UNIT-II

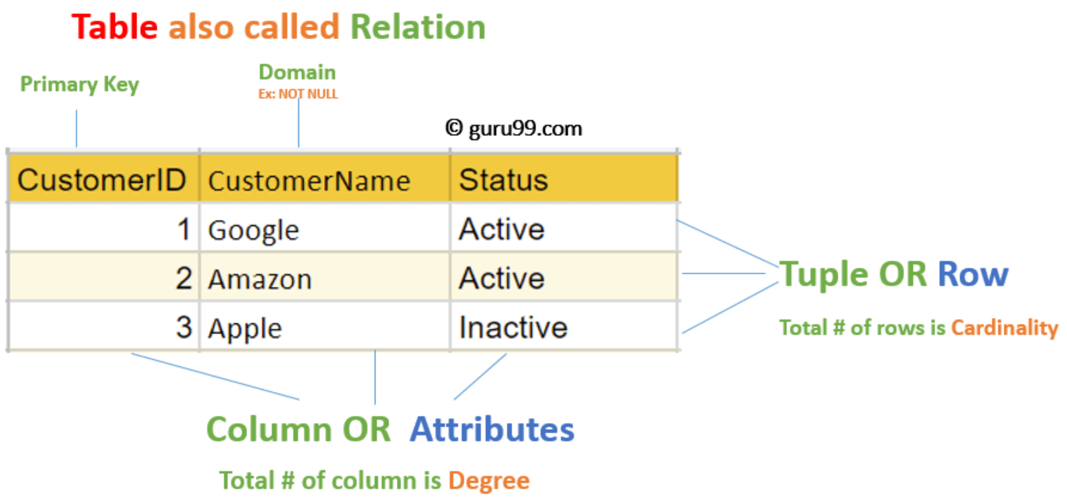
Relational Model (RM)

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:** 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



Creating and Modifying Relations using SQL

The first pair are CREATE TABLE and DROP TABLE. While they are written as two words, they are actually single commands. The first one creates a new table; its arguments are the names and types of the table's columns. For example, the following statements create the four tables in our survey database:

```
CREATE TABLE Person(id text, personal text, family text);
CREATE TABLE Site(name text, lat real, long real);
CREATE TABLE Visited(id integer, site text, dated text);
CREATE TABLE Survey(taken integer, person text, quant text, reading real);
```

We can get rid of one of our tables using:

```
DROP TABLE Survey;
```

Be very careful when doing this: if you drop the wrong table, hope that the person maintaining the database has a backup, but it's better not to have to rely on it.

Different database systems support different data types for table columns, but most provide the following:

data type	**Use**
INTEGER	a signed integer
REAL	a floating point number

TEXT	a character string
BLOB	a “binary large object”, such as an image

Data Definition Language

SQL uses the following set of commands to define database schema –

CREATE

Creates new databases, tables and views from RDBMS.

For example –

Create database tutorialspoint;
Create table article;
Create view for_students;

DROP

Drops commands, views, tables, and databases from RDBMS.

For example–

Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;

ALTER

Modifies database schema.

Alter object_type object_name parameters;

For example–

Alter table article add subject varchar;

This command adds an attribute in the relation **article** with the name **subject** of string type.

Data Manipulation Language

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section –

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

INSERT INTO/VALUES

This command is used for inserting values into the rows of a table (relation).

Syntax–

INSERT INTO table (column1 [, column2, column3 ...]) VALUES (value1 [, value2, value3 ...])

Or

INSERT INTO table VALUES (value1, [value2, ...])

For example –

INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous", "computers");

UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation).

Syntax –

UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]

For example –

UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";

DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation).

Syntax –

DELETE FROM table_name [WHERE condition];

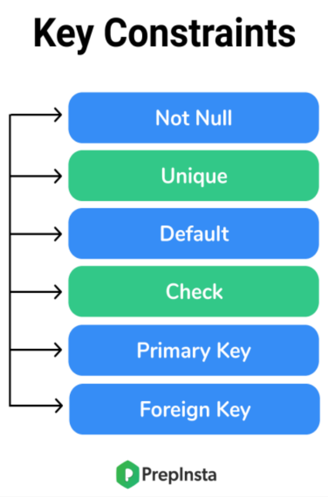
For example –

```
DELETE FROM tutorialspoints
   WHERE Author="unknown";
```

INTEGRITY CONSTRAINTS OVER RELATION

- Constraints or nothing but the rules that are to be followed while entering data into columns of the database table
- Constraints ensure that data entered by the user into columns must be within the criteria specified by the condition
- For example, if you want to maintain only unique IDs in the employee table or if you want to enter only age under 18 in the student table etc
- We have ~~5~~ types of key constraints in DBMS
  6

  - NOT NULL: ensures that the specified ***column doesn't contain a NULL value.***
  - UNIQUE : ***provides a unique/distinct values*** to specified columns.
  - DEFAULT: ***provides a default value to a column*** if none is specified.
  - CHECK :***checks for the predefined conditions before inserting*** the data inside the table.
  - PRIMARY KEY: it ***uniquely identifies a row*** in a table.
  - FOREIGN KEY: ensures ***referential integrity*** of the relationship



Not Null

- 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***

Example

```
CREATE TABLE STUDENT
(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

- In the above example, we have applied not null on three columns ID, name and age which means ***whenever a record is entered using insert statement all three columns should contain a value other than null***
- We have two other columns address and salary, ***where not null is not applied*** which means that ***you can leave the row as empty or use null value while inserting the record into the table***

Unique

- Sometimes we need to maintain only unique data in the column of a database table, this is possible by using a unique constraint
- Unique constraint ensures that all values in a column are unique

Example

```
CREATE TABLE Persons (
   ID int UNIQUE,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
);
```

In the above example, ***as we have used unique constraint on ID column we are not supposed to enter the data that is already present***, simply no two ID values are same

DEFAULT

- Default clause in SQL is used to add default data to the columns
- When a column is specified as default with some value then all the rows will use the same value i.e each and every time while entering the data we need not enter that value
- But ***default column value can be customized*** i.e it can be overridden when inserting a data for that row based on the requirement.

Example for DEFAULT clause

The following SQL sets a DEFAULT value for the "city" column when the "emp" table is created:

```
CREATE TABLE emp (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'hyderabad'
);
```

- As a result, whenever you insert a new row each time you need not enter a value for this default column that is ***entering a column value for a default column is optional and if you don't enter the same value is considered that is used in the default clause***

Check

- Suppose in real-time if you want to give access to an application only if the age entered by the user is greater than 18 this is done at the back-end by using a check constraint
- Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

Example for check constraint

