

The Design of Relations

Leanne Wu

lewu@ucalgary.ca

Department of Computer Science



UNIVERSITY OF
CALGARY

Relational databases use relations

What makes a good database table?

Rows

Only one unique row in each table.

Row are not ordered.

An easy way to ensure each row is unique is to have at least one column (or combination of columns) which only has unique values.

Primary key


Columns

Clearly defined domains and semantics for each column

Relational databases have constraints

Not just anything should go into a database

- What are the appropriate data types for each attribute?
- What are permitted values?
- How much space does each attribute need?

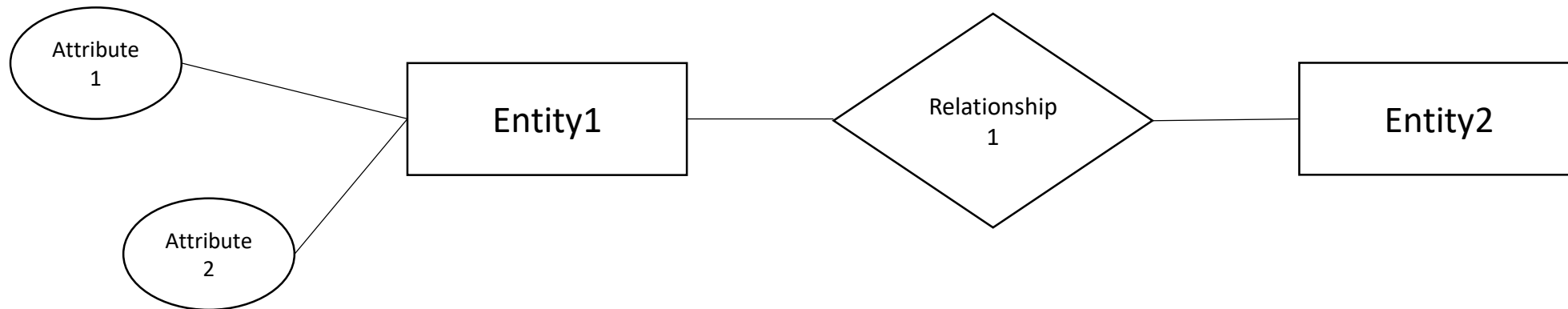


- How should each attribute be related to attributes in other tables? (Can someone have more than one car? More than one address?)
- How are attributes in each table related to each other? What should happen if one of them is updated?

Designing databases

There are many methods to model what goes in a table

- **Entity-Relationship (ER) model:**
 - Allows a conceptual view of the data
 - **Entities:** An object in the real world
 - **Attributes:** A property of that object. Keys are underlined
 - **Relationships:** An attribute of an entity which refers to another entity



An entity?

- Entities are a stereotype of objects in the real world



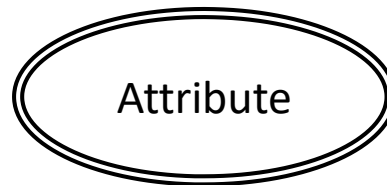
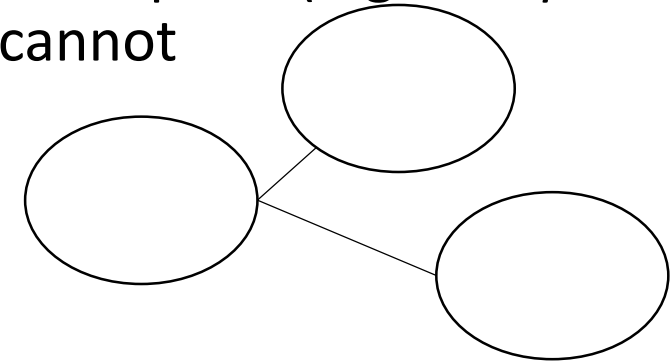
Entity: A person (general)
Exists at the conceptual level while we are
designing the database before we implement it



Record: Leanne (specific)
Not part of the database until we have
designed the database and
implemented it

ER diagrams: Notation for Attributes

- Simple and Composite attributes
 - **Composite** attributes can be decomposed into smaller parts (e.g. a *shipment* could consist of several *items*); **Simple** attributes cannot
 - Look for attributes with their own attributes
- Single-valued vs Multi-valued attributes
 - Some entities may have attributes for which they have more than one value
 - People may have one car, two cars, no cars
 - You can use a double line for the attribute to indicate it is multi-valued



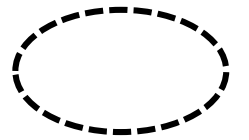
ER diagrams: Notation for Entities and Relationships

- Some entities do not have keys. These are known as **weak entities**.
Use a box with a double line to indicate a weak entity
- Relationships may have **cardinality**.
 - For example, a library has many books, but each book only belongs to one library
- Relationships may indicate **participation**. If the existence of an entity depends on its relationship with another entity, then that first entity can exist only if it participates in one relationship.
 - For example, a car is not registered to anything if it does not have an owner.
 - Use a double line between that relationship and entity to indicate the entity must participate



Other notation

- Stored or Derived Attributes: Sometimes, it is more space-efficient to calculate an attribute based on the value of other attributes
- Complex attributes: attributes can be nested inside each other
 - Consider personal email addresses from more than one provider
- There are other systems of notation used to design databases at the conceptual level (for instance, UML)



How do I create an ER diagram?

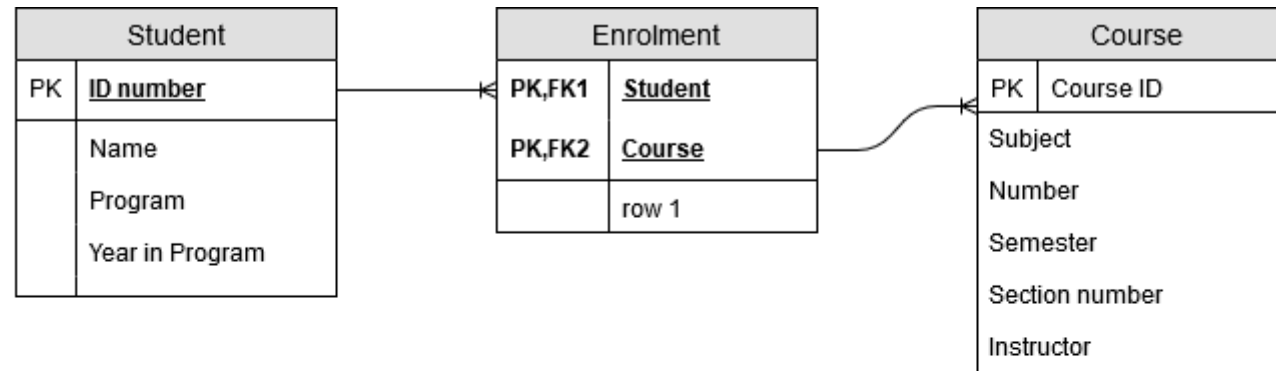
- List your entities
- For each entity, list the relevant attributes
- Identify which of your attributes might:
 - be keys
 - be related to other entities
- Work out what your relationships will be
- Decorate: Identify weak entities, cardinality, participation

This process should be iterative, so expect to do this again and again.

Building a database from an ER diagram

- A **database schema** is a blueprint for your database
 - Contains objects such as tables, and relationships between them
- Formally, a database schema includes a set of tables, and a set of constraints between them
- Presents the database design as a logical view

Example



Translating between an ER diagram and a relational schema

Map the easy stuff

- Strong (non-weak) entities each become a table
 - Include all simple attributes
 - Include simple component attributes of any composite attributes
 - Pick a primary key

Next: Weak entities

- Include all simple attributes
- Include a foreign key which is the primary key of the strong table linked to your table
- Primary key will be a partial key of your weak table (if any) combined with the foreign key

Then: Binary 1:1 relationships

- There are choices
 - Pick one of the relations (X) and pick its primary key as the foreign key of the other (Y)
 - If one of the entities has total participation in the relationship, then it should become X
 - Simple attributes of the relationship should be included as attributes of Y
 - If both relations sharing in the relationship have total participation, then you can merge the entities
 - Create a new relation with foreign keys from both relationships

Translating between an ER diagram and a relational schema (continued)

Keep going! Next, consider:

Binary 1:N Relationships

- For one-to-many relationships, use the primary key of the relation representing the “one” side of the relationship as a foreign key of the relation representing the “many” side of the relationship. Any simple attributes of the relationship should be included in the relation representing the “many” side of the relationship
- Alternately, build a new relation to represent the relationship, as before

Binary M:N Relationships

- Build a new relation to represent the relationship
- Include the primary keys of the relations participating in the relationship as foreign keys in the new relation; its primary key will be the combination of the two foreign keys

Translating between an ER diagram and a relational schema (continued some more)

Finally, these two:

Multi-valued Attributes

- Create a new relation for each multivalued attribute
- For any relationship which had the attribute, include a foreign key from your new relation

Non-binary relationships

- Create a new relation, with foreign keys that are the relations which participate in the relationship
- Include any simple attributes

Is this a good design?

Good database designs should help us to:

- Minimize redundancies
 - Not only is this more efficient, but redundancies can lead to inaccurate data
- Protect referential integrity
 - The design should help us to maintain constraints
- Perform better
 - Selecting keys carefully should reduce the number of operations we need to look up data
 - Minimize NULL values
- Support semantics
 - The design should reinforce the meaning of the data, not distract from it

Dependencies in Plain English

Dependencies are constraints that exist between two sets of attributes in the dataset

- **Functional dependencies** happen when the value of one set of attributes depends on the value of another
 - Consider storing:
 - the price of gas
 - the cost to fill the tank of a car
 - The semantics of your design determine when functional dependencies will appear