SQL Constraints

- **Rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data**

- **Two types of Constraints:**

  - **Column level constraints : limits only column data**

  - **Table level constraints : limits whole table data**

NOT NULL Constraint

- Restricts a column to have a NULL value.
- Once **NOT NULL** constraint is applied to a column, a null value can not be passed to that column enforcing a column to contain a proper value.
- NOT NULL constraint cannot be defined at table level.

SQL Constraints

- ❑ NOT NULL
- ❑ UNIQUE
- ❑ PRIMARY KEY
- ❑ FOREIGN KEY
- ❑ CHECK
- ❑ DEFAULT

NOT NULL Constraint

- CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);

- The above query will declare that the **s_id** field of **Student** table will not take NULL value.

UNIQUE Constraint

- Ensures that a field or column will only have unique values.

- A UNIQUE constraint field will not have duplicate data.

- UNIQUE constraint can be applied at column level or table level.

UNIQUE Constraint

- **UNIQUE constraint (Table Level)**
- CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
- The above query will declare that the **s_id** field of **Student** table will only have unique values and wont take NULL value.

UNIQUE Constraint

- **UNIQUE constraint after Table is created (Column Level)**

- ALTER table Student add UNIQUE(s_id);

- The above query specifies that **s_id** field of **Student** table will only have unique value.

Primary Key Constraint

- Uniquely identifies each record.

- A Primary Key must contain unique value and it must not contain null value.

- Usually Primary Key is used to index the data inside the table.

Primary Key Constraint

- **PRIMARY KEY constraint at Table Level**

- CREATE table Student (s_id int **PRIMARY KEY**, Name varchar(60) NOT NULL, Age int);

- The above command will create a PRIMARY KEY on the s_id.

Primary Key Constraint

- **PRIMARY KEY constraint at Column Level**
- ALTER table Student add PRIMARY KEY (s_id);
- The above command will creates a PRIMARY KEY on the s_id.

Candidate Key

□ Candidate keys are defined as the set of fields from which primary key can be selected. It is an attribute or set of attribute that can act as a primary key for a table to uniquely identify each record in that table.

Foreign Key Constraint

□ To relate two tables.

□ To restrict actions that would destroy links between tables.

Foreign Key Constraint

□ **Customer_Detail Table :**

c_id	Customer_Name	address
101	Amjad	Taxila
102	Amir	Pindi
103	Sameer	Attock

□ **Order_Detail Table :**

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

Foreign Key Constraint

- In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table.

- The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key.

- Preventing invalid data to be inserted into c_id column of **Order_Detail** table.

Foreign Key Constraint

- **FOREIGN KEY constraint at Column Level**

- ALTER table Order_Detail add **FOREIGN KEY** (c_id) REFERENCES Customer_Detail(c_id);

Foreign Key Constraint

- **Behaviour of Foriegn Key Column on Delete**
- When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.

Foreign Key Constraint

- **FOREIGN KEY constraint at Table Level**

- CREATE table Order_Detail(order_id int PRIMARY KEY, order_name varchar(60) NOT NULL, *c_id int* **FOREIGN KEY** REFERENCES **Customer_Detail**(*c_id*));

- In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column of Customer_Detail.

Foreign Key Constraint

Deleting Foriegn Key

Cascade
(on Delete cascade)
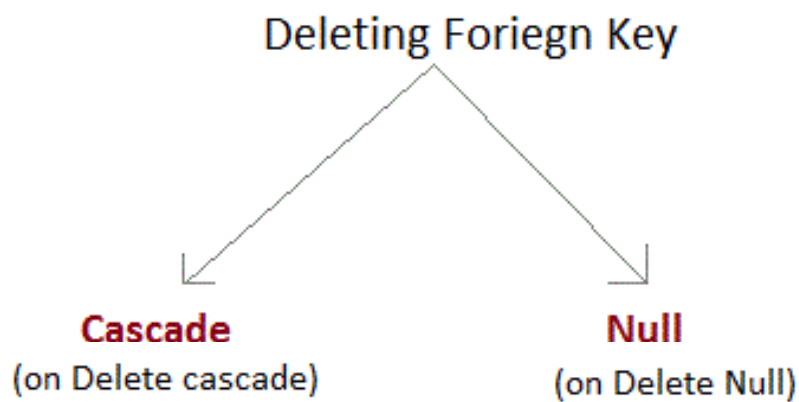
Null
(on Delete Null)