```
CREATE TABLE STUDENT (
   ID int ,
   Name varchar(255) ,
   Age int,
   CHECK (Age>=18)
);
```

- As we have used a ***check constraint as (Age>=18)*** which means ***values entered by the user for this age column while inserting the data must be less than or equal to 18*** otherwise an error is shown
- Simply, the only possible values that the ***age column will accept is [0 -17]***

Primary Key

A primary key is a constraint in a table which uniquely identifies each row record in a database table by enabling one or more the column in the table as primary key.

Creating a primary key

A particular column is made as a primary key column by using the primary key keyword followed with the column name

```
CREATE TABLE EMP (
 ID   INT
 NAME VARCHAR (20)
AGE  INT
 COURSE VARCHAR(10)
PRIMARY KEY (ID)
);
```

- Here we have used the primary key on ID column then ID column must contain unique values i.e ***one ID cannot be used for another student***.
- If you try to ***enter duplicate value while inserting in the row you are displayed with an error***
- Hence ***primary key will restrict you to maintain unique values and not null values in that particular column***

Foreign Key

- The foreign key constraint is a column or list of columns which points to the primary key column of another table
- The main purpose of the foreign key is only those values are allowed in the present table that will match to the primary key column of another table.

Example to create a foreign key

Reference Table

```
CREATE TABLE CUSTOMERS1(
  ID   INT ,
 NAME VARCHAR (20) ,
 COURSE VARCHAR(10) ,
  PRIMARY KEY (ID)
);
```

Child Table

```
CREATE TABLE CUSTOMERS2(
  ID   INT ,
  MARKS INT,
  REFERENCES CUSTOMERS1(ID)
);
```

Referential Integrity constraint

A referential integrity constraint is also known as **foreign key constraint**. A foreign key is a key whose values are derived from the Primary key of another table.

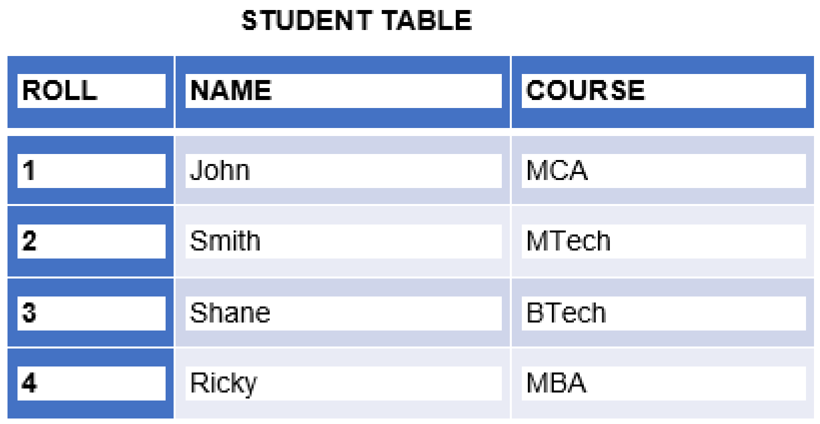
The table from which the values are derived is known as **Master or Referenced** Table and the Table in which values are inserted accordingly is known as **Child or Referencing** Table, In other words, we can say that the table containing the **foreign key** is called the **child table**, and the table containing the **Primary key/candidate key** is called the **referenced or parent table.** When we talk about the database relational model, the candidate key can be defined as a set of attribute which can have zero or more attributes.

The syntax of the Master Table or Referenced table is:

1.

```
CREATE TABLE Student (Roll int PRIMARY KEY, Name varchar(25) , Course varchar(10) );
```

Here column Roll is acting as **Primary Key,** which will help in deriving the value of foreign key in the child table.



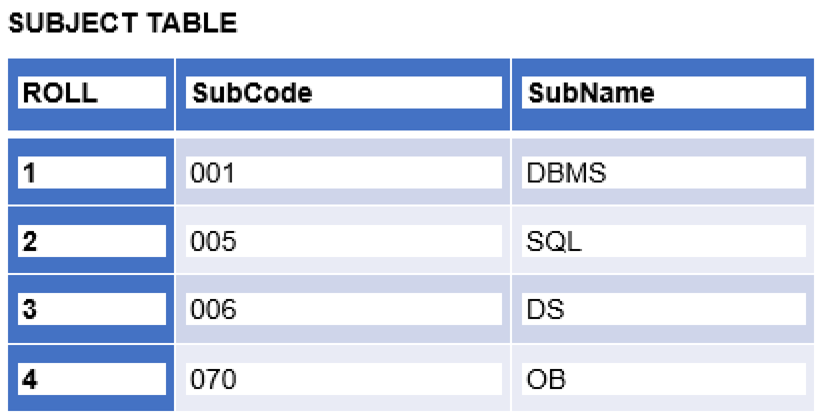
STUDENT TABLE

ROLL	NAME	COURSE
1	John	MCA
2	Smith	MTech
3	Shane	BTech
4	Ricky	MBA

The syntax of Child Table or Referencing table is:

1.

