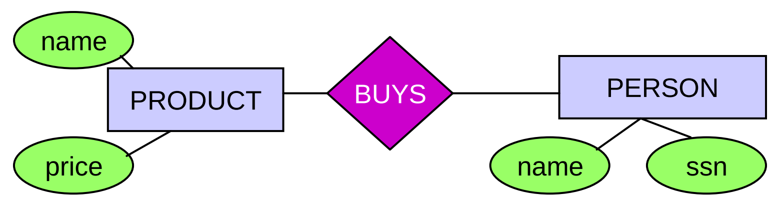
Introduction to Database Systems

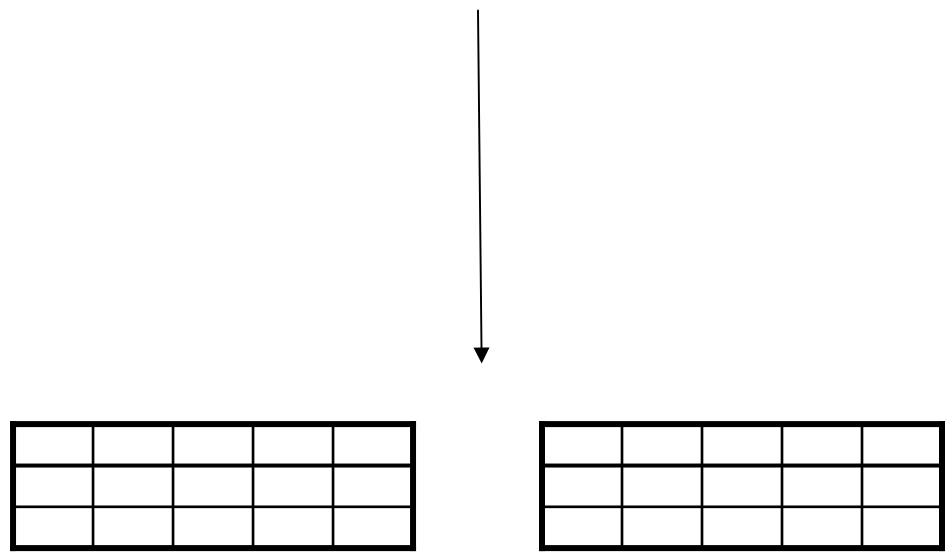
Mapping ER Models to Relational Schemas

Werner Nutt

Conceptual and Logical Design

Conceptual Model:





Relational Model:

Mapping an E-R Diagram to a Relational Schema

We cannot store date in an ER schema

(there are no ER database management systems)

➔ We have to translate our ER schema into a relational schema

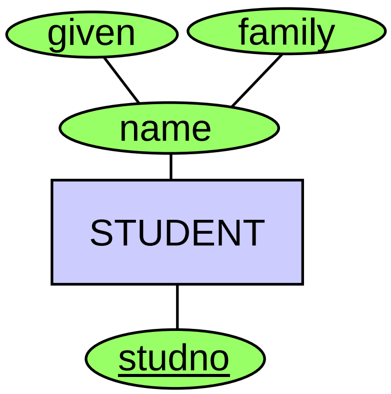
➔ What does "translation" mean?

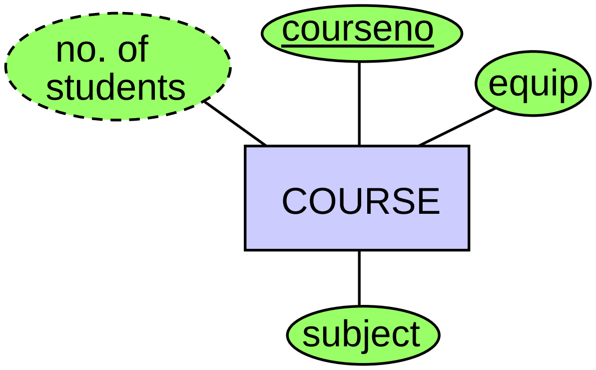
Translation: Principles

- Maps
  - ER schemas to relational schemas
  - ER instances to relational instances
- Ideally, the mapping should
  - be one-to-one in both directions
  - not lose any information
- Difficulties:
  - what to do with ER-instances that have identical attribute values, but consist of different entities?
  - in which way do we want to preserve information?

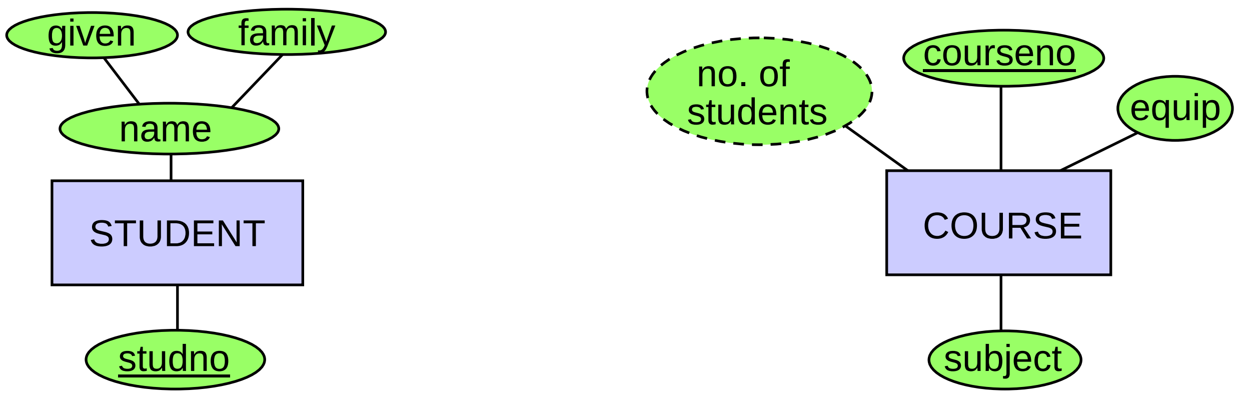
Mapping Entity Types to Relations

- For every *entity type* create a *relation*
- Every *atomic attribute* of the entity type becomes a *relation attribute*
- *Composite attributes*: include *all the atomic attributes*
- *Derived attributes* are not included
  (but remember their *derivation rules*)
- Relation instances are subsets of the cross product of the domains of the attributes
- Attributes of the *entity key* make up the *primary key* of the relation





Mapping Entity Types to Relations (cntd.)



STUDENT (<u>studno</u>, givenname, familyname)

COURSE (<u>courseno</u>, subject, equip)

Mapping Many:many Relationship Types to Relations

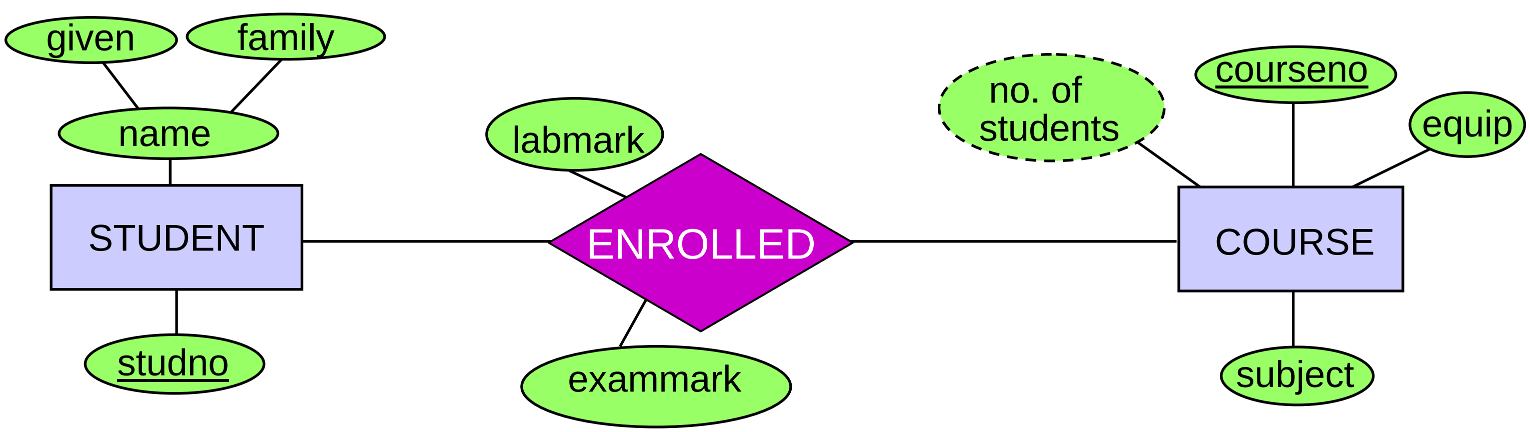
Create a relation with the following set of attributes:

$$\bigcup_{i=1}^{N} \text{primary\_key}(E_i) \ \cup \ \{a_1, \ldots, a_M\}$$

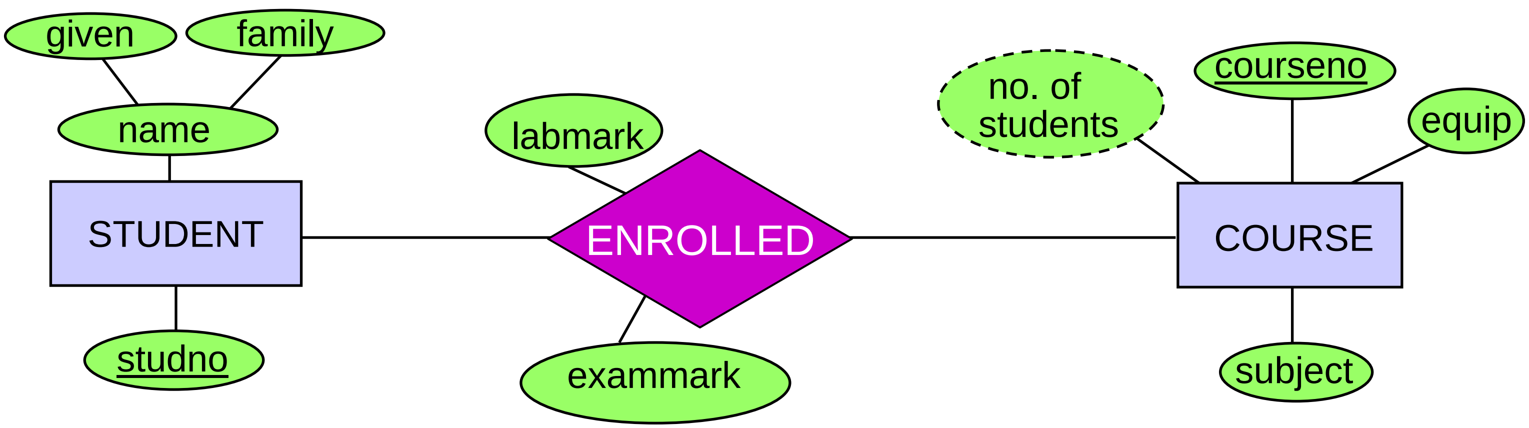
N *(degree of relationship)*

primary keys of each entity type participating in the relationship

attributes of the relationship type (if any)



Mapping Many:many Relationship Types to Relations (cntd.)

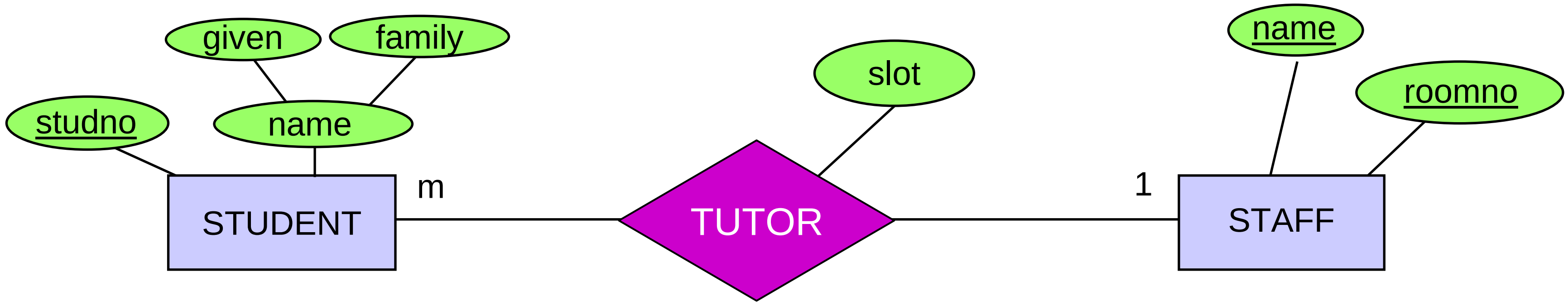


ENROL(<u>studno,</u> <u>courseno</u>, labmark, exammark)

Foreign Key ENROL(studno) references STUDENT(studno)

Foreign Key ENROL(courseno) references COURSE(courseno)

Mapping Many:one Relationship Types to Relations



Idea: *"Post the primary key"*

- Given E1 at the 'many' end of relationship and E2 at the 'one' end of the relationship, add information to the relation for E1
- The primary key of the entity at the 'one' end (the *determined* entity) becomes a foreign key in the entity at the 'many' end (the *determining* entity). Include any relationship attributes with the foreign key entity

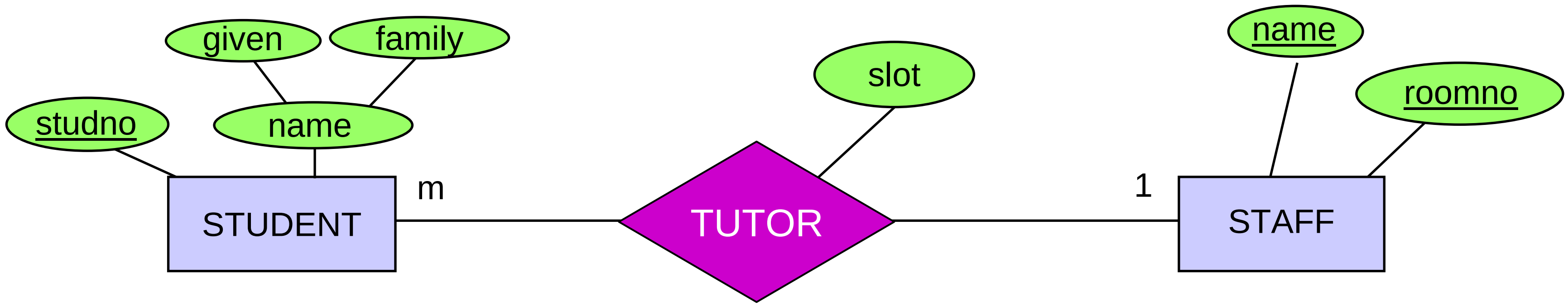
$$E1 \; U \; primary\_key(E2) \; U \; \{a_1,\ldots,a_n\}$$

relation for entity E1

primary key for E2, *is now a* foreign key *to* E2

attributes on the relationship type (if any)

Mapping Many:one Relationship Types to Relations: Example



The relation

STUDENT(studno, givenname, familyname)

is extended to

STUDENT(studno, givenname, familyname, tutor, roomno, slot)

and the constraint

Foreign Key STUDENT(tutor,roomno) references STAFF(name,roomno)

Mapping one:many Relationship Types to Relations (cntd.)

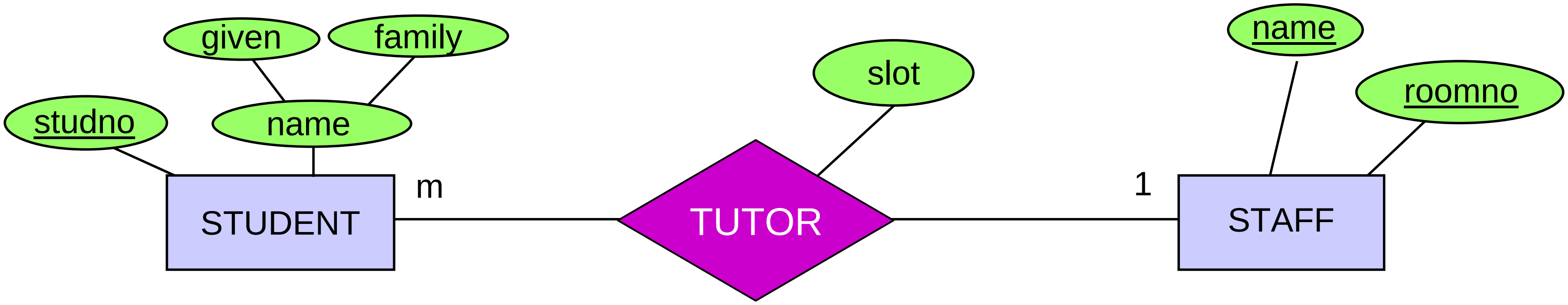
STUDENT

studno	given	family	tutor	roomno	slot
s1	fred	jones	bush	2.26	12B
s2	mary	brown	kahn	IT206	12B
s3	sue	smith	goble	2.82	10A
s4	fred	bloggs	goble	2.82	11A
s5	peter	jones	zobel	2.34	13B
s6	jill	peters	kahn	IT206	12A

STAFF

name	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

Mapping One:many Relationship Types to Relations (cntd.)



