

Entity, Entity Set and Entity Type

An *entity set* is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. For example, the entity is EMPLOYEE.

EmployeeID	FirstName	LastName	DateOfBirth	Gender	Position	Department
1	John	Doe	1985-06-15	Male	Software Dev	IT
2	Jane	Smith	1990-09-20	Female	Data Analyst	IT
3	Mike	Brown	1988-12-05	Male	HR Manager	Human Resources
4	Emily	Davis	1992-02-10	Female	Accountant	Finance
5	David	Wilson	1983-11-25	Male	Project Manager	Operations



Entity – Notation

- **Entity Occurrence:** A uniquely identifiable object of an entity type.
- **Entity Instance:** Values associated with the entities. (rows in the table at a given point in time)

Notation: *rectangle* labelled with the name of the entity.

Notation for an Entity



Identify the Entity & entity sets from the data below;

Movie

MovieID	Title	ReleaseDate	Duration	Rating	GenreID	DirectorID
1	Inception	2010-07-16	148	8.8	1	1
2	The Dark Knight	2008-07-18	152	9.0	2	1
3	Interstellar	2014-11-07	169	8.6	1	1
4	Titanic	1997-12-19	195	7.8	3	2

Actor

ActorID	FirstName	LastName	DateOfBirth	Nationality
1	Leonardo	DiCaprio	1974-11-11	American
2	Joseph	Gordon-Levitt	1981-02-17	American
3	Ellen	Page	1987-02-21	Canadian
4	Christian	Bale	1974-01-30	British
5	Kate	Winslet	1975-10-05	British

Kinds of Entities

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities. These are described below.

Types of Entities

Regular Entities

- Called as Strong Entities.
- Independent Entities
- E.g. Employee

Weak Entities

- Its existence depends on some other entity
- E.g. EmployeeDependants
- If an Employee Instance is deleted, then the dependants too are deleted. They do not exist without the employee.

Notation for a Regular Entity

Employee

Notation for a Weak Entity

EmpDependant

Regular and Weak Entities

Example Data:

Student Table:

StudentID	FirstName	LastName	DateOfBirth	Email
1	John	Doe	2000-01-15	john.doe@example.com
2	Jane	Smith	1999-03-22	jane.smith@example.com

Course Table:

CourseID	CourseName	Credits	Department
101	Database Systems	4	Computer Science
102	Calculus I	3	Mathematics

Enrollment Table:

StudentID	CourseID	EnrollmentDate	Grade
1	101	2023-09-01	A
2	101	2023-09-01	B
1	102	2023-09-01	A-

Entities - Examples

Employees

Dependents

Faculty

Evaluations



Knowledge Check - Entities



<https://forms.office.com/e/KGnQsAxqmt>

Kinds of Entities

Independent entities

Independent entities, also referred to as **kernels**, are the backbone of the database. They are what other tables are based on. **Kernels** have the following characteristics:

- They are the building blocks of a database.
- The primary key may be simple or composite.
- The primary key is not a foreign key.
- They do not depend on another entity for their existence.

If we refer back to a database, examples of an independent entity include the **Customer table**, **Employee table** or **Product table**.

Kinds of Entities

Dependent entities

Dependent entities, also referred to as *derived entities*, depend on other tables for their meaning. These entities have the following characteristics:

