ER Model & Relational Model

ER Model (Entity Relationship Model)

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.
- ER Diagram graphically expresses the logical structure of database (schema), and it uses-
  - **Rectangle**: Represents Entity sets.
  - **Ellipses**: Attributes
  - **Diamonds**: Relationship Set
  - **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set
  - **Double Ellipses:** Multivalued Attributes
  - **Dashed Ellipses**: Derived Attributes
  - **Double Rectangles**: Weak Entity Sets
  - **Double Lines**: Total participation of an entity in a relationship set

ER Model (Entity Relationship Model)

- ER model stands for an Entity-Relationship model.
- It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database.
- It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

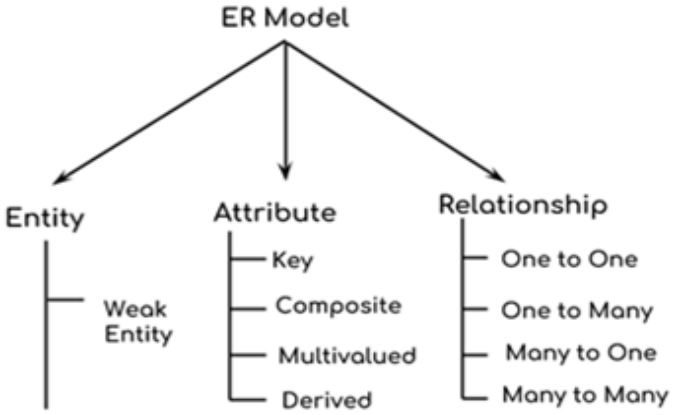
For example:

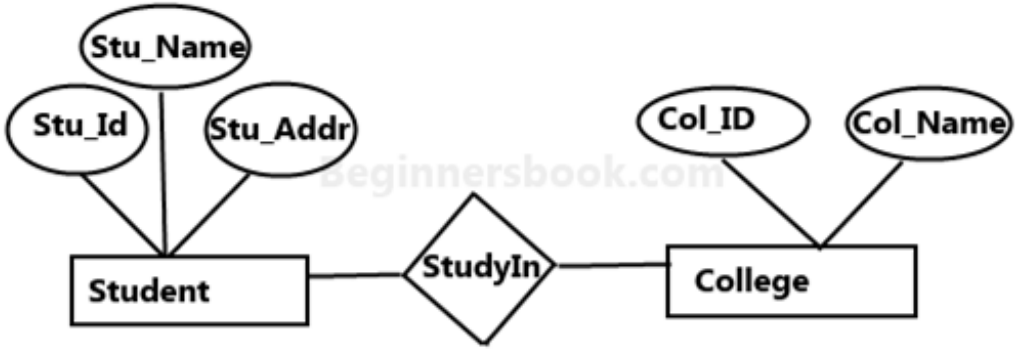
Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

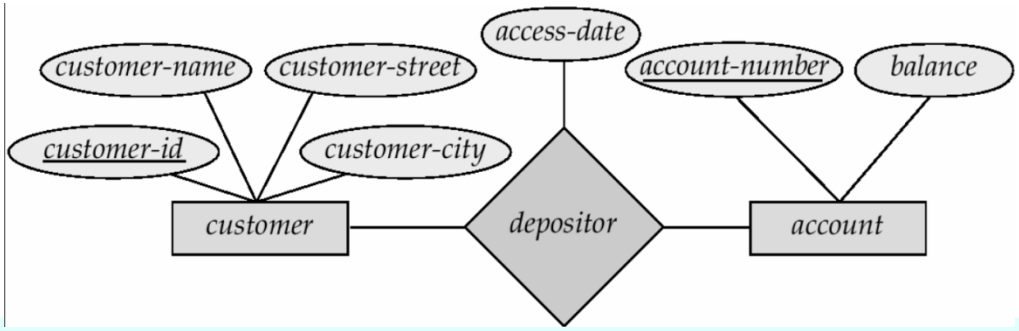
ER Model (Entity Relationship Model)

A simple ER Diagram:

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



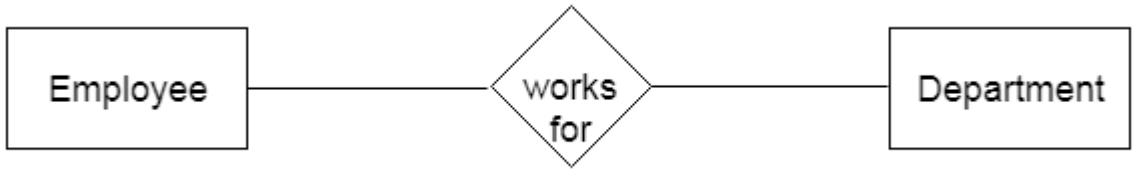




ER Model (Entity Relationship Model)

Entity:

- An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.
- An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set.
- Consider an organization as an example manager, product, employee, department etc. can be taken as an entity.



ER Model (Entity Relationship Model)

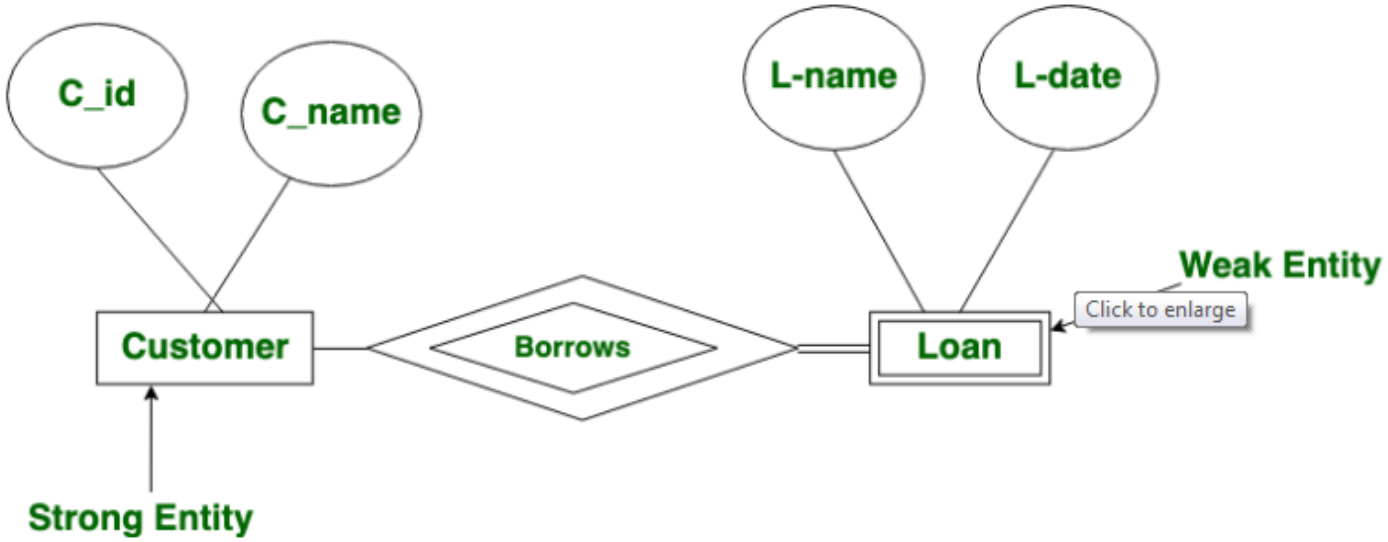
Strong Entity

A strong entity is not dependent of any other entity in the schema. A strong entity will always have a primary key.
Strong entities are represented by a single rectangle.
The relationship of two strong entities is represented by a single diamond.
Various strong entities, when combined together, create a strong entity set.

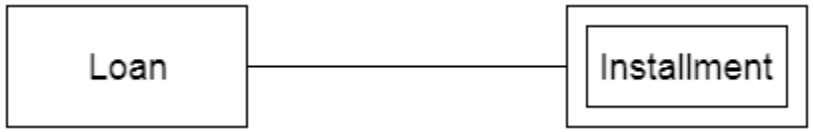
Weak Entity

A weak entity is dependent on a strong entity to ensure the its existence.
Unlike a strong entity, a weak entity does not have any primary key.
It instead has a partial discriminator key. A weak entity is represented by a double rectangle. The relation between one strong and one weak entity is represented by a double diamond.

ER Model (Entity Relationship Model)



A bank loan installment cannot be uniquely identified without knowing the loan to which the installment belongs, so loan installment is a weak entity.



ER Model (Entity Relationship Model)

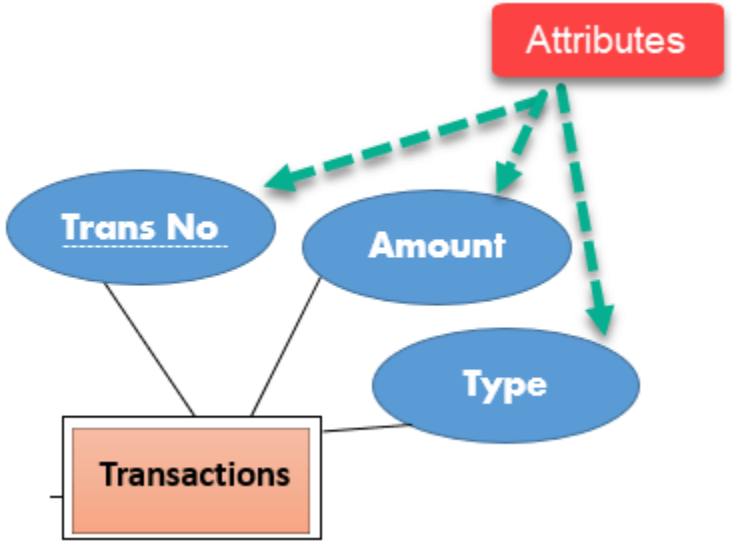
Attribute

Attributes are the **properties which define the entity type**. For example, RollNo, Name, DOB, Age, Address, MobileNo are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

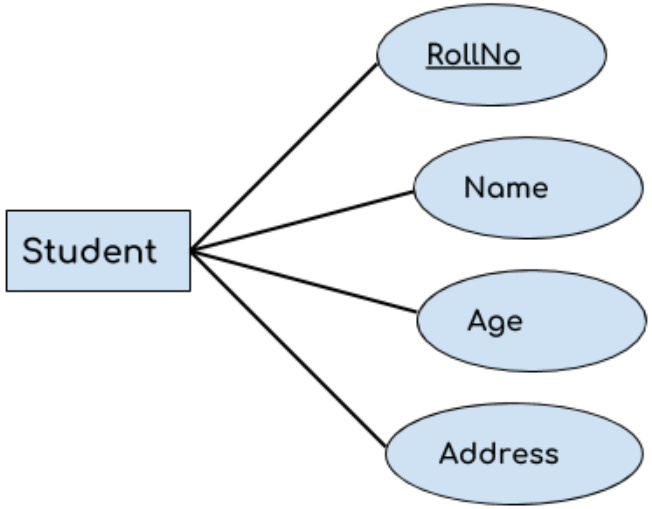
1. Simple attribute
2. Composite attribute
3. Derived attribute
4. Single-value attribute
5. Multi-value attribute



ER Model (Entity Relationship Model)

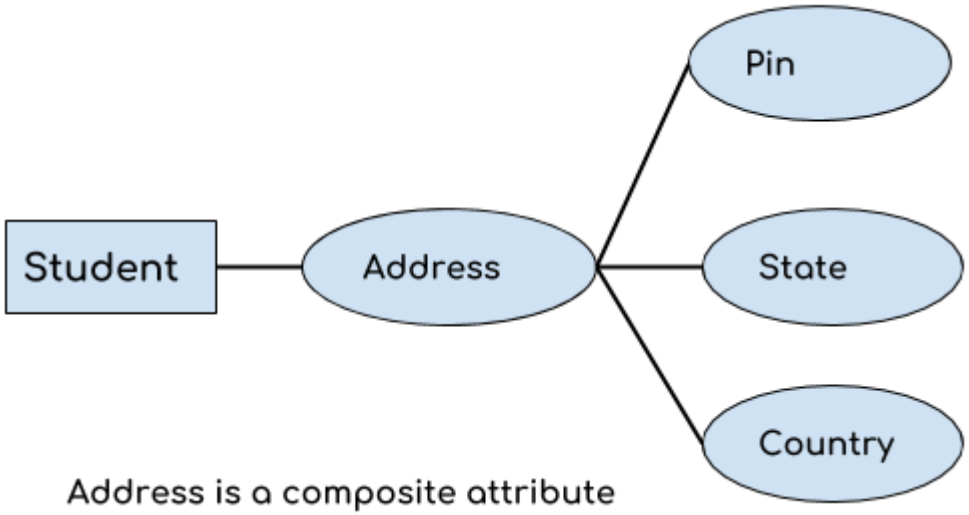
- **Key Attribute:**
  The attribute which **uniquely identifies each entity** in the entity set is called key attribute. For example, RollNo will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

- **Simple attribute:** Simple attributes are atomic values, which cannot be divided further. For example, In the diagram, RollNo & Age are the Simple Attributes.



ER Model (Entity Relationship Model)

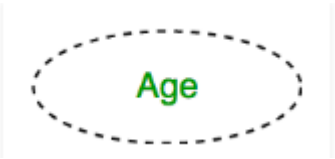
- **Composite Attribute:**
  An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country.
  In ER diagram, composite attribute is represented by an oval comprising of ovals. For example, a student's complete name may have first_name and last_name.



