Relational Model

Prof. Sharad Mehrotra

Information and Computer Science Department

University of California at Irvine

Chapter 3 and 6 from SKS

Chapter 3 in UW

Outline

- Relational model
  - basic modeling concepts for schema specification
- Mapping ER diagrams to Relational Model
- Relational Languages
  - relational algebra (algebraic)
    - basic operators, expressions in relational algebra
  - relational calculus (logic based) /*will not be covered in class */

Relational Model -- Quick Example

Ssno	Name	Street	city
NULL	Frank	12 Main	Champaign
1200331	Cathy	3 Neil	Urbana
2000312	Bill	45 W. Oak	Urbana

Account Number	Custid	balance
34	1000201	10,799
45	1200331	1,345
63	2000312	100,345

- A relational schema = tables and constraints.
- Tables: customer, account
- Constraints:

Key constraints:

•ssno is the key for customer table

•both accountno and custid are keys for account table

Referential Integrity constraints: **(foreign keys)**

The custid attribute in account table takes values from ssno in customer table

Null Constraint:

customer name cannot take null values

Relational Model

- Database schema consists of
  - a set of relation schema
  - a set of constraints over the relation schema
- Relational Schema = name(attributes). Graphically drawn as table.
- Example: employee(ssno, name, salary)
- Recall:
  - relation is a <u>subset</u> of cartesian product of sets
  - relation is a set of n-tuples where n = degree of the relation

Attributes

- With each attribute a domain is specified
- In relational model attributes are atomic:
  - the values are not divisible. That is, we cannot refer to or directly see a subpart of the value.
- an attribute can take a special *null* value
- Null value represents either attributes whose value is not known, or do not exist

Example of a Relation

Diagnosis: an example relation/table

Patient	Disease
Jim	Schizophrenic
Jane	Obsessive-Comp
Jerry	Manic
Joe	null

null in this case may mean that diagnosis is not complete and disease has not been identified. Notice possibility of confusion that null means that the patient has no disease! This is one of the reasons why using nulls is not a great idea! We will see other reasons as well later

Constraints

- **What are they?**
  - **represent the semantics of the domain being modeled.**
  - **restrict the set of possible database states**
- **Why do we want to specify the constraints?**
  - **Useful information to application programmers. They can write programs to prevent constraints violation**
    - **constraint -- acct balance should not fall below 0 dollars**
    - **programmer writing code for debit application should check at the end if the account gets overdrawn!**
  - **DBMS might *enforce* specified constraints directly making task of application writer easier**
    - **If DBMS guarantees that account does not get overdrawn, then debit application programmer need not worry about checking for overdrawn account condition.**

Constraints

- Why specify constraints?
  - knowledge of some type of constraints enables us to identify redundancy in schemas and hence specification of constraints helps in database design (we will see this later)
  - Knowledge of some type of constraints can also help the DBMS in query processing

Specifying Constraints in Data Models

- ER model
  - domain and key constraints over entities
  - participation and cardinality constraints over relationships
- Relational Model
  - domain constraints, entity identity, key constraint, functional dependencies -- generalization of key constraints, referential integrity, inclusion dependencies -- generalization of referential integrity.

Domain Constraint

- In the schema, every attribute is declared to have a type --- integer, float, date, boolean, string, etc.
- An insertion request can violate the domain constraint.
- DBMS can check if insertion violates domain constraint and reject the insertion.

Key Constraint

- **Each relation has a *primary key.***
- *Superkey:*
  - **set of attributes such that if two tuples agree on those attributes, then they agree on all the attributes of the relation**
  - **Note: the set of all the attributes of a relation is always a superkey.**
- *Candidate key:*
  - **superkey no subset of which i s a superkey**
- *Primary key:*
  - **one of the candidate keys**

Disallowing Null Values

- **Some fields of a relation are too important to contain null values.**
- **Example:**
  - **in sales(customer, salesman, date, amount, saleID) we may not want ‘customer’ to contain a null value.**

Entity Integrity Constraint

- **A *primary key* must not contain a null value.**
  - **Else it may not be possible to identify some tuples.**
  - **For Example, if more than one tuple has a null value in its primary key, we may not be able to distinguish them .**

Foreign Key and Referential Integrity Constraint

- Consider following 2 relation schemas:
  - R1(A1, A2, …An)   and R2(B1, B2, … Bm)
- Let PK be subset of {A1, …,An} be primary key of R1
- A set of attributes FK is a *foreign key* of R2 if:
  - attributes in FK have same domain as the attributes in PK
  - For all tuples t2 in R2, there exists a tuple t1in R1 such that t2[FK] = t1[PK].
- A referential integrity constraint from attributes FK of R2 to R1 means that FK is a foreign that refers to the primary key of R1.

Example of Referential Integrity

- **student-grades**

student	C#	Semester	grade
Susan	CS101	1-91	A
Jane	CS101	1-91	B

- **LegalGrades**

Grade
A
B
C
D
F
Audit
Ex

we will have a referential integrity constraint saying that every value of student-grades.grade must also be a value of LegalGrades.grade,

Inclusion Dependencies

- Generalization of referential integrity constraint.
- Inclusion dependency R1[A1,...,An] $\subseteq$ R2 [B1,...,Bn] means that the values in the first relation R1 refer to the values in the second relation
- Formally, R1[A1,...,An] $\subseteq$ R2 [B1,...,Bn] iff the following holds:
  - for all t1 in R1, there exists a t2 in R2 such that t1[A1, ..., An] = t2[B1, ..., Bn]
- Notice that referential integrity constraint is an inclusion dependency in which {B1, .. Bn} is the primary key of R2.

Example

- student-grade[Grade] $\subseteq$ LegalGrade[Grade]
- CourseOffering[C#] $\subseteq$ Courses[C#]
- Takes[S#] $\subseteq$ Students[S#]
- CourseOffering[Professor] $\subseteq$ UCEmployee[E#]

Data Modification and Integrity Constraints

- Modification, insertion and deletion requests can lead to violations of integrity constraints.
  - Key constraint, entity identity, null value constraint, referential integrity, inclusion dependencies, functional dependencies, multivalued dependencies.
- must check for violation at end of each operation and suitably allow or disallow operation.
- Impact of data modification on inclusion dependencies can be sometimes tricky!

