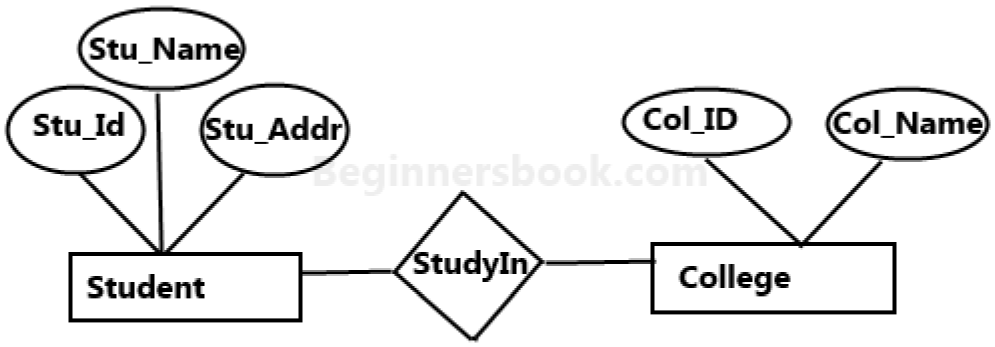
DBMS Notes

ER Model

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.

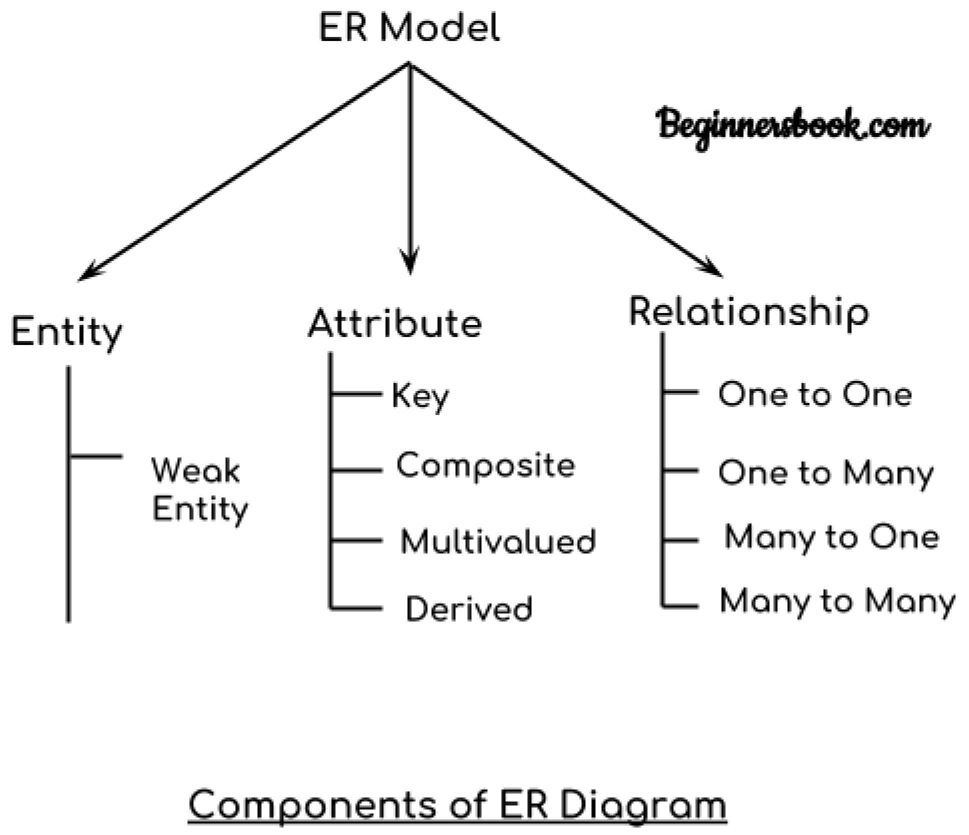


Sample E-R Diagram

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.

