Unit III
Relational Database Design

Relational Database Design

Database design is a three-step process. It starts with functional analysis which is responsible for modeling a real world process in terms of the functions performed and the data which supports these functions. Functional analysis identifies data and the business rules which are to be applied to the process.

The next step is called data analysis. Data analysis can be further broken down into three phases, namely conceptual design, logical design, and physical design. This unit concentrates only on the first two phases.

In the conceptual design phase, we take the data and business rules identified by functional analysis and use them to create a data model. In Module 6, we will study how to create a conceptual data model using the Entity-Relationship Model.

Modules 7 and 8 deal with the design of the preliminary logical model and the rules of normalization, respectively. In the logical design phase, the conceptual data model is mapped onto the chosen data model of the DBMS that will be used. We will discuss the designing of a logical model based on the relational data model.

The resulting DBMS-specific database model is used in the subsequent physical design phase, in which the physical features of the database are specified. These features include the form of file organization and the internal storage structures; these were discussed in the previous Unit.

The last step in database design is the physical implementation. In this step, the outcome of the physical design phase is finally implemented to produce the physical database (i.e., database description, program communication block).

Module 6
Conceptual Design

Conceptual design looks at the reality in which the organization wishes to store data and present information. The emphasis is on modeling the data exactly as they are viewed by the organization, rather than how they will be subsequently implemented. This means efficiency and performance issues are not yet considered.

The scope and content of the reality model are identified at this time. Data about the organization have been gathered and corresponding business rules have been laid. On this basis, the conceptual data model is created step-by-step using the Entity-Relationship approach.

Objectives

At the end of this module, you should be able to:

1. List and describe the steps which constitute the conceptual design; and
2. Draw a conceptual model given the description of a real world situation.

Conceptual Data Model

Usually, the very first step in designing a database is the **functional analysis** which is the collection of the data requirements and their analysis. It is in this step that database designers interview prospective database users to understand and document their data requirements. The result of this step is a concisely written set of user's requirements. This set of requirements comes in the form of **business rules** and **functional requirements.** These contain the user-defined operations that will be applied to the database and include operations like retrievals and updates.

Once all the requirements have been collected and analyzed, the next step is called the **conceptual database design**. In this design phase, the goal is to create a **conceptual data model** which is a brief description of the data requirements. It shows detailed descriptions of data types, relationships, and data constraints. All these are expressed using the concepts provided by the data model. Because these concepts do not include any implementation details, they are usually easier to understand and can be used to communicate with non-technical users.

The conceptual data model can also be used as a guide to ensure that all user's data requirements are met and that the requirements do not include any conflicts. This approach enables the database designers to concentrate on specifying the properties of the data without being concerned with storage details.

Entity-Relationship (ER) Model

The **ER Model** is a popular conceptual data model. It is frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

There are several variations to the ER modeling approach. One feature, however, is common–all data may be represented using any one of the following major constructs:

1. **Entity** - This is an abstract or concrete thing in the real world which is of interest to the organization. It is a thing that exists and is distinguishable. An entity may be an object with a physical existence (e.g., PERSON, HOUSE, CAR) or it may be an object with a conceptual existence (e.g., COMPANY, JOB, University COURSE).

2. **Attribute** - A named property or characteristic that describes an entity (e.g., employee NAME, ADDRESS, AGE) or a relationship. A particular entity or relationship will have a **value** for each of its attributes. The attribute values that describe each entity become a primary part of the data stored in the database.

3. **Relationship** - An association between two entities (e.g., person OWNScar)

Top-Down vs. Bottom-Up Approach

There exist two approaches to conceptual design: the **top-down** approach and the **bottom-up** approach. The top-down approach is generally used for designing a new database from scratch. Considerable amount of knowledge of the real world is required to come up with a good design. The other approach, bottom-up, is generally used when existing data is to be organized. Information about the real world is still required to easily identify the relationships among the gathered data.

The top-down approach consists of the following sequence of steps:

- Identify the entities
- Identify the relationships of the entities, and
- Identify the different attributes of the entities.

In the bottom-up approach, the designer is given a collection of existing data that will partially, if not fully, comprise the database. As such, the steps are reduced to the following:

- Identify the roles of the data (i.e., determine if the data are entity types or attributes), and
- Identify the relationships of the entities.

Basic ER Model Concepts

A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute. An entity type defines a set of entities that have the same attributes. A single occurrence of an **entity type** is called an **instance**.

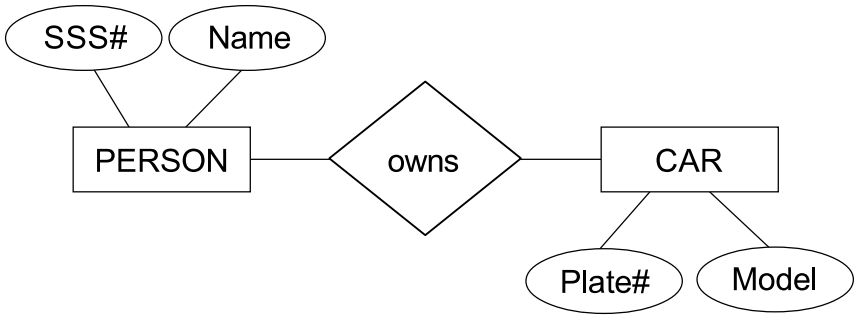
An important constraint on the entities of an entity type is the uniqueness constraint on attributes. An entity type usually has an attribute (or possibly a combination of attributes) whose values are distinct for each individual entity. This attribute is called a candidate key, and its values can be specified to identify each entity uniquely. Usually, entity types have more than one **candidate key**. It is in this set of candidate keys that a primary key is chosen. A **primary key** is a candidate key that has been selected as the identifier for an entity type.

Each attribute of an entity type is associated with a **domain,** which specifies the set of values that may be assigned to that attribute. For example, the domain of an attribute named AGE can be specified as the set of integers between 1 and 100. Similarly, we can specify the domain of attribute NAME as the set of strings of alphabetic characters.

Entity-Relationship Diagram (ERD)

An **entity-relationship diagram** is a graphical portrayal of entities and their relationships. It is similar to the structure shown below, but it supports the representation of more general relationships.

Example:



The above diagram shows the OWNS relationship PERSON:CAR. The entity types PERSON and CAR are shown in rectangular boxes. The relationship OWNS is shown in a diamond shaped box attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type by a straight line. Note that each primary key attribute has its name underlined inside the oval.

SAQ 6-1

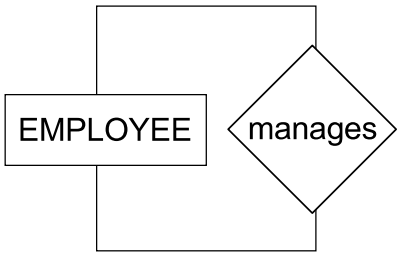
I've picked-up a list of words from an existing ER diagram. The list includes: Address, Age, Course, Course#, Day, Enrolls, Name, Sex, Student, Student#, Time, Title. Can you reconstruct the ERD for this set of data? You can pattern your answer from the example given above. Draw your ERD on the space provided.

Degree of a Relationship

The **degree of a relationship** is defined as the number of entity types that participate in a relationship. Hence, the relationship ENROLLS shown in the figure above is of degree two. A relationship with degree one is called unary or recursive, a relationship of degree two is called binary, and one of degree three is called ternary. Here are some examples of each kind.

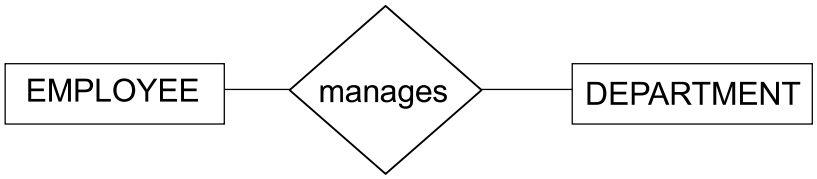
Example 1:

The diagram on your right shows an example of an entity type which has a recursive relationship and can be interpreted as follows: in a company, a boss being an employee has subordinates whom he manages. And recursively, the same boss reports to an immediate supervisor who manages him.



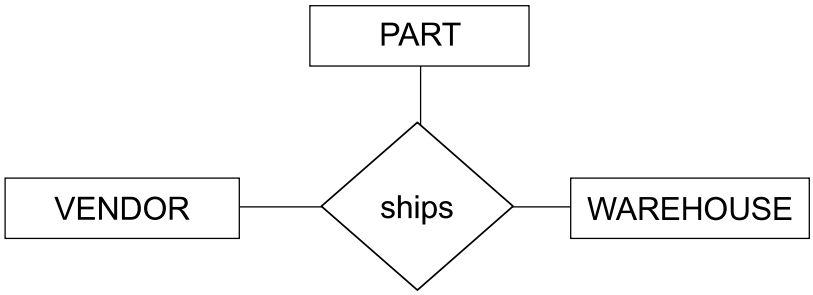
Example 2:

The diagram below is an example of a binary relationship. In this ERD, an employee manages a department or conversely, each department is managed by an employee.



Example 3:

Here is an example of a ternary relationship among the entity types PART, VENDOR, and WAREHOUSE. Each vendor ships a (automobile) part taken from a particular warehouse. Or a part taken from a specific warehouse is shipped by a vendor. Or still, from a warehouse, a specific part is taken by a vendor to be shipped. Any where you start the diagram should be interpreted in the same manner.