Example

- Relations
  - CourseOfferings(C#, semester, instructor)
  - Takes(S#, C#, semester, grade)
- Referential Integrity Constraint:
  - Takes(C#,semester) $\subseteq$ CourseOffering(C#,semester)
- Consider canceling a course.
  - Delete from courseOfferings where c# = "CS101" AND Semester = "2-91";
- What should happen to tuples in Takes that refer to CS101 and semester 2-91??

Example (cont)

Takes	S#	C#	Semester	Grade
	1001	CS101	2-91	^
	1002	CS101	2-91	^
	1003	CS101	1-91	A

- **Possible Solutions:**

1) reject update (*abort*) - or -

2) delete tuples from 'Takes' that refer to 'CS101, 2-91' (*cascade*)

Functional Dependencies

FDs is a generalization of concept of keys.

Given a relation R with attributes

$A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_l$,

we say that

$A_1,\ldots,A_n$ <u>functionally determine</u> $B_1,\ldots,B_m$

$(A_1,\ldots,A_n \longrightarrow B_1,\ldots,B_m)$

if whenever two tuples agree on their values for

$A_1,\ldots,A_n$, they agree on $B_1,\ldots,B_m$

The key of a relation functionally determines all the attributes of the relation.

(by definition of a key)

Example

- Takes(C#, S#, semester, grade).
- Key = (C#,S#,semester)

C#	S#	Semester	grade
CS101	13146	1-91	A
CS101	13146	1-91	B

illegal since it violates FD that C#,S#,Semester functionally determine grade

Logical Implication of Functional Dependencies

- Consider R(A,B,C)
- Let the following functional dependencies hold:
  - $A \longrightarrow B$ (f1)
  - $B \longrightarrow C$ (f2)
- We can show that f1 and f2 together logically imply that the following functional dependency also holds:
  - $A \longrightarrow C$ (f3)

Proof

- **say f3 does not hold.**
- **then there exists tuples t1, t2 in R such that t1[A] = t2[A] and t1[C] is not equal to t2[C]**
- **Since f1 holds and since t1[A] = t2[A], it must be the case that t1[B] = t2[B]**
- **Hence since t1[B] = t2[B] and f2 holds, it must be the case that t1[C] = t2[C]**
- **This is a contradition!**
- **Hence, f3 must also hold!**

Closure of Functional Dependency Set

Definition: Let R be a relation scheme, and F be the set of functional dependencies defined over R. F+ denotes the set of all functional dependencies that hold over R. That is,

$$F^+ = \{ X \longrightarrow Y \mid F \text{ logically implies } X \longrightarrow Y \}$$

Example:

Let F = {$A \longrightarrow B$, $B \longrightarrow C$}

then

$A \longrightarrow C$ is in F+

F+ is called the closure of F

Inferring Functional Dependencies

Given a set of fds F over a relation scheme R, how to compute F+ ?

1. **Reflexivity:**

   If Y is a subset of X, then $X \longrightarrow Y$

   Examples: $AB \longrightarrow A$, $ABC \longrightarrow AB$, etc.

2. **Augmentation:**

   If $X \longrightarrow Y$, then $XZ \longrightarrow YZ$

   Examples: If $A \longrightarrow B$, then

   $AC \longrightarrow BC$

3: **Transitivity:**

   If $X \longrightarrow Y$, and $Y \longrightarrow Z$,

   then $X \longrightarrow Z$.

These 3 rules are called ARMSTRONGS AXIOMS!!

Using AA to infer dependencies

- Union Rule:
  - if $X \longrightarrow Y$, $X \longrightarrow Z$, then $X \longrightarrow YZ$
- Proof:
  - Since $X \longrightarrow Y$, using augmentation, $X \longrightarrow XY$ (1)
  - Since $X \longrightarrow Z$, using augmentation, $XY \longrightarrow XZ$ (2)
  - Using (1) and (2) and transitivity we get

    $X \longrightarrow YZ$ Q.E.D.
- Pseudo-Transitivity Rule:
  - If $X \longrightarrow Y$, $WY \longrightarrow Z$, then $WX \longrightarrow Z$
- Proof:
  - Since $X \longrightarrow Y$, using augment $XW \longrightarrow YW$ (1)
  - Given $WY \longrightarrow Z$, and (1) using transitivity
  - $WX \longrightarrow Z$ Q.E.D.

Armstrongs Axioms

Armstrongs Axioms are sound:

If $X \longrightarrow Y$ is deduced using AA from F, then $X \longrightarrow Y$ holds over any relation r in which fd's in F hold.

Armstrongs Axioms are complete:

If a functional dependency $X \longrightarrow Y$ is in the closure of F (that is, in F+), then it can be deduced using Armstrongs Axioms over F.

Since AAs are sound and complete, they provide a mechanism to us to compute F+ from F.

Even though we defined F+ as the set of all fds that are logical implication of F, since AA are sound and complete, we can redefine F+ as the fds that follow from F using AA.

Superkeys and FDs

- Using FDs we can formally define the notion of keys
- Let R(A1, A2, ...,An) be a relation
- Let X be a subset of {A1, A2, ...An}
- X is a *superkey* of R if and only if the functional dependency X $\longrightarrow$ A1,A2, ...,An is in F+
- Naïve Algorithm to test for superkey
  - compute F+ using AA
  - If X -----> A1,A2,...,An is in F+
    - X is a superkey

Cardinality of Closure of F

Let F = {A $\longrightarrow$ B1, A $\longrightarrow$ B2, ..., A $\longrightarrow$ Bn} (**cardinality of F = n)**

then

{A $\longrightarrow$ Y | Y is a subset of {B1, B2, ..., Bn}} is a subset of F+

(cardinality of F+ is more than 2^n).

So computing F+ may take exponential time!

