3d) Special rel'n: **Manifestation** (*"action or fact of showing an abstract idea" – can itself be abstract!*)

Book vs **BookEdition** vs **BookCopy**

You cannot buy the first two kinds of books!

Course
vs
CourseOffering

/* Books have information about authors, etc. */

Book hasAuthors
hasTitle

/* Editions of books are related to the book ("described by it") but have their own properties/relationships too */

BookEdition: ***describedBy*** **Book ... AND**
publishedBy PublishingCompany
hasIsbnNr Number
hasFormat { 'printed 'audio}

/* Book copies are related to book editions, and in turn have their own properties */

BookCopy: ***describedBy*** **BookEdition**
hasCallNr Number
hasCopyNr Number
locatedAtBranch LibraryBranch

Manifestation: ***Bad things to do***

Even though you might want BookEdition to "inherit" properties of Book (e.g., title) you should almost never make

- **BookCopy** IsA **BookEdition** IsA **Book** **(a common mistake!)**
  - **Note that (BookEdition *subclass_of* Book) does not pass the IsA test: not every instance of BookEdition is an instance of Book (do a count test)**
- Nor should you make book copies be *instanceOf* **Book**
  - **when you sell the last copy of some book, you will have no record of the book itself ever having existed (never heard of something titled "Harry Poter and the Chamber of Secrets" by J.K.Rowling). So you won't be able to talk about the book being out of print (or not having any first editions of the book)**
- nor should you create open ended sets of classes
  - **Hamlet** isa **Book**
  - **HarryPotter&ChamberOfSecrets** isa **Book**
  - ...

Manifestation:

- A test: for each instance of the "more general" concept (*BookEdition*) there are many instances (possibly 0) of the more specialized concept (*BookCopy*)?
- Contrast this with *Dog IsA Animal*: an individual animal is either not a dog, or is <u>one</u> individual dog.
- [Messy aside: if you are thinking of individuals animals to be objects like "dog", "golden retriever",... then these are breeds. Dog on the other hand has instances "Fido", "Rover",... What you want to say is that "Fido" is instance of "golden retriever" but the relationship instanceOf is between individuals and classes. So for you, *Animal* has as instances classes – these are called *meta-classes* and lead to even more complexity. Try to avoid. Ignore in this course.]

Manifestation: ***Solutions***

- ***Approach 1****: if you truly don't care about Book, only editions and copies, "merge" the information about Book into BookEdition. (Potential problems: some attributes, such as "dateWhenPrinted" mean different things for Book and BookEdition.)
- ***Approach 2***: relate the descriptor to the described with an explicitly named relationships (**editionOf , copyOf,descrOf**);

