CS221 Lecture: The Relational Data Model

last revised June 25, 2012

Objectives:

1. To understand the fundamentals of the relational model
2. To understand the concept of "key" (superkey, candidate, primary, foreign)

Materials:

1. Projectable version of OO model of faculty-student advising relationship
2. Projectable version of simple database
3. Projectable of second version of Advises table - with an attribute
4. Projectable version of schema diagram for simple database
5. Projectable version of schema diagram for lab database
6. Projectable of CPS122 Video store class diagram

I. Introduction to Relational Database Systems

A. There are quite a few different kinds of database system that are in common use today.

1. There are database systems - largely legacy systems - based on the earliest database model used in DBMS's, known as the hierarchical model.

2. There are database systems known as object-oriented databases, whose structure is object oriented. The entities stored in such databases are objects

3. However, most commercial databases use the RELATIONAL model. There are a whole host of good reasons for using a relational database to provide persistence for an OO program, even though the two models are different:

   a) Wide-scale availability of relational databases.

b) Standardization - the relational model and the language most commonly used to access it are standardized.

c) Legacy data - many organizations have large quantities of data already stored in relational databases, which it would be nice to be able to access from OO programs.

d) Ad-hoc queries: the relational model allows many operations on a database to be done interactively from a terminal session, without needing to write a specialized program. This facilitates extracting information from such a database as needed, without having to anticipate all possible queries and write software for them.

e) Solid mathematical underpinnings: the relational model is grounded in the mathematics of sets and relations, and thus has a sound theoretical basis which we can use to reason about the behavior of relational databases.

4. For this reason, we'll devote the rest of this unit in the course to

   a) Learning about the relational model

   b) Learning about accessing data in a relational database using the most widely-used query language: SQL.

   c) Learning how to access relational databases from Java programs.

   d) Learning some of the rudiments of designing relational databases.

5. This is a big topic. (We offer an entire course on DBMS's, and that just scratches the surface.) We will look at only a small subset of it.

B. Since much of your prior experience has been with the object-oriented data model, we should note, at the outset, that the "OO world" and the "relational world" are two distinct worlds.

1. They have distinct histories

    a) OO: Comes out of the world of discrete simulation; much subsequent work arose motivated by the development of GUI's in the PC world - OO is the natural paradigm for designing a GUI.

    b) Relational Database systems: comes out of the world of business data processing; much of the work has been done in the mainframe world. Database systems historically have had a strong batch processing flavor.

2. They grew up in different parts of the world.

    a) OO originated in Norway, and is very strong in Europe

    b) Early work on relational databases was done by IBM and at Berkeley.

3. In a couple of places, they have very distinct ways of handling certain key issues - which we will refer to as we get to them.

4. Nonetheless, they also have some striking similarities, in some cases having discovered fairly similar solutions to certain problems.

5. In recent years, the relational database model has been evolving toward the OO model, with facilities being added that correspond to facilities present in OO (though still handled in a quite different way). We deal with some of these in CPS352. For this series of lectures, however, we will deal with the "traditional" relational model.

6. For now, then, we will leave the OO world and enter the relational database world, with occasional glances back at OO.

C. OO and the Relational data model differ on how they approach modeling some reality.

1. In an OO model, we represent four basic kinds of things

   a) Classes

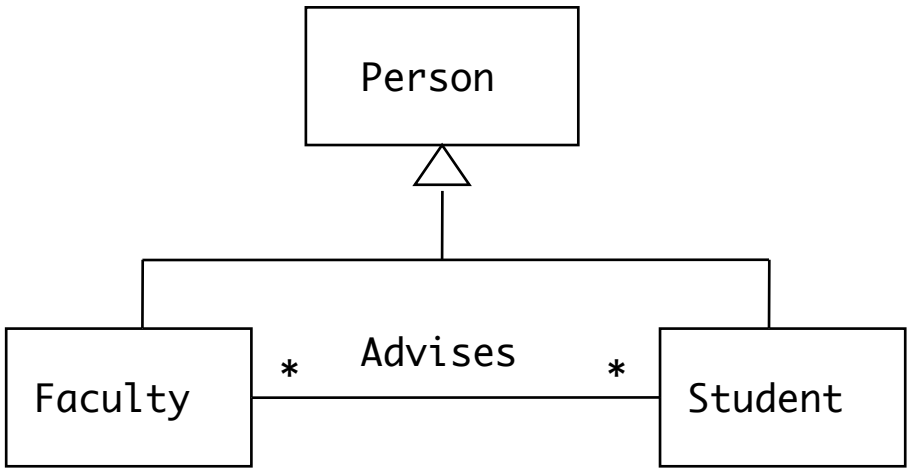
   b) Relationships between classes (e.g. inheritance)

   c) Objects which belong to classes.

   d) Relationships between objects (e.g. association, aggregation, and composition).