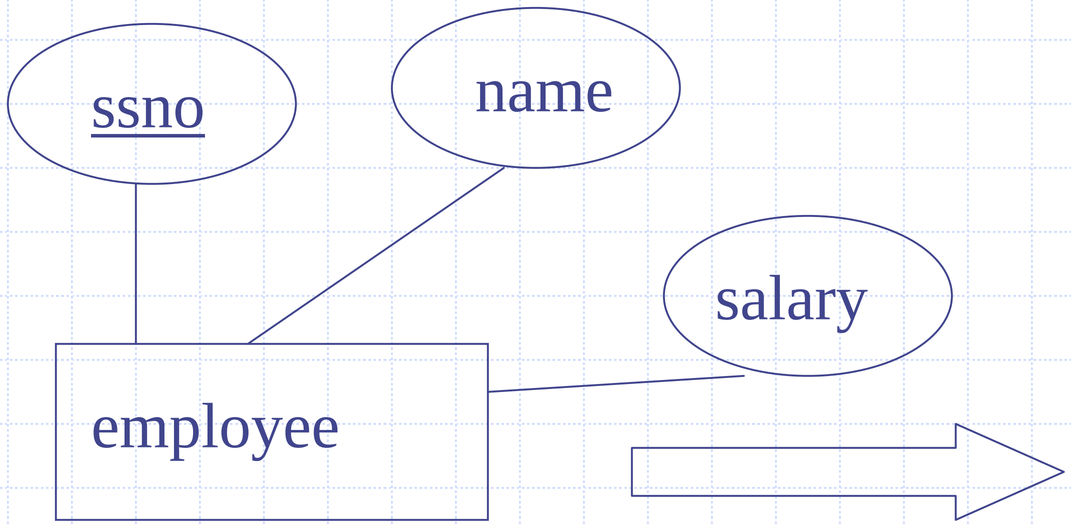
Fortunately, membership in F+ can be tested without computing F+. (we will study the algorithm later)

General Integrity Constraints

- **Commercial systems allow users to specify many other types of constraints besides the ones discussed in class. We will study them when we study SQL. :**
- **Example:**
  - **Relations:**
    - **Takes(C#, S#, semester, grade)**
    - **courses(C#, UnitsOfCredit, ...)**
  - **Integrity Constraints:**
    - **Every student must enroll for at least three credits every semester.**

ER to Relational Mapping

Strong Entity



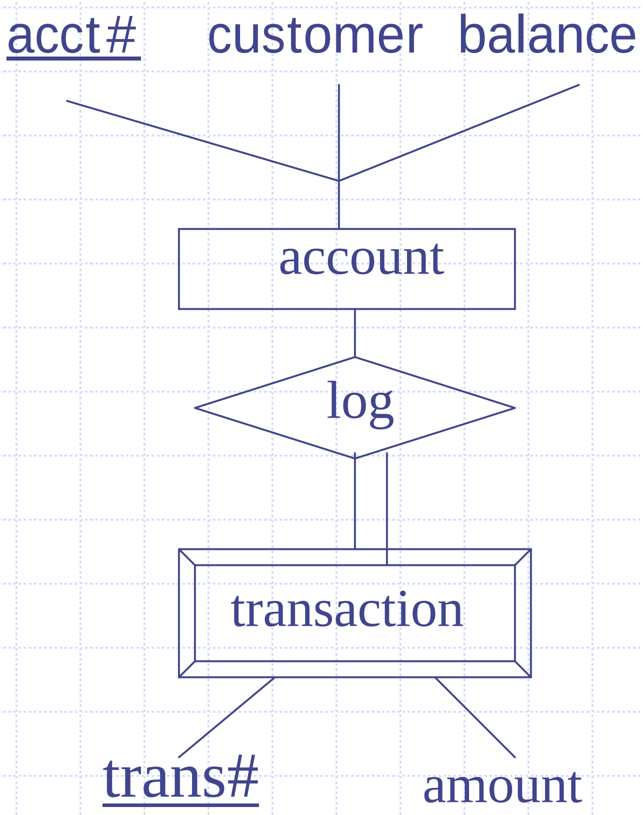
Relation:

Employee(<u>ssno</u> name salary)

Key : ssno

ER to Relational Mapping

Weak Entity



Relation:

transaction(acct#,trans#, amount)

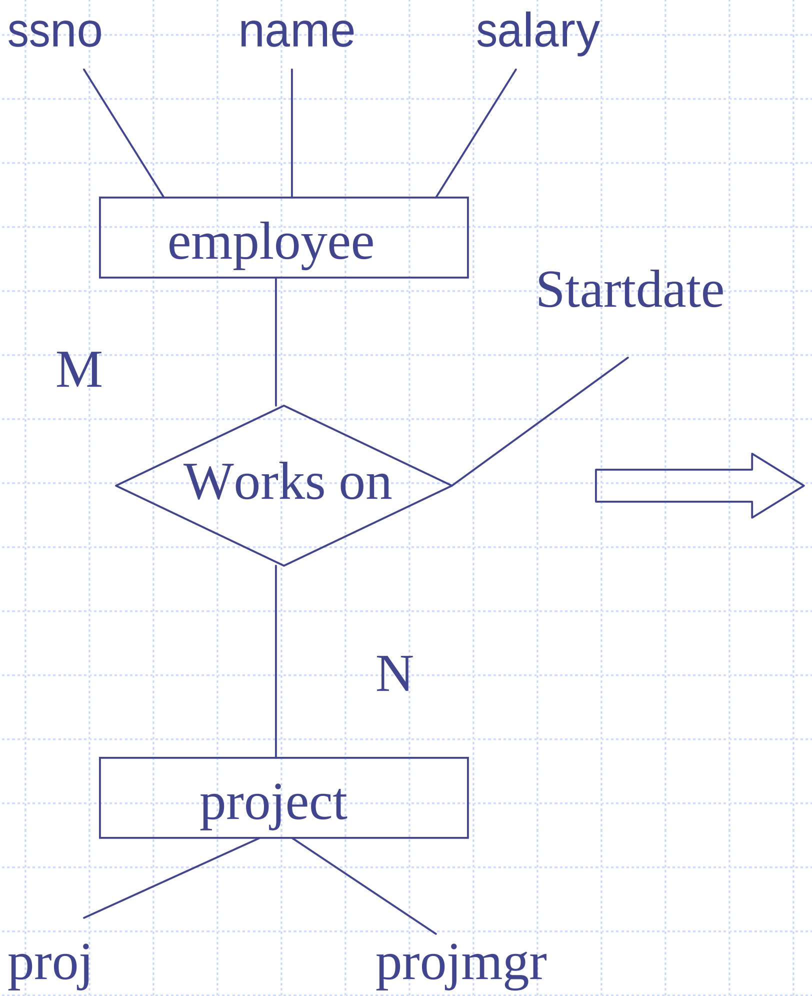
Key:

acct# trans#

IND:

transactional[acct#] $\subseteq$ account[acct#]

