# Logical Database Design 1

# Topics List

- Logical Database Design for the Relational Model
- Build and Validate Logical Data Model
- Derive relations for logical data model

# Logical Database Design for the Relational Model

- The purpose of logical design is to translate the conceptual representation to the logical structure of the database, which includes designing the relations.

- Logical design is the process of constructing a model of the data used in an enterprise based on a specific data model (in our case *relational*), but independent of a particular DBMS and other physical considerations.

# Logical Database Design for the Relational Model

- Logical design is a **semiphysical** realisation of the concepts. We say semiphysical because we are really not concerned with the actual physical file that is stored in memory; rather, we are concerned with placing data into relational tables that we will visualise as a physical organisation of data.

- Recall, a relational database is a database of two-dimensional tables called *relations*. The tables are composed of rows (tuples) and columns (attributes). In a relational database, all attributes must be atomic (simple), and keys must be not null.

- The process of converting an ER diagram into a database is called **mapping**.

# Logical Database Design for the Relational Model

- Recall the Overview Database Design Methodology:
  - Step 1 Build conceptual data model.
  - Step 2 Build and validate logical data model.
  - Step 3 Translate logical data model for target DBMS.
  - Step 4 Design file organisations and indexes.
  - Step 5 Design user views.
  - Step 6 Design security mechanisms.
  - Step 7 Consider the introduction of controlled redundancy.
  - Step 8 Monitor and tune the operational system.

# Logical Database Design for the Relational Model
## Build and Validate Logical Data Model

- Translate the conceptual data model into a logical data model and then validate this model to check that it is structurally correct using normalization and supports the required transactions.
  - Step 2.1 Derive relations for logical data model.
  - Step 2.2 Validate relations using normalization.
  - Step 2.3 Validate relations against user transactions.
  - Step 2.4 Define integrity constraints.
  - Step 2.5 Review logical data model with user.
  - Step 2.6 Merge logical data models into global model (optional step).
  - Step 2.7 Check for future growth.

# Topics List

- Logical Database Design for the Relational Model
- Build and Validate Logical Data Model
- Derive relations for logical data model

# Build and Validate Logical Data Model

- In this step, we create relations for the logical data model that will represent the entities, relationships, and attributes that have been identified.

- We will look at:
  - How to represent entity types
  - How to represent relationship types

# Topics List

- Logical Database Design for the Relational Model
- Build and Validate Logical Data Model
- Derive relations for logical data model

# Derive relations for logical data model

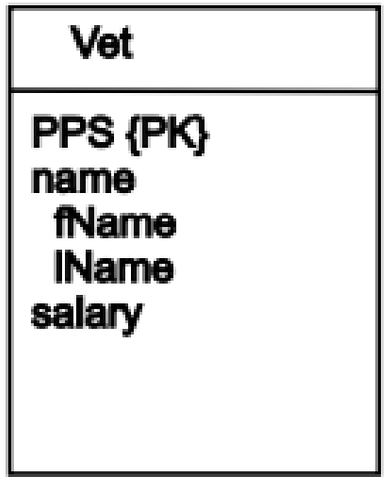
## How to represent entity types

- For each strong entity in the data model:
  - Create a relation that includes all the atomic (simple) single valued attributes of that entity.
  - For composite attributes, include only the constituent simple attributes.
  - Make the indicated primary key of the strong entity type the primary key of the table.
  - If the entity type has multi valued attribute(s), do not include them. (We will look at how to represent them later).

# Derive relations for logical data model

## How to represent entity types

- **Example:**
  - We have a **Vet** Entity with the following attributes: simple attribute PPS; composite attribute *name* (made up of fName and lName), and simple attribute salary. PPS is the primary key field.



# Derive relations for logical data model

## How to represent entity types

- The resulting relation is as follows:

  Vet(PPS, fName, lName, salary)

  Primary key PPS

- **Notes:**
  - The name of the entity type (Vet) becomes the name of the relation.
  - The composite attribute *name* is not included in the relation, only its constituent attributes are included.
  - The primary key of the entity type is the primary key of the relation.