Activity 6-1

Aside from the different examples given above, can you think of other scenarios where a unary, binary or ternary relationship exists among entity types? You can apply this concept to situations in your workplace or even to the confines of your home. Try to list them down and draw the corresponding ERD. You can pattern them on the examples above. See if you can come up with at least one example for each type. I allotted the next page for your drawings. Make your drawings neat; we will be using them for the next activity.

Unary Relationship

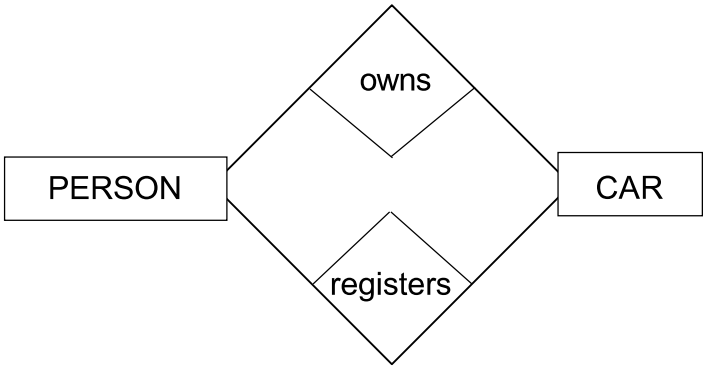
Binary Relationship

Ternary Relationship

Multiple Relationships

It is possible for two entity types to maintain multiple relationships. In the example below, the entity types PERSON and CAR have the relationships OWNS and REGISTERS. That is, a person can own a car (a car can be owned by a person) and a person registers a car (a car is registered by a person).

Example:



Cardinalities in Relationships

Relationships usually have constraints that limit the possible combinations of entities that may participate in the relationship. For example, the diagram above is limited to the possibility that a PERSON owns one or more CARs, but a CAR can only be owned by one PERSON.

Given entity types A and B, the **cardinality ratio** specifies the number of instances of entity type B that can (or must) be associated with each instance of entity A. The OWNS binary relationship PERSON:CAR is of cardinality ratio 1:N, meaning that each person can be related to numerous cars, but a car can be related to only one person. Common cardinality ratios for relationships are one-to-one (1:1), one-to-many (1:N), and many-to-many (M:N).

An example of a 1:1 relationship is MANAGES (Example 6.3), which relates a department entity to the employee who manages that department. This represents the constraints that an employee can mange only one department and that a department has only one manager.

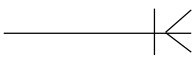
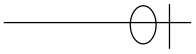
How about the relationship ENROLLS in the ERD in ASAQ 1.1? It is an M:N relationship if the rule is that a student can enroll in several courses and that several students can enroll in a course. This seems to be a real world situation, isn't it?

Minimum and Maximum Cardinality

The **minimum cardinality** is the smallest number of instances allowed to participate in a relationship. On the other hand, the maximum cardinality is the greatest number of instances allowed to participate in a relationship.

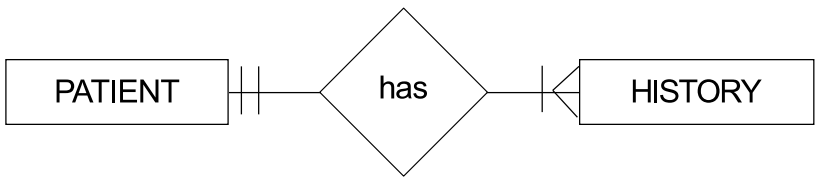
The minimum cardinality value allowed for a relationship is zero or one. In general, if the minimum cardinality is zero, the participation is said to be **optional.** If the minimum cardinality is one, the participation is said to be **mandatory.**

The following are the different notations used for specifying the minimum and maximum cardinalities:

Notation	Meaning
	Mandatory (One)
	Mandatory (One or Many)
	Optional (One)
	Optional (Zero, One or Many)

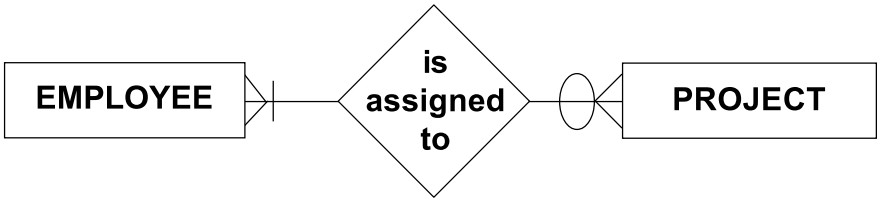
Example 1:

The relationship HAS below is an example of a relationship with mandatory cardinalities. The minimum and maximum cardinalities on the PATIENT side are both one and on the HISTORY side are one and many. The corresponding rule perhaps is that once a patient exists, then that patient must have at least one history, possibly more. On the history side, if a history exists, then that history must be associated to one and only one patient.

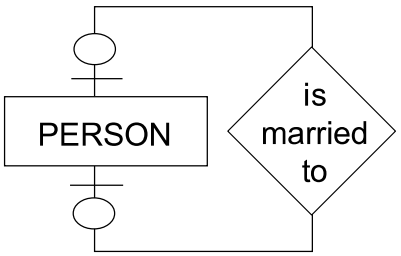


Example 2:

IS ASSIGNED TO is a relationship which has one optional and one mandatory cardinality. The corresponding rule is that an employee is assigned to a number of projects, possibly none and that given a project, there should be at least one employee (possibly many employees) assigned to it. The minimum and maximum cardinality on the EMPLOYEE side is one and many, while it is zero or many on the PROJECT side.



Example 3:



The diagram on your left is an example of a relationship with optional cardinalites. It simply implies that a person can be married to a minimum of zero person or a maximum of one person. Of course, this rule is true only for countries like the Philippines where polygamy, besides being immoral, is considered viewed as illegal.

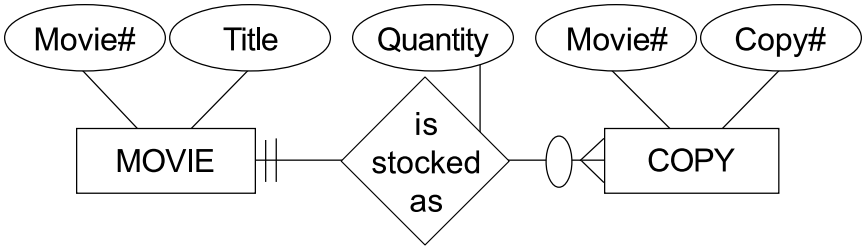
Activity 6-2

Do you still remember the entity-relationship diagrams you drew when you did Activity 6-1? Go back to that page and this time, with just a few strokes of your pen, indicate the minimum and maximum cardinalities of the relationships. There are four possible combinations. You can choose among these combinations that which you think is most appropriate for the relationships.

Existence Dependency and Weak Entity

An **existence dependency** occurs when an instance of one entity cannot exist without the existence of an instance of some other (related) entity. In Example 6.6, the entity type HISTORY has an existence dependency on the entity type PATIENT. That is, an instance of HISTORY cannot exist unless the related PATIENT entity exists. We say that the entity type HISTORY is a **weak entity**. A weak entity is an entity type that has an existence dependency. Below is another example of a scenario where an existence dependency exists between entities.

Example:



The ERD above shows an entity type named MOVIE which has a mandatory one cardinality relationship with entity type COPY. The ERD suggests that an instance of COPY cannot exist if the related MOVIE instance does not exist. It does make sense, doesn't it? For how can a copy of a movie exist when no such movie exists? Also, the diagram contains an existence dependency with the entity type COPY being the weak entity.

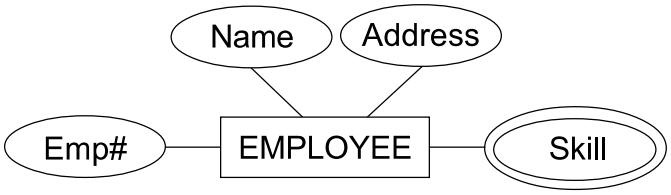
The relationship IS STOCKED AS is an example of an **identifying relationship**. An identifying relationship is a relationship in which the primary key of the parent entity is used as part of the primary key of the dependent entity. Identifying relationships usually exist whenever existence dependency occurs between entities.

Did you notice that the attribute QUANTITY is associated with the relationship and not with any of the entities? Relationships are also allowed to have attributes. In the case above, QUANTITY tells how many copies each instance of a MOVIE has.

Modeling Multivalued Attributes

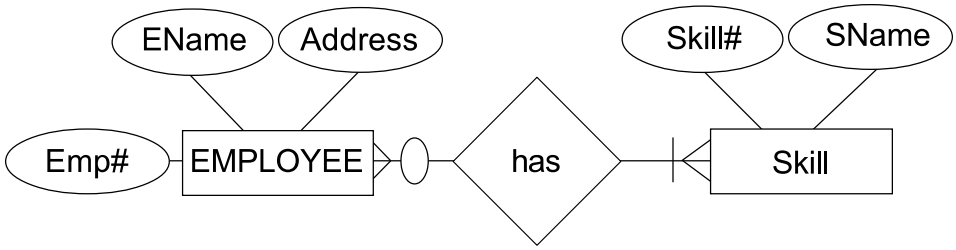
It is possible for an entity type attribute to have more than one value for each entity instance. In this case, the attribute is called a multivalued attribute. Here is an example of an entity with a multivalued attribute:

Example:



The entity type EMPLOYEE has the attribute Emp# as its primary key. Other keys include Name, Address and Skill. Skill is a special kind of attribute. Special because it is a multivalued attribute. This means that an instance of an EMPLOYEE can have more than one value for the attribute SKILL. Or we can simply say that an employee may possess more than one skill.