Foreign Key Constraint

- **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
- **On Delete Set Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.
- ERROR : Record in child table exists

```sql
CREATE TABLE products ( product_id INT PRIMARY KEY, product_name VARCHAR(50) NOT NULL,
category VARCHAR(25) );
CREATE TABLE inventory ( inventory_id INT PRIMARY KEY, product_id INT, quantity INT, min_level
INT, max_level INT, CONSTRAINT fk_inv_product_id FOREIGN KEY (product_id) REFERENCES
products (product_id) ON DELETE SET NULL );
----------------------
CREATE TABLE products ( product_id INT PRIMARY KEY, product_name VARCHAR(50) NOT NULL,
category VARCHAR(25) );
CREATE TABLE inventory ( inventory_id INT PRIMARY KEY, product_id INT NOT NULL, quantity
INT, min_level INT, max_level INT, CONSTRAINT fk_inv_product_id FOREIGN KEY (product_id)
REFERENCES products (product_id) ON DELETE CASCADE );
```

CHECK Constraint

- To restrict the value of a column between a range.

- Performing check on the values, before storing them into the database.

- Condition checking before saving data into a column.

CHECK Constraint

- **CHECK constraint at Table Level**

- create table Student(s_id int NOT NULL **CHECK(s_id > 0)**, Name varchar(60) NOT NULL, Age int);

- The above query will restrict the s_id value to be greater than zero.

CHECK Constraint

□ **CHECK constraint at Column Level**

□ ALTER table Student add CHECK(s_id > 0);

Sample Projects

- Student admission Management System
- Hotel Reservation system
- Library Management System
- Airline Management system
- Shopping Mall Management system
- Hospital Management system
- Voting/polling management system
- Bank management system
- Or anyother based on your interest

Entity-Relationship Data Model

E-R Data Model

- Graphical representation of conceptual DB design
- Major Components are;
  - Entities
  - Attributes
  - Their Relationships

Entity

- Term used to mean three different meanings
  - Entity type
  - Entity instance
  - Entity set

Entity

□ An **Entity** Type is a collection of occurrences of **entities** that have common properties (attributes). An example of an **entity** type is EMPLOYEE. An **Entity Instance** is a single occurrence of an **entity** type, e.g., the employee named TOM GREEN.

□ An **entity set** is a **set** of **entities** of the same type (e.g., all persons having an account at a bank). **Entity sets** need not be disjoint. For example, the **entity set** employee (all employees of a bank) and the **entity set** customer (all customers of the bank) may have members in common.

Entity

- Entity Type: Employee
- Entity Instance: J. Rashid
- Entity Set: All employees

Entities

- An entity is any object in the system that we <mark>want to model</mark> and <mark>store information</mark> about
  - Individual objects are called entities
  - Groups of same type objects are entity types
  - Entities are represented by rectangles (either with round or square corners)

Lecturer	Lecturer
Chen's notation	other notations

- There are two types of entities; weak/dependent and strong/ independent entity types.
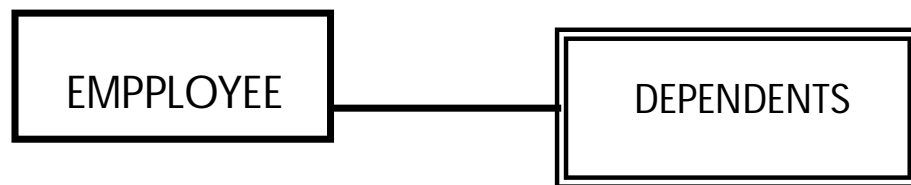
Weak Entity Types