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

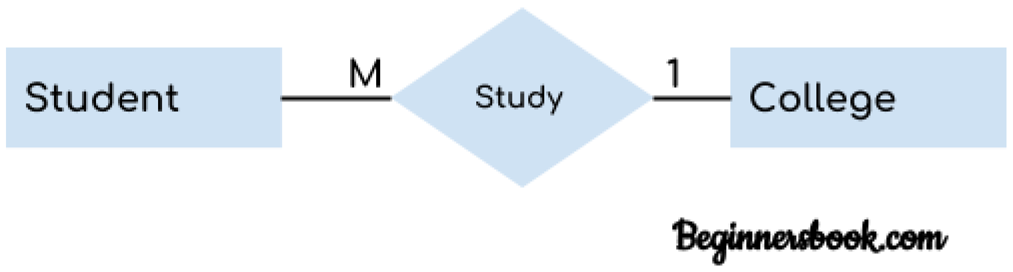
Components of a ER Diagram:



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

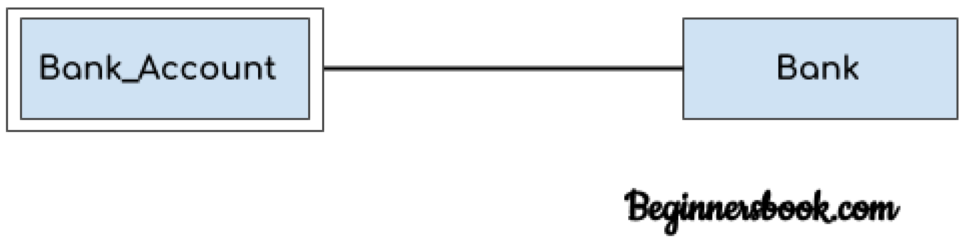
1. Entity

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.



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.



