	Less than [A < B]
>	Greater than [A > B]
=	Equal to [A = B]

Compound Operators

Operator	Description
|*=	Bitwise OR equals [A |*= B]
^-=	Bitwise Exclusive equals [A ^-= B]
&=	Bitwise AND equals [A &= B]

%=	Modulo equals [A %= B]
/=	Divide equals [A /= B]
*=	Multiply equals [A*= B]
-=	Subtract equals [A-= B]
+=	Add equals [A+= B]

Logical Operators

The Logical operators present in SQL are as follows:

-
  - AND
  - OR
  - NOT
  - BETWEEN
  - LIKE
  - IN
  - EXISTS
  - ALL
  - ANY

AND Operator

This operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE Condition1 AND Condition2 AND Condition3 ...;
```

Example

```
1                                SELECT * FROM Employee_Info

2                          WHERE City='Mumbai' AND City='Hyderabad';</pre>
```

OR Operator

This operator displays all those records which satisfy any of the conditions separated by OR and give the output TRUE.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE Condition1 OR Condition2 OR Condition3 ...;
```

Example

```
SELECT * FROM Employee_Info
WHERE City='Mumbai' OR City='Hyderabad';
```

NOT Operator

The NOT operator is used, when you want to display the records which do not satisfy a condition.

Syntax

```
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE NOT Condition;
```

Example

```
SELECT * FROM Employee_Info
 WHERE NOT City='Mumbai';
```

NOTE: You can also combine the above three operators and write a query as follows:

```
SELECT * FROM Employee_Info
WHERE NOT Country='India' AND (City='Bangalore' OR City='Hyderabad');
```

NOTE: You can also combine the above three operators and write a query as follows:

```
SELECT * FROM Employee_Info
WHERE NOT Country='India' AND (City='Bangalore' OR City='Hyderabad');
```

BETWEEN Operator

The BETWEEN operator is used, when you want to select values within a given range. Since this is an inclusive operator, both the starting and ending values are considered.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName BETWEEN Value1 AND Value2;
```

Example

```
SELECT * FROM Employee_Salary
```

```
2                    WHERE Salary BETWEEN 40000 AND 50000;
```

LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column of a table. There are mainly two wildcards that are used in conjunction with the LIKE operator:

- **%** – It matches 0 or more character.
- **_** – It matches exactly one character.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName LIKE pattern;
```

Refer to the following table for the various patterns that you can mention with the LIKE operator.

Like Operator Condition	Description
WHERE CustomerName LIKE 'v%	Finds any values that start with "v"
WHERE CustomerName LIKE '%v'	Finds any values that end with "v"
WHERE CustomerName LIKE '%and%'	Finds any values that have "and" in any position
WHERE CustomerName LIKE '_q%'	Finds any values that have "q" in the second position.
WHERE CustomerName LIKE 'u_%_%'	Finds any values that start with "u" and are at least 3 characters in length
WHERE ContactName LIKE 'm%a'	Finds any values that start with "m" and end with "a"

Example

```
1    SELECT * FROM Employee_Info
2    WHERE EmployeeName LIKE 'S%';
```

IN Operator

This operator is used for multiple OR conditions. This allows you to specify multiple values in a WHERE clause.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName IN (Value1,Value2...);
```

Example

```
1                                   SELECT * FROM Employee_Info
2                           WHERE City IN ('Mumbai', 'Bangalore', 'Hyderabad');
```

NOTE: You can also use IN while writing Nested Queries.

EXISTS Operator

The EXISTS operator is used to test if a record exists or not.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE EXISTS
(SELECT ColumnName FROM TableName WHERE condition);
```

Example

```
1                                                SELECT EmergencyContactName
2                                                  FROM Employee_Info
3          WHERE EXISTS (SELECT EmergencyContactName FROM Employee_Info WHERE EmployeeId =
```

ALL Operator

The ALL operator is used with a WHERE or HAVING clause and returns TRUE if all of the subquery values meet the condition.