Another Idea: If

- the relationship type is *optional* to both entity types, and
- an instance of the relationship is *rare*, and
- there are *many attributes* on the relationship then…

… create a new relation with the set of attributes:

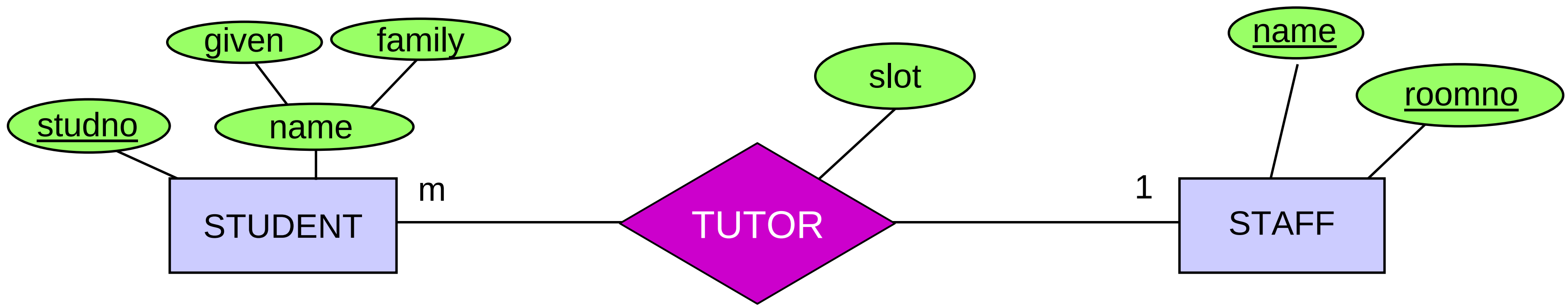
$$\text{primary\_key(E1)} \;\; U \;\; \text{primary\_key(E2)} \;\; U \;\; \{a_1,\ldots,a_m\}$$

primary key for E1*,*
is now a foreign key *to* E1*;*
also the PK for this relation

primary key for E2*,*
is now a foreign key
to E2

attributes on the relationship type (if any)

Mapping One:many Relationship Types to Relations (cntd.)



Compare with the mapping of many:many relationship types!

TUTOR(<u>studno</u>, staffname, rommno, slot)

and

Foreign key TUTOR(studno) references STUDENT(studno)

Foreign key TUTOR(staffname, roomno) references STAFF(name, roomno)

Mapping One:many Relationship Types to Relations (cntd.)

STUDENT

studno	given	family
s1	fred	jones
s2	mary	brown
s3	sue	smith
s4	fred	bloggs
s5	peter	jones
s6	jill	peters

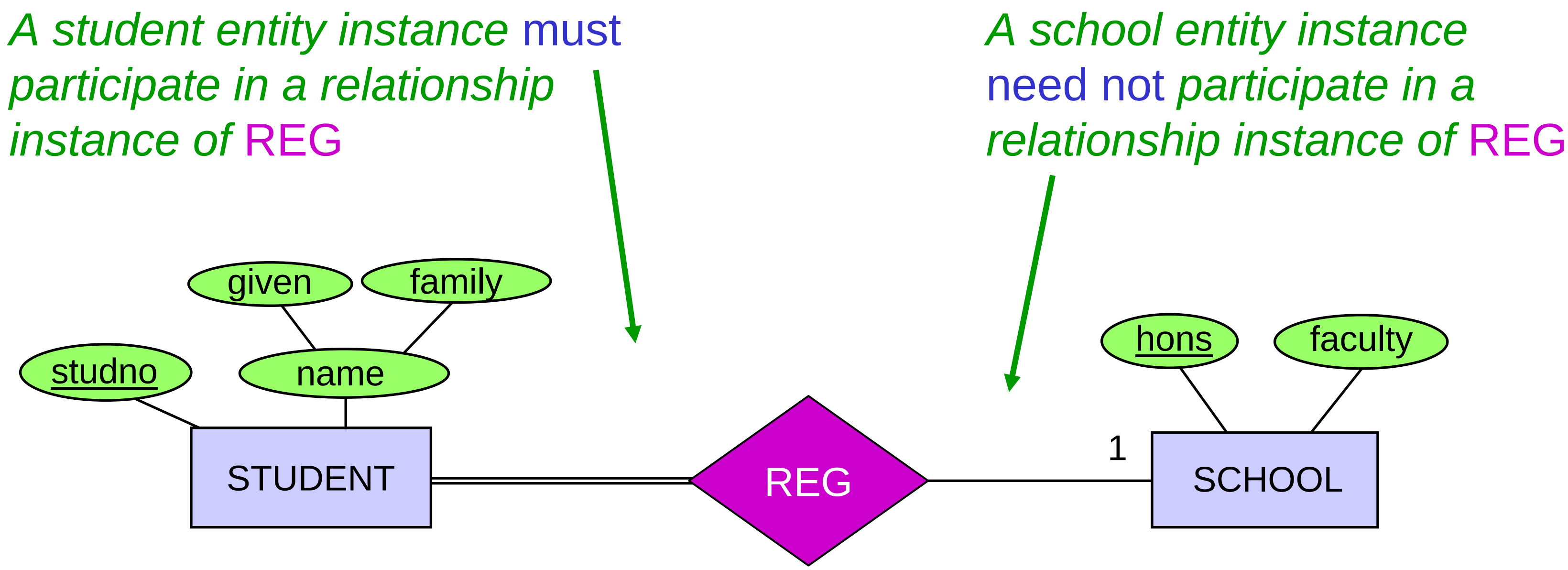
STAFF

name	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

TUTOR

studno	tutor	roomno	slot
s1	bush	2.26	12B
s2	kahn	IT206	12B
s3	goble	2.82	10A
s4	goble	2.82	11A
s5	zobel	2.34	13B
s6	kahn	IT206	12A

Optional Participation of the Determined Entity ('one end')



SCHOOL (<u>hons</u>, faculty)

STUDENT (<u>studno</u>, givenname, familyname, ???)

Optional Participation of the Determined Entity

STUDENT

studno	given	family	hons
s1	fred	jones	ca
s2	mary	brown	cis
s3	sue	smith	cs
s4	fred	bloggs	ca
s5	peter	jones	cs
s6	jill	peters	ca

“hons” can’t be NULL because it is mandatory for a student to be registered for a school

➔ “not null” constraint

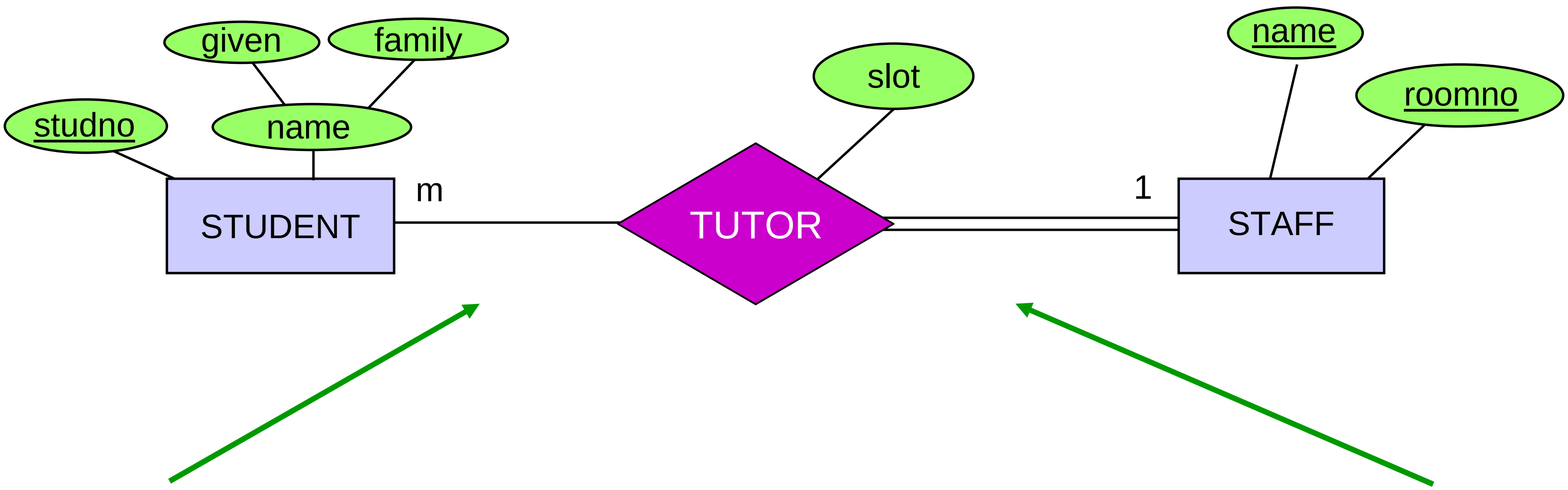
SCHOOL

hons	faculty
ac	accountancy
is	information systems
cs	computer science
ce	computer science
mi	medicine
ma	mathematics

No student is registered for “mi”, so “mi” doesn’t occur as a foreign key value

(but that’s no problem)

Optional Participation of the Determinant Entity ('many end')



A student entity instance need not *participate in a relationship instance of* TUTOR

A staff entity instance must *participate in a relationship instance of* TUTOR

Optional Participation of the Determinant Entity

1. STUDENT (studno, givenname, familyname, tutor, roomno, slot)

   STAFF(name, roomno)

   Integrity constraint:

   $$\pi_{name,roomno}\ \text{STAFF} \setminus \pi_{tutor,roomno}\ \text{STUDENT} = \emptyset$$

2. STUDENT(studno, givenname, familyname)

   STAFF(name, roomno)

   TUTOR(studno, tutor, roomno, slot)

Do we also need an integrity constraint?