/* Book editions are manifestations of books */

BookEdition ---<editionOf >---- Book

/* Book copies are manifestations of book editions */

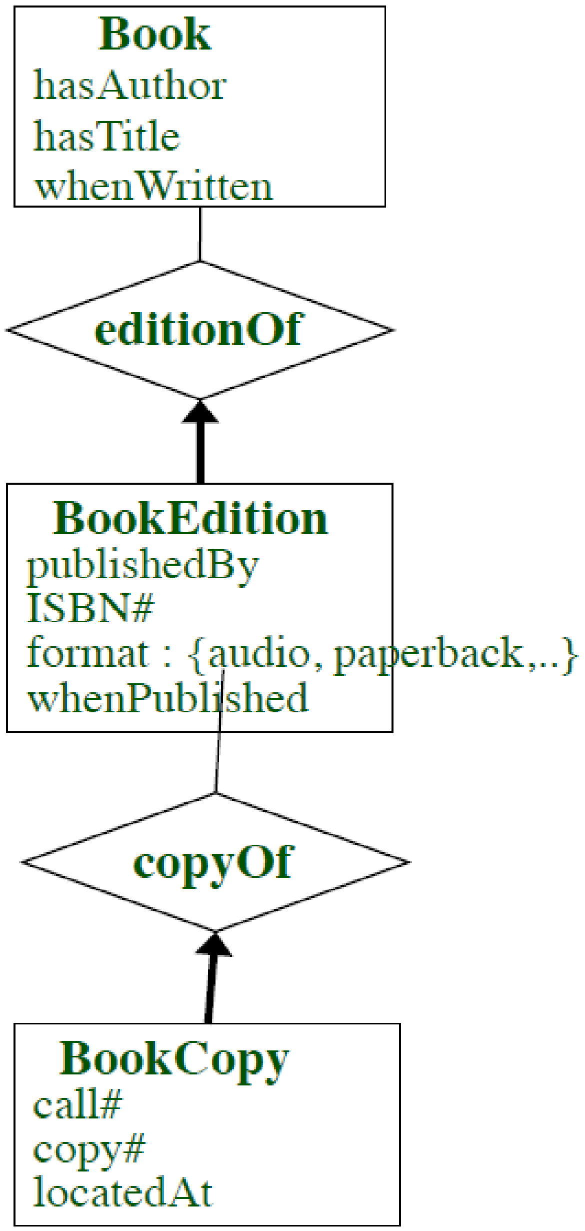
BookCopy ---<copyOf>--- BookEdition

To "inherit" properties, state constraints

(**hasTitle** *of* **BookEdition**) *is the same as* (**hasTitle** *of* **isCopyOf** *of* **BookEdition**)

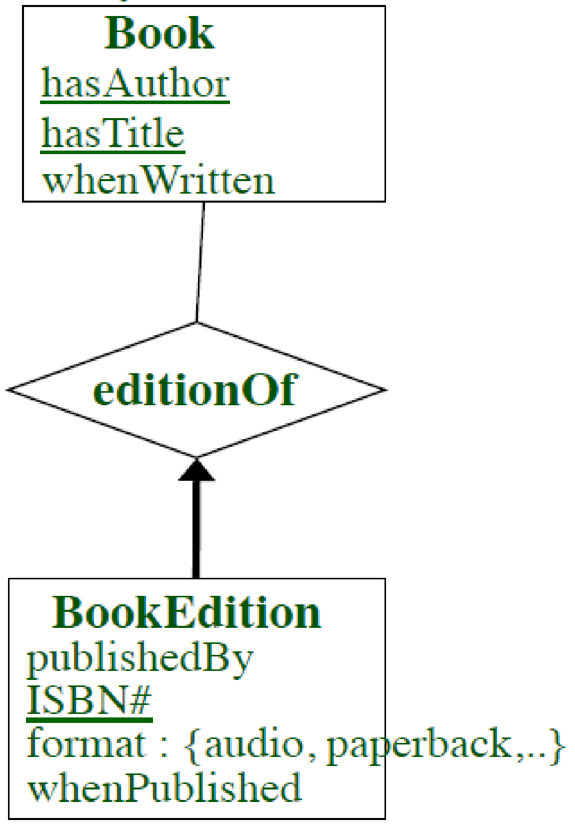
3d) Manifestation in ER notation

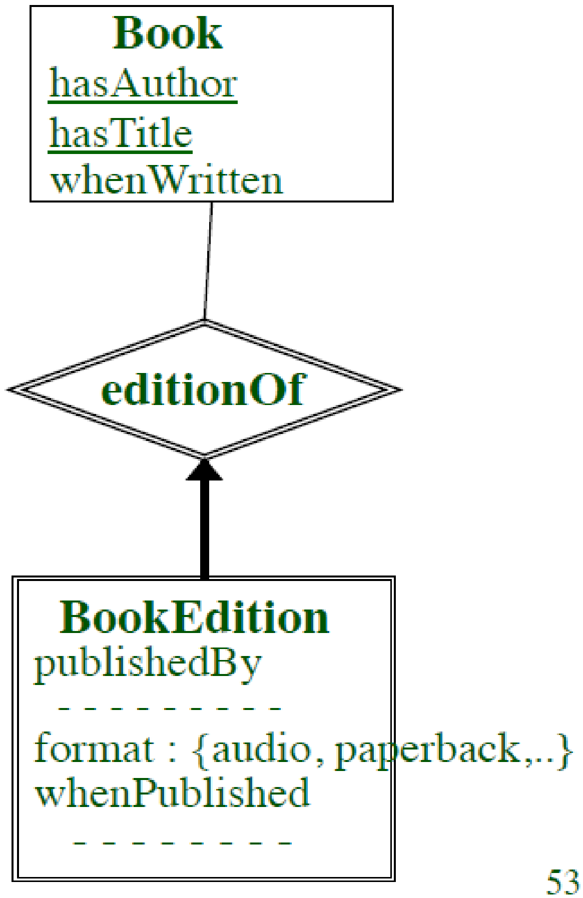
```
/* Books have information about authors, etc. */
Book      hasAuthors
             hasTitle
/* Editions of books are related to the book ("described by it")
   but have their own properties/relationships too */
BookEdition: describedBy Book    ... AND
             publishedBy PublishingCompany
              hasIsbnNr Number
              hasFormat { 'printed 'audio}
/* Book copies are related to book editions, and in turn have their own
    properties */
BookCopy: describedBy BookEdition
             hasCallNr Number
             hasCopyNr Number
              locatedAtBranch  LibraryBranch
```



