

## V. Database Design

How to obtain a good relational database schema

- Deriving new relational schema from ER-diagrams
- Normal forms: use of constraints in evaluating existing relational schema

## Translating an E-R Model to a Relational Schema

General approach is straightforward

- Each entity set maps to a new table
- Each attribute maps to a new table column
- Each relationship set maps to either new table columns or to a new table

## Representing Strong Entity Sets

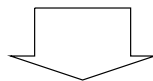
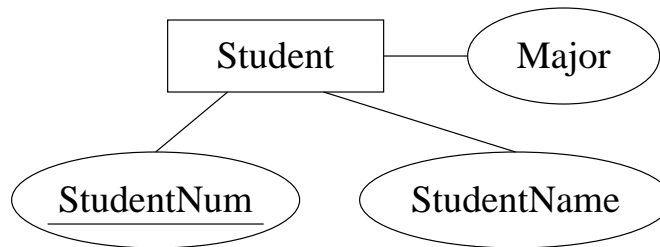
Entity set  $E$  with attributes  $a_1, \dots, a_n$

→ table  $E$  with attributes  $a_1, \dots, a_n$

Entity of type  $E \leftrightarrow$  row in table  $E$

Primary key of entity set → primary key of table

**Ex.**



Student

<u>StudentNum</u>	StudentName	Major
-------------------	-------------	-------

## Representing Weak Entity Sets

Weak entity set  $E \rightarrow$  table  $E$

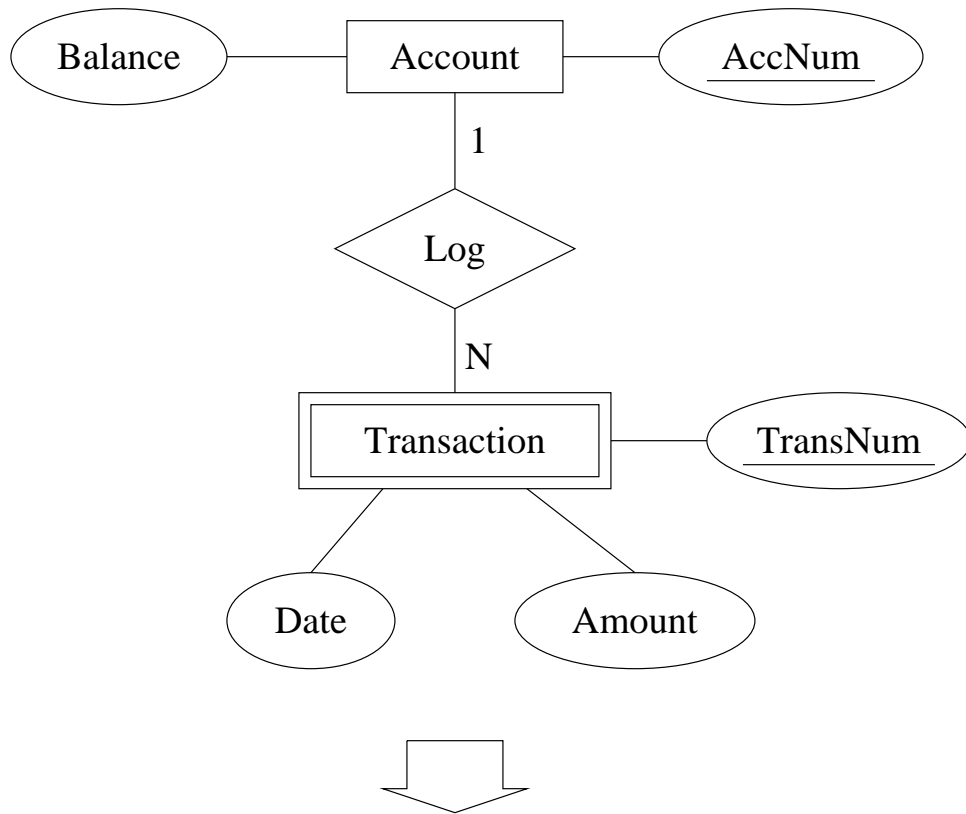
Columns of table  $E$  should include

- Attributes of the weak entity set
- Attributes of the identifying relationship set
- Primary key attributes of entity set for dominating entities

Primary key of weak entity set  $\rightarrow$  primary key of table

## Representing Weak Entity Sets (cont'd)

Ex.