Syntax

```
SELECT ColumnName(s)
FROM TableName
WHERE ColumnName operator ALL
(SELECT ColumnName FROM TableName WHERE condition);
```

Example

```
1                                          SELECT EmployeeName
2                                          FROM Employee_Info
3         WHERE EmployeeID = ALL (SELECT EmployeeID FROM Employee_Info WHERE City = 'Hyde
```

ANY Operator

Similar to the ALL operator, the ANY operator is also used with a WHERE or HAVING clause and returns true if any of the subquery values meet the condition.

Syntax

```
SELECT ColumnName(s)
FROM TableName
```

```
WHERE ColumnName operator ANY
(SELECT ColumnName FROM TableName WHERE condition);
```

Example

```
1                                                    SELECT EmployeeName

2                                                    FROM Employee_Info

3          WHERE EmployeeID = ANY (SELECT EmployeeID FROM Employee_Info WHERE City = 'Hyd
```

Next, in this article on SQL Commands, let us look into the various Aggregate Functions provided in SQL.

Aggregate Functions

This section of the article will include the following functions:

-
- MIN()
- MAX()
- COUNT()
- SUM()
- AVG()

MIN() Function

The MIN function returns the smallest value of the selected column in a table.

Syntax

```
SELECT MIN(ColumnName)
FROM TableName
WHERE Condition;
```

Example

```
1                          SELECT MIN(EmployeeID) AS SmallestID

2                                FROM Employee_Info;
```

MAX() Function

The MAX function returns the largest value of the selected column in a table.

Syntax

```
SELECT MAX(ColumnName)
FROM TableName
WHERE Condition;
```

Example

```
1                          SELECT MAX(Salary) AS LargestFees

2                                FROM Employee_Salary;
```

COUNT() Function

The COUNT function returns the number of rows which match the specified criteria.

Syntax

```
SELECT COUNT(ColumnName)
FROM TableName
WHERE Condition;
```

Example

```
1  SELECT COUNT(EmployeeID)
2  FROM Employee_Info;
```

SUM() Function

The SUM function returns the total sum of a numeric column that you choose.

Syntax

```
SELECT SUM(ColumnName)
FROM TableName
WHERE Condition;
```

Example

```
1  SELECT SUM(Salary)
2  FROM Employee_Salary;
```

AVG() Function

The AVG function returns the average value of a numeric column that you choose.

Syntax

```
SELECT AVG(ColumnName)
FROM TableName
WHERE Condition;
```

Example

```
1  SELECT AVG(Salary)
2  FROM Employee_Salary;
```

NULL Functions

The NULL functions are those functions which let you return an alternative value if an expression is NULL. In the SQL Server, the function is **ISNULL()**.

Example

```
1  SELECT EmployeeID * (Month_Year_of_Salary + ISNULL(Salary, 0))
2  FROM Employee_Salary;
```

Aliases & Case Statement

In this section of this article on SQL Commands, you will go through the Aliases and Case statement one after the other.

Aliases

Aliases are used to give a column/table a temporary name and only exists for a duration of the query.

Syntax

```
--Alias Column Syntax

SELECT ColumnName AS AliasName
FROM TableName;

--Alias Table Syntax

SELECT ColumnName(s)
FROM TableName AS AliasName;
```

Example

```
1                    SELECT EmployeeID AS ID, EmployeeName AS EmpName
2                                FROM Employee_Info;
3
4          SELECT EmployeeName AS EmpName, EmergencyContactName AS [Contact Name]
5                                FROM Employee_Info;
```

Case Statement

This statement goes through all the conditions and returns a value when the first condition is met. So, if no conditions are TRUE, it returns the value in the ELSE clause. Also, if no conditions are true and there is no ELSE part, then it returns NULL.