- Note that if this was not a library, just a book store, it would not make much sense to make a separate BookCopy entity, since they could not be distinguished.
- Better to merge BookEdition and Book, and add a *#ofCopiesInStock* attribute

ER notation requires keys for every entity. In this case (left) we have no problem (key attribs underlined). But if there was no ISBN# then we would have to model it as on the right, as a weak entity:





Summary of Conceptual Design using ER

- *Requirements Analysis*
- *Conceptual design* follows *requirements analysis*,
  - **Yields a high-level description of data to be stored**
- EER model popular for conceptual design for Relational DB
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *ISA hierarchies*, *aggregation*, *weak entities*
- Note: There are many variations on ER model.

Summary of ER (Contd.)

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *participation constraints*, and *overlap/ covering constraints* for ISA hierarchies.
- *Some constraints cannot be expressed in the standard ER model:
  - **enumerations, ranges of values**
  - **"deletion dependencies": if you delete an object (e.g., a car), also delete its <u>parts</u> (e.g, wheels,...); but not other things related to it (e.g., owner)**
  - **multiple ways of partitioning same entity class (e.g., PERSON by Continent of origin (Asian, European,...) vs by age (Young, MiddleAged,Old)**
  - **redundancy and relationship:** ***age, birthdate***
  - **Definitions: BlackCabs (by definition, all cabs with color="black")**
- *"Meta" aspects not modeled properly: *average_salary, retirement_age* attributes

Summary of ER (Contd.)

- Alternative notations/languages that are more up-to-date and expressive include: UML class diagrams, Halpin's ORM, ontology languages (e.g., OWL)
- model design is *subjective*. There are often many ways to model a given UofD! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - **Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, and whether or not to use aggregation.**
- Advice: do not be lead by the relational/table viewpoint; take clues from natural language (see Atzeni book for a good example)

ER methodology (1)

- Figure out what entities/objects are mentioned
  - **beware of synonyms and homonyms**
  - **in general, new entity classes should be "useful" (have something to say about them)**
- Figure out what relationships between the entities are useful, and their arity (binary, ternary)
  - **remember to label roles on relationships involving the same entity more than once**
- Add attributes describing entities
- Abstract out commonalities into super-classes related by IsA relationship
  - **add disjoint and/or covers annotations on the set of subclasses**
- Add attributes describing relationships
  - **remember that there can be only one relationship tuple with the same participants**
- Specify participation constraints on relationships (total: lower bound > 0, functional:upper bound=1)

ER methodology (2)

- Find keys for entities
  - **if only partial key exists, see if there are identifying relationships so you can make it a weak entity**
  - **otherwise, you may need to generate internal identifiers (e.g., studentId)**
  - **remember that every IsA hierarchy needs a key at the top, which is inherited to all subclasses**
- If relationships need to participate in other relationships they need to be made aggregates or reified.
- * Look out for "manifestations" (Course vs CourseOffering):
  - **did you conflate them, and if so is that ok?**
  - **did you by mistake relate them by IsA or InstanceOf?**
  - **you should relate them by some relationship (at the very least "manifestation")**
- * If you used partOf relationship
  - **was it transitive?**
  - **if the part was faulty would you say that the whole was faulty?**