Address is a composite attribute

ER Model (Entity Relationship Model)

- **Derived attribute:** Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
  For example, Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

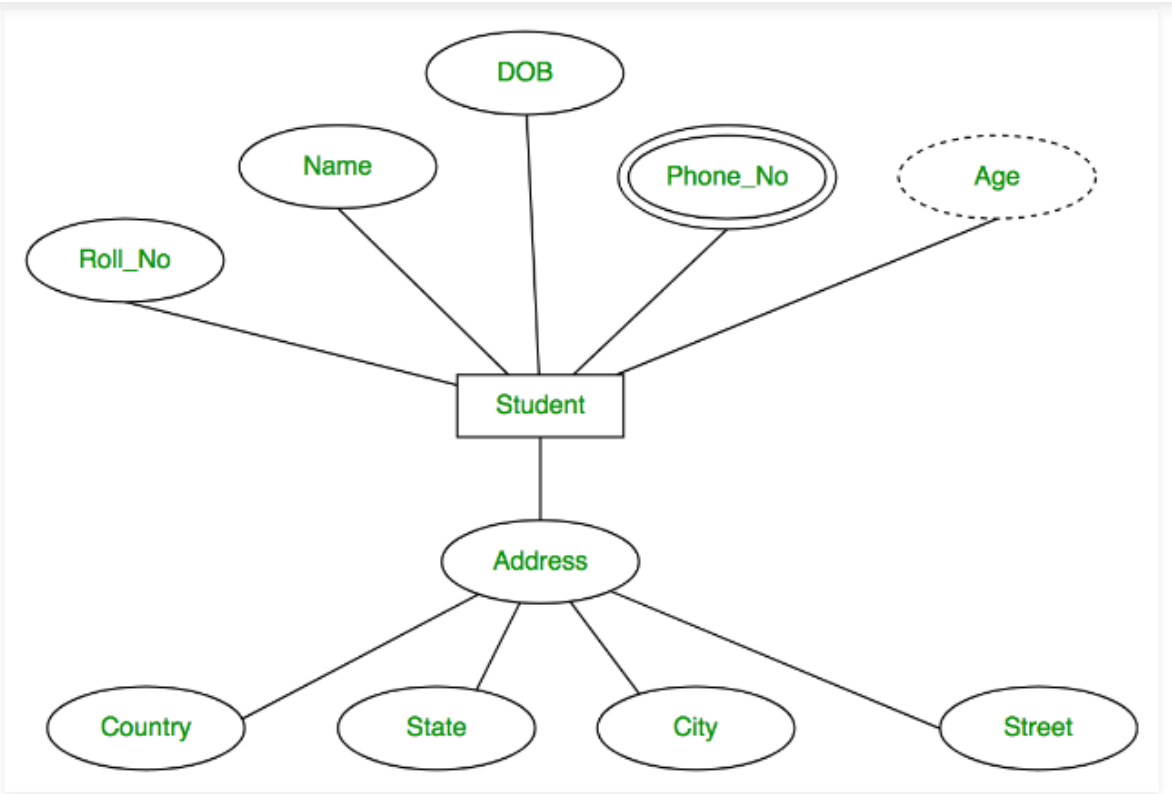


- **Single-value attribute:** Single-value attributes contain single value.
  For example – Roll Number

- **Multi-value attribute:** Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

ER Model (Entity Relationship Model)

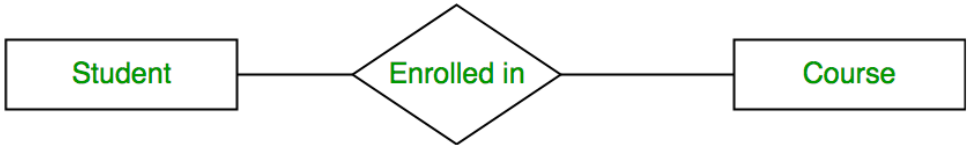
The complete entity type **Student** with its attributes can be represented as:



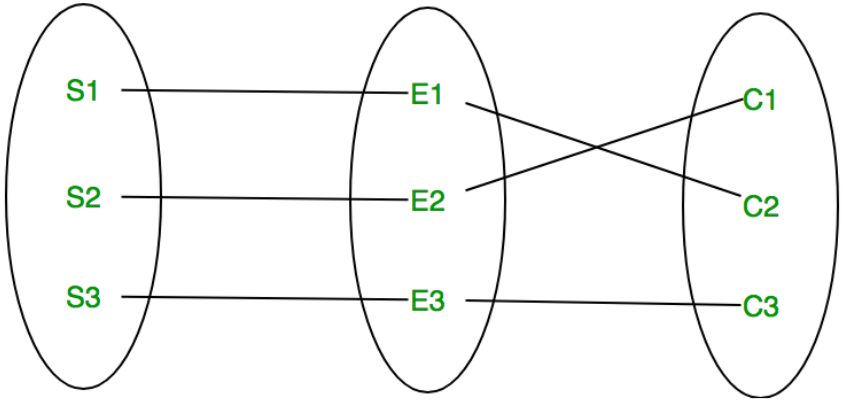
ER Model (Entity Relationship Model)

Relationship Type and Relationship Set

A relationship type represents the **association between entity types**. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



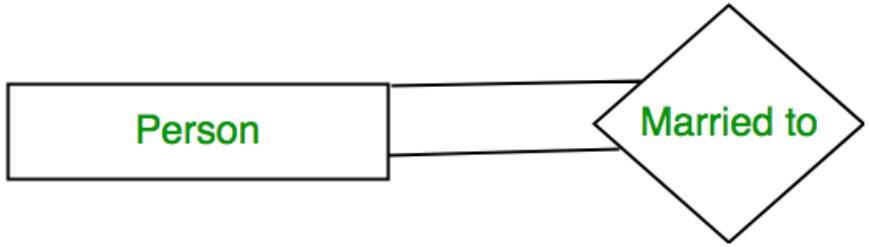
ER Model (Entity Relationship Model)

Degree of a relationship set

The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

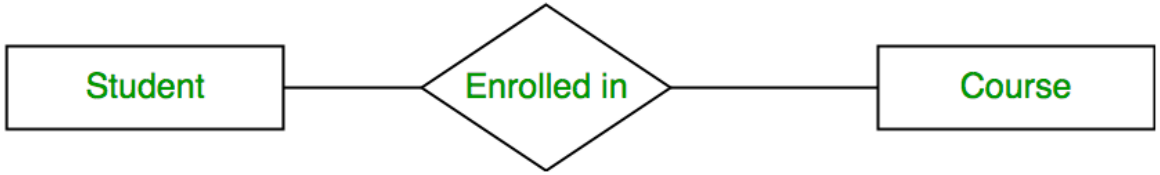
Unary Relationship

When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only (



Binary Relationship

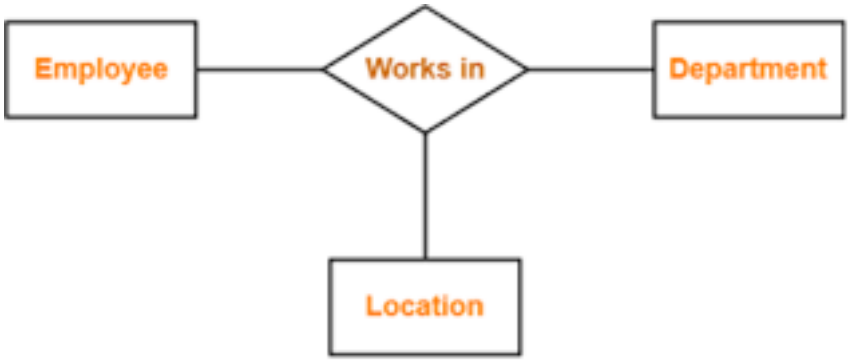
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship. For example, Student is enrolled in Course.



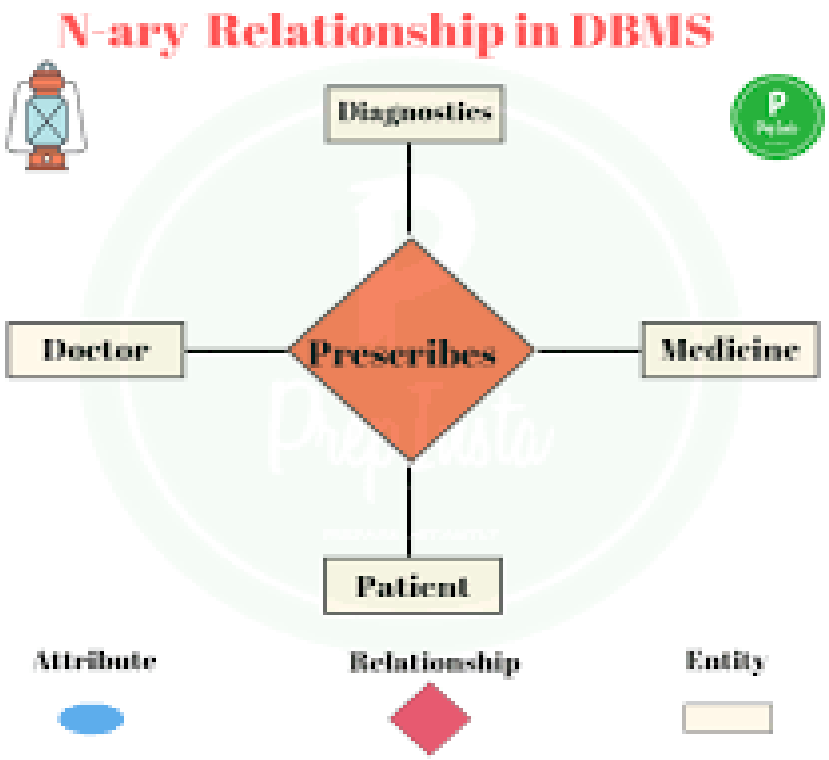
ER Model (Entity Relationship Model)

n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.



Ternary Relationship Set

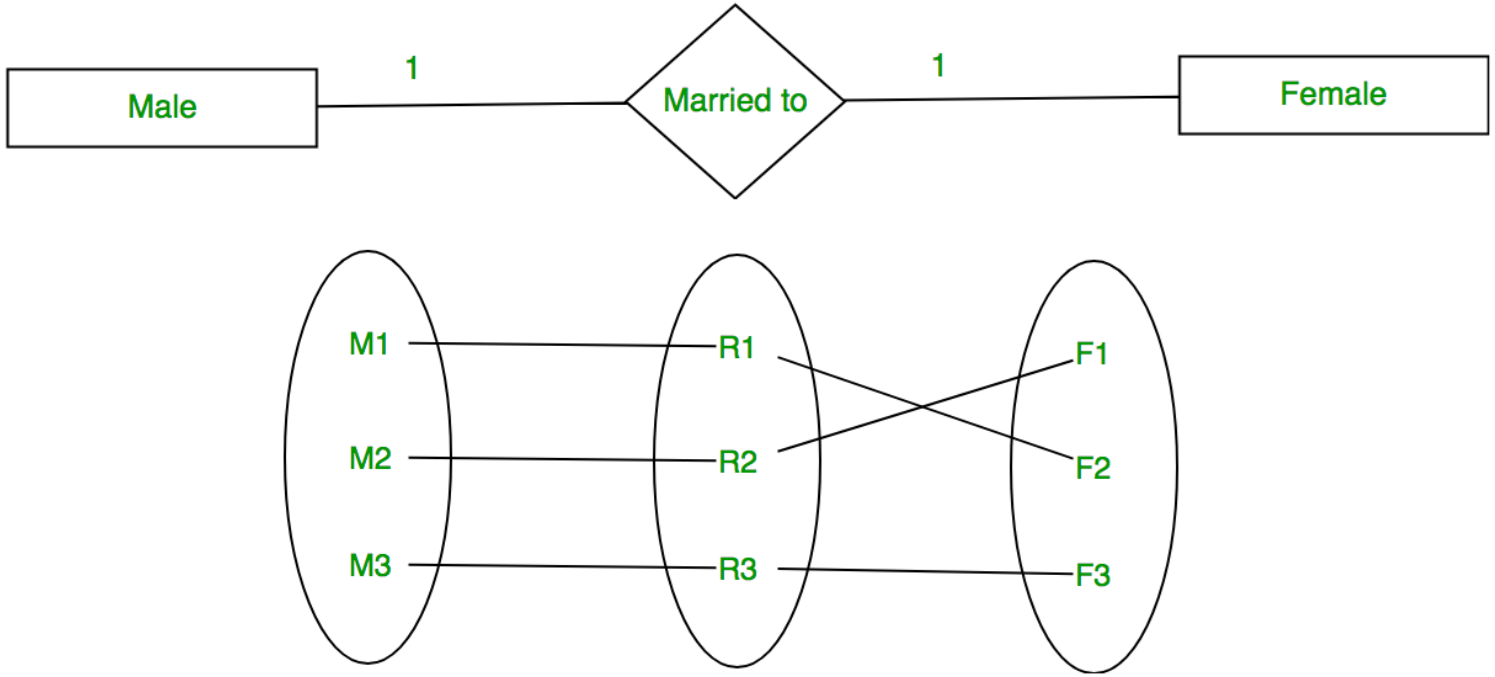


ER Model (Entity Relationship Model)

Cardinality

The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:

One to one – When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

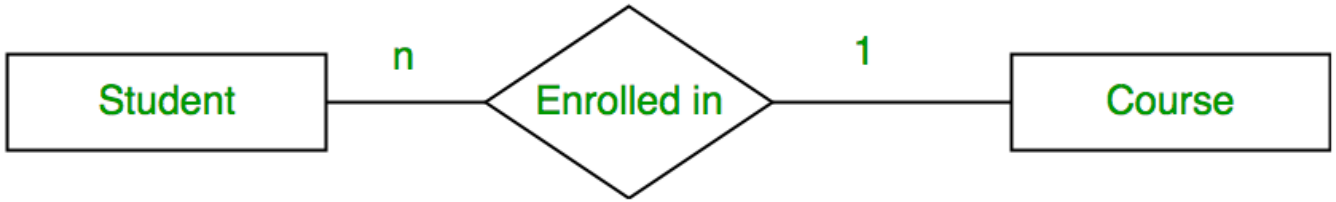


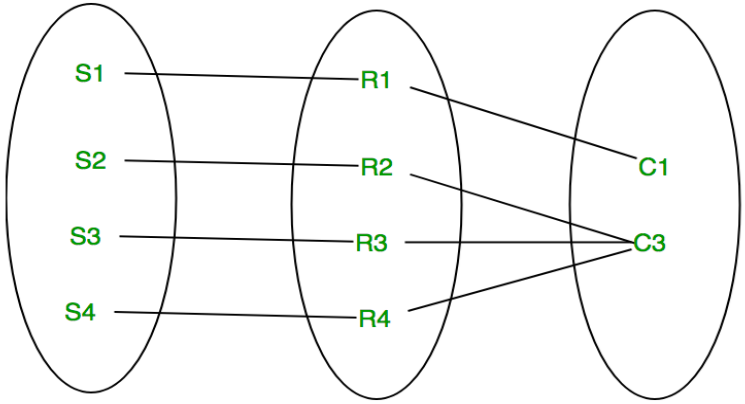
ER Model (Entity Relationship Model)

Many to one – When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one.

Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

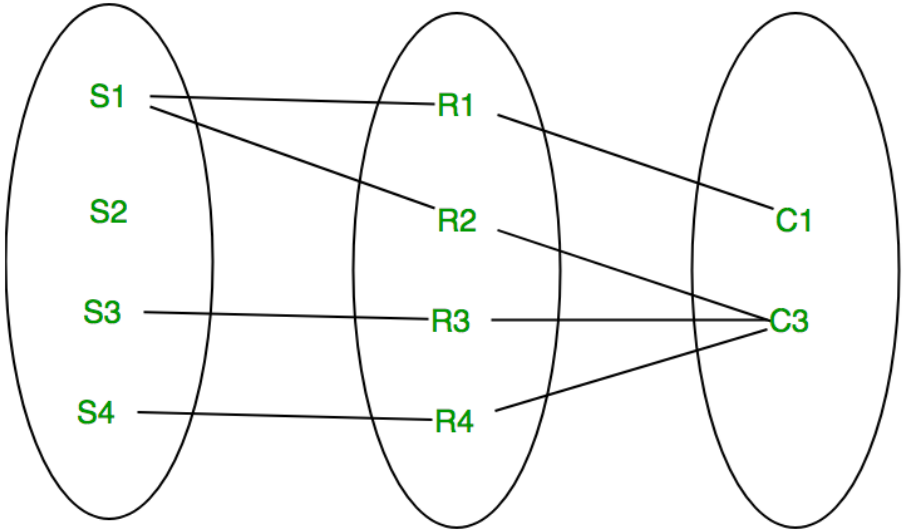
In this case, each student is taking only 1 course but 1 course has been taken by many students.





ER Model (Entity Relationship Model)

Many to many – When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.
In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

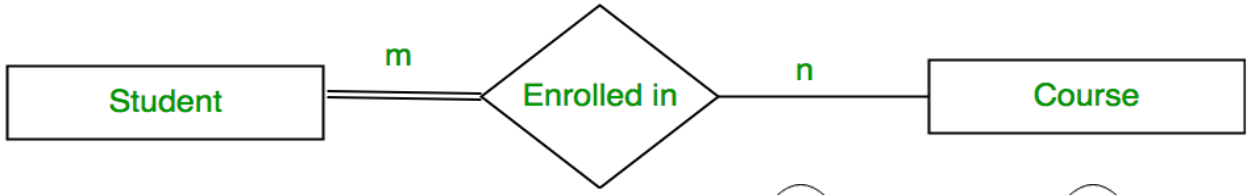


ER Model (Entity Relationship Model)

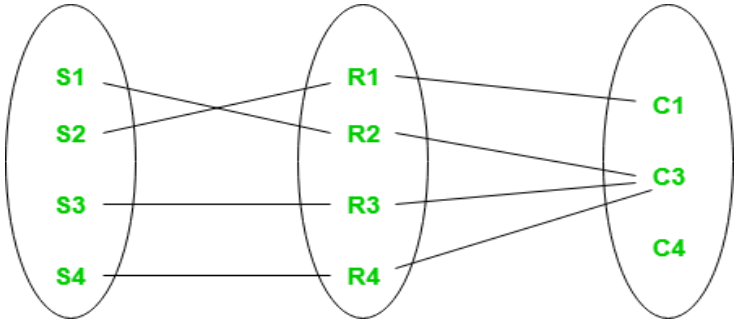
Participation Constraint

Participation Constraint is applied on the entity participating in the relationship set.

1. **Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.
2. **Partial Participation –** The entity in the entity set **may or may NOT participate** in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial. The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



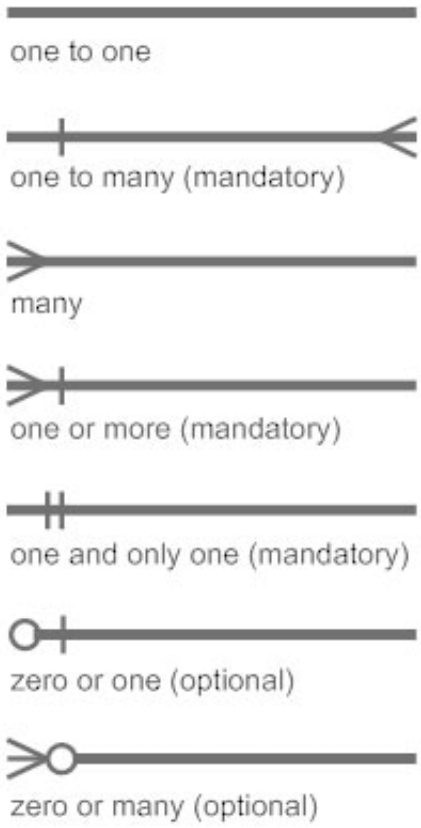
Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.



ER Model (Entity Relationship Model)

Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:



ER Model (Entity Relationship Model)

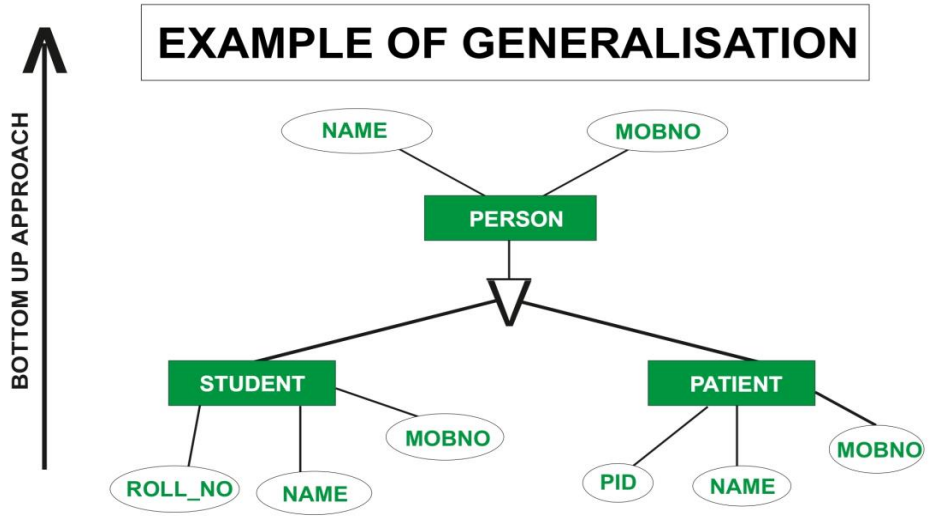
Generalization

Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.

In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.

Generalization is more like subclass and super class system, but the only difference is the approach. Generalization uses the bottom-up approach.

In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a super class.



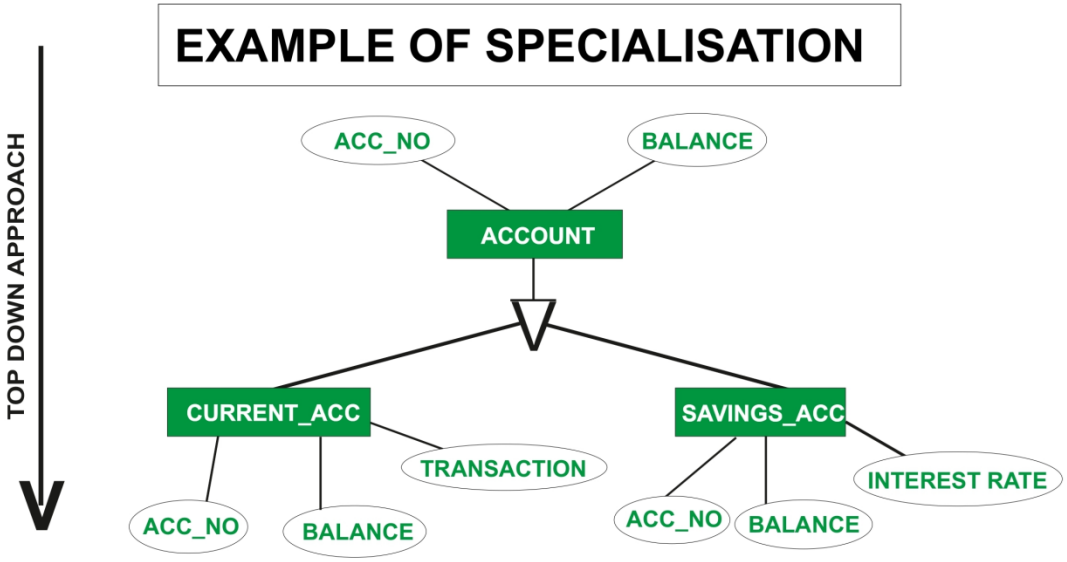
ER Model (Entity Relationship Model)

Specialization

Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

Normally, the super class is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

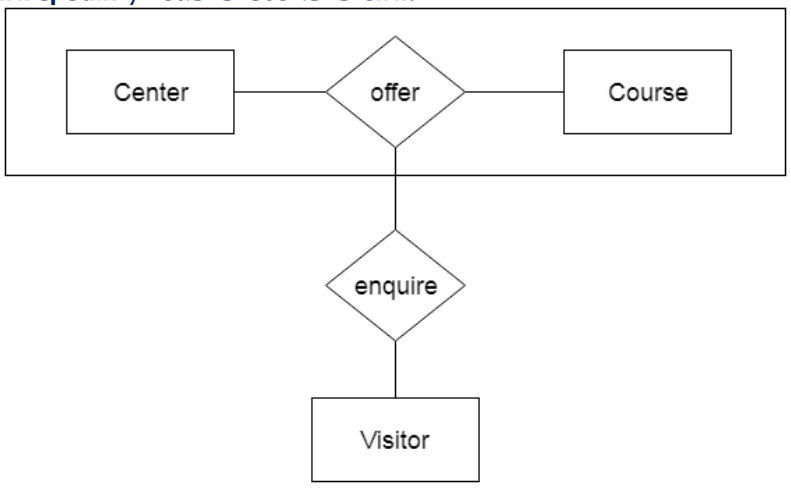


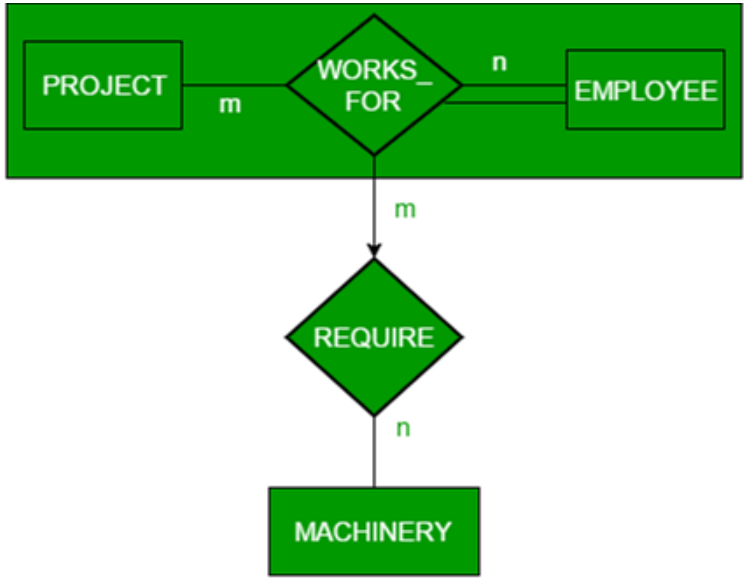
ER Model (Entity Relationship Model)

Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

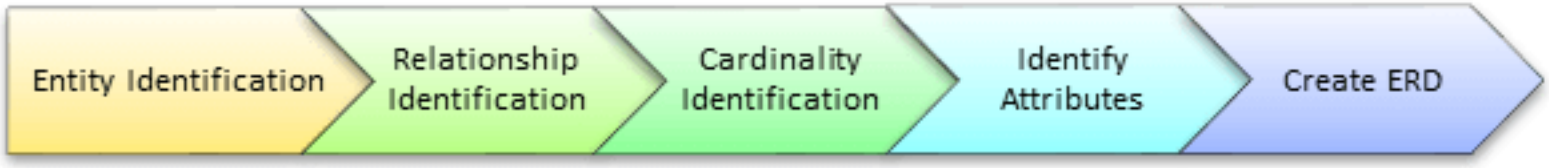




Aggregation

Steps to Create an ERD

Following are the steps to create an ERD.

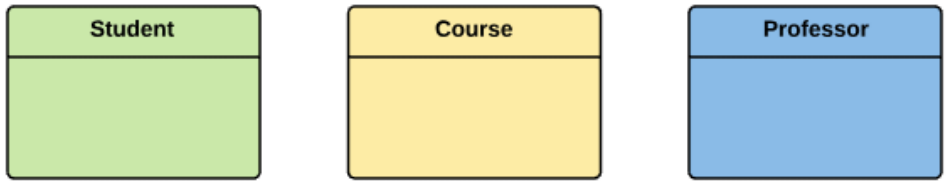


Let's study them with an example:
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

Step 1) Entity Identification

We have three entities

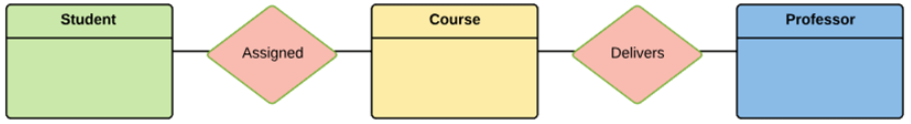
1. Student
2. Course
3. Professor



Step 2) Relationship Identification

We have the following two relationship

1. The student is **assigned** a cours
2. Professor **delivers** a course

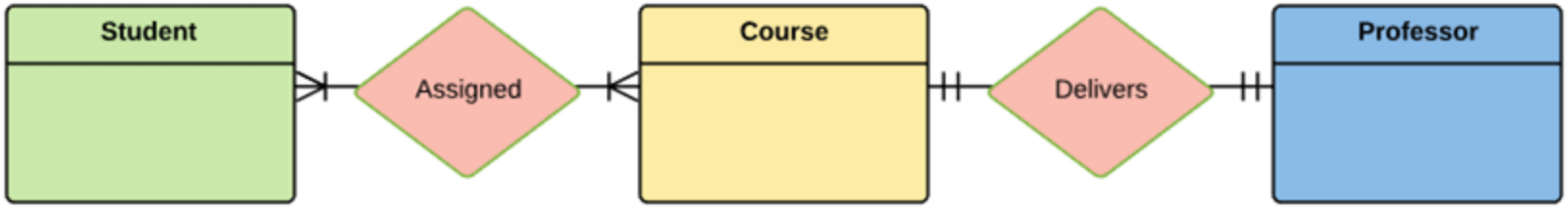


Steps to Create an ERD

Step 3) Cardinality Identification

For them problem statement we know that,

1. A student can be assigned **multiple** courses
2. A Professor can deliver only **one** course



Step 4) Identify Attributes

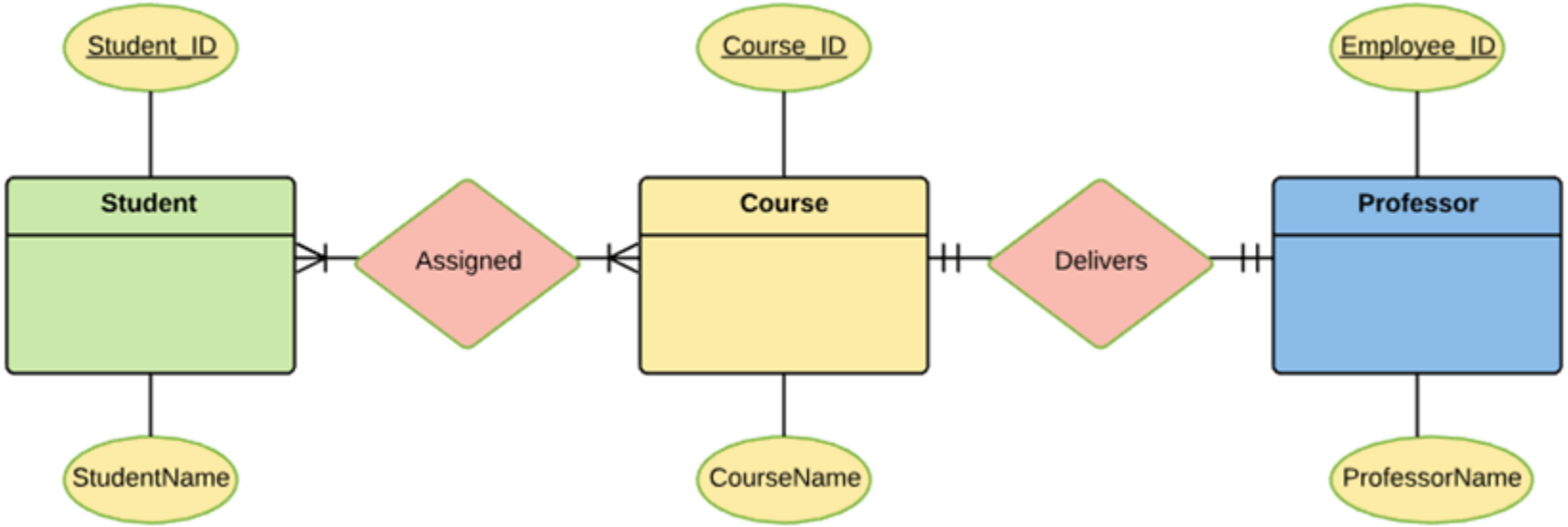
You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Steps to Create an ERD