   For example, consider how we would represent information about students and faculty advisors. We might construct a simple model like this:

   PROJECT



   In our model, we have

   (1) Three classes

   (2) Inheritance relationships connecting two of these classes to the third.

   (3) Presumably, many objects belonging to the two concrete classes

(4) An association connecting Faculty objects and Student objects

2. In the relational data model there is only one kind of thing: the relation or table.

   Example: The following is a relational model similar to the above:

   PROJECT

   Student

studentID	lastName	firstName
1111111	Aardvark	Anthony
2222222	Cat	Charlene
3333333	Dog	Donna
4444444	Fox	Frederick
5555555	Gopher	Gertrude
6666666	Zebra	Zelda

   Faculty

facultyID	lastName	firstName	livesIn
1	Bjork	Russell	Beverly
2	Brinton	Stephen	Hamilton
3	Crisman	Karl	Lynn
4	Levy	Irvin	Hamilton
5	Senning	Jonathan	Hamilton
6	Stout	Richard	Ipswich
7	Veatch	Michael	Danvers

   Advises

studentID	facultyID
1111111	1
2222222	2
3333333	1
3333333	5

a) There are three tables.

   (1) Two correspond directly to two of the classes in the OO model

   (2) The third corresponds to the association.

b) There is no table corresponding to the Person base class, and no representation of the inheritance relationship. (Extended relational models do have mechanisms for representing inheritance, but we will defer that discussion to CPS352).

c) The individual "objects" are represented by rows in the Faculty and Student tables, The individual relationships between them is represented by rows in the Advises table.

3. In the relational data model, relations or tables actually serve two different purposes

   a) Some represent entity sets - i .e. sets whose elements are (real or abstract) <u>things.</u> Entities have identity and state, but the basic relational model has no mechanism for representing behavior. (Extended relational models do have mechanisms for representing behavior, but again we will defer that discussion to CPS352).

      (Thus, a relational database typically does not deal with controller or boundary classes (for which behavior is the main thing))

   b) Others represent relationship sets - i.e. sets whose elements are relationships between entitites appearing in other tables.

      Each row represents the existence of a relationship (association) between one student entity and one faculty entity - i.e. the first row indicates that Bjork advises Anthony Aardvark.

c) Actually, it is sometimes the case that one will have a table that serves both of these purposes.

4. Thus, though OO and Relational systems are somewhat similar in the way they handle entities, they are quite different in the way they handle relationships.

   a) In an OO system, entities and relationships are represented in totally different ways - the one by objects belonging to some class, the other by either a reference or a collection of references.

   b) In a relational system, entities are relationships are represented in exactly the same way - by relations (tables).

   c) But if the association itself has attributes, then in an OO system, it is objectified as a relationship class - i.e a very different representation is used. In a relational system, there is no change to way it is represented - e.g. if the "Advises" relationship included a "lengthOfTiime" attribute, it could be represented by a table like the following (which simply adds a column for the attribute):

Advises

studentID	facultyID	lengthOfTime
1111111	1	4
2222222	2	1
3333333	1	3
3333333	5	1
...		

PROJECT

II. **The "Key" Concept**

A. The example we just looked at illustrates another crucial notion in a database systems - the concept of a key.

1. This concept is why we chose to use the studentID and facultyID to represent the fact that Bjork advises Anthony Aardvark, rather than the two names.
2. We now explore this concept in detail.

B. In the relational model, a relation (table) is a set, in the mathematical sense - that is, each of its members must be distinct. This implies that the members of a relation must be DISTINGUISHABLE - there must be some difference among them whereby we can tell them apart.

Normally, we expect that the value(s) of a single attribute or a group of attributes will suffice to distinguish one member of a relation from all others.