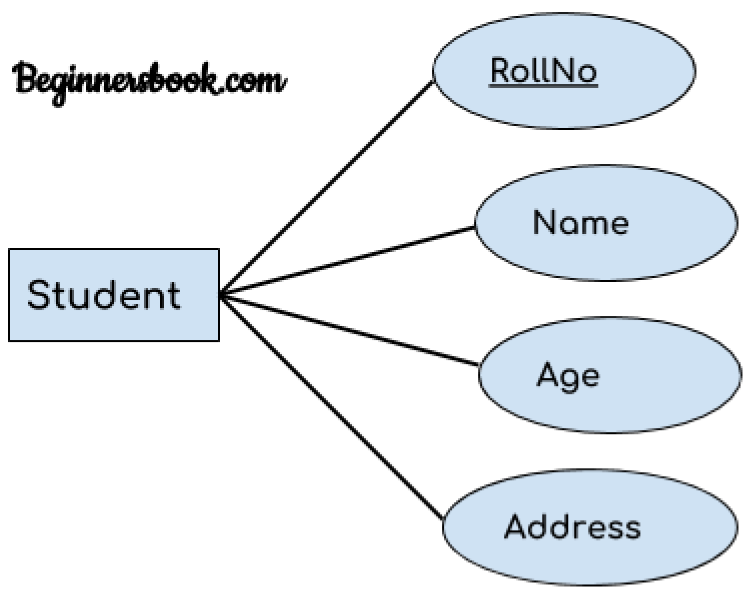
2. Attribute

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

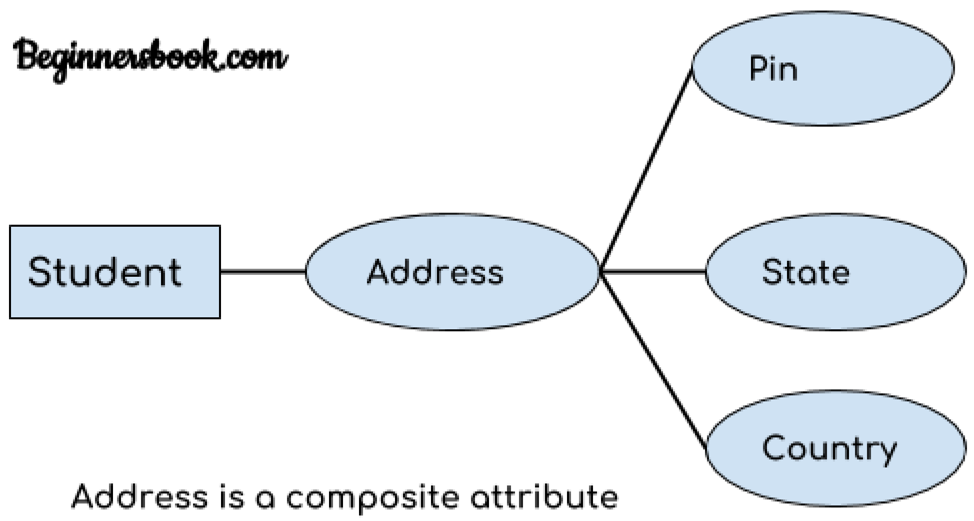
1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

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.



Composite attribute:



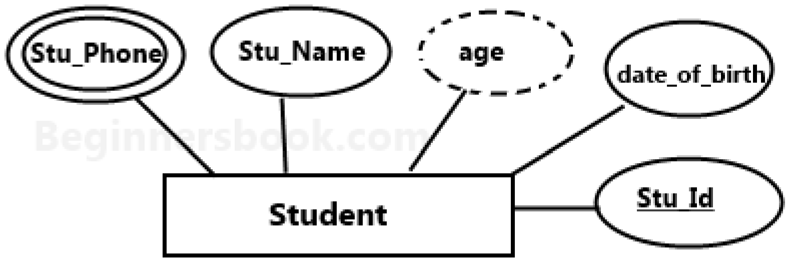
An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

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).