Multivalued attributes are usually modeled differently in a relational database model. When an attribute is identified to be multivalued, it is usually modeled as shown below. The multivalued attribute is removed as an attribute of the original entity type and is made into a separate entity type and a relationship HAS is established. You will learn the reason for this when we reach the topic on Logical Design. For now, study the ERD below and compare it with the previous diagram. Can you see the resemblance?



The entity type EMPLOYEE still has the attributes EMP#, ENAME and ADDRESS associated with it like in the original ERD. The attribute SKILL became a separate entity type which acquired its own attributes, namely SKILL# and SNAME. The existing relationship HAS is a binary many-to-many type. Expounding on this, the diagram suggests that an EMPLOYEE has at least one SKILL and that it is possible for a particular SKILL not to be possessed by anyone.

Repeating Group

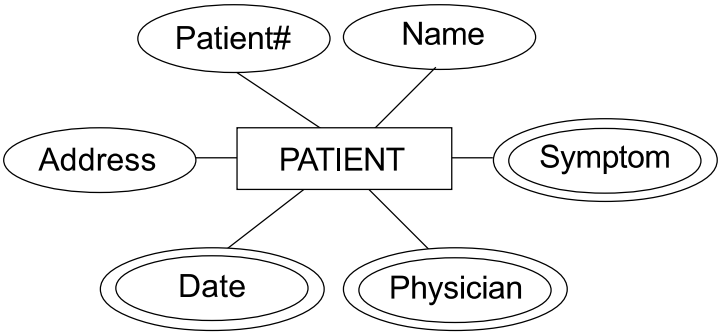
Now, we extend the concept of multivalued attributes further and discuss **repeating groups.** A repeating group is a set of two or more multivalued attributes that are logically related. Consider the sample data below which shows a patient's chart records in a hospital.

PATIENT CHART

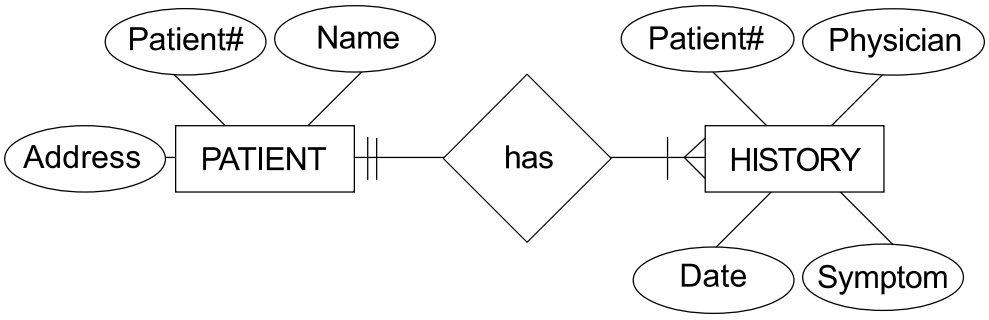
Patient Number : 03-6379
Patient Name : Michael Smith
Address : 45 Ipil Road, Los Baños, Laguna

Date of Visit	Physician	Symptom
03-20-1998	Ryan	Sore Throat
04-07-1998	Steve	Fever
…	…	…

We can represent this chart by using only one entity type as shown below. Based on the chart, the entity type will have six attributes, three of which are multivalued.



Since the three multivalued attributes are related, this set can be considered as a repeating group. In the same manner that a multivalued attribute is removed from being an attribute of the original entity, the repeating group will also be removed and a separate entity type will be created where these attributes will be associated. In the ERD below, the entity type HISTORY is introduced and the attributes DATE, PHYSICIAN, and SYMPTOM are associated with it.



As previously explained, the entity type HISTORY is a weak entity and thus an identifying relationship exists between HISTORY and PATIENT. As such, the primary key of PATIENT which is Patient# is used as part of the primary key of HISTORY.

More Examples

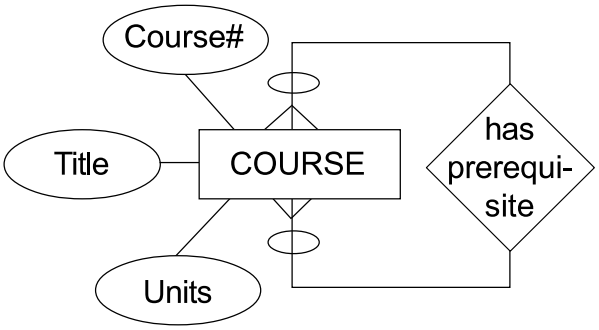
We've come to the end of this module. You have learned that the expected output from the conceptual design phase is the conceptual model which is based on facts gathered about the real world situation. Having learned all the different ER model concepts , you are now ready to design your own conceptual data model. The next two examples are simple in that they are limited to at most one relationship. In the real world, the number of relationship and entities could number to ten or more. Still, let these examples serve as your guide in designing.

Example:

A university has a large number of courses in its catalog. A course is described by a course number, course title and an equivalent number of units. Each course may have one or more other courses as prerequisites, or may have no prerequisites.

The situation suggests that there is one entity type, let's name it COURSE, and that this entity type possess the following attributes: course number (COURSE#), course title (TITLE) and units (UNITS). Now, what could be the relationship in this situation? The last statement above gives us a hint - each course may or may not have a prerequisite. We can then use PREREQUISITE as the relationship.

Given all the above information, all we need to determine now is the cardinality of the relationship. We know that it is a unary relationship for there's only one entity type present. But is it a one-to-one, one-to-many or many-to-many kind of a relationship? The diagram below will answer this question.

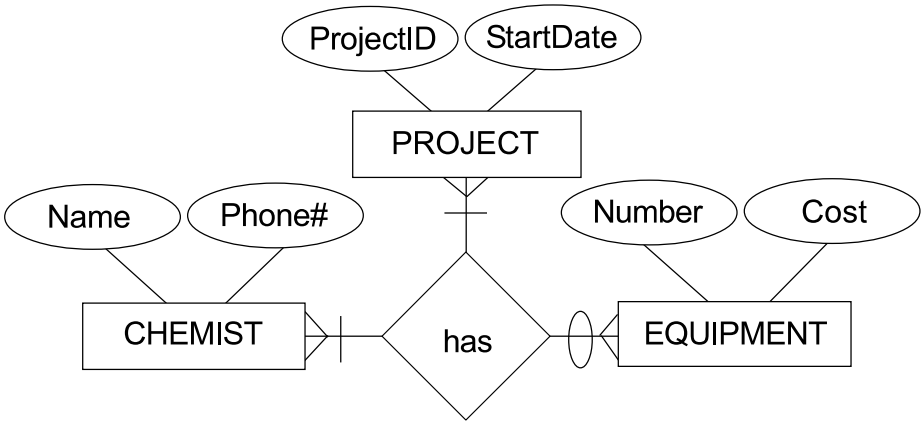


Since it is described in the situation that each course may have one or more other courses as prerequisites, or may have no prerequisites, then the proper cardinality of the relationship is many-to-many. Notice that the minimum cardinality for both sides is zero.

I'll leave the next example as a preliminary exercise before you put your hands on the next SAQ. Just remember, for each data item, identify whether it is an entity type, an attribute or a relationship. Then draw the ERD corresponding to the information you've gathered and finish it off by supplementing the details (i.e., cardinality of the relationship).

Example:

A laboratory has several chemists who work on various projects and who may use certain kinds of equipment on each project. A chemist has a name and phone number. A project has a project ID and a start date. An equipment has the attributes number and cost.



SAQ 6-2

It is now time to integrate everything you have learned. Allot more time than usual for this SAQ for you will be asked to design a conceptual model based on a real world situation. The business rules and the functional requirements are given below. I suggest that you write the list of data that you'll identify on the space provided and draw your ERD on a separate sheet of paper. Are you ready?

Draw an ER Diagram for the following situation:

1. The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. The company keeps track of the start date when that employee began managing the department. A department has several locations.

2. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

3. The company stores each employee's name, social security number, address, salary, sex, and birthday. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. The number of hours per week that an employee works on each project is also kept. The direct supervisor of each employee is also recorded.

4. The company wants to keep track of the dependents of each employee for insurance purposes. Data on each dependent's name, sex, birthday, and relationship to the employee is also stored.

This is the last SAQ for this module. I suggest you look for more problems in the references listed. It takes lots of practice to perfect this skill.

Summary

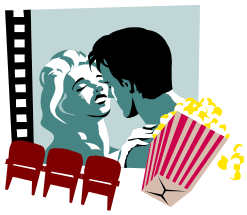
In this module, you learned the different steps in creating a conceptual database model. There are two approaches, namely, top-down and bottom-up. Both approaches employ the basic step, which is the classification of data either as an entity, an attribute, or a relationship.