Optional Participation of the Determinant Entity (cntd.)

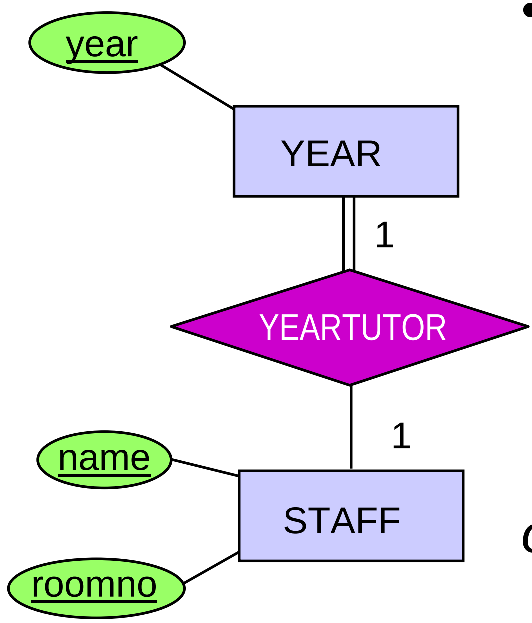
STUDENT

studno	given	family	tutor	roomno	slot
s1	fred	jones	bush	2.26	12B
s2	mary	brown	kahn	IT206	12B
s3	sue	smith	goble	2.82	10A
s4	fred	bloggs	goble	2.82	11A
s5	peter	jones	zobel	2.34	13B
s6	jill	peters	kahn	IT206	12A

STAFF

name	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

Mapping One:one Relationship Types to Relations



- Post the primary key of one of the entity types into the other entity type as a foreign key, including any relationship attributes with it

or

- Merge the entity types together

Which constraint holds in this case?

YEAR

year	yeartutor
1	zobel
2	bush
3	capon

STAFF

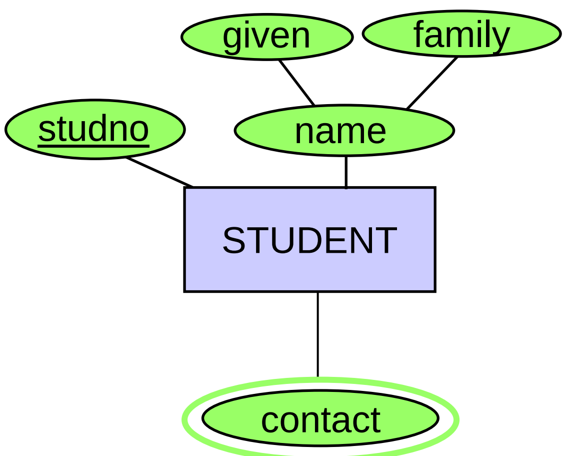
name	roomno	year
kahn	IT206	NULL
bush	2.26	2
goble	2.82	NULL
zobel	2.34	1
watson	IT212	NULL
woods	IT204	NULL
capon	A14	3
lindsey	2.10	NULL
barringer	2.125	NULL

Multi-Valued Attributes

For each multi-valued attribute of E_i, create a relation with the attributes

primary_key(E_i) U multi-valued attribute

The primary key comprises *all* attributes



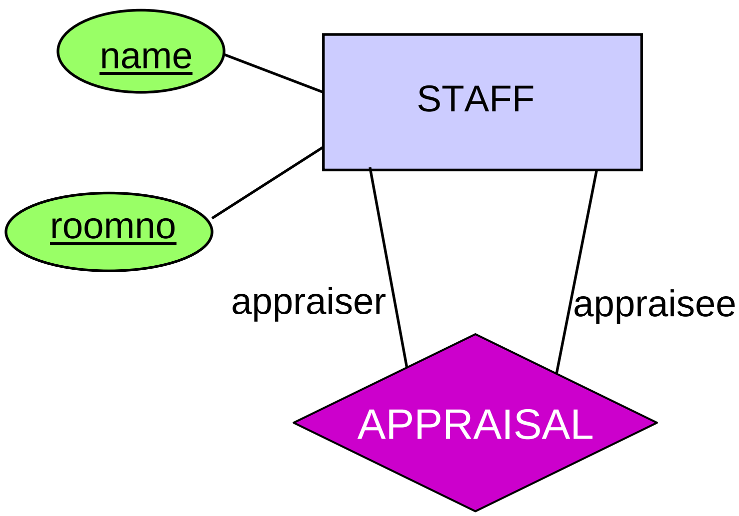
STUDENT

studno	given	family
s1	fred	jones
s2	mary	brown

STUDENT_CONTACT

studno	contact
s1	Mr. Jones
s1	Mrs Jones
s2	Bill Brown
s2	Mrs Jones
s2	Billy-Jo Woods

Mapping Roles and Recursive Relationships

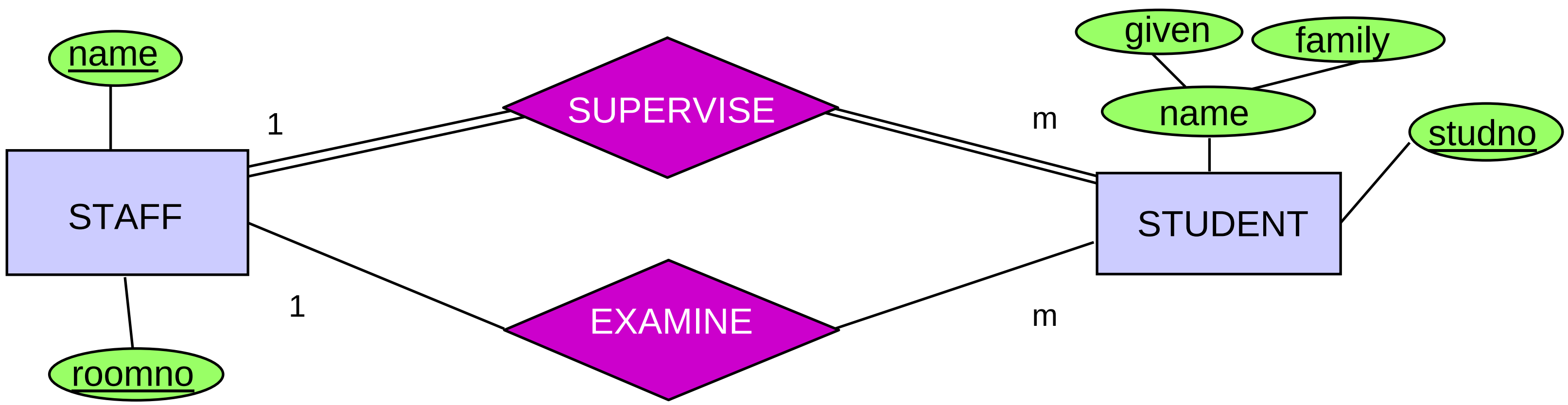


How can the entity STAFF *appear in both of its roles ?*

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

Multiple Relationships between Entity Types

1. Treat each relationship type separately
2. Represent distinct relationships by different foreign keys drawing on the same relation



