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

## ERD Review:

**Entity:** Anything that can have an independent existence and it has to be uniquely identifiable. An entity is an abstraction of the complexities found in some real world situations.

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 optional or mandatory. The minimum number of times an instance of one of the entities can be associated/interact 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 there is exactly one row in the student entity for every one row in the chair entity

Read from left to right for labels

One and only one students sits on a chair, Dashed lines are to be ignored in this class

### One to Many

Implying that there are many courses that can be taught by each instructor;



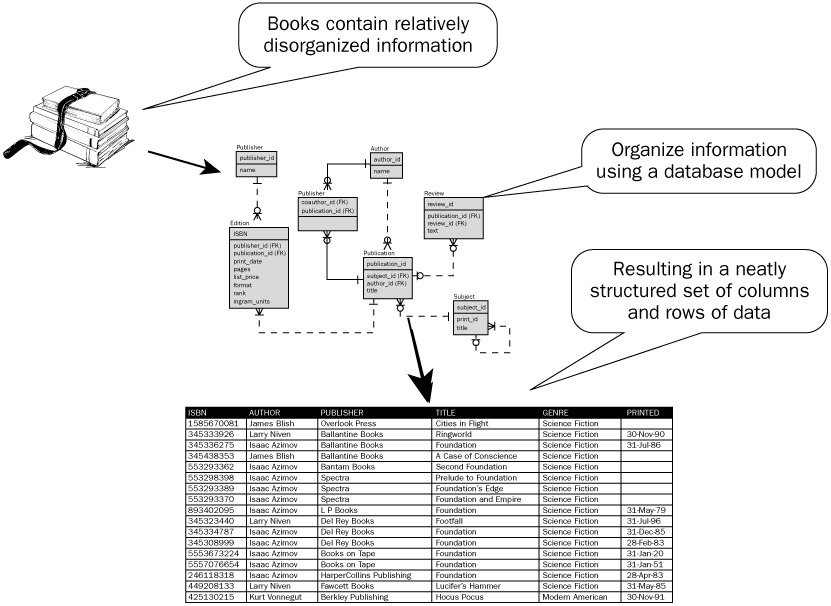


### Many to Many

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

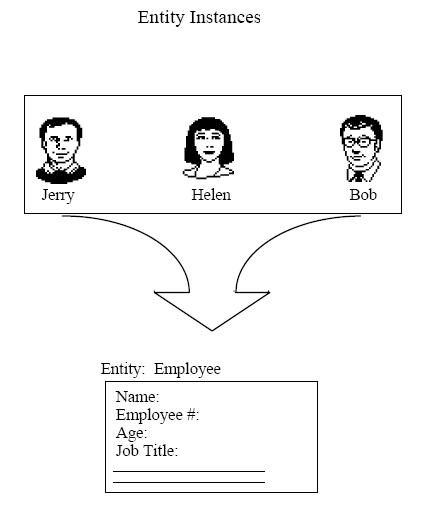
A relationship captures how the entities are related to one another. In its simplest form a relationship can be thought of as a verb linking two nouns. For example, a company owns a computer, an employee supervises a department, an artist performs a song, or a developer writes a computer program.

## 2.2 Model Problems with Entity-Relationship Diagrams

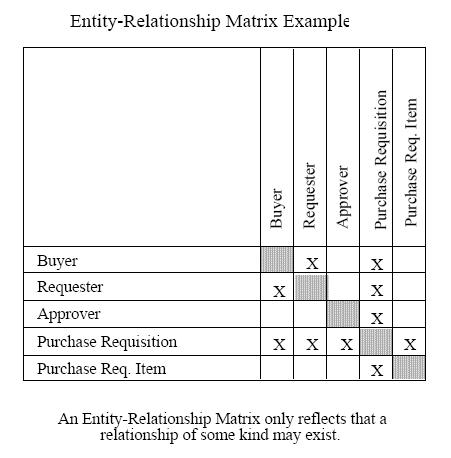
The ERD is constructed in an iterative manner. The following approach is taken:



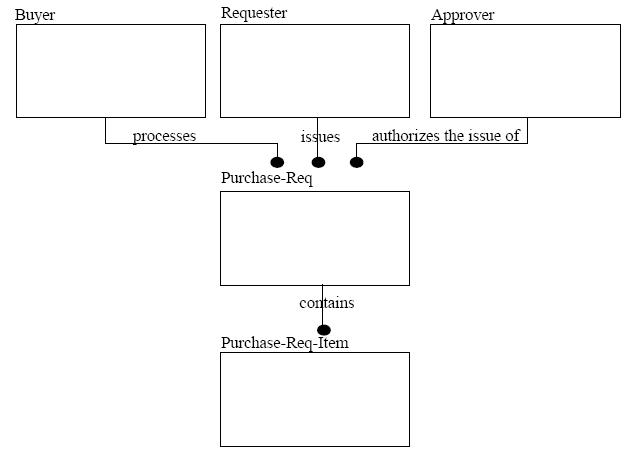
During the analysis phase, customers are asked to list the items that the application or business process addresses. These items evolve into a list of input and output data objects as well as external entities that produce or consumer information. Entities are nouns from requirement descriptions.