You were introduced to the different concepts used by the ER model. Some of the basic concepts are entity type, instance, candidate key, primary key, domain. A list of the more advanced concepts include the degree of a relationship, cardinality ratio, existence, dependency, and weak entity, to name a few.

Of course, who can forget the ER diagram (ERD) which graphically represents the ER model. For all the different concepts you've learned, there is a specific notation available to represent it. Can you still visualize each? You need not worry; you can always go back through the pages of this module for a review.

That's it for this module. The next stop is logical design. But before we proceed, stop by the ice cream stand and eat your favorite cobbler. After all this hard work, you deserve a break.

I have a better idea: Why not grab a box of popcorn and soda and watch that critically acclaimed movie now showing. That should serve as a good breather. Don't forget to bring your favorite companion along. Relax and have fun while watching a movie.



References

Adamski, J and P. Pratt. 1991. *Database systems: management and design*. Boston, Massachusetts: Boyd & Fraser Publishing Co.

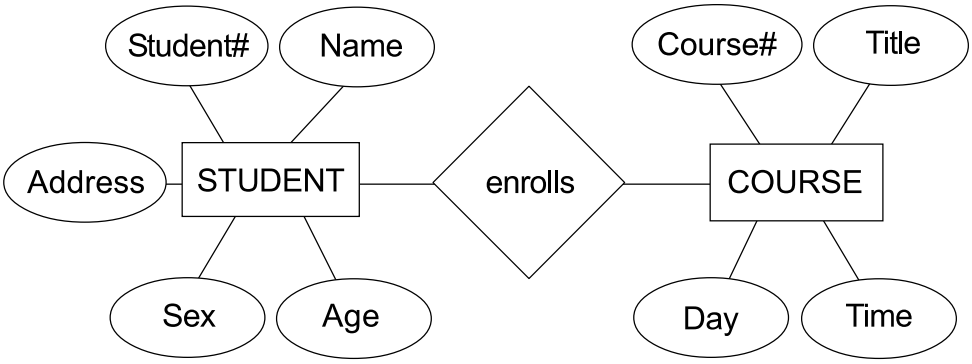
Elmasri, R. and SB Navathe. 1994. *Fundamentals of database systems*. Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc.

Ullman, JD. 1982. *Principles of database systems.* Rockville, Maryland: Computer Science Press, Inc.

Answers to Self-Assessment Questions

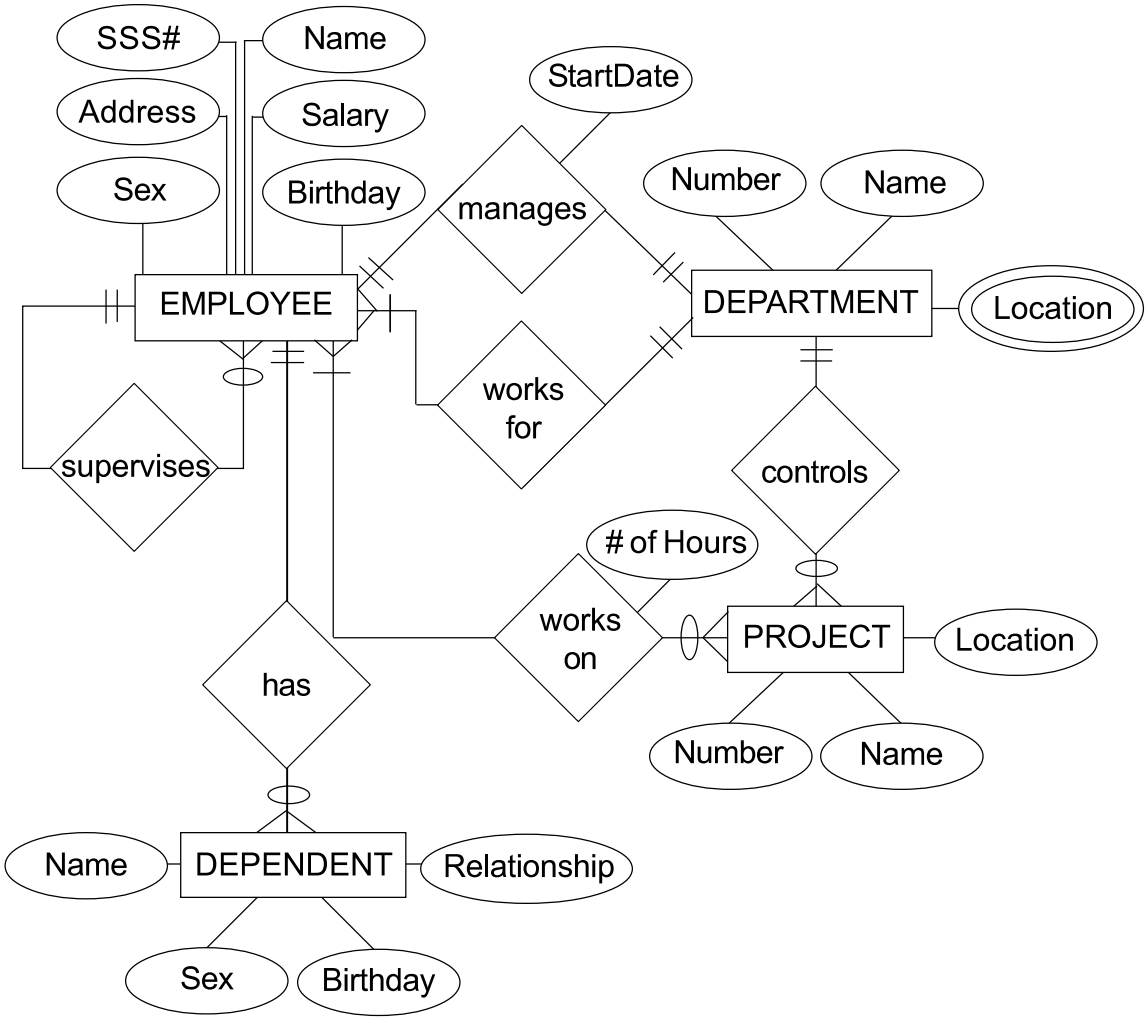
ASAQ 6-1

Basically you should have identified the relationship ENROLLS and the two entity types STUDENT and COURSE. That is a student enrolls in a course or a course is enrolled by a student. The entity type STUDENT is described by the attributes Address, Age, Name, Sex, Student#, while COURSE is characterized by Course#, Title, Day and Time. If you identified Student# and Course# as the primary keys of STUDENT and COURSE, respectively, then you are correct. You should have drawn a diagram which looks like the one below.



ASAQ 6-2

You should have drawn an ERD similar to the one below. Pay special attention to the attributes associated with relationships as these are the most commonly misrepresented in a conceptual model. Also, check if the cardinalities of the relationships are appropriate. I might have overlooked something. It is possible that you came up with an ERD that is different from the one below. That is acceptable, but then again check if it really follows the rules and requirements given in the problem. I will no longer explain the ERD in detail for it should be easily interpreted.



Module 7
Logical Design

Logical design is a process of transforming the conceptual data model into a logical database model. A logical database model is simply a design that conforms to the data model for a class of database management system. You have learned in the first module about the different data models, namely relational, hierarchical and network. In this module, we'll concentrate on the relational data model.

The logical design of the relational data model consists of the following phases: [1] transforming the conceptual model to a preliminary logical data model; and [2] optimizing the logical data model by applying the rules of normalization. These two phases require much discussion and so I allotted two modules to cover them. The designing of the preliminary logical data model is discussed in this module and the discussion on normalization is reserved for the next module.

Objectives

At the end of this module, you should be able to:

1. List and describe the steps in designing a logical database model; and
2. Create a preliminary logical database model given a conceptual database model.

Relation

The most important construct in a relational data model is the **relation.** A relation is a named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows. A **tuple** corresponds to a row in a relation. The number of tuples is called the **cardinality** and the number of attributes is called the **degree.**

In the relational data model, a **primary key** is a unique identifier for the table. It is a column or a column combination with the property that, at any given time, no two rows of the table contain the same value in that column or column combination.
Example:

The structure below shows an example of an EMPLOYEE relation. Each tuple in the relation represents a particular employee entity. The relation is displayed as a table, where each tuple is shown as a row and each attribute corresponds to a column header indicating a role or interpretation of values in that column. **Null values** represent attributes whose values are unknown or do not exist for some individual employee tuples. The attribute Employee# is the primary key of the relation.

Employee#	Name	Department	Phone	Salary
88-100	Margaret Forget	Marketing	749-1253	42,000
90-110	Allen Beeton	Accounting	null	39,000
97-140	Chris Lucero	Finance	749-6492	41,500
72-190	Lorenzo Davis	Finance	null	68,000
85-150	Susan Martin	Marketing	null	38,500

Relation schema

A **relation schema** R, denoted by R (A_1, A_2, ..., A_n), is made up of a relation name R and a list of attributes A_1, A_2, ..., A_n. A relation schema is used to describe a relation and R is called the **name** of this relation. Again, the degree of a relation is the number of attributes n of its relational schema.

Example:

The EMPLOYEE relation above is described by the schema

EMPLOYEE (Employee#, Name, Department, Phone, Salary).

