UNIT-II

What is Relational Model?

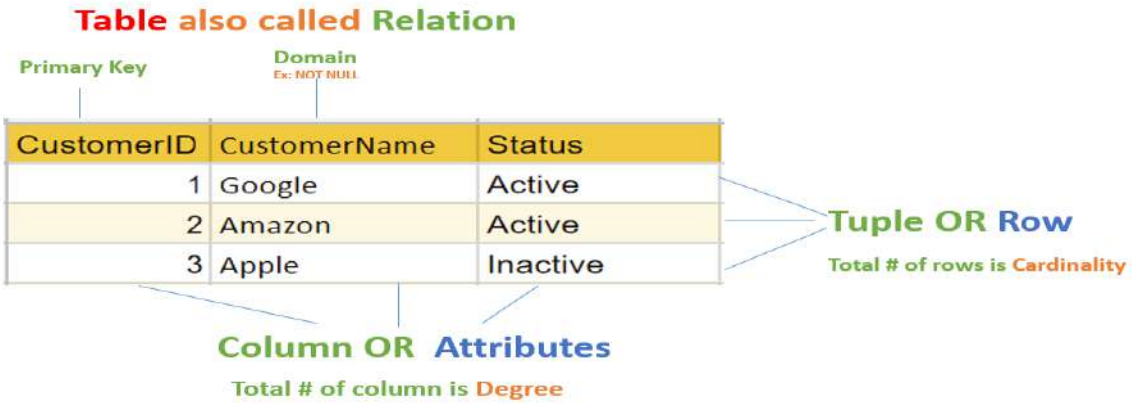
Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

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

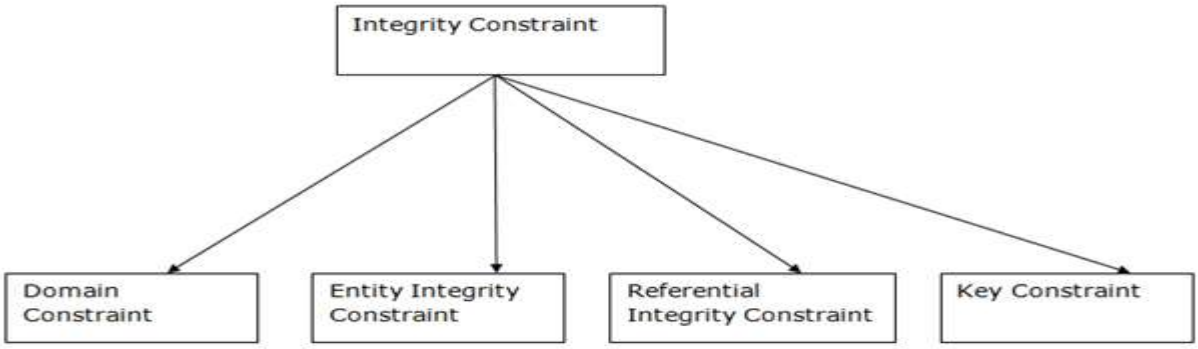
1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree (Arity)**: The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain
11. **Relational database** – A collection of relations with distinct relation names
12. **Relational database schema** - The collection of schemas for the relations in the database



Integrity Constraints:

- An integrity constraint (IC) is a condition specified on a database schema and restricts the data that can be stored in an instance of the database.
- ICs are specified when schema is defined by DBA or end user.
- IC's are used to ensure accuracy and consistency of the data in relational database.
- ICs are checked when relations are modified.
- IC's are set of rules that database is not permitted to violate.
- If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
- Avoids data entry errors, too!
- IC's 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.
- Domain Constraints are user-defined columns that help the user to enter the value according to the data type.
- And if it encounters a wrong input it gives the message to the user that the column is not fulfilled properly.
- Or in other words, it is an attribute that specifies all the possible values that the attribute can hold like integer, character, date, time, string, etc.
- It defines the domain or the set of values for an attribute and ensures that the value taken by the attribute must be an atomic value(Can't be divided) from its domain

Example:

Eid	Ename	Job	Sal	Location	Deptno
100	Rama	Scientist	102000	Hyd	12
101	Srikar	Manager	80300	Hyd	20
102	Krishana	Analyst	95600	4578	24

Here Emp_id 102 , location 4578 is not allowed because location is character 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.

Eid	Ename	Job	Sal	Location	Deptno
100	Rama	Scientist	102000	Hyd	12
101	Srikar	Manager	80300	Hyd	20
	Krishana	Analyst	95600	Pune	24

Example:

In the above employee table Emp_id as primary key and can't contain NULL value.

3. Referential Integrity Constraints(Foreign Key)

- A referential integrity constraint is specified between two tables.
- The foreign key a constraint is a column or list of columns that points to the primary key column of another table
- 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.
- The main purpose of the foreign key is only those values are allowed in the present table that will match the primary key column of another table

Example:

Emp (Table-1)

Primary Key

Foreign Key

Eid	Ename	Job	Sal	Location	Deptno
100	Rama	Scientist	102000	Hyd	12
101	Srikar	Manager	80300	Hyd	20
102	Krishana	Analyst	95600	Pune	24

Not allowed Deptno 20 because it is not defined as a primary key in the Dept table. Dept_no is a foreign key in the Emp table.

Primary Key

Relationships

Dept (Table-2)

Deptno	Dname	Location
12	Accounts	Hyd
17	Exam Branch	Hyd
24	Research	Pune

4. **Key Constraints or Uniqueness Constraints**

1. These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
2. Keys are the entity set that is used to identify an entity within its entity set uniquely.
3. 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.
4. A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as the primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
5. Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

Example:

In the below example Eid is primary key but Eid have the same value 100. so, it is violating the key constraint

Eid	Ename	Job	Sal	Location	Deptno
100	Rama	Scientist	102000	Hyd	12
101	Srikar	Manager	80300	Hyd	20
100	Krishana	Analyst	95600	Pune	24

Other Key Constraints:

1. **NOT NULL**
2. **NULL**
3. **PRIMARY KEY**
4. **FOREIGN KEY**
5. **UNIQUE**
6. **CHECK**

NOT NULL Constraint:

The NOT NULL constraint in SQL is used to ensure that a column in a table doesn't contain NULL (empty) values, and prevent any attempts to insert or update rows with NULL values.

NOT NULL Ensures that a column cannot have a NULL value.

The NOT NULL constraint enforces a column to NOT accept NULL values, which means that you cannot insert a new record, or update a record without adding a value to this field.