# Derive relations for logical data model

## How to represent relationship types

- We will use the primary key/foreign key mechanism (i.e. copy the primary key attribute from one entity type (on one side of the relationship) into another entity type as a foreign key (on the other side of the relationship)).

- In deciding where to *post* (or copy) the foreign key attribute(s), we must first identify the ‘parent’ and ‘child’ entities involved in the relationship. The parent entity refers to the entity that posts a copy of its primary key into the table that represents the child entity, to act as the foreign key.

# Derive relations for logical data model

## How to represent relationship types

- Depending on the cardinality of the relationship and subsequently the participation, we consider how to represent the following relationships:
  - Binary relationships:
    - one-to-many (1:*);
    - many-to-many (*:*)
    - one-to-one (1:1);

# Derive relations for logical data model

How to represent relationship types

---

## ***one-to-many (1:*)***

The entity on the ‘one side’ of the relationship is designated as the **parent** entity and the entity on the ‘many side’ is designated as the **child** entity.

- To represent this relationship, ***post a copy of the primary key attribute(s) of the parent entity (one side) into the relation representing the child entity (many side), to act as a foreign key***.

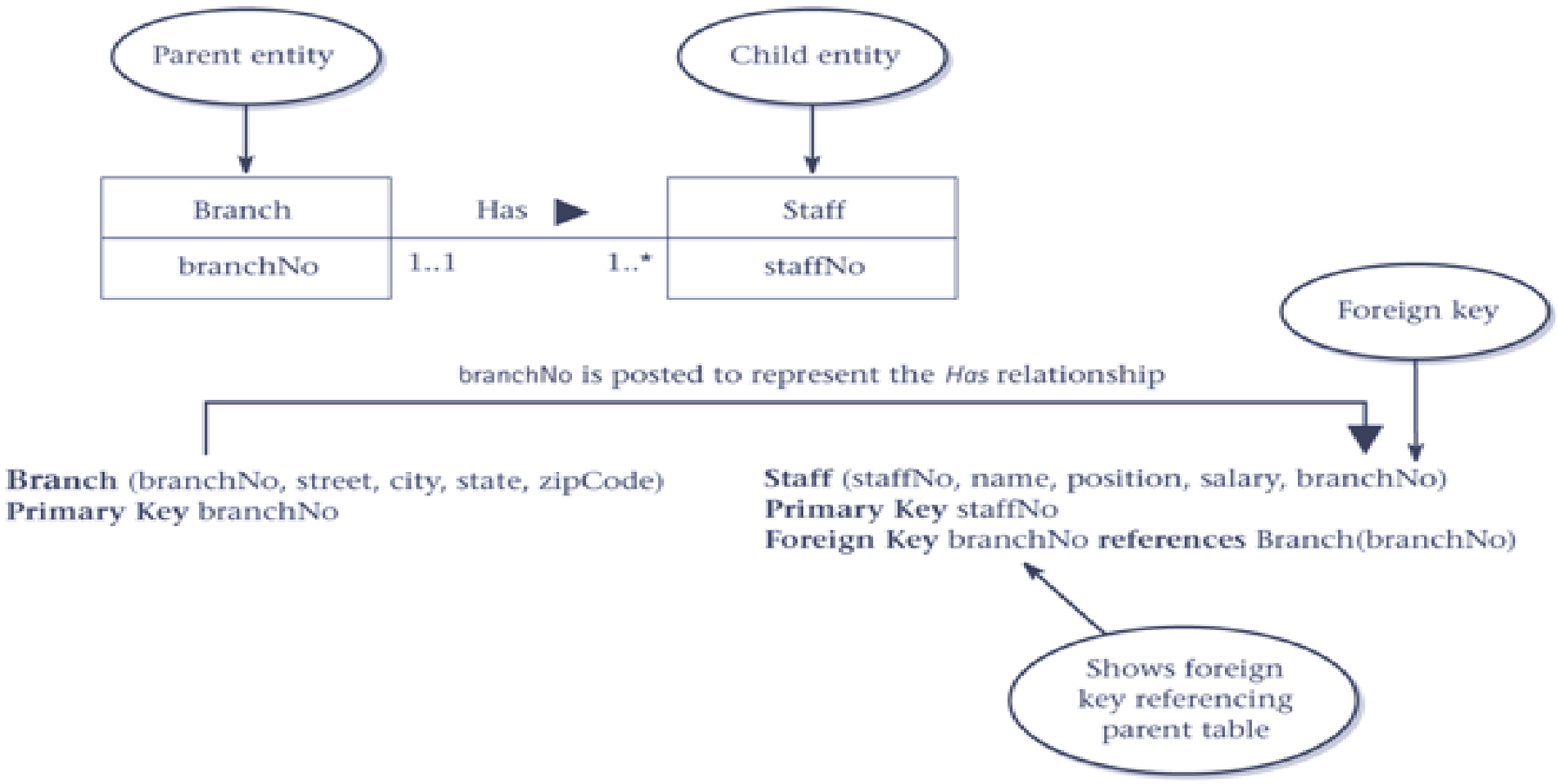
- If the relationship has one or more attributes, these attributes should follow the posting of the primary key to the child relation.