Properties of a relation

Relations possess certain properties that distinguish them from any common table. I listed these properties below. We shall discuss each of these properties shortly.

A relation has the following properties:

1. Entries in columns are **atomic**.
2. Entries in columns are from the **same domain**.
3. Each **row is unique.**
4. The **sequence of the columns** is insignificant.
5. The **sequence of rows** is **insignificant**.

Entries in columns are atomic. An entry at the intersection of each row and column is atomic or single-valued and never a list of values. There can be no multivalued attributes or repeating groups in a relation. This is the main reason why multivalued attributes are discouraged when designing the conceptual model of a database.

Entries in columns are from the same domain. Domain refers to the definition of the types and ranges of values that attributes may assume. This is the same definition the conceptual data model uses. If a column in a relation is defined to contain integer values, then a relation instance must have an integer value for that specific column.

Each row is unique. This property follows the property of a mathematical set which states that a set does not include duplicate elements. This means that there should be no duplicate rows if a relation is viewed as a set.

An important corollary of the fact that there are no duplicate tuples is that there is always a primary key. Since tuples are unique, it follows that at least the combination of all attributes of the relation has the uniqueness property, so that at least the combination of all attributes can serve as the primary key.

Sequence of the columns is insignificant. The reason for this property is similar to mathematical sets once more. Like the rows, there is no "1st attribute" or "2nd attribute." The columns are always referred to by their names and never by their position. Thus, the sequence of the columns is insignificant.

Sequence of rows is insignificant. Again, since a relation is viewed as a mathematical set, sets in mathematics are not ordered. Thus, there is no such thing as "the 1st tuple" or "the 100th tuple." In other words, the first tuple could very well be the last tuple and the reordering does not make the table less of a relation.

SAQ 7-1

The following tables are not relations. Can you figure out why? Write your answers on the space provided immediately following each table. When you are done answering, compare your answers with my answers. I've written my answers at the end of this module.

Table A

Company	Number	Name	Type
Midwest Life	108-1234 114-8213	Orthorne Hawthorne	Full Liability
Southeast Life	210-1974	Osborne	Limited

Table A is not a relation because

Table B

X	Y	Z
1	4.3	10%
2	8.9	0.25
3	3.1	0.15

Table B is not a relation because

Table C

A	B	C	D
1	10	3	2
5	6	8	9
3	7	10	15
8	9	10	4
5	6	8	9
1	7	3	2

Table C is not a relation because

Transforming ERD to Relations

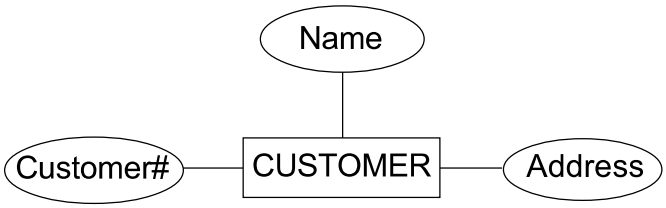
Transforming a conceptual data model (e.g. entity-relationship model) to its equivalent relational preliminary logical model is a two-step process. The first step involves the representation of entities while the second involves the representation of the relationships. We will discuss these steps in detail for the remainder of this module.

Step 1: Representing entities

Representing entities is perhaps the easiest step in the logical design phase. All you need to do is to create a relation whose number of columns is equal to the number of attributes associated with the entity and give the relation the name of the entity type. Formally, for each entity type E in the ERD, create a relation R that includes all the attributes of E.

Example:

In the ERD below, the entity type is named CUSTOMER and it has three attributes, namely Customer#, Name and Address. The primary key is Customer#.



The corresponding relation for this one, together with sample instances, is the table shown below. The relation schema for CUSTOMER is CUSTOMER (Customer#, Name, Address). The relation is composed of three columns, which is the actual number of attributes associated with CUSTOMER. The attributes' names become the column names or headings. Notice that the name of the first column, Customer#, is underlined. This implies that this column contains the identifier or primary key of the relation.

CUSTOMER

Customer#	Name	Address
1	Jason	123 Oak St., NY
2	Oliver	18 Horsey Dr., Baltimore
...	...	...

Any attribute which contains a value that corresponds to an attribute value in another relation is called a **foreign key.** In order to maintain consistency, it is desirable to use the same attribute name in both relations. To identify a foreign key, the attribute is underlined using a broken line. Foreign keys play an important role in representing relationships between entities. You'll come across many examples in the next few topics.

Step 2: Representing relationships

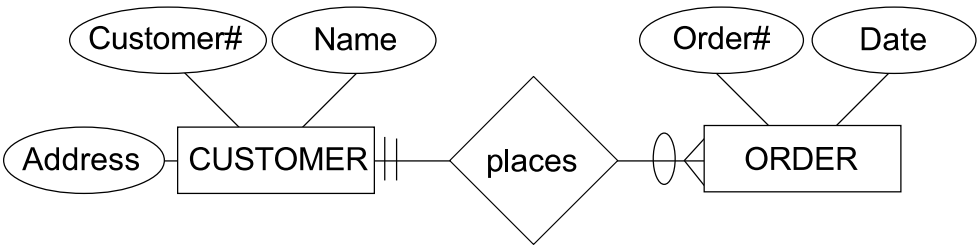
After transforming all the entities into relations, the next step is to represent relationships. Representing a relationship in a conceptual ER model to its equivalent construct in the relational logical model depends on both the degree of the relationship and its cardinality.

Representing binary 1:N relationships

The most common type of relationship and the easiest to represent is a 1:N relationship. Assume that two entities A and B are maintaining a 1:N relationship, we represent this relationship as follows: we simply add the primary key of the entity on the one-side of the relationship as a foreign key in the relation that is on the many-side of the relationship. An example will make this easier to understand.

Example:

Let us link the CUSTOMER entity type in Example 7.2 to another entity type ORDER. Let the relationship have a cardinality ratio of 1:N and let's call this relationship PLACES. The corresponding ERD is shown below.



After representing the two entity type CUSTOMER and ORDER in their equivalent relations, we now map the 1:N relationship PLACES. We maintain the original CUSTOMER relation we created in Example 7.2. But for the ORDER relation, we include the primary key of the CUSTOMER relation as foreign key. The resulting relation schema is ORDER (Order#, Date, Customer#) and the corresponding relation is shown below.

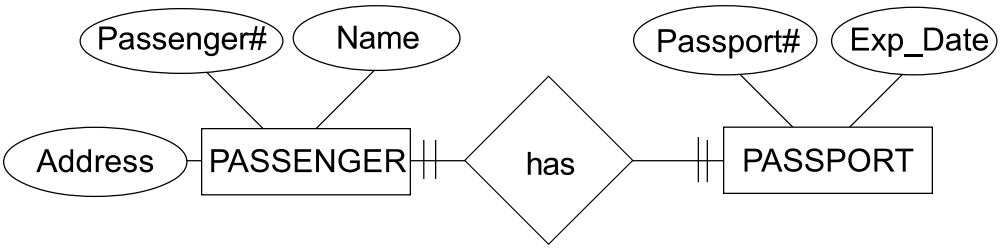
CUSTOMER

Order#	Date	Customer#
1001	03/15/1990	1
1002	03/17/1990	2
1003	04/12/1990	1
...	...	...

Representing binary 1:1 relationships

If two entities are maintaining a one-to-one relationship, and they can share the same primary key (i.e., an identifying relationship exists between them), these may be resolved by combining the entities into one. This means that the attribute of one entity will be moved into the other and a common primary key will be established.

Example:



This ERD tells us that each passenger must have a passport or conversely, each passport must be owned by a passenger. This is clearly a relationship with a cardinality ratio of 1:1. This kind of relationship can be resolved simply by merging the two entity types and the relationship into a single relation. This technique is particularly appropriate when the participation of the entity types are **total** (i.e., an instance of one entity type ensures the existence of an instance of the other entity type) and the entity types do not participate in any other relationships. Thus, the above ERD can be fully represented by the relation schema

PASSENGER (Passenger#, Name, Address, Passport#, Exp_Date).

In cases where the above technique is not admissible (i.e., at least one of the entity types participates in more than one relationship), there is yet another way. In general, if the entities A and B are maintaining a 1:1 relationship, then it can be represented either by:

1. Adding the primary key of A as a foreign key of B, or
2. Adding the primary key of B as a foreign key of A, or
3. Both of the above.

Using the ERD in Example 7.5, we let the entity type PASSENGER be entity A and the entity type PASSPORT be entity B. The resulting relation schema if the above rules are applied respectively are as follows:

1. PASSENGER (Passenger#, Name, Address) and
   PASSPORT (Passport#, Exp_Date, Passenger#), or

2. PASSENGER (Passenger#, Name, Address, Passport#) and
   PASSPORT (Passport#, Exp_Date), or

3. PASSENGER (Passenger#, Name, Address, Passport#) and
   PASSPORT (Passport#, Exp_Date, Passenger#)

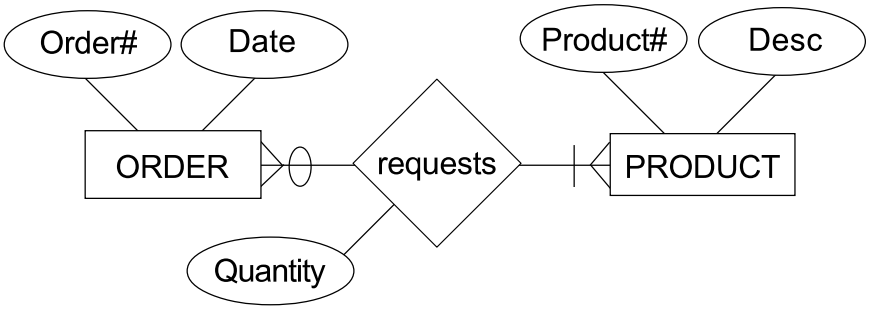
The designer is left with the discretion as to which of the three is most appropriate.

