From UML to Relations

Classes

- Every class becomes a relation
    - Student(SID, SName, Grade)
    - College(CName, State, Enrollment)

Associations

Many-to-many associations

- Add a relation with key from each side
    - Student(SID, SName, Grade)
    - College(CName, State, Enrollment)
    - Applied(SID->Student, CName->College)

Many-to-one associations

- Add foreign key to the **many** side of the relationship to the relation in the one side
    - Most common
    - Less relations in the schema
    - Increased perfomance due to a smaller number of relations
    - Example:
        - Student(SID, SName, Grade, College->College)
        - College(CName, State, Enrollment)
- Add a relation with key from the many side
    - Increased rigour of the schema
    - Increased extensibility
    - Example:
        - Student(SID, SName, Grade)
        - College(CName, State, Enrollment)
        - Applied(SID->Student, CName->College)

One-to-one associations

- Add a foreign key from one of the relations to the other
    - C1(atr1, atr2, c2_id->C2)
    - C2(atr3, atr4)
- Add the foreign key to the relation that is expected to have less tuples
- Add a unique key constraint to the foreign key

Association Classes

- Add atributes to relation for association
    - Student(SID, SName, Grade)
    - College(CName, State, Enrollment)
    - Applied(SID->Student, CName->College, Date, Decision)

Self associations

- Student(id, ...)
    - Sibling(sid1->Student, sid2->Student)
- Person(id, ...)
    - Relationship(mother->Person, child->Person)

N-ary associations

- Relation with key from each (many) side
    - Team(ID, ...)
    - Player(ID, ...)
    - Season(ID, ...)
        - PlayerSeasonTeam(PlayerID->Player, SeasonID->Season, TeamID->Team)
- If we could only have a team per season of a player:
    - Team(ID, ...)
    - Player(ID, ...)
    - Season(ID, ...)
        - PlayerSeasonTeam(PlayerID->Player, SeasonID->Season, TeamID->Team)

Qualified associations

- Club(ClubID, ...)
- Person(PersonID, ...)
- Member(ClubID->Club, PersonID->Person, MemberNr)
    - {ClubID, MemberNr} Unique key

Generalizations

- 3 conversions strategies
    - E/R style
    - Object-oriented
    - Use nulls

- Best translation may depend on the properties of the generalization

E/R style

- A relation per each class
- Subclass relations contain superclass key + specialized attributes
    - S(k, A)
        - S1(k->S, B)
        - S2(k->S, C)
- (S1 and S2 are subclasses of S)
    - Movie(title, year, lenght)
        - Cartoon(title->Movie.title, year->Movie.year)
        - Crime(title->Movie.title, year->Movie.year, weapon)
        - Biography(title->Movie.title, year->Movie.year, who)

Object-Oriented

- Create a relation for all possible subtrees of the hierarchy
    - Schema has all the possible attributes in that subtree
- In complete generalizations, the relation for the subtree with only the superclass may eliminated
- Object-oriented because each object belongs to one and only one subtree

Overlapping generalizations case

- Movie(title, year, lenght) (Not necessary if generalization is complete)
    - MovieCartoon(title, year, lenght)
    - MovieCrime(title, year, lenght, weapon)
    - MovieCartoonCrime(title, year, lenght, weapon)

Disjoint generalizations case

- Student(SID, SName, Grade)
    - ForeignStudent(SID, SName, Grade, country)
    - DomesticStudent(SID, SName, Grade, state, SSN)
- Or, if is complete:
    - ForeignStudent(SID, SName, Grade, country)
    - DomesticStudent(SID, SName, Grade, state, SSN)

Use nulls

- One relation with all attributes of all the classes
- NULL values on non-existing attributes for a specific object

- Movie(title, year, lenght, weapon, who)

Comparison of approaches answering queries

- It is more expensive to answer queries involving several relations
  - Nulls approach has advantage
- "What movies of 2020 where longer than 150 minutes?"
  - In E/R, only the Movie Relation is needed
  - In object-oriented, all the relations are needed
- "What weapons were used in cartoons of over 150 minutes in lenght"
  - In E/R, the Movie, Cartoon and Crime relations are needed
  - In object-oriented, only the MovieCartoonCrime is needed

Comparison of approaches in space

- E/R approach
  - Several tuples for each object but only the key attributes are repeated
  - Can use more or less space than the nulls method
- Object-Oriented
  - Only one tuple per object with components for only the attributes that makes sense
  - Minimum possible space usage
- Use nulls
  - Only one tuple per object but these tuples are "long", they have components for **all** attributes
  - Used space depends on the attributes not being used

Summary

- E/R style is good for
  - overlapping generalizations with a large number of subclasses
- Object-oriented is good for
  - disjoint generalizations
  - superclass has few attributes and subclasses many attributes
- Use nulls good for
  - heavily overlapping generalizations with a small number of subclasses

Composition

- Treat it as a regular association
  - College(CName, State)
  - Department(DName, Building, CName->College)

Aggregation

- Treat is as a regular association
    - College(CName, State)
    - Apartment(address, #units, CName->College)

Constraints and Derived Elements

- Constraints
    - NOT NULL
    - UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK
        - Ensures that the value in a column meets a specific condition
    - DEFAULT
        - Specifies a default value for a column
- Derived Elements
    - Treat them as regular elements