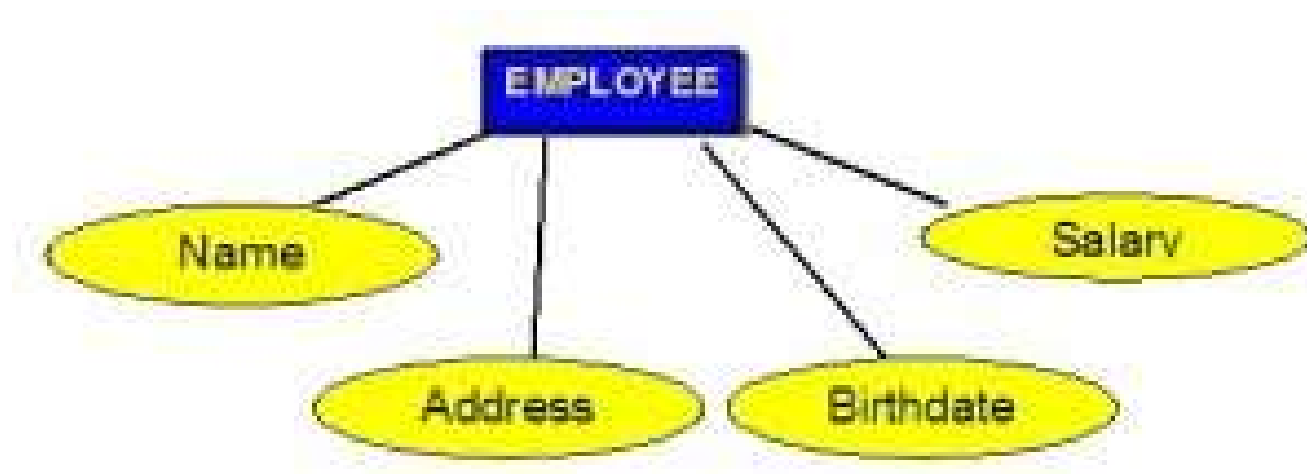
- Dependent entities are used to connect two kernels together.
- They are said to be existence dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.

Attributes

- Each entity is described by a set of attributes (e.g., Employee = (Name, Address, Birthdate (Age), Salary).
- Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the ERD.

Entity and Attributes in ERD

In the entity relationship diagram, shown bellow, each attribute is represented by an oval with a name inside.



Attribute Types

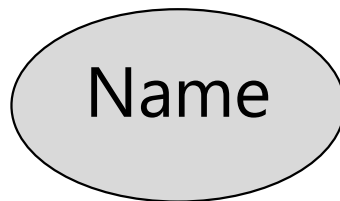
- Each entity has attributes – **properties that describe it.**
- A particular entity will have values for its attributes.
- E.g. an employee entity may have name, age, home address, home phone etc as their attributes.
- **Keys:** Unique value in the context.

Attribute Types			
Simple vs Composite	Single vs Multivalued	Stored vs Derived	Null vs Complex

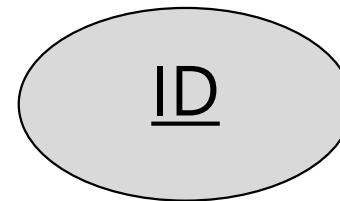
Attributes Notation

- **Attribute Domain:** The set of allowable values for one or more attributes.
- Each attribute is associated with a set of values called a domain.
- **Notation:** Attributes are represented by ovals.

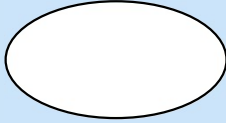
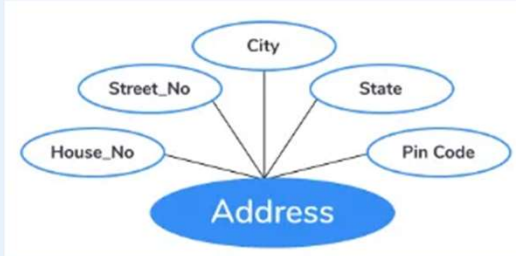
Notation for an Attribute



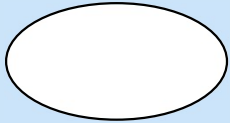
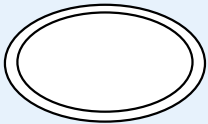
Notation for an **Key** Attribute



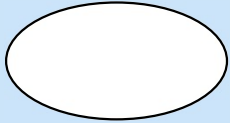
Attribute Types - 1

Type	Description	Example
Simple Attribute 	An attribute composed of a single component with an independent existence. Also called as atomic attributes.	Employee entity: Position, salary etc.
Composite Attribute 	An attribute composed of multiple components, each with an independent existence.	Employee entity: Address → can be made up of <i>street, town, country, postcode</i>

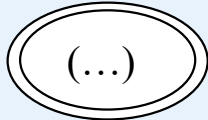
Attribute Types - 2

Type	Description	Example
Single Valued Attribute 	An attribute that holds a single value for each occurrence of an entity type.	Employee entity: Salary → can be only one value. Cannot draw more than one salary.
Multi Valued Attribute 	An attribute that holds multiple values for each occurrence of an entity type.	Employee entity: PhoneNo → A person can have one or more mobile phone number, landline etc.