```
CREATE TABLE Subject (Roll int references Student, SubCode int, SubName varchar(10) );
```



SUBJECT TABLE

ROLL	SubCode	SubName
1	001	DBMS
2	005	SQL
3	006	DS
4	070	OB

In the above table, column Roll is acting as **Foreign Key,** whose values are derived using the Roll value of Primary key from Master table.

Foreign Key Constraint OR Referential Integrity constraint.

There are two referential integrity constraint:

Insert Constraint: Value cannot be inserted in CHILD Table if the value is not lying in MASTER Table

Delete Constraint: Value cannot be deleted from MASTER Table if the value is lying in CHILD Table

Suppose you wanted to insert Roll = 05 with other values of columns in SUBJECT Table, then you will immediately see an error "**Foreign key Constraint Violated**" i.e. on running an insertion command as:

Insert into SUBJECT values(5, 786, OS); will not be entertained by SQL due to Insertion Constraint (As you cannot insert value in a child table if the value is not lying in the master table, since Roll = 5 is not present in the master table, hence it will not be allowed to enter Roll = 5 in child table)

Similarly, if you want to delete Roll = 4 from STUDENT Table, then you will immediately see an error "**Foreign key Constraint Violated**" i.e. on running a deletion command as:

Delete from STUDENT where Roll = 4; will not be entertained by SQL due to Deletion Constraint. (As you cannot delete the value from the master table if the value is lying in the child table, since Roll = 5 is present in the child table, hence it will not be allowed to delete Roll = 5 from the master table, lets if somehow we managed to delete Roll = 5, then Roll = 5 will be available in child table which will ultimately violate insertion constraint.)

ON DELETE CASCADE.

As per deletion constraint: Value cannot be deleted from the MASTER Table if the value is lying in CHILD Table. The next question comes can we delete the value from the master table if the value is lying in the child table without violating the deletion constraint? i.e. The moment we delete the value from the master table the value corresponding to it should also get deleted from the child table.

The answer to the above question is YES, we can delete the value from the master table if the value is lying in the child table without violating the deletion constraint, we have to do slight modification while creating the child table, i.e. by adding **on delete cascade**.

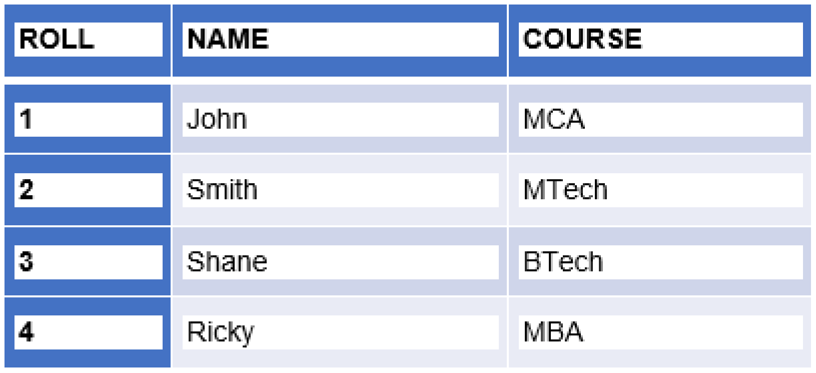
TABLE SYNTAX

1.

CREATE TABLE Subject (Roll **int** references Student on delete cascade, SubCode **int**, SubNa
me varchar(10));

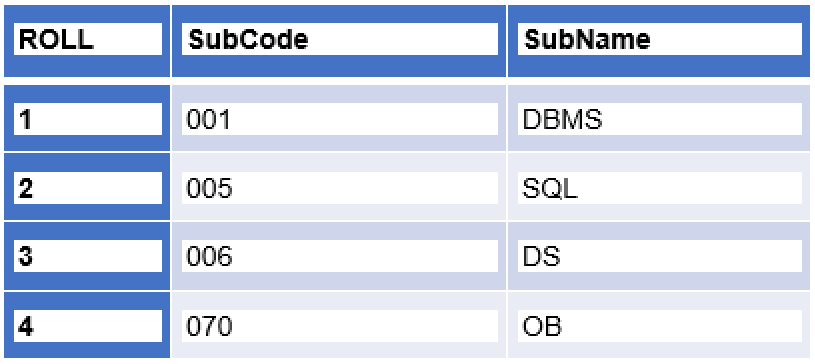