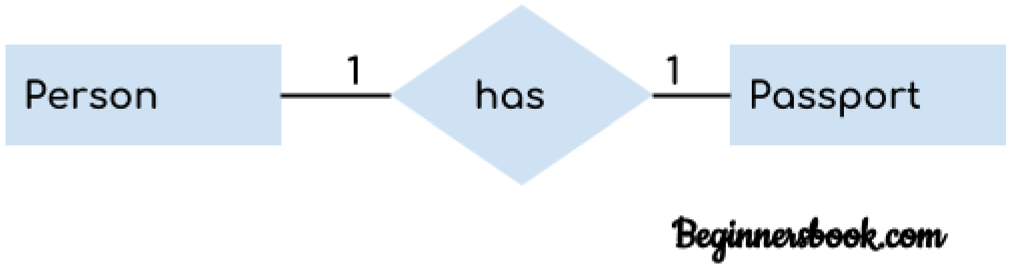


3. Relationship

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

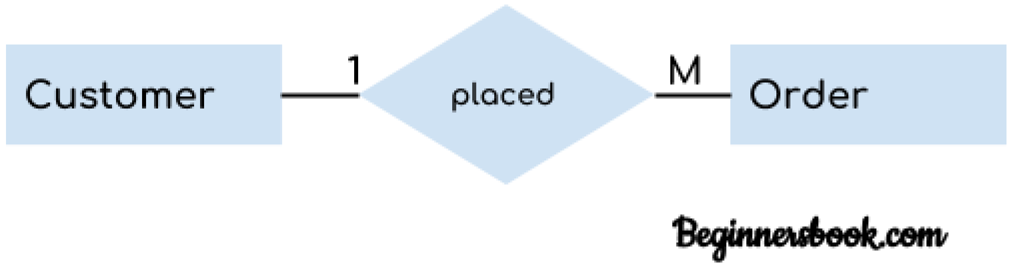
1. One to One Relationship



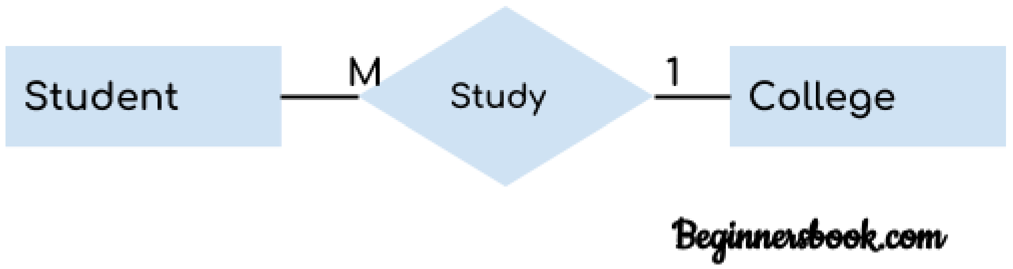
When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

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.

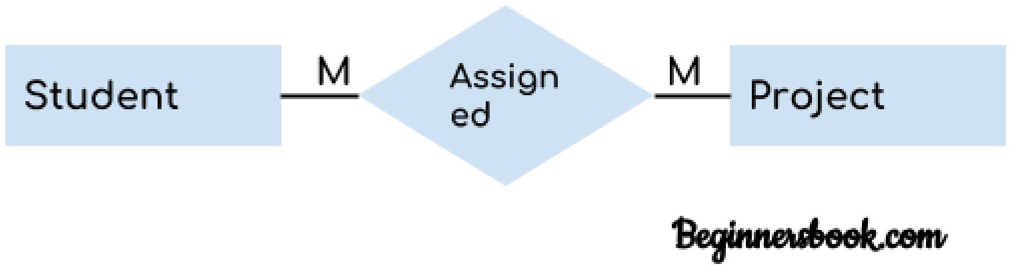


3. Many to One Relationship:
When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

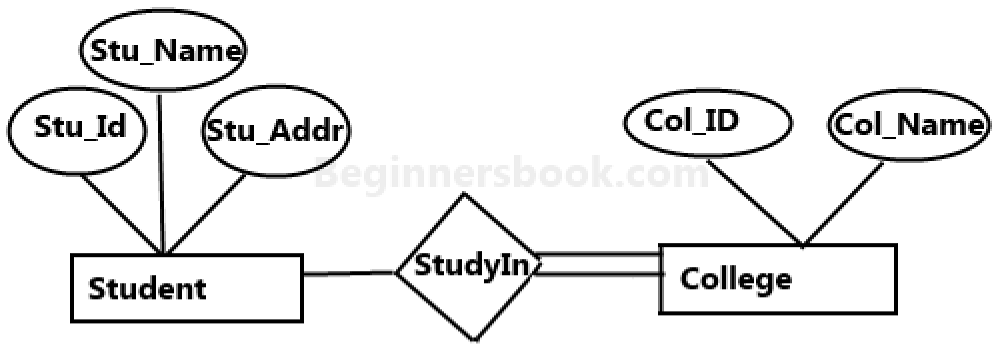


4. Many to Many Relationship

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.



Total Participation of an Entity set
Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. It is also called mandatory participation. For example: In the following diagram each college must have at-least one associated Student. Total participation is represented using a double line between the entity set and relationship set.



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.

Partial participation of an Entity Set

Partial participation of an entity set represents that each entity in the entity set may or may not participate in the relationship instance in that relationship set. It is also called as optional participation

Partial participation is represented using a single line between the entity set and relationship set.

Example: Consider an example of an IT company. There are many employees working for the company. Let's take the example of relationship between employee and role software engineer. Every software engineer is an employee but not every employee is software engineer as there are employees for other roles as well, such as housekeeping, managers, CEO etc. so we can say that participation of employee entity set to the software engineer relationship is partial.

keys in DBMS

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

Types of keys in DBMS

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

DBMS Notes

Candidate Key – A super key with no redundant attribute is known as candidate key

Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Primary key in DBMS

A primary key is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) of that table. For example, you want to store student data in a table "student". The attributes of this table are: student_id, student_name, student_age, student_address. The primary key is a set of one or more of these attributes to uniquely identify a record in the table. In the case, since student_id is different for each student, this can be considered a primary key.

