Relational Database Design

There are two approaches for designing any database, the top-down method and the bottom-up method.

- **Entity-Relationship Modelling** is a **top-down** approach :
  - The top-down method starts from the general and moves to the specific.
  - Top-down means you start with a set of requirements. Analyst intuitively identifying objects of importance about which a system is to store data (entities) from the requirement and then identifying the attributes which describe each of the entities and the relationships between the entities.
  - Requires detailed understanding of the system.

- **Normalization** is a **bottom-up** approach:
  - The bottom-up approach begins with the specific details and moves up to the general.
  - To begin, the system analyst will inspect all the interfaces that the system has, checking reports, screens, and forms. The analyst will work backwards through the system to determine what data should be stored in the database.
  - It starts with a collection of attributes or tables and attributes and organizes them into well-structured relations which are free from redundant data.
  - Typically this might happen where you have an existing database or data source, as is common in business intelligence or data integration projects.

Data modeling is a set of tools and techniques to understand and analyze how an organization should collect, update, and store data. It is often the first step in database design and object-oriented programming.

The outcome of data modeling is a **data model** which describing the data objects (entities) and their relationship. A complete model is a mechanism to answer any query. Traditionally, data models have been built during the analysis and design phases of a project to ensure that the requirements for a new application are fully understood.

In data modeling phase we generally create **Entity Relation Diagram.** It is a graphical representation for understanding and organizing the data independent of the actual database implementation. The ER model defines the conceptual view of a database.

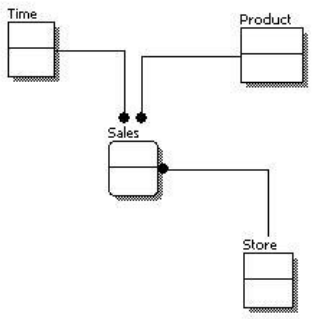
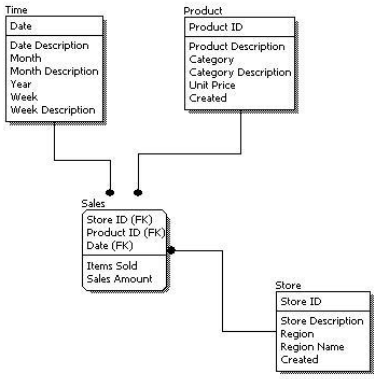
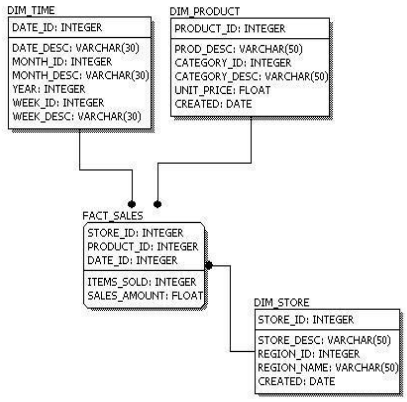
The **ER Model** was Proposed by Peter Chen (1976) and widely used to designing a database.

There is no standard for representing data objects in ER diagrams. Each modelling methodology uses its own notation and various notations are in place. Some of the widely used notations are :

- Information Engineering notations
- Chen notation
- Bachman notation
- Martin notation

As data model creation is a complex process, it is generally divided into multiple steps. During analysis phase we generally create conceptual and logical data model. During the design phase we generally create the physical data model through successive refinement of the model created in analysis phase and then create the actual database based on this model.

All the steps are briefly described below. Sometimes some of the steps may be merged.

Conceptual Data Model preparation	Identifies the highest-level relationships between the different entities (business entities). • Done in requirement analysis step • Only identify entities and relationship • No attribute/ primary key specified • Generally ER Model is used • Independent of specific DBMS features. Thus, the model is not invalidated, if a different DBMS is used later on.	
Logical Data Model preparation	A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. • It is refinement of conceptual model • Attribute, primary key, foreign key specified • Normalization occurs at this level. • Further enhancement of earlier ER model • The model is not invalidated, if a different DBMS is used later on.	
Physical Data Model preparation	A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. At this we also design indexes, table distribution, buffer size, etc., to maximize performance of the final system, considering expected work load. We also address security issues and enforce appropriate access control. • It is specific to RDBMS used. • ER diagram may be used. • DBA generally build the RDB based on the physical model.	

The table below compares the different features of above models :

Feature	Conceptual	Logical	Physical
Entity Names	√	√	
Entity Relationships	√	√	
Attributes		√	
Primary Keys		√	√
Foreign Keys		√	√
Table Names			√
Column Names			√
Column Data Types			√
Constraints			√

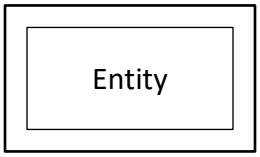
We can see that the complexity increases from conceptual to logical to physical.

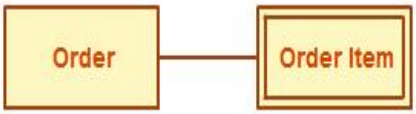
The **Database Design process** can be carried out in the following sequence:

- Modelling the data using an Entity-Relationship Diagram
- Mapping the Entity-Relationship Diagram to a **relational database design**
- Normalizing the **relational database design** to a non-redundant **relational database design**.

Some Useful Terms

We need to be familiar with the following terms to draw ER diagram. The terms and notations used for the respective terms are discussed in following table.

Entity 	• An entity can be a real-world object, either animate (living) or inanimate(nonliving), that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity. • We will collect data about these identified entities of the system. It is also known as **entity type**. • An **entity set** is a collection of similar types of entities. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties.
Weak entity 	A weak entity is an entity that depends on the existence of another entity. In more technical terms it can defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key. An entity like order item is a good example for this. The order item will be



EmpId	EmpName
123	Susil Pal
456	Bupen Hazarika
890	Papu Jadav

EmpId	DepId	DepName
123	1	Rahat
123	2	Chahat
456	1	Raju
456	2	Ruhi
456	3	Raja

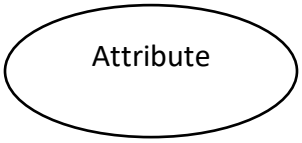
meaningless without an order so it depends on the existence of order.
In a parent/child relationship, a parent is considered as a **strong** entity and the child is a **weak** entity.

Let us consider another scenario, where we want to store the information of employees and their dependents. The every employee may have zero to n number of dependents. Every dependent has an **Depid** and **DepName**.

Now let us consider the data in these two table as shown in figure. Employee 123 has two dependent, Employee 456 has three dependents and Employee 890 has no dependent.

Now, in case of Dependent entity Depid cannot act as primary key because it is not unique. Thus, Dependent is a weak entity set having Depid as a **discriminator**. It has a total participation with the relationship "has" because no dependent can exist without the employees. In Employee table EmpId is primary key and in dependent table **EmpId + DepId** is primary key. As DepId in addition to EmpId is used to identify the dependent the column **DepId** is known as **discriminator**.

Attributes



Different notations are used to represent different attributes as shown below.

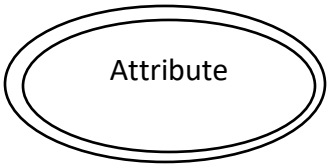
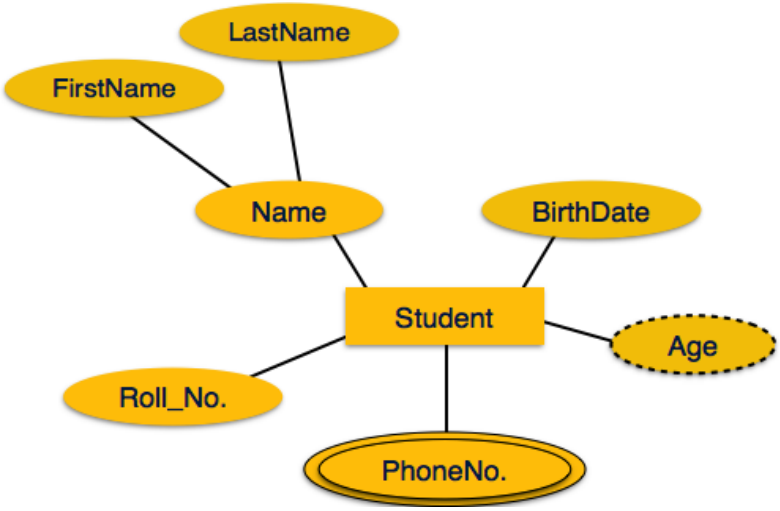
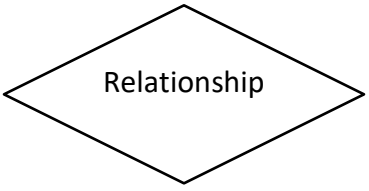
Properties/characteristics which describe entities or relationship are called attributes.