Attribute Types - 3

Type	Description	Example
Stored Attribute 	An attribute that holds a value provided as is.	Employee entity: Name → can be set by the user as used as is.
Derived Attribute 	An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.	Employee entity: Age → can be derived from dateOfBirth and currentDate.

Attribute Types - 4

Type	Description	Example
NULL Attribute 	An attribute that means either the value is unknown or inapplicable. It does not mean zero or blank.	Employee entity: Commission → can be NULL
Complex Attribute 	An attribute that is formed by nesting composite attributes and multi-valued attributes in arbitrary way. We can say this as the both are in the attribute.	Employee entity: A person can have many phone numbers, many e-mail addresses, home addresses etc. So if there is an attribute in the name of 'Contact detail', it can be a complex attribute.

Types of Attributes

There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model..

Types of Attributes

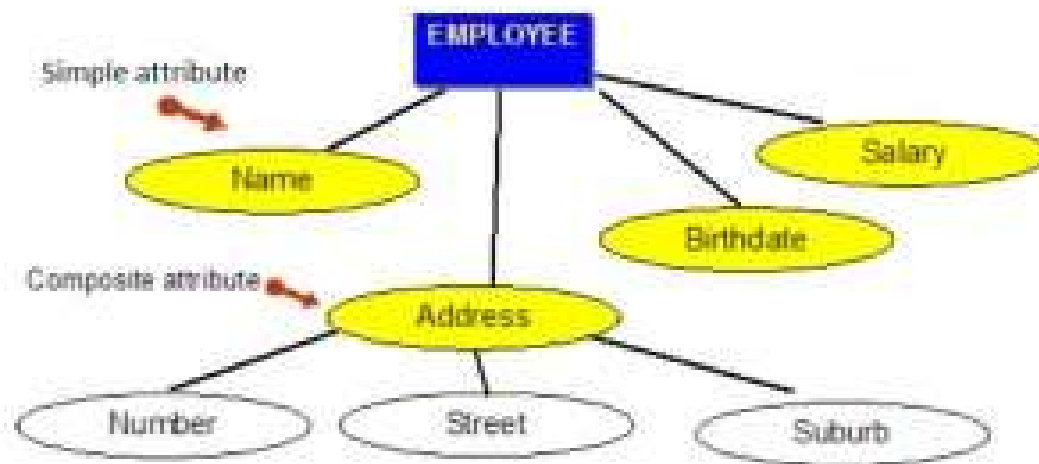
Simple attributes

Simple attributes are those drawn from the atomic value domains; they are also called *single-valued attributes*. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

Types of Attributes

Composite attributes

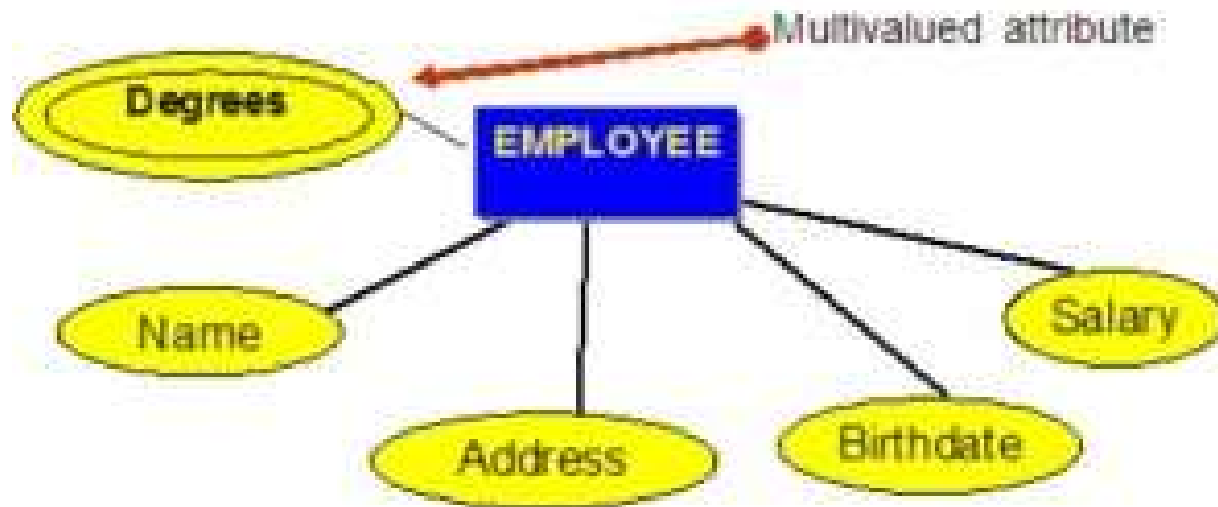
Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown, Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}



