UNIT-II

1. What is Relational Model? [CO-2, BTL-1]

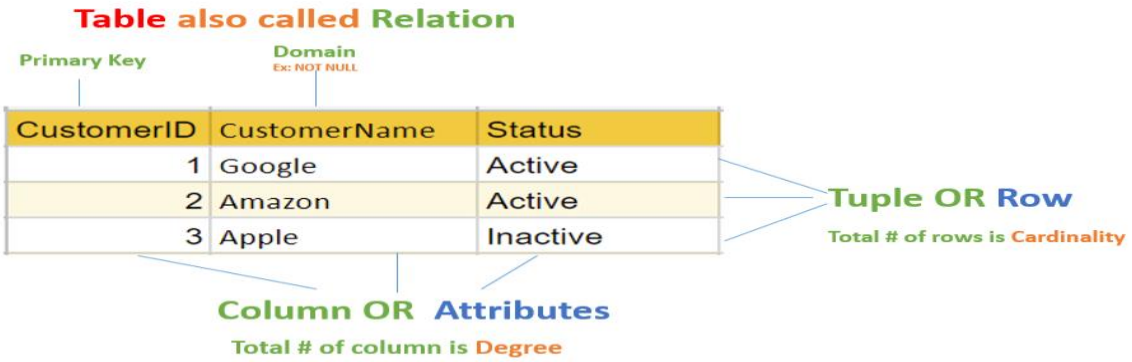
Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
2. **Tuple** – It is nothing but a single row of a table, which contains a single record.
3. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
4. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
5. **Cardinality:** Total number of rows present in the Table.
6. **Column:** The column represents the set of values for a specific attribute.
7. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
8. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain



2. Discuss about various DDL commands with examples [CO-2, BTL-2]

DATA DEFINITION LANGUAGE (DDL)

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

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

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

Syntax :

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

Example:

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DAT E);

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

Syntax

DROP TABLE table_name;

Example

DROP TABLE EMPLOYEE;

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

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;
To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);
EXAMPLE

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

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

Syntax:

TRUNCATE TABLE table_name;
Example:

TRUNCATE TABLE EMPLOYEE;

3. Define a View. Explain the fundamental operations on views with examples. [CO-2, BTL-1]

SQL View
SQL provides the concept of View, which hides the complexity of the data and restricts unnecessary access to the database. It permits the users to access only a particular column rather than the whole data of the table.

The **View** in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement.

Like the SQL tables, Views also store data in rows and columns, but the rows do not have any physical existence in the database.

Any database administrator and user can easily create the View by selecting the columns from one or more database tables. They can also delete and update the views according to their needs.

A view can store either all the records of the table or a particular record from the table using the WHERE clause.

Create a SQL View

You can easily create a View in Structured Query Language by using the CREATE VIEW statement. You can create the View from a single table or multiple tables.

Syntax to Create View from Single Table

CREATE VIEW View_Name **AS**
SELECT Column_Name1, Column_Name2, ..., Column_NameN
FROM Table_Name
WHERE condition;

In the syntax, View_Name is the name of View you want to create in SQL. The SELECT command specifies the rows and columns of the table, and the WHERE clause is optional, which is used to select the particular record from the table.

Syntax to Create View from Multiple Tables

You can create a View from multiple tables by including the tables in the SELECT statement.

CREATE **VIEW** View_Name **AS**
SELECT Table_Name1.Column_Name1, Table_Name1.Column_Name2, Table_Name2.Column_Name2, ..., Table_NameN.Column_NameN
FROM Table_Name1, Table_Name2, ..., Table_NameN
WHERE condition;

Ex. Suppose, you want to create a view with Stu_ID, Stu_Subject, and Stu_Marks of those students whose marks are greater than 85. For this issue, you have to type the following query:

CREATE VIEW Student_View **AS**
SELECT Student_ID, Stu_Subject, Stu_Marks
FROM Student_Details
WHERE Stu_Marks > 85;

Example to Create a View from Multiple tables

Let's consider two tables, **Student_Details** and **Teacher_Details.** The Student_Details table consists of Stu_ID, Stu_Name, Stu_Subject, and Stu_Marks columns. And, the Teacher_Details

table consists of Teacher_ID, Teacher_Name, Teacher_Subject, Teacher_City columns. The data of the Student_Details and Teacher_Details is shown in the following two tables:

Update an SQL View

We can also modify existing data and insert the new record into the view in the Structured Query Language. A view in SQL can only be modified if the view follows the following conditions:

1. You can update that view which depends on only one table. SQL will not allow updating the view which is created more than one table.
2. The fields of view should not contain NULL values.
3. The view does not contain any subquery and DISTINCT keyword in its definition.
4. The views cannot be updatable if the SELECT statement used to create a View contains JOIN or HAVING or GROUP BY clause.
5. If any field of view contains any SQL aggregate function, you cannot modify the view.

Syntax to Update a View

CREATE OR REPLACE **VIEW** View_Name **AS**
SELECT Column_Name1, Column_Name2, ..., Column_NameN
FROM Table_Name
WHERE condition;

Example to Update a View
If we want to update the above Student_View and add the Stu_Name attribute from the Student table in the view, you have to type the following Replace query in SQL:

CREATE OR REPLACE **VIEW** Student_View **AS**
SELECT Student_ID, Stu_Name, Stu_Subject, Stu_Marks
FROM Student_Details
WHERE Stu_Subject = 'Math';
The above statement updates the existing Student_View and updates the data based on the SELECT statement.

Now, you can see the virtual table by typing the following query:

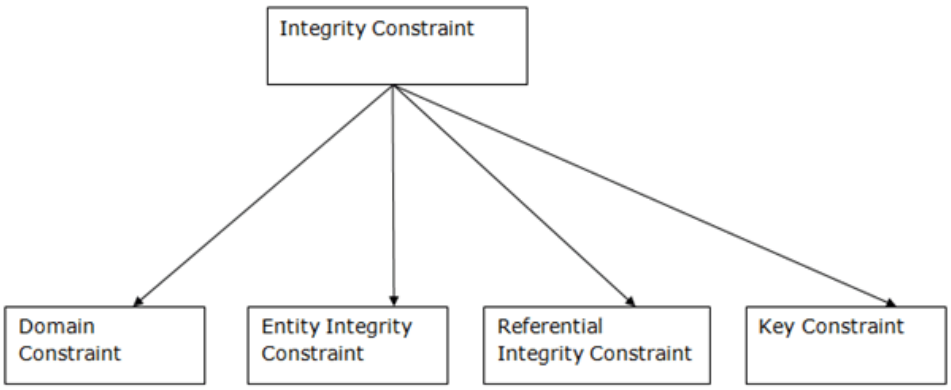
SELECT * **FROM** Student_View;

4. What is integrity Constraint? Discuss the different types of integrity constraints. [CO-2, BTL-2]

Integrity Constraints

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

Types of Integrity Constraint



1. Domain constraints

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

Example:

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

Not allowed. Because AGE is an integer attribute

2. **Entity integrity constraints**

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

Example:

EMPLOYEE

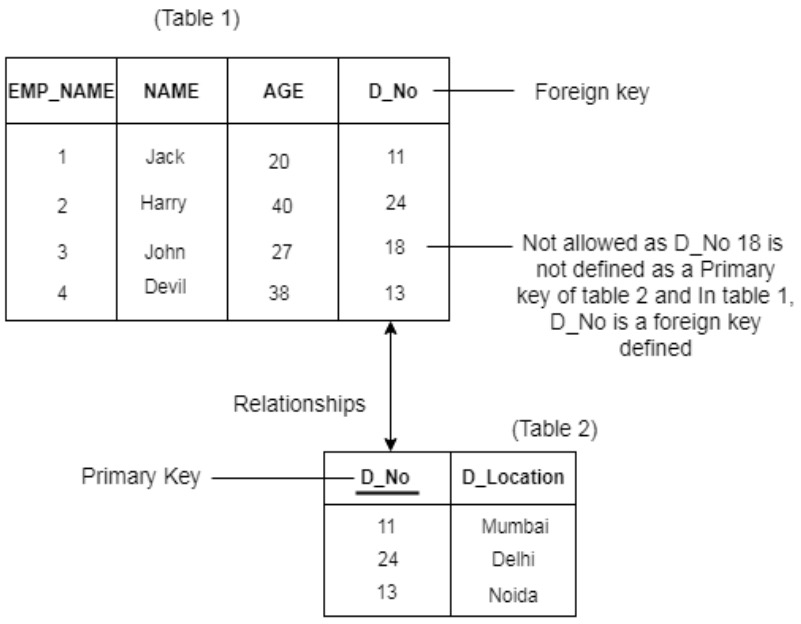
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