Syntax

```
CASE
WHEN Condition1 THEN Result1
WHEN Condition2 THEN Result2
WHEN ConditionN THEN ResultN
ELSE Result
END;
```

Example

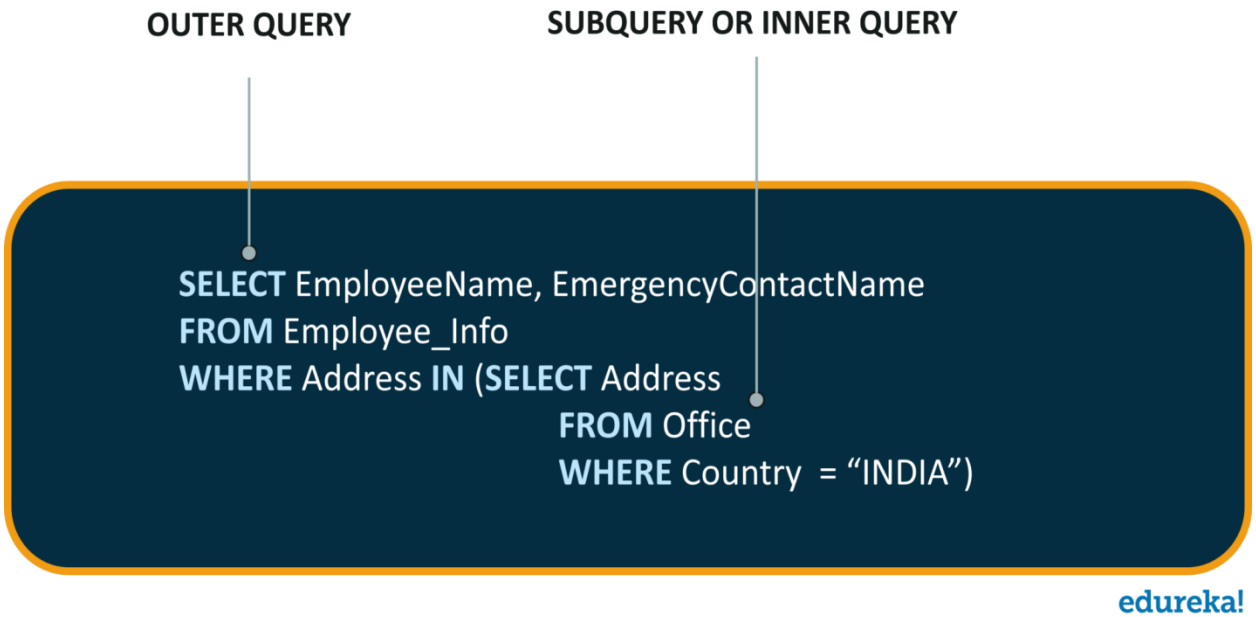
```
1                          SELECT EmployeeName, City
                              FROM Employee_Info
2                                 ORDER BY
```

```
3                                   (CASE
4                 WHEN City IS NULL THEN 'Country is India by default'
5                                   ELSE City
6                                   END);
7
```

Now, that I have told you a lot about DML commands in this article on SQL Commands, let me just tell you in short about Nested Queries, Joins, Set Operations, and Dates & Auto Increment.

SQL Commands: Nested Queries

Nested queries are those queries which have an outer query and inner subquery. So, basically, the subquery is a query which is nested within another query such as SELECT, INSERT, UPDATE or DELETE. Refer to the image below:



SQL Commands: Joins

JOINS are used to combine rows from two or more tables, based on a related column between those tables. The following are the types of joins:

- **INNER JOIN:** This join returns those records which have matching values in both the tables.
- **FULL JOIN:** This join returns all those records which either have a match in the left or the right table.
- **LEFT JOIN:** This join returns records from the left table, and also those records which satisfy the condition from the right table.

- **RIGHT JOIN:** This join returns records from the right table, and also those records which satisfy the condition from the left table.

Refer to the image below.

Let's consider the below table apart from the Employee_Info table, to understand the syntax of joins.