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

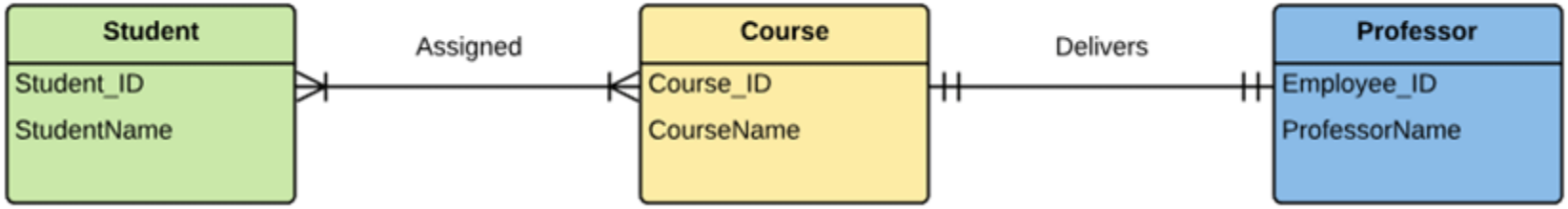


For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Steps to Create an ERD

Step 5) Create the ERD

A more modern representation of ERD Diagram

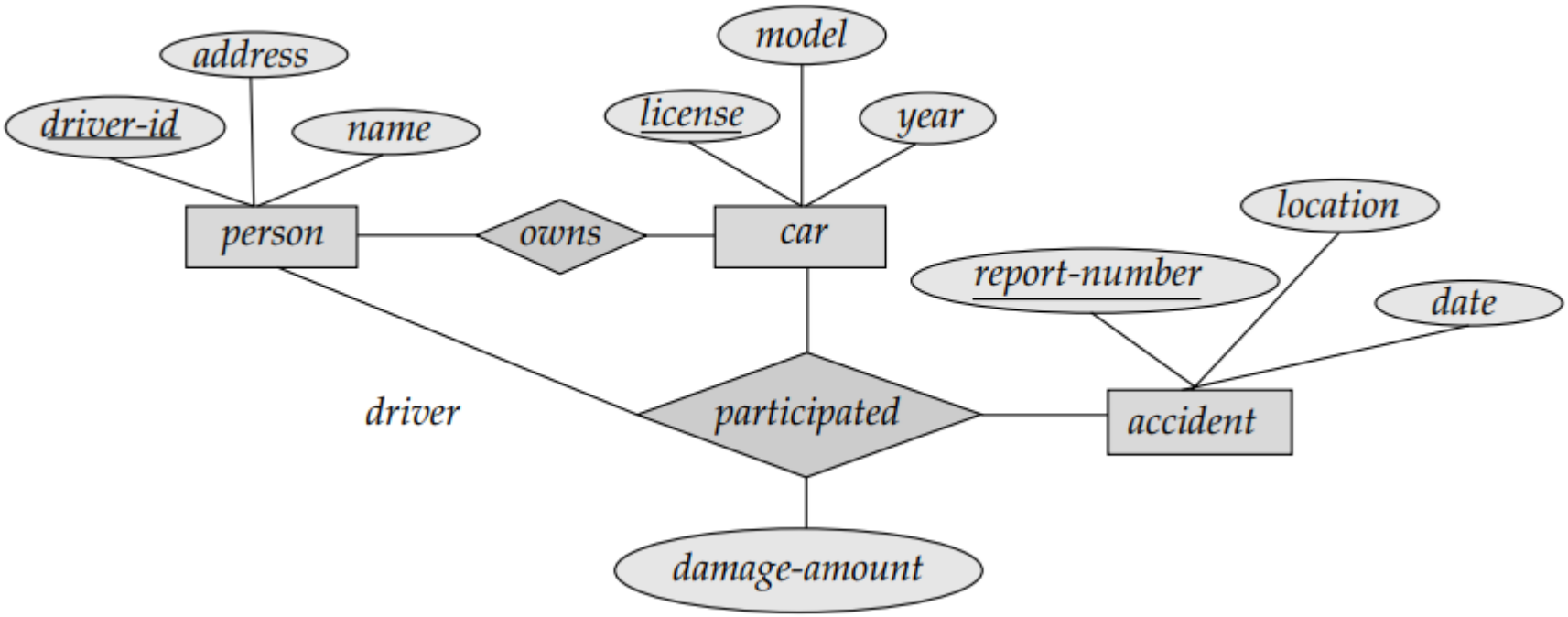


Best Practices for Developing Effective ER Diagrams

1. Eliminate any redundant entities or relationships
2. You need to make sure that all your entities and relationships are properly labeled
3. There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
4. Never connect relationships to each other

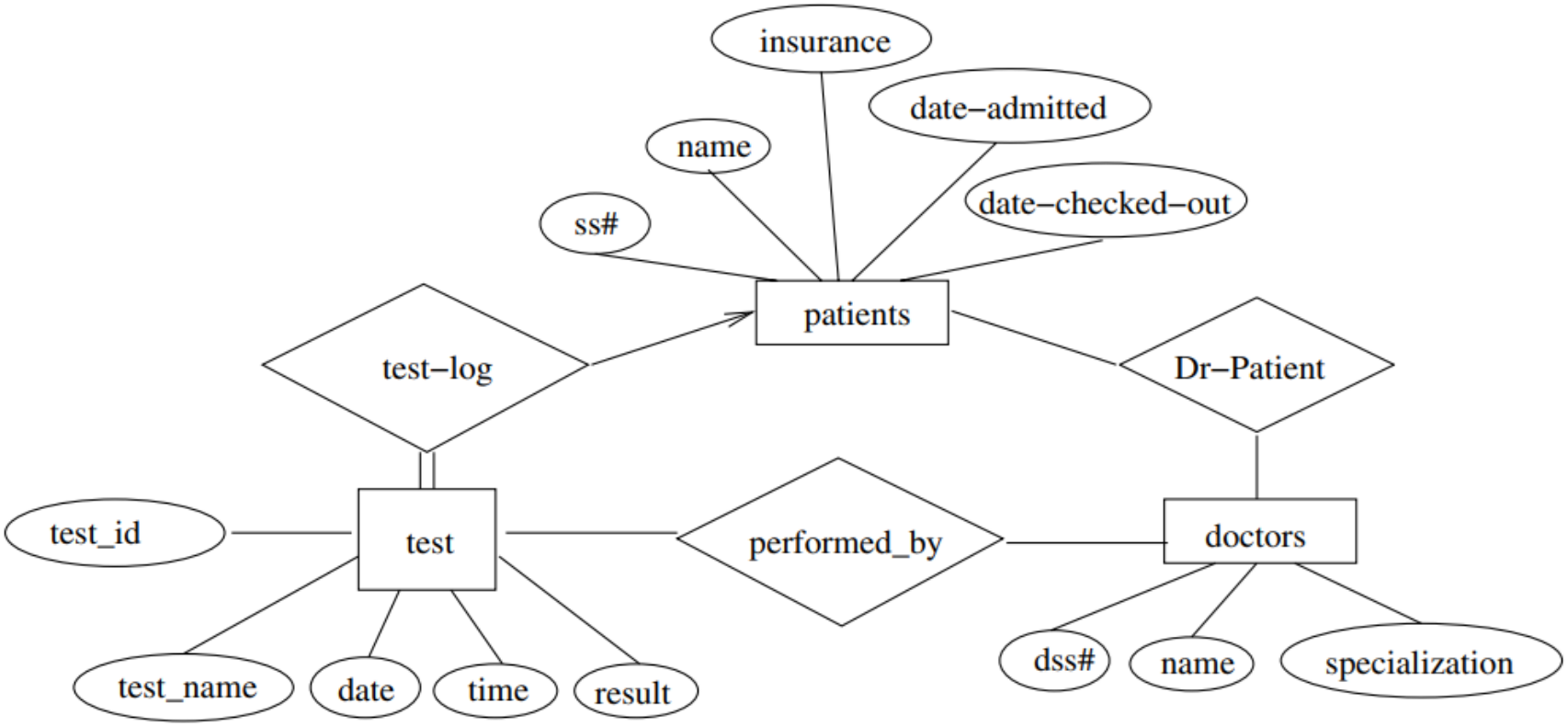
Car-Insurance Company

Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.



Hospital Record Management System

Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.



University Registrar Office

A university registrar's office maintains data about the following entities:

(a) courses, including number, title, credits, syllabus, and prerequisites;

(b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;

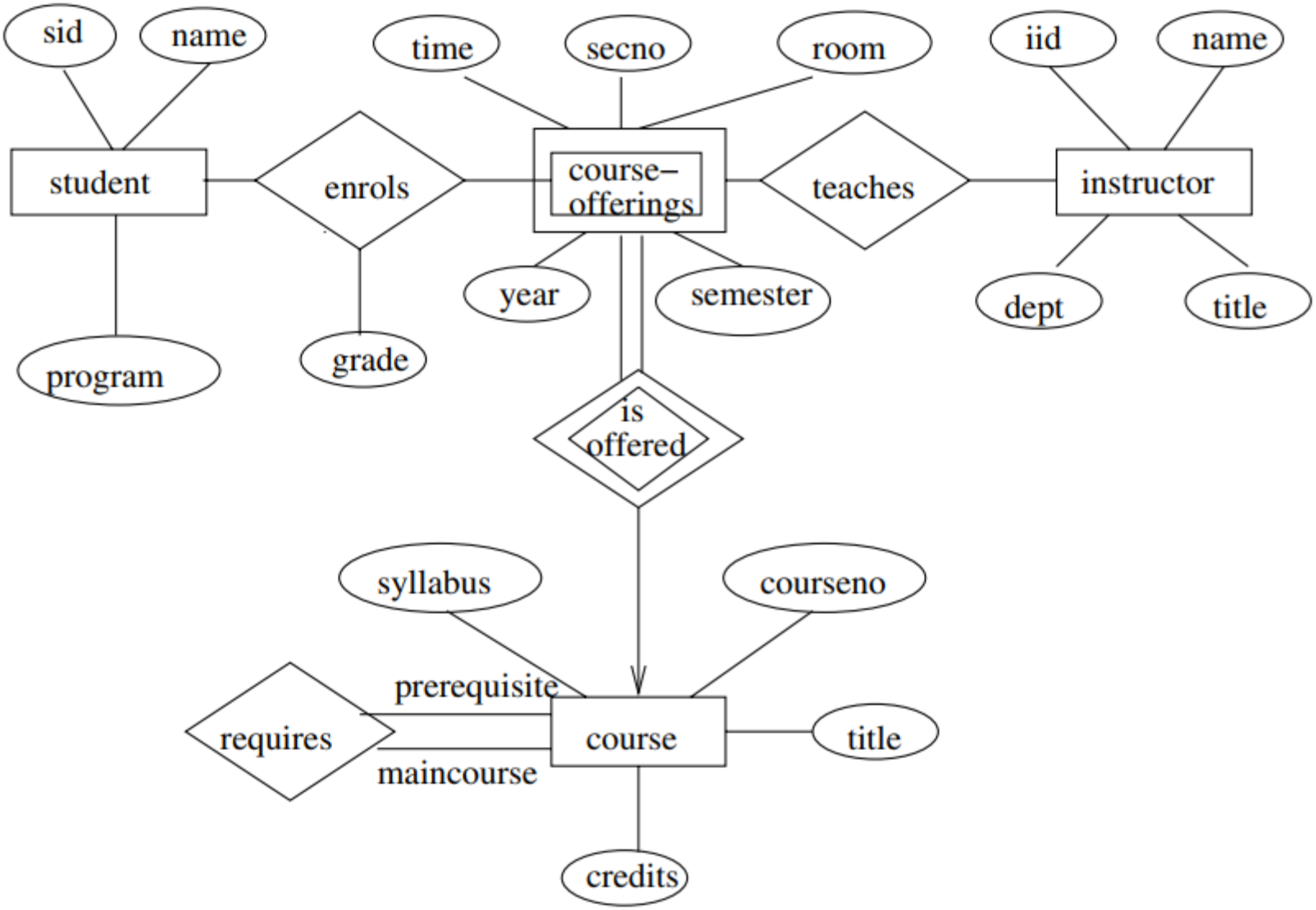
(c) students, including student-id, name, and program; and

(d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that

you make about the mapping constraints.