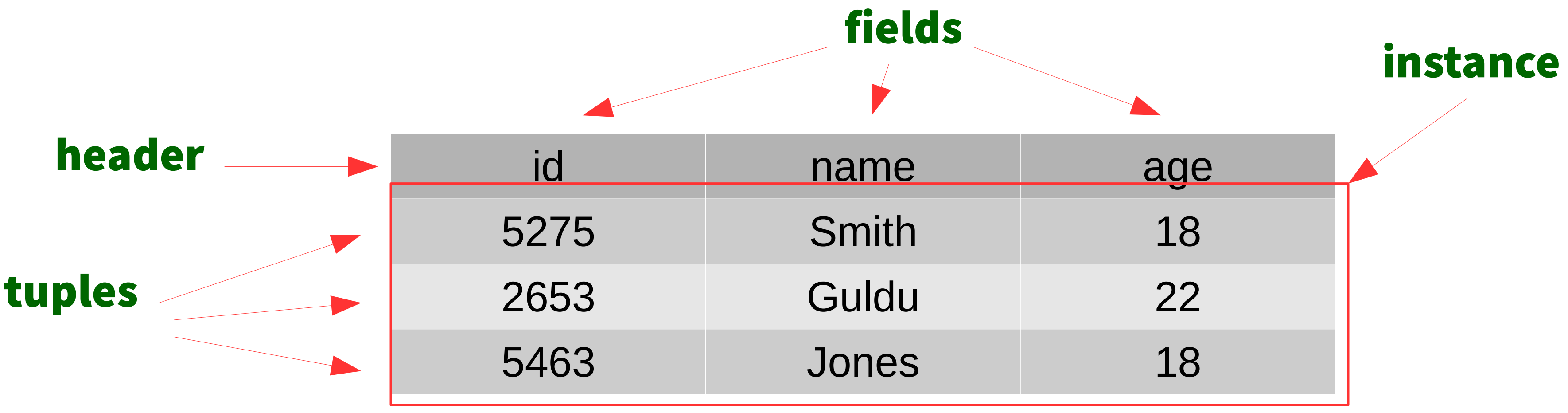
ER methodology (3)

- Lookout for redundancy, especially among reltionships.
  - **Not a good idea to model both <husbandOf> and <wifeOf>**
- State any other constraints that cannot be expressed in the ER notation (e.g., managers earn more than their subordinates) in English.

THE RELATIONAL MODEL

Created by Ted Codd in 1970

- **Everything is described by Relations**
- **Relations are tables that are described by:**
  - **Schema (header): includes the column names and and their domain.**
  - **Instance (contents): are the actual rows of the table, which are called tuples.**



id	name	age
5275	Smith	18
2653	Guldu	22
5463	Jones	18

- **No two rows are identical**
- **Order of tuples or fields is not important**

SCHEMA OF A RELATION

Formally, a relation is defined by:

$R(f_1{:}d_1,\ f_2{:}d_2,\ f_3{:}d_3\ ,\ldots,\ f_n{:}d_n)$

Where:

- **R is the name of the relation**
- **f_i are the names of the fields**
- **d_i is the domain of field f_i. Domain constraints restrict the values that can be used**

A tuple is an ordered list of n elements:

$$(e_1, e_2, e_3, \ldots, e_n) \in d_1 \times d_2 \times d_3 \times \ldots \times d_n$$

An instance of a relation is a set of tuples:

$$\{(e_1, e_2, e_3, \ldots, e_n) \mid e_1 \in d_1, e_2 \in d_2, \ldots e_n \in d_n\}$$

DOMAIN

The domain of a field is more general than its type, since it can restrict the values that are stored in the table, for example we can say that a given field is an integer with values between 0 and 120.

In most of our examples we will use the following basic types:

int – represents integers
double, float – represents real numbers
varchar(n) – represents Strings of length bounded by n
char(n) – represents Strings of fixed length equal to n
datetime – String representing a date and time

PROPERTIES OF RELATIONS

- **The number of fields is called the degree (or ar-ity) of a relation**
- **The number of tuples of an instance of a relation is called its cardinality**
- **The domain constraints must always be satisfied, so only those relational instances that satisfy them will be considered**
- **Since an instance of a relation R is defined as a set, then no duplicate tuples are allowed (in theory). Commercial DBMS will allow duplicates unless constraints are given**

RELATIONAL DATABASE

A Relational Database is a collection of Relations with distinct relation names.