Representing binary M:N relationships

If two entities A and B are maintaining an M:N relationship, create a separate relation to represent the relationship. Include as foreign key attributes in the newly created relation the primary keys of entities A and B. This combination will form the primary key of the new relation. If the relationship has any attributes associated with it, include them as attributes in the new relation. Notice that we cannot represent an M:N relationship simply by adding a single foreign key attribute to one of the participating relations – as we did for 1:1 or 1:N relationships - because of the M:N cardinality ratio.

Example:

In the example below, we map the REQUESTS M:N relationship by creating the relation ORDER_LIST. The primary keys of the participating entity types ORDER and PRODUCT are included as foreign keys in ORDER_LIST. The attribute QUANTITY in the ERD is included as an attribute in the relation ORDER_LIST. The primary key of ORDER_LIST is the combination of the foreign key attributes {Order#, Product#}.



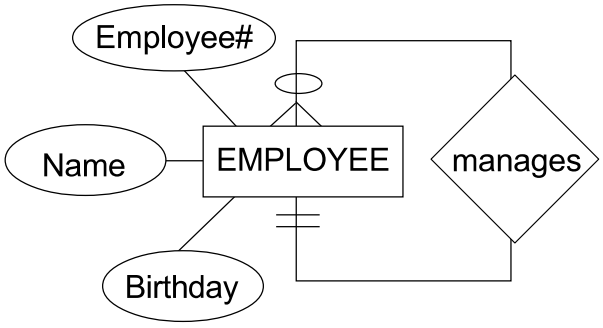
ORDER (Order#, Date)
PRODUCT (Product#, Desc)
ORDER_LIST (Order#, Product#, Quantity)

Representing unary 1:N relationships

As in representing a binary 1:N relationship, we represent a recursive relationship with cardinality ratio 1:N by simply adding a rename of the primary key with reference to the relationship. This is quite a confusing procedure but let's go through the example below together.

Example:

Consider the ERD below. It suggests that an employee manages zero, one or more employees and that an employee is managed by at most one and only one manager. To represent this kind of a relationship, we add the attribute MANAGER_ID of the EMPLOYEE relation's primary key EMPLOYEE#, which is actually just a rename. The value of MANAGER_ID is just the EMPLOYEE# of the employee's manager. MANAGER_ID is considered as a foreign key in the relation. Hence the relation schema is EMPLOYEE (Employee#, Name, Birthday, ManagerID).

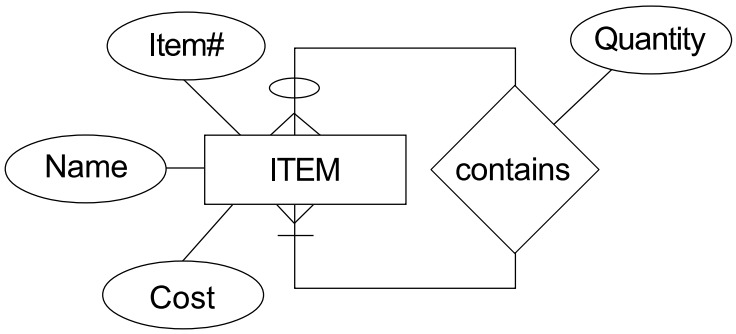


Representing unary M:N relationships

For each unary M:N relationship, create a new relation to represent the relationship. This is similar to representing binary M:N relations. Include as foreign key attributes the primary keys of participating entity types. Since there is only one entity type, we include its primary key and a rename of it as foreign keys. The primary key of the new relation is composed of its set of foreign keys.

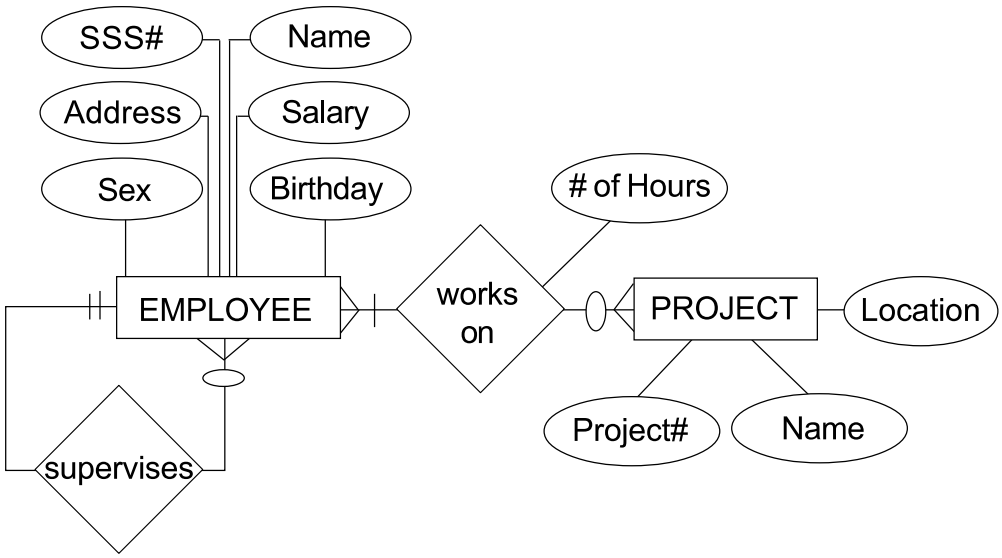
Example:

The next diagram is transformed to its equivalent set of relations: ITEM (Item#, Name, Cost) and ITEM_BILL (Item#, Part#, Quantity).



SAQ 7-2

Do you think you can create the preliminary logical data model of the ERD below? Quickly, grab your pencil and start designing. Do this by following the two-step process we discussed in this module.



Summary

After designing the conceptual model of a database, the next step is to design the logical model. Basically, it has two phases. In this module, I discussed the first phase which is the designing of the preliminary logical model. This phase can be considered as a two-step process. The first step is to represent the entity types and the second step is to represent the relationships of the entity types.

Representing entity types can be done easily. For every entity type, we create a relation corresponding to it which contains all the attributes associated with it. Representing relationships is the more difficult one. The concept of foreign keys was introduced so that relationships can be properly represented. It must be noted that the degree and the cardinality of a relationship is an important factor in its representation.

A binary 1:N relationship can be represented by placing the primary key of the entity type on the one side as a foreign key in the relation on the many side. Relationships between two entity types maintaining a binary 1:1 relationship can be resolved in many ways. If possible, the best way is to combine the two entity types into one. Otherwise, it can be represented by adding the primary key of one entity type to the other, or vice versa or do both. For binary M:N relationships, the creation of a separate relation to represent the relationship is usually called for. This new relation will have as its primary key the set of primary keys of the entity types associated with the relationship being represented.

More or less, the representation of unary and ternary relationships follow from the steps described above. If you want a more detailed discussion of the logical design, you can read them in the references listed.

Whew! That wraps up our discussion on logical design. All I can think of now is to find a good form of recreation. Hey! Have you checked out the latest band playing in the circuit? I heard they are great. Do you think there's any chance we can see them play? Just a thought…

References

Adamski, J. and P. Pratt. 1991. *Database systems: management and design.* Boston, Massachusetts: Boyd & Fraser Publishing Co.

Date, CJ. 1990. *An introduction to database systems*. Reading, Massachusetts: Addison-Wesley Publishing Co., Inc. 5th ed.

Elmasri, R., and SB. Navathe. 1994. *Fundamentals of database systems*. Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc.

Answers to Self-Assessment Questions

ASAQ 7-1

Table A is not a relation because some of its column entries are not atomic or single-valued. In fact, the table contains the repeating group {Number, Name, Type}. Entries in columns of a relation must be atomic. This is the first property of a relation.

Table B is not a relation because entries under column Z do not come from the same domain. An entry is written in percentage format while the rest are in decimal format. By the second property, this table cannot be considered as a relation.

Table C is not a relation. Compare the second and the fifth rows. Did you notice that these two rows are identical? If not, look again. Remember, the third property stipulates that there should be no duplicate tuples in a relation.

How many correct answers did you get? Three ☺? Two 😐? One ☹?

If you got three correct answers, collect★★★★★.
If you got two correct answers, collect ★★★.
If you got only one correct, collect ★.
No correct answer, no ★. Review the text and try the SAQ once more.

ASAQ 7-2

The first step is to represent the entity types. In the given ERD, there are two entity types (i.e., EMPLOYEE and PROJECT). Following the steps in representing entities, our logical model initially includes the following relations:

EMPLOYEE (SSS#, Name, Address, Sex, Birthday, Salary)
PROJECT (Project#, Name, Location).