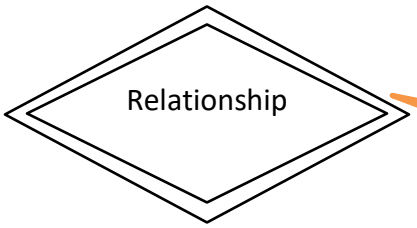
- The set of possible values that an attribute can take is called the **domain** of the attribute. For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}.
- All entities in a given entity set have same attributes.

Simple attribute : If an attribute cannot be divided into simpler components. Example : employee_id of an employee.

Composite attribute : If an attribute can be split into components, it is called a composite attribute. Example : Name of the employee which can be split into First_name, Middle_name, and Last_name.

Single valued Attributes : Can take only a single value for each entity instance, it is a single valued attribute. Example : age of a student.

Key attribute	• The attribute (or combination of attributes) which is unique for every entity instance is called **key** attribute. For each entity set we choose a key. There could be more than one **candidate** key, if so, we designate one of them as the **primary key**. ○ Super Key − A set of attributes (one or more) that collectively identifies an entity in an entity set. ○ Candidate Key − A minimal super key is called a candidate key. An entity set may have more than one candidate key. ○ Primary Key − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.
Multi-valued Attributes 	An attribute can take more than one value for each entity instance. Example : telephone number/ email id of an employee, a particular employee may have multiple telephone numbers or email id.
Derived Attribute 	An attribute which can be calculated or derived based on other attributes is a derived attribute. Example : age of employee which can be calculated from date of birth and current date. Example with different types of attributes: 
Relationships 	• Association between entities are called relationship and is represented by diamond-shaped box. • Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line. Example : An employee works for an organization. Here "works for" is a relation between the entities **employee** and **organization**.

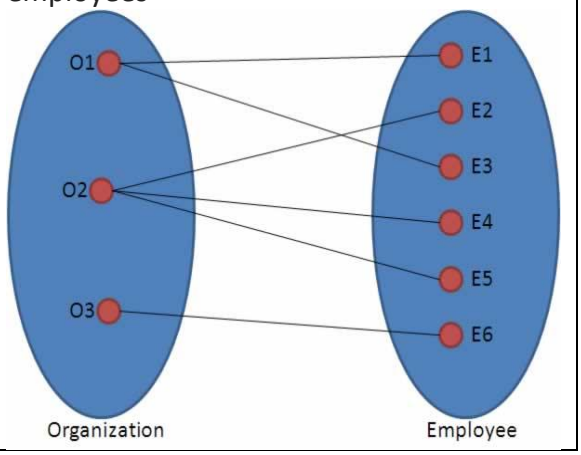
<table>
<tr>
<td> 
</td>
<td>However in ER Modeling, To connect a weak Entity with others, you should use a weak relationship notation as shown.

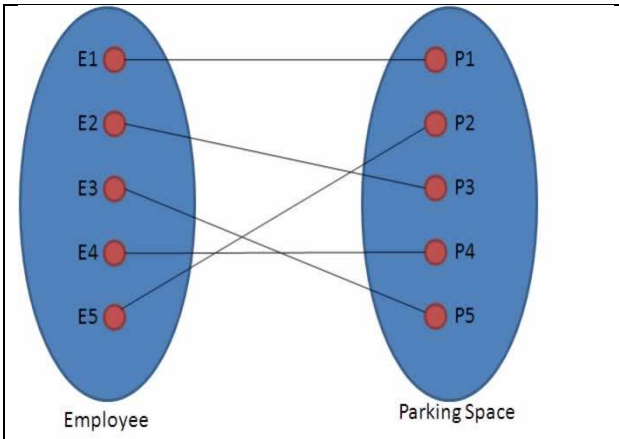
Degree of a Relationship
Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary , where the degree is 1, 2, and 3, respectively.
Example for unary relationship : An employee is a manager of another employee
Example for binary relationship : An employee works-for department.
Example for ternary relationship : customer purchase item from a shop keeper</td>
</tr>
<tr>
<td>Cardinality of a Relationship</td>
<td>Relationship cardinalities specify how many of each entity type is allowed. Four possible connectivity are as follows:

1. One to one (1:1) relationship
2. One to many (1:N) relationship
3. Many to one (M:1) relationship
4. Many to many (M:N) relationship

The minimum and maximum values of this connectivity is called the cardinality of the relationship</td>
</tr>
</table>

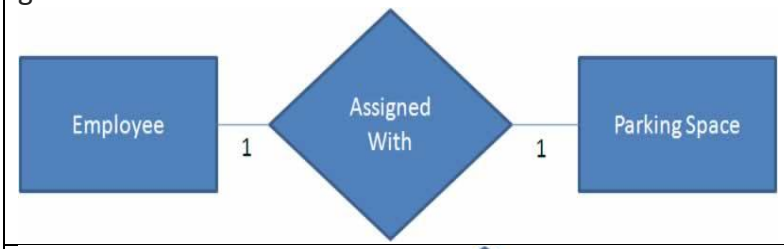
Example for Cardinality

<table>
<tr>
<td>One-to-One (1:1) - Employee is assigned with a parking space.</td>
<td>One-to-Many (1:N) - Organization has employees
 
</td>
</tr>
</table>



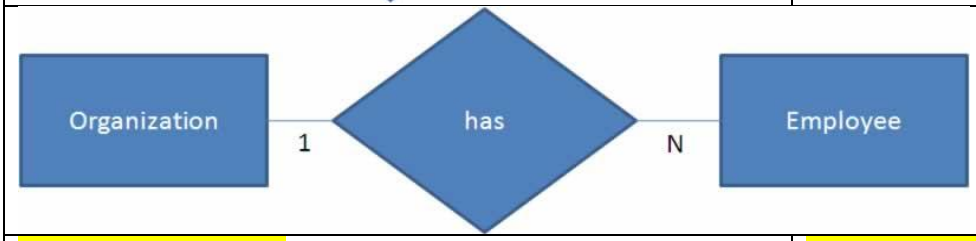
One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)

In ER modeling, this can be mentioned using notations as given below

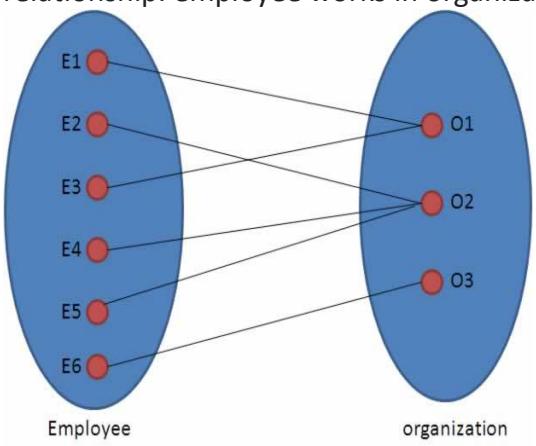


One organization can have many employees, but one employee works in only one organization. Hence it is a 1:N relationship and cardinality is One-To-Many (1:N)

In ER modeling, this can be mentioned using notations as given below



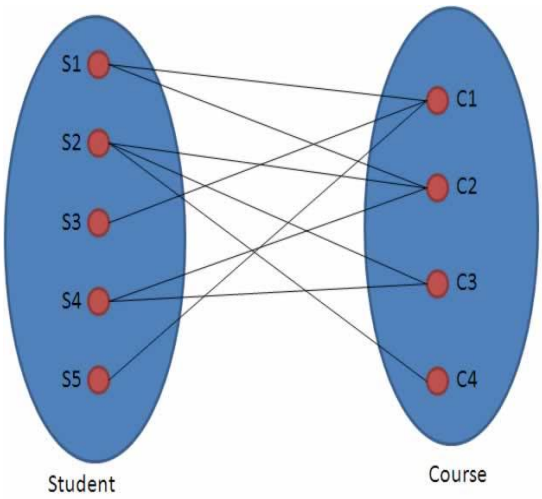
Many-to-One (M :1) - It is the reverse of the One to Many relationship. employee works in organization