# Derive relations for logical data model

## How to represent relationship types

---

## ***one-to-many (1:*)***



# Derive relations for logical data model

How to represent relationship types

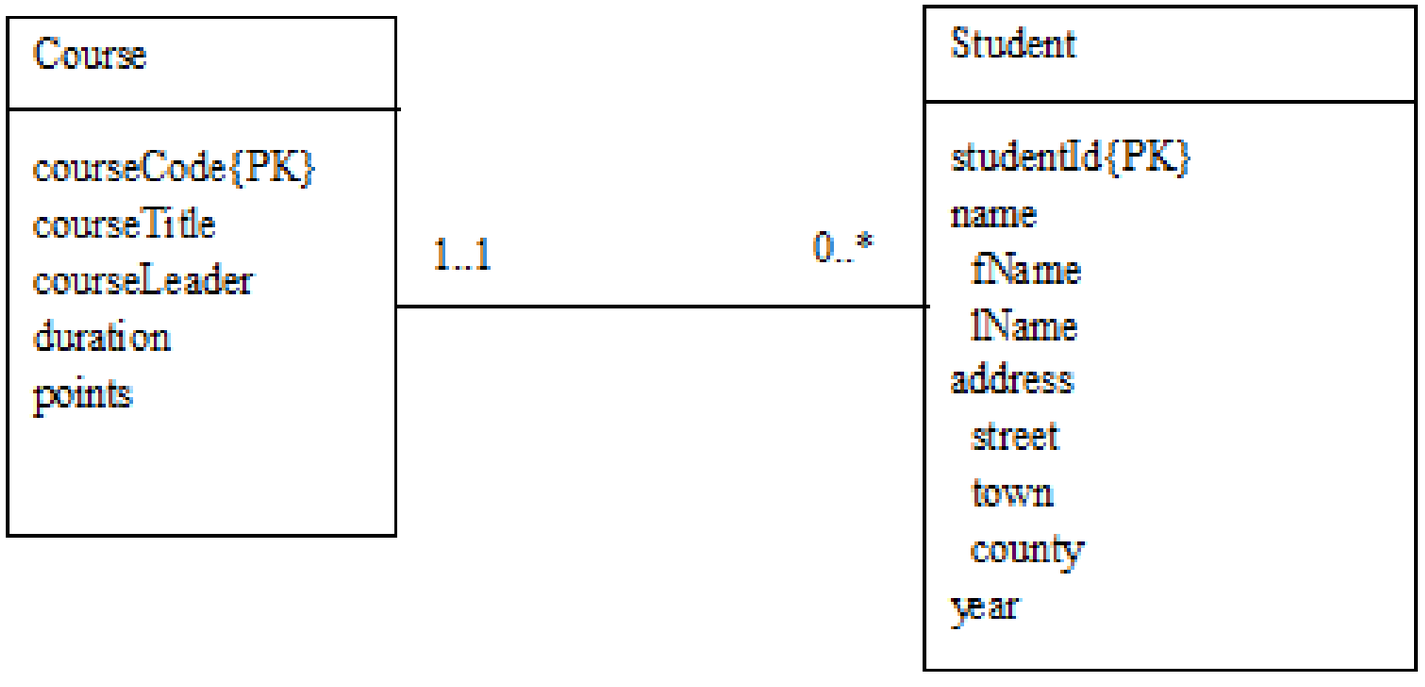
***one-to-many (1:*)***

- In the above example, *branchNo* is added as an extra attribute in the Staff relation. It is a foreign key which means it will draw its' value from *branchNo* in the *Branch* relation.

# Derive relations for logical data model

## How to represent relationship types

- **Exercise**
  - Transform the following ER diagram into a set of relations



# Derive relations for logical data model

## How to represent relationship types

- **Exercise**