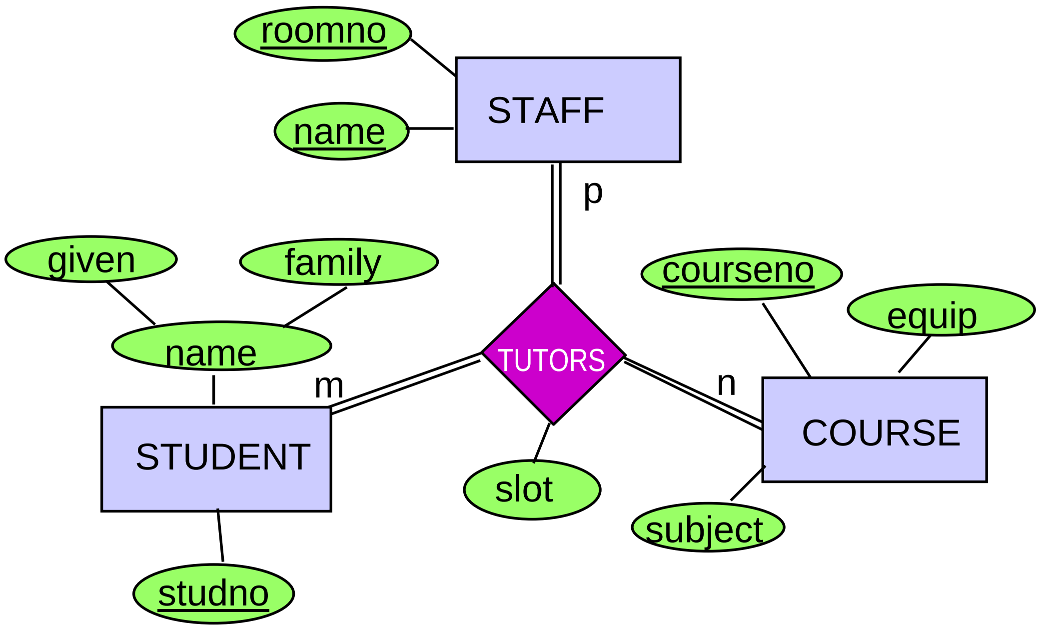
STAFF(<u>name</u>, <u>roomno</u>)

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

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

EXAMINER-SUPERVISOR(*???*)

Non-binary Relationship



COURSE(<u>courseno</u>, subject, equip)

STUDENT(<u>studno</u>, givenname, familyname)

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

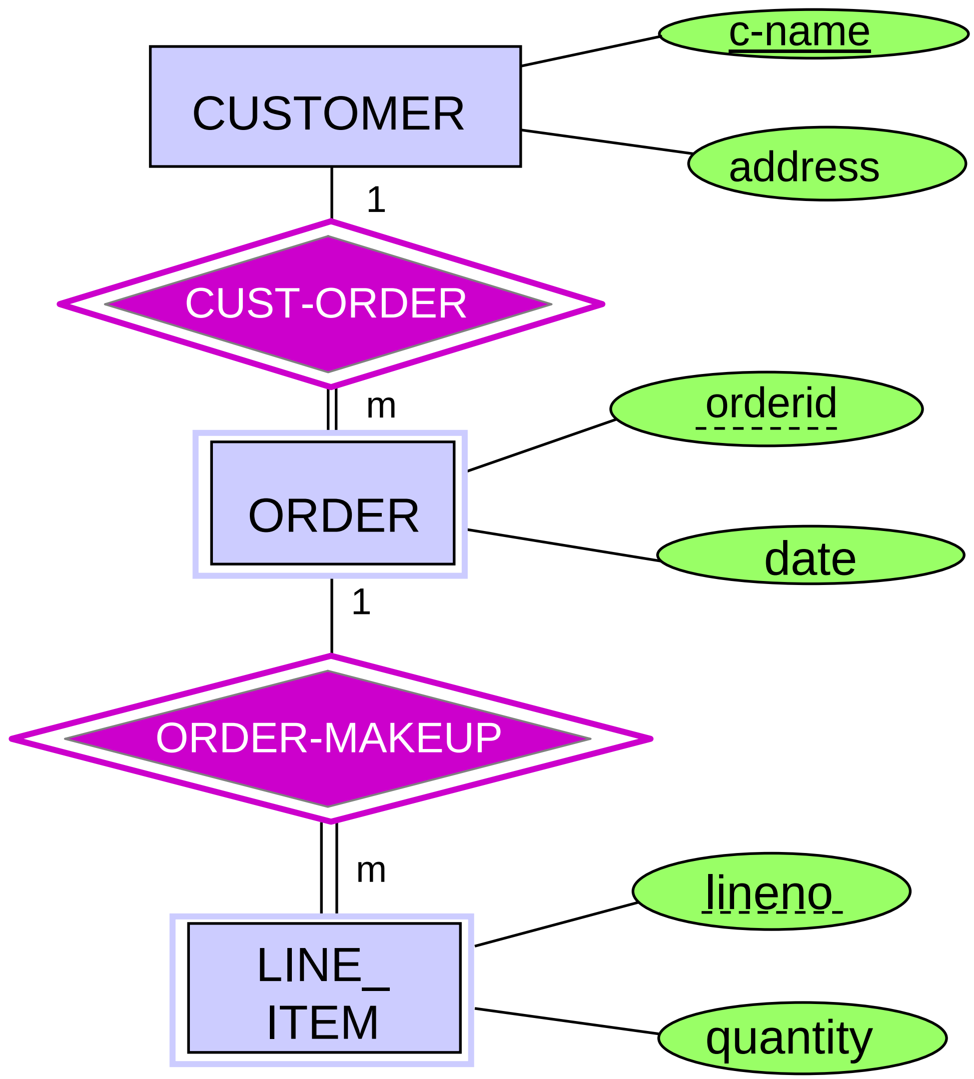
TUTORS(*???*)

Weak Entities

- Strong entity type
- Identifying entity for ORDER
- Identifying entity for LINE_ITEM

- Weak entity type
- Identifying entity for LINE_ITEM

- Weak entity type



Mapping Weak Entities to Relations

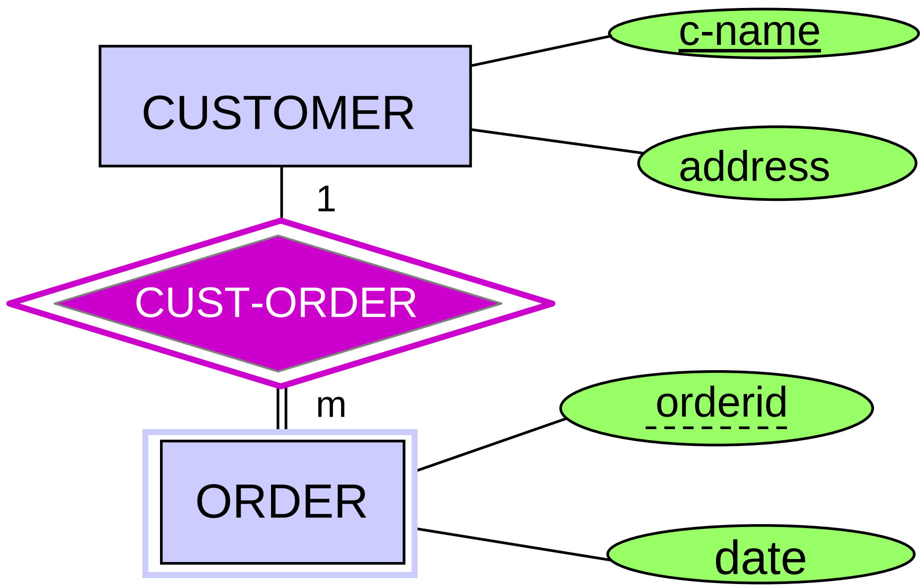
Create a relation with the attributes:

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

Primary key of identifying strong entity type

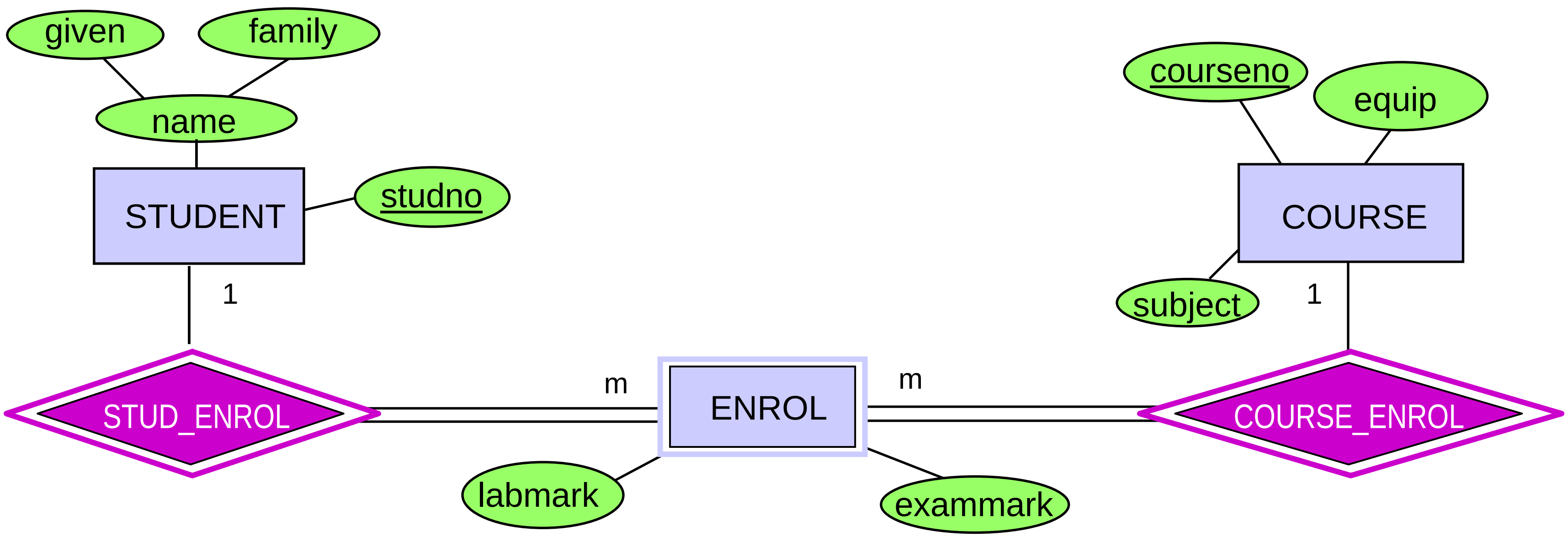
Discriminators of identifying weak entity types

Attributes of the weak entity type



Association Entity Types

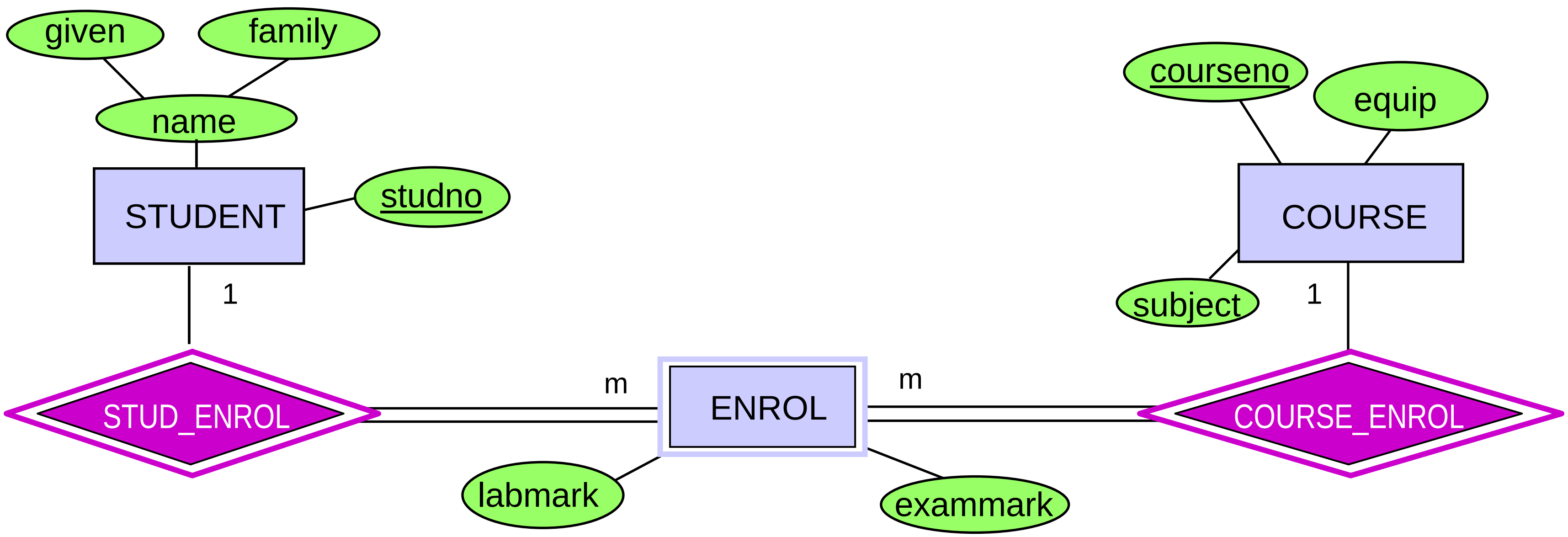
An entity type that represents a relationship type:



Association Entity Types

We have:

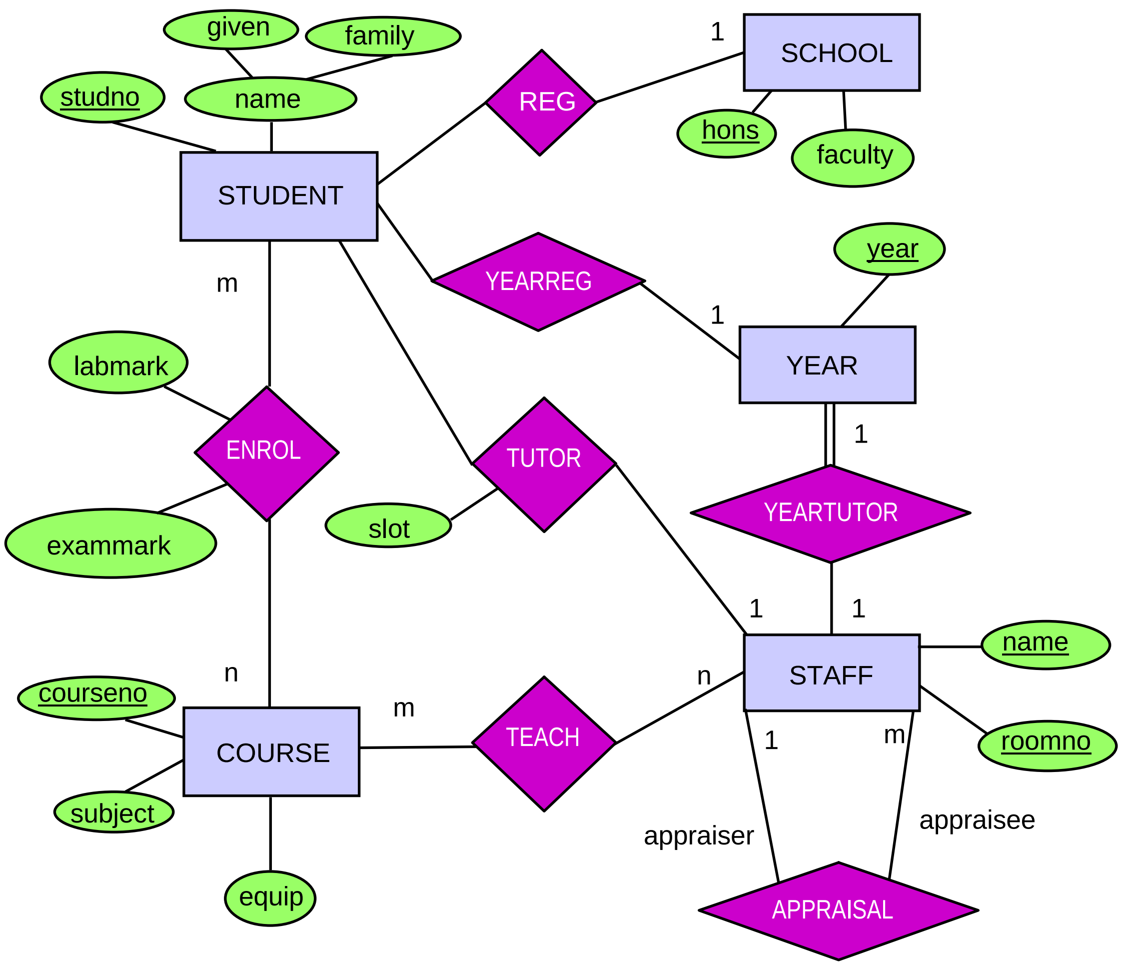
- COURSE(courseno, subject, equip)
- STUDENT(studno, givenname, familyname)



Then:

- ENROL(courseno, studno, labmark, exammark)

Translation of the University Diagram



STUDENT
(studno, givenname, familyname, hons, tutor, tutorroom, slot, year)

ENROL(studno, courseno, labmark,exammark)

COURSE(courseno, subject, equip)

STAFF(lecturer,roomno, appraiser, approom)

TEACH(courseno, lecturer,lecroom)

YEAR(year, yeartutor, yeartutorroom)

SCHOOL(hons, faculty)

Exercise: Supervision of PhD Students

A database needs to be developed that keeps track of PhD students:

- For each student store the name and matriculation number. Matriculation numbers are unique.
- Each student has exactly one address. An address consists of street, town and post code, and is uniquely identified by this information.
- For each lecturer store the name, staff ID and office number. Staff ID's are unique.
- Each student has exactly one supervisor. A staff member may supervise a number of students.
- The date when supervision began also needs to be stored.