Types of Attributes

Multivalued attributes

Multivalued attributes are attributes that have a set of values for each entity. An example of a multivalued attribute from the COMPANY database, as seen are the degrees of an employee: BSc, MIT,



Knowledge Check: Attributes

<https://forms.office.com/e/EkZ8AhrXAx>



Keys

An important constraint on an entity is the key. The *key* is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set..

Types of Keys

Candidate key

A **candidate key** is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

Keys

From a COMPANY database example, if the entity is **Employee**(EID, First Name, Last Name, Address, Phone, BirthDate, Salary, DepartmentID), *possible candidate keys are:*

- EID,
- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and DepartmentID – assuming two people with the same last name don't work in the same department

Keys

Composite key

A **composite key** is composed of two or more attributes, but it must be minimal.

Using the example from the candidate key section, possible composite keys are:

- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and Department ID – assuming two people with the same last name don't work in the same department

Keys

Primary key

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
- A key is chosen by the database designer to be used as an identifying mechanism for the whole

entity set. This is referred to as the primary key. The key is indicated by underlining the attribute in the ER model.



Keys

Primary key

The primary key

In the following example, EID is the primary key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Foreign key

A **foreign key (FK)** is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type.

In the COMPANY database example below, DepartmentID is the foreign key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Nulls

A **null** is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

- No data entry
- Not permitted in the primary key
- Should be avoided in other attributes
- Can represent
 - An unknown attribute value
 - A known, but missing, attribute value
 - A “not applicable” condition
- Can create problems when functions such as COUNT, AVERAGE and SUM are used
- Can create logical problems when relational tables are linked