EXAMPLE:

Database: myUniversity

```
Students(sid:int, name:varchar(50), phone:varchar(15))
Professors(ssn:varchar(11), name:varchar(50))
Courses(cid:int, name:varchar(20), credits:int)
Rooms(building:varchar(20), number:int, capacity:int)
Teaches(ssn:varchar(11), cid:int)
Enrollment(sid:int, cid)
```

INTEGRITY CONSTRAINTS

Provide a way to make sure that changes made to a database do not result in loss of data consistency

- **The DBMS must prevent the entry of incorrect data (domain constraint)**
- **A Database is a legal instance if it satisfies all of its Integrity Constraints**
- **The DBMS must enforce all Integrity constraints**
- **The DBMS might automatically make changes to enforce the given Integrity Constraints**

The DBMS designer must specify the time at which the Integrity Constraints will be checked

KEY CONSTRAINTS

- **Candidate Key:** minimal set of fields that uniquely identify a tuple
- **Primary Key:** a candidate key that was selected to be used as a key constraint
- **Superkey:** any set of fields that contains a key

Properties:

- **Two distinct tuples in a legal instance of a relation must not have identical values of all the fields (not necessarily true in commercial systems)**
- **Any proper subset of a candidate key will not uniquely identify a tuple. This is because the candidate key is supposed to be a minimal set of fields**

EXAMPLE (KEY CONSTRAINTS)

Given the following relation:

Student(sid, name, dob, phone)

Candidate keys:

1) sid

2) (name, dob)

In our example, we assume that no two students with the same name can have the same dob (date of birth), but it might be possible for two students to have the same name, or the same date of birth. This shows that (name, dob) is minimal.

PRIMARY KEY CONSTRAINT

- **Is one of the candidate keys.**
- **Is selected by the database designer.**
- **Is what the DBMS will expect and enforce as an Integrity Constraint.**
- **An index is created on it for optimization. It is possible to create indices for other fields.**

The designer of a database must make sure that the primary key constraints do not prevent the storage of a correct tuple!

If name was the primary key in our example, then no students with the same name would be allowed in the database.

PRIMARY KEY NOTATION

There are two ways in which we can represent the primary key in a relation:

1) Underline the fields: R(<u>a</u>:int, <u>b</u>:float, d:varchar(10)), indicates that the fields "a" and "b" are the primary key.

2) Write it down: R(a:int, b:float, d:varchar(10), primary key (a,b)) indicates that the fields "a" and "b" are the primary key.

In most of our examples we will use the second notation. It is also similar to the SQL command.

FOREIGN KEYS

In the Hierarchical and Network models, pointers (reference links) are used to find related data in a database.

In the case of the Relational Model, the data stored in each table is used to make reference to data stored on another table.

If we have a students Relation:

Student(sid, name, login, age, gpa, primary key (stid))

And an enrollment Relation:

Enrollment(cid, grade, stuid, primary key(cid, stid))

Where stid is the id of a student, and cid is the id of a course. Notice that (cid, stuid) is the primary key of Enrollment. Why?

Also notice that the field stuid of Enrollment makes reference to a particular student whose data is stored in table Student.

FOREIGN KEYS (2)

Let us see the Relations as tables:

Enrollment:

cid	grade	stuid
700:320	C	1524
700:210	A	2653
640:356	B	3842
198:205	A	4563

Student:

sid	name	login	age	gpa
5038	Dave	dave@cs	19	3.2
4563	Jones	jones@cs	18	3.3
5275	Smith	smith@cs	18	2.2
3842	Smith	smith@math	19	3.7
1524	Madayan	madayan@music	21	1.8
2653	Guldu	guldu@music	22	3.9

The field *stuid* in the Enrollment Relation makes reference to the key of the Primary Key of the Student Relation.
So *stuid* in Enrollment is called a Foreign key and in the Schema of the relation is denoted as:

Enrollment(cid, grade, stuid, primary key(cid, stid),
foreign key (stuid) references Student)

FOREIGN KEYS (3)

A Foreign key in the referencing relation (Enrollment in our example) must match the Primary Key of the referenced relation (Student) so:

1) Must have the same number of attributes
2) Must be of compatible types (domain)
3) The column names might or might not be the same

Rephrasing:

- Not all the Candidate Keys of Student must appear in Enrollment
- All Foreign Keys in Enrollment must be a PK (Primary Key) of Student

Notice that a foreign key can make reference to the same table:

```
person(ssn, name, parent,
       primary key (ssn),
       foreign key (parent) references person)
```

What if the parent is not known? WE CAN USE NULL!