ER to Relational Mapping



Relation:

works_on(ssno,proj#,startdate)

Key:

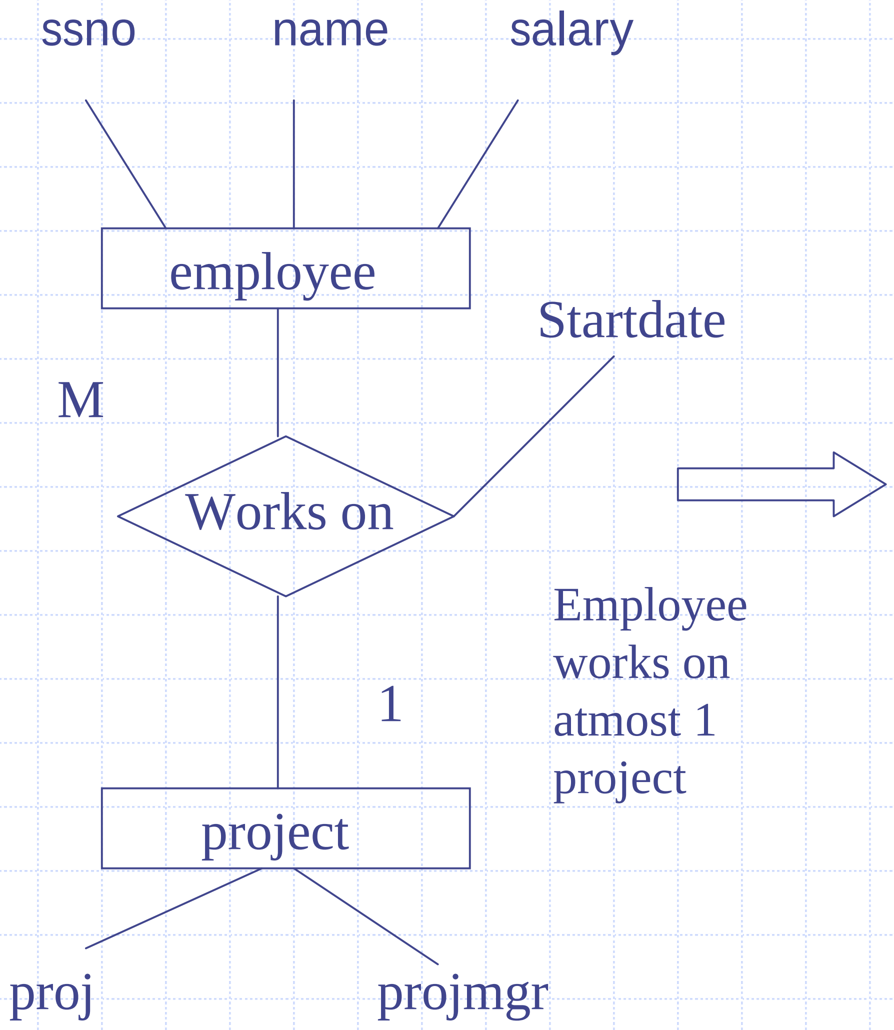
ssno,proj#

Ind:

workson[proj#] $\subseteq$ project[proj#]

workson[ssno] $\subseteq$ employee[ssno]

ER to Relational Mapping



Employee works on atmost 1 project

Relation:

works_on(ssno,proj#,startdate)

Key:

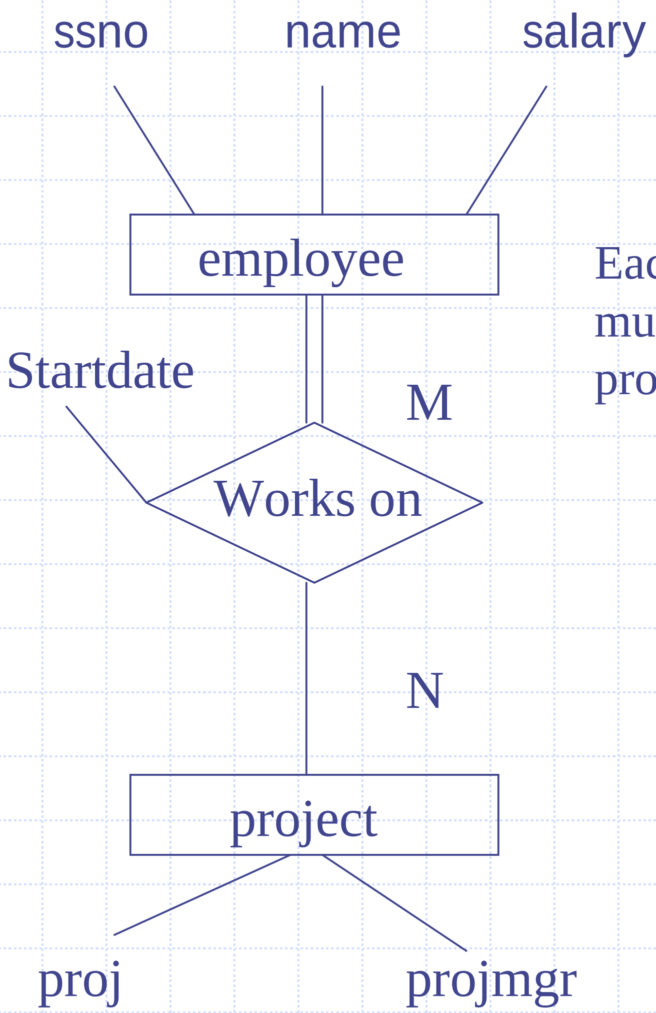
ssno

Ind:

workson[proj#] $\subseteq$ project[proj#]

workson[ssno] $\subseteq$ employee[ssno]