An entity type whose instances cannot exist without being linked with instances of some other entity type, i.e. ;
they cannot exist independently

Strong Entity Type

□ A strong/regular entity type is the one whose instances can exist independently, i.e., without being linked to other instances

□ Strong ETs have their own identity

```
┌──────────────┐            ╔══════════════╗
│              │            ║              ║
│  EMPPLOYEE   │────────────║  DEPENDENTS  ║
│              │            ║              ║
└──────────────┘            ╚══════════════╝
```
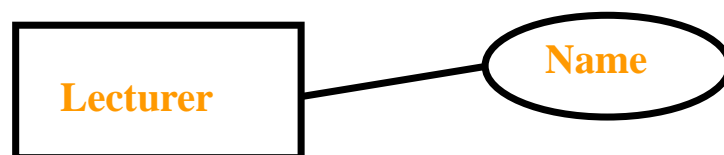
Attribute

An **attribute** of an entity type is a defining property of the instances of that entity type. Entity instances of same entity type have the same attributes. (e.g. Student ID, Employee Name)

Types of Attributes

- Single - composite
- Single valued - multi-valued
- Stored - derived

Attributes

- Each represented as an oval, linked with an ET symbol
  - They appear inside ovals and are attached to their entity.
  - Note that entity types can have a large number of attributes... If all are shown then the diagrams would be confusing. Only show an attribute if it adds information to the ER diagram, or clarifies a point.
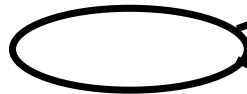
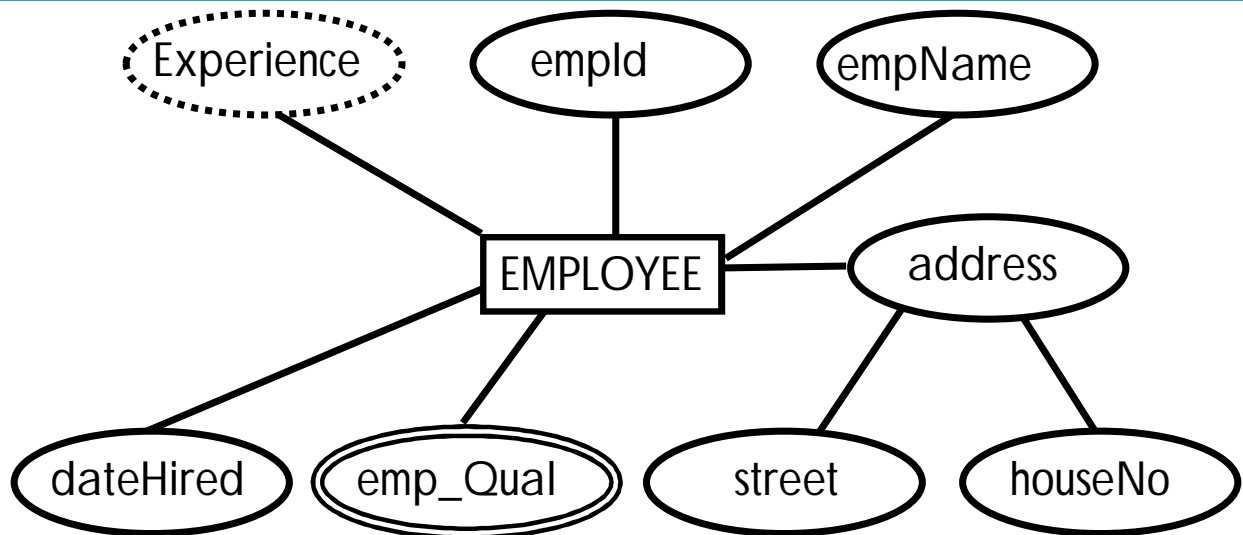# Symbols for Attributes