Nulls

Example of how null can be used

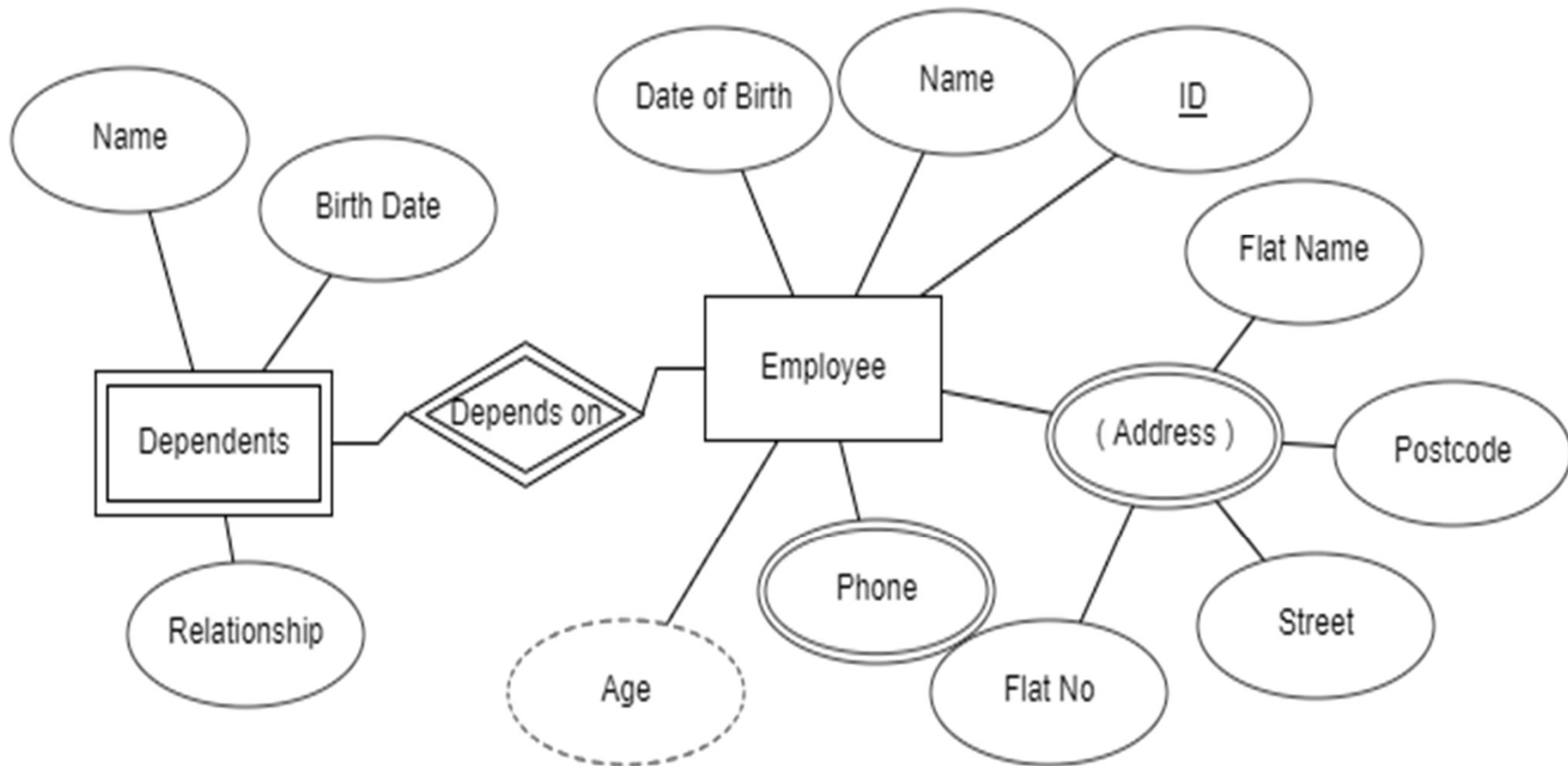
Use the Salary table (Salary_tbl) to follow an example of how null can be used.

Salary_tbl

emp#	jobName	salary	commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null