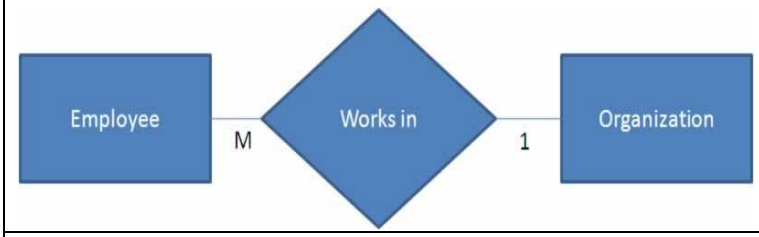
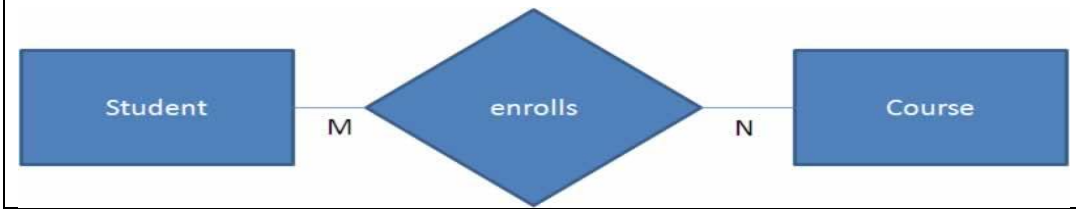


One employee works in only one organization But one organization can have many employees. Hence it is a M:1

Many-to-Many (M:N) -

Students enrolls for courses

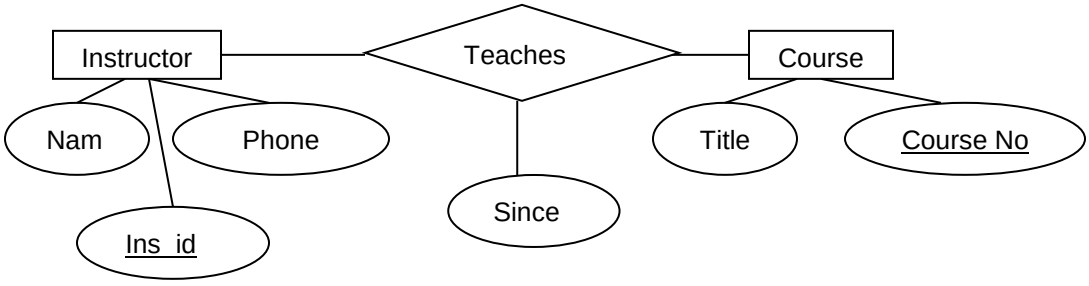


relationship and cardinality is Many-to-One (M :1) In ER modeling, this can be mentioned using notations as given below.  	One student can enroll for many courses and one course can be enrolled by many students. Hence it is a M:N relationship and cardinality is Many-to-Many (M:N) In ER modeling, this can be mentioned using notations as given below
 	

Relationship with descriptive attributes

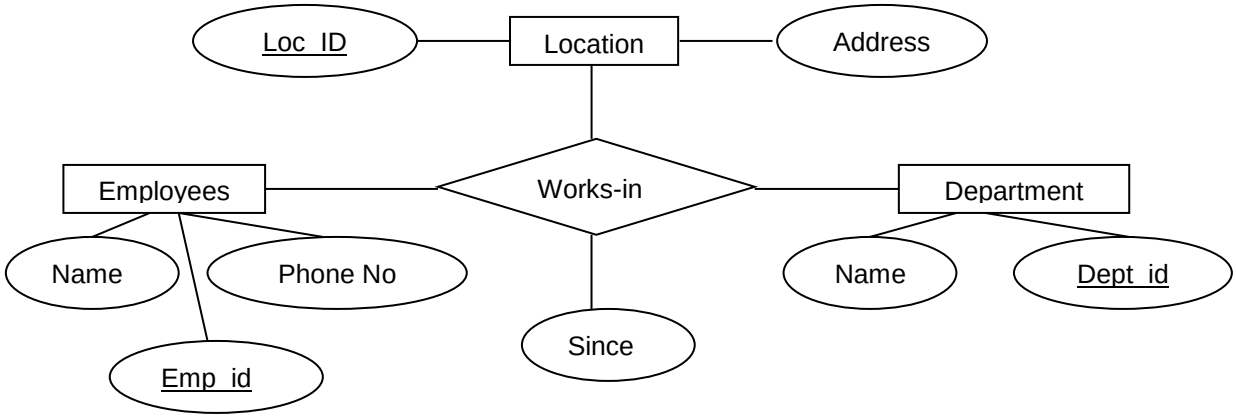
A relationship can also have **descriptive attributes**.

In the following ER diagram : Instructors teach courses. Instructors have a name, phone number and Instructor ID (primary key). Courses are numbered and have titles. The relationship Teaches must be uniquely identified by the participating entities (Ins_id + Course no), without referencing the descriptive attribute. Thus for a given instructor-course pair, we cannot have more than one associated **since** value.

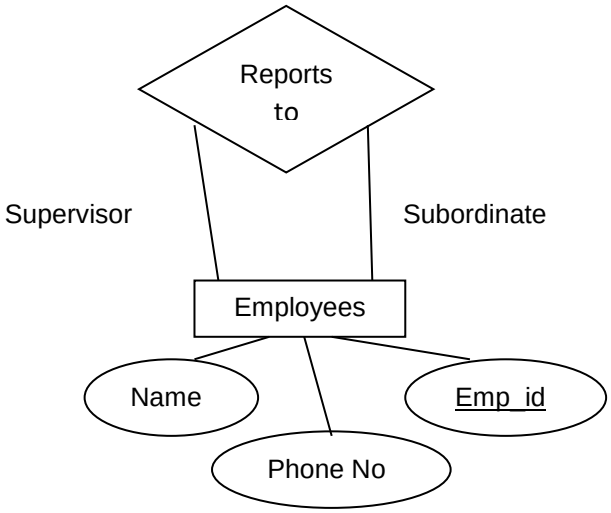


Example of Ternary Relationship

Suppose that each department has offices in several locations and we want to record the locations at which each employee works. This is an example of **ternary** relationship because we must record an association between an employee, a department, and a location.

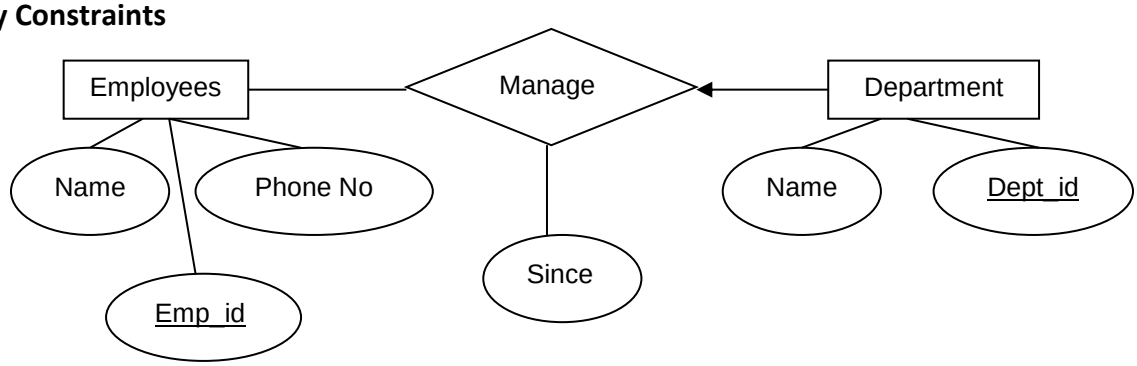


Example of Role Indicator

Here two entities from the same entity set are involved in the relationship, since employees report to other employees. However they will play different role, which is reflected in the **role indicator** supervisor and subordinate. In relationship set of the relation Report_to, the attribute Emp_id of both the entities will be involved by changing the attribute name suitably (generally role indicator is concatenated e.g. Supervisor_Emp_id and Subordinate_Emp_id	

Additional Features

Key Constraints



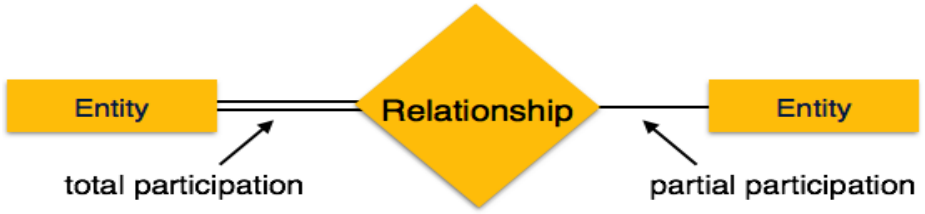
<table>
<tr>
<td>Employees manages departments such that
1. Each department has at most one manager – It is an example of key constraint. It is indicated by arrow. It indicates that we can uniquely determine the managers, if we know the department id from the relationship set.
2. Single employee is allowed to manage more than one departments.