In the above syntax, just after references keyword(used for creating foreign key), we have added on delete cascade, by adding such now, we can delete the value from the master table if the value is lying in the child table without violating deletion constraint. Now if you wanted to delete Roll = 5 from the master table even though Roll = 5 is lying in the child table, it is possible because the moment you give the command to delete Roll = 5 from the master table, the row having Roll = 5 from child table will also get deleted.

STUDENT TABLE



ROLL	NAME	COURSE
1	John	MCA
2	Smith	MTech
3	Shane	BTech
4	Ricky	MBA

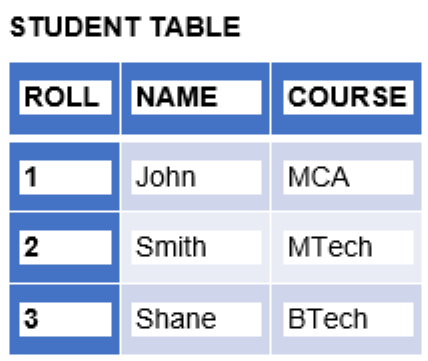
SUBJECT TABLE



ROLL	SubCode	SubName
1	001	DBMS
2	005	SQL
3	006	DS
4	070	OB

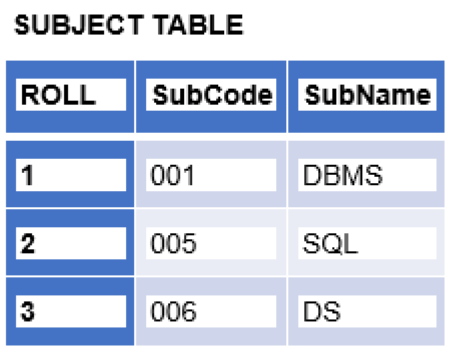
The above two tables STUDENT and SUBJECT having four values each are shown, now suppose you are looking to delete Roll = 4 from STUDENT(Master) Table by writing a SQL command: **delete from STUDENT where Roll = 4;**

The moment SQL execute the above command the row having Roll = 4 from SUBJECT(Child) Table will also get deleted, The resultant **STUDENT and SUBJECT** table will look like:



STUDENT TABLE

ROLL	NAME	COURSE
1	John	MCA
2	Smith	MTech
3	Shane	BTech



SUBJECT TABLE

ROLL	SubCode	SubName
1	001	DBMS
2	005	SQL
3	006	DS

From the above two tables STUDENT and SUBJECT, you can see that in both the table Roll = 4 gets deleted at one go without violating deletion constraint.

Sometimes a very important question is asked in interviews that: Can Foreign Key have NULL values?

The answer to the above question is YES, it may have NULL values, whereas the Primary key cannot be NULL at any cost. To understand the above question practically let's understand below the concept of delete null.

ON DELETE NULL.

As per deletion constraint: Value cannot be deleted from the MASTER Table if the value is lying in CHILD Table. The next question comes can we delete the value from the master table if the value is lying in the child table without violating the deletion constraint? i.e. The moment we delete the value from the master table the value corresponding to it should also get deleted from the child table or can be replaced with the NULL value.

The answer to the above question is YES, we can delete the value from the master table if the value is lying in child table without violating deletion constraint by inserting NULL in the foreign key, we have to do slight modification while creating child table, i.e. by adding **on delete null**.

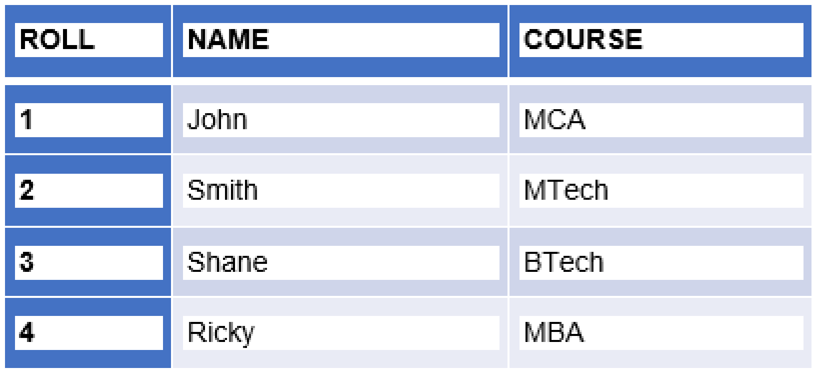
TABLE SYNTAX:

1.

CREATE TABLE Subject (Roll **int** references Student on delete **null**, SubCode **int**, SubName varchar(10));

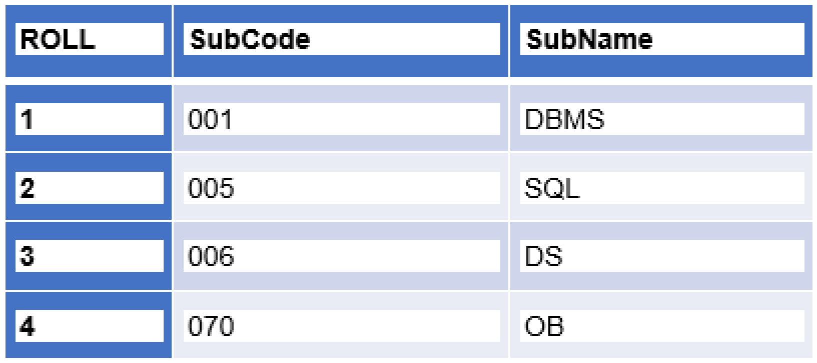
In the above syntax, just after references keyword(used for creating foreign key), we have added on delete null, by adding such now, we can delete the value from the master table if the value is lying in the child table without violating deletion constraint. Now if you wanted to delete Roll = 4 from the master table even though Roll =4 is lying in the child table, it is possible because the moment you give the command to delete Roll = 4 from the master table, the row having Roll = 4 from child table will get replaced by a NULL value.