TechID	EmpID	TechName	ProjectStartDate
1	10	DevOps	04-01-2019
2	11	Blockchain	06-07-2019
3	12	Python	01-03-2019

INNER JOIN

Syntax

```
SELECT ColumnName(s)
FROM Table1
INNER JOIN Table2 ON Table1.ColumnName = Table2.ColumnName;
```

Example

```
1                    SELECT Technologies.TechID, Employee_Info.EmployeeName
2                                        FROM Technologies
3            INNER JOIN Employee_Info ON Technologies.EmpID = Employee_Info.EmpID;
```

FULL JOIN

Syntax

```
SELECT ColumnName(s)
FROM Table1
FULL OUTER JOIN Table2 ON Table1.ColumnName = Table2.ColumnName;
```

Example

```
1                    SELECT Employee_Info.EmployeeName, Technologies.TechID
```

```
2                                   FROM Employee_Info
3           FULL OUTER JOIN Orders ON Employee_Info.EmpID=Employee_Salary.EmpID
4                                   ORDER BY Employee_Info.EmployeeName;
```

LEFT JOIN

Syntax

```
SELECT ColumnName(s)
FROM Table1
LEFT JOIN Table2 ON Table1.ColumnName = Table2.ColumnName;
```

Example

```
1                   SELECT Employee_Info.EmployeeName, Technologies.TechID
2                                   FROM Employee_Info
3         LEFT JOIN Technologies ON Employee_Info.EmployeeID = Technologies.EmpIDID
4                                   ORDER BY Employee_Info.EmployeeName;
```

RIGHT JOIN

Syntax

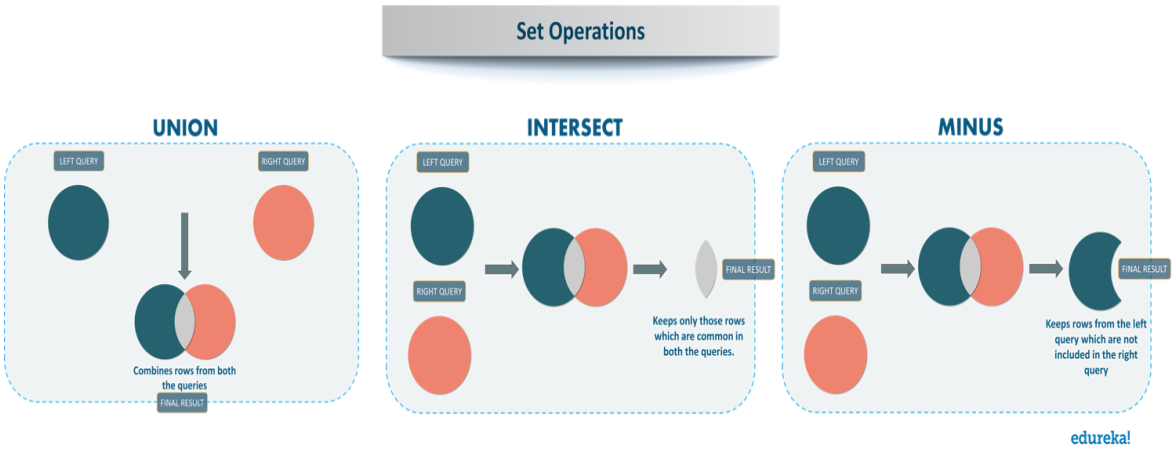
```
SELECT ColumnName(s)
FROM Table1
RIGHT JOIN Table2 ON Table1.ColumnName = Table2.ColumnName;
```

Example

```
1                                   SELECT Technologies.TechID
2                                   FROM Technologies
3         RIGHT JOIN Employee_Info ON Technologies.EmpID = Employee_Info.EmployeeID
4                                   ORDER BY Technologies.TechID;
```

SQL Commands: Set Operations

There are mainly three set operations:UNION, INTERSECT, EXCEPT. You can refer to the image below to understand the set operations in SQL.



UNION

This operator is used to combine the result-set of two or more SELECT statements.