A relationship like this is called one-to-many relationship.</td>
<td>Dept Id Emp Id Since
001 501 3/4/09
002 504 9/5/08
003 501 10/2/10

If we impose the restriction – one employee is allowed to manage only one department (adding arrow from employee to manages) then it will be a one-to-one relationship.</td>
</tr>
<tr>
<td>In contrast, in the Works In relationship set, an employee is allowed to work in several departments and a department is allowed to have several employees – it is an example of many-to-many relationship.</td>
<td>Dept Id Emp Id Since
001 501 3/4/09
002 504 9/5/08
002 501 10/2/10
004 501 10/2/08
005 501 8/8/09</td>
</tr>
</table>

Participant Constraints

- Total Participation – Each entity is involved in the relationship. Total participation is represented by double lines.



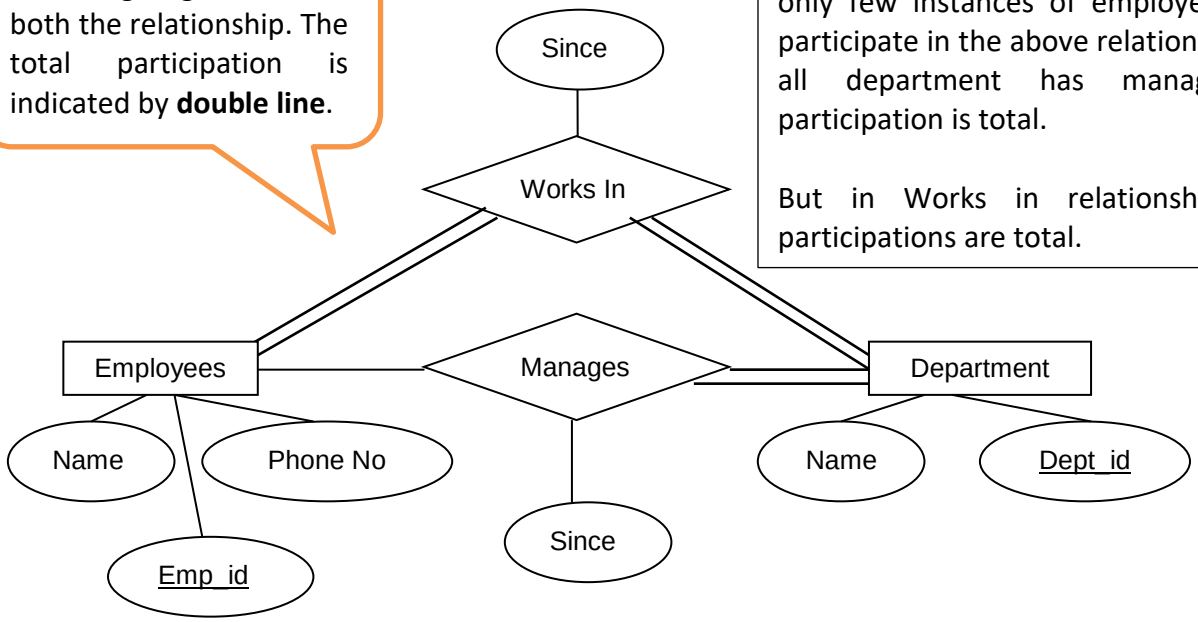
- Partial participation – Not all entities are involved in the relationship. Partial participation is represented by single lines.

Say every department is required to have a manager. In this case all the department entities in department set must appear in the manages relationship. This is an example of **participation constraint** and the participation of department entity set is said to be **total**. A participation that is not total is said to be partial. Example : participation of Employees entity set in manages is partial, since not every employee gets to manage a department.

Following figure shows both the relationship. The total participation is indicated by **double line**.

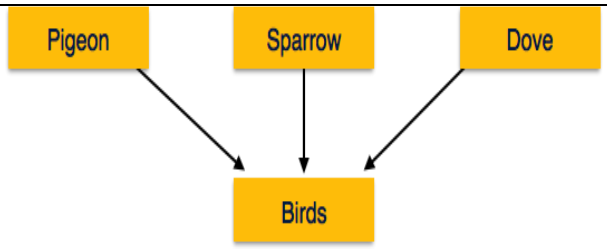
All employees will not be the manager of the department (only one manager). So only few instances of employee entity participate in the above relationship. But all department has manager, so participation is total.

But in Works in relationship both participations are total.



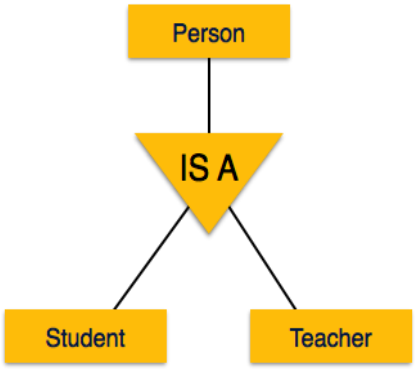
Generalization Aggregation

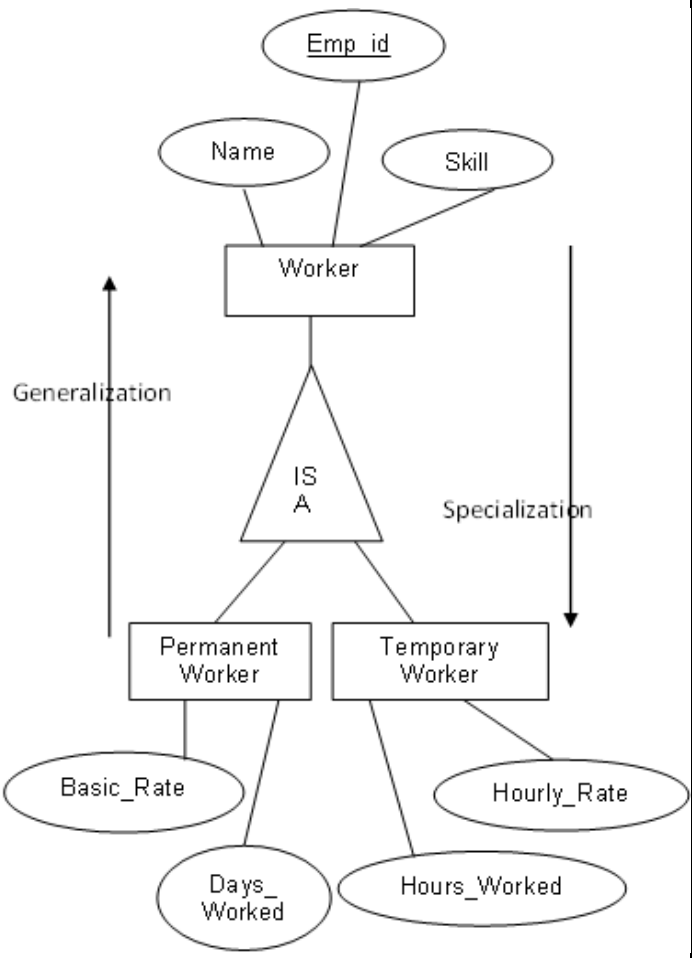
The ER Model has the power of expressing database entities in a conceptual hierarchical manner. Going up in this structure is called **generalization**, where entities are clubbed together to represent a more generalized view. For example, a particular student named Mira can be generalized along with all the students. The entity shall be a student, and further, the student is a person. The reverse is called **specialization** where a person is a student, and that student is Mira.

Generalization In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.	

Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. In a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.



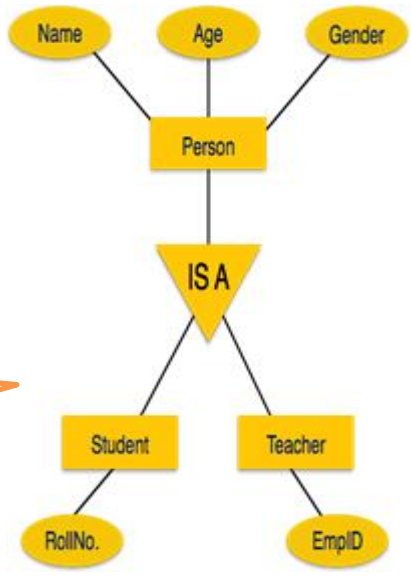


Inheritance

We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as abstraction.

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.

For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.



Entity Relationship (ER) Modeling

Example - 1

We are going to design an Entity Relationship (ER) model for a college database. Say we have the following requirement statements.