EXAMPLE: In the relation Student, one of the attributes is studentID. We ensure (when students are admitted) that each student has a studentID distinct from all others - though it might be that two students happen to have the same name.

C. Superkey - We call any set of attribute values which suffices to distinguish one member of an entity set from all others a superkey for that entity set.

1. In general, there can be many superkeys for a given entity set.

EXAMPLE: For the relation Faculty, the facultyID is a superkey. But if we insist that no two faculty can have the same name, then so is lastName + firstName.

EXAMPLE: For the entity set Faculty, livesIn is <u>not</u> a superkey.

EXAMPLE: In the case of Student, studentID is a superkey. However, in general, we would not expect a student's name to be a superkey for the entity set - we can easily have two students with the same name.

2. Indeed, if a given set of attributes is a superkey, then any superset of those attributes is also a superkey.

   EXAMPLE: Since studentID is a superkey for Students, so is studentID + lastName.

3. Sometimes a superkey needs to be composite - i.e. to consist of more than one attribute

   EXAMPLE: In a library, books are identified by call number. But a library might have more than one copy of some book. In this case, each copy is given a unique copy number (copy 1, copy 2 ...) In this case, the attributes callNumber + copyNumber together are a superkey, though neither is by itself - we can't have books that have both the same callNumber and the same copyNumber (though we could have two copies of any one callNumber, or "copy 2" of two different books).

4. Note that - in almost every case - the complete set of attributes for an entity is a superkey for the entity set. All of the examples we work with in this course will have this property.

   In many cases, this is uninteresting, though we will encounter entity sets for which the full set of attributes is the only possible superkey.

5. Finally, note that the notion of superkey is determined by the logic of the database scheme, not by a particular instance.

   EXAMPLE: Suppose that, at a certain point in time, a college had no two students with the same name. That would not make lastName + firstName a superkey for Student, because there is no inherent reason why two students <u>couldn't</u> have the same lastName + firstName.

D. Candidate key - a candidate key is a superkey which has no proper subsets that are also superkeys - i.e. it is, in some sense minimal.

EXAMPLE: For our college example, for Faculty, both facultyID and lastName + firstName are candidate keys (assuming we disallow having two faculty with the same name). However, the combination facultyID + livesIn, though a superkey, is <u>not</u> a candidate key.

E. Primary key - the primary key of an entity set is a particular candidate key chosen by the database designer as the basis for uniquely identifying entities in the entity set.

1. Candidate keys are called that because they are candidates for being chosen as the primary key.

2. If a given entity set has only one candidate key, then the choice of primary key is obvious. But if there are multiple candidate keys, the database designer must choose one to be the primary key.

   EXAMPLE: For Faculty, if we disallowed having two faculty with the same name, we could choose either facultyID or lastName + firstName as the primary key. (In this case, we would almost certainly choose facultyID, because it's simpler and wouldn't "break" if we did hire two faculty with the same name.)

3. Note that it's often the case that, in establishing the attributes for an entity, we include some "ID" attribute that is a natural choice for primary key - e.g. studentID, facultyID.

4. As a matter of good design, we always identify a primary key for each relation (table) we create in a relational database. Often, the primary key attribute(s) is/are underlined when describing a relation scheme.

   EXAMPLE: The attributes of Faculty could be represented as <u>facultyID</u>, lastName, firstName, livesIn.

F. What about relations (tables) that represent relationships? Since relations are always sets, they, too, need a primary key.

1. The primary key of a relation that represents a relationship between entity sets is generally the union of the primary key attributes of all of the entity sets it relates.

   EXAMPLE: If studentID is the primary key for Student, and facultyID is the primary key for Faculty, then studentID + facultyID is the primary key for Advises.

   Note that there are multiple rows for studentID 3333333 and for facultyID 1, but there can be only one row for studentID 3333333 and facultyID 1.

2. The primary key of a relation (table) representing a relationship is thus typically composite - and may be the whole scheme (if the relationship has no attributes of its own).

3. An exception to this statement occurs in the case of a relationship that is 1 to anything (or 0..1 to anything). In this case, the primary key of the other entity is also, by itself, the primary key for the relationship, since no two rows could contain the same value.

