# LO2 Model User’s Data Requirements Using Conceptual Modeling Techniques

## ERD Review:

**Entity:** anything that can have an independent existence and that can be uniquely identified. An entity is an abstraction of the complexities found in some real world situation.

Cardinality and Modality are indicators of the business rules around a relationship.

**Cardinality:** how many instances of an entity relate to an instance of another entity.

**Modality or Ordinality:** describes the relationship as either optional or mandatory. (Example: one-to-one). The minimum number of times an instance of one entity can be associated with an instance of a related entity.

Cardinality can be 1 or Many with the symbol placed on the outside ends of the relationship. Modality can be 1 or zero and the symbol is placed on the inside next to the cardinality symbol.

Cardinality

Modality

**Zero or more**



**One or More**



**One and only One**



**Zero or One**



## Relationships

### One to One

One-to-one implies that there is exactly one row in the Student entity for every row in the Chair entity.





### One to Many

One-to-many implies that there can be many courses taught by each instructor.



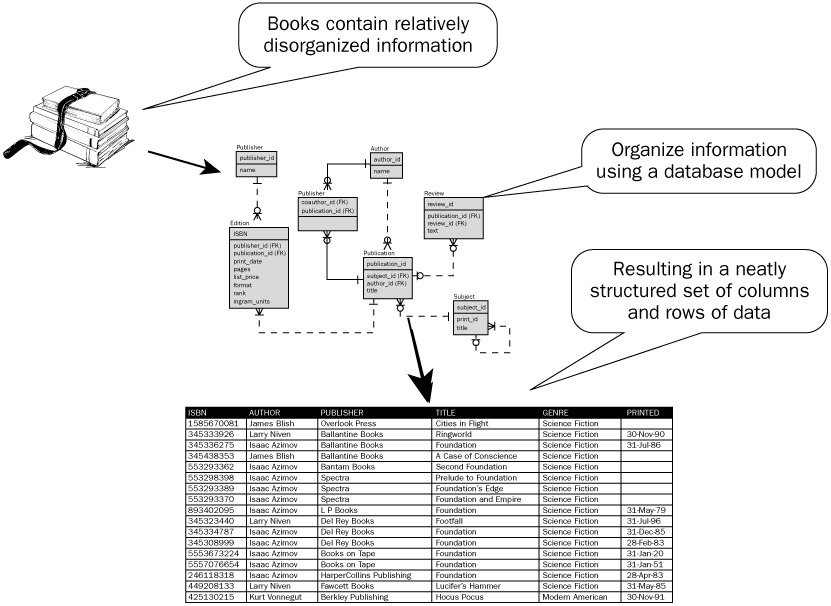


### Many to Many

Many-to-many implies an instructor can teach many courses and a single course can be taught by many instructors, when assuming multiple offerings exist for a single course.



## 2.1 Analyze problems to identify entities and relationships or objects



In its simplest form an entity can be thought of as a noun. For example: a computer, employee, song, mathematical theorem.

A relationship captures how the entities are related to one another. In its simplest form a relationship can be thought of as a verb linking two nouns. For example, a **company** *owns* a **computer**, an **employee** *supervises* a **department**, an **artist** *performs* a **song**, or a **developer** *writes* a **computer program**.

Usually, clients only give us a small part of the problem. It’s going to be up to us to ask the questions to complete our model.

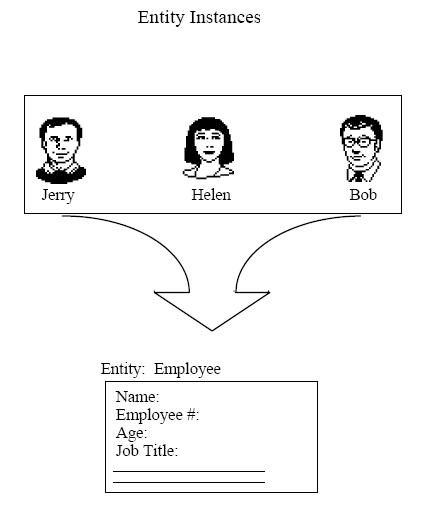
**STOPPED HERE ON 09/06**

## 2.2 Model Problems with Entity-Relationship Diagrams

The ERD is constructed in an iterative manner. The following approach is taken:

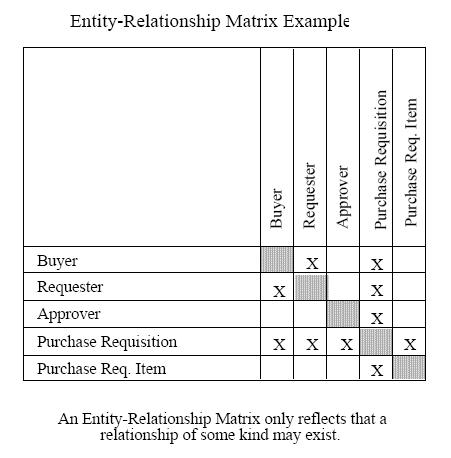
1. **Determine the Entities**

During the analysis phase, customers are asked to list the items that the application or business process addresses. These items evolve into a list of input and output data objects as well as external entities that produce or consume information. Entities are nouns from requirement descriptions.

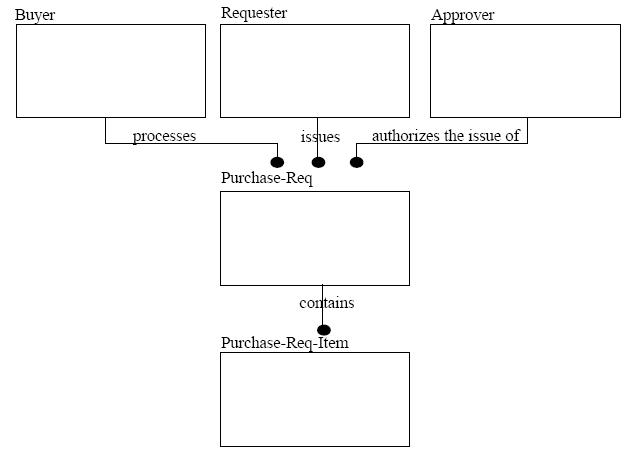


We have some instances of an employee entity – Jerry, Helen, and Bob. Each of these instances will have their own attributes like name, employee#, age, job title, gender, etc.

1. **Define the relationships**. Relationships are often the verbs found in the requirements description.  
   1. Taking the objects one at a time, the analyst and customer define whether or not a connection (unnamed at this stage) exists between the data object and other objects.



* 1. Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.



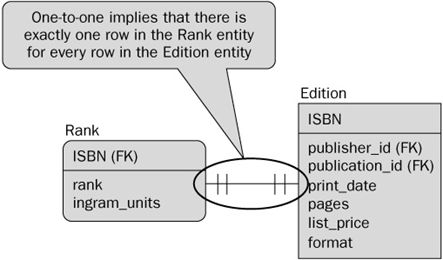
* 1. For each object/relationship pair, cardinality and modality are explored. What are our relationships? Are they 1 to 1? Are they 1 to many?

### Reading/Defining the Relationship

When determining what type of relationship 1-1,1-n, or n-n, we look only at cardinality of both entities.

When reading the relationship, you read the cardinality of one side with the modality of the other.

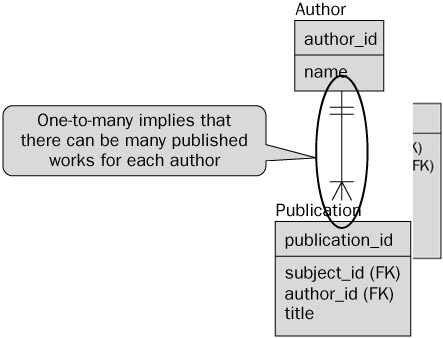
#### One to One



Reading from left to right: For each rank, there is exactly one edition

Reading from right to left: For each edition, there is exactly one rank.