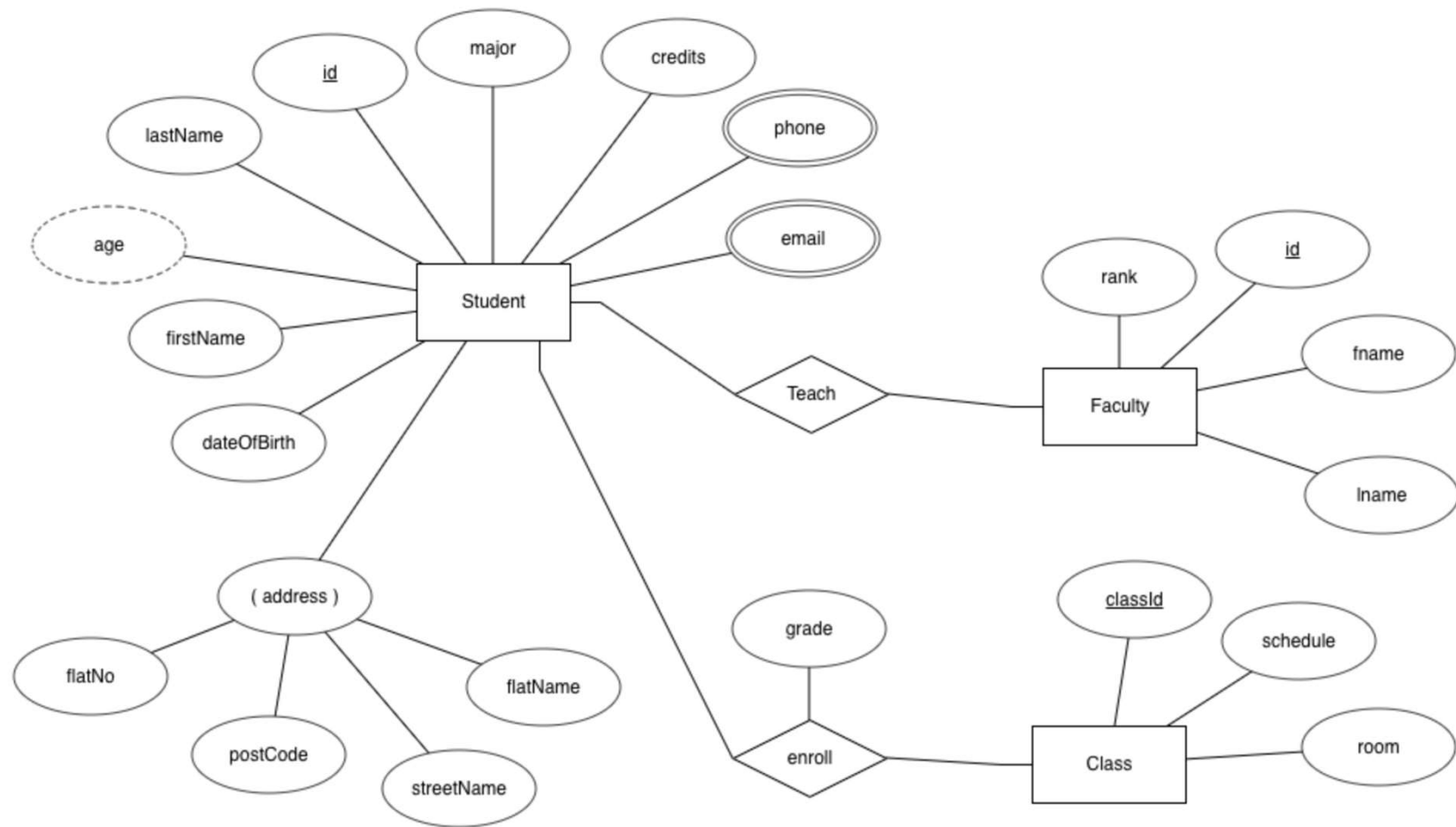
Entity, Attributes & Relationship



Entity, Attributes & Relationship

- Identify the entities, attributes and possible relation for University Database and represent it in ERD

Entity & Attributes



ERD Activity

- Identify the entities, attributes and possible relation for Car Parking Management System and represent it in ERD

In-Class Activity – Car Parking Management System

<https://forms.office.com/e/4VR3SfxnEU>



Relationships

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. Company database examples include:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, Date)

Relationships

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity.

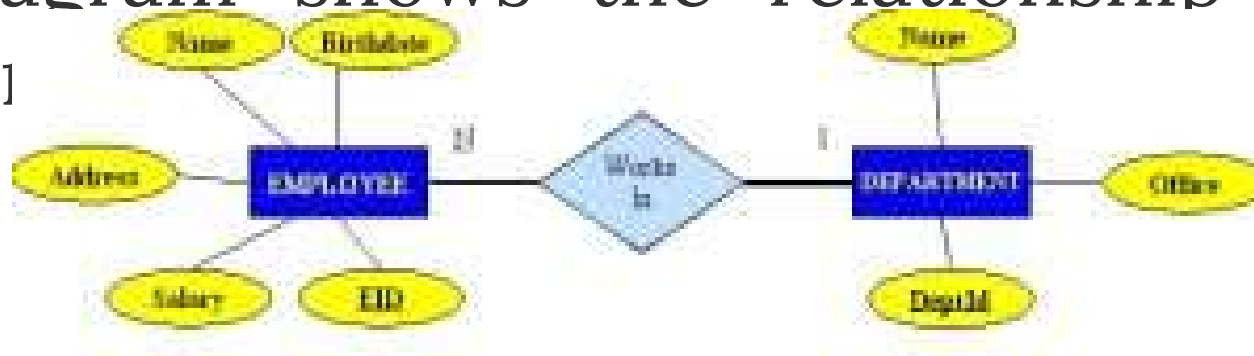
Examples include:

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode**, **Section**, ClassTime...)