   EXAMPLE: Suppose we represented information about a library by entities Borrower (primary key borrowerID) and Book (primary key callNumber + copyNumber), together with a CheckedOut relationship (with columns borrowerID, callNumber, copyNumber and dateDue). By the "primary key of a relationship is the union of the primary keys of the entities" rule, the primary key for CheckedOut should be borrowerID + callNumber + copyNumber. But since the relationship is 1 .. many from Borrowers to Books (a book can only be checked out to one person at a time), callNumber + copyNumber alone is a candidate key and therefore the primary key.

4. Note that, while we sometimes have a choice to make concerning the primary key for an entity set (if there are several candidate

keys), we have no such choice to make for a relationship set - once the primary keys of the participating entities are specified, so is the primary key of the relationship set.

G. When a relation (table) includes the primary key of another relaton (table), the included key is called a FOREIGN KEY.

1. Foreign keys necessarily occur in relations (tables) representing relationships

   EXAMPLE: In the Advises table, borrowerID is a foreign key and facultyID is a foreign key.

2. Foreign keys can occur in other relations (tables) as well.

   For example, suppose we have a Departments table, with each department identified by a department code (e.g. CS). Suppose further that we include a "firstMajor" attribute in our Student table. Of necessity, the value of this attribute must be a valid department code (you can't major in ZZ at Gordon) - so we consider this attribute to be a foreign key.

H. It is with regard to keys that there is another fundamental distinction between the OO model and the relational model.

1. In the OO model, objects have identity, state, and behavior. In particular, we insist that two different objects can have distinct identity, even if their state happens to be the same.

   EXAMPLE: Suppose two different people have bank accounts that both happen to have a balance of $100. Those are still distinct bank accounts, even though their state is the same.

2. In the relational model, an entity set is a set. Therefore, two different entities are not allowed to have identical attributes - they must at least differ in their primary key. Thus, "identity" and "state" are not two distinct concepts, but one.

3. This point of divergence is often moot, because in designing either an OO system or a relational database we often include an attribute in each entity precisely for the purpose of serving as a unique identifier - either a naturally occurring one (e.g. SSN) or one created for that purpose (e.g. a Gordon College student ID).

4. Actually, in an OO system each object does have an "attribute" that that is always unique. What is it?

   ASK

   The location in memory where the object is stored - represented by the reserved word this

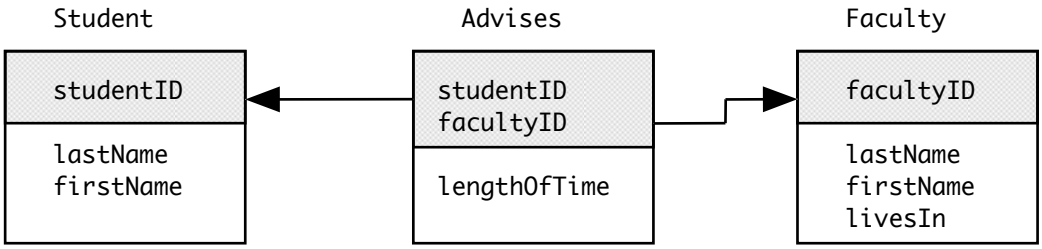
   (But we don't really regard this as an attribute or store it persistently.)

III. **Schema Diagrams**

A. The structure of a relational database is called its schema.

B. A schema can be depicted graphically using a schema diagram. In some sense, a schema diagram plays a role similar to that of a class diagram for an OO system.

C. The following is a schema diagram for the very simple database we have been using for examples:



PROJECT

1. Each box represents a relation (table). Each relation (table) has a name.

2. Each name inside the box represents an attribute (column in the table).

3. The box is divided by a horizontal line. The attributes above the line (in the shaded section of the box) constitute the primary key. (If there are two or more attributes, then the primary key is composite). Those below the line are not part of the primary key.