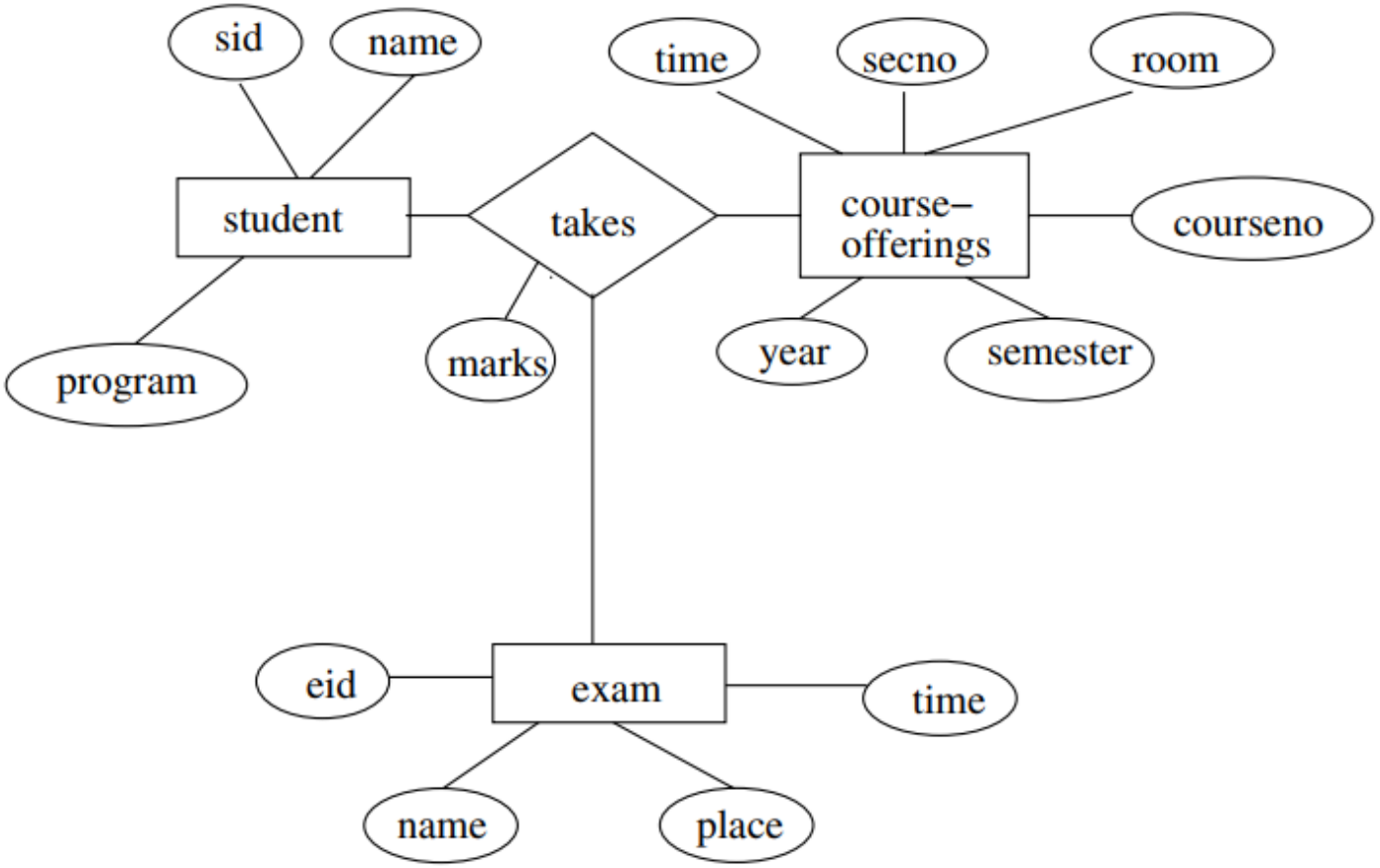
University Registrar Office



Model Examination System

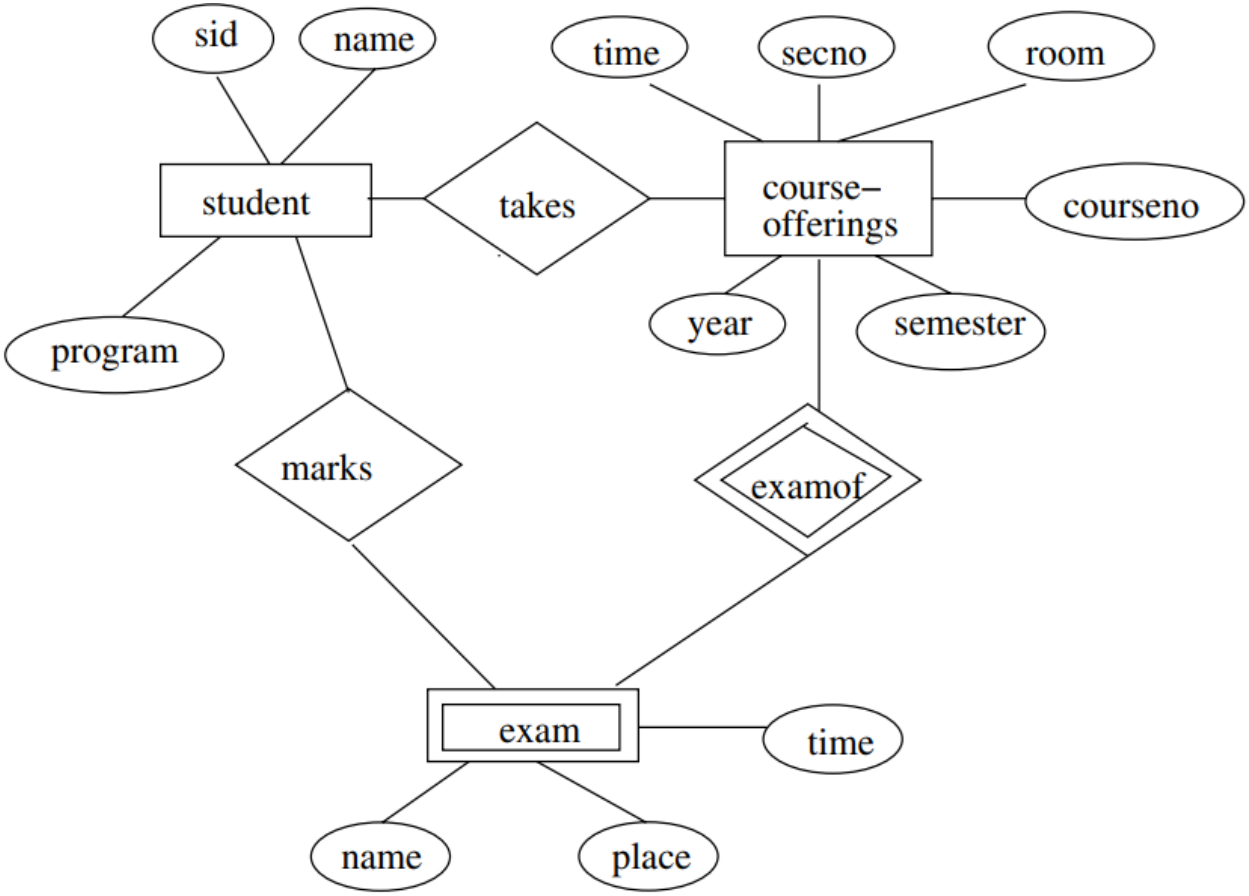
Consider a database used to record the marks that students get in different exams of different course offerings.

1. Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.



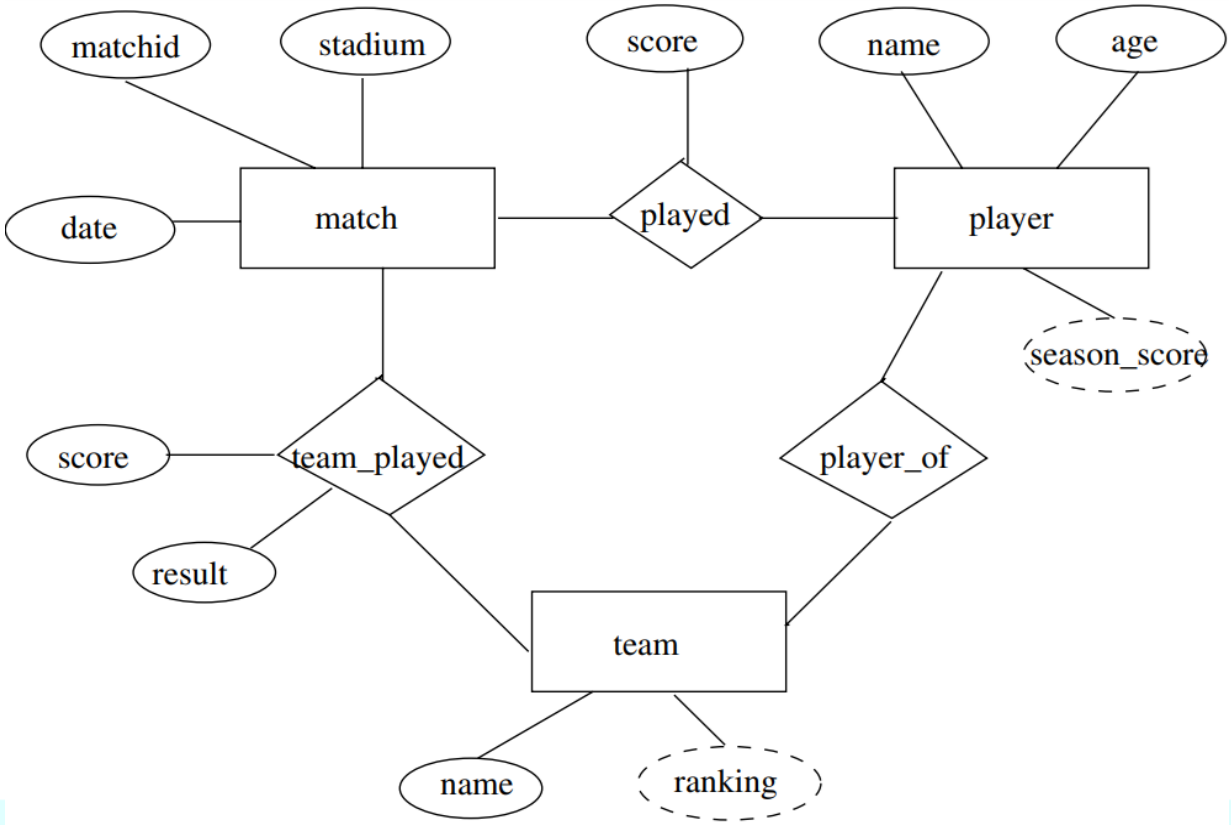
Model Examination System

2. Construct an alternative E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering



Model Examination System

3. Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes. Extend the E-R diagram of the previous question to track the same information for all teams in a league.



Music Database

The Music Database

The music database stores details of a personal music library, and could be used to manage your MP3, CD, or vinyl collection. Because this database is for a personal collection, it's relatively simple and stores only the relationships between artists, albums, and tracks. It ignores the requirements of many music genres, making it most useful for storing popular music and less useful for storing jazz or classical music.

We first draw up a clear list of requirements for our database:

- The collection consists of albums.
- An album is made by exactly one artist.
- An artist makes one or more albums.
- An album contains one or more tracks
- **Artists, albums, and tracks each have a name.**
- Each track is on exactly one album.
- Each track has a time length, measured in seconds.
- When a track is played, the date and time the playback began (to the nearest second) should be recorded; this is used for reporting when a track was last played, as well as the number of times music by an artist, from an album, or a track has been played.

Music Database

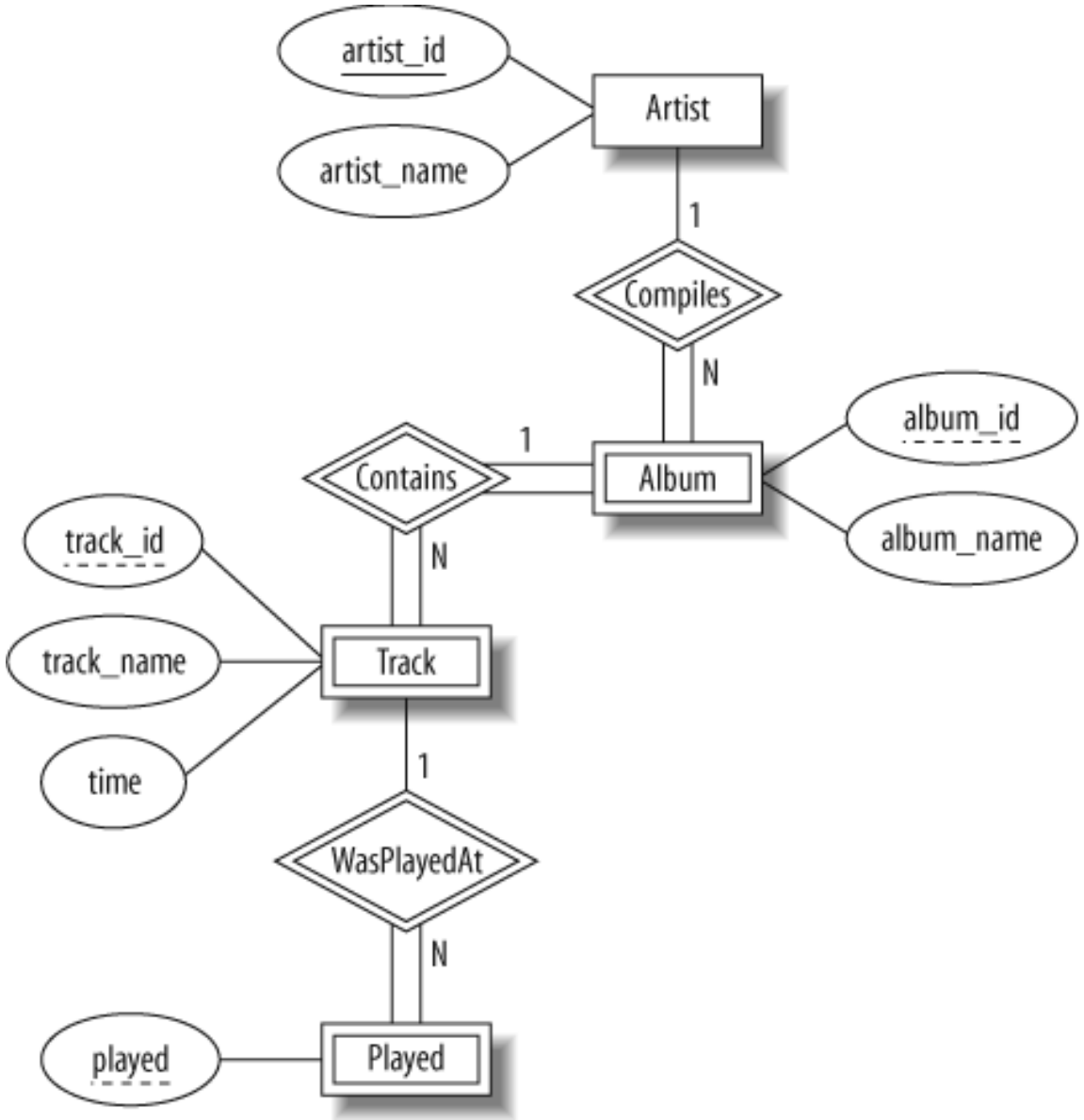
There's no requirement to capture composers, group members or sidemen, recording date or location, the source media, or any other details of artists, albums, or tracks.
The ER diagram derived from our requirements is shown in Figure.

You'll notice that it consists of only one-to-many relationships:

- one artist can make many albums,
- one album can contain many tracks, and
- one track can be played many times.

Conversely, each play is associated with one track, a track is on one album, and an album is by one artist. The attributes are straightforward: artists, albums, and tracks have names, as well as identifiers to uniquely identify each entity. The track entity has a time attribute to store the duration, and the played entity has a timestamp to store when the track was played.

Music Database



University Database

The University Database

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in. The database is a long way from one that'd be suitable for a large tertiary institution, but it does illustrate relationships that are interesting to query, and it's easy to relate to when you're learning SQL. We explain the requirements next and discuss their shortcomings at the end of this section.