# Derive relations for logical data model

## How to represent relationship types

---

### ***many-to-many (*:*)***

- When a relationship is many-to-many, we cannot post from one side to the other as we cannot post from the *many* side.

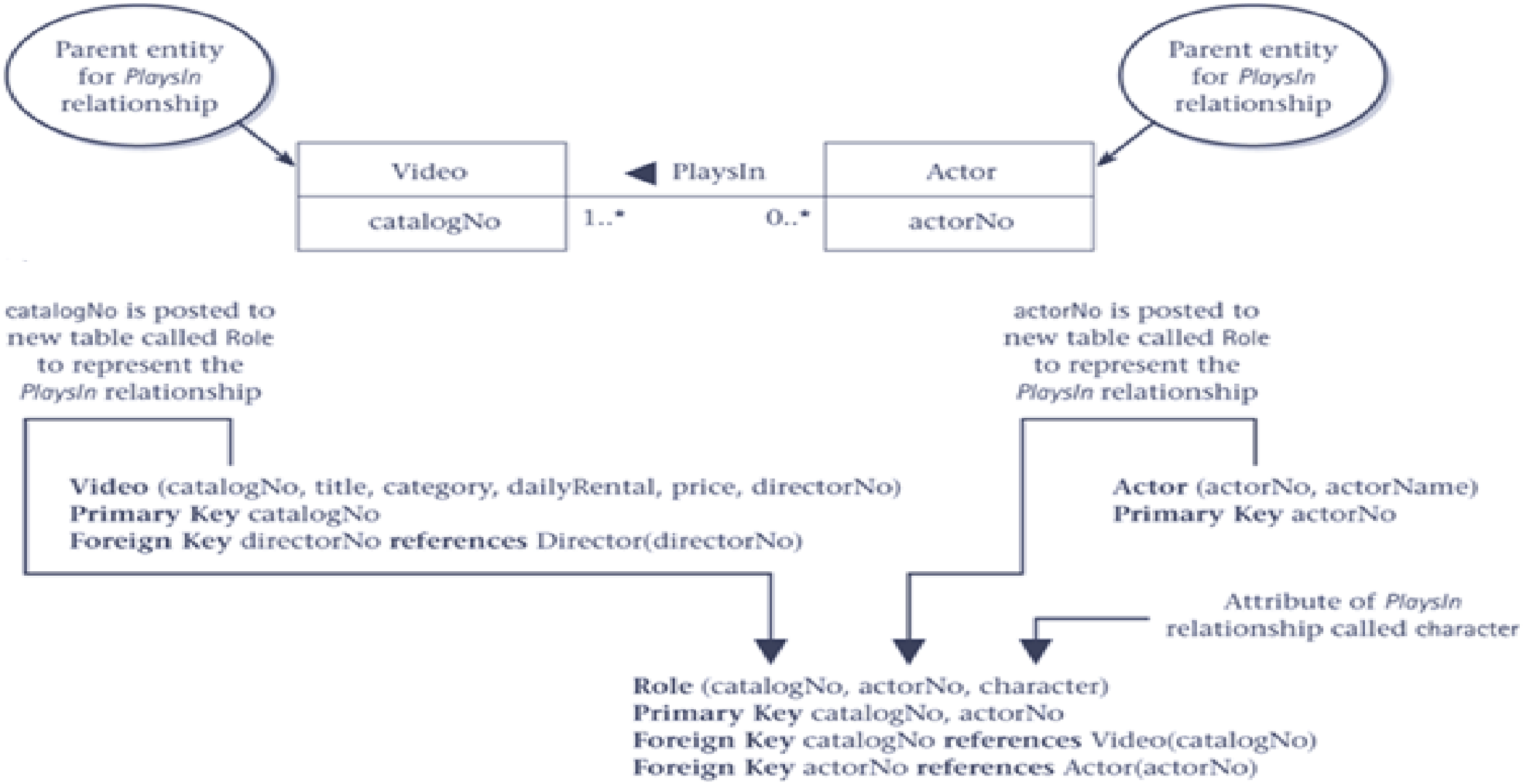
- ***Create a relation to represent the relationship and include any attributes that are part of the relationship. We post a copy of the primary key attribute(s) of the entities that participate in the relationship into the new relation, to act as foreign keys.*** These foreign keys will also form the primary key of the new relation, possibly in combination with some of the attributes of the relationship.