1. A college contains many departments
2. Each department can offer any number of courses
3. Many instructors can work in a department
4. An instructor can work only in one department
5. For each department there is a Head
6. An instructor can be head of only one department
7. Each instructor can take any number of courses
8. A course can be taken by only one instructor
9. A student can enroll for any number of courses
10. Each course can have any number of students

Step 1 : Identify the Entities

From the statements given, the entities are **Department, Course, Instructor, Student**

Step 2 : Identify the relationships

- One department offers many courses. But one particular course can be offered by only one department. Hence the cardinality between department and course is One to Many (1:N)
- One department has multiple instructors. But instructor belongs to only one department. Hence the cardinality between department and instructor is One to Many (1:N)
- One department has only one head and one head can be the head of only one department. Hence the cardinality is one to one. (1:1)
- One course can be enrolled by many students and one student can enroll for many courses. Hence the cardinality between course and student is Many to Many (M:N)
- One course is taught by only one instructor. But one instructor teaches many courses. Hence the cardinality between course and instructor is Many to One (N :1)

Step 3: Identify the key attributes

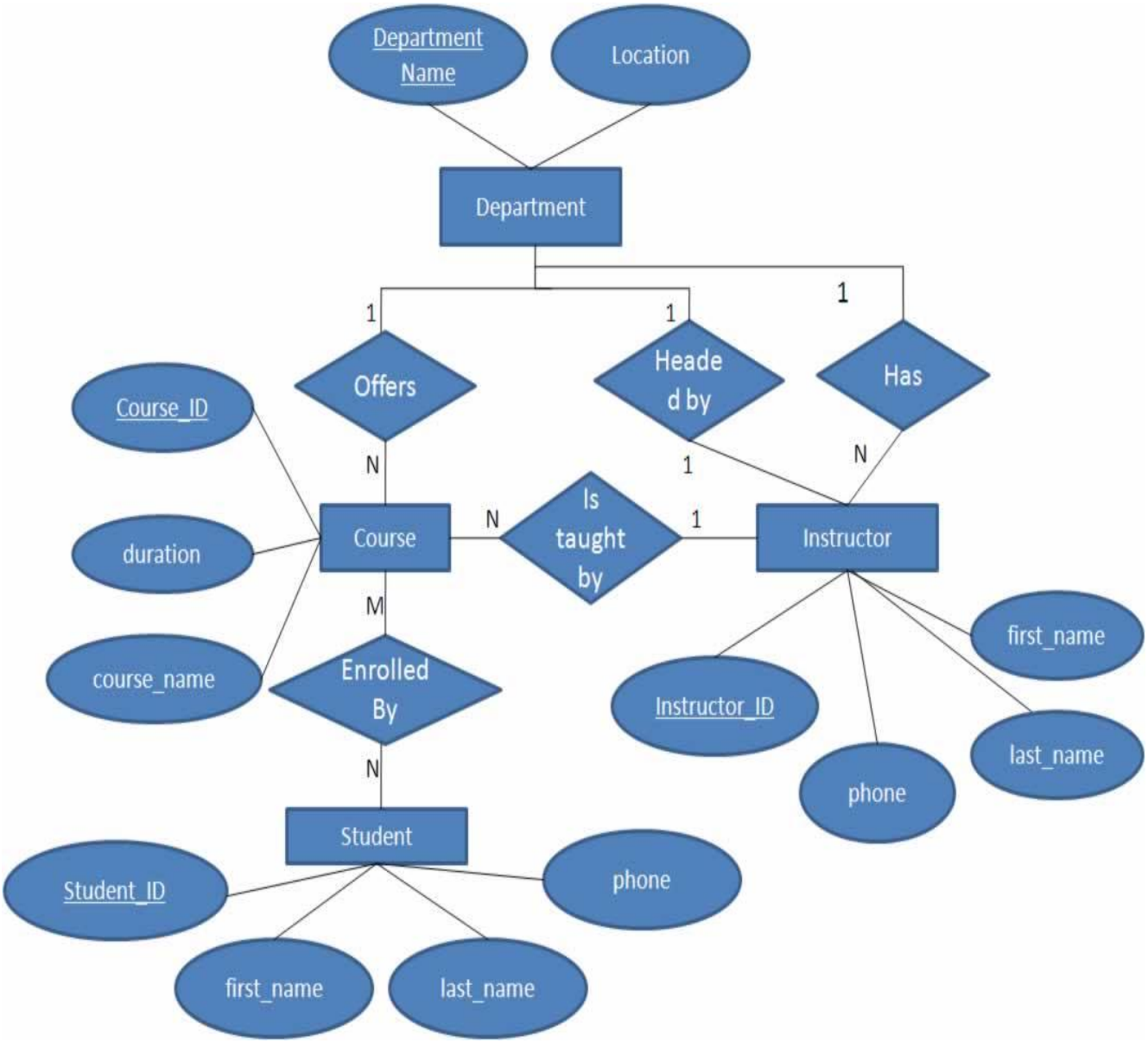
- "Departmen_Name" can identify a department uniquely. Hence Department_Name is the key attribute for the Entity "Department".
- Course_ID is the key attribute for "Course" Entity.
- Student_ID is the key attribute for "Student" Entity.
- Instructor_ID is the key attribute for "Instructor" Entity.

Step 4: Identify other relevant attributes

- For the department entity, other attributes are location
- For course entity, other attributes are course_name,duration
- For instructor entity, other attributes are first_name, last_name, phone
- For student entity, first_name, last_name, phone

Step 5: Draw complete ER diagram

By connecting all these details, we can now draw ER diagram as given below.



Example - 2

The music database is designed to store details of a music collection, including the albums in the collection, the artists who made them, the tracks on the albums, and when each track was last played.

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.

List of requirements:

- 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.

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.

Step 1 : Identify the Entities

From the statements given, the entities are **Artists, albums, Tracks and Played**.

Step 2 : Identify the relationships

- One artist can make many albums
- One album can contain many tracks
- 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.

Step 3: Identify the key attributes

- The only strong entity in the database is Artist, which has an artist_id attribute that uniquely identifies it.
- Each Album entity is uniquely identified by its album_id combined with the artist_id of the corresponding Artist entity.

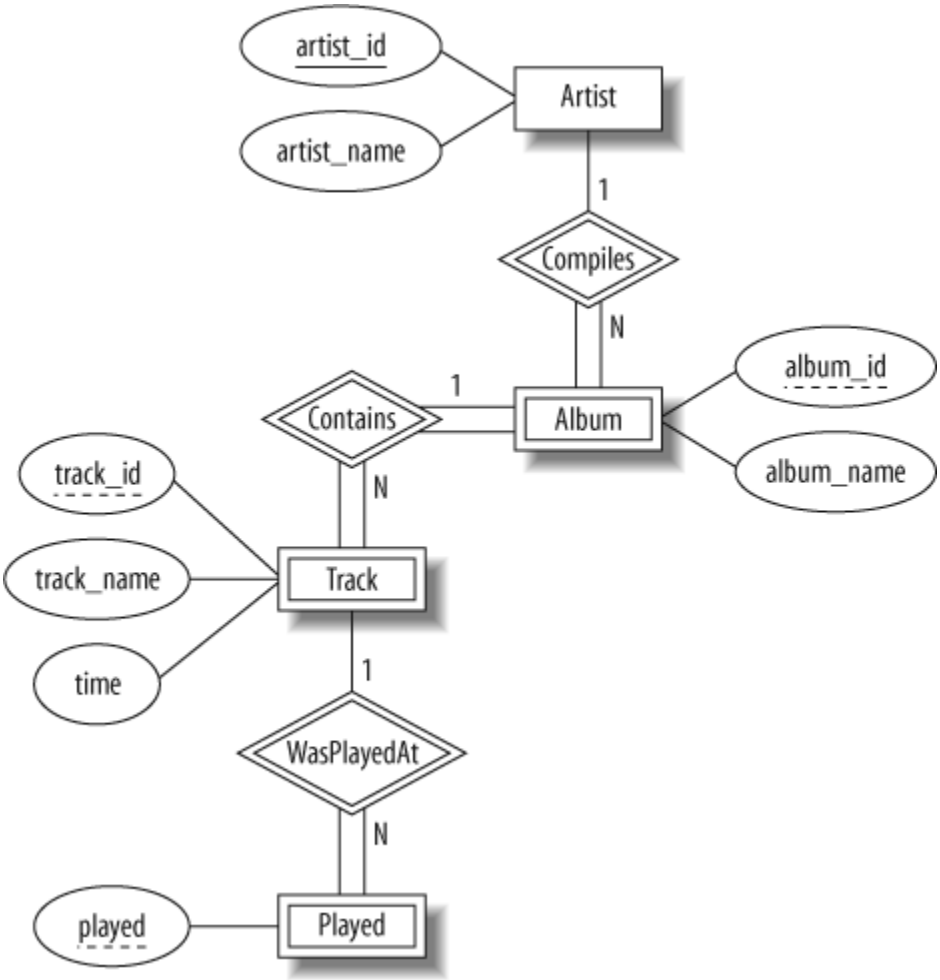
- A Track entity is similarly uniquely identified by its track_id combined with the related album_id and artist_id attributes.
- The Played entity is uniquely identified by a combination of its played time, and the related track_id, album_id, and artist_id attributes.