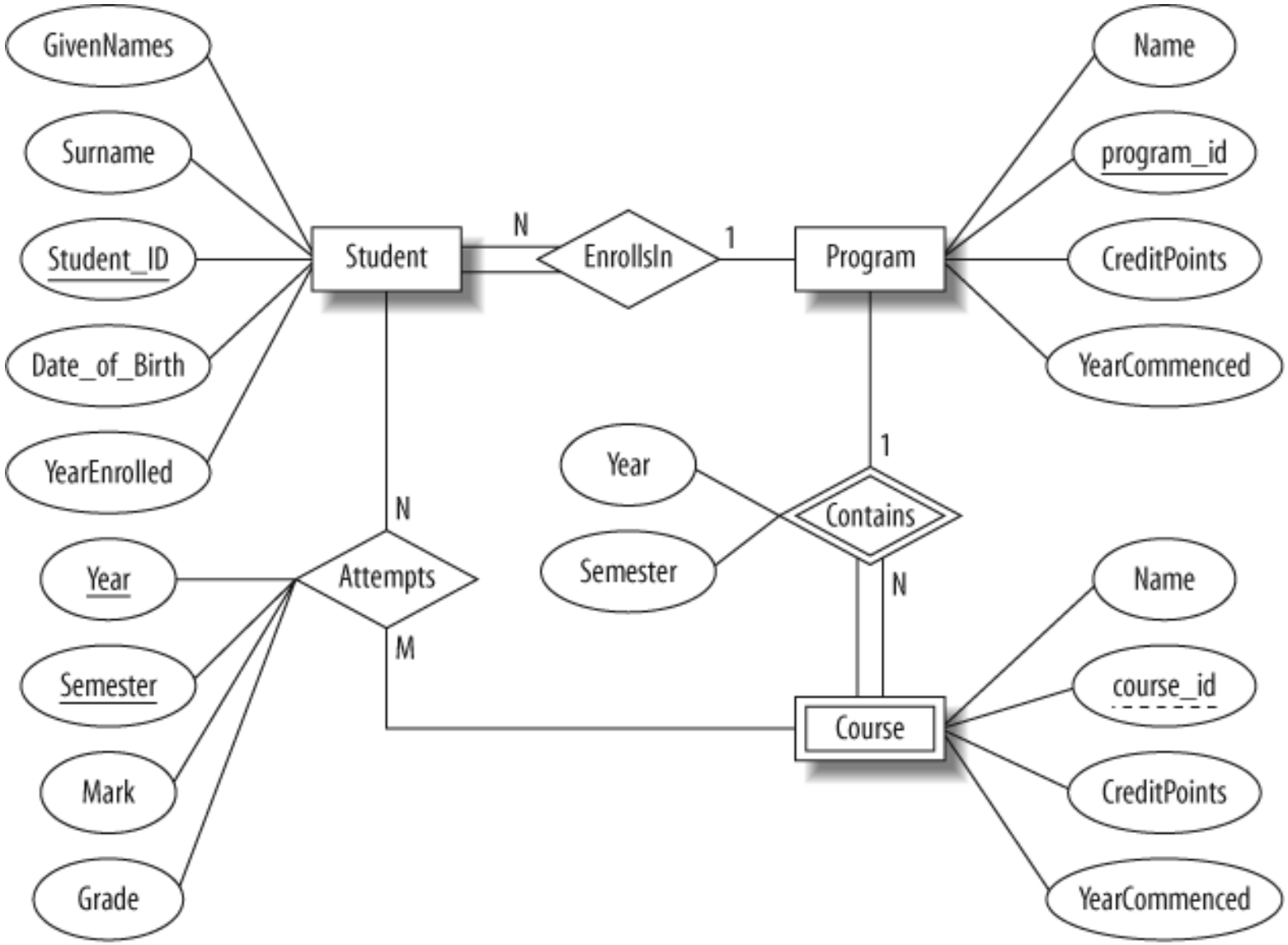
Consider the following requirements list:

- The university offers one or more programs.
- A program is made up of one or more courses.
- A student must enroll in a program.
- A student takes the courses that are part of her program.
- A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
- A course has a name, a course identifier, a credit point value, and the year it commenced.

University Database

- A course has a name, a course identifier, a credit point value, and the year it commenced.
- Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, “John Paul.”
- When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
- Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).
- The ER diagram derived from our requirements is shown in Figure. Although it is compact, the diagram uses some advanced features, including relationships that have attributes and two many-to-many relationships.

University Database



University Database

In our design:

- Student is a strong entity, with an identifier, student_id, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).
- Program is a strong entity, with the identifier program_id as the primary key used to distinguish between programs.
- Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one Enrolls In relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.
- A Course has meaning only in the context of a Program, so it's a weak entity, with course_id as a weak key. This means that a Course is uniquely identified using its course_id and the program_id of its owning program.
- As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.
- Student and Course are related through the many-to-many Attempts relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.
- When a student attempts a course, there are attributes to capture the Year and Semester, and the Mark and Grade.

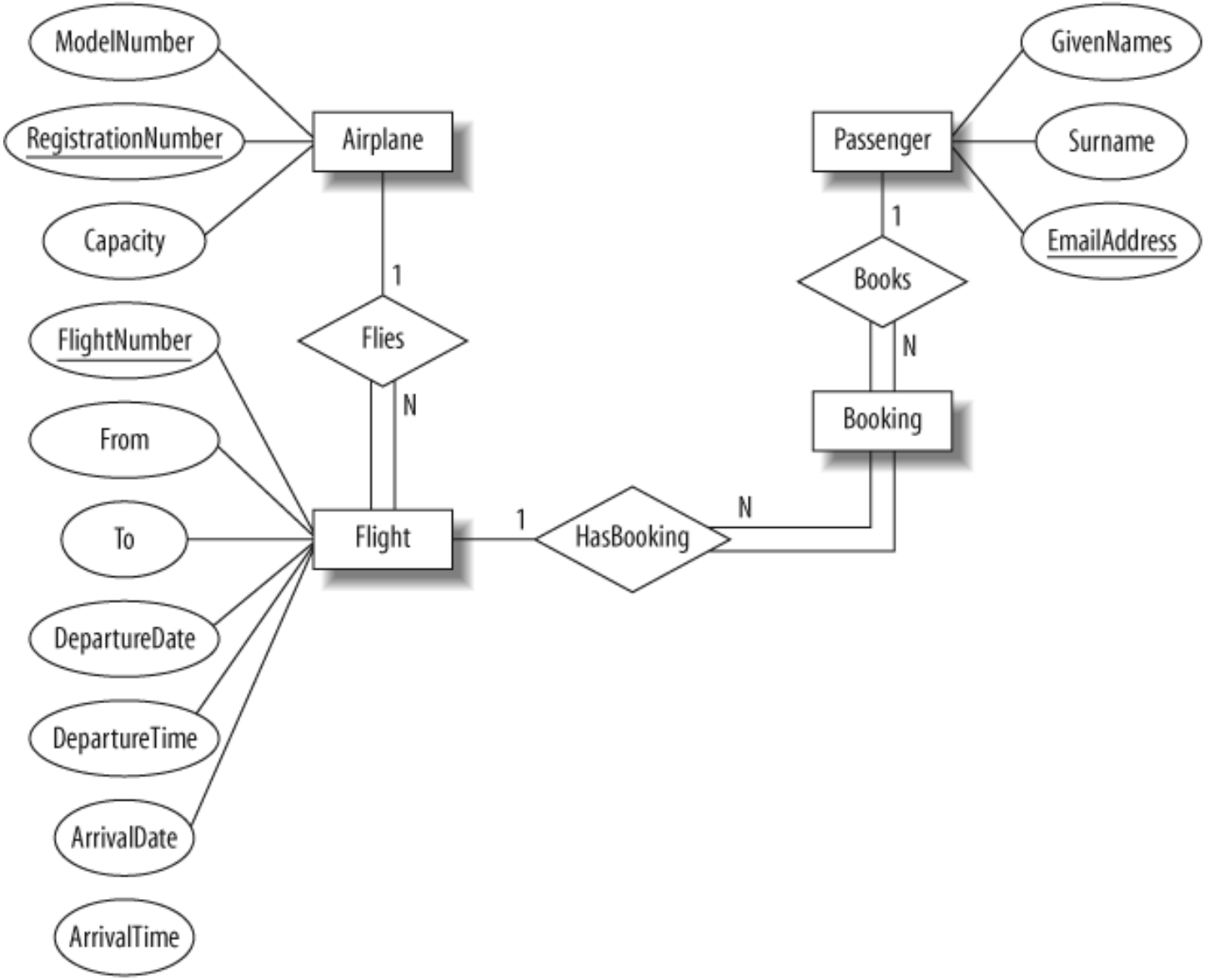
Flight Database

The Flight Database

- The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same.
- Consider the following requirements list:
- The airline has one or more airplanes.
- An airplane has a model number, a unique registration number, and the capacity to take one or more passengers.
- An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time.
- Each flight is carried out by a single airplane.
- A passenger has given names, a surname, and a unique email address.
- A passenger can book a seat on a flight.

The ER diagram derived from our requirements is shown in Figure:

Flight Database



Flight Database

- An Airplane is uniquely identified by its RegistrationNumber, so we use this as the primary key.
- A Flight is uniquely identified by its FlightNumber, so we use the flight number as the primary key. The departure and destination airports are captured in the From and To attributes, and we have separate attributes for the departure and arrival date and time.
- Because no two passengers will share an email address, we can use the EmailAddress as the primary key for the Passenger entity.
- An airplane can be involved in any number of flights, while each flight uses exactly one airplane, so the Flies relationship between the Airplane and Flight relationships has cardinality 1:N; because a flight cannot exist without an airplane, the Flight entity participates totally in this relationship.
- A passenger can book any number of flights, while a flight can be booked by any number of passengers.
- We could specify an M:N Books relationship between the Passenger and Flight relationship, but considering the issue more carefully shows that there is a hidden entity here: the booking itself.
- We capture this by creating the intermediate entity Booking and 1:N relationships between it and the Passenger and Flight entities. Identifying such entities allows us to get a better picture of the requirements.

Relational Data Model

Relational Data Model uses a collection of tables to represent both data and the relationship among those data.

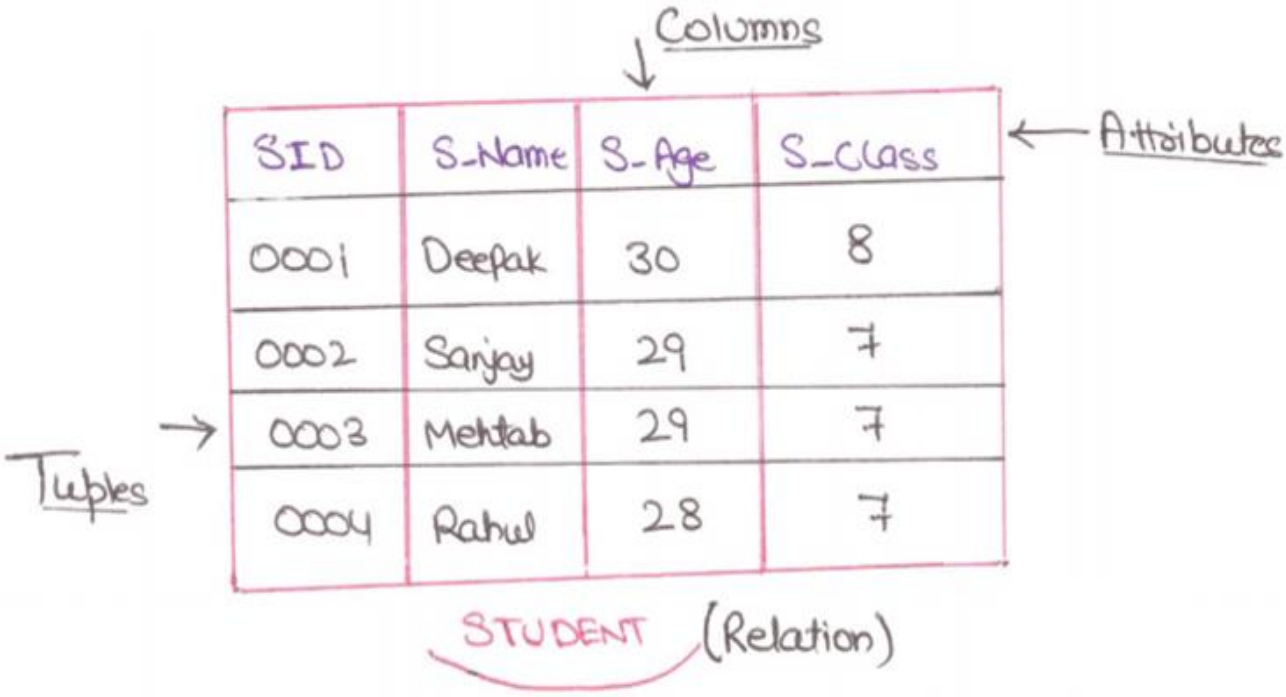
1) Each table has multiple columns and each column as **unique name.**
2) The Data is arranged in a Relation which is visually represented in a **two dimensional table.**
3) The data is inserted into the table in the form of Tuples (Rows).A Tuple is formed by one or more than one attributes. **Tuple in this example is a row (complete row).**
4) Attributes are used as basic building block in the formation of various expression that are used to drive a meaningful information.
5) There can be number of **tuples** in the table (relation), but all the tuples contains fixed and some attributes with varying values.
6) A **Relation** is represented by a **table.**

Relational Data Model

Tuple is represented by row.
An **attribute** is represented by a **column of the table.** Attribute name is the name of the column

Example S_ID, S_Name, S_Age.

Attribute values contains the values for the column in the row.



Codd's Rules

Codd's Rules, formulated by **Dr. Edgar F. Codd**, define the criteria that a database management system (DBMS) must satisfy to be considered a **relational database management system (RDBMS)**. These **12 rules (actually 13, as Rule 0 is included)** ensure data integrity, consistency, and efficient handling of relational data.

Codd's 12 Rules for RDBMS

Rule 0: Foundation Rule

- The system must manage databases entirely through its **relational capabilities**.

Rule 1: Information Rule

- All information in the database must be **stored in tables** as values in rows and columns.

Rule 2: Guaranteed Access Rule

- Every data item should be uniquely accessible using a **combination of table name, primary key, and column name**.

Codd's Rules

Rule 3: Systematic Treatment of NULL Values

- NULL values should be **distinctly stored** and **treated differently** from other values (not confused with zero or empty string).

Rule 4: Dynamic Online Catalog Based on the Relational Model

- The database metadata (schema, tables, etc.) must be stored **in relational tables** and accessible using SQL queries.

Rule 5: Comprehensive Data Sublanguage Rule

- The system must support at least one **relational language** (like SQL) for data definition, manipulation, and transaction control.

Rule 6: View Updating Rule

- The database must allow updates to **views (virtual tables)** as if they were real tables, whenever possible.

Rule 7: High-Level Insert, Update, and Delete

- The system must support **set-based operations**, allowing manipulation of multiple rows in a single operation.

Codd's Rules

Rule 8: Physical Data Independence

- Changes in **physical storage** should not affect how data is accessed by users or applications.

Rule 9: Logical Data Independence

- Changes in **table structure (schema modifications)** should not require application modifications.

Rule 10: Integrity Independence

- Integrity constraints (like primary keys, foreign keys, and domain constraints) should be **stored in the database** and not in application code.

Rule 11: Distribution Independence

- The system should allow **distributed databases** without requiring changes to applications.

Rule 12: Nonsubversion Rule

- If the system provides a **low-level access method**, it must not bypass the relational security and integrity constraints.