# Derive relations for logical data model

## How to represent relationship types

---

## ***many-to-many (*:*)***



# Derive relations for logical data model

## How to represent relationship types

---

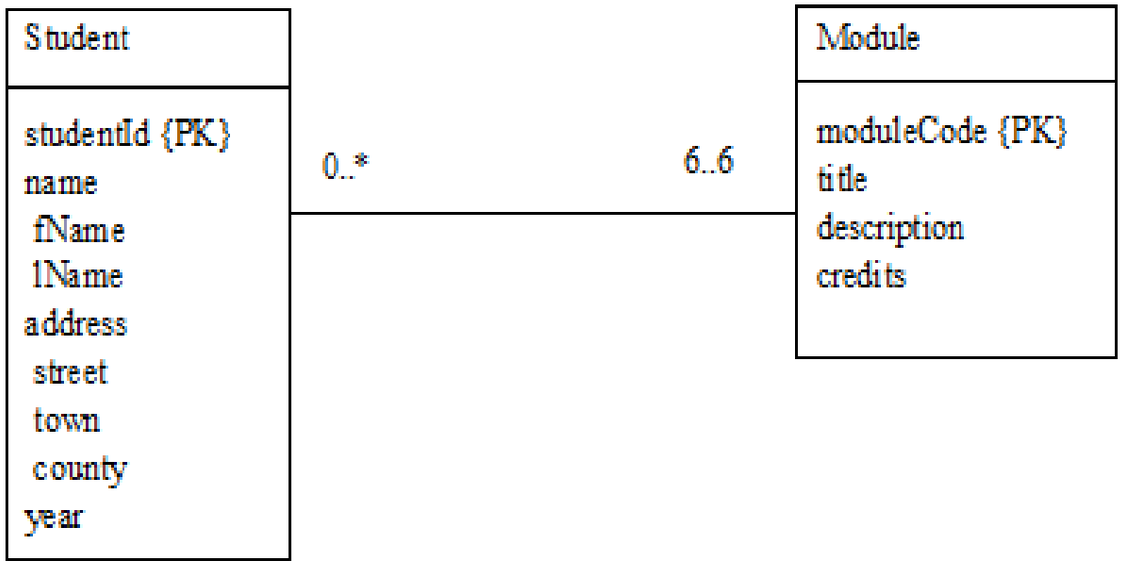
### ***many-to-many (*:*)***

- In the above example, we create a new relation (Role) to represent the relationship. Attributes *catalogNo* (Primary key of Video) and *actorNo* (primary key of Actor) are 2 attributes of this new relation. Both attributes are foreign key values related back to their respective relations (i.e. catalogNo draw its' value from catalogNo in the Video relation; and actorNo draw its' value from actorNo in the Actor relation). There will be a composite Primary key in the new relation (Role), the key is *catalogNo, actorNo*.