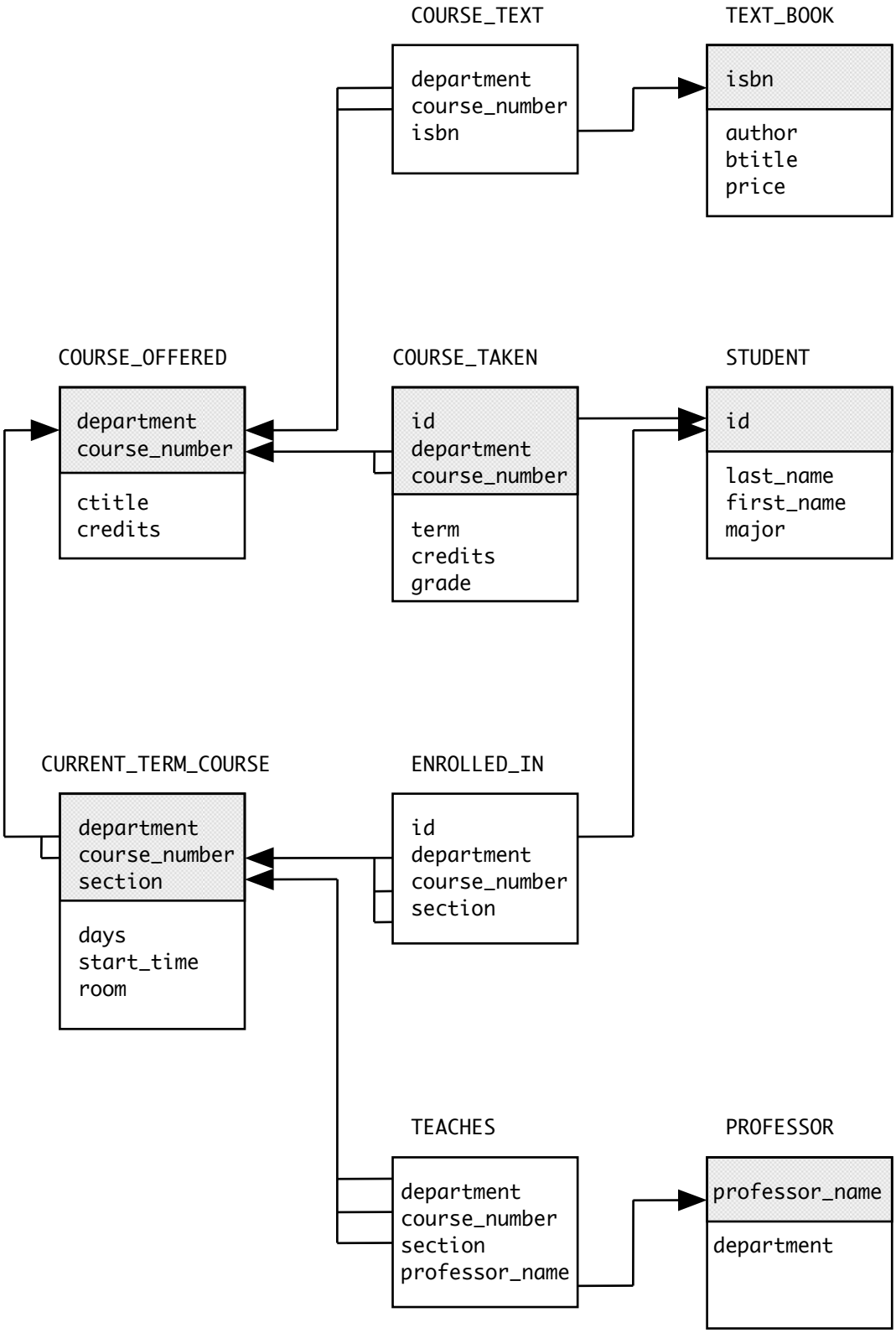
   If a relation is "all key" (i.e. every attribute is part of the primary key), the box is not divided at all - hence a box without a horizontal line represents an "all key" relation.

4. Each foreign key is connected by an arrow to the place where the key is defined, typically as (part of) the primary key of some other relation.

D. The following is a more complicated schema diagram - for the database we will be using in lab.

   Note: This schema assumes that "professor_name" is a superkey - which is, in fact, the way Gordon's current system works. (A professor_name may be something like Dewees-Boyd, I or Dewees-Boyd, M to distinguish Ian DeWeese-Boyd from Margie DeWeese-Boyd.