Relational Data Model

Constraint - Set of rules and limitations:

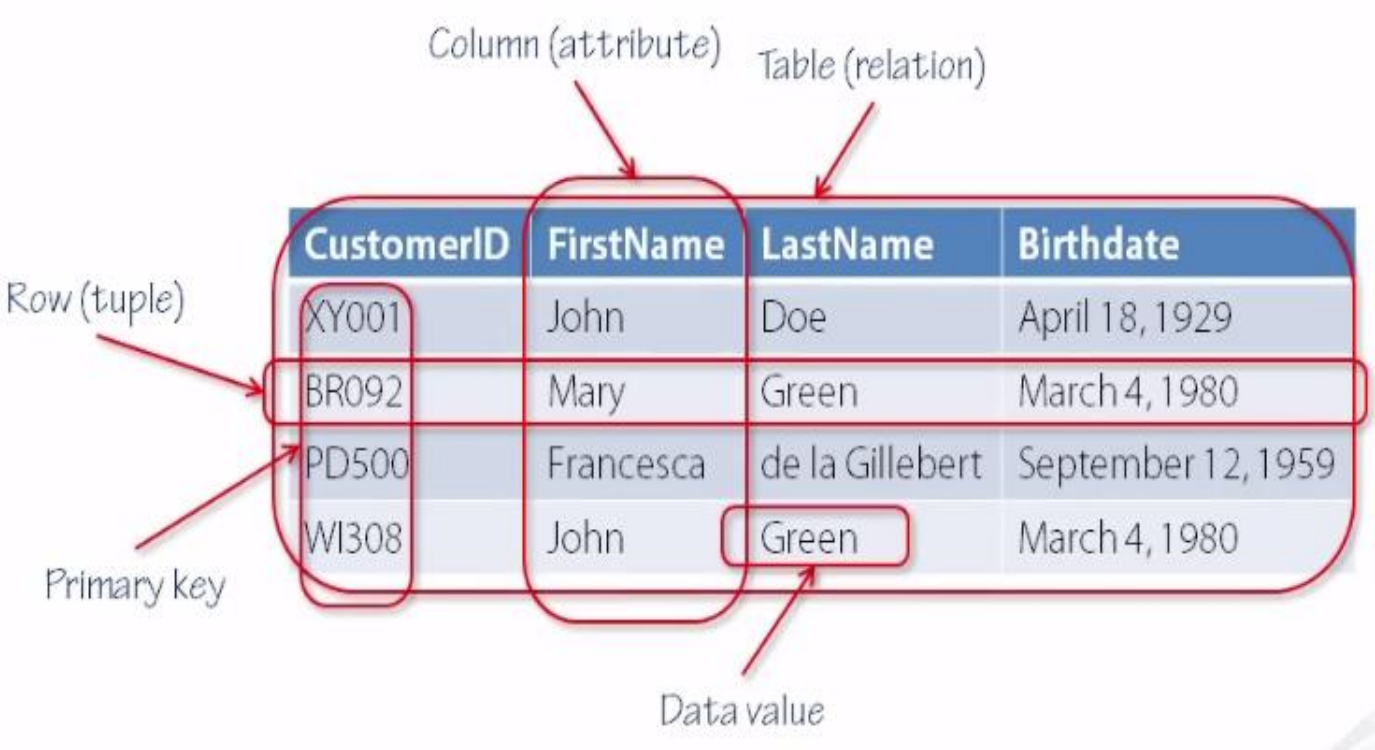
Constraints are applied to tables and forms the logical schema.

- To select a particular row/tuple from table/relation we use attribute/columns name with the help of unique value field of an attributes.
- This field which are unique from other fields are used as indexes which helps in searching fast.
- All the relational algebra operations, like Select, Intersection, Product, union, join, Division, Merge can also be performed on the relation data model.
- Operations on RDM (Relational Data Model) are facilitated with the help of different conditional expression, various key attributes, and predefined constraints etc.
- Data Integrity is maintained by process like Normalization.
- Description of data in terms of this model is called a schema.

Schema for relation specifies its name , name of each field.

Student(S_id: Integer, S_Name: String, S_Login : String etc)

Relational Data Model



CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

Relational Data Model

Relational Model Concepts

- **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student, Rollno, NAME etc.
- **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.
- **Tuple** – It is nothing but a single row of a table, which contains a single record.
- **Relation Schema:** A relation schema represents the name of the relation with its attributes.
- **Degree:** The total number of attributes which in the relation is called the degree of the relation.
- **Cardinality:** Total number of rows present in the Table.
- **Column:** The column represents the set of values for a specific attribute.
- **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
- **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
- **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain.

Relational Data Model

Example: STUDENT Relation

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>.

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 913988	Delhi	40

Relational Data Model

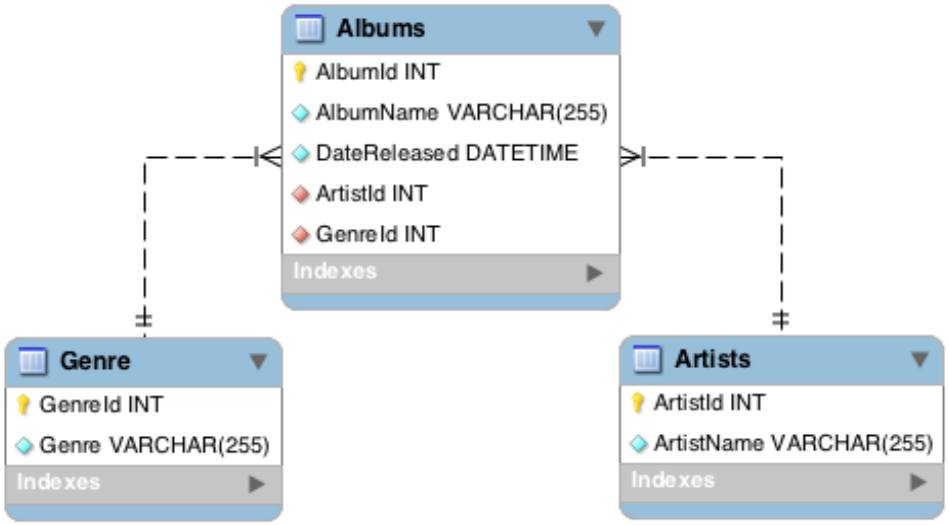
Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Schema, Subschema and Instance

Schema

- Schema is a the logical description of the database.
- The overall design of the database is called Database Schema.
- In database terms, a *schema* (pronounced “skee-muh” or “skee-mah”) is the organization and structure of a database. Both *schemas* and *schemata* can be used as plural forms.
- A schema contains schema objects, which could be tables, columns, data types, views, stored procedures, relationships, primary keys, foreign keys, etc.
- A database schema can be represented in a visual diagram, which shows the database objects and their relationship with each other.



Schema, Subschema and Instance

Schema and Database are the same things or different?

Part of the reason for the confusion is that database systems tend to approach schemas in their own way.

- **MySQL Documentation**
  A schema is synonymous with a database. Therefore, a schema and a database are the **same thing**.

- **Oracle Database Documentation**
  Certain objects can be stored inside a database but not inside a schema. Therefore, a schema and a database are **two different things**.

- **SQL Server Technical Article**
  A schema is a separate entity inside the database. So, they are **two different things**.

Schema, Subschema and Instance

Schema

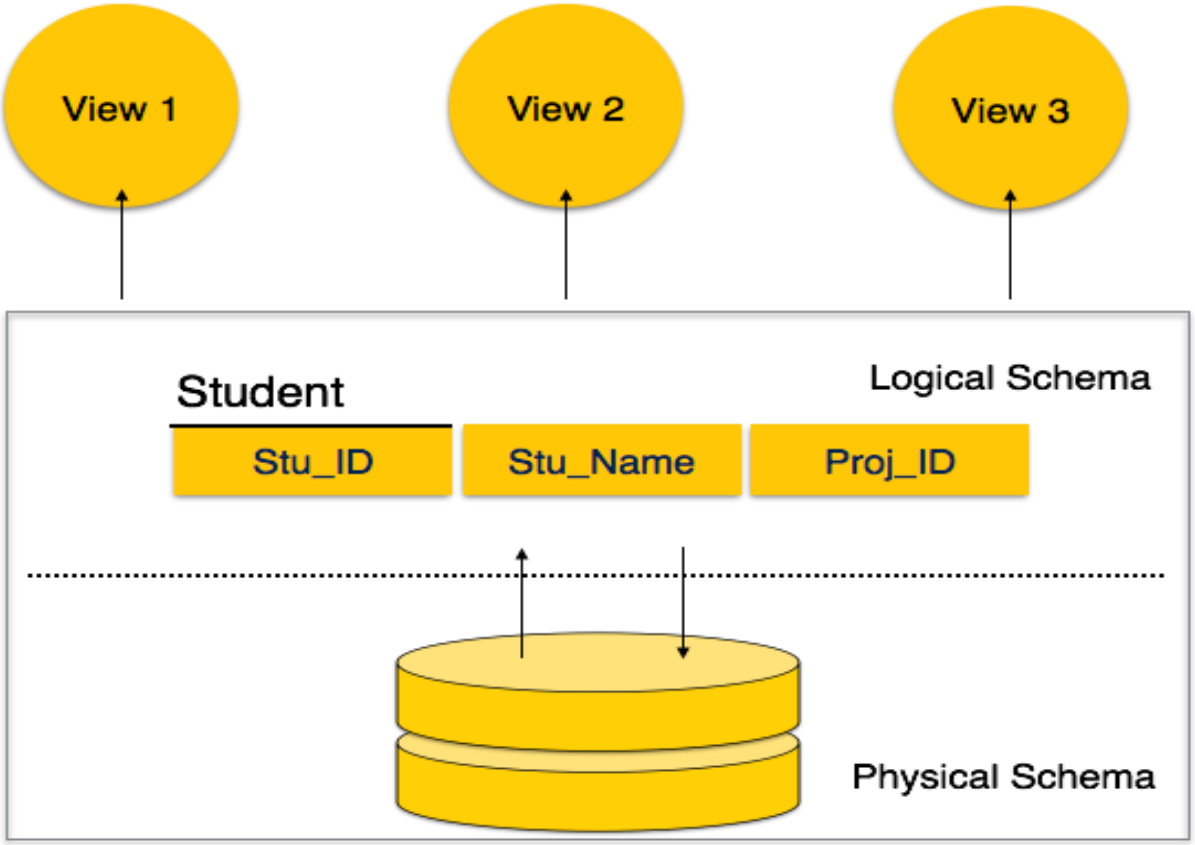
Schema is a the logical description of the database.
The overall design of the database is called Database Schema.
Database system has several schemas and partitioned according to the levels of abstractions. There are three level of schemas in database system.

1. **Internal Schema or Physical Schema**
   It describes that how data are actually stored in the blocks of storage devices such as hard disk.

2. **Logical Schema or Conceptual Schema**
   It describes the structure of the database to the database designer. Programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures

3. **External Schema or Subschema**
   It is a subset of main schema having the same properties as schema. It describes the different parts, sets, records and data names of the database to the end users. It allows users to view only the parts of the main database.

Schema, Subschema and Instance



Schema, Subschema and Instance

Database Instance

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

A database schema is variable declarations in a program. Variable has particular value at a given instant. Then, the value of variable at particular instant is called database instance.

Database schema (names of columns + the types associated with them)

Name	DOB	Address	Job	Scale
String	*Date*	*String*	*String*	*Int*

A database instance

Name	DOB	Address	Job	Scale
A. Johnson	2/04/1960	London	Programmer	12
B. Holiday	3/10/1947	Leeds	Analyst	14
C. Clark	12/08/1971	York	Programmer	10

Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

Integrity = (Correctness + Consistency)

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Relational Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

Create DOMAIN CustomerName CHECK (value not NULL)

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

Relational Integrity Constraints

Constraint Type	Code Snippet	Explanation
NOT NULL	CREATE TABLE Employees (name VARCHAR(50) NOT NULL);	Ensures that the name column cannot have NULL values, enforcing that every employee must have a name.
CHECK	CREATE TABLE Employees (age INT CHECK (age >= 18 AND age <= 65));	Ensures that the age column values must be between 18 and 65, enforcing that employees fall within this age range.
DEFAULT	CREATE TABLE Employees (hire_date DATE DEFAULT CURRENT_DATE);	Ensures that the hire_date column will default to the current date if no value is provided during insertion.
UNIQUE	CREATE TABLE Employees (email VARCHAR(100) UNIQUE);	Ensures that the email column values must be unique across all rows, preventing duplicate

Relational Integrity Constraints

```
CREATE TABLE Employees
(    id INT PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     age INT CHECK (age >= 18 AND age <= 65),
     email VARCHAR(100) UNIQUE,
     hire_date DATE DEFAULT CURRENT_DATE
);
```

This ensures domain integrity in our SQL Server by validating that all the values of the table are the ones we would expect:

- Every employee has a name.
- Employees are of a working age.
- Each employee has their own unique email.
- All hire dates are valid, and have a default just in case.

Relational Integrity Constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number.

Key constraints force that –

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute can not have NULL values.
- Key constraints are also referred to as Entity Constraints.

Relational Integrity Constraints

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

Types of keys in DBMS

1. **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
2. **Super Key** – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
3. **Candidate Key** – A super key with no redundant attribute is known as candidate key
4. **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
5. **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
6. **Foreign Key** – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Relational Integrity Constraints

Primary Key

A **primary key** is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) in that table.

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Primary Key Example in DBMS

Lets take an example to understand the concept of primary key. In the following table, there are three attributes: Stu_ID, Stu_Name & Stu_Age. Out of these three attributes, one attribute or a set of more than one attributes can be a primary key.

1. Attribute Stu_Name alone cannot be a primary key as more than one students can have same name.
2. Attribute Stu_Age alone cannot be a primary key as more than one students can have same age.
3. Attribute Stu_Id alone is a primary key as each student has a unique id that can identify the student record in the table.

Note: In some cases an attribute alone cannot uniquely identify a record in a table, in that case we try to find a set of attributes that can uniquely identify a row in table. We will see the example of it after this example.

Relational Integrity Constraints

Definition of Super Key in DBMS: A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table. Often **DBMS beginners** get confused between super key and **candidate key**, so we will also discuss candidate key and its relation with super key in this article.

Candidate Key Vs Super Key

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

Table: Employee

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

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

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

Relational Integrity Constraints

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

{Emp_SSN}

{Emp_Number}

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

Primary key

A Primary key is selected from a set of candidate keys. This is done by database admin or database designer.

We can say that either {Emp_SSN} or {Emp_Number} can be chosen as a primary key for the table Employee.

Alternate Key

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

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

Relational Integrity Constraints

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

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

Composite key Example

All of the following sets of super key are able to uniquely identify a row of the employee table.

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

Out of the above given keys, the following are the composite keys.

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

As all the keys are made up of more than one attributes.

Referential Integrity Constraints

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

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Referential Integrity Constraints

When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CSE
2	RAMESH	GURGAON	9652431543	18	CSE
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI	9136263161	18	IT

Branch

BRANCH_CODE	BRANCH_NAME
CSE	COMPUTER SCIENCE & ENGINEERING
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CE	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

Referential Integrity Constraints

An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.

Insertion Anomaly in Referencing Relation:

We can't insert a row in REFERENCING RELATION if referencing attribute's value is not present in referenced attribute value. e.g.; Insertion of a student with BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.

Deletion/Updation Anomaly in Referenced Relation:

We can't delete or update a row from REFERENCED RELATION if value of REFRENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE. e. g.; if we try to delete tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation. It can be handled by following method:

Referential Integrity Constraints

ON DELETE CASCADE: It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. e.g.;, if we delete a row from BRANCH with BRANCH_CODE 'CS', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.

ON UPDATE CASCADE: It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. e.g.;, if we update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.

Converting ER Diagram into Tables

- ER diagram is converted into the tables in relational model.
- This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.