# Derive relations for logical data model

## How to represent relationship types

- **Exercise**
  - Transform the following ER diagram into a set of relations



# Derive relations for logical data model

## How to represent relationship types

- **Exercise**

# Derive relations for logical data model

## How to represent relationship types

---

### ***one-to-one (1:1)***

- Creating relations to represent a 1:1 relationship is more complex as the cardinality cannot be used to identify the parent and child entities in a relationship. Instead, the *participation constraints* are used to decide whether it is best to represent the relationship by combining the entities involved into one relation or by posting a copy of the primary key from one relation to the other.

- Consider the following:
  - *mandatory* participation on *both* sides of 1:1 relationship
  - *mandatory* participation on *one* side of 1:1 relationship
  - *optional* participation on *both* sides of 1:1 relationship

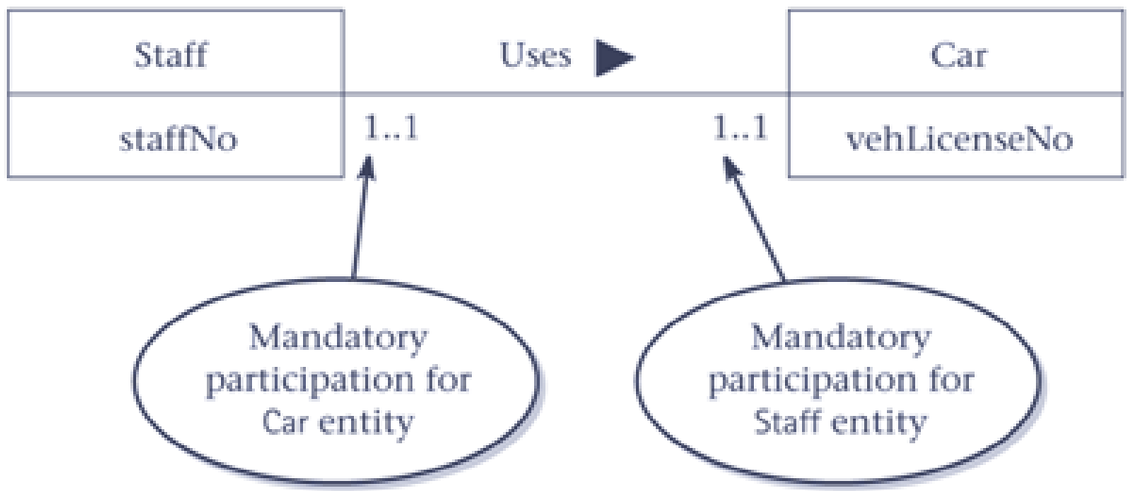
# Derive relations for logical data model

## How to represent relationship types

---

***one-to-one (1:1) - Mandatory* participation on *both* sides**

- One alternative is to ***post a copy of the primary key attribute(s) from one side of the relationship to the other side as a foreign key.***



# Derive relations for logical data model

## How to represent relationship types

---

***one-to-one (1:1) - Mandatory* participation on *both* sides**

- Option One:

  Staff(staffNo, name, position, salary)
  Primary Key staffNo

  Car(regNo, make, model,  staffNo)
  Primary Key regNo
  Foreign Key staffNo references Staff(staffNo)

- In this example, we post a copy of staffNo into the Car relation as a foreign key field.

# Derive relations for logical data model

## How to represent relationship types

---

***one-to-one (1:1) - Mandatory* participation on *both* sides**

- Option Two:

  Car(regNo, make, model)
  Primary Key regNo

  Staff(staffNo, name, position, salary, regNo)
  Primary Key staffNo
  Foreign Key regNo references Car (regNo)

- In this example, we post a copy of regNo into the Staff relation as a foreign key field.

# Derive relations for logical data model

## How to represent relationship types

***one-to-one (1:1) - Mandatory* participation on *both* sides**

- Another alternative is to ***combine the entities involved into one relation and choose one of the primary keys of original entities to be primary key of the new relation, while the other is used as an alternate key***.