Syntax

```
SELECT ColumnName(s) FROM Table1
UNION
SELECT ColumnName(s) FROM Table2;
```

INTERSECT

This clause used to combine two SELECT statements and return the intersection of the data-sets of both the SELECT statements.

Syntax

```
SELECT Column1 , Column2 ....
FROM TableName
WHERE Condition

INTERSECT

SELECT Column1 , Column2 ....
FROM TableName
WHERE Condition
```

EXCEPT

This operator returns those tuples that are returned by the first SELECT operation, and are not returned by the second SELECT operation.

Syntax

```
SELECT ColumnName
FROM TableName

EXCEPT

SELECT ColumnName
FROM TableName;
```

Next, in this article, let us look into the date functions and auto-increment fields.

SQL Commands: Dates & Auto Increment

In this section of this article, I will explain to you how to use the Date functions and also the Auto-Increment fields.

Dates

The following data types are present in a SQL Server to store a date or a date/time value in a database.

Data Type	Format
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MI:SS
SMALLDATETIME	YYYY-MM-DD HH:MI:SS
TIMESTAMP	A Unique Number

Example

```
1    SELECT * FROM Technologies WHERE ProjectStartDate='2019-04-01'
```

Auto Increment

This field generates a unique number automatically when a new record is inserted into a table. The MS SQL Server uses the **IDENTITY** keyword for this feature.

Example

```
1    <span>/* To define the "EmployeeID" column to be an auto-increment primary ke
2
3                                                    <span>CREATE TABLE Employee_Info (</
4                                                 <span>EmployeeID INT IDENTITY(1,1) PRIMARY
5                                                    <span>EmployeeName VARCHAR(255) NOT NUL
6                                                 <span>EmergencyContactName VARCHAR(255) NOT
7                                                                                 <span>);</span>
```

Now, that you guys know the DML commands, let's move onto our next section in this article on SQL Commands i.e. the DCL commands.

SQL Commands: Data Control Language Commands (DCL)

This section of the article will give you an insight into the commands which are used to enforce database security in multiple user database environments. The commands are as follows:

-
  - GRANT
  - REVOKE

GRANT

This command is used to provide access or privileges on the database and its objects to the users.

Syntax

```
GRANT PrivilegeName
ON ObjectName
TO {UserName |PUBLIC |RoleName}
[WITH GRANT OPTION];
```

where,

- **PrivilegeName** – Is the privilege/right/access granted to the user.
- **ObjectName** – Name of a database object like TABLE/VIEW/STORED PROC.
- **UserName** – Name of the user who is given the access/rights/privileges.
- **PUBLIC** – To grant access rights to all users.
- **RoleName** – The name of a set of privileges grouped together.
- **WITH GRANT OPTION** – To give the user access to grant other users with rights.

Example

```
1    -- To grant SELECT permission to Employee_Info table to user1

2    GRANT SELECT ON Employee_Info TO user1;
```

REVOKE

This command is used to withdraw the user's access privileges given by using the GRANT command.

Syntax

```
REVOKE PrivilegeName
ON ObjectName
FROM {UserName |PUBLIC |RoleName}
```

Example

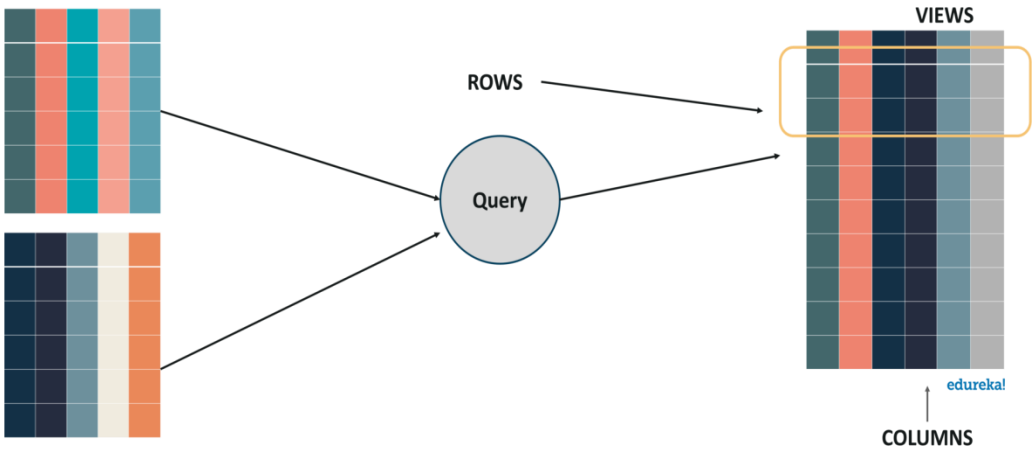
```
1    -- To revoke the granted permission from user1

2    REVOKE SELECT ON Employee_Info TO user1;
```

