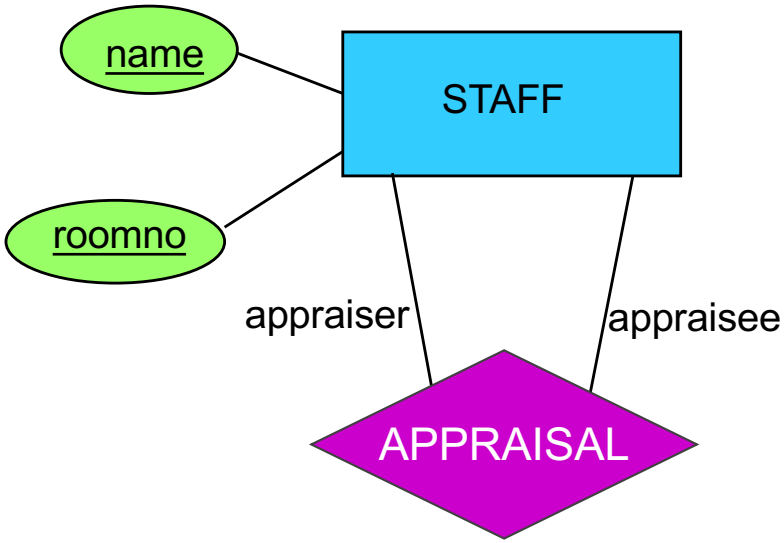
Mapping Roles and Recursive Relationships

How can the entity STAFF appear in both of its roles ?



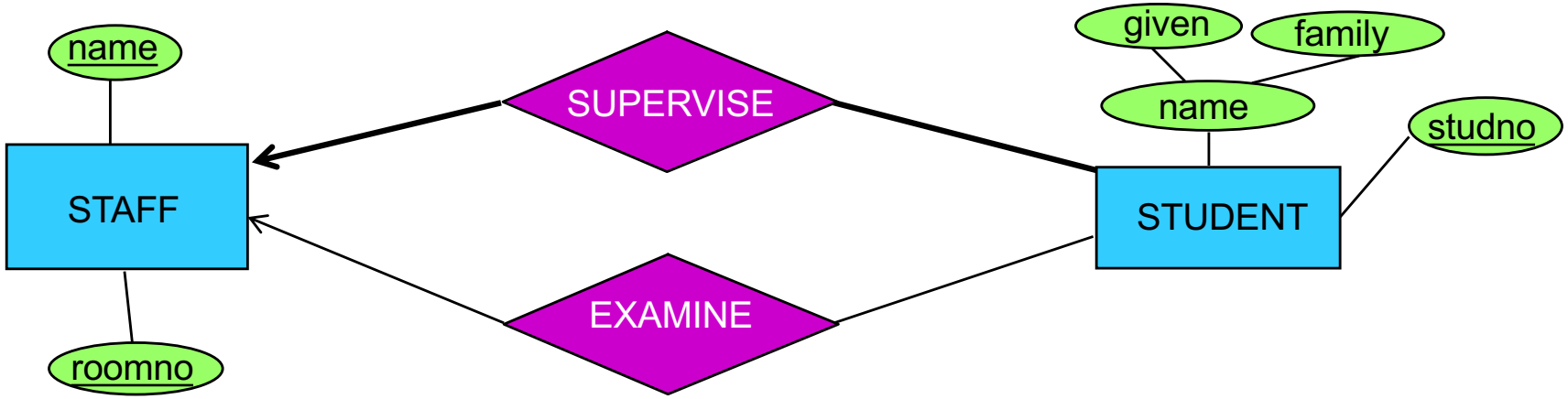
STAFF(name, roomno, appraiser, approomno)

Other option??

Multiple Relationships between Entity Types

Treat each relationship type separately

Represent distinct relationships by different foreign keys drawing on the same relation

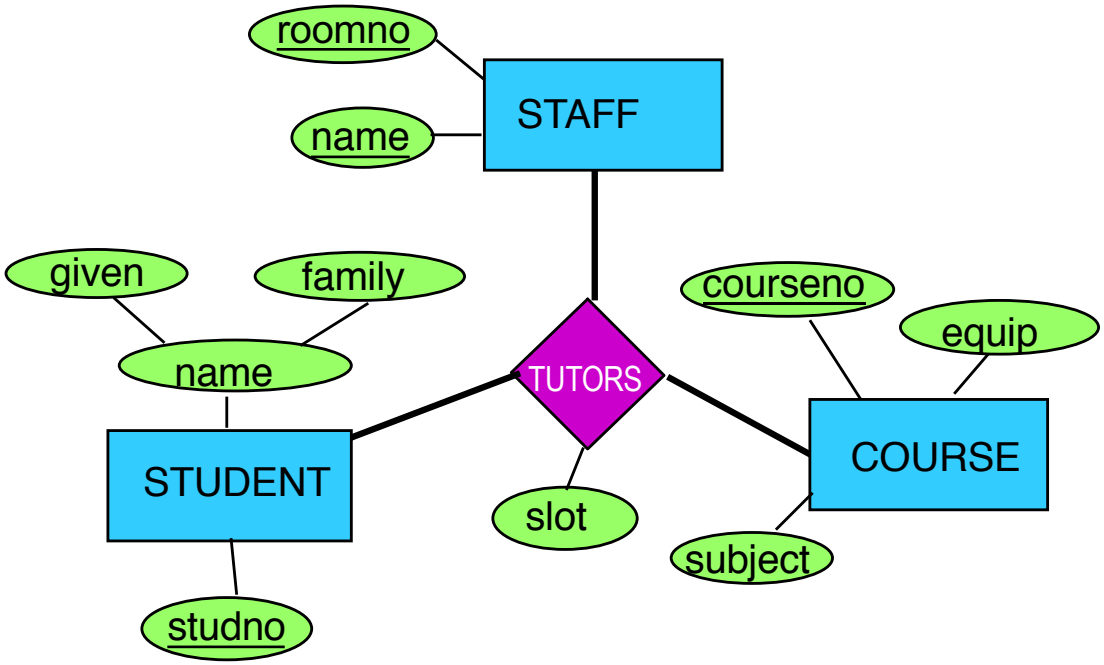


STAFF(<u>name</u>, <u>roomno</u>)

STUDENT(<u>studno</u>, given, family, *???*)

STUDENT(<u>studno</u>, given, family, *???*)
EXAMINER(*???*)
SUPERVISOR(*???*)
EXAMINER_SUPERVISOR(???)