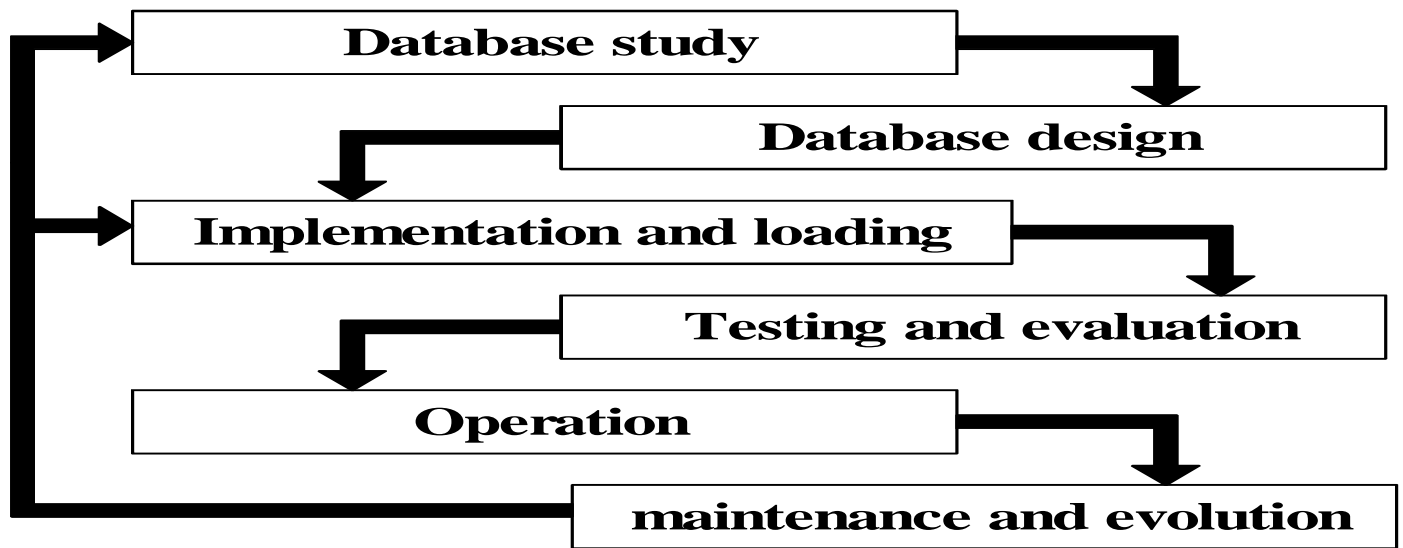Simple

Composite

Multi-valued

Derived

Example

A derived attribute is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm.

Entity Relationship Modelling

Overview

☐ Database Analysis Life Cycle

☐ Components of an Entity Relationship Diagram

☐ What is a relationship?

☐ Entities, attributes, and relationships in a system

☐ The degree of a relationship

☐ Construct an Entity Relationship Diagram

DB Analysis Life Cycle

Database study → Database design → Implementation and loading → Testing and evaluation → Operation → maintenance and evolution

Entity Relationship Modelling

- □ is a design tool
- □ is a graphical representation of the database system
- □ supports the user s perception of the data
- □ is DBMS and hardware independent
- □ **is composed of entities, attributes, and relationships**

ERD Benefits

□ ERDs contain metadata, or data about other data. For example, ERD data include boxes, names, and lines. The data that a line represents is the relationship between two entities in the ERD.

□ An entity name represents a table which will also contain data of its own.