Now, next in this article on SQL Commands, I will discuss Views, Stored Procedures, and Triggers.

SQL Commands: Views

A view in SQL is a single table, which is derived from other tables. So, a view contains rows and columns similar to a real table and has fields from one or more table.



The 'CREATE VIEW' statement

This statement is used to create a view, from a table.

Syntax

```
CREATE VIEW ViewName AS
SELECT Column1, Column2, ..., ColumnN
FROM TableName
WHERE Condition;
```

Example

```
1    CREATE VIEW [Kolkata Employees] AS
2     SELECT EmployeeName, PhoneNumber
3          FROM Employee_Info
4        WHERE City = "Kolkata";
```

The 'CREATE OR REPLACE VIEW' statement

This statement is used to update a view.

Syntax

```
CREATE VIEW OR REPLACE ViewName AS
SELECT Column1, Column2, ..., ColumnN
FROM TableName
```

```
WHERE Condition;
```

Example

```
CREATE VIEW OR REPLACE [Kolkata Employees] AS
SELECT EmployeeName, PhoneNumber
FROM Employee_Info
WHERE City = "Kolkata";
```

The 'DROP VIEW' statement

This statement is used to delete a view.

Syntax

```
DROP VIEW ViewName;
```

Example

```
DROP VIEW [Kolkata Employees];
```

SQL Commands: Stored Procedures

A code which you can save and reuse it again is known as StoredProcedures.

Syntax

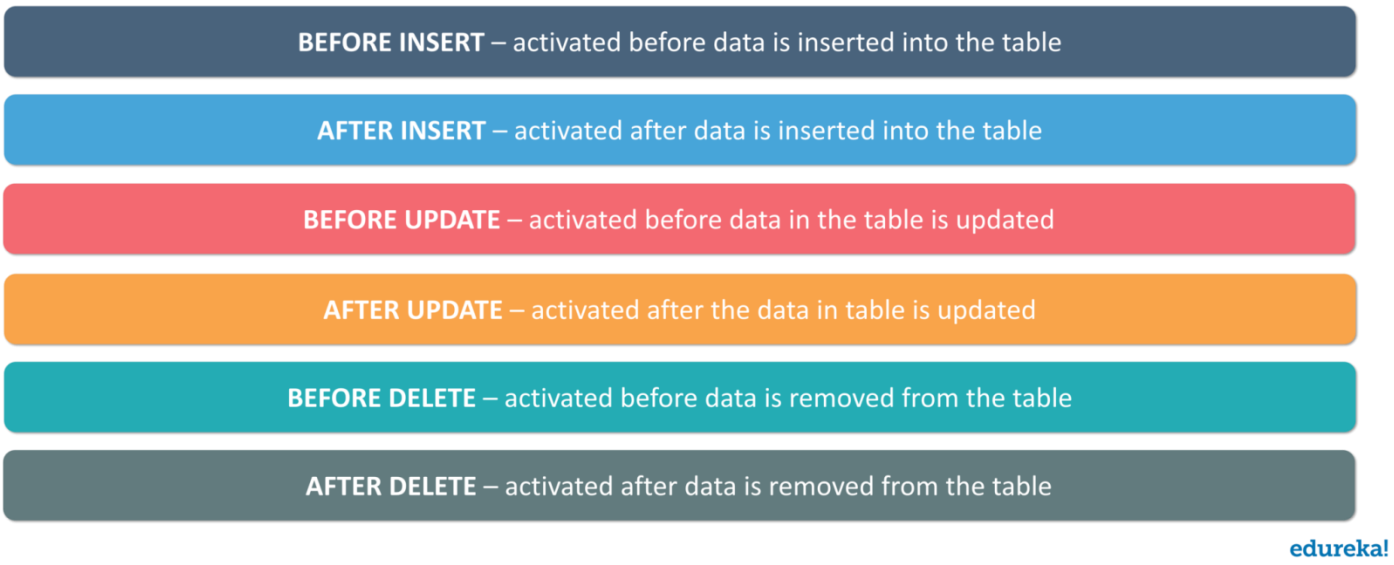
```
CREATE PROCEDURE ProcedureName
AS
SQLStatement
GO;
```