NULL represents a record where data may be missing data or data for that record may be optional

Once not null is applied to a particular column, you cannot enter null values to that column and restricted to maintain only some proper value other than null

A not-null constraint cannot be applied at table level

NOT NULL on CREATE TABLE

```
CREATE TABLE Persons (
   PID int (3) NOT NULL,
   PName varchar (10) NOT NULL,
   PADD varchar (20) NOT NULL,
   PAge int (2)
 );
```

NULL Constraint:

- By default, a column can hold NULL values

The term **NULL** in SQL is used to specify that a data value does not exist in the database. It is not the same as an empty string or a value of zero, and it signifies the absence of a value or the unknown value of a data field.

Some common reasons why a value may be NULL −

- The value may not be provided during the data entry.
- The value is not yet known.

It is important to understand that you cannot use comparison operators such as “=”, “<”, or “>” with NULL values. This is because the NULL values are unknown and could represent any value. Instead, you must use “IS NULL” or “IS NOT NULL” operators to check if a value is NULL

Syntax

The basic syntax of **NULL** while creating a table.

```
CREATE TABLE CUSTOMERS (
  ID   INT(3)            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT(2)            NOT NULL,
  ADDRESS  VARCHAR (25) ,
  SALARY   FLOAT(6, 2),
   );
```

In the above example SALARY and ADDRESS are NULL constraints.

PRIMARY KEY Constraint :

The PRIMARY KEY constraint uniquely identifies each record in a table.

- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE Persons (
    PID int (3) NOT NULL,
    PName varchar (10) NOT NULL,
    PADD varchar (20) NOT NULL,
    PAge int (2),
 PRIMARY KEY (ID)
);
```

FOREIGN KEY Constraint :

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

FOREIGN KEY constraints enforce referential integrity.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Only students listed in the students relation should be allowed to enroll for courses.

```
CREATE TABLE students
(
 Sid char(20),
 name varchar(20),
login varchar(15),
age int(2),
gpa float(2,2),
 Primary key(sid));
```

```
CREATE TABLE Enrolled
(
sid CHAR(20),  cid CHAR(20),  grade CHAR(2),
PRIMARY KEY (Sid, Cid),
FOREIGN KEY (sid) REFERENCES Students(sid));
```

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

UNIQUE Constraint:

The UNIQUE constraint in SQL is used to ensure that no duplicate values will be inserted into a specific column or combination of columns that are participating in the UNIQUE constraint and not part of the PRIMARY KEY.
In other words, the index that is automatically created when you define a UNIQUE constraint will guarantee that no two rows in that table can have the same value for the columns participating in that index, with the ability to insert only NULL value to these columns.

The UNIQUE constraint ensures that all values in a column are different.
Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns
A PRIMARY KEY constraint automatically has a UNIQUE constraint.
Difference between UNIQUE constraints and PRIMARY KEY constraint is unique allows NULL values whereas primary key not allowed the NULL values.

```
CREATE TABLE Persons (
    PID int (3)  UNIQUE,
    PName varchar (10) NOT NULL,
    PADD varchar (20),
    PAge int (2),
   Padhar varchar(12)
 PRIMARY KEY (Padhar)
);
```

CHECK Constraint:

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    PID int (3),
    PName varchar (10) NOT NULL,
    PAdd varchar (20) NOT NULL,
    PAge int (2),
    PRIMARY KEY(PID),
    CHECK (PAge>=18)
);
```

If we enter the Page below 18 then shows that CHECK Constraint is violated.

Enforcing integrity constraints:

Data integrity refers to the correctness and completeness of data within a database. To enforce data integrity, you can constrain or restrict the data values that users can insert, delete, or update in the database.

For example, the integrity of data in the pubs2 and pubs3 databases requires that a book title in the titles table must have a publisher in the publishers table. You cannot insert books that do not have a valid publisher into titles, because it violates the data integrity of pubs2 or pubs3.

Transact-SQL provides several mechanisms for integrity enforcement in a database such as rules, defaults, indexes, and triggers. These mechanisms allow you to maintain these types of data integrity:

- Requirement – requires that a table column must contain a valid value in every row; it cannot allow null values. The create table statement allows you to restrict null values for a column.

- Check or validity – limits or restricts the data values inserted into a table column. You can use triggers or rules to enforce this type of integrity.

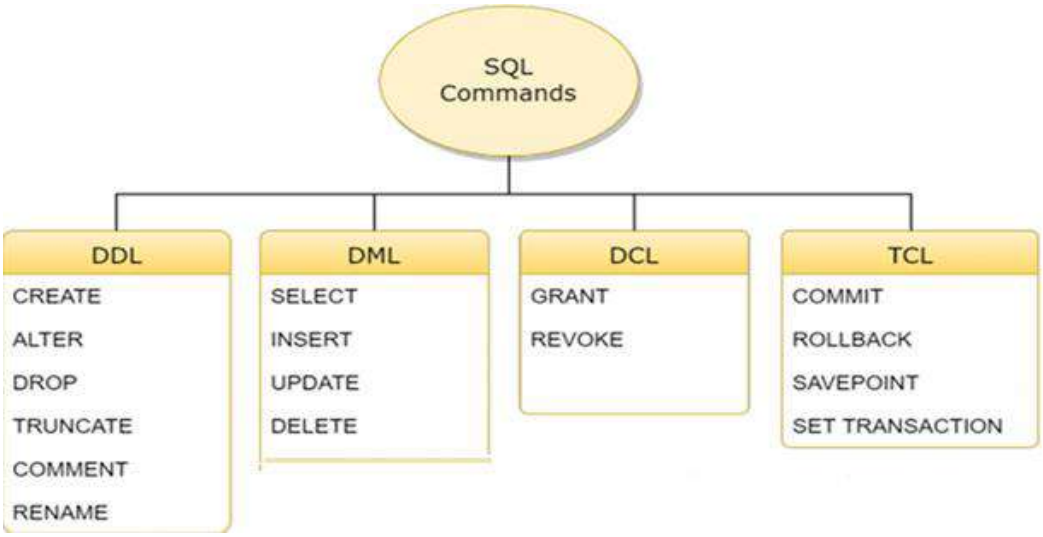
- Uniqueness – no two table rows can have the same non-null values for one or more table columns. You can use indexes to enforce this integrity.

- Referential – data inserted into a table column must already have matching data in another table column or another column in the same table. A single table can have up to 192 references.

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)



Types of SQL

What is DDL?

Data Definition Language helps you to define the database structure or schema. Let's learn about DDL commands with syntax.

CREATE:

To create a database and its objects like (table, index, views, store procedure, function, and triggers)

The CREATE TABLE statement is used to create a new table in a database.

Syntax

CREATE TABLE table_name (

column1 datatype(size),

column2 datatype(size),

column3 datatype(size),

....

);

•

Example: create table emp (eid int(5), ename varchar(15));

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold

(e.g. varchar, integer, date, etc.)

DROP

Drops commands remove tables and databases from RDBMS.