ER to Relational Mapping



Each employ must work on a project

Relation:

works_on(ssno,proj#,startdate)

Key:

ssno,proj#

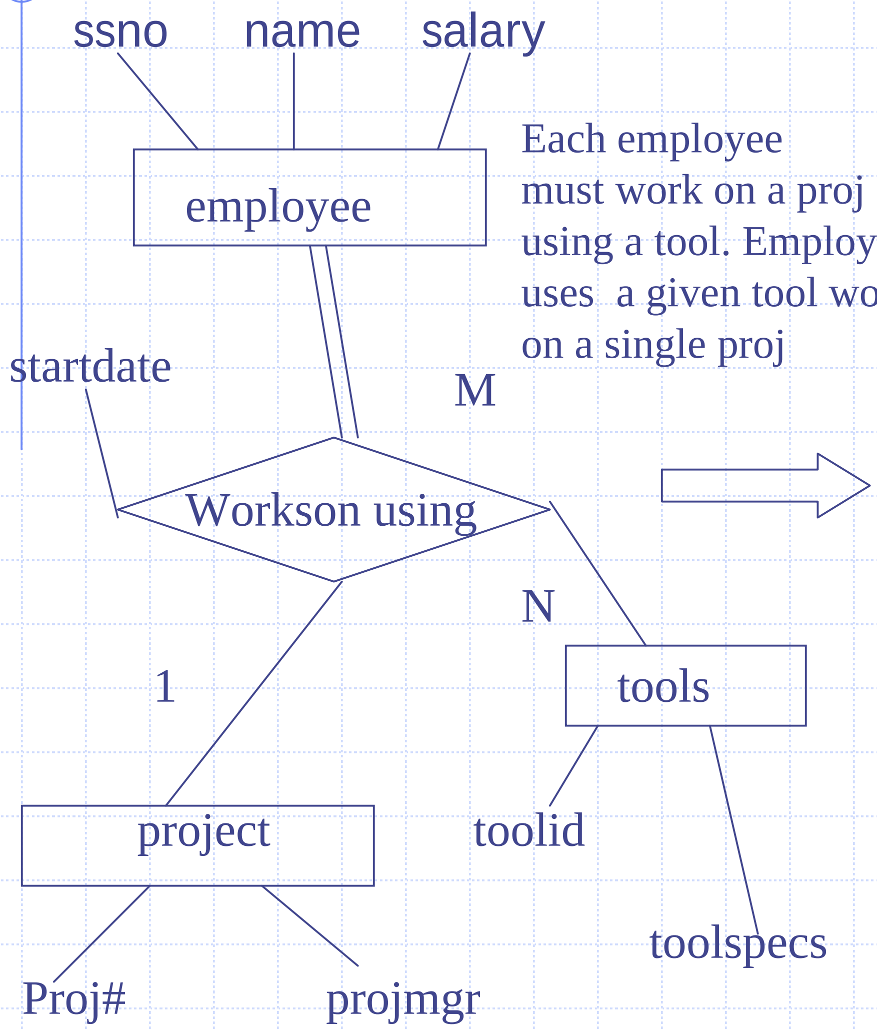
Ind:

$\text{workson[proj\#]} \subseteq \text{project[proj\#]}$

$\text{workson[ssno]} \subseteq \text{employee[ssno]}$

$\text{employee[ssno]} \subseteq \text{workson[ssno]}$

ER to Relational Mapping



Each employee must work on a proj using a tool. Employee uses a given tool works on a single proj

Relation :

worksonusing(ssno,proj#,toolid, startdate)

Key :

ssno,toolid

IND :

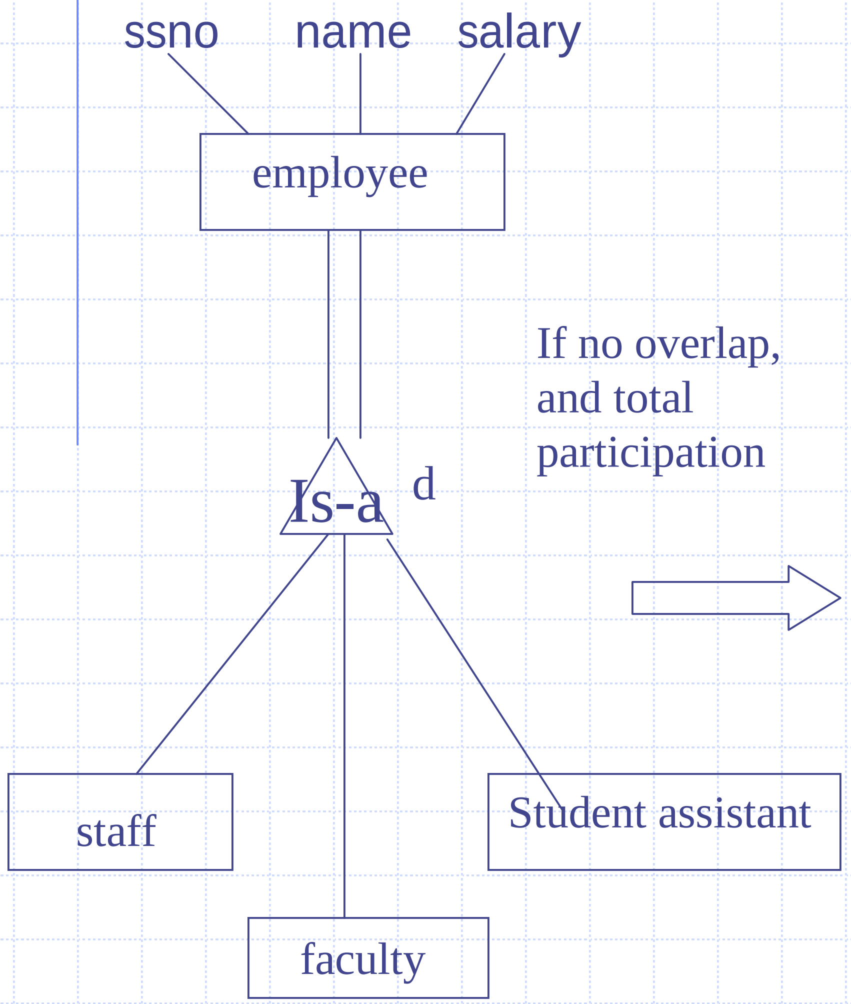
worksonusing[proj#] $\subseteq$ project[proj#]