Step 4: Identify other relevant attributes

- 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.

Step 5: Draw complete ER diagram

By connecting all these details, we can now draw ER diagram as given below.



What it doesn't do

- We've kept the music database simple because adding extra features doesn't help you learn anything new, it just makes the explanations longer. If you wanted to use the music database in practice, then you might consider adding the following features:
- Support for compilations or various-artists albums, where each track may be by a different artist and may then have its own associated album-like details such as a recording date and time. Under this model, the album would be a strong entity, with many-to-many relationships between artists and albums.
- Playlists, a user-controlled collection of tracks. For example, you might create a playlist of your favorite tracks from an artist.
- Track ratings, to record your opinion on how good a track is.
- Source details, such as when you bought an album, what media it came on, how much you paid, and so on.
- Album details, such as when and where it was recorded, the producer and label, the band members or sidemen who played on the album, and even its artwork.
- Smarter track management, such as modeling that allows the same track to appear on many albums.

Example - 3

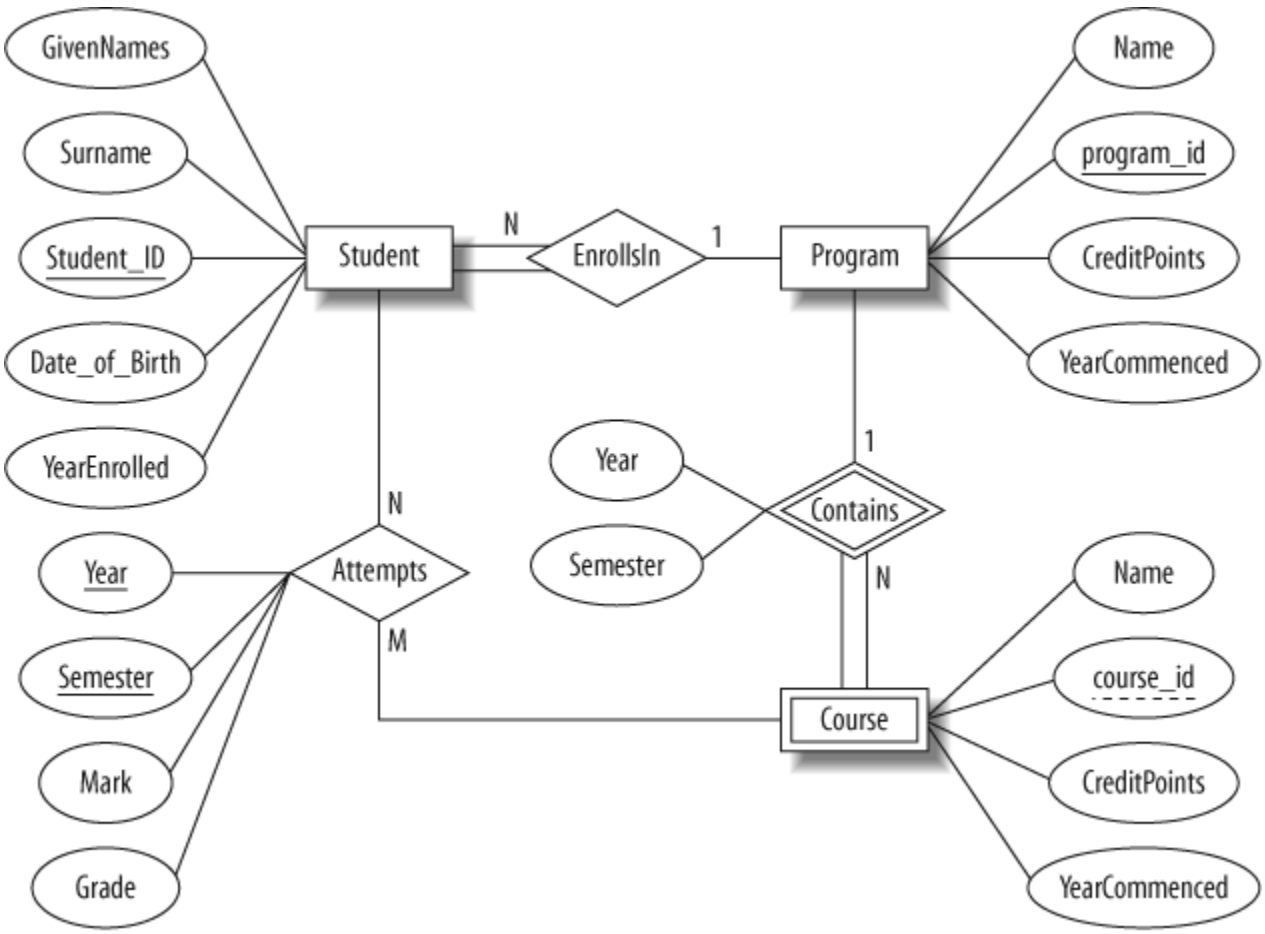
The **university database** stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in. The database is a long way from one that'd be suitable for a large tertiary 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.

List of Requirements

- 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.
- 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).

By analyzing above requirements following are our findings:

- 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 EnrollsIn 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.



What it doesn't do

Our database design is rather simple, but this is because the requirements are simple. For a real university, many more aspects would need to be captured by the database. For example, the requirements don't mention anything about campus, study mode, course prerequisites, lecturers, timetabling details, address history, financials, or assessment details. The database also doesn't allow a student to be in more than one degree program, nor does it allow a course to appear as part of different programs.

Example - 4

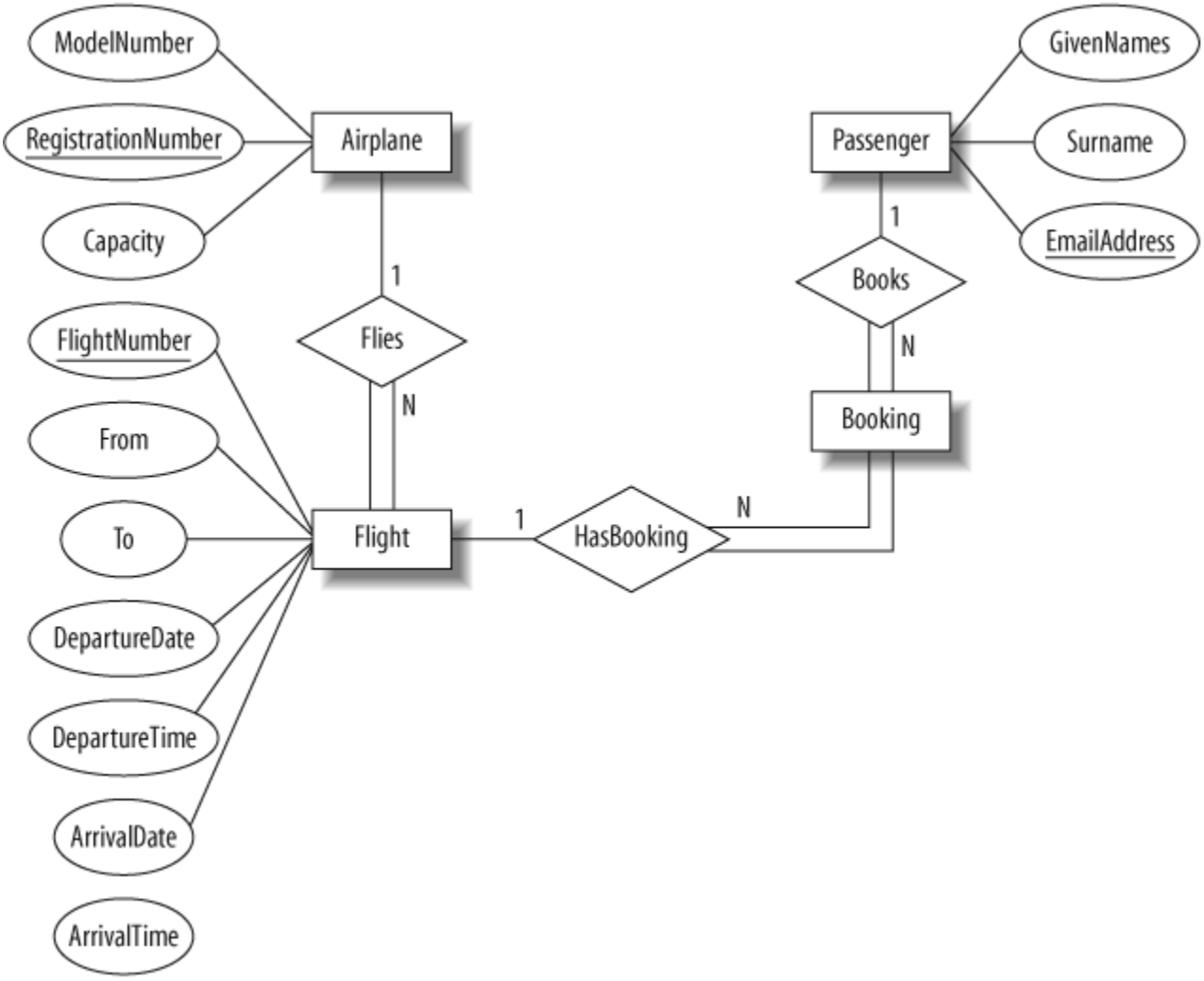
The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same.