To drop a Table

Syntax: DROP TABLE tablename;

Ex: DROP TABLE emp ;

To drop a database

syntax: DROP DATABASE databasename;

Ex: Drop database kmitcsma;

ALTER: This command is used to modify the structure of the table. Using this command we can perform 4 operations on table they are:

1. ALTER-MODIFY
2. ALTER-ADD
3. ALTER-DROP

v **ALTER-MODIFY**: This command is used to increase or decrease the size of the data type and also change the data type from old data type to new data type.

***Syntax*:** ALTER TABLE tablename MODIFY columnname datatype(size);

Example: alter table emp modify ename varchar(25);

Syntax to modify more than one column:

Alter table tablename modify (columnname-1 datatype(size), columnname-2 datatype(size),…….,columnname-n datatype(size));

ALTER-ADD: This command is used to add the new column to the existing table.

Syntax: ALTER TABLE tablename ADD (columnname-1 datatype(size), columnname-2 datatype(size),…….,columnname-n datatype(size));

Example: alter table emp add(dob date, mobileno int(10));

Constraints can be added using ALTER command:

ALTER TABLE emp

Add primary key (eid);

ALTER-DROP: This command is used to remove the columns from the existing table.

Syntax: ALTER TABLE tablename DROP column columnname;

Example: alter table employee drop column mobileno;

TRUNCATE:
This command 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 students;

COMMENT:

Comments are used to explain sections of SQL statements.

Comments can be written in the following three formats:

1.Single-line comments

2.Multi-line comments

3.In-line comments

1.Single Line Comments

–Comments starting and ending in a single line are considered single-line comments. A line starting with '- - is a comment and will not be executed.

Ex: create database CSMA; --THIS IS SINGLE LINE COMMENT:

2.Multi-Line Comments

–Comments starting in one line and ending in different lines are considered as multi-line comments.

–A line starting with '/*' is considered as starting point of the comment and is terminated when '*/' is encountered.

Ex: /*This is for multiline comments

specifying the database is created:*/

Create database kmitCSMA;

3. In-Line Comments

–In-line comments are an extension of multi-line comments, comments can be stated in between the statements and are enclosed in between '/*' and '*/'.

SYNTAX:

```
SELECT column name
 /* This column contains the name of the customer / order_date /
 This column contains the date the order was placed */
 FROM tablename;
```

EX:

```
SELECT ename
 /* This column contains the name of the customer / order_date /
 This column contains the date the order was placed */
 FROM emp
```

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

- INSERT
- SELECT
- UPDATE
- DELETE

INSERT:

This is a SQL query. This command is used to insert data into the row of a table.

Tuples are inserted, using the INSERT command. We can insert a single tuple OR multiple tuples into the table.

Two(2) ways of inserting rows/tuples into table

1. By specifying column names

2. Without specifying column names

1.By specifying column names

Syntax:

INSERT INTO table_name (column1, column2, column3....)

VALUES (value1, value2, value3.....);

Example: Insert into emp(eid, ename, eadd, dob, mobileno) values (100,'srikar',;Hyderabad','2023-04-20');

2.Without specifying column names

Syntax: INSERT INTO table_name

VALUES (value1, value2, value3....);

Ex: Insert into emp values (100,'rama',;Hyderabad','2023-04-20');

Inserting multiple rows into the table at once.

Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3,.... col N) VALUES

(value1, value2, value3, ... valueN),

(value1, value2, value3, ... valueN),

……

value1, value2, value3, ... valueN)

);

Example:

Insert into emp(eid, ename, eadd, dob, mobileno) values
(100,’rama’,;Hyderabad’,’2023-04-20’),

(101, ‘krishna’, ‘Hyderabad’, ‘2023-03-29’)

(102, ‘srikar’, ‘Atlanta’, ‘2022-08-19’)

(103, ‘srinidhi’, ‘USA’, ‘2005-02-19’)

SELECT:

This command is used to retrieve the data from the existing table. Using this command we can retrieve all the records and also we can retrieve specific records in the table.

Syntax to select all rows and columns:

Syntax: SELECT * FROM tablename;

Example: select * from employee;

Syntax to select selected columns and all rows:

Syntax: SELECT column1, column2... from tablename;

Here, column1, column2, ... are the field names of the table you want to select data from.

Example: select eid, ename from employee;

Syntax to select selected rows and selected columns:

Syntax: SELECT column1, column2.. from tablename where condition;

Example: select eid, ename from employee where eid=101;

UPDATE:

UPDATE command is used to modify the data in the table. Using this command, we can modify all the records in the table and also we can modify specific records in the table.

Syntax to update all records in a single column:

Syntax: UPDATE tablename SET col-1=value1, col-2=value2..;

Example: update employee set ename='sachin';

Syntax to update a particular record (using where clause):

Syntax: UPDATE tablename set column name=value where condition;

Example: update employee set deptno=20 where eid=101;

DELETE:

Delete command is used to delete the records from the existing table. Using this command we can delete all records and also specific record in the table (using where clause).

Syntax for deleting all records:

Syntax: DELETE FROM tablename;

Example: delete from employee;

Syntax for deleting a particular record:

Syntax: DELETE from tablename where condition;

Example: delete from employee where eid=102;

What is DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions." Other permission controls parameters of the database system.

Examples of DCL commands:
Commands that come under DCL:

- Grant
- Revoke

Grant:
This command is use to give user access privileges to a database.

Syntax:

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
For example:

GRANT SELECT ON Users TO 'Tom'@'localhost;

Revoke:
It is useful to back permissions from the user.

Syntax:

REVOKE privilege_name ON object_name FROM {user_name |PUBLIC |role_name}
For example:

REVOKE SELECT, UPDATE ON student FROM BCA, MCA;

What is TCL?

Transaction control language or TCL commands deal with the transaction within the database.

Commit
This command is used to save all the transactions to the database.

Syntax:

Commit;
For example:

DELETE FROM Students WHERE RollNo =25;
COMMIT;

Rollback
Rollback command allows you to undo transactions that have not already been saved to the database.

Syntax:

ROLLBACK;

Example:

DELETE FROM Students WHERE RollNo =25;

SAVEPOINT

This command helps you to sets a savepoint within a transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

Example:

SAVEPOINT RollNo;

Logical Database Design:

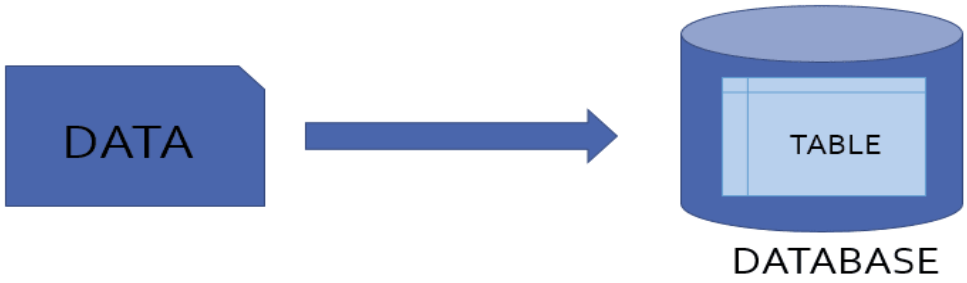
ER Diagram to Relational Model Conversion

First step of any relational database design is to make ER Diagram for it and then convert it into relational Model.