worksonusing[ssno] $\subseteq$ employee[ssno]

worksonusing[toolid] $\subseteq$ tools[toolid]

employee[ssno] $\subseteq$ worksonusing[ssno]

ER to Relational Mapping



Relation:

staff(ssno, name, salary, position)

faculty(ssno, name, salary, rank)

student_assistant(ssno, name, salary, percentage_time)

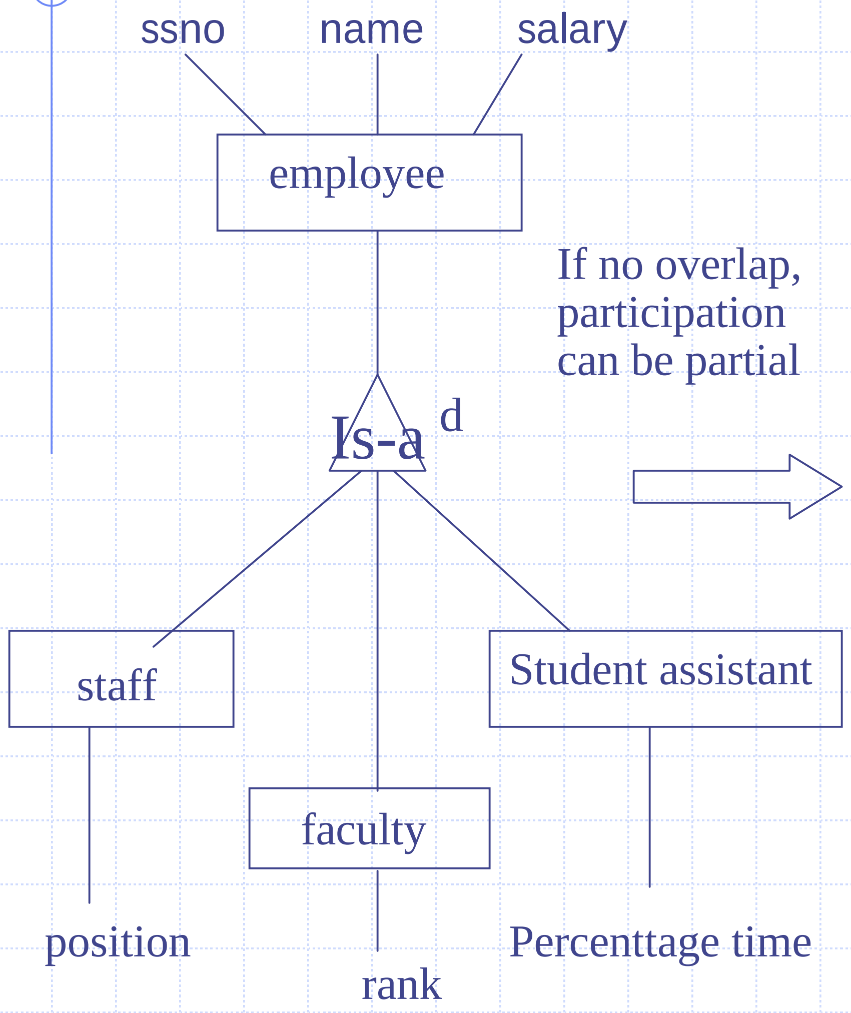
Key: ssno for all the relations

cannot use if partial:cannot represent employees who are neither staff, nor faculty, not student assistants!

Cannot use if overlap:if staff could also be a student assistant, then redundancy

requires a union to construct list of all employees

ER to Relational Mapping



Relation:
employee(ssno, name, salary, jobtype, position, rank, percentage-time)

Key : ssno

job type can be used to specify whether an employee is a staff, a faculty, or a student assistant

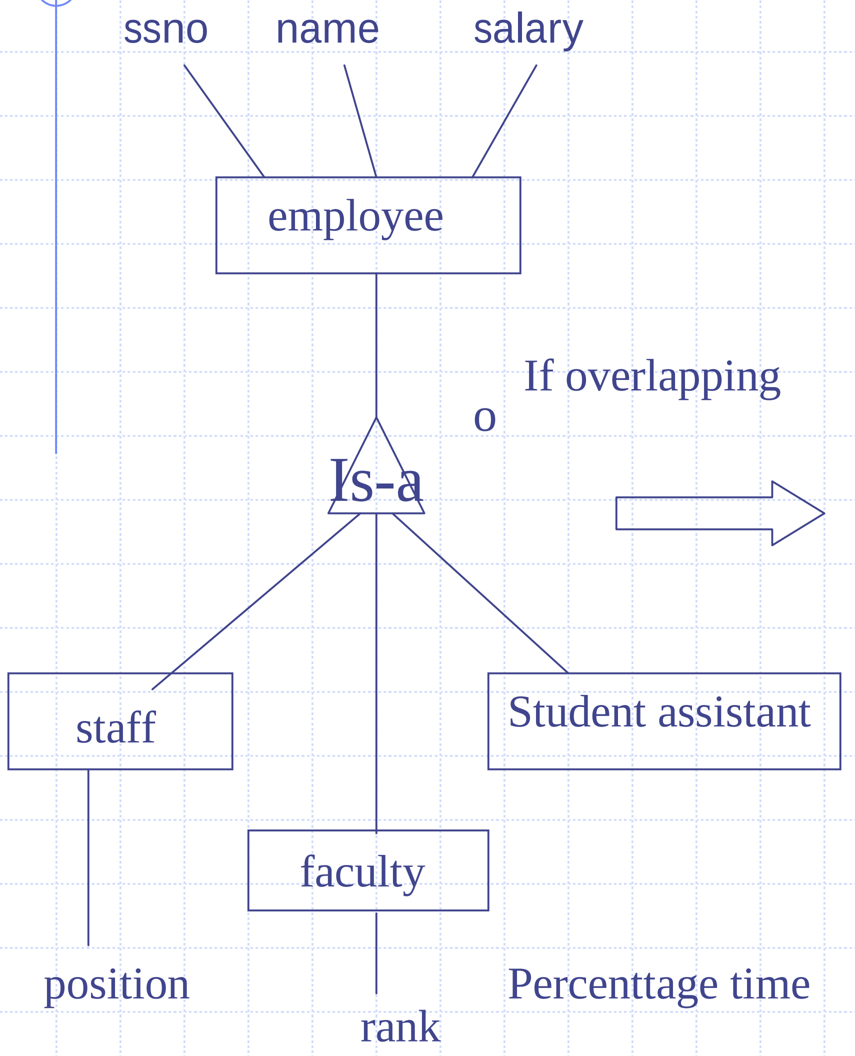
a lot of null values will be used.