List of Requirements:

- 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.

After analyzing above requirement following are our findings:

- 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. As discussed earlier in Intermediate Entities," 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. Note that even if we didn't notice this hidden entity, it would come out as part of the ER-to-tables mapping process we'll describe next in Using the Entity Relationship Model."



What it doesn't do

- Again, this is a very simple flight database. There are no requirements to capture passenger details such as age, gender, or frequent-flier number.
- We've treated the capacity of the airplane as an attribute of an individual airplane. If, instead, we assumed that the capacity is determined by the model number, we would have created a new AirplaneModel entity with the attributes ModelNumber and Capacity. The Airplane entity would then not have a Capacity attribute.
- We've mapped a different flight number to each flight between two destinations. Airlines typically use a flight number to identify a given flight path and schedule, and they specify the date of the flight independently of the flight number. For example, there is one IR655 flight on April 1, another on April 2, and so on. Different airplanes can operate on the same flight number over time; our model would need to be extended to support this.
- The system also assumes that each leg of a multihop flight has a different FlightNumber. This means that a flight from Dubai to Christchurch via Singapore and Melbourne would need a different FlightNumber for the Dubai-Singapore, Singapore-Melbourne, and Melbourne-Christchurch legs.

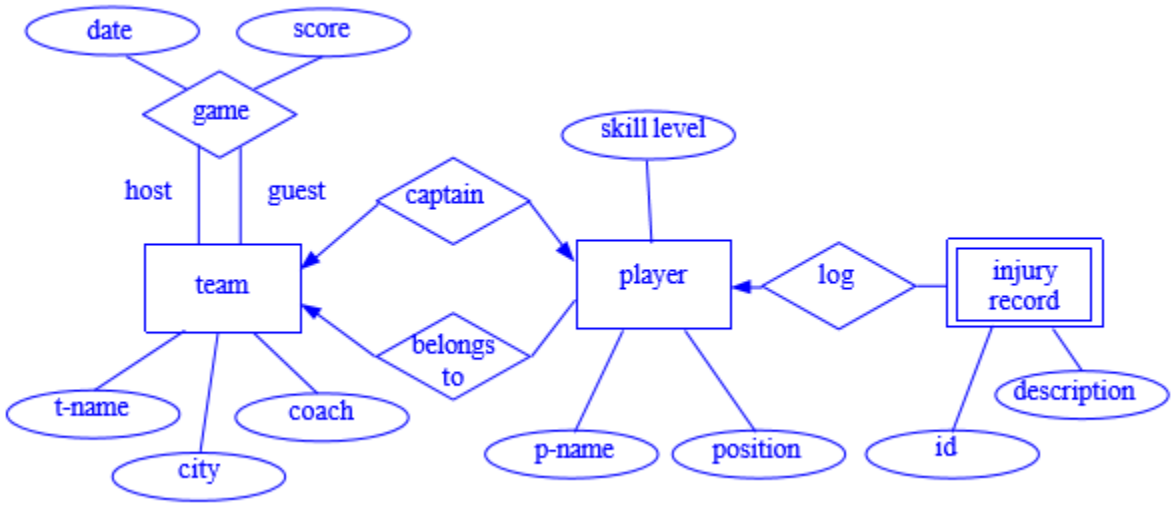
- Our database also has limited ability to describe airports. In practice, each airport has a name, such as “Melbourne Regional Airport,” “Mehrabad,” or “Tullamarine.” The name can be used to differentiate between airports, but most passengers will just use the name of the town or city. This can lead to confusion, when, for example, a passenger could book a flight to Melbourne, Florida, USA, instead of Melbourne, Victoria, Australia. To avoid such problems, the International Air Transport Association (IATA) assigns a unique airport code to each airport; the airport code for Melbourne, Florida, USA is MLB, while the code for Melbourne, Victoria, Australia is MEL. If we were to model the airport as a separate entity, we could use the IATA-assigned airport code as the primary key. Incidentally, there’s an alternative set of airport codes assigned by the International Civil Aviation Organization (ICAO); under this code, Melbourne, Florida is KMLB, and Melbourne, Australia is YMML.

Example - 6

Suppose you are given the following requirements for developing a simple database for a league conducted by All India Football Association (IFAL) :

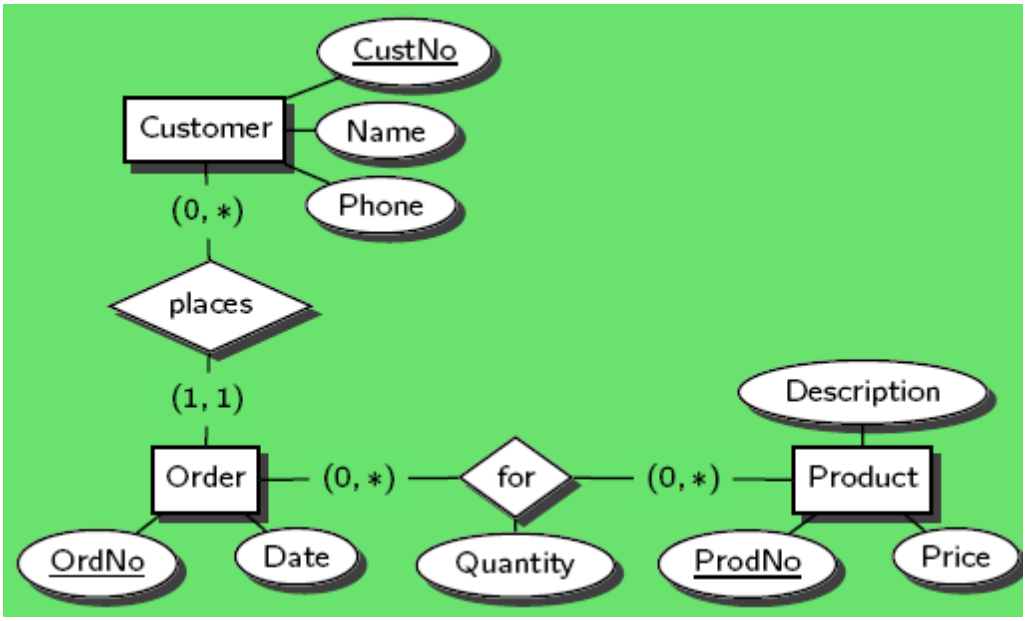
- The IFAL has many teams
- Each team has a name, a city, a coach, a captain, and a set of players
- Each player belongs to only one team
- Each player has a name, a position (such as left wing or goalie), a skill level, and a set of injury records
- A team captain is also a player
- A game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 1999) and a score (such as 4 to 2).

Construct a clean and concise ER diagram for the IFAL database using the Chen notation. List your assumptions and clearly indicate week entity, the cardinality mappings as well as any role indicators in your ER diagram.



Translation into the Relational Model

- Suppose we have prepared a simplified ER model for an order placement system. Following steps will guide you to translate this ER model into the Relational model.