4. Key constraints

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

Example:

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

Not allowed. Because all row must be unique

Other constraints

5. CHECK CONSTRAINT

- Whenever a check constraint is applied to the table's column, and the user wants to insert the value in it, then the value will first be checked for certain conditions before inserting the value into that column.
- **For example:** if we have an age column in a table, then the user will insert any value of his choice. The user will also enter even a negative value or any other invalid value. But, if the user has applied check constraint on the age column with the condition age greater than 18. Then in such cases, even if a user tries to insert an invalid value such as zero or any other value less than 18, then the age column will not accept that value and will not allow the user to insert it due to the application of check constraint on the age column.

Syntax to apply check constraint on a single column:

CREATE TABLE TableName (ColumnName1 datatype **CHECK** (ColumnName1 Condition), ColumnName2 datatype,…., ColumnNameN datatype);

Example:

Create a student table and apply CHECK constraint to check for the age less than or equal to 15 while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40), Age **INT CHECK**(Age <= 15));

Syntax to apply check constraint on multiple columns:

CREATE TABLE TableName (ColumnName1 datatype, ColumnName2 datatype **CHECK** (ColumnName1 Condition AND ColumnName2 Condition),…., ColumnNameN datatype);

Example:

Create a student table and apply CHECK constraint to check for the age less than or equal to 15 and a percentage greater than 85 while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Stude nt_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VA RCHAR**(40), Age **INT**, Percentage **INT**, **CHECK**(Age <= 15 AND Percentage > 85));

6. DEFAULT CONSTRAINT

Whenever a default constraint is applied to the table's column, and the user has not specified the value to be inserted in it, then the default value which was specified while applying the default constraint will be inserted into that particular column.

Syntax to apply default constraint during table creation:

CREATE TABLE TableName (ColumnName1 datatype **DEFAULT** Value, ColumnName2 dat atype,…., ColumnNameN datatype);
Example:

Create a student table and apply the default constraint while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Stude nt_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VA RCHAR**(40) **DEFAULT** "anuja.k8@gmail.com");

Syntax to apply default constraint on an existing table's column:

ALTER TABLE TableName **ALTER** ColumnName **SET DEFAULT** Value;
Example:

Consider we have an existing table student. Later, we decided to apply the DEFAULT constraint on the student table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ALTER** Student_Email_ID **SET DEFAULT** "anuja.k8@gma il.com";

5. Define a key. Explain the different types of keys with example. [CO-2, BTL-2]

Keys

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

Types of keys:

1. Primary Key
2. Foreign key
3. Super Key
4. Candidate key
5. Composite key

SQL PRIMARY KEY

A column or columns is called **primary key (PK)** that *uniquely identifies each row in the table*.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

When multiple columns are used as a primary key, it is known as **composite primary key**.

In designing the composite primary key, you should use as few columns as possible. It is good for storage and performance both, the more columns you use for primary key the more storage space you require.

In terms of performance, less data means the database can process faster.

Points to remember for primary key:

- Primary key enforces the entity integrity of the table.
- Primary key always has unique data.
- A primary key length cannot be exceeded than 900 bytes.
- A primary key cannot have null value.
- There can be no duplicate value for a primary key.
- A table can contain only one primary key constraint.

Main advantage of primary key:
The main advantage of this uniqueness is that we get **fast access**.

SQL primary key for one column:
The following SQL command creates a PRIMARY KEY on the "S_Id" column when the "students" table is created.