If an employee does not belong to any subclass, use null value for job type
cannot be used if overlap

total participation can be represented by preventing null in jobtype

does not require union to construct the list of employees

ER to Relational Mapping



Relation:

employee (ssno, name, salary)

staff(ssno, position)

faculty(ssno, rank)

student_assistant(ssno, percentage_time)

Key : ssno for all the relations

IND

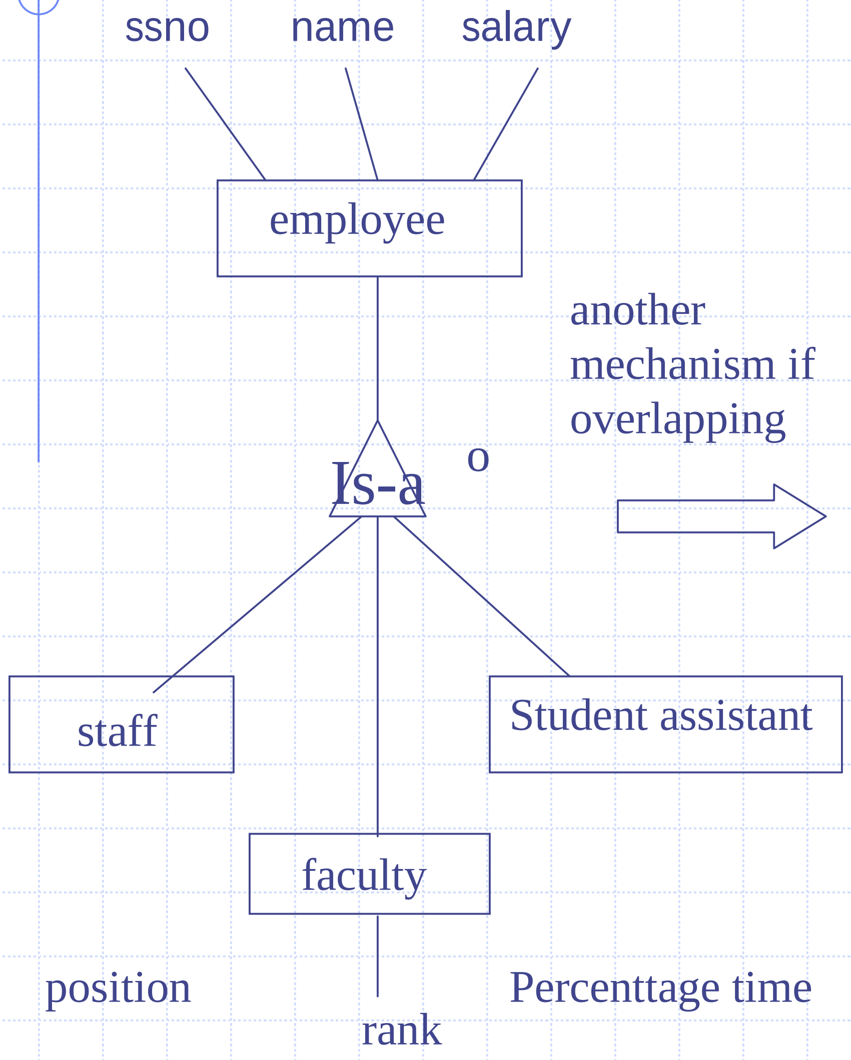
$\text{staff[ssno]} \subseteq \text{employee[ssno]}$

$\text{faculty[ssno]} \subseteq \text{employee[ssno]}$

$\text{student\_assistant[ssno]} \subseteq \text{employee[ssno]}$

cannot represent total constraint

ER to Relational Mapping



Relation:

employee(ssno, name, salary, Isstaff, position, Isfaculty, rank, Isstudentassistant, percentage-time)

Key : ssno

Isstaff, Isfaculty, Isstudent_assistant are boolean values which are either true or false. The relation will contain lot of null values

cannot represent total constraint.

ER to Relational Mapping

- ER Diagrams can be mapped to relational model (except sometimes total participation in a superclass/ subclass relationship is difficult to model)
- Recall that at times during the design cycle we wish to do the reverse-- that is, map relational schema to ER model.
- Can this always be done ?
- So far the mapping to be correct, we should be able to represent the constraints over a relational schema in the ER model.
- Constraints in relational schema -- functional dependencies, inclusion dependencies.
- Constraints in ER model: key constraints, cardinality constraints, participation constraints.
- Can we model fds and INDs using the constraints in ER model?