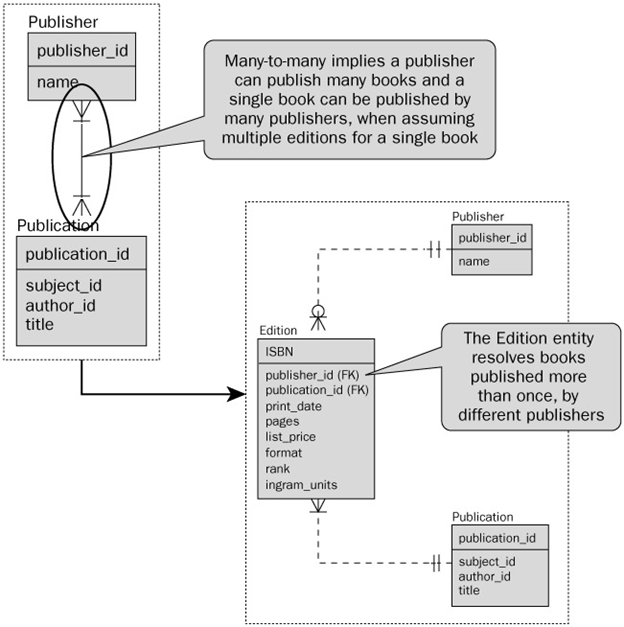
#### One to Many



Reading from top to bottom: One author has many publications

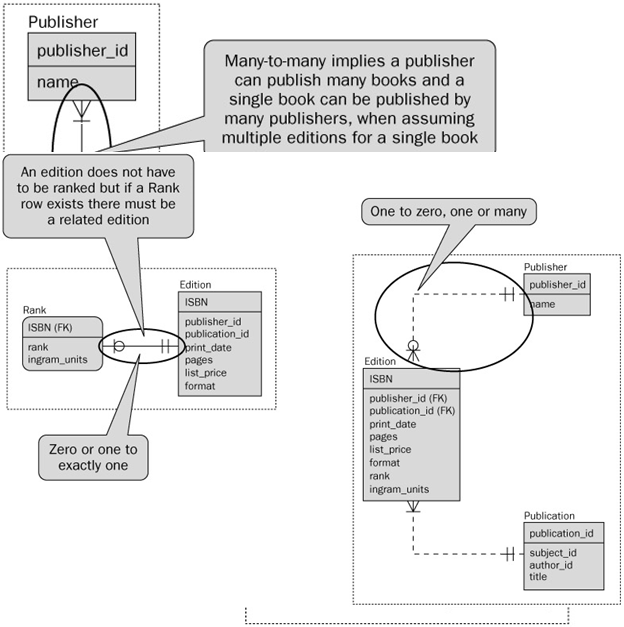
Reading from bottom to top: One publication has one author

#### Many to Many



When we have a many to many relationship, it is best practice to break up this relationship to have a joining table between the two original tables. This will break the n-n relationship into tow one to many relationships. Some organizations will have different naming conventions. I’ve seen one that would take both entities and add on XREF to the end. So, in this case, the entity would be called PublisherPublicationXref (instead of “Edition”).

#### Modality – Zero or One



We can also modality of 0. What this means is that for each record you could have or more matching records in the corresponding entities.

List the following for each entity:

1. Modality
2. Cardinality
3. Type of relationship (not to be confused with how you read the relationship)



L2R (Student): Modality: 1, Cardinality: 1

R2L (Seat): M=1, C=1

Relationship: 1-1 relationship



L2R (Instructor): M=1, C=1

R2L (Course): M:1, C=Many

Relationship: 1-M

L2R (Student): M=1, C=Many

R2L (Course): M=1, C=Many

Relationship: M-M (M-N)



L2R (Professor): M=1, C=1

R2L (Section): M=0, C=Many.

Relationship: 1-M

* 1. Steps a, b and c are continued iteratively until all object/relationships have been defined. It is common to discover omissions as this process continues. New objects and relationships will invariably be added as the number of iterations grows.