Account

<u>AccNum</u>	Balance
---------------	---------

Transaction

<u>TransNum</u>	<u>AccNum</u>	Date	Amount
-----------------	---------------	------	--------

## Representing Relationship Sets

If the relationship set is an identifying relationship set for a weak entity set then no action needed

If we can deduce the general cardinality constraint (1,1) for a component entity set  $E$  then add following columns to table  $E$

- Attributes of the relationship set
- Primary key attributes of remaining component entity sets

Otherwise: relationship set  $R \rightarrow$  table  $R$

## Representing Relationship Sets (cont'd)

Columns of table  $R$  should include

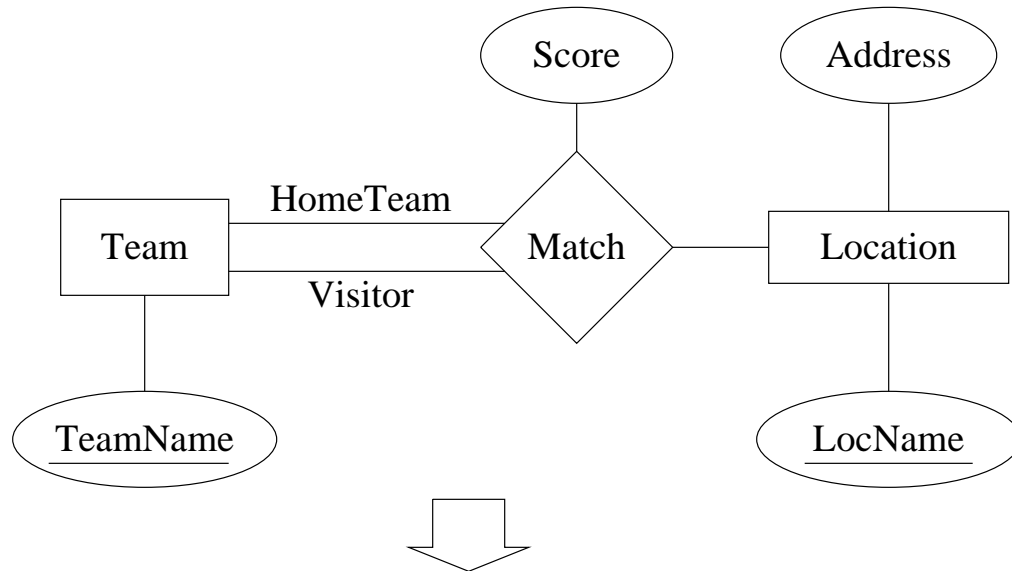
- Attributes of the relationship set
- Primary key attributes of each component entity set

Primary key of table  $R$  determined as follows

- If we can deduce the general cardinality constraint  $(0,1)$  for a component entity set  $E$ , then choose the primary key attributes for  $E$
- Otherwise, choose primary key attributes of each component entity

## Representing Relationship Sets (cont'd)

Ex.