MySQL:

CREATE TABLE students (
S_Id **int** NOT NULL,
LastName **varchar** (255) NOT NULL,
FirstName **varchar** (255),
Address **varchar** (255),

City **varchar** (255),
PRIMARY KEY (S_Id)) ;

SQL primary key for multiple columns:

```
CREATE TABLE students (
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
CONSTRAINT pk_StudentID PRIMARY KEY (S_Id, LastName) ) ;
```

SQL primary key on ALTER TABLE

When table is already created and you want to create a PRIMARY KEY constraint on the "S_Id" column you should use the following SQL:

Primary key on one column:

```
ALTER TABLE students
ADD PRIMARY KEY (S_Id);
```

Primary key on multiple column:

```
ALTER TABLE students
ADD CONSTRAINT pk_StudentID PRIMARY KEY (S_Id,LastName) ;
```

DROP a PRIMARY KEY constraint

DROP (remove) a primary key constraint, you should use following syntax:

```
ALTER TABLE students DROP PRIMARY KEY ;
```

SQL FOREIGN KEY

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables. In simple words you can say that, a foreign key in one table used to point primary key in another table.

Let us take an example to explain it:

Here are two tables first one is students table and second is orders table.

Here orders are given by students.

First table:

S_Id	LastName	FirstName	CITY
1	MAURYA	AJEET	ALLAHABAD
2	JAISWAL	RATAN	GHAZIABAD
3	ARORA	SAUMYA	MODINAGAR

Second table:

O_Id	OrderNo	S_Id
1	99586465	2
2	78466588	2
3	22354846	3
4	57698656	1

- The "S_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
- The "S_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The foreign key constraint is generally prevents action that destroy links between tables.

It also prevents invalid data to enter in foreign key column.

SQL FOREIGN KEY constraint ON CREATE TABLE:
(Defining a foreign key constraint on single column)

To create a foreign key on the "S_Id" column when the "Orders" table is created:

MySQL:

CREATE TABLE orders (
O_Id **int** NOT NULL,
Order_No **int** NOT NULL,

S_Id **int**,
PRIMAY **KEY** (O_Id),
FOREIGN KEY (S_Id) **REFERENCES** Persons (S_Id)) ;

SQL FOREIGN KEY constraint for ALTER TABLE:
If the Order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

Defining a foreign key constraint on single column:

ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY(S_Id)
REFERENCES Students (S_Id) ;

DROP SYNTAX for FOREIGN KEY COSTRAINT:
If you want to drop a FOREIGN KEY constraint, use the following syntax:

MySQL:

ALTER TABLE Orders
DROP **FOREIGN KEY** fk_PerOrders;

COMPOSITE KEY

A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness.

Sometimes more than one attributes are needed to uniquely identify an entity. A primary key that is made by the combination of more than one attribute is known as a composite key.

In other words we can say that: Composite key is a key which is the combination of more than one field or column of a given table. It may be a candidate key or primary key.

Columns that make up the composite key can be of different data types.

SQL Syntax to specify composite key:

CREATE TABLE TABLE_NAME
(COLUMN_1, DATA_TYPE_1,
COLUMN_2, DATA_TYPE_2,

PRIMARY KEY (COLUMN_1, COLUMN_2, ...));
In all cases composite key created consist of COLUMN1 and COLUMN2.

MySQL:

```
CREATE TABLE SAMPLE_TABLE
(COL1 integer,
COL2 varchar(30),
COL3 varchar(50),
PRIMARY KEY (COL1, COL2));
```

UNIQUE KEY

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

There is an automatically defined unique key constraint within a primary key constraint.

There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

SQL UNIQUE KEY constraint on CREATE TABLE:

If you want to create a UNIQUE constraint on the "S_Id" column when the "students" table is created, use the following SQL syntax:

```
CREATE TABLE students  (
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
City varchar (255),
UNIQUE (S_Id)  )  ;
```

```
CREATE TABLE students  (
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
City varchar (255),
CONSTRAINT uc_studentId UNIQUE (S_Id, LastName) )  ;
```

CANDIDATE KEY

What is a Candidate Key

A candidate key is a subset of a super key set where the key which contains no redundant attribute is none other than a **Candidate Key**. In order to select the candidate keys from the set of super key, we need to look at the super key set.

Role of a Candidate Key

The role of a candidate key is to identify a table row or column uniquely. Also, the value of a candidate key cannot be Null. The description of a candidate key is "no redundant attributes" and being a "minimal representation of a tuple," according to the Experts.

Example of Candidate Key

Let's look at the same example took while discussing Super Key to understand the working of a candidate key.