Properties of a primary key

Primary key is denoted by underlining the attribute name (column name).

Minimal:

The primary key should contain minimal number of attributes. The example we seen above, where student_id is able to uniquely identify a record, here combination of two attributes such as {student_id, student_name} can also uniquely identify record. However since we should choose minimal set of attribute thus student is chosen as primary key instead of {student_id, student_name}.

Unique:

The value of primary key should be unique for each row of the table. The column(s) that makes the key cannot contain duplicate values. This is because non-unique value would not help us uniquely identify record. If two students have same student_id then updating a record of one student based on primary key can mistakenly update record of other student.

Non null:

The attribute(s) that is marked as primary key is not allowed to have null values.
Not dependent on Time: The primary key value should not change over time. It should remain as it is until explicitly updated by the user.
Easily accessible:

The primary key of the record should be accessible to all the users who are performing any operations on the database.

Primary keys are not necessarily to be a single attribute (column).
It can be a set of more than one attributes (columns). For example {Stu_Id, Stu_Name} collectively can identify the tuple in the above table, but we do not choose it as primary key because Stu_Id alone is enough to uniquely identifies rows in a table and we always go for minimal set. Having that said, we should choose more than one columns as primary key only when there is no single column that can uniquely identify the tuple in table.

Super key in DBMS

A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.

How candidate key is different from super key?
Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That's the reason they are also termed as minimal super key.

Let's take an example to understand this:
Table: Employee

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

Super keys:
The above table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.

{Emp_SSN}
{Emp_Number}
{Emp_SSN, Emp_Number}
{Emp_SSN, Emp_Name}
{Emp_SSN, Emp_Number, Emp_Name}
{Emp_Number, Emp_Name}

Candidate Keys:

DBMS Notes

As I mentioned in the beginning, a candidate key is a minimal super key with no redundant attributes. The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets.

{Emp_SSN}
{Emp_Number}
Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification.

Super key vs Candidate Key

I have been getting lot of comments regarding the confusion between super key and candidate key. Let me give you a clear explanation.

1. First you have to understand that all the candidate keys are super keys. This is because the candidate keys are chosen out of the super keys.
2. How we choose candidate keys from the set of super keys? We look for those keys from which we cannot remove any fields. In the above example, we have not chosen {Emp_SSN, Emp_Name} as candidate key because {Emp_SSN} alone can identify a unique row in the table and Emp_Name is redundant.

Candidate Key in DBMS

Definition of Candidate Key in DBMS: A super key with no redundant attribute is known as candidate key. Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is that the candidate key should not have any redundant attributes. That’s the reason they are also termed as minimal super key.

Candidate Key Example

Lets take an example of table “Employee”. This table has three attributes: Emp_Id, Emp_Number & Emp_Name. Here Emp_Id & Emp_Number will be having unique values and Emp_Name can have duplicate values as more than one employees can have same name.

Emp_Id	Emp_Number	Emp_Name
------	----------	--------
E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

How many super keys the above table can have?

1. {Emp_Id}

2. {Emp_Number}
3. {Emp_Id, Emp_Number}
4. {Emp_Id, Emp_Name}
5. {Emp_Id, Emp_Number, Emp_Name}
6. {Emp_Number, Emp_Name}

Lets select the candidate keys from the above set of super keys.

1. {Emp_Id} – No redundant attributes
2. {Emp_Number} – No redundant attributes
3. {Emp_Id, Emp_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.
4. {Emp_Id, Emp_Name} – Redundant attribute Emp_Name.
5. {Emp_Id, Emp_Number, Emp_Name} – Redundant attributes. Emp_Id or Emp_Number alone are sufficient enough to uniquely identify a row of Employee table.
6. {Emp_Number, Emp_Name} – Redundant attribute Emp_Name.