The next step is to represent the relationships. The ERD contains two relationships, namely WORKS and SUPERVISES. Analyzing these relationships further, you'll notice that WORKS is a binary M:N relationship and SUPERVISES is a unary 1:N relationship. To represent WORKS, we introduce a new relation defined below:

WORK (SSS#, Project#, Hours)

The primary keys of EMPLOYEE and PROJECT form the set of attributes that comprises the primary key of the table.

For the SUPERVISES relationship, if you reviewed the discussion in representing this kind of relationship, you would have handled this by simply adding a new attribute, let's say 'SupervisorID' to the EMPLOYEE relation as shown below:

EMPLOYEE (SSS#, Name, Address, Sex, Birthday, Salary, **SupervisorID**)

Thus, the final set of relations that translates the given conceptual ER model to its equivalent logical relational model is as follows:

EMPLOYEE (SSS#, Name, Address, Sex, Birthday, Salary, Supervisor_ID)
PROJECT (Project#, Name, Location)
WORK (SSS#, Project#, Hours).

How many relations did you get correctly? Three☺? Two 😐 One☹?

If you got three correct, collect ★★★★★.
If you got two correct, collect ★★★.
If you got only one, collect ★.
None, no ★. But don't worry. This is not an easy task. Review the process and see if you can collect more ★ next time.

Module 8
Normalization

Not all relational database designs are equal because the notions of entity types and relationships are not precise. It is possible to define a set of entities and the relationships among them in a number of different ways. Thus, we are often faced with a choice among alternative sets of relation schema. Some choices are preferred to others for various reasons. Still, a design that meets the user's needs is better than one that does not, but there are other criteria as well.

In this module we will discuss the next phase in logical design, which is the normalization process. This is actually one of the most interesting topics in database design. The theories behind normalization have been developed to create the preferred well-structured relations. We will study these theories which will help us to structure new and evaluate existing relational database designs. Study this module well. Enjoy!

Objectives

At the end of this module, you should be able to:

1. Identify functional dependencies in a relation;
2 Explain the steps in the normalization process; and
3. Apply the rules of normalization on existing relations.

Normalization and Functional Dependency

Normalization is a process that involves the elimination of inappropriate functional dependencies in a relation. It takes a relational schema through a series of tests to certify whether or not it belongs to a certain **normal form**. But before we discuss the different normal forms, we need first to examine that very important concept called functional dependency.

A **functional dependency**, denoted by X → Y, is a relationship between attributes. Attribute Y is said to be functionally dependent on attribute X if the value of X determines the value of Y. Another way of saying this is that if we know the value of X, we can determine the value of Y. Attribute X is called a **determinant** of attribute Y.

Example 1:

Suppose we rent an apartment for P5,000 per month. If we know how many months we have lived in the apartment (M), then we know how much rent we have paid (P5,000 times M). In this case, we say that Total_Rent_Paid is functionally dependent on Months_In_Residence. Illustrating, we write

Months_In_Residence → Total_Rent_Paid

We call Months_In_Residence a determinant of Total_Rent_Paid or say that Total_Rent_Paid is determined by Months_In_Residence.

Example 2:

Consider the EMPLOYEE relation schema below,

EMPLOYEE (Employee#, Name, Address, Birthday)

In this relation, the employee number (Employee#) of a particular employee (in an organization) determines the values of the attributes Name, Address and Birthday. That is, if we know the employee's number, we can readily determine his name, address and birthday by looking it up in the EMPLOYEE relation. In this case, the primary key Employee# is the determinant of all other non-key attributes. We illustrate the dependency as follows:

Employee# → Name, Address, Birthday

SAQ 8-1

Now let's see if you can determine two functional dependencies existing in the ACTIVITIES relation below. Write them down on the space provided and check if your answer is the same as mine. I listed my answer at the end of this module.

ACTIVITIES

Student#	Activity	Fee
100	Skiing	200
100	Golf	65
150	Swimming	50
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Did you get both functional dependencies? If you did, then you're going to have an easy time with this module. Otherwise, you may find the rest of the module challenging. Hopefully you'll get the hang of it as we move on to the next topic in this module. Shall we proceed?

It was Dr. Edgar F. Codd, the originator of the relational model, who first proposed three normal forms which he called first, second and third normal form. All these forms are based on the functional dependencies existing among attributes of the relation. Later, other normal forms were proposed such as Boyce-Codd normal form, fourth and fifth normal forms which are based on other concepts as well. We will discuss these different normal forms in the succeeding sections.

First normal form (INF)

The **first normal form** is now considered to be part of the formal definition of a relation; historically, it was defined to disallow multivalued attributes. It states that the domains of attributes must include only **atomic** (simple, indivisible) **values** and that the value of any attribute in an instance must be a **single value** from the domain of that attribute.

Example:

The DEPARTMENT relation below is an example of a relation that is not in 1NF. The reason is that the attribute Location is multivalued. Consider Department# 5; the department can be located in either BelAir, Otis or Alabang. This is not allowed in 1NF.

DEPARTMENT

Name	Department#	Location
Research	5	{Alabang, BelAir, Otis}
Administration	4	{Quezon City}
Headquarters	1	{Makati}

To normalize the DEPARTMENT relation into 1NF relations, we break up its attributes into the two relations DEPARTMENT and DEPT_LOCATIONS.

DEPARTMENT

Name	Department#
Research	5
Administration	4
Headquarters	1

DEPT_LOCATIONS

Department#	Location
1	Makati
4	Quezon City
5	Alabang
5	BelAir
5	Otis

The idea is to remove the attribute Location that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Department# of DEPARTMENT. The primary key of this new relation is the combination {Department#, Location} as shown above.

SAQ 8-2

Take a deep breath. Relax. Now, is the relation given below in 1NF? If not, follow the technique used in decomposing the relation in the example given.

EMP_PROJ

Employee#	EmpName	Project #	Hours
123	Smith, John B.	1	32.5
		2	7.5
251	Gray, Ramesh K.	3	40.0
353	English, Joyce A.	1	20.0
		2	20.0
433	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
977	Zelaya, Alicia J.	30	30.0
		10	10.0

Are you getting the hang of it? If you have another 15 minutes to spare then you can still finish the next topic. You can continue with the module and learn more.

Second normal form (2NF)

A relation is in **second normal form** if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key.

A relation that is in 1NF will be in 2NF if **any one** of the following conditions apply:

1. the primary key consists of only one attribute;
2. no non-key attribute exists in the relation; or,
3. every non-key attribute is functionally dependent on the full set of primary key attributes.

Given the first two conditions, we can readily determine if a relation is in second normal form. However, for those cases under the third condition, a careful analysis of the dependencies existing in the relation is needed before a conclusion can be made.

Example 1:

The SALES relation below is in 2NF.

SALES

Customer#	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7942	Hobbs	Smith	South
6837	Tucker	Hernandez	East
7018	Arnold	Faulb	North

Since the SALES relation's primary key is composed of only one attribute, Customer#, then by the first condition we can readily say that the above relation is in 2NF.

Example 2:

The given relation STDADV about students and advisers is in 2NF. This conclusion can be made because the relation's primary key is composed of all the attributes in the relation. Thus, no non-key attribute exists in this relation.

STDADV

StudentID	Adviser
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

Example 3:

Analyze the given EMPLOYEE relation schema.

EMPLOYEE(Employee#, Name, Department, Salary, Course, Date_Completed)

Consider further that a course can be offered many times in a year and that an employee can attend many different courses. Thus, from the above relation, we can assume the following dependencies:

Employee#, Course → Name, Department, Salary, Date_Completed
and
Employee# → Name, Department, Salary

Again, the first dependency, is trivial for the determinant is composed of the set of attributes which comprises the primary key. The other dependency is derived intuitively.

Also, the second dependency shows the attributes Name, Department and Salary as partially functionally dependent on the primary key {Employee#, Course}. This violates the last rule on 2NF which states that every non-key attribute should be fully functionally dependent on the set of primary key attributes.

To make the EMPLOYEE relation into 2NF relations, it is decomposed as follows:

EMPLOYEE(Employee#, Name, Department, Salary)
and
EMP_COURSE(Employee#, Course, Date_Completed)

The first relation is based on the functional dependency Employee# → Name, Department, Salary. The second relation is based on the trivial dependency with the partially dependent attributes removed.

SAQ 8-3

The EMP_PROJ relation below is in 1NF but is not in 2NF. List down the different (partial or full) functional dependencies you can think of. Based on your list, what should be the equivalent set of relations in 2NF?

EMP_PROJ(Employee#, Project#, Hours, EmpName, ProjName, ProjLocation)

Third normal form (3NF)

A relation is in **third normal form** if it is in 2NF and no transitive dependencies exist.

Transitive dependency is a functional dependency between two (or more) non-key attributes in a relation. A functional dependency X → Y in a relation R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R, and both functional dependencies X → Z and Z → Y hold.

Example:

The SALES relation belwo is not in 3NF.

SALES

Customer#	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7942	Hobbs	Smith	South
6837	Tucker	Hernandez	East
7018	Arnold	Faulb	North

The relation above contains the following dependencies:

Customer# →Name, Salesperson, Region
Salesperson → Region

The implied functional dependency Customer# → Region is a transitive dependency through the attribute Salesperson. Observe that from the given dependencies, both Customer →Salesperson and Salesperson → Region hold and Salesperson is not a subset, let alone the key of SALES.