What is relational model?

Relational Model represents how data is stored in database in the form of table

.

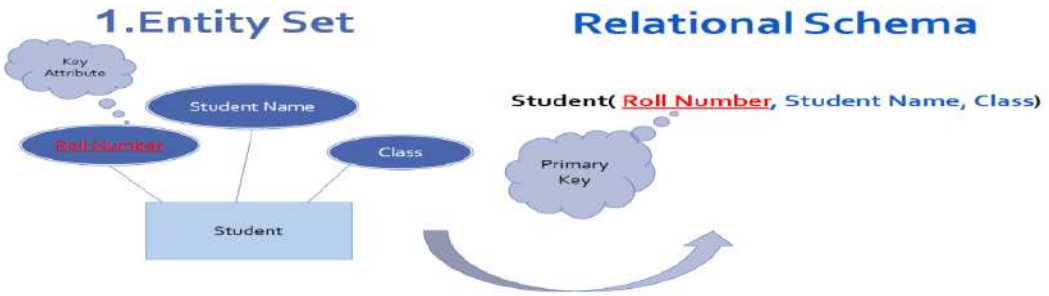


Lets learn step by step how to convert ER diagram into relational model

1.Entity Set:

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

To convert this entity set into relational schema
1.Entity is mapped as relation in Relational schema
2.Attributes of Entity set are mapped as attributes for that Relation.
3.Key attribute of Entity becomes Primary key for that Relation.

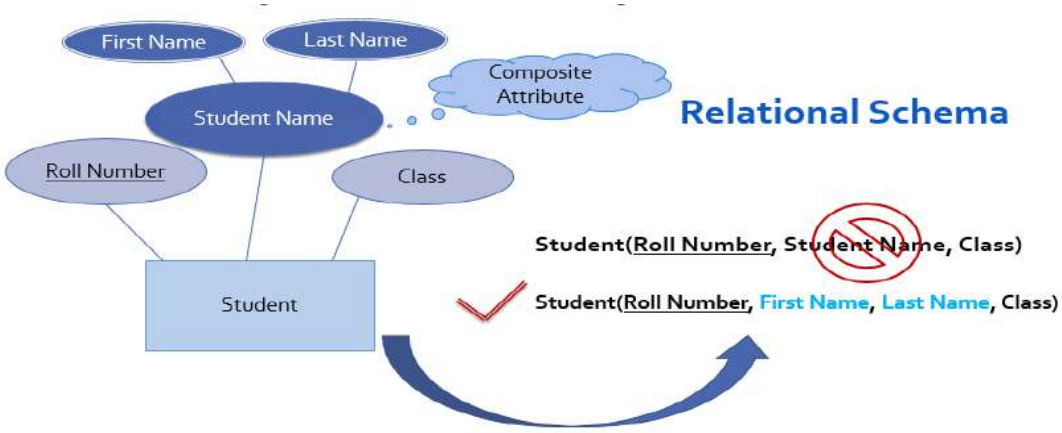


2.Entity set with multi valued attribute:

Consider we have entity set Employee with attributes Employee ID, Name and Contact number. Here contact number is multivalued attribute as it has multiple values. as an employee can have more than one contact number for that we have to repeat all attributes for every new contact number. This will lead to data redundancy in table.
Hence to convert entity with multivalued attribute into relational schema
separate relation is created for multivalued attribute in which
1.Key attribute and multivalued attribute of entity set becomes primary key of relation.
2. Separate relation employee is created with remaining attributes.
Due to this instead of repeating all attributes of entity now only one attribute is need to repeat.

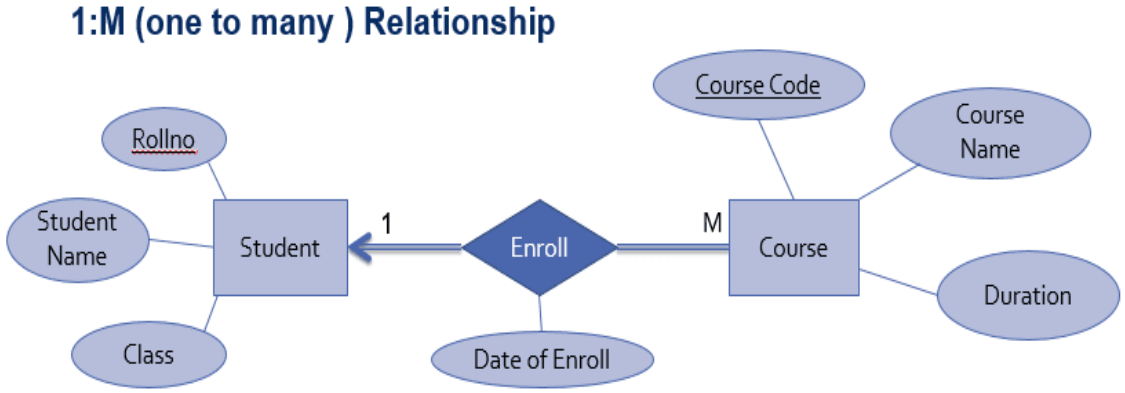
3.Entity set with Composite attribute:

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.
In this case to convert entity into relational schema,
composite attribute student name should not be include in relation but all parts of composite attribute are mapped as simple attributes for relation.



