CSCI317 Database Performance Tuning

Denormalizations

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Denormalizations

Outline

Simplifications

Simplification of a conceptual schema is performed in the following four steps:

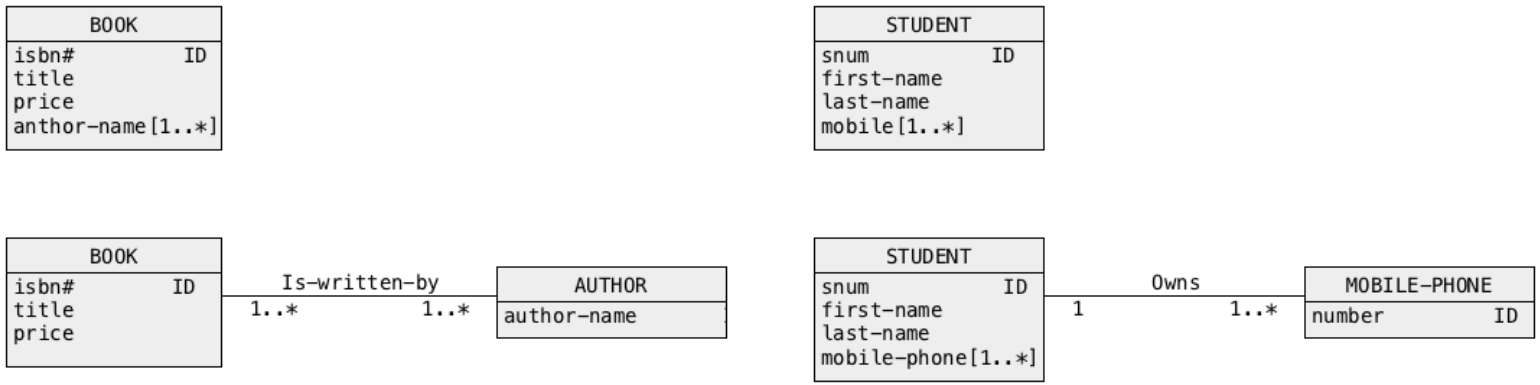
- Elimination of multivalued attributes
- Elimination of association classes
- Elimination of link attributes
- Elimination of many-to-many associations

An outcome of simplification is a conceptual schema that contains only one-to-many and one-to-one associations, qualifications, and generalizations

A simplified conceptual schema is an intermediate form of design between a conceptual schema and the relational schemas

Elimination of multivalued attributes

Elimination of multivalued attributes is performed in the same way as in a process of logical design



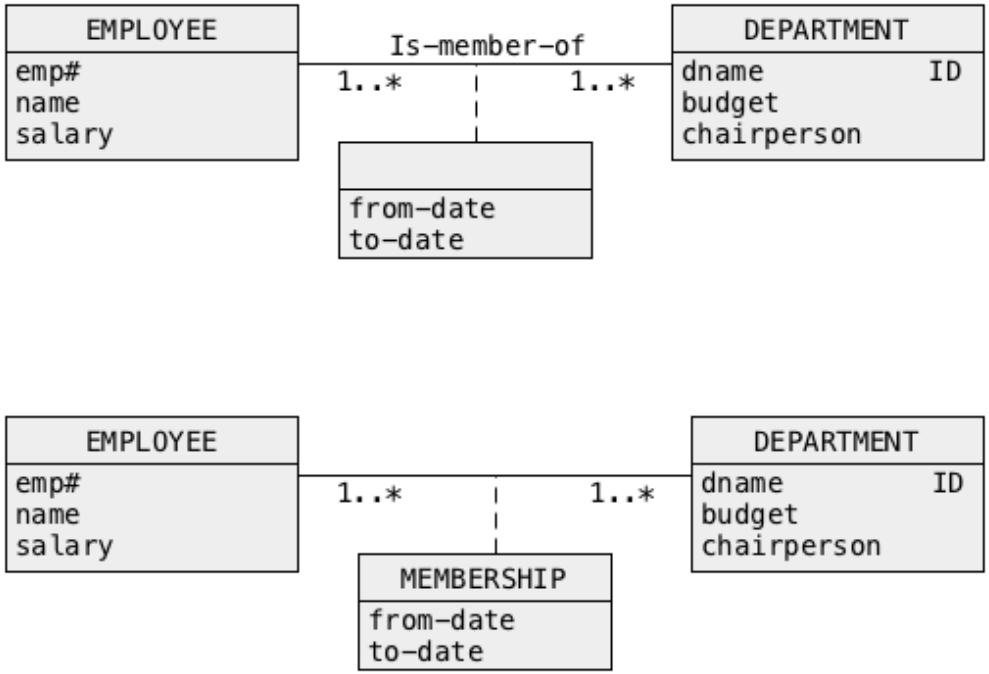
The values of multivalued attribute are promoted to objects

A multivalued attribute is replaced with a class of objects and either one-to-many or many-to-many association depending on the semantics of a multivalued attribute

A new class is described by a single valued attribute with the same name as multivalued attribute

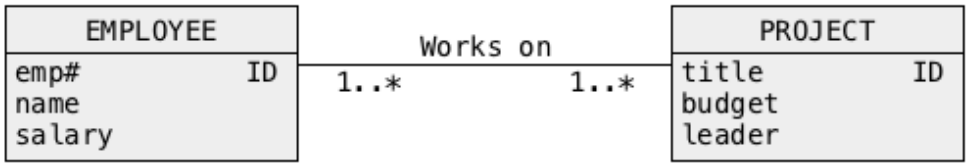
Elimination of link attributes

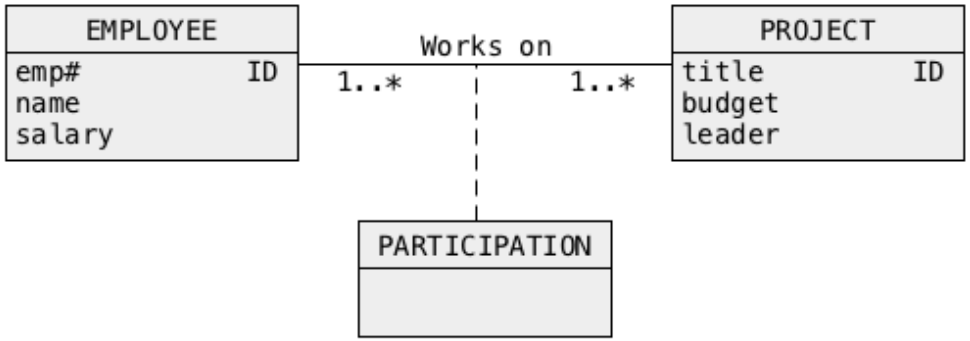
Elimination of link attributes is performed by promotion of link attributes to association classes (to be eliminated in the next step)



Elimination of many-to-many associations

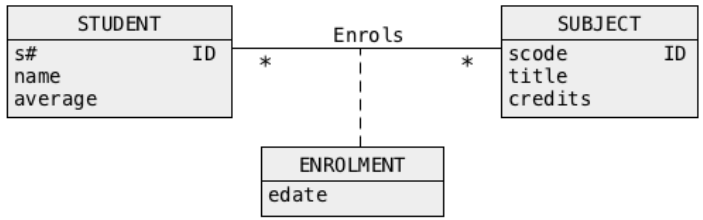
Many-to-many association is transformed into the same many-to-many association and empty association class that can be eliminated in the next step

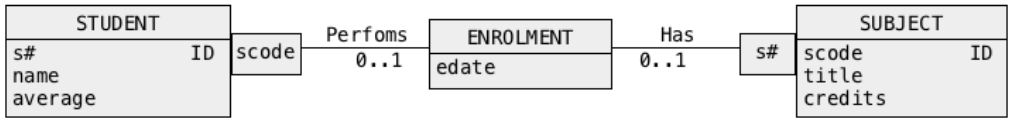




Elimination of association classes

Elimination of association classes is performed by the replacement of many-to-many association and association class attached with one-to-many and many-to-one associations





An association class becomes a class of objects involved in one-to-many and many-to-one associations and qualifications

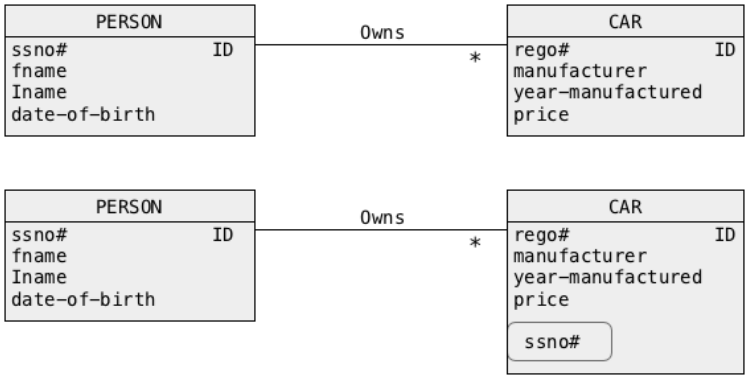
In a general case many-to-many association is equivalent to one-to-many and many to many-to-one associations with a class of objects representing many-to-many association

Denormalizations

Outline

Migration of identifiers

To implement an association we have to copy an identifier from "one" side to "many" side of the association (and NOT the opposite !)



Performance related observations:

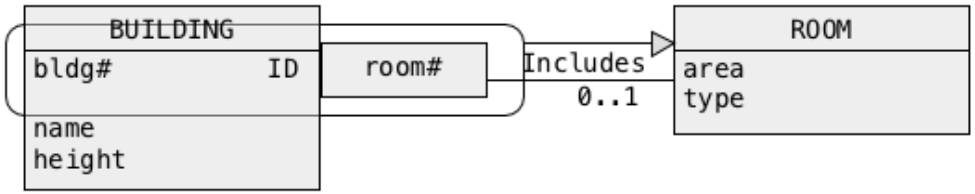
A long composite identifier allows for longer relational schemas without loosing normalization

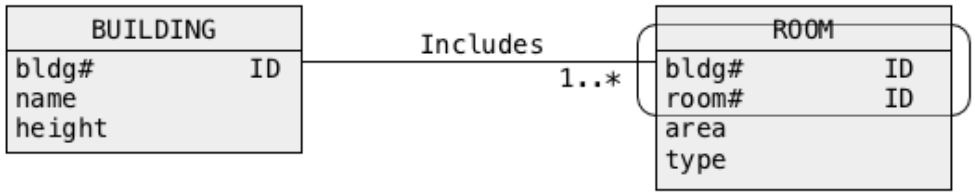
With a long composite foreign key probability that query can be computed without join is higher

Artificial identifiers (surrogate keys) contribute to join operations

Migration of identifiers

If an identifier of a class on "many" side of an association is determined through a qualification then both identifier from "one" side and qualification attributes are copied from "one" side to "many" side of the association (and NOT the opposite !)





Migration of identifiers

Advantages

- Migration of identifiers allows for "link-by-value" implementation of associations

Disadvantages

- Migration of identifiers creates redundancies inevitable for "link-by-value" implementation of associations

Comments

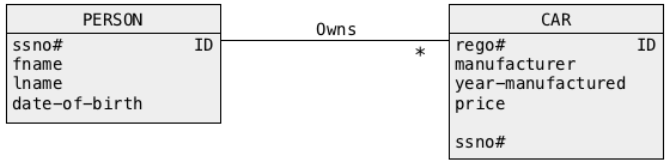
- "Link-by-reference" is the other implementation of association (object-oriented, object-relational database models)

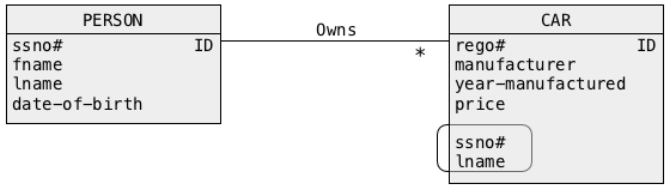
Denormalizations

Outline

Migration of attributes from "one" to "many" side

It is always possible to copy an attribute from "one" side of an association to "many" side (and NOT the opposite) of the association





Performance related observations:

Copying nonID attributes allows for elimination of join operations in some queries

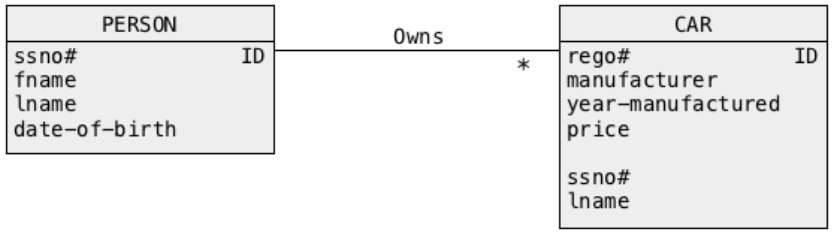
Copying nonID attributes destroys normalization and introduces redundancies

Copying nonID attributes is called as denormalization

Migration of attributes from "one" to "many" side

Applications

- Find the last names of people who own a car manufactured in year 2000
- Find how old are the cars owned by Smith



Performance related observations:

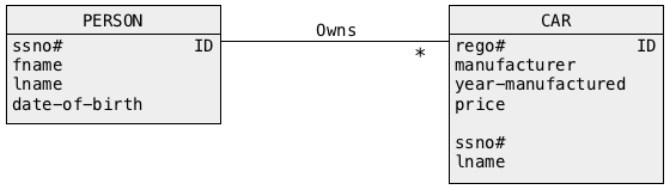
To process both queries there is no need to join the relational tables `PERSON` and `CAR`

A relational table `CAR` is extended by a redundant column `lname`

If a relational table `CAR` contains `n` rows then `n * size(lname)` extra bytes are required to implement the table

Migration of attributes from "one" to "many" side

What about normalization ?



Functional dependencies:

`CAR:`

`rego# → manufacturer`

...

`rego# → ssno#`

`ssno# → lname`

Minimal key: (`rego#`)

2NF because every nonprime attribute is fully functionally dependent on a minimal key

Migration of attributes from "one" to "many" side

What about normalization ?

Functional dependencies:

`CAR:`

`rego# → manufacturer`

...

`rego# → ssno#`

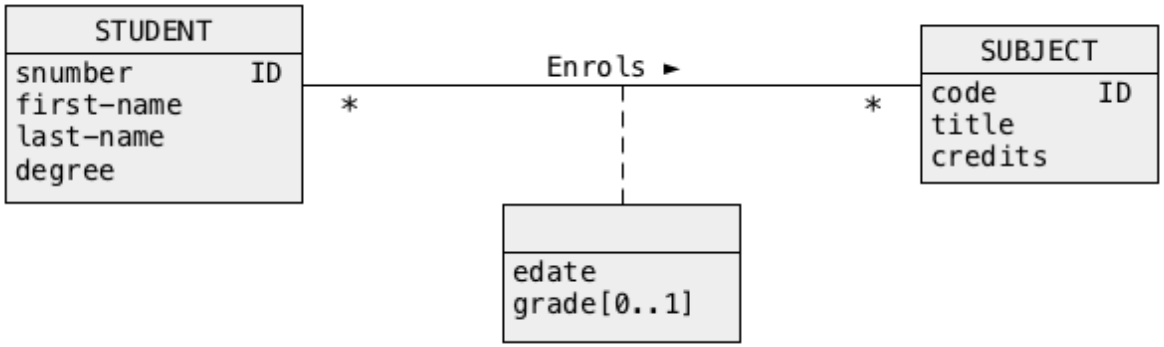
`ssno# → lname`

Minimal key: (`rego#`)

Not 3NF because of transitive functional dependencies
`rego# → ssno#` and `ssno# → lname`

Migration of attributes from "one" to "many" side

Example



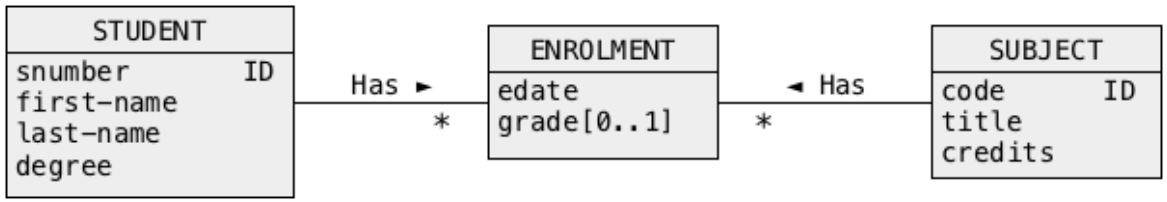
Assume that we would like to speed up processing of the following classes of queries:

- Find the titles of subjects enrolled by the students who belong to a given degree
- Find the names of degrees taken by the students who enrolled a subject with a give title

The attributes `title` and `degree` must be copied to the same class

Migration of attributes from "one" to "many" side

After simplification

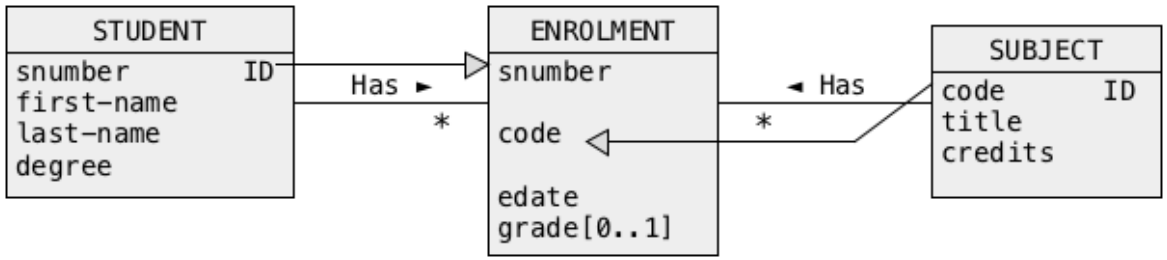


Many-to-many association described by the link attributes `edate` and `grade[0..1]` is replaced with a class `ENROLMENT` and "one-to-many" association `STUDENT Has ENROLMENT` and "one-to-many" association `SUBJECT Has ENROLMENT`

A class `ENROLMENT` is described by the attributes `edate` and `grade[0..1]`

Migration of attributes from "one" to "many" side

After implementation of associations

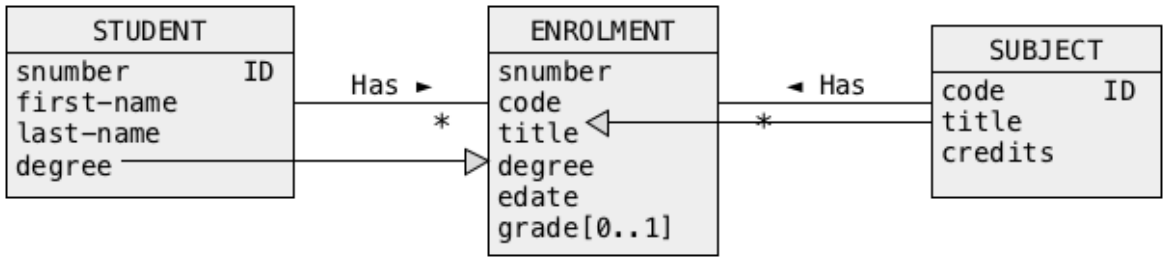


An identifier `snumber` of a class `STUDENT` is copied to a class `ENROLMENT`

An identifier `code` of a class `SUBJECT` is copied to a class `ENROLMENT`

Migration of attributes from "one" to "many" side

After denormalization



An attribute `degree` in a class `STUDENT` is copied to a class `ENROLMENT`

An attribute `title` in a class `SUBJECT` is copied to a class `ENROLMENT`

Now, the attributes `degree` and `title` can be used in the same query without the joins of relational tables

The design is redundant because the values of atributes `degree` and `title` are repeated each time a student enrols a subject