The corresponding 3NF relations of SALES are the following:

SALES

Customer#	Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7942	Hobbs	Smith
6837	Tucker	Hernandez
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

As a rule, to transform a relation in 3NF, we decompose the relation by separating the attributes that caused the transitive dependency to exist. Thus, with the relation SALES, we removed the attributes Salesperson and Region and grouped them together to compose a new relation. The attribute Salesperson was retained in the original relation and acts as a foreign key.

SAQ 8-4

Consider the HOUSING relation below. In the relation, a student lives in only one building and for each building there is only one fee. What are the functional dependencies existing in this relation? Is the relation in 3NF? If not, can you decompose the relation to come up with relations in 3NF? The fortune-teller has her own answers; how about you? My answers are at the end of this module.

HOUSING

Student#	Building	Fee
100	Narra	1200
150	Yakal	1100
200	Narra	1200
250	Guijo	1000
300	Narra	1200

Boyce-Codd normal form (BCNF)

A relation is in **Boyce-Codd normal form** if every determinant in the relation is a candidate key. BCNF is stricter than 3NF, meaning that every relation in BCNF is also in 3NF. However, it is possible for a relation to be in 3NF but not in BCNF. Consider the next example.

Example:

The given STDMAJADV relation is in 3NF but not in BCNF. It complies with the following business rules:

- Each student may major in several subjects.
- For each major, a given student has only one adviser.
- Each major has several advisers.
- Each adviser advises only one major.
- Each adviser advises several students in one major.

STDMAJADV

StudentID	Major	Adviser
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

Since students can have several majors and hence several advisers, StudentID is not a sufficient key. Either the combination {StudentID, Major} or the combination {StudentID,Adviser} is required. Remember, when two or more attribute collections can be keys, they are called **candidate keys**. When one of the candidate keys is selected to be the key, it is called the **primary key.**

Let us choose the set of attributes {StudentID, Major} as the primary key. STDMAJADV is in 1NF, by definition. It is in 2NF since no partial dependency exists. And it is in 3NF because no transitive dependency exists.

Now consider the following dependencies:
StudentID, Major → Adviser Adviser → Major

Since the determinant Adviser is not a candidate key, the relation is not in BCNF.

The corresponding BCNF relations of STDMAJADV are the following:

STDADV

StudentID	Adviser
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

ADVMAJ

Adviser	Major
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

Other Normal Forms

So far I have discussed only functional dependency, which is by far the most important type of dependency in relational database design theory. In most cases, a relation in 3NF is an adequate basis for a good database design. However, there are cases where relations have constraints that cannot be specified as functional dependencies. In the remaining pages of this module we describe additional types of dependencies that may be used to represent other types of constraints on relations. Some of these dependencies lead to further normal forms which we will discuss correspondingly.

Multivalued dependency is a type of dependency that exists when there are at least three attributes (A, B and C) in a relation. For each value of A, there is a well-defined set of values for B and a well-defined set of values for C, but the set of values for B is independent of C. We write A → B and A → C.

Example:

Consider the EMPLOYEE relation below. A tuple in this relation represents the fact that an employee whose name is EmpName works on a project named ProjName and has a dependent whose name is DepName. An employee may work on several projects and may have several dependents, and the employees' projects and dependents are not directly related to one another. We say that the EMPLOYEE relation has two multivalued dependencies and they are EmpName → ProjName and EmpName → DepName.

EMPLOYEE

EmpName	ProjName	DepName
Juan	X	John
Juan	Y	Anna
Juan	X	Anna
Juan	Y	John

Fourth normal form (4NF)

A relation is in **fourth normal form** if it is in BCNF and contains no multivalued dependencies.

Example:

The relation below named OFFERING is not in 4NF and it conforms with the following business rules:

- Each course may have several instructors.
- Each course uses several textbooks.
- The textbook that is used for a given course is independent of the instructor.

OFFERING

Course	Instructor	Textbook
Management	White	Drucker
	Green	Peters
Finance	Gray	Weston
Gilford		

The relation OFFERING contains two multivalued dependencies namely, Course → Instructor and Course → Textbook. To create an equivalent set of relations in 4NF, we decompose the original relation as follows:

TEACHER

Course	Instructor
Management	White
Management	Green
Finance	Gray
Finance	Gilford

TEXT

Course	Textbook
Management	Drucker
Management	Peters
Finance	Weston

Notice that the two new relations include each of the identified multivated dependcies.

SAQ 8-5

Multivalued dependencies exist in the EMPLOYEE relation given in the example on page 95. Normalize this relation so that it is in 4NF. Write down the normalized relations on the space below.

Now, just one more topic and you are finished this module.

Fifth normal form (5NF)

A relation is in fifth normal form if it is in 4NF and does not have any **join dependency**.

A relation that has **join dependency** cannot be divided into two (or more) relations such that the resulting tables can be recombined to form the original table.

At the moment, join dependencies seem a bit mysterious. We do not know what the consequences of a relation having join dependencies are, nor even if there are any practical consequences. If you like to know more about this topic, I suggest you read Ullman (1982) or Date (1990).

Summary

In this module, I introduced the concept of functional dependency and used it to define normal forms based on primary keys. A relation in first normal form (1NF) must include only attributes with atomic, single values. The second normal form (2NF) requires the relation to be in 1NF and that every non-key attribute is fully functionally dependent on the primary key. The third normal form (3NF) on the other hand does not allow transitive dependency to exist in a 2NF relation. Boyce-Codd normal form (BCNF) is a stricter form of 3NF such that it requires all candidate keys to be determinants in the relation.

Usually, having database relations in 3NF is adequate. However, there are other normal forms that exist based on other kinds of dependency. Relations in fourth normal form (4NF) are based on the concept of multivalued dependency and relations in fifth normal form (5NF) are based on join dependency.

I hope you learned how to identify functional dependencies in a relation and apply the rules of normalization. You can go over the module again and pay special attention to the examples given. The more familiar you are with the different normal forms, the easier it will be to normalize relations.

References

Adamski, J., and P. Pratt. 1991. *Database systems: management and design.* Boston, Massachusetts: Boyd & Fraser Publishing Co.

Date, CJ. 1990. *An introduction to database systems.* Reading, Massachusetts: Addison-Wesley Publishing Co., Inc. 5th ed.

Elmasri, R., and SB. Navathe. 1994. *Fundamentals of database systems*. Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc.

Silberschatz, AH F. Korth, and S. Sudarshan. 1997. *Database system concepts*. Singapore: The McGraw-Hill Companies, Inc. International ed.

Ullman, JD. 1982. *Principles of database systems*. Rockville, Maryland: Computer Science Press, Inc.

Answers to Self-Assessment Questions

ASAQ 8-1

For the ACTIVITIES relation, you must have identified the following functional dependencies:

Student#, Activity → Fee
and
Activity → Fee

The first dependency is a trivial one. The attributes Student# and Activity comprise the primary key of the relation. Thus, this set of attributes is the determinant of all other non-key attributes. In case of the ACTIVITIES relation, the only non-key attribute is Fee.

The second dependency means that the value of Activity determines the value of Fee. Look at the table again and see if this is indeed the case.

ASAQ 8-2

The schema of the relation EMP_PROJ can be represented as follows

EMP_PROJ(Employee#, EmpName, {PROJS(Project#, Hours)})

The set of braces {} identify the attribute PROJS as multivalued and we list the component attributes that form PROJS between parentheses ( ). To normalize the EMP_PROJ relation into 1NF, we remove the nested relation attributes into a new relation and come up with the following relations:

EMP_PROJ1(Employee#, EmpName)
and
EMP_PROJ2(Employee#, Project#, Hours)

ASAQ 8-3

Just a snitch! Now compare your answers with mine. You should have identified the following dependencies:

Employee#, Project# → Hours
Employee# → EmpName
and
Project# →ProjName, ProjLocation

The corresponding 2NF relations (remember, I only asked for the schema) are

EMP_PROJ1(Employee#, Project#, Hours)
EMP_PROJ2(Employee#, EmpName)
and
EMP_PROJ3(Project#, ProjName, ProjLocation)

Remember, if a relation is not in 2NF, it can be further normalized into a number of 2NF relations in which non-key attributes are associated only with the part of the primary key on which they are fully functionally dependent. The three functional dependencies above hence lead to the decomposition of EMP_PROJ into three relations also given above, each of which is in 2NF.

ASAQ 8-4

The primary key is Student#. The existing dependencies are Student# → Building and Building → Fee. Since Student# determines Building and since Building determines Fee, then, indirectly, Student# → Fee. Thus, a transitive dependency exists in the relation.

Transforming the HOUSING relation into relations in 3NF, we come up with the following relations:

STD_HOUSE

Student#	Building
100	Narra
150	Yakal
200	Narra
150	Guijo
300	Narra

BLDG_FEE

Building	Fee
Narra	1200
Yakal	1100
Guijo	1000

ASAQ 8-5

Here are the equivalent relations of EMPLOYEE in 4NF.

EMP_PROJ

EmpName	ProjName
Juan	X
Juan	Y

EMP_DEP

EmpName	DepName
Juan	John
Juan	Anna

Yahoo! That's the last SAQ for this module. Did you get the correct answer? If so, good for you. If not, you can always review your work. The EMPLOYEE relation contains two multivalued dependencies. Decomposing it will result in the two relations above.