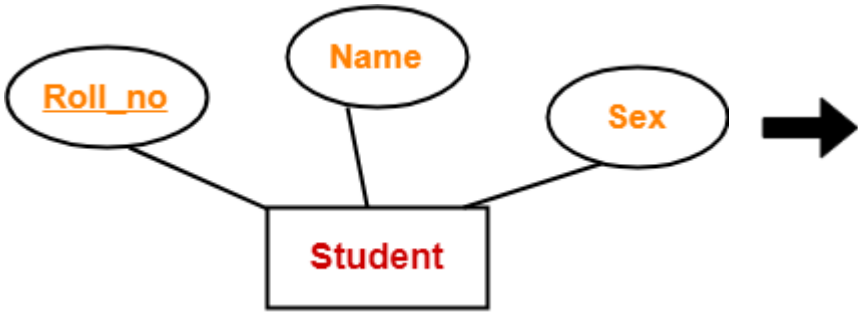
Following rules are used for converting an ER diagram into the tables-

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

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

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

Example:



Roll_no	Name	Sex

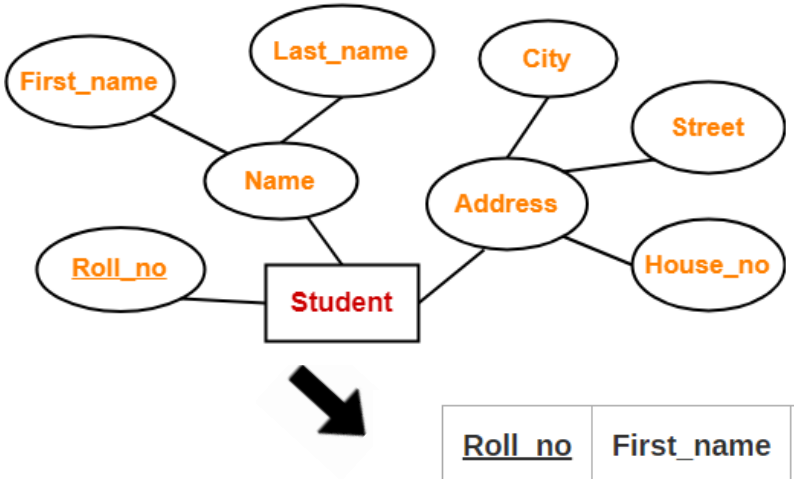
Schema : Student (Roll_no , Name , Sex)

Converting ER Diagram into Tables

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

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

Example-



Roll_no	First_name	Last_name	House_no	Street	City

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

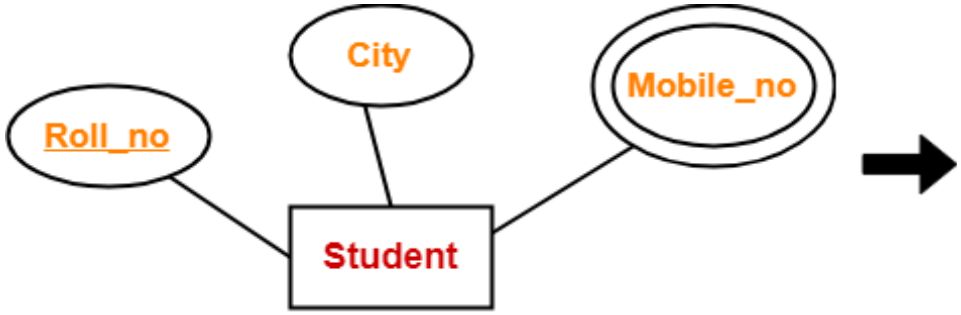
Converting ER Diagram into Tables

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

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

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

Example-



Roll_no	City

Roll_no	Mobile_no

Converting ER Diagram into Tables

Rule-04: Translating Relationship Set into a Table-

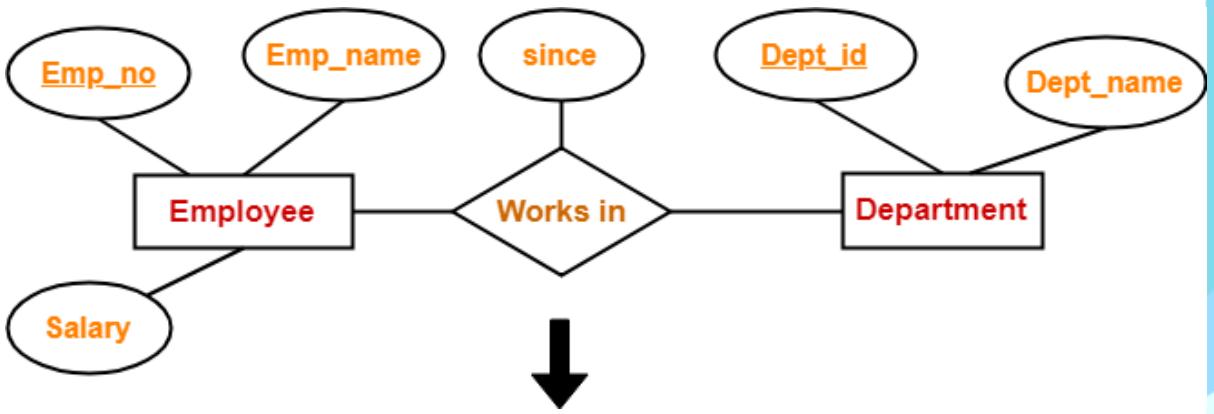
A relationship set will require one table in the relational model.

Attributes of the table are-

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

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

Example:



NOTE-

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

1. **One table for the entity set "Employee"**
2. **One table for the entity set "Department"**
3. **One table for the relationship set "Works in"**

Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

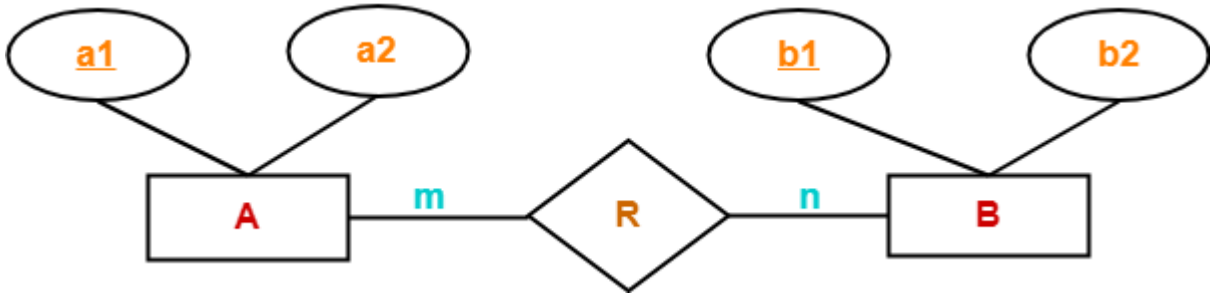
Converting ER Diagram into Tables

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

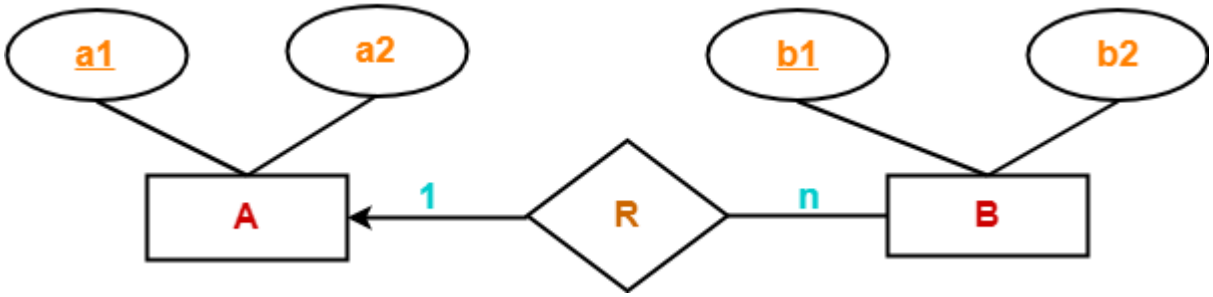


Here, three tables will be required-

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

Converting ER Diagram into Tables

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

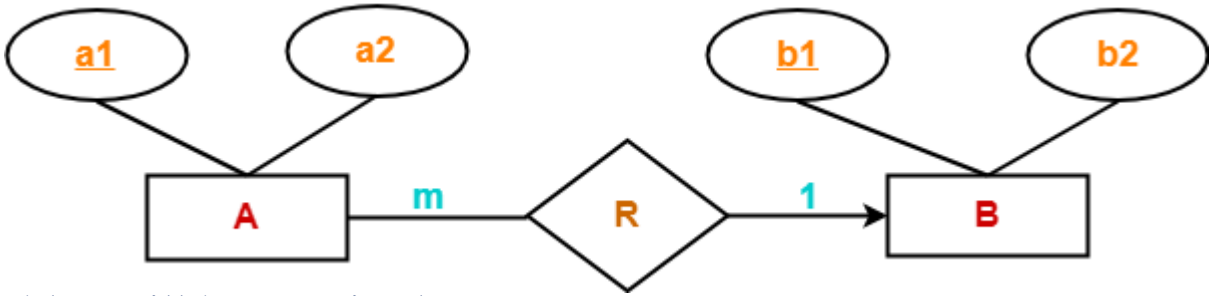


Here, two tables will be required-

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

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

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



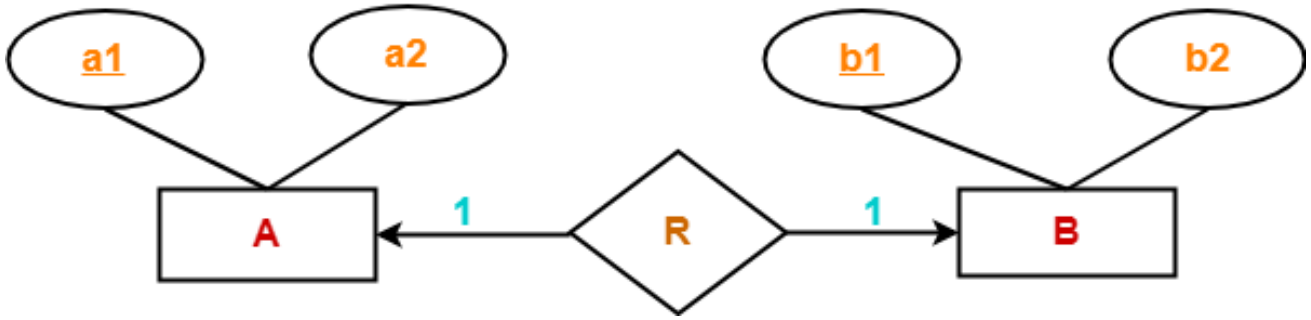
Here, two tables will be required-

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

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

Converting ER Diagram into Tables

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



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

Way-01:

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

Way-02:

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

Converting ER Diagram into Tables

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.

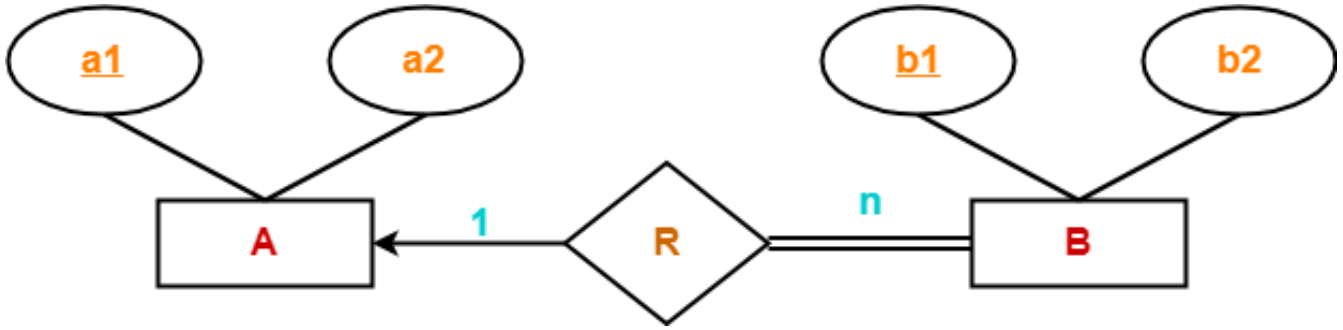
- For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets.

Converting ER Diagram into Tables

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

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

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



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

Then, two tables will be required-

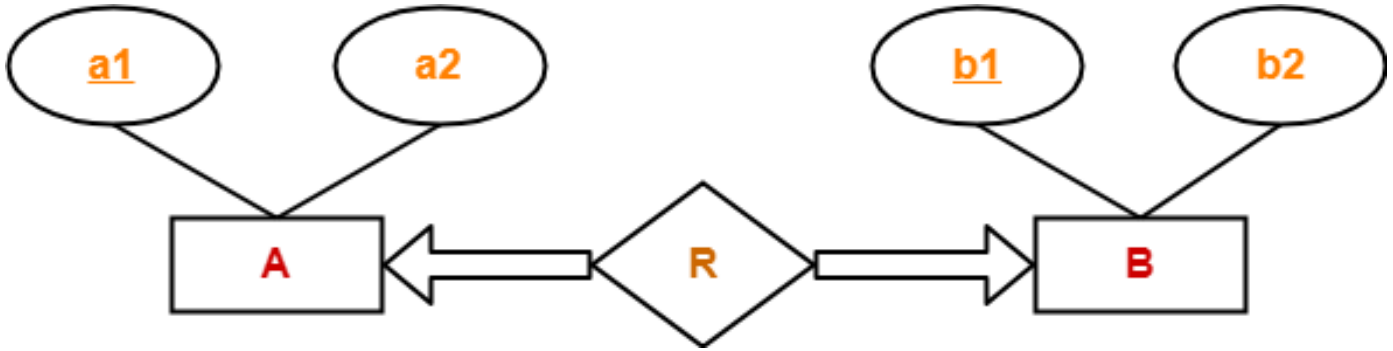
1. A (a1 , a2)
2. BR (a1 , b1 , b2)

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

Converting ER Diagram into Tables

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.



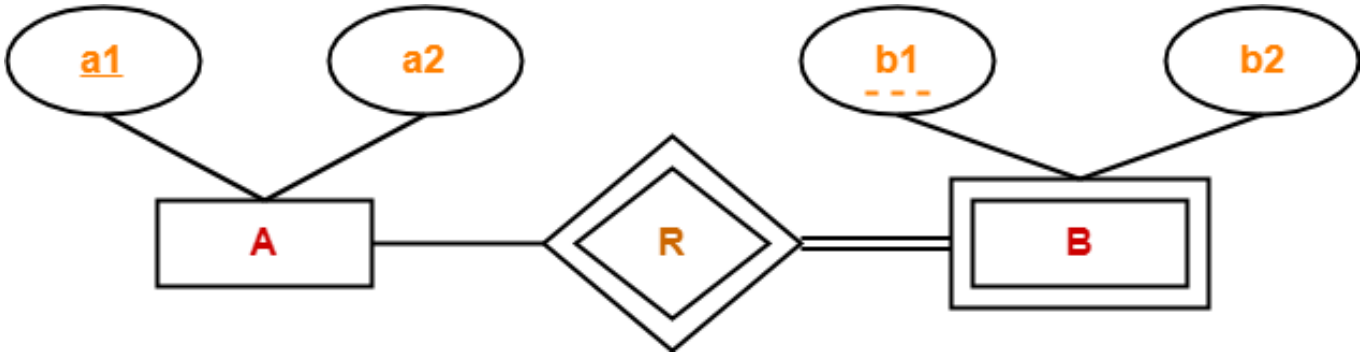
Here, Only one table is required.

1. ARB (a1 , a2 , b1 , b2)

Converting ER Diagram into Tables

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)