□ **The ERD's great advantage is that it is simple enough that** both technical and nontechnical people can quickly learn to use it to communicate productively about data requirements

Entities in ERD

- Initially, identification of entities should be a freewheeling process. There are no dumb questions or ideas as you attempt to identify and capture the potential entities for your system.
- The idea is to capture as many potential entities as possible and let the ERD process weed out entities
- So that the first cut ERD can evolve into a complete product. They may prove to be useful later as the model evolves

Attributes Naming Conventions

- Create the attribute names in plain English so that the users and other developers can understand them.
- Be clear and specific with attribute names. For instance, create Annual_Performance_Bonus, not just Bonus.
- This is especially true of date attributes. Use Date_Invoice_Received, not just Date_Received.
- Avoid cryptic abbreviations, technical jargon, or other generally meaningless attribute names that may be fine for you but will just irritate and inconvenience other people trying to work with the system.
- What's more, a year later you, yourself, probably won't be clever enough to interpret your abbreviations, let alone some poor developer trying to maintain your system.

Attributes

PRODUCT	ORDER
PROD_ID	ORD_ID
NAME	PROD_ID
DESCRIP	DATE_ORD
COST	QUANTITY

☐ Break attributes up into their smallest logical pieces. For example, use first_name, last_name, and middle_initial rather than just name

☐ Important: An entity must have attributes; otherwise, it's not an entity.

Attributes Data types

□
   columns, the only attributes that should be assigned a numeric data type are those that contain values that are treated as numbers, such as those values used in calculations.

□ All other attributes should be assigned a character or date data type, as appropriate.
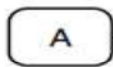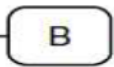
Entity

Mandatory relationship

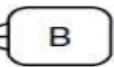Optional relationship
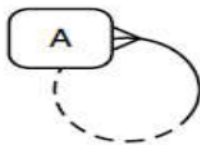
Mandatory/optional relationship

| A | B | One-to-one relationship |

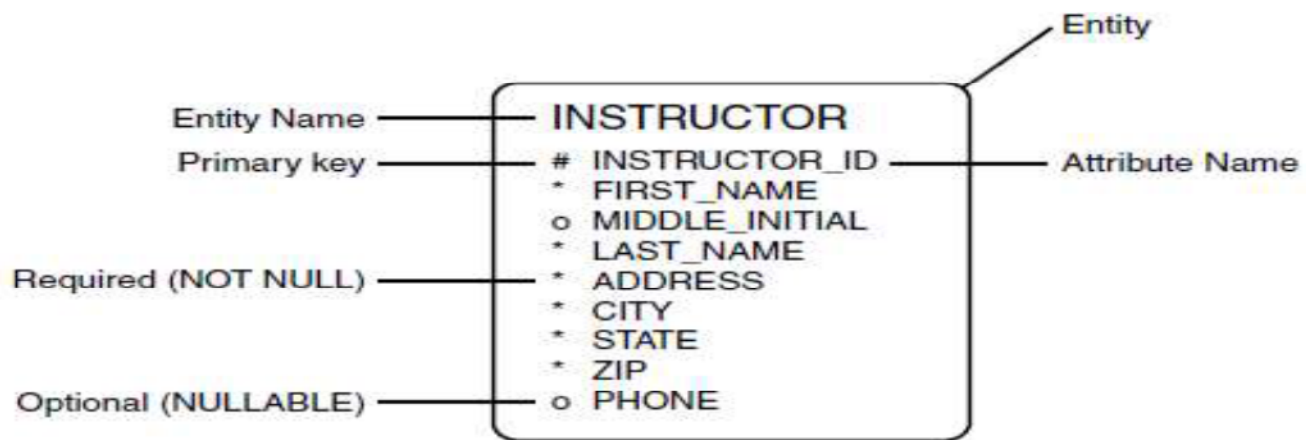| A | B | Many-to-one relationship (same as one-to-many) |

| A | B | Many-to-many relationship |

| A | Recursive relationship |

Typical Entity in ERD

Database Keys

- Keys are very important part of Relational database. They are used to establish and identify relation between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

Keys

□ **Super Key:** is defined as a set of attributes within a table that uniquely identifies each record within a table. Super Key is a superset of Candidate key.