   PROJECT; GO OVER


COURSE_TEXT
department
course_number
isbn
TEXT_BOOK
isbn
author
btitle
price
COURSE_OFFERED
department
course_number
ctitle
credits
COURSE_TAKEN
id
department
course_number
term
credits
grade
STUDENT
id
last_name
first_name
major
CURRENT_TERM_COURSE
department
course_number
section
days
start_time
room
ENROLLED_IN
id
department
course_number
section
TEACHES
department
course_number
section
professor_name
PROFESSOR
professor_name
department

IV. Some other Differences between OO and Relational Systems

A. We have already seen some key differences between OO and relational systems.

1. The representation of both entities and relationships using the same basic approach: tables.

2. The "key" concept.

3. The related notion that identity is established by values of attributes.

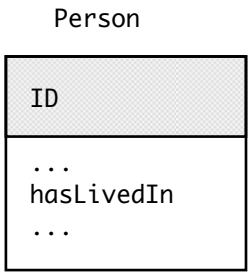
4. We now consider a few more.

B. The relational model requires attributes to be simple, atomic, single values. This is not a requirement in some other database models, nor is it a requirement in OO, and represents one place where some work may need to be done when using a relational database with an OO system.

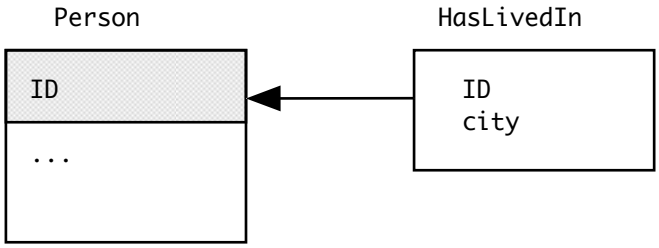
1. EXAMPLE: If an address is stored as street address, city, state, and zip, then a table cannot have an "address" column. Instead, it must have streetAddress, city, state, and zip columns. (Attributes cannot be composite).

2. EXAMPLE: if one of the pieces of information we store about a person is the cities in which they have lived, we cannot have a "cities" column in a table describing the person. Instead, we must use a separate table representing the relationship between the person and the various cities the person has lived in - e.g.

   <u>not:</u>



but rather:



C. Sometimes, we will not know the value of a particular attribute for a particular entity, or it somehow does not apply in a particular case - in which case the value of that attribute is said to be NULL.

EXAMPLE:

Course grades are assigned at the end of a semester. Thus, while a course is in progress, there will be a row in the EnrolledIn table for each Student in the course, but the grade attribute in each row will be NULL.

1. Note that this appears similar to the concept of a null reference in a language such as Java. But it is not the same.

2. One property of NULL is that it is never treated as any value during a query - the principle being that, since NULL means we don't know a value, it should never participate in any query.

   EXAMPLE:

   Suppose we are calculating the GPA for a student. The NULL grades for courses the student is taking now don't count. (Notice that this gives quite a different result than what we would get if we took the grade to be 0!)

(Contrast this with the null pointer exception you might get in Java if you tried to access an attribute whose value is null!)

3. If two different entities have the value NULL for some attribute, the two attributes are not considered equal - i.e. NULL never equals anything - even NULL.

D. Sometimes, a given attribute can be calculated from other information in the database - in which case, instead of storing it we may compute it upon demand. Such an attribute is called a DERIVED attribute.

EXAMPLE:

In a registration system, we may have tables Student, Course, and EnrolledIn. We may wish to include a "total enrolled" attribute for the Course table. This does not need to be an ordinary stored attribute; it can be a derived attribute that is calculated when needed by counting the number of EnrolledIn rows having that course's ID in them. (In a case like this, use of a derived attribute is not only a convenience, but also a good design practice - since it prevents the possibility of having inconsistent stored results if a row is inserted into/deleted from the EnrolledIn table but the enrollment count for the course is not updated.)

V. **Representing Data using the Relational Model**

A. We use the same basic approach to model entity objects as we use with OO:

1. Identify the entities

   - These will become relational database tables

2. Identify the relationships between the entities

   a) 1:1 relationships can be represented using foreign keys

b) 1:many relationships can be represented using foreign keys provided we don't have navigability from the "one" to the "many" or attributes of the relationship

c) many:many relationships will always have to be represented by tables.

B. An example: the video store problem used in CPS122

PROJECT: Class diagram

Develop relational scheme as a class exercise