ENFORCING FOREIGN KEY INTEGRITY CONSTRAINTS

Enrollment:

cid	grade	stuid
700:320	C	1524
700:210	A	2653
640:356	B	3842
198:205	A	4563

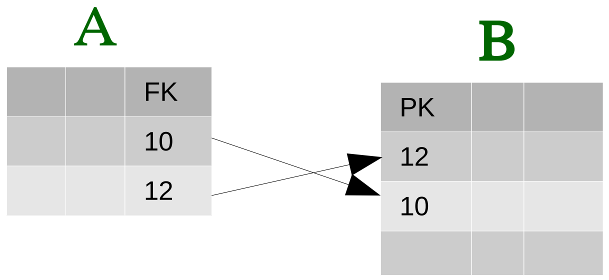
Student:

sid	name	login	age	gpa
5038	Dave	dave@cs	19	3.2
4563	Jones	jones@cs	18	3.3
5275	Smith	smith@cs	18	2.2
3842	Smith	smith@math	19	3.7
1524	Madayan	madayan@music	21	1.8
2653	Guldu	guldu@music	22	3.9

What should happen if we try to:

- **Insert (198:205,'A',5555) into table Enrollment?**
  The DBMS should reject it because 5555 is not in the Student table!
- **Delete (2653,Guldu,'Guldu@music',22,3.9) from the Student table?**
  There are two choices:
  - **The DBMS could Reject the operation because 2653 in Enrollment would not refer to an actual Student… Inconsistent state of the Database**
  - **DBMS could also delete (700:210,'A',2653) from Enrollment to keep the database in a consistent state**

ENFORCING FOREIGN KEY INTEGRITY CONSTRAINTS (2)



INSERTION into Relation A of a tuple whose Foreign Key is not in B:

- Always Reject

DELETION from B of an element referenced from A:

- *Restrict (default):* reject the deletion
- *Cascade:* delete all rows in A that reference the deleted tuple in A
- *Set Default:* set the dangling FKs to an existing default element of B
- Set NULL: this might not be possible if the Foreign Key is part of the Primary Key of A

UPDATE the Primary Key of B of an element referenced from A:

- *Restrict (default):* reject the update
- *Cascade:* modify the corresponding FKs to the new value
- *Set Default:* set the dangling FKs to an existing default element of B
- *Set NULL:* this might not be possible if the Foreign Key is part of the Primary Key of A

FOREIGN KEY CONSTRAINTS IN THE SCHEMA

EXAMPLE:

Enrollment:

cid	grade	stuid
700:320	C	1524
700:210	A	2653
640:356	B	3842
198:205	A	4563

Student:

sid	name	login	age	gpa
5038	Dave	dave@cs	19	3.2
4563	Jones	jones@cs	18	3.3
5275	Smith	smith@cs	18	2.2
3842	Smith	smith@math	19	3.7
1524	Madayan	madayan@music	21	1.8
2653	Guldu	guldu@music	22	3.9

RELATIONAL SCHEMA (CONSISTS OF SEVERAL SCHEMAS OF RELATIONS):

```
Enrollment(cid:varchar(7),grade:varchar(2),stuid:int,
           primary key (cid),
           foreign key (stuid) references Student on delete cascade)

Student(sid:int,name:varchar(50),login:varchar(20),age:int,gpa:float,
        primary key (sid))
```

REVIEW OF CONSTRAINTS

- Integrity Constraints are specified by the database designer when the schema is created.

- Integrity Constraints are enforced when a relation is modified

TYPES OF INTEGRITY CONSTRAINT WE HAVE SEEN:

- Domain constraints: Value, Range, Format
  Examples: not null, unique, date cannot be in the past,
  number must be between 0 and 120
- Primary Key constraints
- Foreign Key constraints (including what to do on update,
  on delete, or on insert)