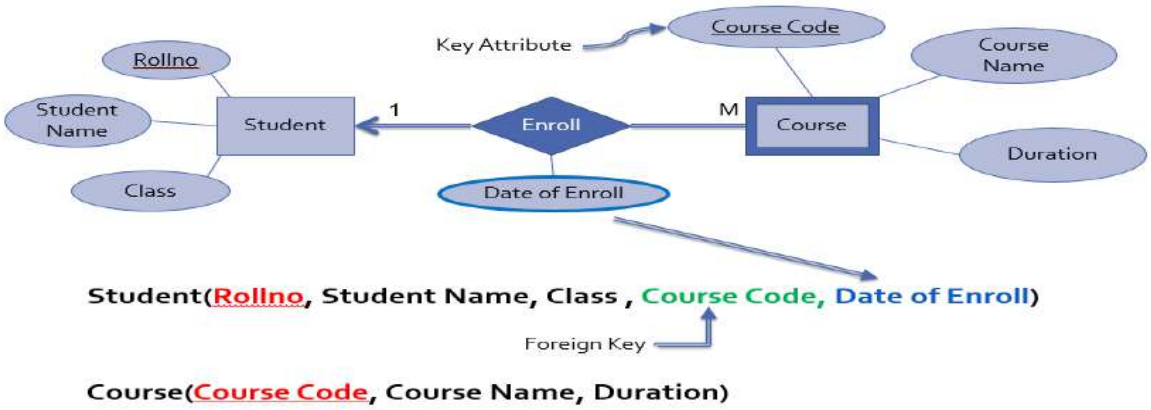
4. 1:M (one to many) Relationship:

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



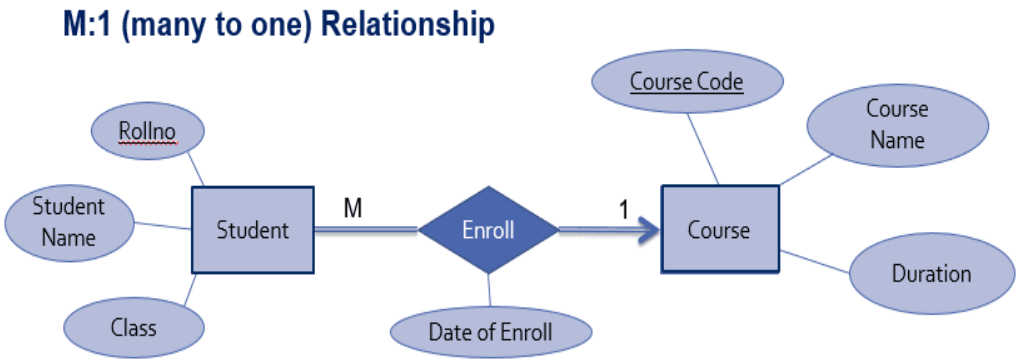
Attributes of entity set student are Roll no which is primary key, student name and class
Attributes of entity set course are Course code which is primary key, Course name and duration
And date of enroll is attribute of relationship set enroll.
Here Enroll is 1:M relationship exist between entity set student and course which means that one student can enroll in multiple courses.

In this case to convert this relationship into relational schema,
1.Separate relation is created for all participating entity sets (student and course)
2.Key attribute of Many's side entity set (course) is mapped as foreign key in one's side relation(Student)
3.All attributes of relationship set are mapped as attributes for relation of one's side entity set (student)

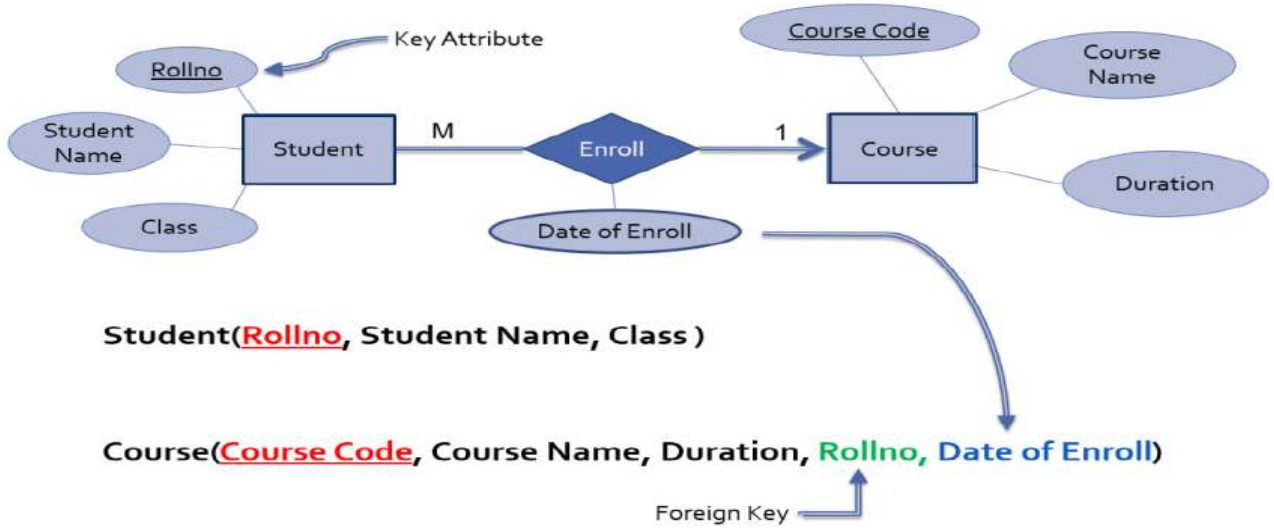


5. M:1 (many to one) Relationship:

Consider same relationship set enroll exist between entity sets student and course . but here student is many side entity set while course is one side entity set. Which means many student can enroll in one course.

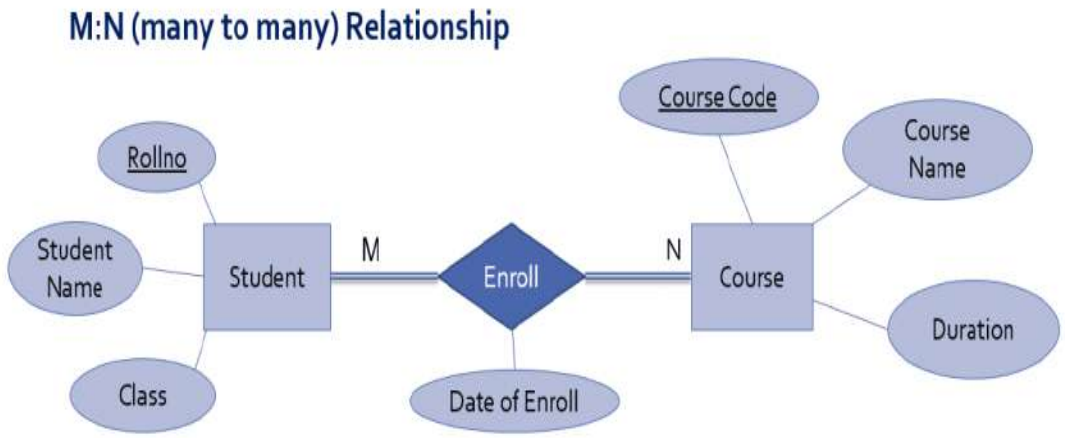


To convert this relationship set into relational schema,
1.Separate relation is created for all participating entity sets.
2.Key attribute of Many's side entity set student is mapped as foreign key in one's side relation
3.All attributes of relationship set are mapped as attributes for one's side relation course.