1. 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.

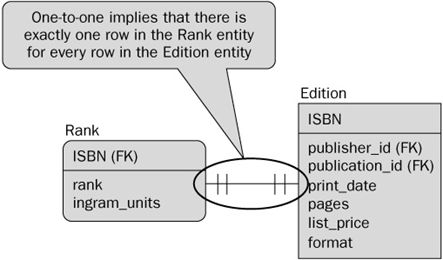


* 1. For each object/relationship pair, cardinality and modality are explored.

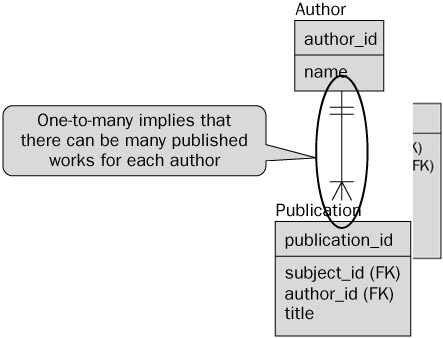
### Reading/Defining the Relationship

When determining what type of relationship (1-1, 1-many, many-many) we look at cardinality of both entities

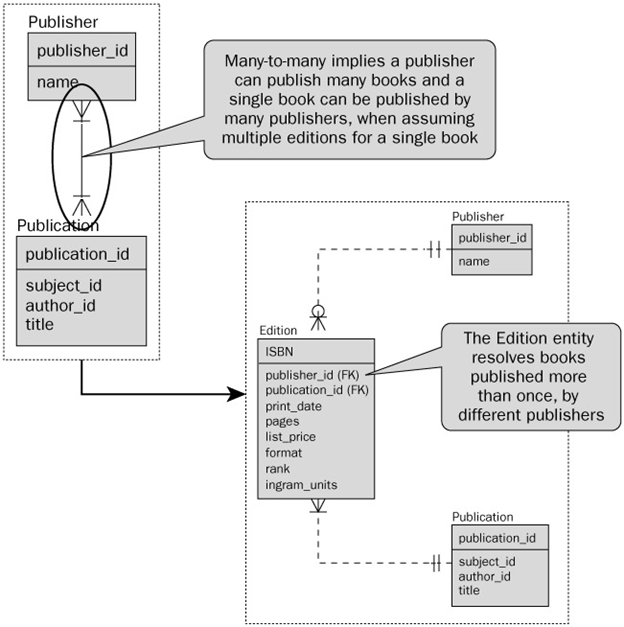
#### One to One



#### One to Many

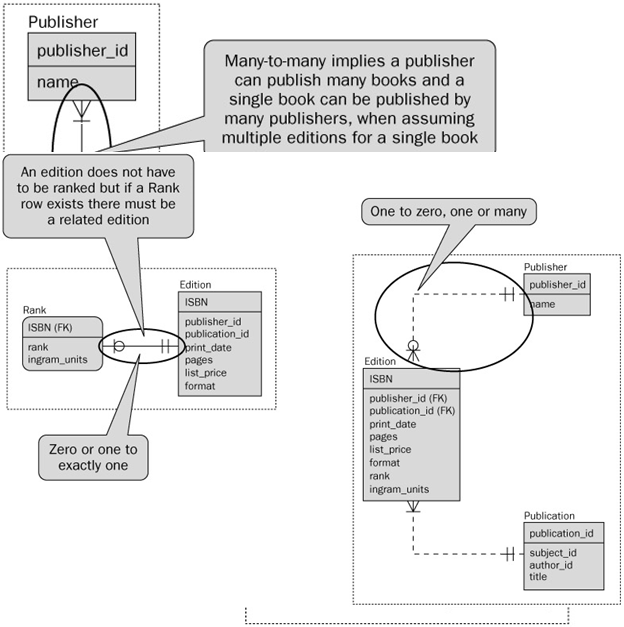


#### Many to Many



When we have a many to many relationship, it is best practice to break them up to have a joining table between the two tables. This will convert the many-many relationship into two 1-many relationships.