Example

```
EXEC ProcedureName;
```

SQL Commands: Triggers

Triggers are a set of SQL statements which are stored in the database catalog. These statements are executed whenever an event associated with a table occurs. So, a **trigger** can be invoked either **BEFORE** or **AFTER** the data is changed by **INSERT**, **UPDATE** or **DELETE** statement. Refer to the image below.



Syntax

```
CREATE TRIGGER [TriggerName]
[BEFORE | AFTER]
{INSERT | UPDATE | DELETE}
on [TableName]
[FOR EACH ROW]
[TriggerBody]
```

Now, let's move on to the last section of this article on SQL Commands i.e. the Transaction Control Language Commands.

SQL Commands: Transaction Control Language Commands (TCL)

This section of the article will give you an insight into the commands which are used to manage transactions in the database. The commands are as follows:

-
  - COMMIT
  - ROLLBACK
  - SAVEPOINT

COMMIT

This command is used to save the transaction into the database.

Syntax

```
COMMIT;
```

ROLLBACK

This command is used to restore the database to the last committed state.

Syntax

```
ROLLBACK;
```

NOTE: When you use ROLLBACK with SAVEPOINT, then you can directly jump to a savepoint in an ongoing transaction. Syntax: ROLLBACK TO SavepointName;

SAVEPOINT

This command is used to temporarily save a transaction. So if you wish to rollback to any point, then you can save that point as a 'SAVEPOINT'.

Syntax

```
SAVEPOINT SAVEPOINTNAME;
```

Consider the below example to understand the working of transactions in the database.

EmployeeID	EmployeeName
01	Ruhaan
02	Suhana
03	Aayush
04	Rashi

Now, use the below SQL queries to understand the transactions in the database.

```
1   INSERT INTO Employee_Table VALUES(05, 'Avinash');
2   COMMIT;
3   UPDATE Employee_Table SET name = 'Akash' WHERE id = '05';
4   SAVEPOINT S1;
5   INSERT INTO Employee_Table VALUES(06, 'Sanjana');
6   SAVEPOINT S2;
7   INSERT INTO Employee_Table VALUES(07, 'Sanjay');
8   SAVEPOINT S3;
9   INSERT INTO Employee_Table VALUES(08, 'Veena');
10  SAVEPOINT S4;
    SELECT * FROM Employee_Table;
```

```
11
```

The output to the above set of queries would be as follows:

EmployeeID	EmployeeName
01	Ruhaan
02	Suhana
03	Aayush
04	Rashi
05	Akash
06	Sanjana
07	Sanjay
08	Veena

Now, if you rollback to S2 using the below queries, the output is mentioned in the below table.

```
1                                   ROLLBACK TO S2;
2                                   SELECT * FROM Employee_Table;
```

EmployeeID	EmployeeName
01	Ruhaan
02	Suhana
03	Aayush
04	Rashi
05	Akash
06	Sanjana