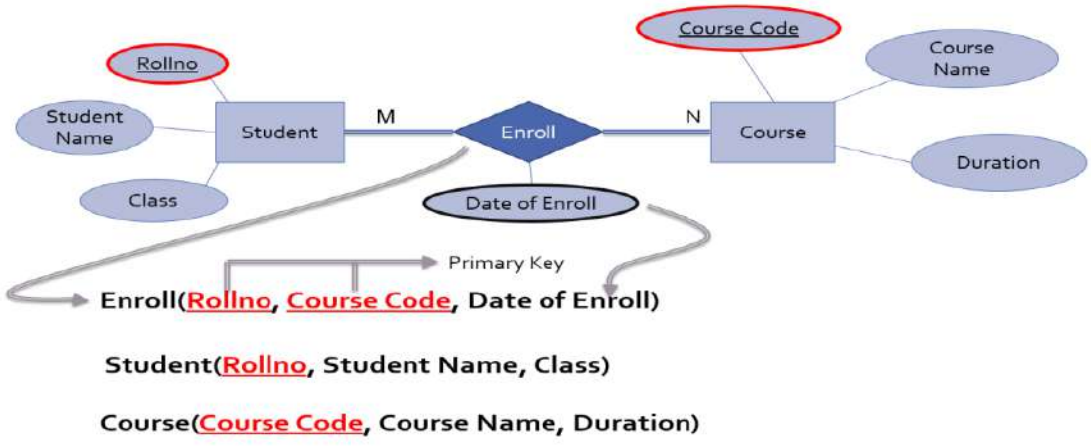
6. M:N (many to many) Relationship:

Consider same relationship set enrolled exist between entity sets student and course ,which means multiple student can enroll in multiple courses.



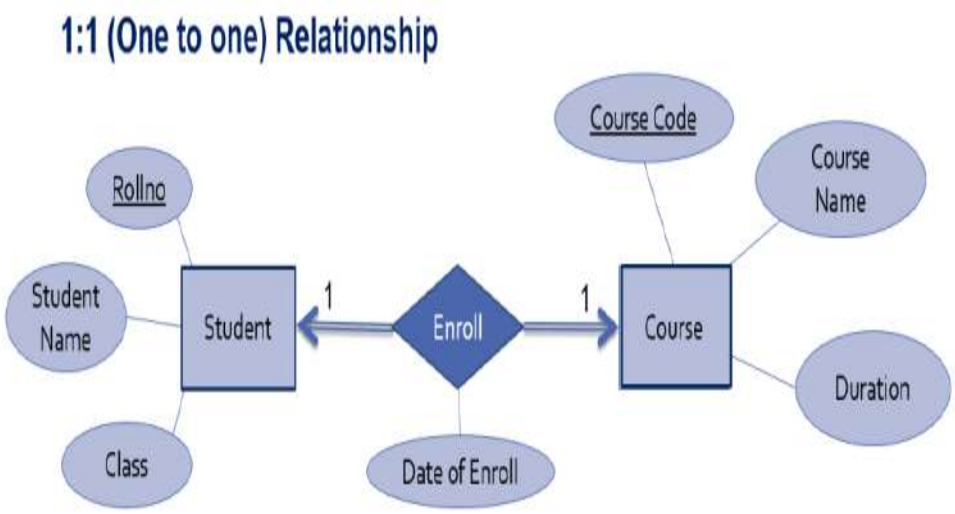
To convert this Relationship set into relational schema,

1. Relationship set is mapped as separate relation
2. Key attributes of participating entity sets are mapped as primary key for that relation
3. Attribute of relationship set becomes simple attributes for that relation
4. And separate relation is created for other participating entities



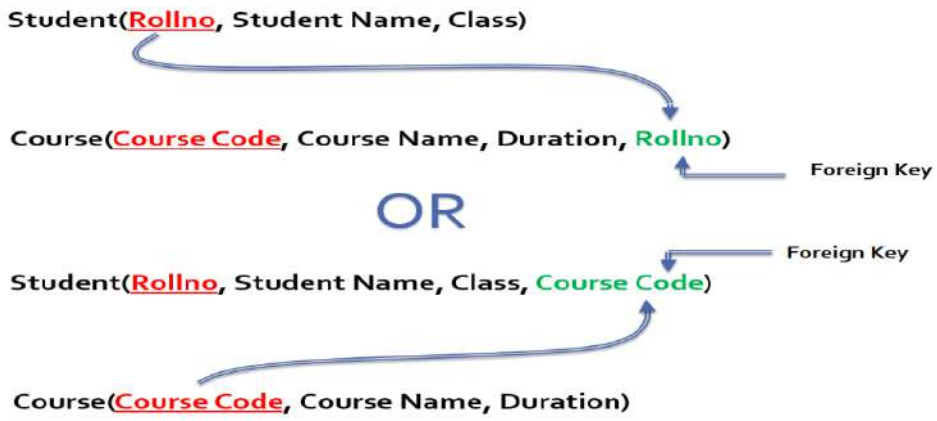
7. 1:1 (one to one) Relationship:

Consider same relationship set enroll exist between entity sets student and course ,which means one student can enroll in only one courses

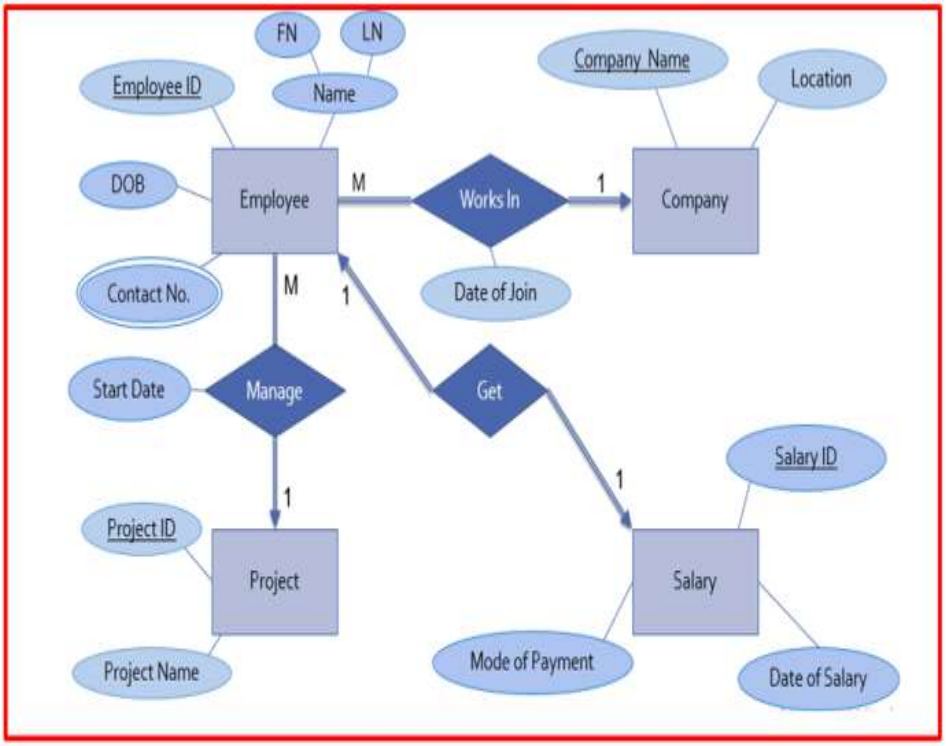


To convert this Relationship set into relational schema,

1. Separate relation is created for all participating entity sets.
2. Primary Key of Relation Student can be act as foreign key for relation Course OR Primary Key of Relation Course act as foreign key for relation Student.



EXAMPLE:



ER Diagram

Employee(Employee ID , DOB , FN , LN , Salary ID)
EmployeeContact(Employee ID, Contact No)
Company(Company Name , Location, Employee ID, Date of Join)
Project(Project ID , Project Name , Employee ID, Start Date)
Salary(Salary ID , Mode of Payment, Date of Salary, Employee ID)

Relational Model

Problem Definition

In this example problem, you will create a database for an organization with many departments. Each department has employees and employees have dependents. To create a database for the company, read the description and identify all the entities from the description of the company.

- The company organized into departments and departments have employees working in it.
- Attributes of Department are *dno, dname*. Attributes of Employee include *eno, name, dob,gender, doj, designation, basic_pay, panno, skills*. *Skills* are multi-valued attribute.
- The Department has a manager managing it. There are also supervisors in Department who supervises a set of employees.
- Each Department enrolls a number of projects. Attributes of Project are *pcode, pname.* A project is enrolled by a department. An employee can work on any number of projects on a given day. The Date of employee work in-time and out-time has to keep track.
- The Company also maintains details of the dependents of each employees. Attributes of dependent include *dname, dob, gender* and *relationship* with the employee.

Model an entity-relationship diagram for the above scenario.

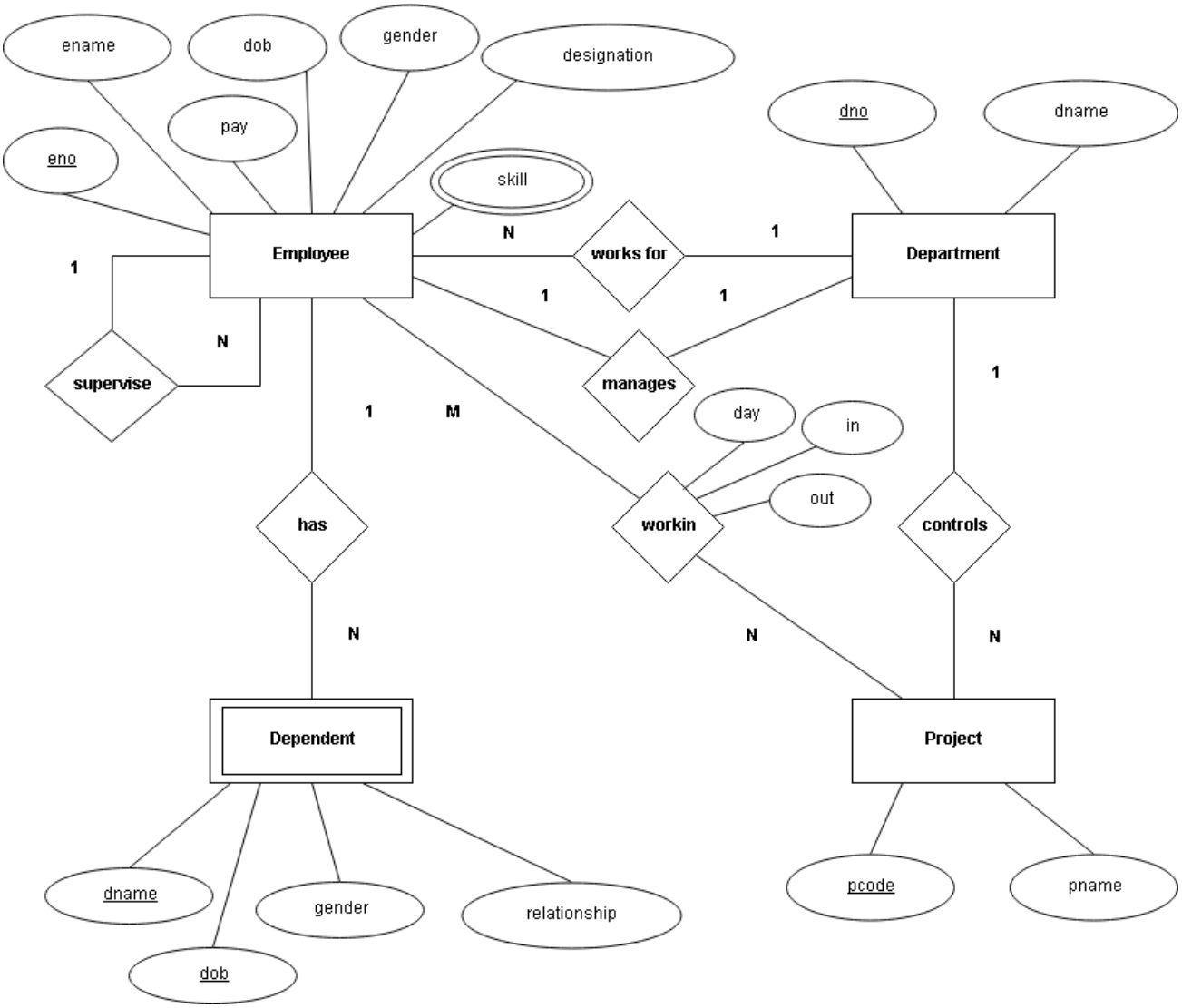
Solution – Converting ER model into Relational model

Developing an ERD

The process of database design is an iterative rather than a linear or sequential process. The verb iterate means "to do again or repeatedly." An iterative process is, thus, one based on repetition of processes and procedures. Building an ERD usually involves the following activities:

1. **Create a detailed narrative of the organization's description of operations.**
2. **Identify the business rules(Constraints) based on the description of operations.**
3. **Identify the main entities and relationships from the business rules.**
4. **Develop the initial ERD.**
5. **Identify the attributes and primary keys that adequately describe the entities.**
6. **Revise and review the ERD.**

From the real world description of the organization, we were able to identify the following entities. These entities will become the basis for an entity-relationship diagram or model.



ENTITY-RELATIONSHIP DIAGRAM OR MODEL.

