COMP3311 Week 03 Lecture

ER-Relational Mapping (cont)

1/28

So far, have considered mappings for ...

- ER attribute → relational attribute
- ER entity → relational table
- ER key → primary key for table
- n:m relationship → relational table
  (with foreign key for each participating entity plus relationship attributes)
- 1:n relationship → foreign key plus relationship attributes
- 1:1 relationship → foreign key plus relationship attributes

n-way Relationships

2/28

Relationship mappings above assume binary relationship.

If multiple entities are involved:

- *n:m* generalises naturally to *n:m:p:q*
  - include foreign key for each participating entity
  - include any other attributes of the relationship
- other multiplicities (e.g. *1:n:m*) ...
  - need to be mapped the same as *n:m:p:q*
  - so not quite an accurate mapping of the ER

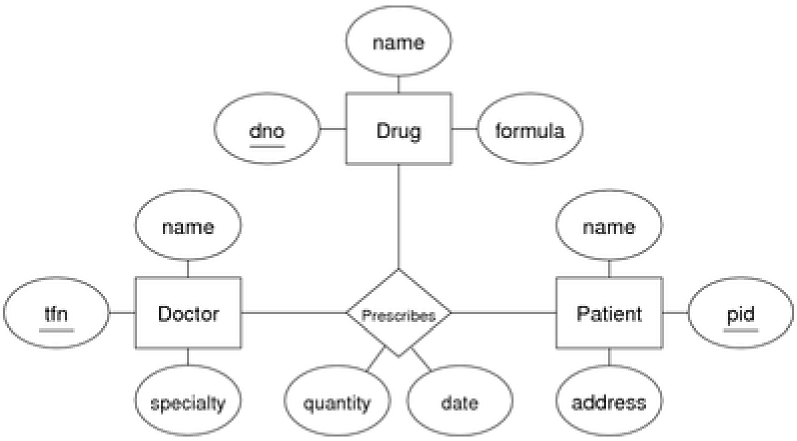
Some people advocate converting n-way relationships into:

- a new entity, and a set of *n* binary relationships

Exercise: 3-way relationship

3/28

Translate the following ER design to a relational schema:

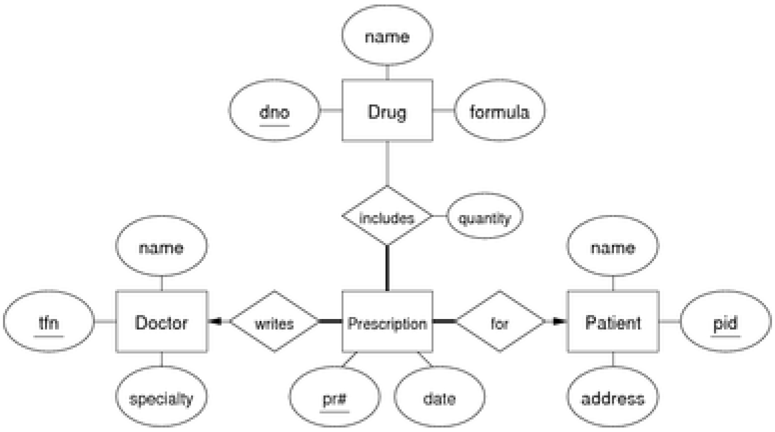


[Solution]

Exercise: Alternative prescription model

4/28

Translate the following ER design to a relational schema:



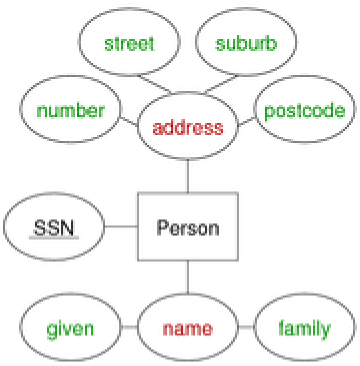
[Solution]

Mapping Composite Attributes

Composite attributes are mapped by concatenation or flattening.

Example:

ER Model



Relational Version #1

Person

SSN	name	address

Relational Version #2

Person

SSN	givenName	familyName	...

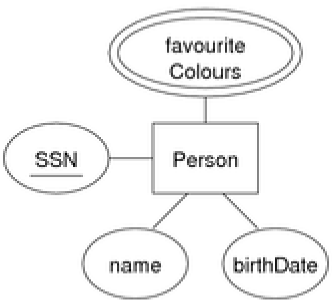
...	number	street	suburb	postcode

Mapping Multi-valued Attributes (MVAs)

MVAs are mapped by a new table linking values to their entity.

Example:

ER Model



Relational Version

Person

SSN	name	birthdate

FavColour

SSN	colour

... Mapping Multi-valued Attributes (MVAs)

Example: the two entities

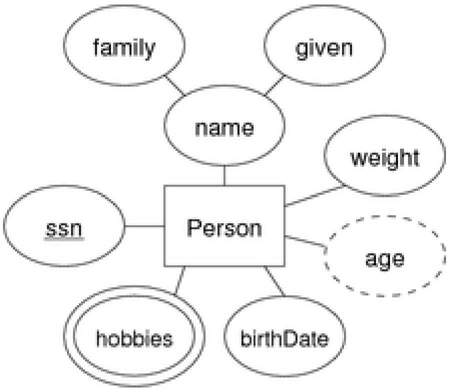
```
Person(12345, John, 12-feb-1990, [red,green,blue])
Person(54321, Jane, 25-dec-1990, [green,purple])
```

would be represented as

```
Person(12345, John, 12-feb-1990)
Person(54321, Jane, 25-dec-1990)
FavColour(12345, red)
FavColour(12345, green)
FavColour(12345, blue)
FavColour(54321, green)
FavColour(54321, purple)
```

Exercise: Attribute Mappings

Convert this ER design to relational form:



[Solution]

Mapping Subclasses

Three different approaches to mapping subclasses to tables:

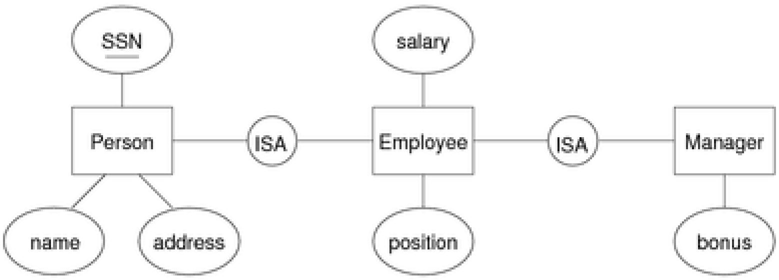
- ER style
  - each entity becomes a separate table,
  - containing attributes of subclass + FK to superclass table
- object-oriented
  - each entity becomes a separate table,
  - inheriting all attributes from all superclasses
- single table with nulls
  - whole class hierarchy becomes one table,
  - containing all attributes of all subclasses (null, if unused)

Which mapping is best depends on how data is to be used.

... Mapping Subclasses

Example of ER-style mapping:

Entity–Relationship Model



Relational Model

Person

<u>**SSN**</u>	**name**	**address**

Employee

<u>**SSN**</u>	**salary**	**position**

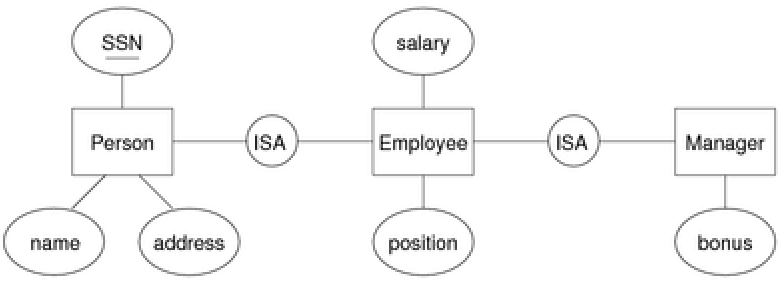
Manager

<u>**SSN**</u>	**bonus**

... Mapping Subclasses

Example of object-oriented mapping:

Entity–Relationship Model



Relational Model

Person

<u>**SSN**</u>	**name**	**address**

Employee

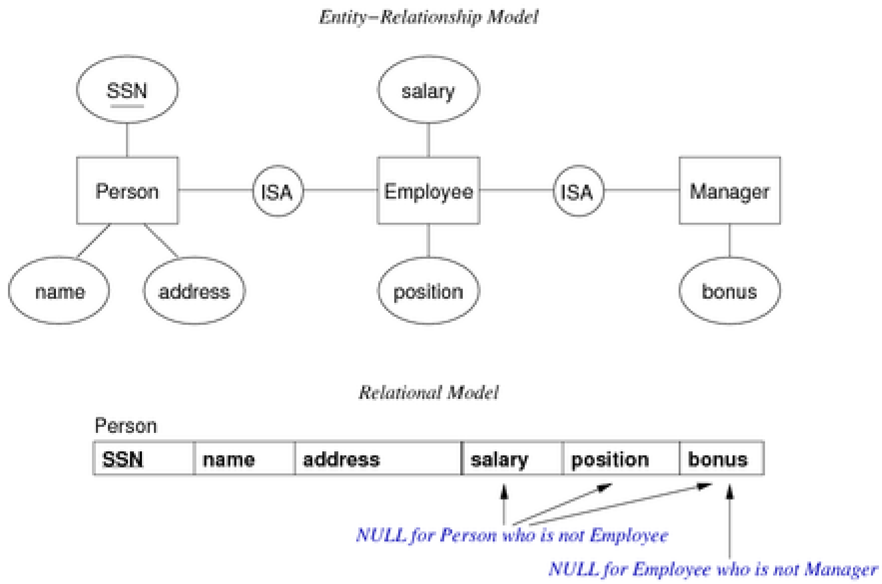
<u>**SSN**</u>	**name**	**address**	**salary**	**position**

Manager

<u>**SSN**</u>	**name**	**address**	**salary**	**position**	**bonus**

... Mapping Subclasses

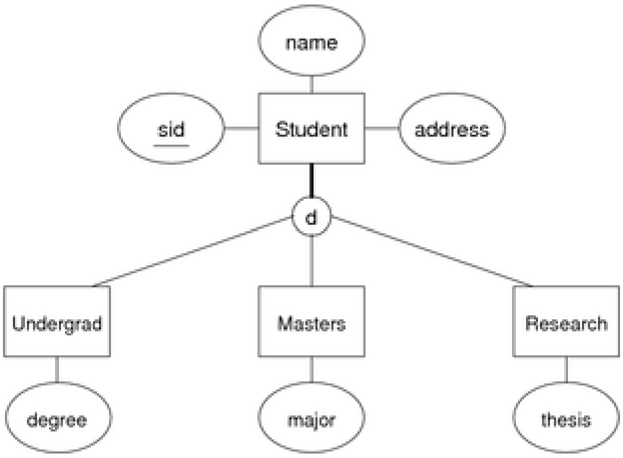
Example of single-table-with-nulls mapping:



Exercise: Disjoint subclasses

13/28

Translate the following ER design to a relational schema:



Use (a) ER-mapping, (b) OO-mapping, (c) 1-table-mapping

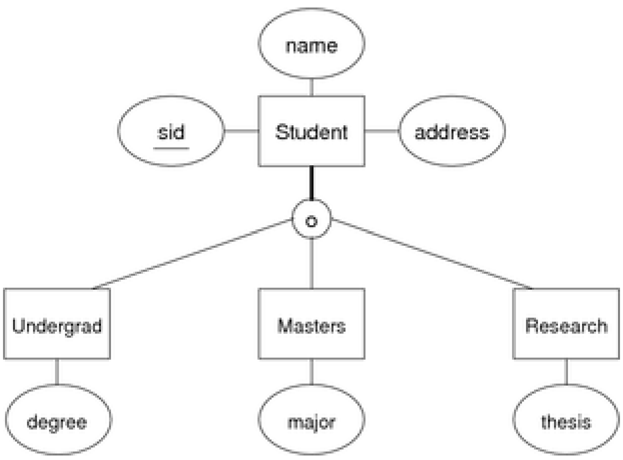
Are there aspects of the ER design that can't be mapped?

[Solution]

Exercise: Overlapping subclasses

14/28

Translate the following ER design to a relational schema:



Use (a) ER-mapping, (b) OO-mapping, (c) 1-table-mapping

Are there aspects of the ER design that can't be mapped?

[Solution]

Relational DBMSs

What is an RDBMS?

16/28

A *relational database management system* (RDBMS) is

- software designed to support large-scale data-intensive applications
- allowing high-level description of data (tables, constraints)
- with high-level access to the data (relational model, SQL)
- providing efficient storage and retrieval (disk/memory management)
- supporting multiple simultaneous users (privilege, protection)
- doing multiple simultaneous operations (transactions, concurrency)
- maintaining reliable access to the stored data (backup, recovery)

Note: databases provide *persistent* storage of information

Describing Data

17/28

RDBMSs implement ≅ the relational model.

Provide facilities to define:

- domains, attributes, tuples, tables
- constraints (domain, key, referential)

Variations from the relational model:

- no strict requirement for tables to have keys
- bag semantics, rather than set semantics
- no standard support for general (multi-table) constraints

RDBMS Operations

18/28

RDBMSs typically provide at least the following:

- create/remove a database or a schema
- create/remove/alter tables within a schema
- insert/delete/update tuples within a table
- queries on data, define named queries (views)
- transactional behaviour (ACID)

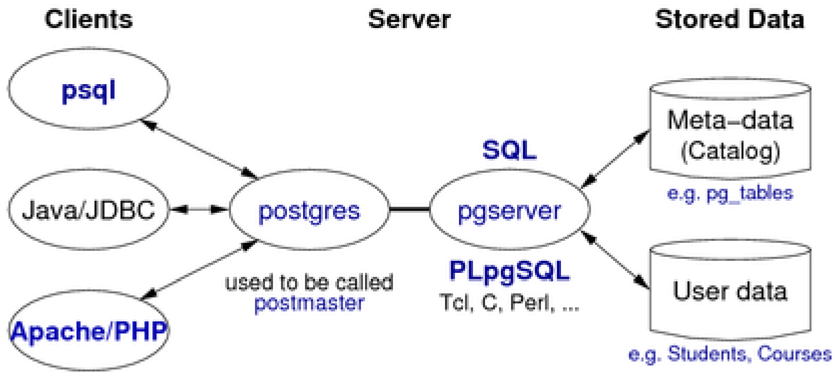
Most also provide mechanisms for

- creating/managing users of the database
- defining/storing procedural code to manipulate data
- implementing complex constraints (triggers)
- defining new data types and operators (less common)

PostgreSQL Architecture

19/28

PostgreSQL's client-server architecture:



Using PostgreSQL

Using your PostgreSQL server in CSE (once installed):

- login to `grieg`, set up environment, start server
- use `psql`, etc. to manipulate databases
- stop server, log off `grieg`

```
wagner$ ssh YOU@grieg
grieg$ priv srvr
grieg$ source /srvr/YOU/env
grieg$ pg start
grieg$ psql mydb
... do stuff with your database ...
grieg$ pg stop
grieg$ exit
```

... Using PostgreSQL

PostgreSQL files (helps to understand state of server)

- PostgreSQL home directory ... `/srvr/YOU/pgsql/`
- under the home directory ...
  - `postgresql.conf` ... main configuration file
  - `base/` ... subdirectoriess containing database files
  - `postmaster.pid` ... process ID of server process
  - `.s.PGSQL.5432` ... socket for clients to connect to server
  - `.s.PGSQL.5432.lock` ... lock file for socket
- PostgreSQL environment settings ... `/srvr/YOU/env`

Building/Maintaining Databases

Managing Databases

Shell commands:

- **`createdb`** *dbname*
- **`dropdb`** *dbname*

(If no *dbname* supplied, assumes a database called *YOU*)

SQL statements:

- **`CREATE DATABASE`** *dbname*
- **`DROP DATABASE`** *dbname*

(Neither of the above is SQL-standard)

... Managing Databases

Shell commands (dump/restore):

- `pg_dump` *dbname* `>` *dumpfile*
- `psql` *dbname* `-f` *dumpfile*

(Database *dbname* is typically created just before restore)

SQL statements (used in *dumpfile*):

- `CREATE TABLE` *table* `(` *Attributes+Constraints* `)`
- `ALTER TABLE` *table* *TableSchemaChanges*
- `COPY` *table* `(` *AttributeNames* `)` `FROM STDIN`

Managing Tables

SQL statements:

- `ALTER TABLE` *table* *TableSchemaChanges*
- `DROP TABLE` *table(s)* [`CASCADE`]
- `TRUNCATE TABLE` *table(s)* [`CASCADE`]

(All conform to SQL standard, but all also have extensions)

`DROP..CASCADE` drops objects which depend on the table

`TRUNCATE..CASCADE` truncates tables which refer to the table

Managing Tuples

SQL statements:

- `INSERT INTO` *table* `(` *attrs* `)` `VALUES` *tuple(s)*
- `DELETE FROM` *table* `WHERE` *condition*
- `UPDATE` *table* `SET` *AttrValueChanges* `WHERE` *condition*

AttrValueChanges is a comma-separated list of:

- *attrname* `=` *expression*

Each list element assigns a new value to a given attribute.

Exercise: Generating IDs

Consider the following schema:

```
create table T (
    id serial primary key,
    x  integer,
    y  varchar(10)
);
```

- what does `serial` actually produce (look in the catalog)?
- write `INSERT` statements to add some tuples
- how could an application program get the generated `id`?
  (`select max(id) from T` may not give the correct result; why not?)

Managing Other DB Objects

Databases contain objects other than tables and tuples:

- views, functions, sequences, types, indexes, roles, ...

Most have SQL statements for:

- `CREATE` *ObjectType* *name* ...

- `DROP` *ObjectType* *name* ...

Views and functions also have available:

- `CREATE OR REPLACE` *ObjectType* *name* ...

See PostgreSQL documentation Section IV, Chapter I for SQL statement details.

Produced: 12 March 2018