STUDENT TABLE



ROLL	NAME	COURSE
1	John	MCA
2	Smith	MTech
3	Shane	BTech
4	Ricky	MBA

SUBJECT TABLE

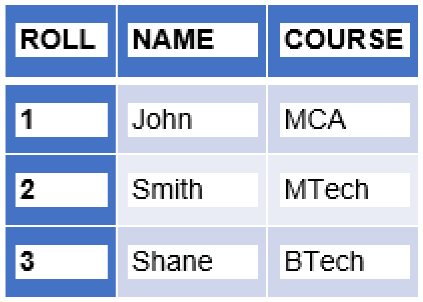


ROLL	SubCode	SubName
1	001	DBMS
2	005	SQL
3	006	DS
4	070	OB

The above two tables STUDENT and SUBJECT having four values each are shown, now suppose you are looking to delete Roll = 4 from STUDENT(Master) Table by writing a SQL command: **delete from STUDENT where Roll = 4;**

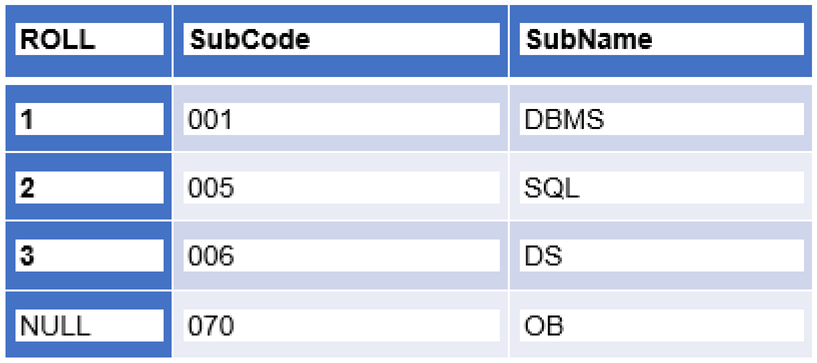
The moment SQL execute the above command the row having Roll = 4 from SUBJECT(Child) Table will get replaced by a NULL value, The resultant **STUDENT and SUBJECT** table will look like:

STUDENT TABLE



ROLL	NAME	COURSE
1	John	MCA
2	Smith	MTech
3	Shane	BTech

SUBJECT TABLE



ROLL	SubCode	SubName
1	001	DBMS
2	005	SQL
3	006	DS
NULL	070	OB

From the above two tables STUDENT and SUBJECT, you can see that in table STUDENT Roll = 4 get deleted while the value of Roll = 4 in the SUBJECT table is replaced by NULL. This proves that the Foreign key can have null values. If in the case in SUBJECT Table, column Roll is Primary Key along with Foreign Key then in that case we could not make a foreign key to have NULL values.

Logical database design

ER Model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram. We cannot import all the ER constraints into relational model, but an approximate schema can be generated.

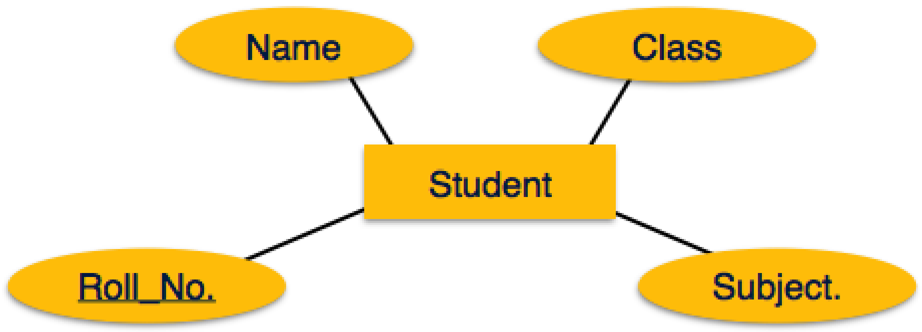
There are several processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual. We may focus here on the mapping diagram contents to relational basics.

ER diagrams mainly comprise of –

- Entity and its attributes
- Relationship, which is association among entities.

Mapping Entity

An entity is a real-world object with some attributes.

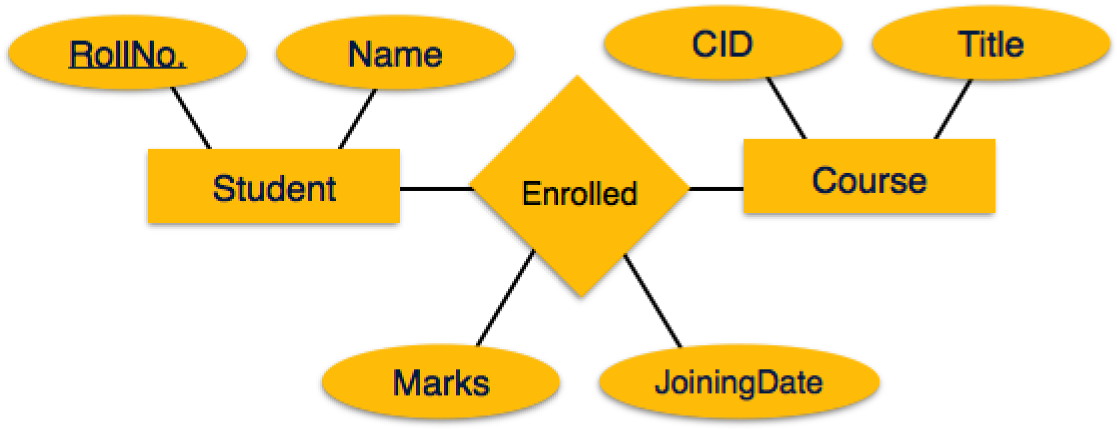


Mapping Process (Algorithm)

- Create table for each entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

Mapping Relationship

A relationship is an association among entities.

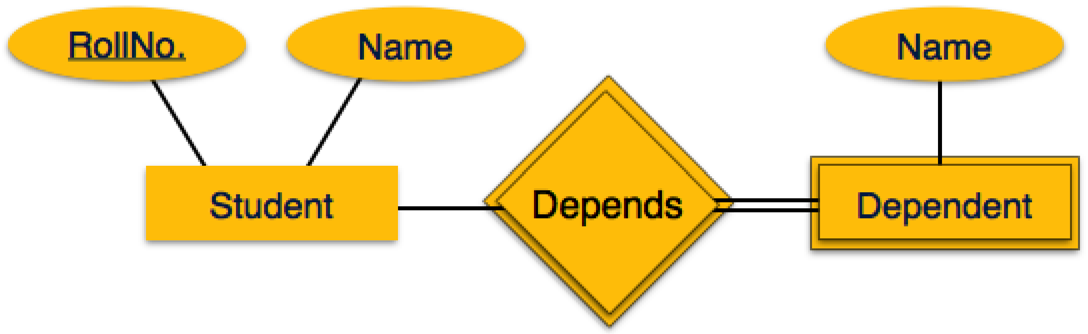


Mapping Process

- Create table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity set is one which does not have any primary key associated with it.

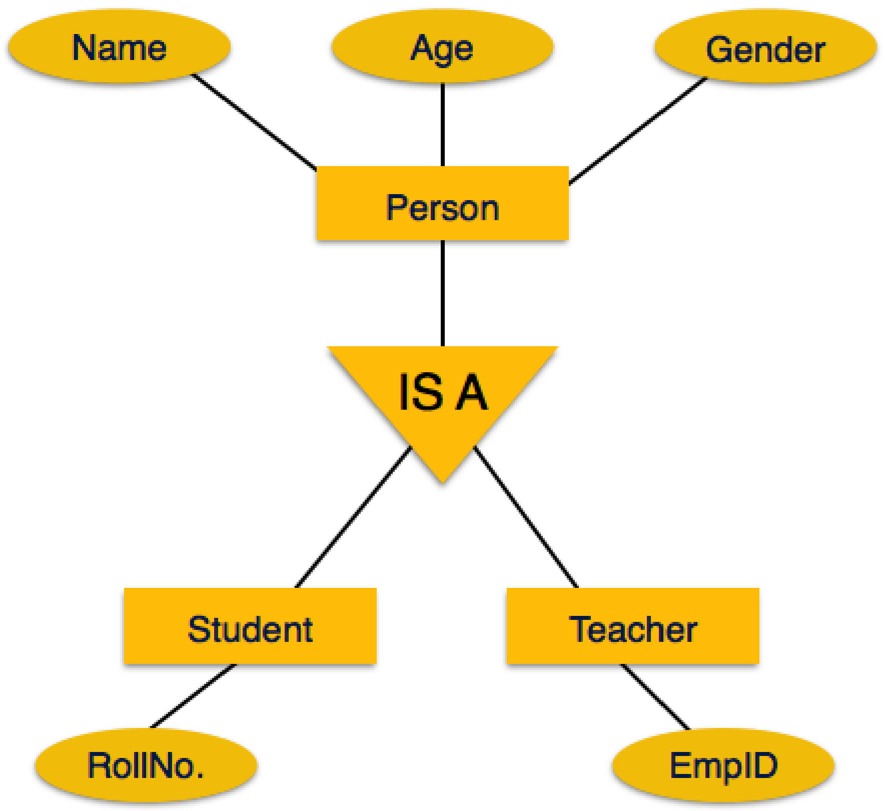


Mapping Process

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints.

Mapping Hierarchical Entities

ER specialization or generalization comes in the form of hierarchical entity sets.



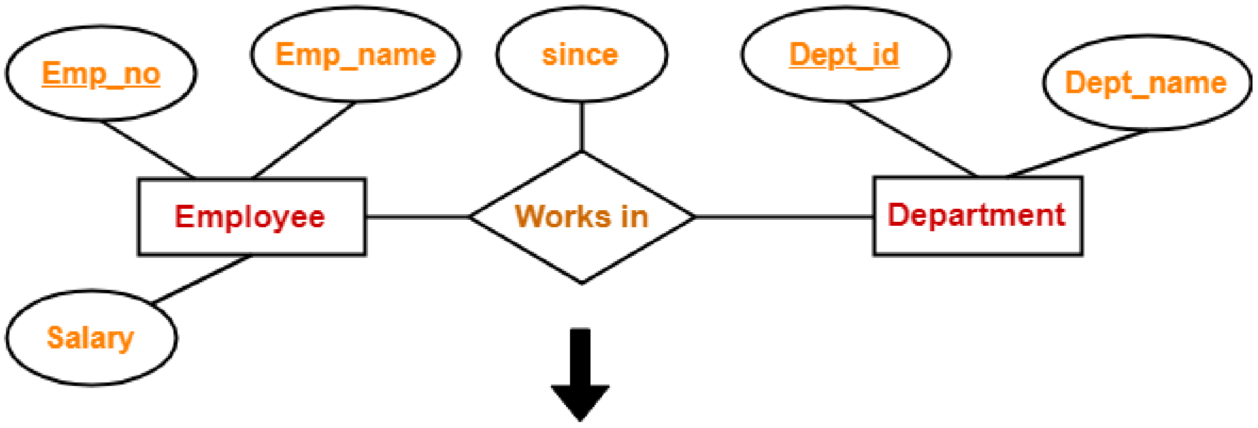
Mapping Process

- Create tables for all higher-level entities.
- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key of higher-level table and the primary key for lower-level table.