We have an **EMPLOYEE_DETAIL** table where we have the following attributes:

Emp_SSN: The SSN number is stored in this field.

Emp_Id: An attribute that stores the value of the employee identification number.

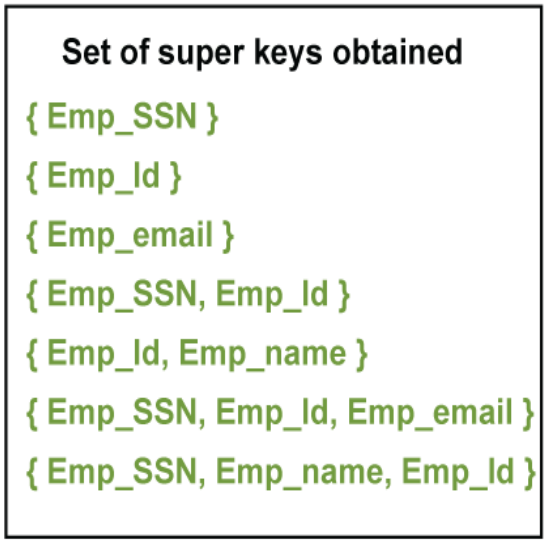
Emp_name: An attribute that stores the name of the employee holding the specified employee id.

Emp_email: An attribute that stores the email id of the specified employees.

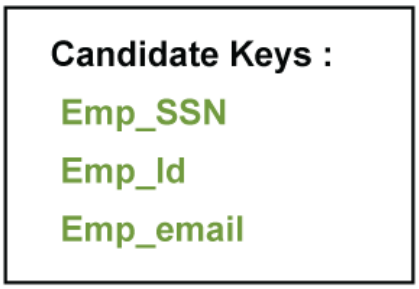
The **EMPLOYEE_DETAIL** table is given below that will help you understand better:

Emp_SSN	Emp_Id	Emp_name	Emp_email
11051	01	John	john@email.com
19801	02	Merry	merry@email.com
19801	03	Riddle	riddle@email.com
41201	04	Cary	cary@email.com

So, from the above table, we obtained the below given super keys (discussed in the previous section):



Now, from these sets of super keys, we can conclude the candidate keys. In order to pick up the candidate keys, the best way is to analyze and form the primary keys as much as we can. So, we need to identify those sets from the super key sets that alone can identify the whole table, or we can say the other attributes of the table. Thus, the result is:



So, these are the three attributes obtained that can identify the other non-prime attributes of the table. All these are the candidate keys and from which we can pick the most appropriate attribute that can easily identify all records of the table, which will be described as the Primary key.

6. Explain the logical database design with example [CO-2 , BTL-1]

Logical database design

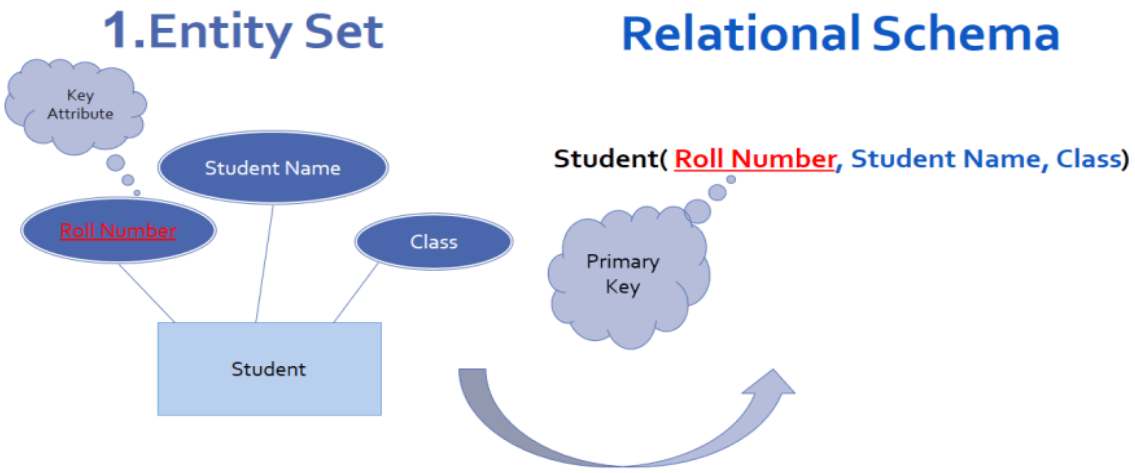
Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model.