Seat

Student

Professor

Instructor

Course

Section

Seat

seatNum  
seatPosition

Instructor

instructorID  
instructorName  
instructorFaculty

Student

studentID  
studentName  
studentAddress

Course

courseName  
courseNum

Professor

professorID  
profName  
profFaculty

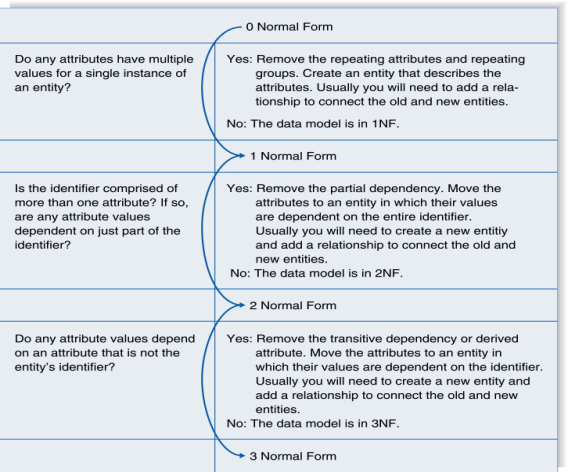
Section

sectionNum

Above is the college example, with some attribute information filled in. Notice that we want to have descriptive attributes attached to the entity they actually belong to.

1. **Formalize and reviewed the diagram to ensure it is Normalized.**

Now that you have completed the process of defining your entities in the ERDs, the next step is to start rearranging those entities and relationships so that when you create the database, it is optimized to both search and store efficiently. The process of optimizing the storage of data is called **normalization**. In order to increase the access speed in retrieving information from the database, denormalization, clustering, and indexing are used. What makes use of denormalized data? Data warehousing.



First Normal Form (1NF): NO repeating elements or groups of elements

Question: Do any attributes have multiple values for a single instance of an entity?

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID | ItemsID1 | ItemsID2 | … |
| 1 | 100 | 101 | … |

The ERD is said to be in first normal form if it does not have any repeating attributes. If the ERD has repeating values then those related values that are repeating should be removed and a new Entity should be created out of those values. For example, in the table above, we see that there can multiple items in an order. Since these values are repeating themselves, they should be removed and placed in a new entity.

In order to solve this problem … we need to remove the repeating columns (ItemsID1, ItemsID2 … ItemsIDN) and create a new table that can still capture this same information.

In effect, we will have one table that tracks order information (ordered, maybe date, customer, payment method/status, etc) and we would have a second table that tracks the items that are part of the order.

|  |  |
| --- | --- |
| OrderID | ItemID |
| 1 | 100 |
| 1 | 101 |

This creates a much simpler set of entities that are easier to work with. This also helps to reduce the amount of empty space in the database since not all our orders will have the same amount of items, yet because of the previous design, would have to allocate enough space to allow for each order to contain a column for multiple items, **even if the space is blank.** This also provides the database with more flexibility because an order can have unlimited number of items.

Second Normal Form (2NF): No partial dependencies on a concatenated key.

This is a complex way of saying that if a column isn’t intrinsically related to the entire primary key, then you should break out the primary key into a different table. The ERD is said to be in second normal form if the ERD is in 1NF and that all the attributes are dependent on the primary key. In order to put the model into 2NF, you will need to move those fields that are dependent on part of the primary key to another table.

Question: Is the identifier comprised of more than one attribute? If so, are any attribute values dependent on just part of that identifier.

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID (PK) | ItemID (PK) | OrderDate | ….. |
| 1 | 100 | 2013-01-01 |  |
| 1 | 101 | 2013-01-01 |  |

The primary key is (OrderID, ItemID)

Consider OrderDate. Is it conceptually part of an order? An order always occurs at some time. But is OrderDate related to a specific item in that order? Not really. You might be saying “But items are part of an order”, and you would be right. OrderDate is independent of the item itself. In the table above, the OrderDate will always be the same for a given order regardless of the value of the ItemID column. This means data duplication, which is denormalization.

Here is the correction:

|  |  |
| --- | --- |
| OrderID (PK) | OrderDate |
| 1 | 2013-01-01 |

|  |  |  |
| --- | --- | --- |
| OrderID(PK) | ItemID(PK) | … |
| 1 | 100 |  |
| 1 | 101 |  |

Third Normal Form (3NF): No dependencies on non-key attributes.

The ERD is said to be in third normal form if the ERD is 1NF and 2NF and none of the attributes are dependent on a non-primary key for any of its entities.