- Declare foreign key constraints.

Attributes of the table are-

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

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



Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

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

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

Rule-05: For Binary Relationships With Cardinality Ratios-

The following four cases are possible-

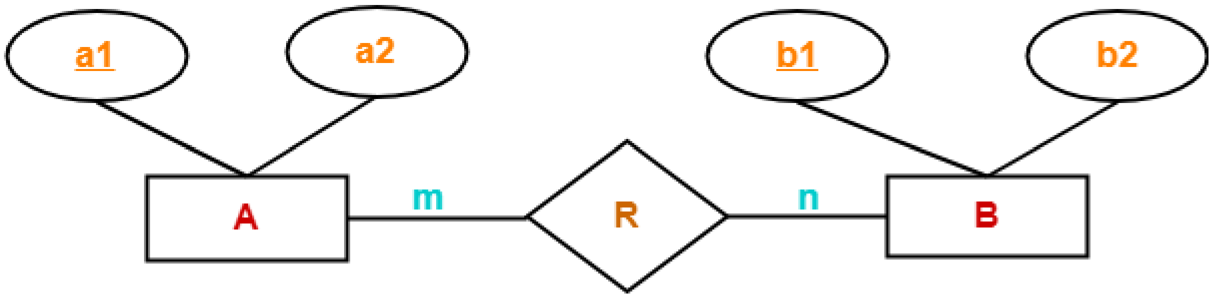
Case-01: Binary relationship with cardinality ratio m:n

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

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

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

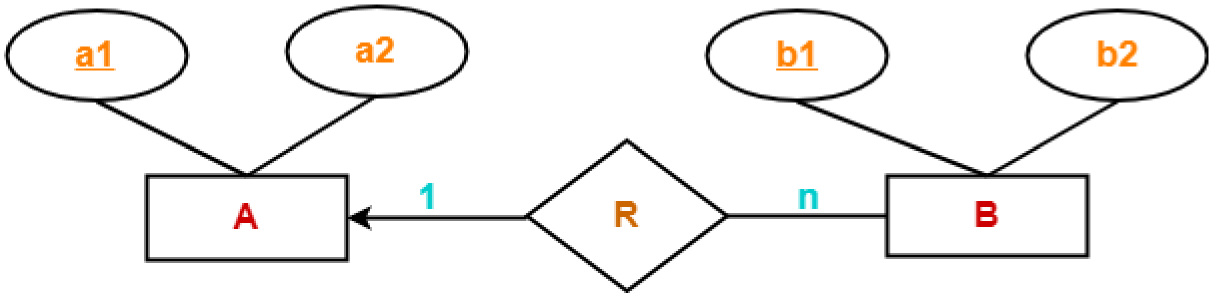
Case-01: For Binary Relationship With Cardinality Ratio m:n



Here, three tables will be required-

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

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

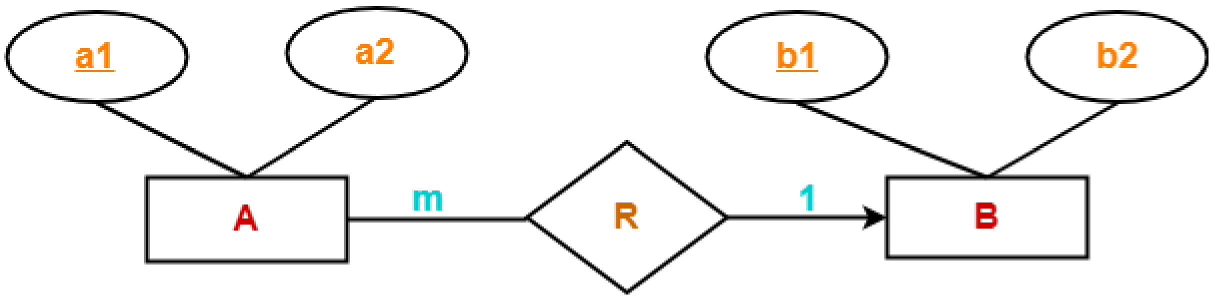


Here, two tables will be required-

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

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

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

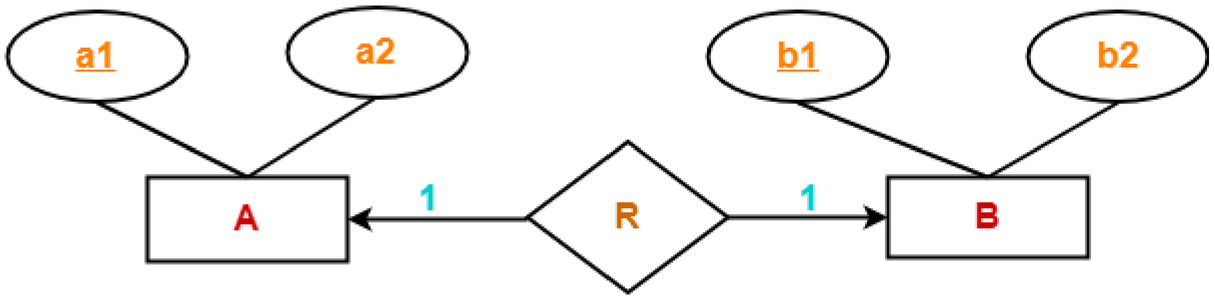


Here, two tables will be required-

1. AR (<u>a1</u> , a2 , b1)
2. B (<u>b1</u> , b2)

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

<u>Case-04: For Binary Relationship With Cardinality Ratio 1:1</u>



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

<u>Way-01:</u>

1. AR (<u>a1</u> , a2 , b1)
2. B (<u>b1</u> , b2)

<u>Way-02:</u>

1. A (<u>a1</u> , a2)
2. BR (a1 , <u>b1</u> , b2)

Thumb Rules to Remember

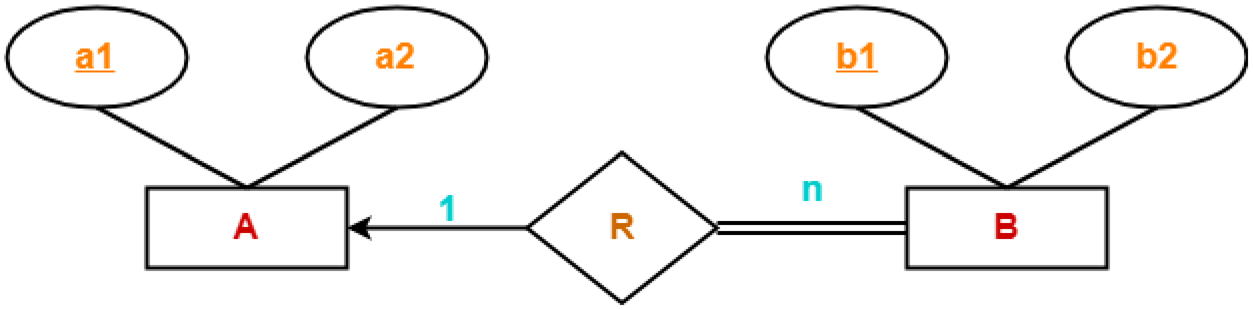
While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-

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

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

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

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



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

Then, two tables will be required-

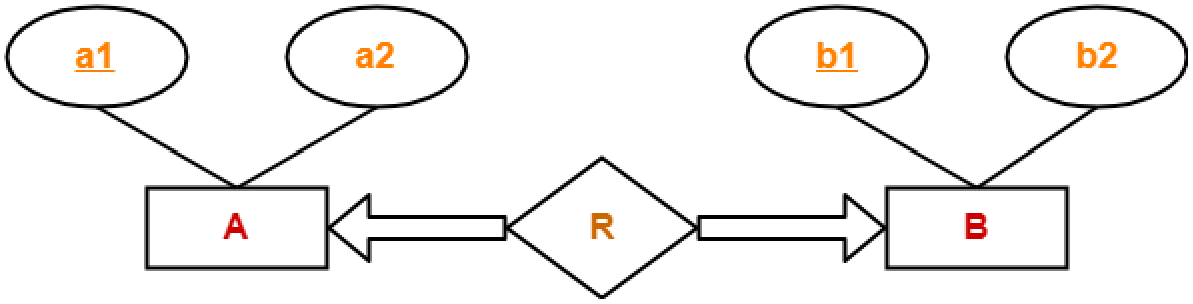
1. A (a1 , a2)

2. BR (a1 , <u>b1</u> , b2)

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

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

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

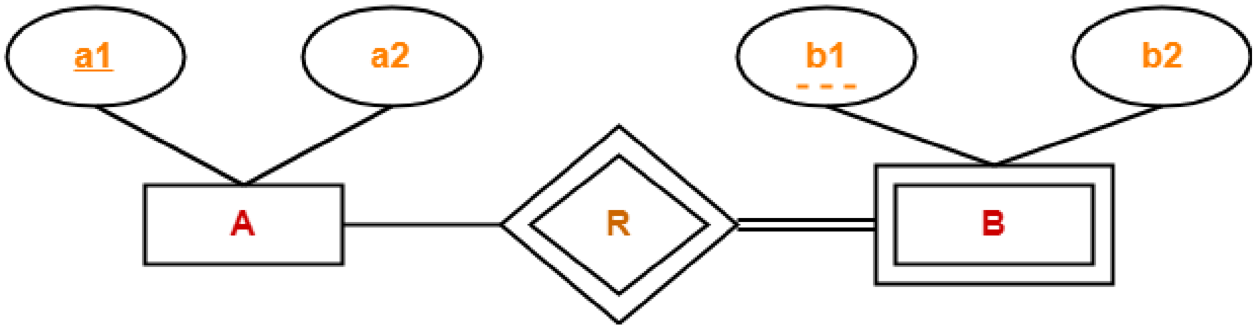


Here, Only one table is required.

- ARB (<u>a1</u> , a2 , <u>b1</u> , b2)

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

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



Here, two tables will be required-

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

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

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

Student_Marks

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:

```
1. CREATE VIEW view_name AS
2. SELECT column1, column2.....
3. FROM table_name
4. WHERE condition;
```

2. Creating View from a single table

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

Query:

```
1. CREATE VIEW DetailsView AS
2. SELECT NAME, ADDRESS
3. FROM Student_Details
4. WHERE STU_ID < 4;
```

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

```
1. 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:

1. CREATE VIEW MarksView AS
2. SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
3. FROM Student_Detail, Student_Mark
4. WHERE Student_Detail.NAME = Student_Marks.NAME;

To display data of View MarksView:

1. 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

1. DROP VIEW view_name;

Example:

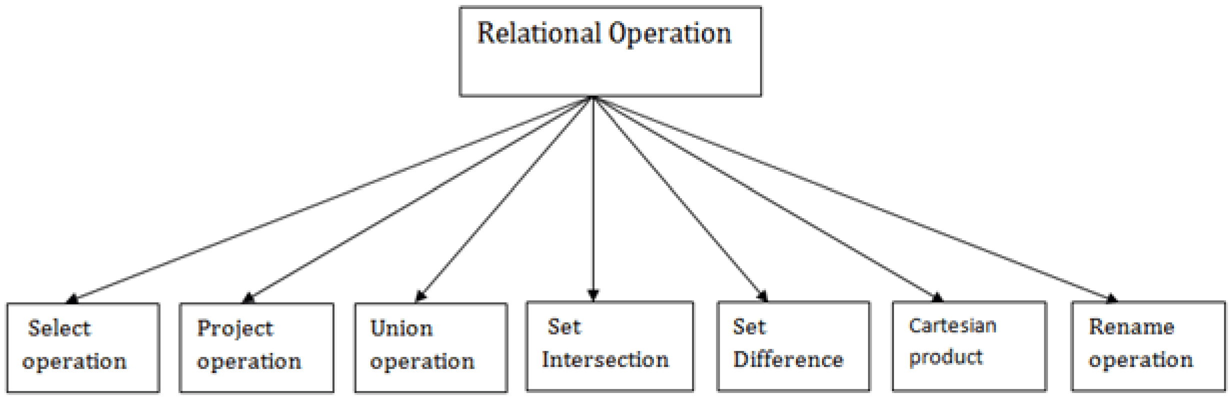
If we want to delete the View **MarksView**, we can do this as:

1. DROP VIEW MarksView;

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.

- It is denoted by sigma (σ).

1. Notation: σ p(r)

   Where:

   σ is used for selection prediction
   r is used for relation
   p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

   For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn

1. Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93

Smith	L-11
Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection   .

1. Notation: R   S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW)   ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME

A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

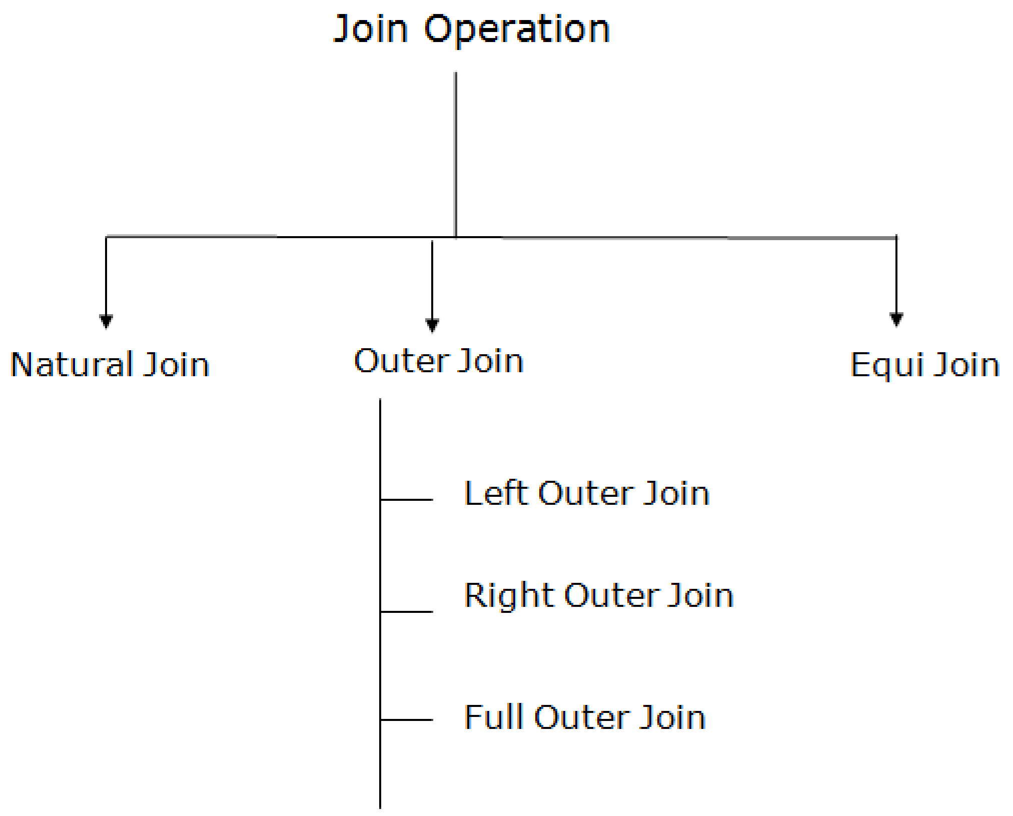
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

   Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata

Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. Left outer join

b. Right outer join
c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by .

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE FACT_WORKERS

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

2. **Output:**

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000

Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

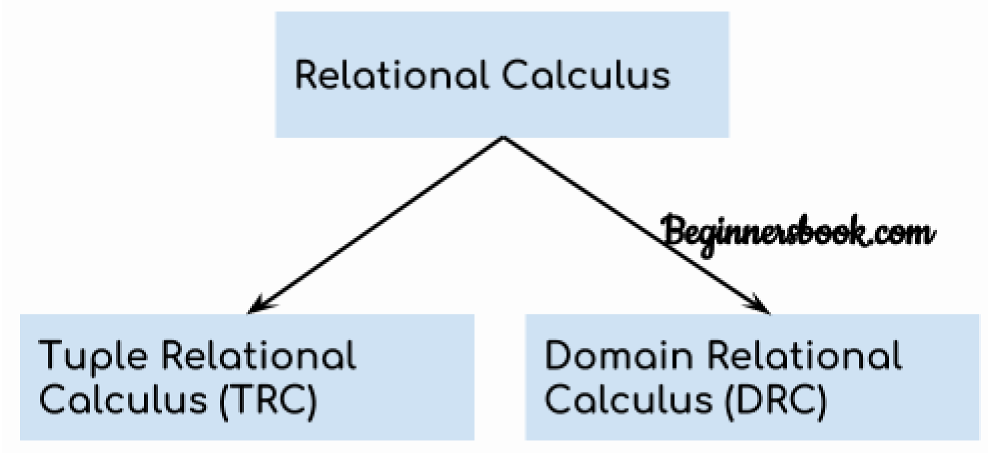
Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Relational calculus

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it.

Types of Relational Calculus


Relational Calculus
Beginnersbook.com
Tuple Relational Calculus (TRC)
Domain Relational Calculus (DRC)

Tupple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tupple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.
In Tupple Calculus, a query is expressed as

{t| P(t)}

where t = resulting tupples,
t is resulting tuples

P(t) = known as Predicate and these are the conditions that are used to fetch t

Thus, it generates set of all tupples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).
It also uses

quantifiers:
∃ t ∈ r (Q(t)) = "there exists" a tuple in t in relation r such that predicate Q(t) is true.
∀ t ∈ r (Q(t)) = Q(t) is true "for all" tuples in relation r.

Example:
Table-1: Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table-2: Branch

Branch name	Branch city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-3: Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table-4: Loan

Loan number	Branch name	Amount
L33	ABC	10000

L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table-5: Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table-6: Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Sumiti	1114

Queries-1: Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, t[amount] is known as tupple variable.

Queries-2: Find the loan number for each loan of an amount greater or equal to 10000.

{t| ∃ s ∈ loan(t[loan number] = s[loan number]

∧ s[amount]>=10000)}

Resulting relation:

Loan number
L33
L35
L98

Queries-3: Find the names of all customers who have a loan and an account at the bank.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name])

$\wedge\ \exists$ u $\in$ depositor(t[customer-name] = u[customer-name])}

Resulting relation:

Customer name
Saurabh
Mehak

Queries-4: Find the names of all customers having a loan at the "ABC" branch.

{t | $\exists$ s $\in$ borrower(t[customer-name] = s[customer-name]

$\wedge\ \exists$ u $\in$ loan(u[branch-name] = "ABC" $\wedge$ u[loan-number] = s[loan-number]))}

Resulting relation:

Customer name
Saurabh

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,
$\{ < x_1, x_2, x_3, ..., x_n > | P (x_1, x_2, x_3, ..., x_n) \}$
where, $< x_1, x_2, x_3, …, x_n >$ represents resulting domains variables and $P (x_1, x_2, x_3, …, x_n)$ represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:
Table-1: Customer

Customer name	Street	City
Debomit	Kadamtala	Alipurduar
Sayantan	Udaypur	Balurghat
Soumya	Nutanchati	Bankura
Ritu	Juhu	Mumbai

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

Customer name	Loan number
Ritu	L01
Debomit	L08
Soumya	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{\prec l, b, a \succ \mid \prec l, b, a \succ \in loan \land (a \geq 100)\}$

Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Query-2: Find the loan number for each loan of an amount greater or equal to 150.

$\{\prec l \succ \mid \exists\, b, a\, (\prec l, b, a \succ \in loan \land (a \geq 150)\}$

Resulting relation:

Loan number
L01

L03

Query-3: Find the names of all customers having a loan at the "Main" branch and find the loan amount .

$\{\prec c, a\succ \mid \exists\, l\, (\prec c, l\succ \in \text{borrower} \wedge \exists\, b\, (\prec l, b, a\succ \in \text{loan} \wedge (b = \text{"Main"})))\}$

Resulting relation:

Customer Name	Amount
Ritu	200
Debomit	60
Soumya	150

Note:
The domain variables those will be in resulting relation must appear before | within ≺ and ≻ and all the domain variables must appear in which order they are in original relation or table.