Non-binary Relationship



COURSE(courseno, subject, equip)

STUDENT(studno, givenname, familyname)

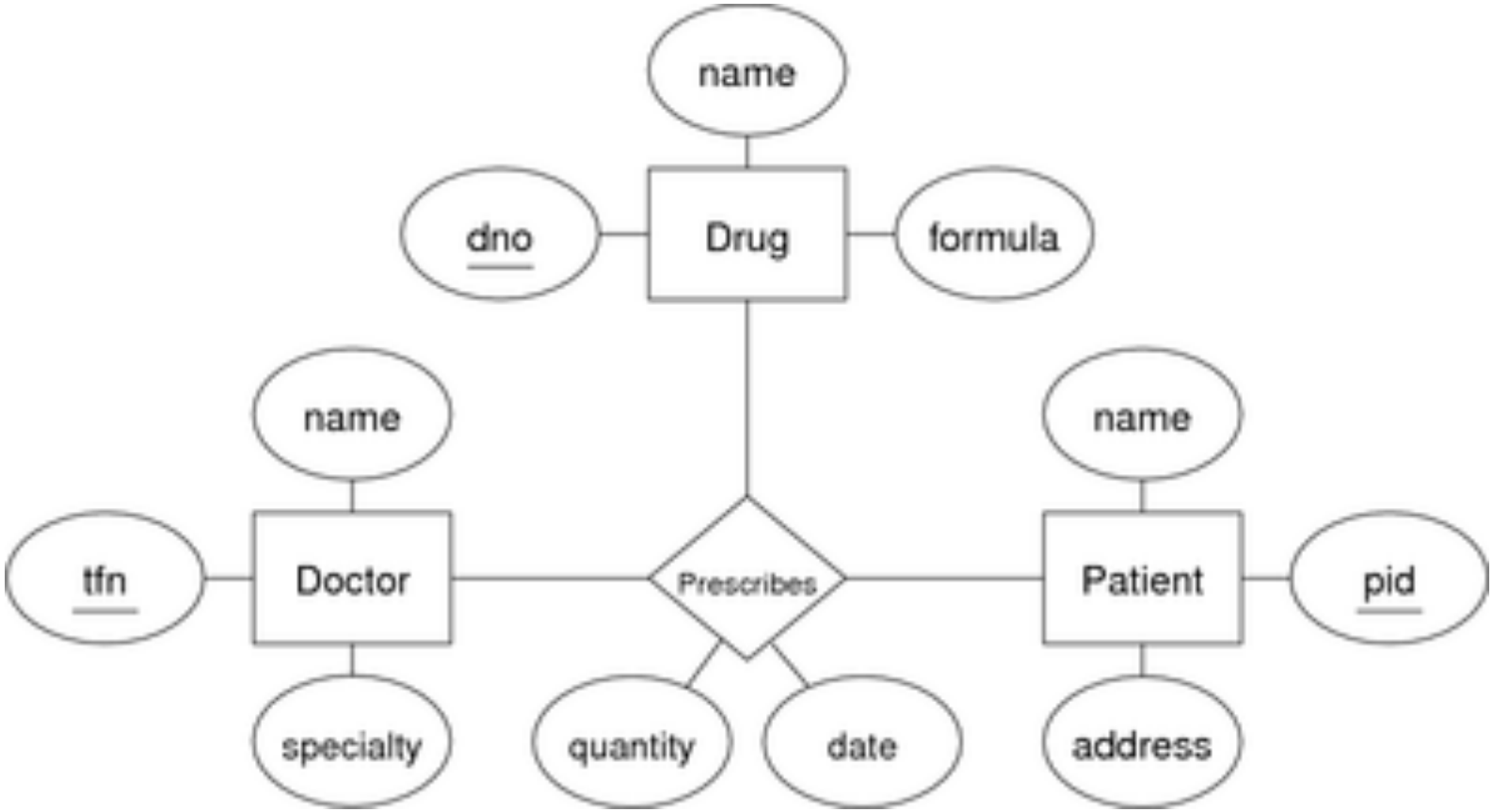
STAFF(staffname, roomno)

TUTORS(*???*)

Non-binary relationship

Some people advocate converting n-way relationships into:a new entity, and a set of *n* binary relationships