Staff(staffNo, name, position, salary, regNo, make, model)

Primary Key staffNo

# Derive relations for logical data model

## How to represent relationship types

---

### ***one-to-one (1:1) - Mandatory* participation on *one* side**

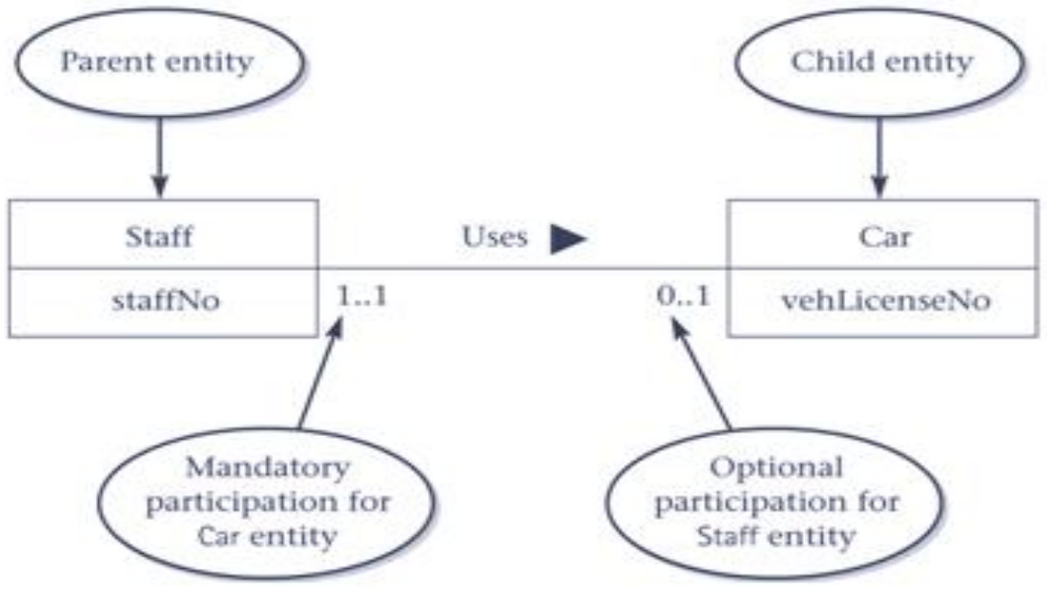
- Identify parent and child entities using participation constraints. Entity with optional participation in relationship is designated as the **parent** entity, and entity with mandatory participation is designated as the **child** entity.

- ***A copy of the primary key of the parent entity (entity with optional participation) is placed in the relation representing the child entity (entity with mandatory participation).*** If the relationship has one or more attributes, these attributes should follow the posting of the primary key to the child relation.

# Derive relations for logical data model

How to represent relationship types

## ***one-to-one (1:1) - Mandatory* participation on *one* side**




Staff(staffNo, name, position, salary)
Primary Key staffNo

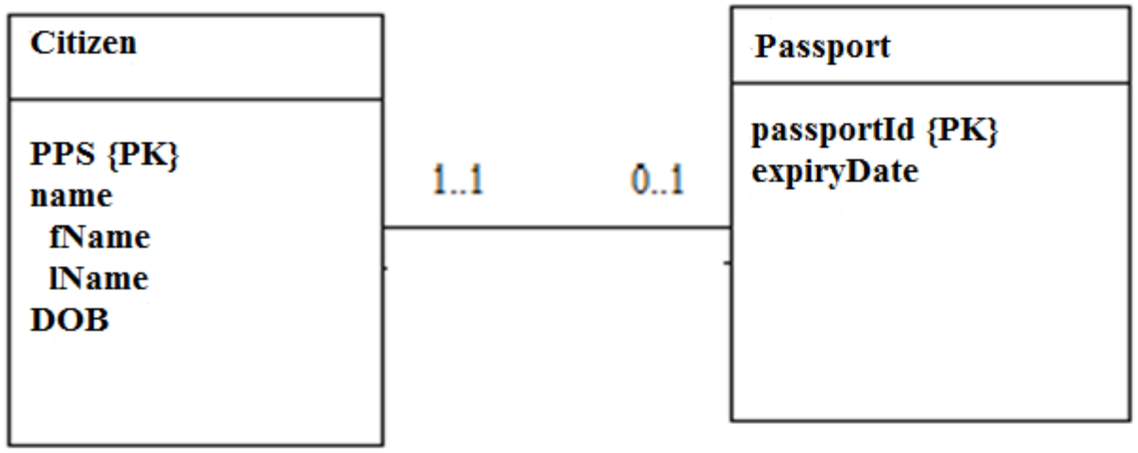
Car(regNo, make, model, staffNo)
Primary Key regNo
Foreign Key staffNo references Staff(staffNo)