- Entity in ER Model is changed into tables, or we can say for every Entity in ER model, a table is created in Relational Model.
- And the **attributes** of the Entity gets converted to columns of the table.

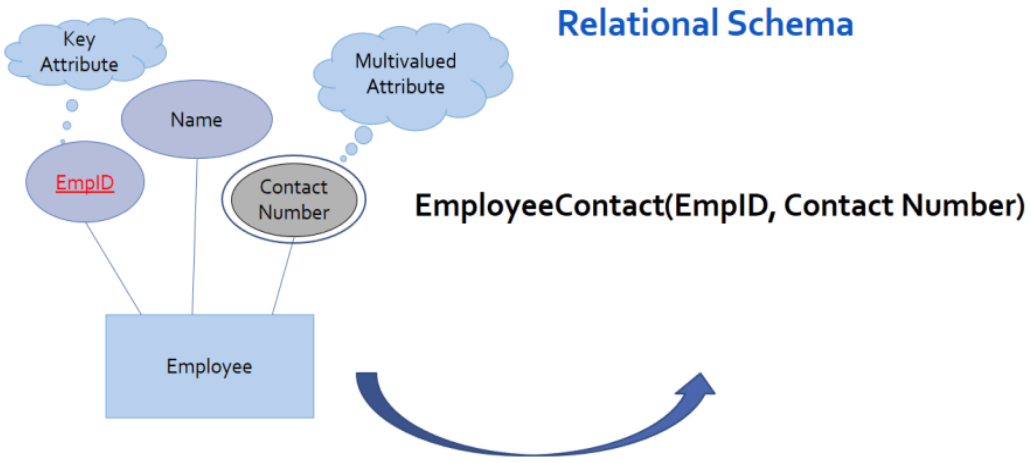
- And the primary key specified for the entity in the ER model, will become the primary key for the table in relational model.
- ER diagram is converted into the tables in relational model.
- This is because the relational model can be easily implemented by MYSQL Oracle etc.

Entity

- Consider we have entity STUDENT in ER diagram with attributes Roll Number, Student Name and Class.



- **Entity set with multi valued attribute:**

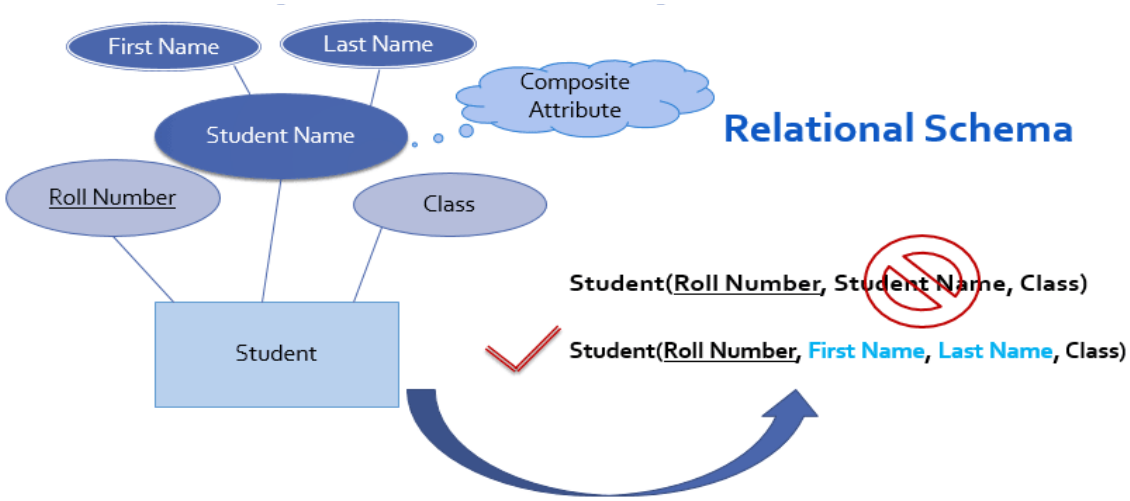


Consider we have entity set Employee with attributes Employee ID, Name and Contact number.