Question: Do any attribute values depend on an attribute that is not the entity’s identifier (PK)?

2NF covers the case of multicolumn primary keys. Simply stated, pull out all columns that don’t directly relate to the subject of the row (primary key) and then put them in their own table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID (PK) | OrderDate | CustomerName | CustomerCity | … |
| 1 | 2013-01-01 | John Smith | Saskatoon |  |

Customer information could be the subject of its own table. Pull out customer name and other customer fields into another table and then put a Customer foreign key into Orders

|  |  |  |
| --- | --- | --- |
| OrderID (PK) | OrderDate | CustomerID(FK) |
| 1 | 2013-01-01 | 123 |

New Customers Entity

|  |  |  |
| --- | --- | --- |
| CustomerID (PK) | CustomerName | CustomerCity |
| 123 | John Smith | Saskatoon |

* During the review process you will need to consider the following:
  + Some of the relations generated from the above steps may be dropped due to duplication.
  + Check all joining tables to make sure that when you created the entities you did not break any business rules
  + Review all the relations to make sure they follow the business rules.

1. Steps 1-4 are repeated until data modeling is complete

## LO 2.3 Compose Relational Schemas from ERDs

We have a picture / image that represents our ER model (ER Diagram). This is how our DB will look eventually but we will need to write some SQL code to create this. The next step is to change our ER Model into a Relational Model.

* + Relational model is based on rigorous mathematical principles, and it has a direct correlation to the database design, so the implementation of an E/R model usually involves converting to a relational model first

### Steps:

#### **Ensure the diagram is normalized:**

#### **For each weak entity** (weak entity to strong entity). A weak entity is an entity that cannot be uniquely identified by its attributes alone.

* Convert to a strong entity by adopting keys of each entity to which it is related
* The primary key will be a combination of the adopted (foreign keys) and any existing key attributes of the weak entity.

#### **For a many-to-many relationship (N:M)**

* Create a new relation with two 1:N relationships.
* The new relation will have a composite key being composed of the key fields from the existing parent relations
* If the relationship has attributes, those attributes should be added to the new relation as well.

#### **Review and Consolidate**

You’ve made (potentially) a bunch of changes, we need to go back and see if these changes affect anything else. Things that can happen is if you’ve added another PK to a table, look for any tables that may have a FK to it. **Ensure that the keys all uniquely identify the rows**.

* Some of the relations generated from the above steps may be dropped due to duplication
* Check all joining tables to make sure that the conversion in step 2 didn't cause any business rules to be broken
* Look at the new relations created to make sure it still follows the business rules

Identify the weak entities and figure out how to make then stronger. Basically, there needs to be FK’s added so that we can easily link back to a record in another table.

#### **For each entity** (Entities to Relations)

* Create a relation with the same name and set of attributes
* If an attribute is composite (made up of more than one possible attribute) you should divide it into multiple pieces. **Example: Address should be split into City, Prov, PC, House#**
* The key of the entity will become the key of the relation
  + EntityName(attribute list)
  + First attribute should be the primary key, and underline

#### **For each relationship (Relationships to Relations)**

* For a one-to-one relationship (1:1)
  + Take the key from either relation and place it as a foreign key in the other. The choice of which one to use as the foreign key is often a matter of careful consideration of the expected number of entries and the expected number of instances of the relationship. **You should put the foreign key into the relation where it will have the fewest null values.**
* For a one-to-many relationship (1:N)
  + Take the key from the one relation (parent) and place it as a foreign key in the many relation (child). Consider: Order and OrderDetails entities and the relation between them. Order entity is obviously the parent entity, with OrderDetails being the child entity.

Note: Reviewing the above, if you see that the same named attribute appears in a few relations, you may rename attributes to avoid confusion. For example: Change name attributes to be instructorName, studentName

For all relationships – list referential integrity constraints for foreign keys and unique attributes.