The candidate keys we have selected are:
{Emp_Id}
{Emp_Number}

Alternate key in DBMS

As we have seen in the candidate key guide that a table can have multiple candidate keys. Among these candidate keys, only one key gets selected as primary key, the remaining keys are known as alternative or secondary keys.

Alternate Key Example
Lets take an example to understand the alternate key concept. Here we have a table Employee, this table has three attributes: Emp_Id, Emp_Number & Emp_Name.

Table: Employee

Emp_Id	Emp_Number	Emp_Name
E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

There are two candidate keys in the above table:

{Emp_Id}
{Emp_Number}

DBA (Database administrator) can choose any of the above key as primary key. Lets say Emp_Id is chosen as primary key.

Since we have selected Emp_Id as primary key, the remaining key Emp_Number would be called alternative or secondary key.

Composite key in DBMS

Definition of Composite key: A key that has more than one attributes is known as composite key. It is also known as compound key.

Note: Any key such as super key, primary key, candidate key etc. can be called composite key if it has more than one attributes.

Composite key Example

Lets consider a table Sales. This table has four columns (attributes) – cust_Id, order_Id, product_code & product_count.

Table – Sales

cust_Id	order_Id	product_code	product_count
C01	O001	P007	23
C02	O123	P007	19
C02	O123	P230	82
C01	O001	P890	42

None of these columns alone can play a role of key in this table.

Column cust_Id alone cannot become a key as a same customer can place multiple orders, thus the same customer can have multiple entires.

Column order_Id alone cannot be a primary key as a same order can contain the order of multiple products, thus same order_Id can be present multiple times.

Column product_code cannot be a primary key as more than one customers can place order for the same product.

Column product_count alone cannot be a primary key because two orders can be placed for the same product count.

Based on this, it is safe to assume that the key should be having more than one attributes:

Key in above table: {cust_id, product_code}

This is a composite key as it is made up of more than one attributes.

Foreign key in DBMS

Definition: Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Constraints in DBMS

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the data integrity during an update/delete/insert into a table.

Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

Key constraints:

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35)  NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Domain constraints in DBMS

A table is DBMS is a set of rows and columns that contain data. Columns in table have a unique name, often referred as attributes in DBMS. A domain is a unique set of values permitted for an attribute in a table. For example, a domain of month-of-year can accept January, February….December as possible values, a domain of integers can accept whole numbers that are negative, positive and zero.

Definition: Domain constraints are user defined data type and we can define them like this:
Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Example:

For example I want to create a table “student_info” with “stu_id” field having value greater than 100, I can create a domain and table like this:

```
create domain id_value int
constraint id_test
check(value > 100);
```

```
create table student_info (
stu_id id_value PRIMARY KEY,
stu_name varchar(30),
stu_age int
);
```

Another example:
I want to create a table “bank_account” with “account_type” field having value either “checking” or “saving”:

```
create domain account_type char(12)
constraint acc_type_test
check(value in ("Checking", "Saving"));

create table bank_account (
account_nbr int PRIMARY KEY,
account_holder_name varchar(30),
account_type account_type
);
```

Cardinality in DBMS

In Context of Data Models:
In terms of data models, cardinality refers to the relationship between two tables. Relationship can be of four types as we have already seen in Entity relationship guide:

One to One – A single row of first table associates with single row of second table. For example, a relationship between person and passport table is one to one because a person can have only one passport and a passport can be assigned to only one person.

One to Many – A single row of first table associates with more than one rows of second table. For example, relationship between customer and order table is one to many because a customer can place many orders but a order can be placed by a single customer alone.

Many to One – Many rows of first table associate with a single row of second table. For example, relationship between student and university is many to one because a university can have many students but a student can only study only in single university at a time.

Many to Many – Many rows of first table associate with many rows of second table. For example, relationship between student and course table is many to many because a student can take many courses at a time and a course can be assigned to many students.