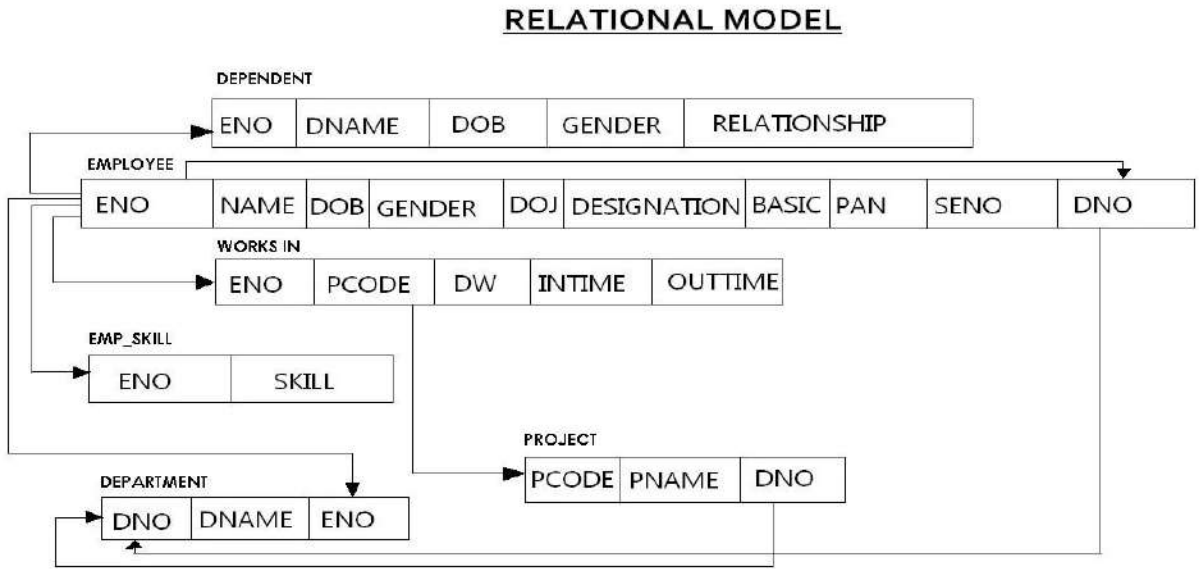
Convert to Relational Model

The next step in the database design is to convert the ER Model intothe Relational Model. In the Relational Model, we will define the schema for relations and their relationships. The attributes from the entity-relationship diagram will become fields for a relationship and one of them is a primary field or primary key. It is usually underlined in the entity-relationship diagram.

An entity in a relational model is a relation. For example, the entity Dependent is a relation in the relational model with all the attributes as fields – eno, dname, dob, gender, and relationship.

In general conversion of E-R diagram into a relational model involves the following:

1. Mapping of an entity set into relation (tables) of the database.
2. The attributes of a table include the attributes of an entity
3. The key attribute of an entity becomes the primary key of the relation



Relational Model of Database

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:
Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

Note: view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 4;
```

Just like table query, we can query the view to view the data.

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

View_name: Name of the View which we want to delete.

Example:

If we want to delete the View **MarksView**, we can do this as:

EX: DROP VIEW MarksView;

Executed Queries on Views:

```
mysql> create table student_detail(std_id int,name varchar(20),address varchar(20));
```

Query OK, 0 rows affected (0.33 sec)

mysql> insert into student_detail(std_id,name,address) values
(1,'stephan','delhi'),(2,'kathrin','noida'),(3,'david','ghaziabad'),(4,'alina','gurugram');

Query OK, 4 rows affected (0.11 sec)

Records: 4 Duplicates: 0 Warnings: 0

mysql> create table student_marks(std_id int,name varchar(20),marks int,age int);

Query OK, 0 rows affected (0.11 sec)

mysql> insert into student_marks(std_id,name,marks,age) values
(1,'stephan',97,19),(2,'kathrin',86,21),(3,'david',74,18),(4,'alina',90,19),(5,’john’,96,18);

Query OK, 5 rows affected (0.02 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> create view detailsview as select name,address from student_detail where std_id<4;

Query OK, 0 rows affected (0.17 sec)

mysql> select * from detailsview;

+---------+-----------+

| name | address |

+---------+-----------+

| stephan | delhi |

| kathrin | noida |

| david | ghaziabad |

+---------+-----------+

3 rows in set (0.08 sec)

mysql> create view marksview as select student_detail.name, student_detail.address, student_marks.marks from student_detail, student_marks where student_detail.name=student_marks.name;

Query OK, 0 rows affected (0.09 sec)

mysql> select * from marksview;

+---------+-----------+-------+

| name | address | marks |

+---------+-----------+-------+

| stephan | delhi | 97 |

| kathrin | noida | 86 |

| david | ghaziabad | 74 |

| alina | gurugram | 90 |

+---------+-----------+-------+

4 rows in set (0.00 sec)

```
mysql> drop view marksview;

Query OK, 0 rows affected (0.00 sec)

mysql> create view marksview1 as select student_detail.name,
student_marks.marks,student_marks.age from student_detail, student_marks where
student_detail.std_id=student_marks.std_id;

Query OK, 0 rows affected (0.20 sec)


mysql> select * from marksview1;

+------+-------+------+

| name | marks | age  |

+------+-------+------+

| xyz  |    98 |   18 |

| pqr  |    95 |   19 |

| abc  |    68 |   17 |

| uvw  |    78 |   18 |

+------+-------+------+

4 rows in set (0.09 sec)
```

Update SQL View:

The SQL view created can also be modified. We can do the following operations with the SQL VIEW.

But, **all views are not updatable.** A SQL view can be updated if the following conditions are satisfied.

1. The view is defined based on only one table.
2. The view should not have any field which is made of an **aggregate** function.
3. The view must not have GROUP BY, HAVING or DISTINCT clause in its definition.
4. The view should not be created using any nested query.

5. The selected output fields of the view must not use constants, string or value expressions.
6. If you want to update a view based on another view then that view should be updatable.

Updating a SQL View

We can use the **CREATE OR REPLACE VIEW** statement to **modify** the SQL view.
Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,column2,..
FROM table_name
WHERE condition;
```

Example: If we want to update the view marksview1 and remove the attribute "*Age"* from in the view then the query would be:

Query

```
mysql> create or replace view marksview1 as select student_detail.name,
student_marks.marks from student_detail, student_marks where
student_detail.std_id=student_marks.std_id;

Query OK, 0 rows affected (0.03 sec)
```

The above **CREATE OR REPLACE VIEW statement** would create a virtual table based on the result of the SELECT statement. Now, you can query the SQL VIEW as follows to see the output:

Output

```
mysql> select * from marksview1;

+---------+-------+

| name    | marks |

+---------+-------+

| stephan |    97 |

| kathrin |    86 |

| david   |    74 |

| alina   |    90 |

+---------+-------+

4 rows in set (0.03 sec)
```