Add to your relational model the FK in the form <attribute name>(FK)

Write the whole constraint in the form:

Entity (foreign key (FK)) references parentEntity (primarykey)

#### **Final Schema**

* List relation names and attributes in the form
  + RelationName(attribute1, attribute2, attribute3(fk), …)
  + List all referential integrity constraints for foreign keys and unique attributes

### Example 1:

Let’s say that one of your buddy analysts hands you this diagram and tells you it’s not quite finished but that you need to convert it to a relational model so that the DBA can create the database. You can see that your buddy was thinking that they needed to add a Section and method of tracking the grade achieved.

**DASHED LINE:** Usually it means the relationship is non-identifying/weak. In relational terms, the foreign key is not part of the primary key of the referencing table.



1. Normalize
   1. Make weak entities strong
      1. Faculty needs a FK to Department
      2. Student needs a FK to Faculty
      3. Student needs a FK to Major
      4. Major needs a FK to Department
   2. Many-to-Many relationships
      1. Add a grade table and a section table
         1. Grade: gradeAchieved
         2. Section: sectionNum, credit, room, time
   3. Review and consolidate
      1. Section needs pk/fk to faculty and to course
      2. Grade needs a pk/fk to Section and Student
      3. NOTE: you could decide that Grade actually needs it’s own surrogate key because maybe a student takes the same course multiple times and has different grades and you want to store a history of all those grades.



1. **For each entity**:

EntityName(attributelist)

First attribute should be the primary key, and underline it. FK’s have (fk) appear behind it.

* + - * Faculty (facultyID, firstName, lastName, position, deptNumber(fk))
      * Department (deptNumber, name, streetNo, city, prov, postalCode)
      * Course (courseNumber, name)
      * Student (studentID, firstName, lastName, streetNo, city, prov, postalCode, majorID(fk), facultyID(fk))
      * Major (majorID, name, description, deptNumber(fk))
      * Section (facultyID(fk), courseNumber(fk), sectionNum, credit, room time)
      * Grade (studentID(fk), facultyID(fk), courseNumber(fk), gradeAchieved)

1. **For each relationship**:

Write the constraint in the form:

Entity (foreign key (FK)) references parentEntity (primaryKey)

* + - * "is member of" 1:N relationship
        + Faculty(facultyID, firstName, lastName, position, deptNumber(fk))
        + Constraint:
      * **Faculty (deptNumber(fk)) references Department(deptNumber)**
      * "advise" 1:N relationship
        + Student(studentID, firstName, lastName, streetNo, city, prov, pCode, facultyID(fk))
        + Constraint:
      * **Student(facultyID(fk)) references Faculty(facultyID)**
      * "chooses" 1:N relationship
        + Student(studentID, firstName, lastName, streetNo, city, prov, pCode, facultyID(fk), majorID(fk))
        + Constraint:
      * **Student(majorID(fk)) references Major(majorID)**
      * "offers" 1:N relationship
        + Major(majorID, name, description, deptNumber(fk))
        + Constraint:
      * **Major(deptNumber(fk)) references Department(deptNumber)**
      * "teach" N:M relationship
        + Section (facultyID(fk), courseNumber(fk), sectionNum, credit, room, time)
    - Constraints
      * **Section (facultyID(fk) references Faculty(facultyID)**
      * **Section (courseNumber(fk)) references Course(courseNumber)**
      * "enrolls" N:M relationship
        + Grade (studentID(fk), facultyID(fk), courseNumber(fk), gradeAchieved)
        + Constraints
      * **Grade (studentID(fk)) references Student (studentID)**
      * **Grade (facultyID(fk), courseNumber(fk)) references Section (facultyID, courseNumber)**

**STOPPED HERE 09/15**

**4. Final Schema:**

* + - * Faculty (facultyID, firstName, lastName, position, deptNumber(fk))
      * Constraint:
        + Faculty (deptNumber(fk)) references Department(deptNumber)
      * Department (deptNumber, name, streetNo, city, prov, postalCode)
      * Section (facultyID (fk), courseNumber(fk), sectionNum, credit, room, time)
      * Constraint:
        + Section (facultyID(fk)) references Faculty(facultyID)
        + Section (courseNumber(fk)) references Course(courseNumber)
      * Student(studentID, firstName, lastName, streetNo, city, prov, pCode, facultyID(fk), majorID(fk))
      * Constraint:
        + Student(facultyID(fk)) references Faculty(facultyID)
      * Major (marjorID, deptNumber(fk),name, description)
      * Constraint:
        + Major(deptNumber(fk)) references Department(deptNumber
      * Course (courseNumber, name )
  + Grade (studentID (fk), facultyID(fk), courseNumber(fk), sectionNumber(fk), gradeAcheived)
    - Constraints:
      * Grade(studentID(fk)) references Student(studentID)
      * Grade(facultyID(fk)) references Faculty(facultyID)
      * Grade (courseNumber,sectionNum(fk)) references Section(courseNumber,sectionNum)