Hence to convert entity with multivalued attribute into relational schema separate relation is created for multivalued attribute

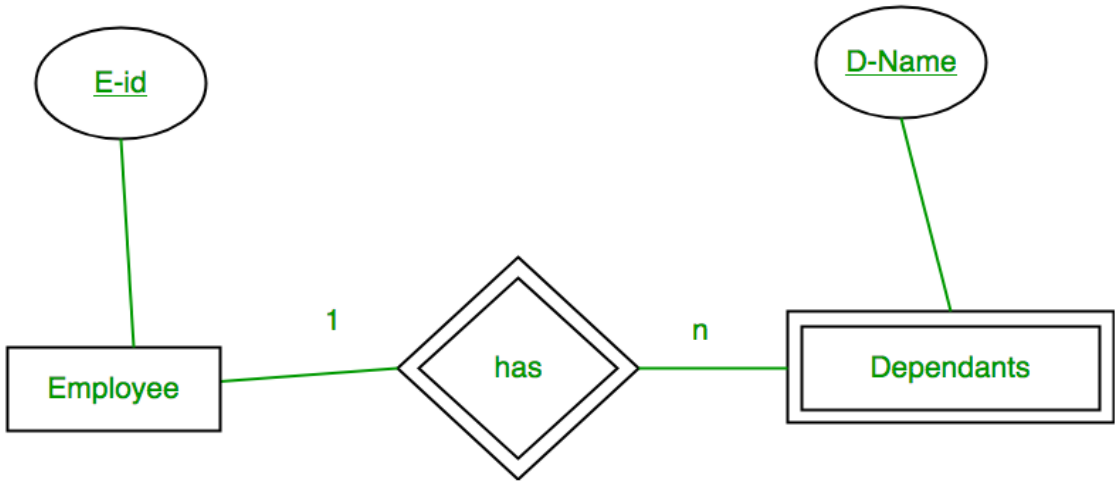
Entity set with Composite attribute:

- Consider entity set student with attributes Roll Number, Student Name and Class. here student name is composite attribute as it has further divided into First name, last name.



1:M (one to many) Relationship:

- Consider 1:M relationship set enrolled exist between entity sets student and course as follow,



First Convert each entity and relationship to tables. Employee table corresponds to Employee Entity with key as E-Id. Similarly Dependents table corresponds to Dependent Entity with key as D-Name and E-Id. represents the relationship between Employee and Dependents (Which employee has which dependents). So it will take attribute E-Id from Employee and D-Name from Dependents.

7.Consider the following Relations and answer the queries [CO-2,BTL-3]

An instance of S3 of Sailors

Sid	Sname	Rating	Age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

An instance of R2 of Reserves

Sid	Bid	Day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

An instance of B1 of Boats

Bid	Bname	Color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

a) Find all sailors with rating above 7.

select * from sailors where rating>7;

b) Find the names of sailors who have reserved boat number 103.

select sname from sailors s, reserves r where r.bid=103 and s.sid=r.sid;

c) Write a query to find total number of boats available.

select count(bid) from boats;

d) Find the names of sailors who have reserved a red or a green boat.

select sname from sailors s, reserves r, boats b where s.sid=r.sid and b.bid=r.bid and b.color='red' or 'green';

e) Find sid of sailors who have reserved red boat.

select sid from boats b, reserves r where b.bid=r.bid and b.color='red;

f) Find the name and age of youngest sailor.

Select sname, age from sailors where age=(select min(age) from sailors);

g) Find the names of sailors with highest rating.

Select sname, rating from sailors where rating=(select max(rating) from sailors);

h) Find the names of Boats starting with 'C' and ending with 'r'.

select banme from boats where bname like 'C%r';

i) Write a query to find total number of boats available.

select count(bid) from boats;

j) Find all information of sailors who have reserved boat number 101.

select * from sailors s, reserves r where r.bid=101 and s.sid=r.sid;

k) Calculate the average age of all sailors.

Select avg(age) from sailors;

l) Find the names of sailors who have reserved at least one boat.

Select s.sname from sailors s, reserves r where s.sid=r.sid;

m) Find the name and the age of the oldest sailor.

Select sname, age from sailors where age=(select max(age) from sailors);