ER to Relational Mapping

Example

- Consider we wish to build a catalog with three fields:

  street, city, zip
- So least we need to do is create an entity with the three attributes:

  - street city uniquely determines zip

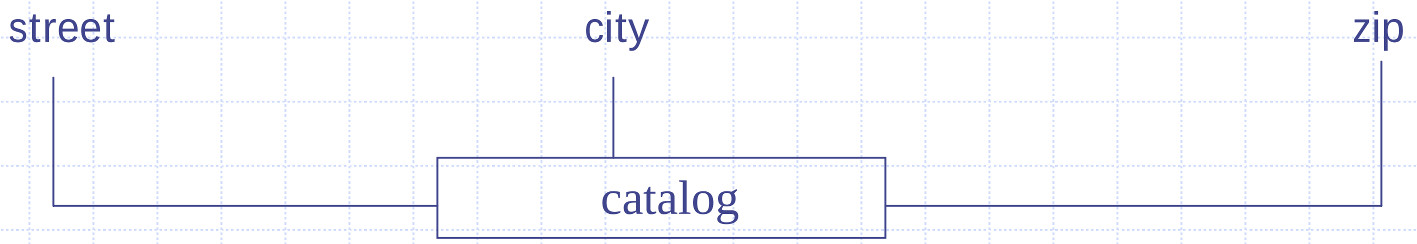
  -zip uniquely determines city
- This can be modelled using the following two FDs in the relational model:

  -street city $\longrightarrow$ zip

  -zip $\longrightarrow$ city
- Can the same be modelled in ER using the set of constraints present?

ER to Relational Mapping

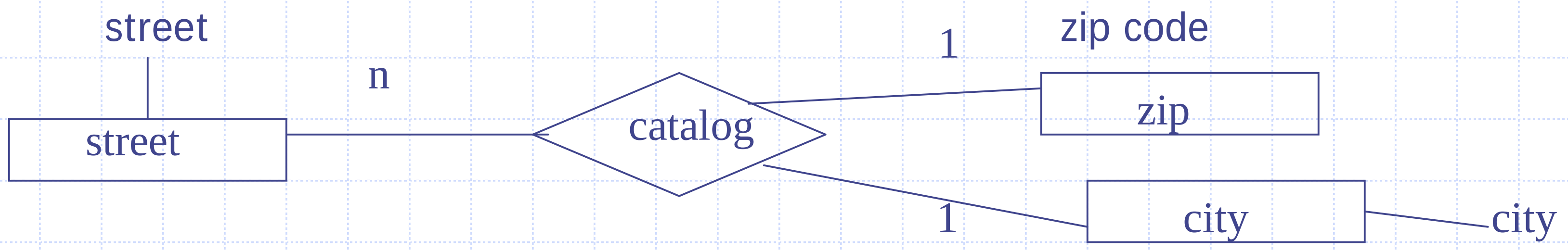
Example



Assume we create a single entity with the three attributes.
The only constraints that can be applied are the key constraints
{street, city} and {street, zip} are keys.
This however, does not prevent presence of two catalog objects:
(kirby, champaign, 61801)
(florida, urbana, 61801)
which should be prevented since zip uniquely determines a city

ER to Relational Mapping

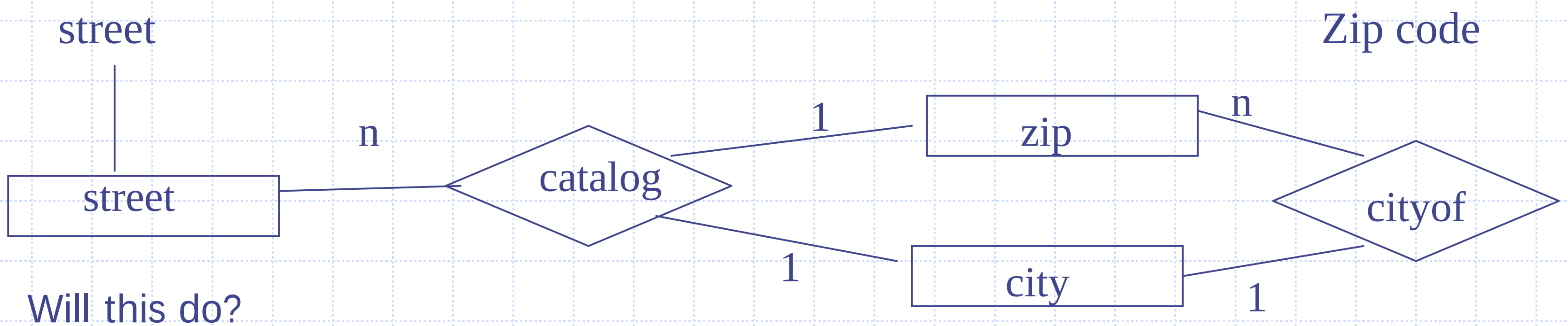
Example



Lets try creating an entity for each attribute and a relationship involving each entity.
We can now use cardinality constraints to get the required constraints?
Notice that street city uniquely determine zip, so relationship is functional wrt zip.
Similarly, street zip uniquely determine city, so relationship is functional wrt city.
But how can we model a constraint zip determines the city which involves only two entities using a ternary relation?
This shema will also not prevent the catalog objects:
(kirby, champaign, 61801)
(florida, urbana, 61801)
which should be prevented since a zip uniquely determines a city !!

ER to Relational Mapping

Example



Will this do?

No! since city-of may be an empty relationship and will thus not prevent
(kirby, champaign, 61801)
(florida, urbana, 61801)

which should be prevented since a zip uniquely determines a city!!

Actually, it can be formally shown that no ER schema can be used to represent the constraints in this example

(you should try other possibilities at home to convince yourself)

ER to Relational Mapping

- Conceptual Modelling -- ER diagrams
- ER schema transformed to relational schema (**ER to relational mapping**).
- Add additional constraints at this stage to reflect real world.
- Resulting relational schema normalized to generate a good schema **(schema normalization process)**

  - avoid redundancy

  - avoid anomalies

  - semantically equivalent to the original schema
- Schema is tested <u>example databases</u> to evaluate its quality
- Correctness results analyzed and corrections to schema are made
- Corrections may be translated back to conceptual model to keep the conceptual description of data consistent **(relations to ER mapping).**
- We have seen the ER to relation mapping. We next study normalization process.