□ **Candidate Key:** Candidate keys are defined as the set of fields from which primary key can be selected. It is an attribute or set of attribute that can act as a primary key for a table to uniquely identify each record in that table.

Keys

- **Primary Key:** Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.

- **Composite Key:** Key that consists of two or more attributes that uniquely identify an entity occurrence is called Composite key. But any attribute that makes up the Composite key is not a simple key in its own.

- **Alternative key:** The candidate key which are not selected for primary key are known as secondary keys or alternative keys.

- Once you have identified a list of potential entities and placed them into an ERD,

- The next step is to identify the relationships between the entities. Relationships are the verbs of your system.

Relationships

- A *relationship type* is a meaningful association between entity types

- A *relationship* is an association of entities where the association includes one entity from each participating entity type.

- Relationship types are represented on the ER diagram by a series of lines.

Steps in finding Relationships

- Check to see if a relationship exists.
- If a relationship exists, identify a verb for each direction of the relationship.
- Identify an **optionality** for each direction of the relationship.
- Identify a **degree** for each direction of the relationship.
- Validate the relationship by reading it.

Check if Relationship Exists

	instructor	department	student
instructor	none	yes	yes
department	yes	none	yes
student	yes	yes	none

	instructor	department	student
instructor	none	is employed by	teaches
department	employs	none	is major of
student	is taught by	majors in	none

□ If there is a relationship, identify a lowercase verb name for each direction of the relationship.

□ Active verbs for at least one direction are best. Verbs such as *is associated with, or is related to should be avoided due to their lack of clarity. The above table illustrates how*

□ relationships and their verbs can be determined using a simple matrix.

Identify Optionality or Mandatory

☐ Identify an **optionality** for each direction of the relationship. The relationship is either *must be* or *may be*. **Mandatory** *(must be relationships are represented by solid lines.* **Optional** (may be) relationships are represented by broken lines

Examples

- Each instructor **must** teach one or more students, and each student is taught by one or more instructors.

- Each department **may be** the major of one or more students, and each student may major in one or more departments.

- Each instructor **must** possess one or more instructor skills, and each instructor skill **may be** possessed by one or more instructors.

Relationships cont...

□ In the original Chen notation, the relationship is placed inside a diamond, e.g. managers manage employees:

| Manager | manages | Employee |

□ For this module, we will use an alternative notation, where the relationship is a label on the line. The meaning is identical

| Manager | manages | Employee |

Degree of a Relationship

- The number of participating entities in a relationship is known as the degree of the relationship.

- If there are two entity types involved it is a *binary* **relationship** type

Manager	manages	Employee

- If there are three entity types involved it is a *ternary* **relationship** type

Sales Assistant — sells — Product
Customer
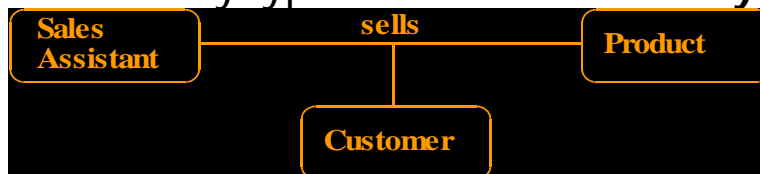
Degree of a Relationship cont...

□ Unary relationships are also known as a *recursive* relationship.

□ It is a relationship where the same entity participates more than once in different roles.

Cardinality cont...

- A one to one relationship   An employee have one pay record, so it is a one to one (1:1) relationship

- A one to may relationship - one manager manages many employees, but each employee only has one manager, so it is a one to many (1:n) relationship

| Manager | 1 | manages | m | Employee |

Cardinality cont…

□ A many to one relationship - many students study one course, so it is a many to one (m:1) relationship

| Student | m   studies   1 | Course |

□ A many to many relationship - One lecturer teaches many students and a student is taught by many lecturers, so it is a many to many (m:n) relationship

| Lecturer | m   teaches   n | Student |

Optionality

A relationship can be option or mandatory.