Types of Relationships

One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. The diagram shows the relationship of one of these e



Relational Schema

EMPLOYEE(EID, Name, Address, Birthdate, Salary, DeptId)

DEPARTMENT(DeptId, Name, Office)

Types of Relationships

One to one (1:1) relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee. (one employee – one department)

Types of Relationships

Many to many (M:N) relationships

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.

Types of Relationships

Many to many (M:N) relationships

- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes



Relational Schema

EMPLOYEE (EID, Name, Address, Birthdate, Salary)

PROJECT (Code, Name, Budget)

JOIN (EID, Code, StartDate)

Integrity Rules and Constraints

Constraints are a very important feature in a relational model. In fact, the relational model supports the well-defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database.

Constraints are the rules that force DBMSs to check that data satisfies the semantics.

Integrity Rules and Constraints

Entity integrity

To ensure **entity integrity**, it is required that every table have a primary key. Neither the PK nor any part of it can contain null values. This is because null values for the primary key mean we cannot identify some rows. For example, in the EMPLOYEE table, Phone cannot be a primary key since some people may not have a telephone.

Integrity Rules and Constraints

Referential integrity

Referential integrity requires that a foreign key must have a matching primary key or it must be null. This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to another table must be valid.

Examples of referential integrity constraint in the Customer/Order database of the Company:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, OrderDate)

Referential integrity using Transact-SQL (MS SQL Server)

When using Transact-SQL, the referential integrity is set when creating the Order table with the FK. Listed below are the statements showing the FK in the Order table referencing the PK in the Customer table.

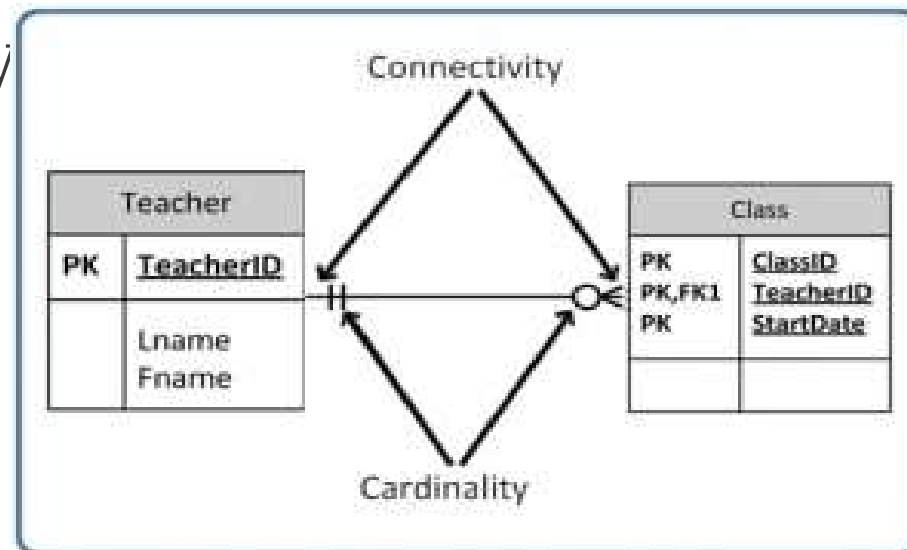
Referential integrity using Transact-SQL (MS SQL Server)

```
CREATE TABLE Customer  
( CustID INTEGER PRIMARY KEY,  
  CustName CHAR(35) )
```

```
CREATE TABLE Orders  
( OrderID INTEGER PRIMARY KEY,  
  CustID INTEGER REFERENCES Customer(CustID),  
  OrderDate DATETIME )
```