#### Modality – Zero or One



List the following for each entity:

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



Student-One and only one, one-one relationship C1 M1

Seat-One and only one, one-one relationship C1 M1



Instructor-teaches one or more courses, 1-many relationship C1 M1

Course-Taught by one and only one teacher, 1-many relationship CMany M1

Student- Takes one or more courses, Many-Many relaionship CMany M1

Course- Taken by one or more student, Many-Many relationship CMany M1



Professor- Teaches zero or more sections, One-Many relationship C1 M1

Section- Taught by one and only one Prof CMany M0

* 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

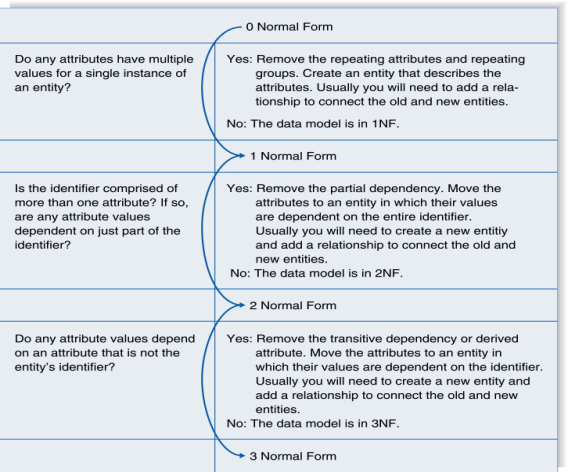
professorID  
profName  
profFaculty

Section

sectionNum



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



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

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

Table that violates first normal form^^^^^^^

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

To solve: Remove the repeating columns and create a table that can still capture the information

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

Second Normal Form (2NF): When you have a composite key:No partial dependencies on a composite key

All columns have to be dependent (or determined by) everypart of the key

Orderdate is only dependent on the OrderID.

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

The primary key is (OrderID, ItemID)

Here is the correction: Pull them apart

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

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

Third Normal Form (3NF): All the customer info isn’t dependent on the OrderID

No dependencies on non-key attributes

Several themes, The “order theme” and the “customer order”, you don’t want several themes in one table

All columns must be dependent on the whole key – if a column is dependent on a non-key there will be a 3NF violation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID (PK) | OrderDate | CustomerName | CustomerCity | … |
| 1 | 2013-01-01 | John Smith | Saskatoon |  |
| 2 | 2013-01-01 | John Smith | Saskatoon |  |

Solution: Create a new table for the stuff that doesn’t relate to the primary key, and add a foreign key in the original table

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

* 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

* + 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)

* Convert to a strong entity by adopting keys of each entity to which it is related
* Mine: Every table has a primary key
* Class: Ensure each entity has an appropriate primary key and foreign keys for each realationship
* 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.
* Class: If the relationship allows the 2 entities to interact more than once(Ex: a customer borrowing a book multiple times) we should use a surrogate(generated) key, May also consider adding an additional attribute to the composite key.
* If the relationship has attributes, those attributes should be added to the new relation as well.
* Mine:

#### **Review and Consolidate**

* 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

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

* Create a relation with the same name and set of attributes
  + RelationName(PrimaryKeyName, attr1, attr2(FK), …)
* If an attribute is composite (made up of more than one possible attribute) you should divide it into multiple pieces. The key of the entity will become the key of the relation
* Mine: If you can break up the composite attribute, DO IT. Ex: Address might become: number, street, city, postal, code

#### **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).

\*\*Foreign keys should be documented for each relation (Underneath the relations description)

RelationName(FKName) references OtherRelationName(attrName)

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

#### **Final Schema**

* List relation names and attributes in the form
  + RelationName(attribute1, attribute2, attribute3(fk), …)

RelationName(attr3)Refences OtherRelationName(attrs)

* + List all referential integrity constraints for foreign keys and unique attributes

### Example 1:



1. Normalize
   * Make weak entities strong
     1. Faculty needs a FK to Department
     2. Student needs a FK to faculty
     3. Student needs fk to Major
     4. Major needs a FK to department
   * Many-to-Many relationships
     1. Add a grade table and a section table
     2. Grade: gradeAchieved
     3. Section: sectionNum, credit, room, time
   * Review and consolidate
     1. Section table needs pk/fk to faculty and course
     2. Grade table needs 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**: Underlined = PK
   * + - 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)
2. **For each relationship**:

Documenting how the relationship entities ties together

* + - * "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)… , …)
    - Constraint:
      * Section(facultyID)) references Faculty(facultyID)
      * Section(CourseNumber(fk) references Course(courseNum))
      * "enrolls" N:M relationship
        + Grade (…, …, …, …)
        + Constraint
      * Gradde(StudentID(FK) references Students(studentID)
      * Grade(facultyID(FK),CourseNumber(FK)) reverences Section(facultyID, CourseNumber)

**4. Final Schema:**

* + - * Faculty (facultyID, firstName, lastName, position, deptNumber(fk))//Each attribute
      * Constraint:
        + Faculty (deptNumber(fk)) references Department(deptNumber)//One for each FK if there is one
      * 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 FK from Customer
      2. DebitCard needs PK
      3. DebitCard needs FK from CardAccount
      4. CreditCard Needs PK
      5. CreditCard Needs FK from CardAccount
   2. Many-to-Many relationships
      1. CreditCard to Merchant needs to have an adjoining table(Charges)
         1. Charges:date amount
   3. Review and consolidate
      * 1. Charges needs pk
        2. Charges needs fk’s to both 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. For each entity
   * + - Customer(custID, custFName, custLNname, custStreet, custCity, custProv, custPostalCode)
       - CardAccount(accountNo, expireDate, custID(fk))
       - DebitCard(accountNo(fk),bankNo)
2. For each relationship
   * + - “is a” 1:1 Relationship
         * DebitCard(accountNo(fk), bankNo)
         * Constraint:
       - "is a" 1:1 Relationship
         * CreditCard(accountNo(fk), currentBalance)
         * Constraint:
       - “owns” 1:N Relationships
         * CardAccount(accountNo, custID(fk), expireDate)
         * Constraint:

“has charges” N:M Relationships

* + - * + Charge(acctountNo(fk), merchantID(fk), date, amount)
        + Constraint:

1. Final schema:
   * Customer(custID, custFName, custLNname, custStreet, custCity, custProv, custPostalCode)
   * Constraint:NA
   * CardAccount(accountNO, expireDate, custID(FK))
   * Constraint:
     + CardAccount(custID(FK)) references Customer(custID)
   * DebitCard(accountNO(FK), bankNo)
   * Constraint:
     + DebitCard(accountNO(FK)) references CardAccount(accountNO)
   * CreditCard(accountNo(FK) , currentBalance)
   * Constraint:
     + CreditCard(accountNO(FK)) references CardAccount(accountNO)
   * Charges(transactionID, accountNO(FK),merchantID(FK),date,amount)
   * Constraint:
     + Charges(accountNO(FK)) references CardAccount(accountNO)
     + Charges(merchantID(FK)) references Merchant(merchantID)
   * Merchant(merchantID, merchantCity, merchantStreet, merchantAddr, merchantProv)
   * Constraint:NA

### Example 3:

Converting Library example to relational schema



Branch(branchName)

Constraints:

CatalogueItem(itemID,BranchName(fk), callNumber(fk), copynumber, circulationStatus)

Constraints:

CatalogueItem(branchName(FK)) REFERENCES Branch(branchname)

CatalogueItem(callNumber(FK)) references Book(callNumber)

Book(callNumber, title, author)

Constriants:

BookLoan(takeOutLoan, itemID(FK), patronID(FK), dueDate, returnDate)

Constraints:

Book Loan(itemID(FK)) References CatalogueItem(itemID)

BookLoan(patronID(FK)) Refrences Patron(patronID)

Patron(patronID, name, address)

Constrinats:

### 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** |
| facultyID | CHAR(17) | No |  |  | Primary |  |
| FirstName | VARCHAR(30) |  |  |  |  |  |
| LastName | VARCHAR(30) |  |  |  |  |  |
| Position | VARCHAR(6) |  |  |  |  |  |
| DeptNumber | NUMBER(3) | no |  |  | Foreign | Department(deptNumber) |

* + Branch(BranchName)
    1. Constraint:
  + Catalogue Item (itemID, branchName(FK), callNumber(FK), copyNumber, circulationStatus)
    1. Constraint:
       - Catalogue Item(branchName(FK)) references Branch(branchName)
       - Catalogue Item(callNumber(FK)) References Book(CallNumber)
  + Book ( callNumber, title, authorFirstName, authorLastName)
    1. Constraint:
  + Book Loan: (takeOutNum, itemID(FK), patronID(FK), dueDate, returnDate)
    1. Constraint:
       - Book Loan (itemID(FK)) references Catalogue Item(itemID)
       - Book Loan (patronID(FK)) references Patron(patronID)
  + Patron(patronID, firstName,Lastname, street, city, addr, prov)