- If the relationship is mandatory
  - an entity at one end of the relationship must be related to an entity at the other end.
- If the relationship is optional
  - Entities at each end of the relationship may or may not have to relate to each other.

- The optionality can be different at each end of the relationship
  - For example, a student must be on a course. This is mandatory. To the relationship 'student studies course is mandatory.
  - But a course can exist before any students have enrolled. Thus the relationship 'course is_studied_by student is optional.
- To show optionality, put a circle or '0' at the 'optional end' of the relationship.
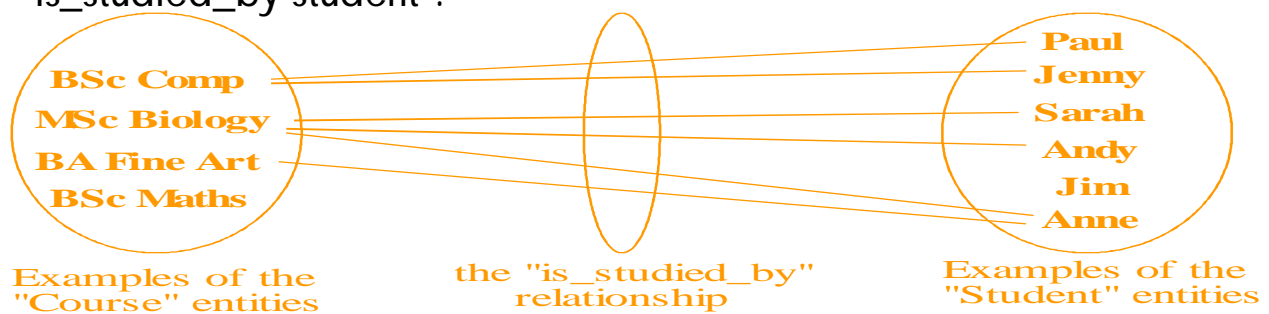
Optionality cont…

□ As the optional relationship is 'course is_studied_by student', and the optional part of this is the student, then the 'O goes at the student end of the relationship connection.
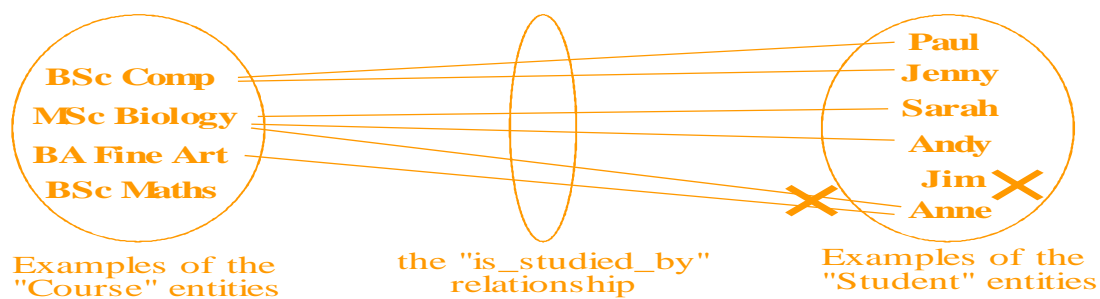

□ It is important to know the optionality because you must ensure that whenever you create a new entity it has the required mandatory links.

Entity Sets (Diagram)

Sometimes it is useful to try out various examples of entities from an ER model. One reason for this is to confirm the correct cardinality and optionality of a relationship. We use an 'entity set diagram' to show entity examples graphically. Consider the example of 'course is_studied_by student'.