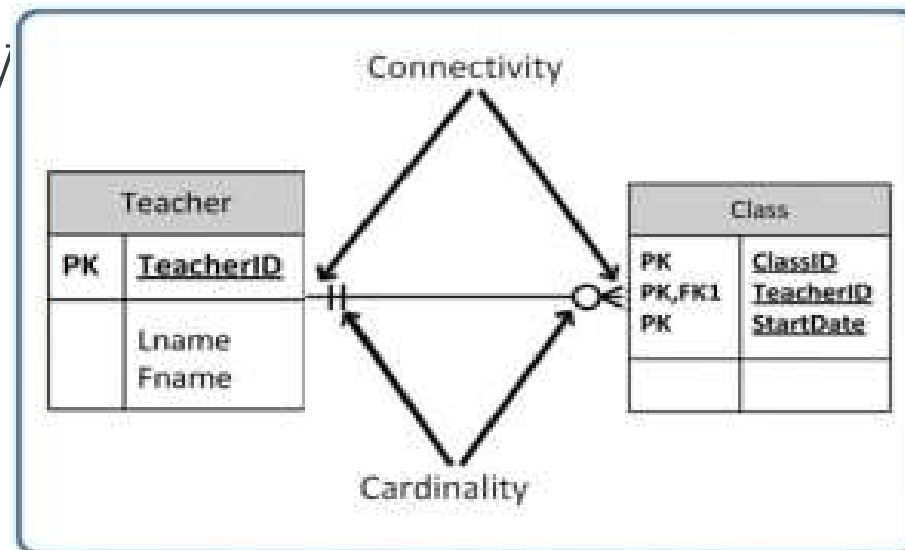
Cardinality and connectivity

Business rules are used to determine cardinality and connectivity. **Cardinality** describes the relationship between two data tables by expressing the minimum and maximum number of entity occurrences associated with one occurrence of a related entity.



Cardinality and connectivity

Business rules are used to determine cardinality and connectivity. **Cardinality** describes the relationship between two data tables by expressing the minimum and maximum number of entity occurrences associated with one occurrence of a related entity.



Cardinality and connectivity

The symbol that represents a **one to many** relationship.



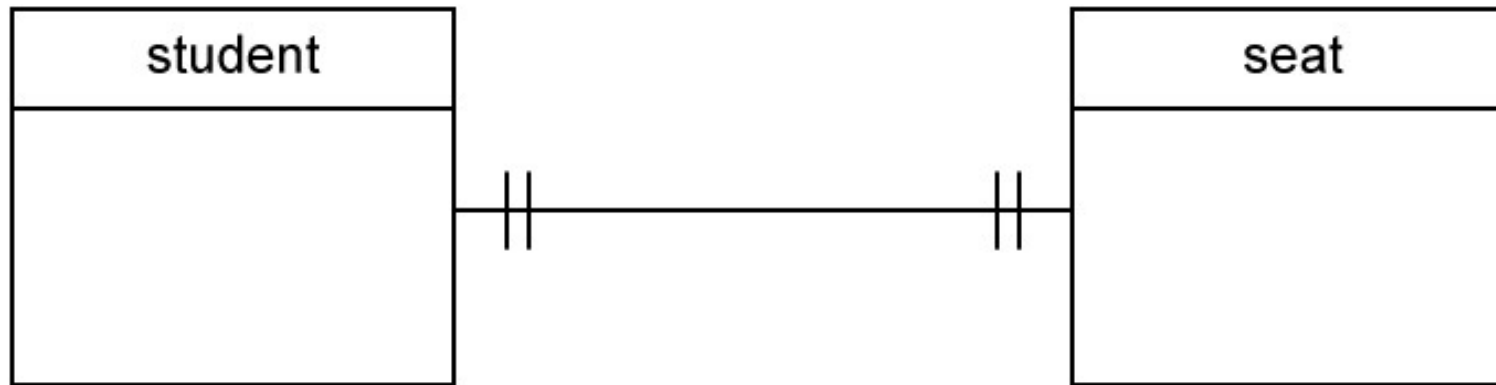
The left side of this symbol is read as minimum 1 and maximum 1. On the right side, it is read as: minimum 1 and maximum many.



Cardinality and connectivity

Relationship degrees make them readable as

One to One



Cardinality and connectivity

Relationship degrees make them readable as

One to Many



Cardinality and connectivity

Relationship degrees make them readable as

Many to Many