1. Transforming an ER entity E:

   - Create a table for each entity - The table name is E (conventionally: E + 's').
   - The columns of this table are the attributes of the entity type.
   - The primary key of the table is the primary key of the entity type. If E's key is composite, so will be the relational key. If E has no key, add an artifical key to the table.

Customers

CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

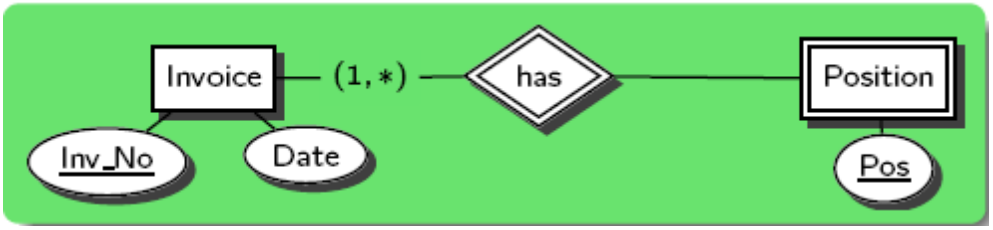
Orders

OrdNo	Date
200	2/15/04
201	2/16/04

Products

ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

2. Transforming Weak Entities



- When a weak entity is translated, the key attributes of the owner entity are added as a key and foreign key. So we have to create another table position (invoice line item) with key as **Inv_No + Pos** . Inv_No is the foreign key.
- This automatically implements the relationship. It makes sense to specify DELETE CASCADES for the foreign key: if an invoice is deleted, all its positions will be removed from the DB state, too.

3. Transforming a relationship R - **One-To-Many Relationships**

- In above example : Customer–(0, *)–places–(1, 1)–Order, “one customer places many orders.” It is a one-to-many relationship
- In this case, add the key of the “one” side as a column to the “many” table to implement R.
- This column will be a foreign key referencing a row in the table representing the related entity.

Orders (OrdNo, Date, CustNo → Customers)

Orders

OrdNo	Date	CustNo
200	2/15/04	11
201	2/16/04	11

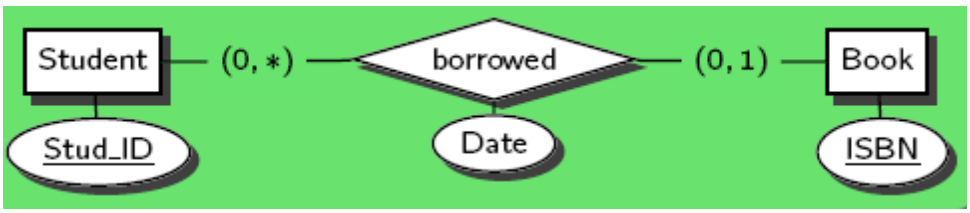
Customers

CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

- Convention: use relationship and role to name foreign key column:
  Orders (OrdNo, Date, placed by → Customers)
- If the minimum cardinality is 1 on the “many” side (see example), null values are not allowed in the foreign key column (column placed by in example).
- If the minimum cardinality is 0, null values are allowed in the foreign key column.
- The foreign key is null for those entities that do not participate in R at all.

Transforming Relationship Attributes

- To transform one-to-many relationship attribute(s), e.g.



store the relationship attribute(s) together with the reference to the related entity, e.g.

Books (ISBN, ..., borrowed_by → Students, Date)

- Alternatively One-to-many relationships R with cardinality (0,1) can be translated into a table of their own:

borrowed_by (ISBN→Books, Stud_ID→Students, Date)

- The extra table holds the key values of the related entities plus the relationship attributes. The key attributes of the side with the (0,1) cardinality become the key of this relation. "Each book can be borrowed only once at the same time."

4. Transforming a relationship R - Many-To-Many Relationships

- If R has maximum cardinality * on both sides, R is many-to-many. Example: **Order–(1, *)–for–(0, *)–Product**, "an order contains many products, a product may be part of many orders."
- R becomes its own table. The columns of this table are the keys of both participating entity types.
- These columns act as foreign keys referencing the entities and, at the same time, together form a composite key for the extra table.
- Relationship attributes are added as columns to the table representing R.
  for (OrdNo →Orders, ProdNo →Products, Quantity)

for (OrdNo → Orders, ProdNo → Products, Quantity)

for

OrdNo	ProdNo	Quantity
200	1	1
200	2	1
201	1	5

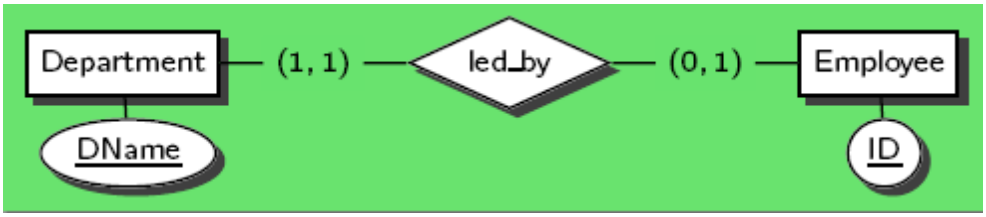
Orders

OrdNo	Date	CustNo
200	2/15/04	11
201	2/16/04	11

Products

ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

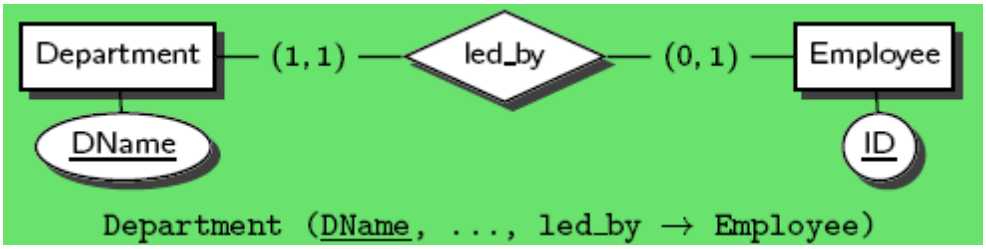
5. Transforming a relationship R - **One-To-One Relationships**



- If R has maximum cardinality 1 on both sides, R is one-to-one.

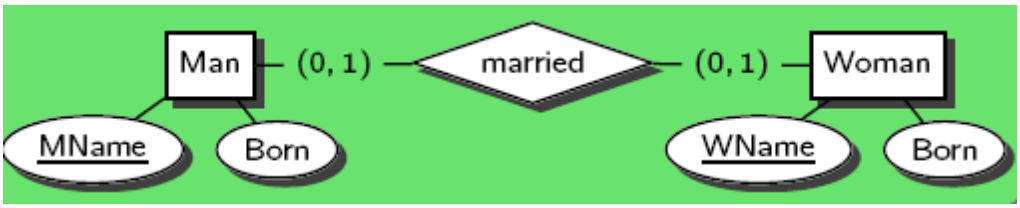
- We can essentially transform as if R were one-to-many, but additional key constraints are generated.
- To which entity table shall we add the led by attribute to represent the relationship?

- Since we have Department–(1, 1)–led by ("every department is led by exactly one employee"), it makes sense to host the relationship in the Department table:
  Department (DName, ..., led by →Employee)
- We may declare the foreign key led by as NOT NULL. (This is not possible if led by is hosted in the Employee table.)



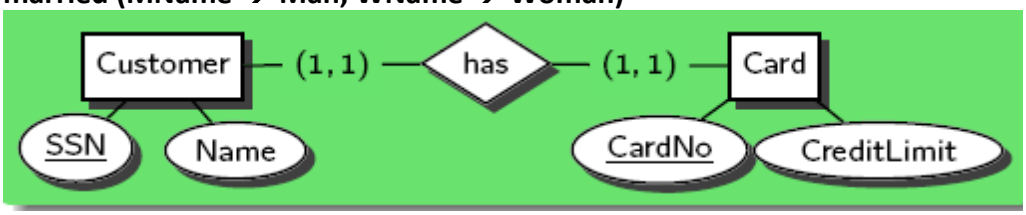
Note: led by now also is a key for the Department table. led by is a key for table Department. This key constraint enforces the maximum cardinality of 1 (on the Employee side).

One-To-One Relationships – Some variation



Two variations possible

- Any of the two (not both!) entity tables may host the married foreign key (null values allowed).
- Translate the relationship into a table of its own:
  married (MName → Man, WName → Woman)



- In order to enforce the minimum cardinality 1 on both sides, the entity tables need to be merged: **CustomerCard** (SSN, Name, CardNo, CreditLimit)
- No null values are allowed.
- Both, SSN and CardNo, are keys of this table. One is selected as primary key, the other is an secondary key.