BSc Comp
MSc Biology
BA Fine Art
BSc Maths

Paul
Jenny
Sarah
Andy
Jim
Anne

Examples of the
"Course" entities

the "is_studied_by"
relationship

Examples of the
"Student" entities

Confirming Correctness

Examples of the "Course" entities — the "is_studied_by" relationship — Examples of the "Student" entities

- Use the diagram to show all possible relationship scenarios.
  - Go back to the spec and check to see if they are allowed.
  - If not, then put a cross through the forbidden relationships
- This allows you to show the cardinality and optionality of the relationship

Deriving the relationship parameters

To check we have the correct parameters (sometimes also known as the degree) of a relationship, ask two questions:

- One course is studied by how many students? Answer => 'zero or more .
  - This gives us the degree at the 'student end.
  - The answer 'zero or more needs to be split.
    - The 'more' means that the cardinality is 'many .
    - The 'zero' means that the relationship is 'optional .
  - If the answer was 'one or more', then the relationship would be 'mandatory'.

Relationship parameters cont...

- □ One student studies how many courses? Answer => 'One
  - ◘ This gives us the degree at the 'course end of the relationship.
    - ■ The answer 'one' means that the cardinality of this relationship is 1, and is 'mandatory
  - ◘ If the answer had been 'zero or one', then the cardinality of the relationship would have been 1, and be 'optional'.

Redundant relationships

Some ER diagrams end up with a relationship loop.

- Check to see if it is possible to break the loop without losing information.
- Given three entities A, B, C, where there are relations A-B, B-C, and C-A, check if it is possible to navigate between A and C via B. If it is possible, then A-C was a redundant relationship.
- Always check carefully for ways to simplify your ER diagram. It makes it easier to read the remaining information.

Redundant relationships example

Consider entities 'customer' (customer details), 'address' (the address of a customer) and 'distance' (distance from the company to the customer address).
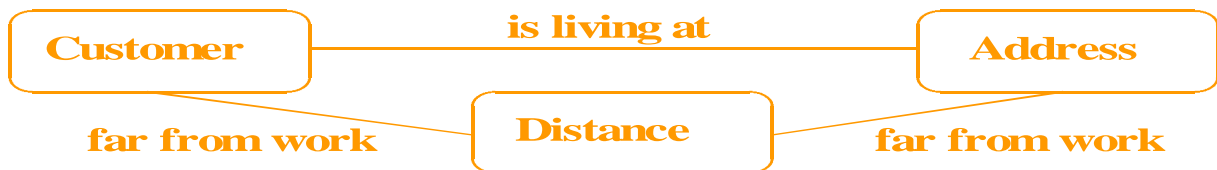
Splitting n:m Relationships

A many to many relationship in an ER model is not necessarily incorrect. They can be replaced using an intermediate entity. This should only be done where:

☐ the m:n relationship hides an entity

☐ the resulting ER diagram is easier to understand.

Splitting n:m Relationships - example

Consider the case of a car hire company. Customers hire cars, one customer hires many cars and a car is hired by many customers.

```
┌──────────┐  m        hire        n  ┌──────────┐
│ Customer │>─────────────────────────│   Car    │
└──────────┘                          └──────────┘
```

The many to many relationship can be broken down to reveal a 'hire' entity, which contains an attribute 'date of hire'.

```
┌──────────┐  m     ┌──────────┐  n    ┌──────────┐
│ Customer │────────│   Hire   │───────│   Car    │
└──────────┘        └──────────┘       └──────────┘
```

Entity Relationship Modelling - 2

Overview

□ construct an ER model

□ understand the problems associated with ER models

□ understand the modelling concepts of Enhanced ER modelling

Country Bus Company

A Country Bus Company owns a number of busses. Each bus is allocated to a particular route, although some routes may have several busses. Each route passes through a number of towns. One or more drivers are allocated to each stage of a route, which corresponds to a journey through some or all of the towns on a route. Some of the towns have a garage where busses are kept and each of the busses are identified by the registration number and can carry different numbers of passengers, since the vehicles vary in size and can be single or double-decked. Each route is identified by a route number and information is available on the average number of passengers carried per day for each route. Drivers have an employee number, name, address, and sometimes a telephone number.