In this example, we post a copy of staffNo into the Car relation as a foreign key field because the participation of Staff in the relationship is optional.

# Derive relations for logical data model

## How to represent relationship types

- **Exercise**
  - Transform the following ER diagram into a set of relations



# Derive relations for logical data model

## How to represent relationship types

- **Exercise**

# Derive relations for logical data model

## How to represent relationship types

---

***one-to-one (1:1) – Optional* participation on *both* sides**

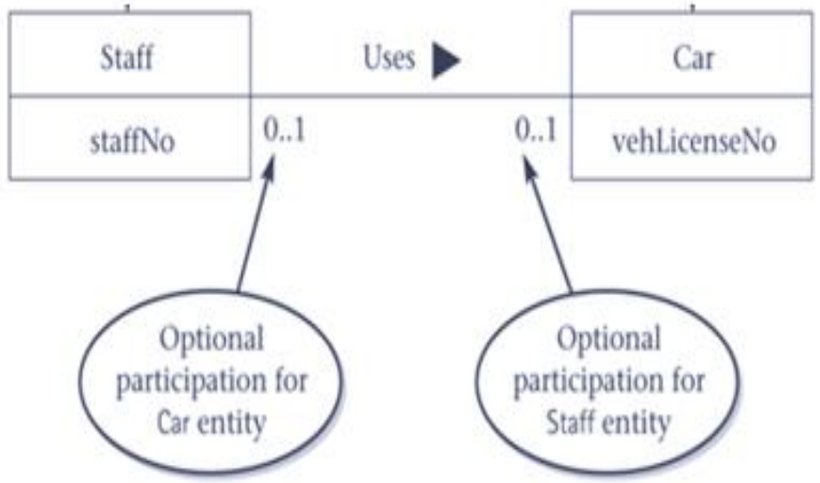
- One solution is to ***post from one side of the relationship to the other side of the relationship. Since both sides are 'optional' you look at the data and post from the side with the most 'optional' participation to the other side.***

# Derive relations for logical data model

## How to represent relationship types

---

### ***one-to-one (1:1) – Optional* participation on *both* sides**




Staff(staffNo, name, position, salary)
Primary Key staffNo

Car(regNo, make, model, staffNo)
Primary Key regNo
Foreign Key staffNo references Staff(staffNo)

Assuming most of the Cars are allocated and only a minority of Staff uses a Car, we will post *staffNo* into *Car* as a foreign key.

# Derive relations for logical data model

## How to represent relationship types

---

### ***one-to-one (1:1) – Optional* participation on *both* sides**

- Another alternative is to ***create a new relation***. *We post a copy of the primary key attribute(s) of the entities that participate in the relationship into the new relation, to act as foreign keys.*

# Derive relations for logical data model

## How to represent relationship types

---

### ***one-to-one (1:1) – Optional* participation on *both* sides**

Staff(staffNo, name, position, salary)
Primary Key staffNo

*Car(regNo, make, model)*
*Primary Key regNo*

*Drives(staffNo, regNo)*
*Primary Key staffNo*
*Foreign Key staffNo references Staff(staffNo)*
*Foreign Key regNo references Car(regNo)*

staffNo or regNo is sufficient as the Primary Key field as each entity only participates once in the relationship.