Summary D&I Test 2

Wouter van den Brink

DB Lecture 2

Class diagrams

Modeling data structures can be done using UML class diagrams. Such diagrams consist of classes and their relationships. Classes represent a kind of thing, like a car, person, student, loan or employee. A class does *not* represent an instance of such a thing. A Student is a class; a student named Wouter is not a class. The name of a class is always a singular noun.

Classes may have association. For example, a Person might have a few Hobbies. An association consists of a name, roles for each associated class and multiplicities. The name speaks for itself. The roles describe the association from the perspective of both classes. Finally, the multiplicities describe how much instances of the associated classes are allowed to exist in one association. In the example of a Person having zero or more Hobbies, the role for a Person would be *has_hobby*, and the role for a Hobby would be *has_enthusiast.* One Person would have zero or more hobbies, and a Hobby would have one or more enthusiasts. Usually only one role is described; the one that is most easily understood.

Any kind of multiplicity is allowed; exactly one, zero or more, one or more, zero or one, or something special (only even, only odd, …).

When designing a data structure, one has the choice between a snapshot, or a collection of historic information. A snapshot is a view of the data as it currently exists, like current addresses of people in your address book. Historic information would be useful in a system recording incidents, used by the police.

Another choice has to be made for each attribute of a class. Should it be a usual, “primitive” attribute, or another class?

A ternary (or higher) association consists of more than two classes. An example would be room allocation on the UT, which consists of a course, a time slot and a room. A ternary association is meaningless if one or more objects are missing.

With any association, an association class might be needed. The association class then describes attributes unique to the association, an example being if the room allocation is provisional in the previous example. Only in the perspective of the combination of the three associated classes does this attribute make sense, and thus it is placed in a separate class. An association class does not have a name; rather, the name is derived from the association name.

Sometimes, a class shares attributes with another class. For example, both a Student and a Teacher are a Person. Then a Person is a generalization of both Student and Teacher. A generalization might be disjoint or covering, or both. In a disjoint generalization, an object can belong to at most one subclass. On the other hand, in a covering generalization, every object of the superclass belongs to (at least) one subclass. Generalizations are used whenever two “types” of the superclass have different, specific attributes, or different associations. A subclass without any special attributes or associations can be deleted, as the superclass suffices to describe the empty subclass.

Composition describes some kind of a tree structure in database design. An object that is part in a composition is associated with a single composed object. If the composed object ceases to exist, its parts also cease to exist.

From class diagram to database schema

For clarity's sake, and because this subject cannot really be explained with prose, a bullet point list follows.

- Every class gets a table in the database.
- If an association has certain attributes, or if it is a many-to-many or one-to-one association, the association also gets a table. Otherwise, a reference to the associated class in the table of one of the classes suffices.
- Every database table needs a primary key. Either you add a separate key (like person_id), or you take a combination of attributes, and make that the primary key. A primary key needs to be unique.
- Variable data as the primary key is a bad choice. If people in a database have their names as a primary key, and they marry and change their last name, all references to them get lost.

DB Lecture 3

From class diagram to database schema (continued)

Database structures are easily described by the SQL query resulting in their creation. However, these queries are often very extensive, which is unnecessary for humans to understand. When describing database tables to humans, a shorthand notation is used:

- The CREATE TABLE portion is left out;
- No attribute domains are specified (depends on the context);
- The primary key is underlined, or indicated with PK(column*[, other column(s)]*);
- References are indicated using FK (*column*) REF(*other table[(referenced key)]*);

A primary key is decided by the database designer. A candidate key could be a primary key, and finally, a superkey contains a candidate key. What follows is that a candidate key is a minimal superkey. The primary key might be indicated in a class diagram by appending "PK" to the attribute's description.

A foreign key references a key attribute of another table. The value of the foreign key attribute must exist in the referred table, if it is not null.

Many-to-many associations require a separate table, with at least references to the associated tables, and possibly other attributes belonging to the association.

A one-to-many association can simply be implemented using a foreign key referencing the *one*-part of the association.

If an association requires at least one other instance (a multiplicity of at least 1), then a check may be used to ensure this. The notation is CHECK(*query*). If only one instance is allowed, using UNIQUE(*foreign attribute*) suffices.

A composition is described exactly like a 1..* association is described. However, a database might be instructed to remove the associated row(s) whenever the “parent” row is removed.

Rel. DB Theory: Keys and Functional Dependencies

Let R(A, B, …) be a relational schema. A → B holds in R if for any instance of R, tuples with the same value for A also have the same value for B. The value of B is determined by the value of A, or B Is functionally dependent on A. This does not mean that B is redundant.

Functional dependencies can also be described for sets of attributes; X → Y means the same set of values for the set of attributes X means that the attributes Y have the same values.

Clearly, A, B, … is used to describe attributes and X, Y, … for sets of attributes.

If X is a key of R, then all other attributes in R are functionally dependent on X. Shorthand version: if X is a key of R(XY), then X → XY.

X is a superkey of R if all attributes of R are functionally dependent on X. A candidate key is a minimal superkey. Given a candidate key X of R, there exists no key Y $\subset$ X that is also a superkey.

DB Lecture 4