### Example 2:

The following diagram represents a simplified credit card environment. There are two types of accounts: debit cards and credit cards. Credit card accounts accumulate charges with merchants. Each charge is identified by the date and amount of the charge. Transform the ER diagram into a set of relations. Be sure to identify the primary (and foreign) keys in the resulting relations.



1. Normalize
   1. Make weak entities strong
      1. CardAccount needs a FK to Customer
      2. DebitCard needs a FK
      3. CreditCard needs a FK
   2. Many-to-Many relationships
      1. Add a Charges table
         1. Charges: date, amount
   3. Review and consolidate
      1. Charges need a FK to Merchant and CreditCard

NOTE: what makes the most sense to have as your pk for Charges? Makes sense that the same merchant could use the same card multiple times. Options: composite key merchant id, date,cc OR lets use a surrogate for this – makes things much less complicated.

* + 1. Charges needs a PK transactionID



1. For each entity
   * + - Customer(custID, custFName, custLNname, custStreet, custCity, custProv, custPostalCode)
       - CardAccount(accountNo, expireDate, custID(fk))
       - DebitCard(accountNo(fk),bankNo)
       - CreditCard(accountNo(fk), currentBalance)
       - Merchant(merchantID, merchName, merchStreet, merchCity, merchProvince, merchPostalCode)
       - Charges(transactionID, merchantID(fk), accountNo(fk), date, amount)
2. For each relationship
   * + - “is a” 1:1 Relationship
         * DebitCard(accountNo(fk), bankNo)
         * Constraint:

**DebitCard(accountNo(fk)) references CardAccount(accountNo)**

* + - * "is a" 1:1 Relationship
        + CreditCard(accountNo(fk), currentBalance)
        + Constraint:

**CreditCard(accountNo(fk)) references CardAccount(accountNo)**

* + - * “owns” 1:N Relationships
        + CardAccount(accountNo, custID(fk), expireDate)
        + Constraint:

**CardAccount(custID(fk)) references Customer(custID)**

“has charges” N:M Relationships

* + - * + Charges(accountNo(fk), merchantID(fk), date, amount)
        + Constraint:

Charges(accountNo(fk)) references CreditCard(accountNo)

Charges(merchantID(fk)) references Merchant(merchantID)

3. Final schema:

* DebitCard(accountNo(fk), bankNo)
  + Constraint:
    - DebitCard(accountNo(fk)) references CardAccount(accountNo)
* CreditCard(accountNo(fk), currentBalance)
  + Constraint:
    - CreditCard(accountNo(fk)) references CardAccount(accountNo)
* CardAccount(accountNo, custID(fk), expireDate)
  + Constraint:
    - CardAccount(custID(fk)) references Customer(custID)
* Charges(accountNo(fk), merchantID(fk), date, amount)
  + Constraints:
    - Charges(accountNo(fk)) references CreditCard(accountNo)
    - Charges(merchantID(fk)) references Merchant(merchantID)
* Customer(custID, custFName, custLName, custStreet, custCity, custProv, custPostalCode)
* Merchant(merchantID, merchStreet, merchName, merchCity, merchProv, merchPostalCode)

### Example 3:

Converting Library example to relational schema



### Put a Relational Schema into a table planning chart

Look at Example 3 above. We use table planning charts to map out the tables the database will have prior to creating them in the DBMS. Once one has composed the relational schema, much of this information can be used to build the planning chart. In LO4 you will cover the Check and Default.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table: Faculty** | | | | | | |
| **Column** | **Datatype** | **Null?** | **Check** | **Default** | **Key** | **References** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |