CS145 Lecture Notes #4

Relational Database Design: The Basics

Steps in Building a Database

1. Understand real-world domain being captured
2. Specify it using a database design model (E/R, ODL)
3. Translate specification to the data model of DBMS (relational, ODL)
4. Create DBMS schema and load data

Relational Data Model

- A database is a collection of relations (or tables)
- Each relation has a set of attributes (or columns)
- Each attribute has a domain (or type)
  - Only atomic types are allowed
- Each relation contains a set of tuples (or rows)
- Each tuple has a value for each attribute
  - Can use a special NULL value

~ Schema: complete description of structure of relations in database
   including relation names, attribute names, domains (optional), etc.
   Example:
   
   Student(SID, name, address),
   Course(dept, number, title)

~ Instance: actual contents (tuples) of relations
   Example:

Keys

A set of attributes K is a key for a relation R if we expect that:
(1) In no instance of R will two different tuples agree on all attributes of K; i.e., K is a “tuple identifier”
(2) No proper subset of K satisfies (1); i.e., K is minimal

Key declarations are also part of the schema
~ declared by underlining key attributes

Example:
Translating E/R Design to Relations

Entity set translates directly to relation
- For a weak entity set, remember the “borrowed” key attributes and watch out for name conflicts
Example: students

Example: rooms in buildings

Relationship set translates to relation containing keys of entity sets
- Don’t forget attributes of the relationship set (if any)
- Multiplicity of the relationship set determines key of the relation
- Watch out for attribute name conflicts
- Special case: double-diamond relationship sets (associated with weak entity sets) need not be translated
Example: students take courses

Example: relationship among students, courses, and TA's

Example: married students; roommates

Example: rooms in buildings
Translating ODL Design to Relations

Assume each class has a key
(if not, imagine there is an “OID” attribute which is a key)

One ODL class translates to one relation

- **Struct**: make one attribute for each field
- **Set**: make one tuple for each member of the set
  - If more than one **Set**, make tuples for all combinations

What about other collection types?

- **ODL relationship**: add the key of the related class
  - Each relationship needs to be translated in only one class
  - Pick the easier one with atomic type if possible

Example:

```java
interface Student (key SID) {
    attribute integer SID;
    attribute string name;
    attribute Struct Addr
        {string street, string city, int zip} address;
    attribute Set<string> clubs;
    relationship Set<Course> courses
        inverse Course::students;
}
```

- Key for the class ≠ key for the relation
- Redundancy
- What if some **Set** is empty?

 Decompose into smaller relations

Example:
Translating Subclasses to Relations

Example: students and graduate students

1. E/R style: an entity is in a network of entity sets related by ISA; create a relation for each entity set
   - An entity is represented in the relation for each subclass to which it belongs
   - Relation has only the attributes attached to that entity set, plus the inherited key

2. ODL style: each object is in one class; create a relation for each class, with all the attributes/relationships for that class
   - Don’t forget inherited properties

3. NULL style: create one relation for the root entity set/class, with all attributes found anywhere in its network of subclasses
   - Put NULL in attributes not relevant to an entity/object