In Context of Query Optimization:
In terms of query, the cardinality refers to the uniqueness of a column in a table. The column with all unique values would be having the high cardinality and the column with all duplicate values would be having the low cardinality. These cardinality scores helps in query optimization.

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

- Trivial functional dependency
- non-trivial functional dependency
- Multivalued dependency
- Transitive dependency

Trivial functional dependency:

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

Symbolically: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

For example: Consider a table with two columns Student_id and Student_Name.

{Student_Id, Student_Name} -> Student_Id is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

Also, Student_Id -> Student_Id & Student_Name -> Student_Name are trivial dependencies too.

Non trivial functional dependency:

If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:

An employee table with three attributes: emp_id, emp_name, emp_address.
The following functional dependencies are non-trivial:
emp_id -> emp_name (emp_name is not a subset of emp_id)
emp_id -> emp_address (emp_address is not a subset of emp_id)

On the other hand, the following dependencies are trivial:
{emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
Refer: trivial functional dependency.

Completely non trivial FD:
If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

Multivalued dependency

Multivalued dependency occurs when there are more than one independent multivalued attributes in a table.

For example: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

bike_model	manuf_year	color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red

M2222	2009	Black
M2222	2009	Red

Here columns manuf_year and color are independent of each other and dependent on bike_model. In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:

bike_model ->> manuf_year

bike_model ->> color

Transitive dependency in DBMS

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.

X -> Z is a transitive dependency if the following three functional dependencies hold true:

X->Y
Y does not ->X
Y->Z
Note: A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Example: Let's take an example to understand it better:

Book	Author	Author_age
Game of Thrones	George R. R. Martin	66
Harry Potter	J. K. Rowling	49
Dying of the Light	George R. R. Martin	66

{Book} ->{Author} (if we know the book, we knows the author name)

{Author} does not ->{Book}

{Author} -> {Author_age}

Therefore as per the rule of transitive dependency: {Book} -> {Author_age} should hold, that makes sense because if we know the book name we can know the author's age.

Normalization in DBMS: 1NF, 2NF, 3NF and BCNF

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

Anomalies in DBMS
There are three types of anomalies that occur when the database is not normalized. These are: Insertion, update and deletion anomaly.

Example:
A manufacturing company stores the employee details in a table Employee that has four attributes: Emp_Id for storing employee's id, Emp_Name for storing employee's name, Emp_Address for storing employee's address and Emp_Dept for storing the department details in which the employee works. At some point of time the table looks like this:

Emp_Id	Emp_Name	Emp_Address	Emp_Dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

This table is not normalized. We will see the problems that we face when a table in database is not normalized.

Update anomaly:
In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly:
Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if Emp_Dept field doesn't allow null.

Delete anomaly:

Let's say in future, company closes the department D890 then deleting the rows that are having Emp_Dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)

- A relation is said to be in 1NF (first normal form), if it doesn't contain any multi-valued attribute. In other words you can say that a relation is in 1NF if each attribute contains only atomic(single) value only.

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:

Let's say a company wants to store the names and contact details of its employees. It creates a table in the database 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) have two mobile numbers that caused the Emp_Mobile field to have multiple values for these two employees.

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 need to create separate rows for the each mobile number in such a way so that none of the attributes contains multiple values.

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

To learn more about 1NF refer this article: 1NF

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:
Let's say a school wants to store the data of teachers and the subjects they teach. They create a table Teacher 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

This 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 disintegrate 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 are in Second normal form (2NF). To learn more about 2NF refer this guide: 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: Let’s say 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. Further 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 disintegrate 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

DBMS Notes

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	D001	200
1001	Austrian	stores	D001	250
1002	American	design	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	D001	200
stores	D001	250
design	D134	100
Purchasing	D134	600

Emp_Dept_Mapping table:

Emp_Id	Emp_Dept
1001	Production
1001	Qstores
1002	design
1002	Purchasing

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