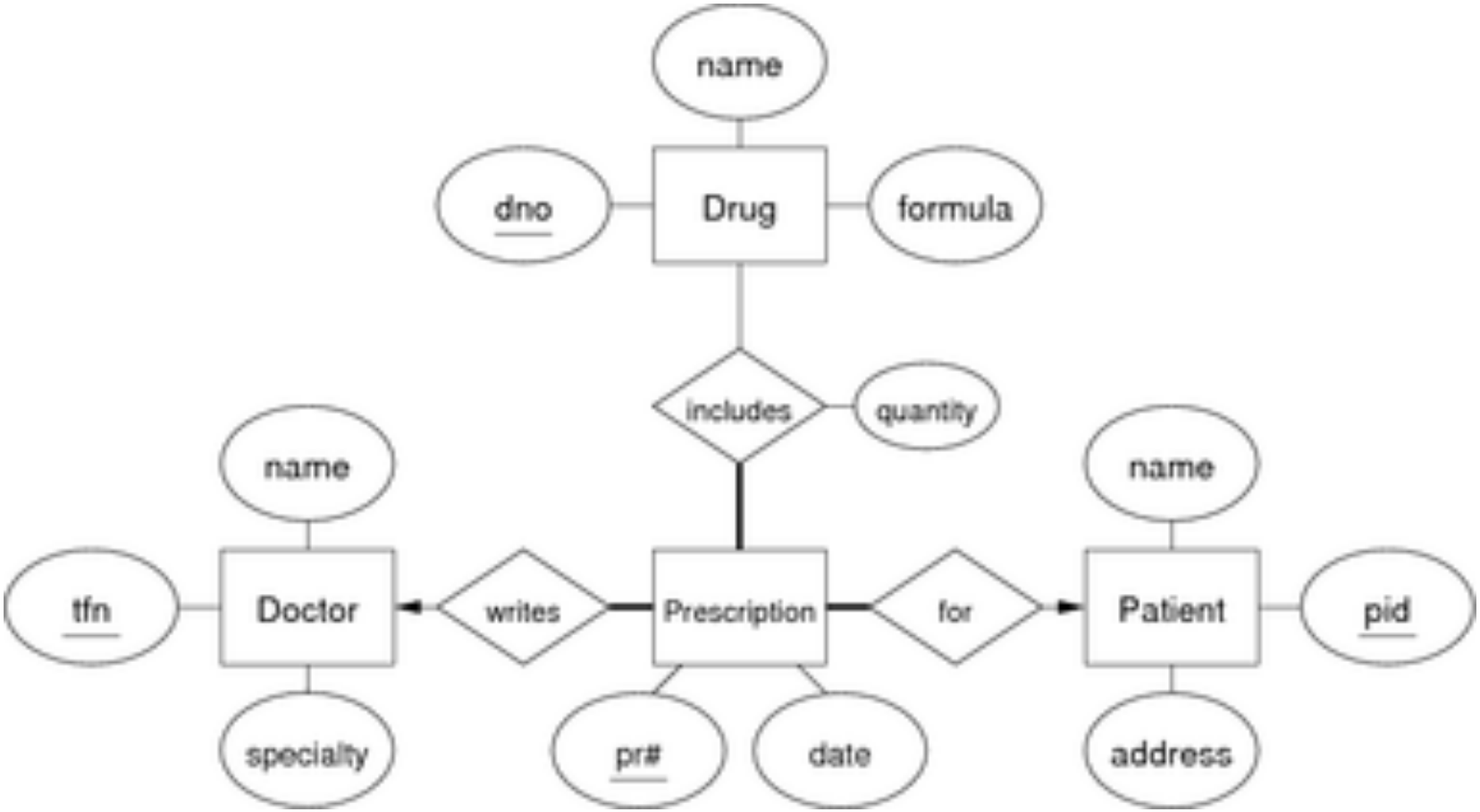
Translate this:



Non-binary relationship

Converting the previous model to binary relationships …

Translate this:



Mapping Weak Entities to Relations

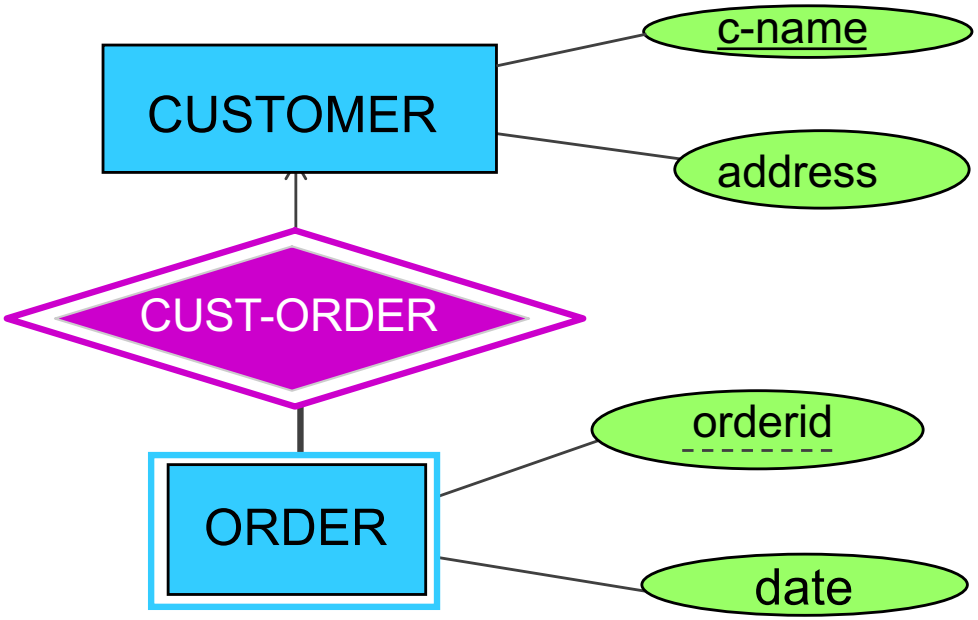
Create a relation with the attributes:

$$\text{primary\_key}(E_0) \cup \bigcup_{i=1}^{n} \text{discriminator}(E_i) \cup \{a_1,\ldots,a_n\}$$

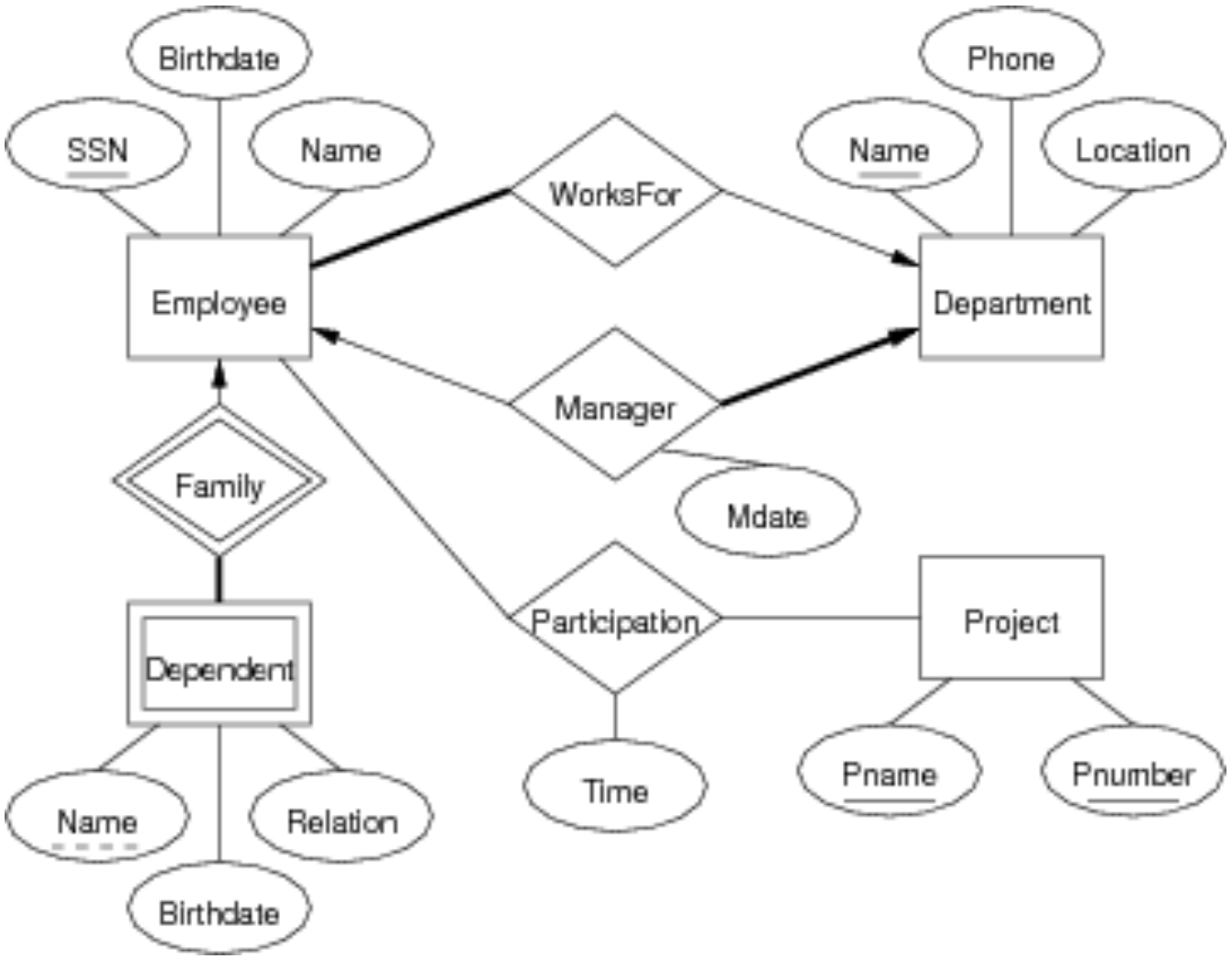
Primary key of identifying strong entity type

Discriminators of identifying weak entity types

Attributes of the weak entity type



Exercise 2


Birthdate
SSN
Name
WorksFor
Phone
Name
Location
Employee
Department
Manager
Family
Mdate
Participation
Project
Dependent
Time
Pname
Pnumber
Name
Relation
Birthdate

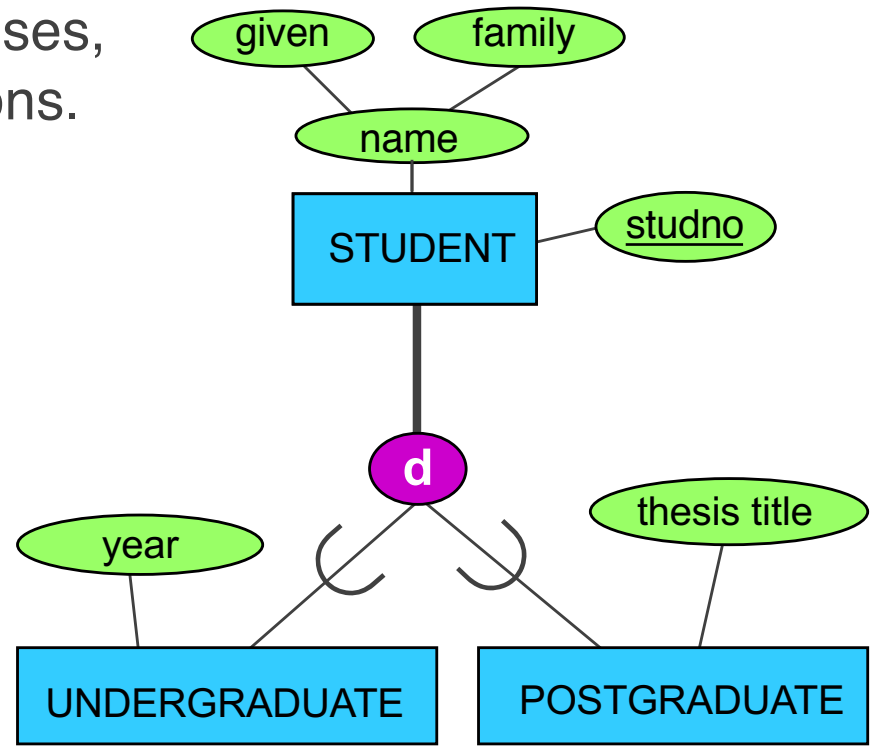
Translating of Hierarchies: Options

To store information about these classes,
We have to define appropriate relations.

For each relation, we have to define:

- set of attributes
- primary key

In principle, there are
three options:



A. Create a relation for each entity type in the schema,
i.e., for both, superclass and subclasses

B. *Create only relations for subclasses (not an option for some cases)*

C. Create only one relation, for the superclass

Translating of Hierarchies: Options

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

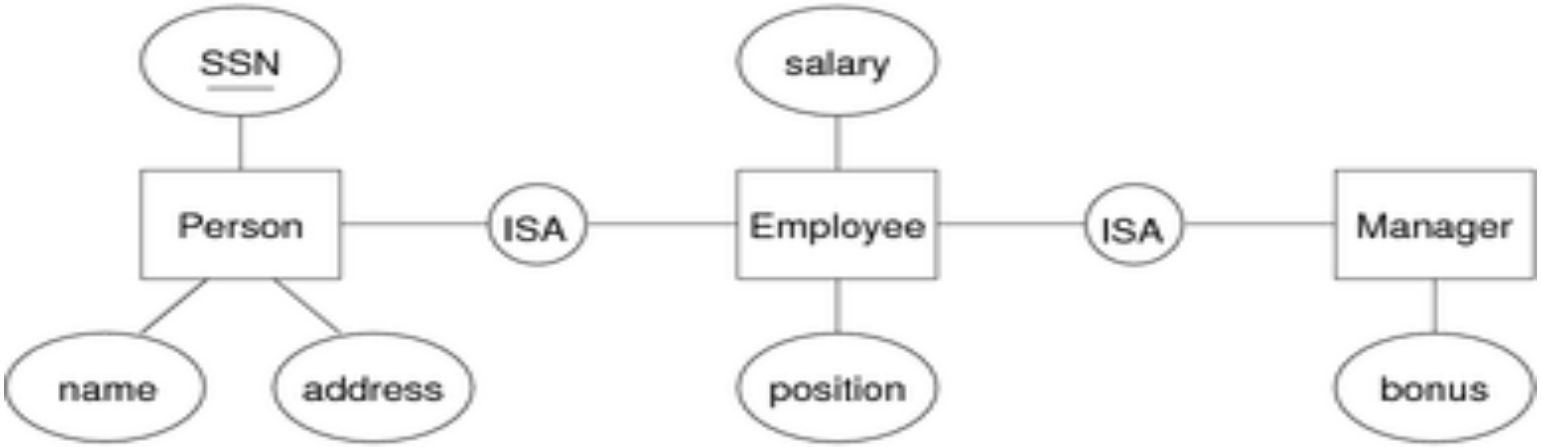
- whole class hierarchy becomes one table,
- containing all attributes of all subclasses (null, if unused)

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

Translating of Hierarchies: Options

An example of ER Style mapping:

Entity–Relationship Model



Relational Model

Person

SSN	name	address

Employee

SSN	salary	position

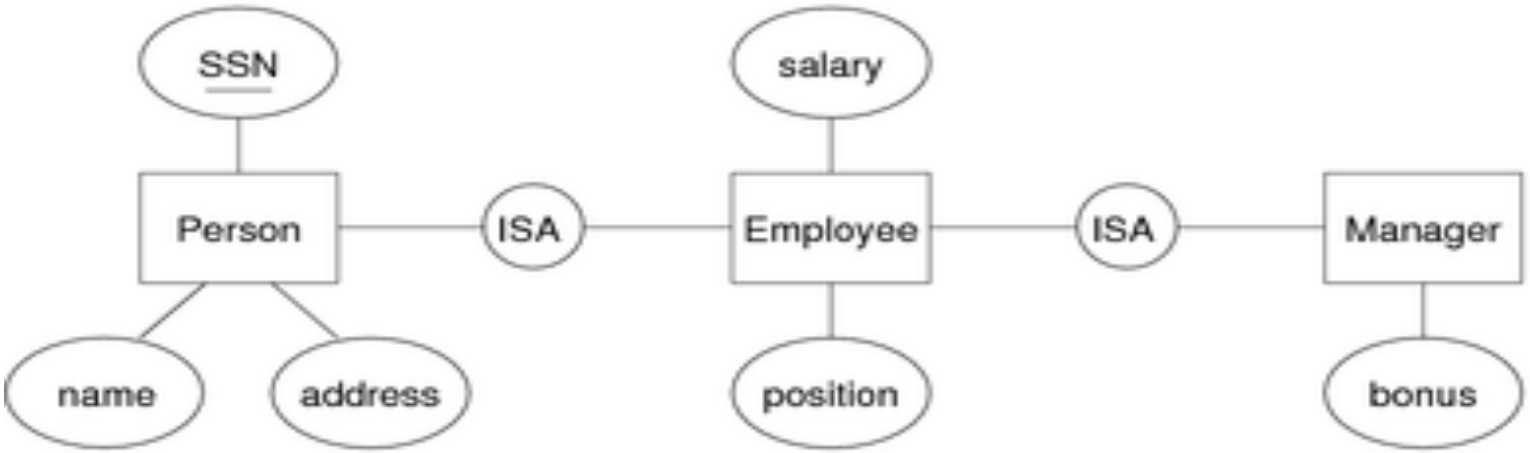
Manager

SSN	bonus

Translating of Hierarchies: Options

An example of O-O Style mapping:

Entity–Relationship Model



Relational Model

Person

SSN	name	address

Employee

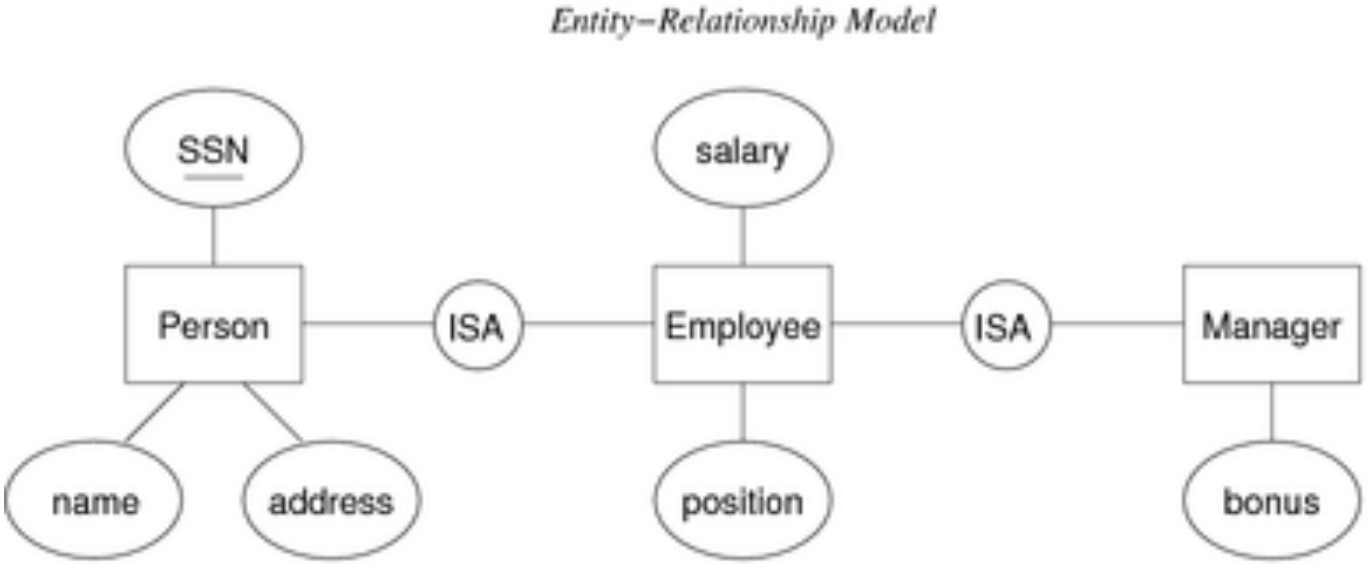
SSN	name	address	salary	position

Manager

SSN	name	address	salary	position	bonus

Translating of Hierarchies: Options

Example of single-table-with-nulls mapping:



Relational Model

Person

SSN	name	address	salary	position	bonus

NULL for Person who is not Employee

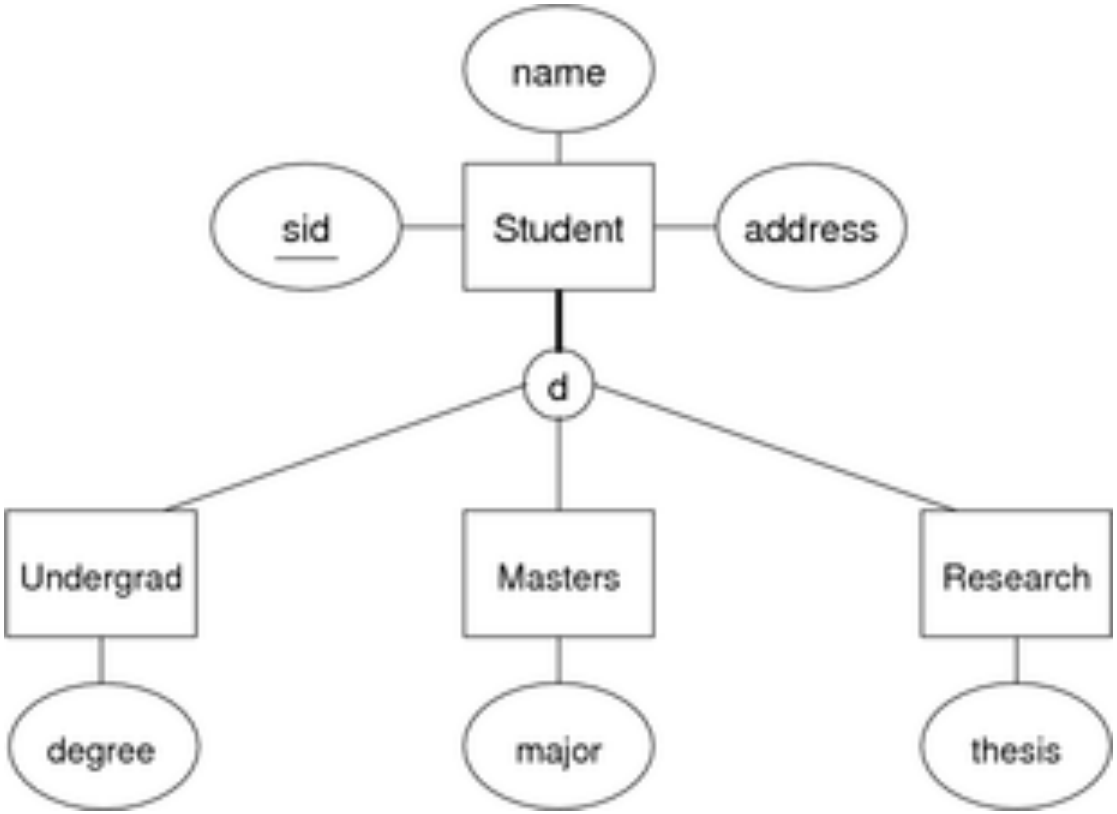
NULL for Employee who is not Manager

Also we can consider adding a 'class' attribute

Exercise: mapping disjoint subclasses

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

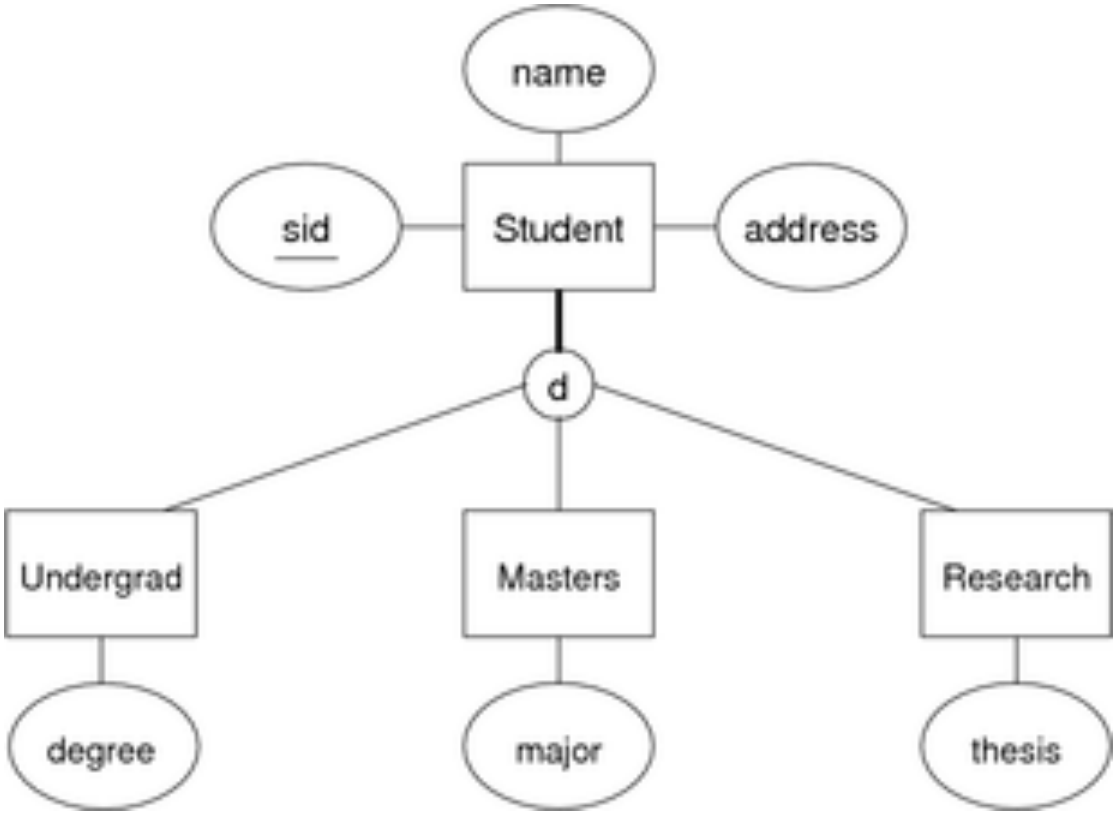
Are there aspects of the ER design that can't be mapped?



Exercise: mapping overlapping subclasses

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

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



What is RDBMS?

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

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

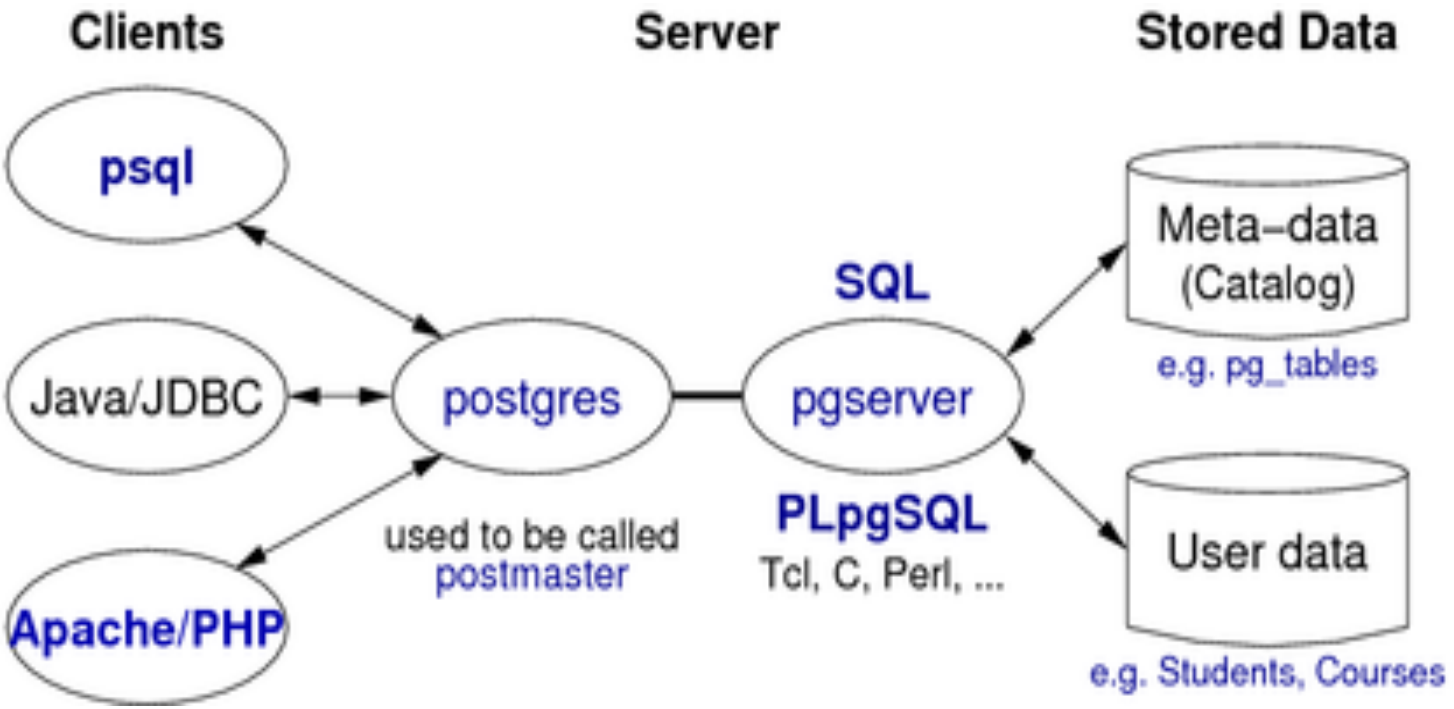
RDBMSs typically provide at least the following:

- 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 forcreating/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 client-server architecture



PostgreSQL client-server architecture

PostgreSQL files (helps to understand state of server)

PostgreSQL home directory ... /srvr/*YOUR_ID*/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/*YOUR_ID*/env

Managing Databases

Shell commands:

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

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

SQL statements (inside dbname)

- **CREATE TABLE** *table* **(** *Attributes+Constraints* **)**
- **ALTER TABLE** *table* *TableSchemaChanges*
- **DROP TABLE** *table(s)* [**CASCADE**]
- **COPY** *table* **(** *AttributeNames* **) FROM STDIN**
- **INSERT INTO** *table* **(** *attrs* **) VALUES** *tuple(s)*
- **DELETE FROM** *table* **WHERE** *condition*
- **UPDATE** *table* **SET** *AttrValueChanges* **WHERE** *condition*

SQL

- Originally "Structured Query Language", today a proper name
- A language with several functionalities: comprises both DDL and DML
- There exist several standards, and companies have added proprietary extensions
- Commercial systems offer features that are not part of the standard
  - Incompatibilities between systems
  - Incompatibilities with newer standards (e.g. triggers in SQL:1999)
- “History”:
  - First proposal of SEQUEL (IBM Research, 1974)
  - First implementation in SQL/DS (IBM) and Oracle (1981)
  - Since around 1983 there is a “de facto standard”
  - Standard definitions (ISO): 1986, then 1989, then 1992, thereafter 1999 (e.g. triggers, oo features), 2003, 2006 (XML)

Data Definitions in SQL

Apart from the command create schema (which is used to create a schema), the most important command of the DDL in SQL is create table

- Defines a relation schema (with attributes and integrity constraints)
- Creates an empty instance of the schema

```
create table Employee (
   EmpNo              character(6) primary key,
   FirstName          character(20) not null,
   LastName           character(20) not null,
   Dept               character(15),
   Salary             numeric(9) default 0,
   City               character(15),
   foreign key(Dept) references Department(DeptName),
   unique (LastName,FirstName)
)
```

SQL and the Relational Model

Difference: a table instance in SQL is defined as a multiset (bag) of tuples.

In particular, if a table does not have a primary key or a set of attributes that are defined as unique, it is possible that two identical tuples appear in an instance of that table.

Thus, in general an SQL table is not a relation.

If, however, a table has a primary key or a set of attributes that are defined as unique, there can never be two identical tuples in a relation.

$\rightarrow$ It is advisable to define at least a primary key for a relation.

Domains

Elementary domains or types (predefined)

- Character: single characters or strings → both of fixed and variable length
- Bitstrings: string elements are 0 and 1
- Numbers: integers and reals
- Dates, timestamps, time intervals
- Introduced in SQL:1999
  - Boolean
  - BLOB, CLOB (binary/character large object):
    for large images or texts
- In some systems, enumeration types can be defined

User defined domains (reusable)

Domain Definitions

create domain

- defines a (simple) domain with integrity constraints and defaults, which can be reused in table definitions.
- data types based on a constrained version of an existing data type;

Syntax

- create domain DomainName
- as Type [Default] [IntegrityConstraint]

Example:

```
create domain EmployeeAge
as smallint default null
check ( value >=18 and value <= 67 )
```

Constraints on a Relation

not null (on single attributes)

unique: allows one to define a (candidate) key:

- single attribute: unique after the specification of the domain
- several attributes (i.e., one or more): unique ($Attr_1$,..., $Attr_n$)

primary key: definition of the primary key
(only one, implies not null); syntax as for unique

Constraints on a Relation (Example)

```
create table Employee (
   EmpNo              character(6) primary key,
   FirstName          character(20) not null,
   LastName           character(20) not null,
   Dept               character(15),
   Salary             numeric(9) default 0,
   City               character(15),
   foreign key (Dept) references Department(DeptName),
   unique (LastName,FirstName)
)
```

primary key (Alternate Definition)

```
create table Employee (
   EmpNo character(6) primary key,
   ...
)
```

or

```
create table Employee (
   EmpNo character(6),
   ...
   primary key (EmpNo)
)
```

Candidate Keys: Mind the Step!

```
create table Employee ( ...
   FirstName  character(20) not null,
   LastName   character(20) not null,
   unique (LastName,FirstName)
)
```

is different from:

```
create table Employee ( ...
   FirstName  character(20) not null unique,
   LastName   character(20) not null unique
)
```

Constraints Between Relations

references and **foreign key** allow one to define referential integrity constraints.

Syntax:

- for single attributes:

  references after the specification of the domain

- for several attributes:

  foreign key($Attr_1$,...,$Attr_n$) references ...

The referenced attributes in the target table must form a key (primary key or unique). If they are missing, the foreign key refers to the primary key of the target table.

Semantics: every combination (not involving NULL) of attribute values in the source table must appear in the target table.

It is possible to add policies that specify how to react to constraint violations (which are caused by changes of the target table).

Foreign Keys (Example)

```
create table Student(
 StudNo   character(10) primary key,
 Name     character(20),
 Hons     character(3),
 Tutor    character(20) references Staff(Lecturer),
 Year     smallint)

create table Staff(
 Lecturer  character(20) primary key,
 RoomNo    character(4),
 Appraiser character(20),
 foreign key (Appraiser) references
             Staff(Lecturer)
             on delete set null
             on update cascade)
```

Policies

Determine the effect of delete and update statements

Syntax

- on delete *Action*

  where *Action* can be

cascade	(propagate the deletion)
restrict	(do nothing if the row is referenced)
no action	(as restrict, but return an error)
set default	
set null	

The same actions exist for updates (on update)

Schema Updates

alter domain: allows one to modify a domain definition

alter table: allows one to modify a table

- add or drop attributes
- add or drop constraints

drop domain: eliminates a domain

drop table: eliminates a table

Catalogue or Data Dictionary

Every relational system offers predefined tables that collect data about:

- tables
- attributes
- domains
- ...

For instance, the table Columns contains the attributes:

- Column_Name
- Table_name
- Ordinal_Position
- Column_Default
- …