Exercise: Supervision of PhD Students

- For each research topic store the title and a short description. Titles are unique.
- Each student can be supervised in only one research topic, though topics that are currently not assigned also need to be stored in the database.

Tasks:

a) Design an entity relationship diagram that covers the requirements above. Do not forget to include cardinality and participation constraints.

b) Based on the ER-diagram from above, develop a relational database schema. List tables with their attributes. Identify keys and foreign keys.

Translating of Hierarchies: Options

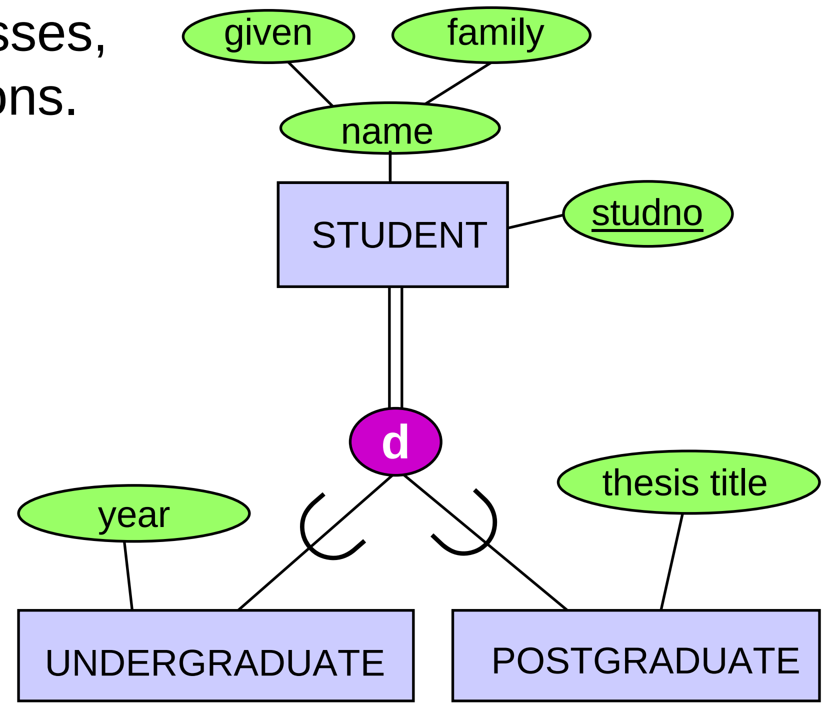
To store information about these classes,
We have to define appropriate relations.

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

C. Create only one relation, for the superclass

Translation into Relations: Option A

1. Create a relation for the superclass
2. For each subclass, create a relation over the set of attributes

primary_key(superclass) U attributes of subclass

The key for each subclass relation is: primary_key(superclass)

Inclusion constraint (foreign keys):

$$\pi_{key}(\text{superclass}) \supseteq \pi_{key}(\text{subclass}_i)$$

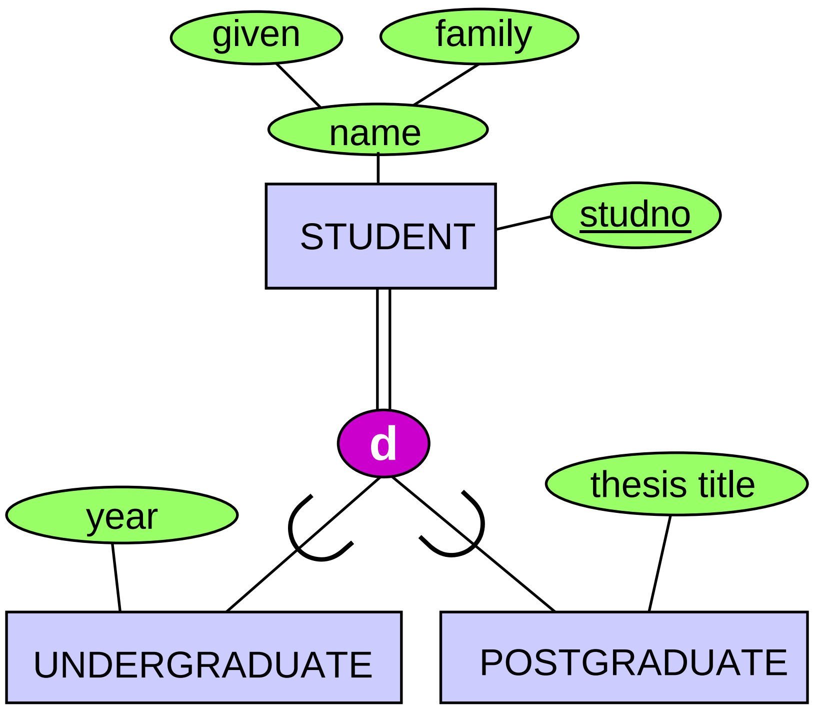
Covering constraint *(n = number of subclasses)*:

$$\bigcup_{i=1}^{n} \pi_{key}(\text{subclass}_i) \supseteq \pi_{key}(\text{superclass})$$

Disjointness constraint:

$$\pi_{key}(\text{subclas s}_i) \cap \pi_{key}(\text{subclass}_j) = \varnothing$$

for $i \neq j$



Translation into Relations: Option B

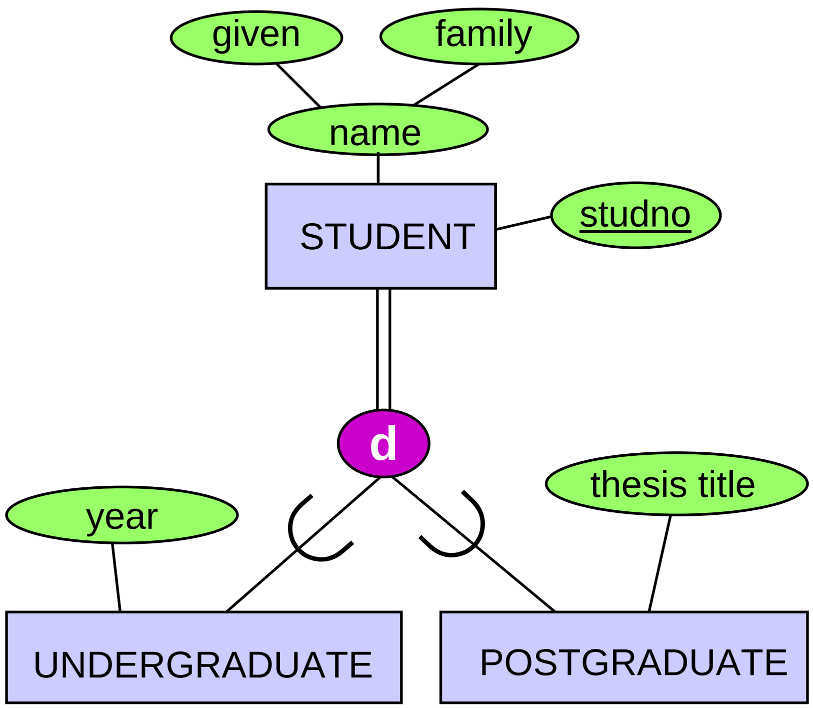
Create only relations for subclasses. Each subclass becomes a relation over the set of attributes:

attributes of superclass U attributes of subclass

The key for each subclass relation is: primary_key(superclass)

- Works only if coverage is total and disjoint
- *Partial coverage*: entities that are not in a subclass are lost
- *Overlapping classes*: redundancy
- Recovery of the superclass: OUTER UNION on the subclass relations

In SQL: UNION JOIN, Outer join on the common attributes



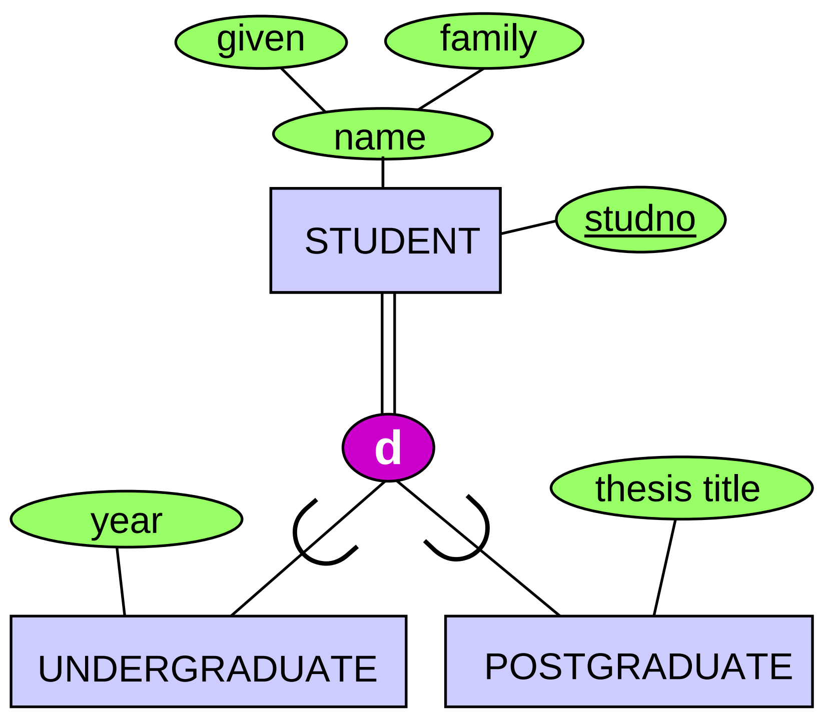
Translation into Relations: Option C

Create a single relation over the set of attributes

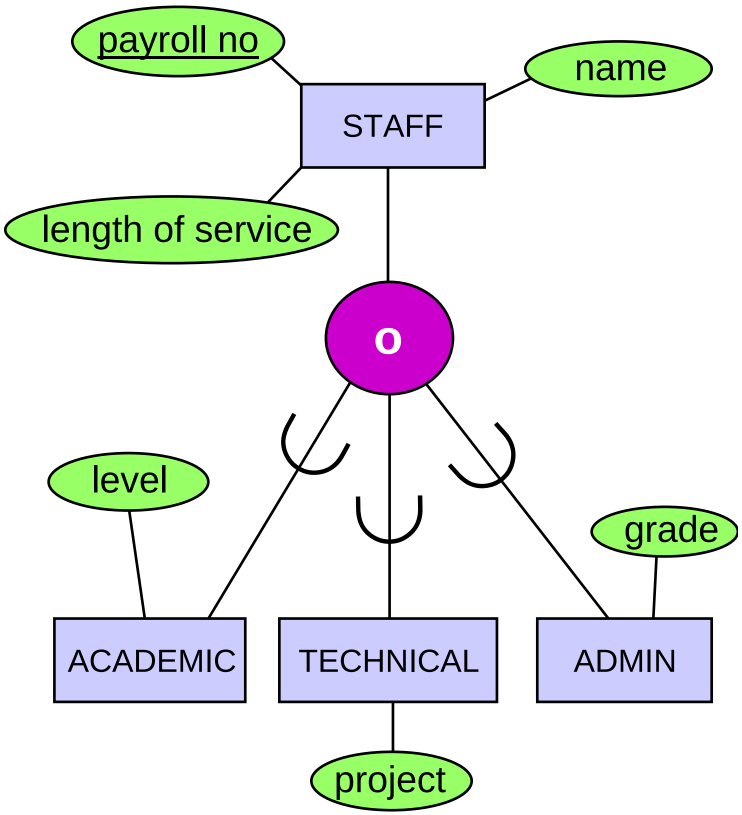
$$\text{attributes of superclass} \;\cup\; \bigcup_{i=1}^{n} \text{attributes of subclass}_i \;\cup\; \{\, \textit{class} \,\}$$

The key is: primary_key(superclass)

- Drawback: many ‘not-applicable’ nulls
- Benefit: No need for joins
- *Disjoint coverage*: one attribute *class* which indicates the subclass the tuple represents
- *Overlapping coverage: class* has to represent a set of classes
- *Partial coverage*: *class* is null
  ∴ entity is from superclass



Applying the Three Translations (Overlapping Coverage)



A. STAFF(payrollno, name, lengthOfService)
ACADEMIC(payrollno, level)
TECHNICAL(payrollno, project)
ADMIN(payrollno, grade)

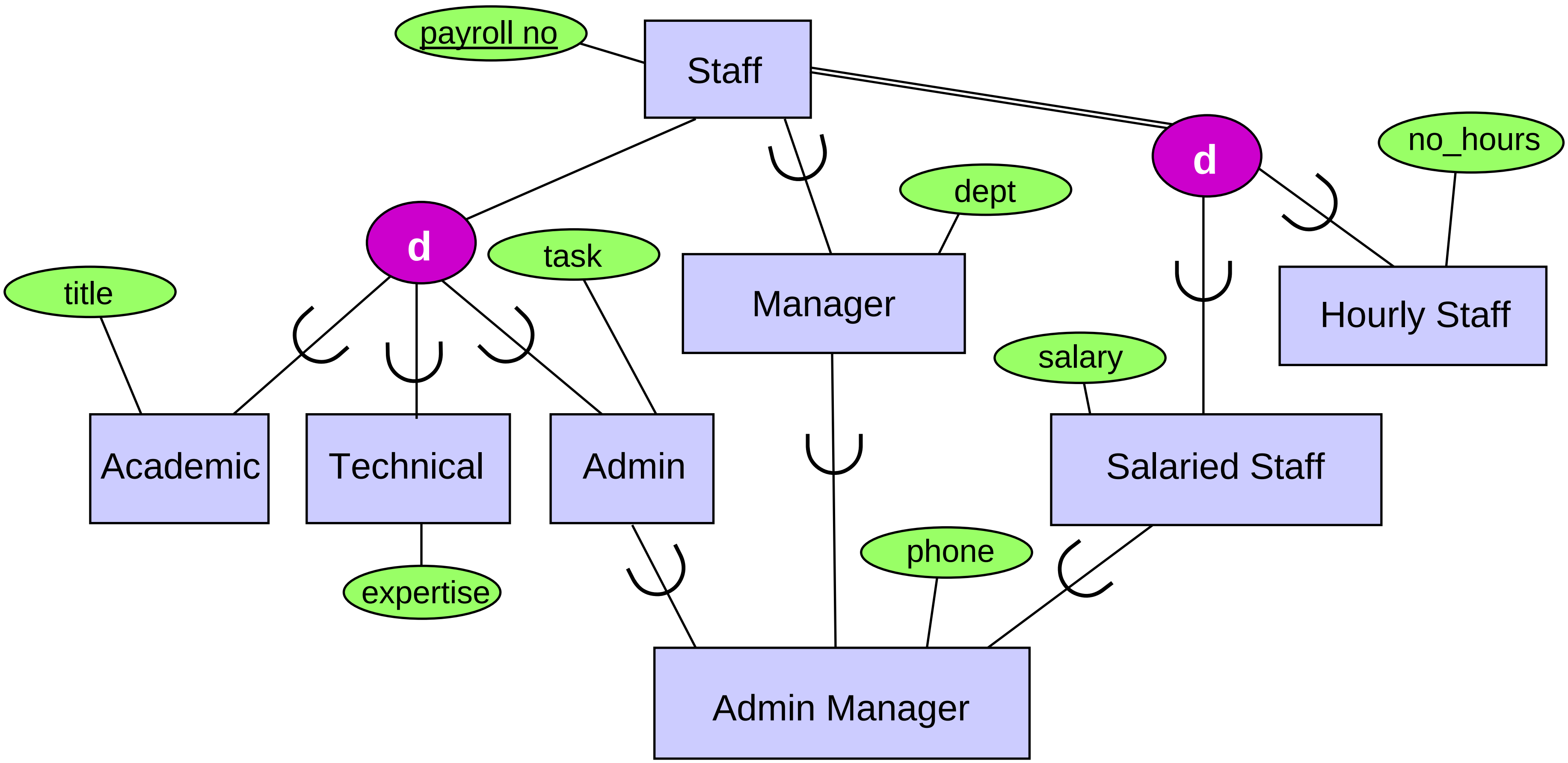
B. ACADEMIC(payrollno, name, lengthOfService, level)
TECHNICAL(payrollno, name, lengthOfService, project)
ADMIN(payrollno, name, lengthOfService, grade)

C. STAFF(payrollno, name, lengthOfService, level, project, grade, class1, class2, class3)
or

STAFF(payrollno, name, lengthOfService, level, project, grade, class)

class = powerset of classes

Specialisation Lattice with Shared Subclass



Exercise: For each of the approaches A, B, C, decide

- Which tables need to be created?
- Which are the attributes? And which are their possible values?

References

In preparing these slides I have used several sources.
The main ones are the following:

Books:

- A First Course in Database Systems, by J. Ullman and J. Widom
- Fundamentals of Database Systems, by R. Elmasri and S. Navathe

Slides from Database courses held by the following people:

- Enrico Franconi (Free University of Bozen-Bolzano)
- Carol Goble and Ian Horrocks (University of Manchester)