Team

<u>TeamName</u>
-----------------

Location

<u>LocName</u>	Address
----------------	---------

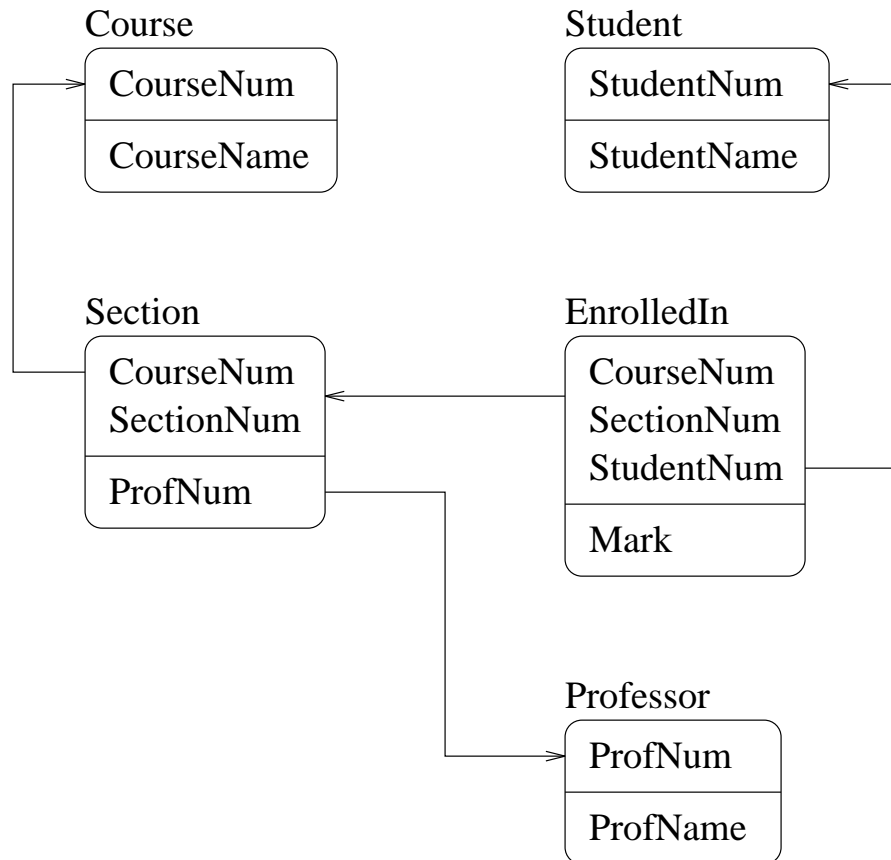
Match

<u>HomeTeamName</u>	VisitorTeamName	<u>LocName</u>	Score
---------------------	-----------------	----------------	-------

Note that the role name of a component entity set should be prepended to its primary key attributes, if supplied



## Example Translation



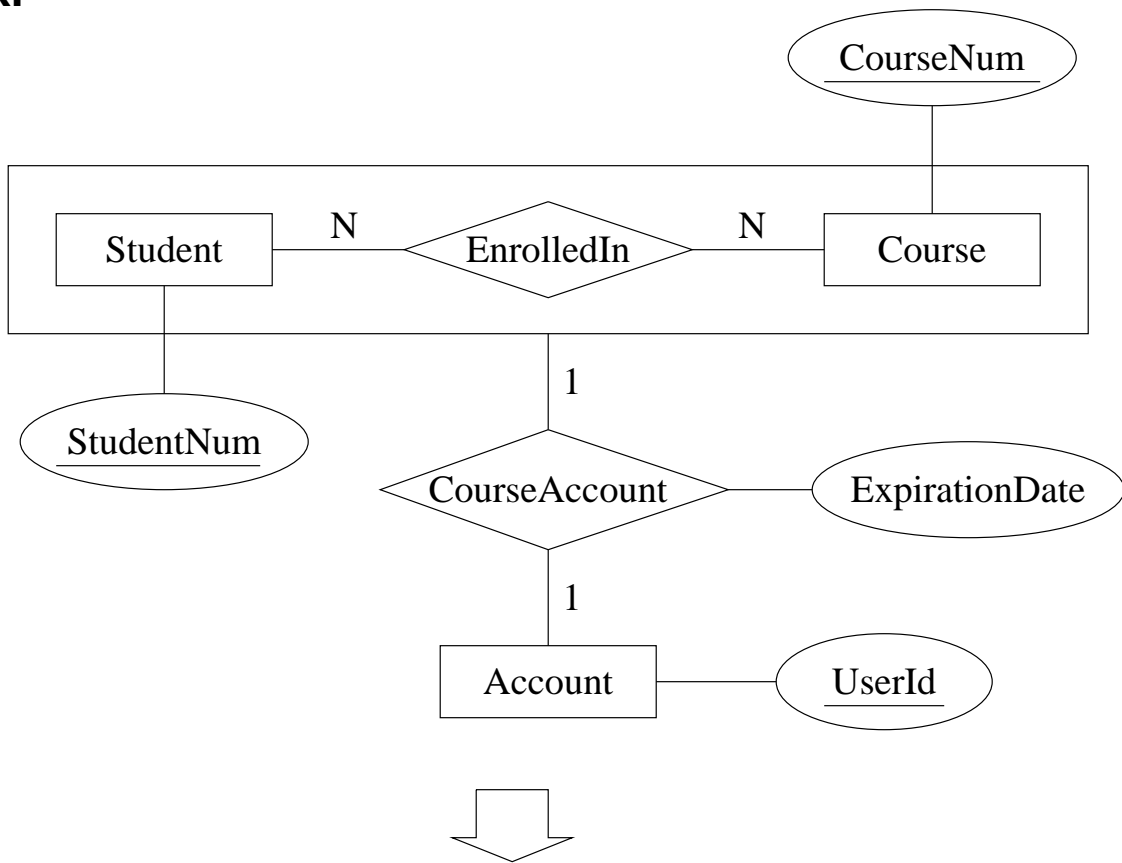
## Representing Aggregation

Tabular representation for aggregation of relationship set  $R$   
= tabular representation for relationship set  $R$

To represent relationship set involving aggregation of  $R$ , treat the aggregation like an entity set whose primary key = primary key of the table for  $R$

## Representing Aggregation (cont'd)

Ex.



Student

<u>StudentNum</u>
-------------------

Course

<u>CourseNum</u>
------------------

Account

<u>UserId</u>
---------------

EnrolledIn

<u>StudentNum</u>	<u>CourseNum</u>
-------------------	------------------

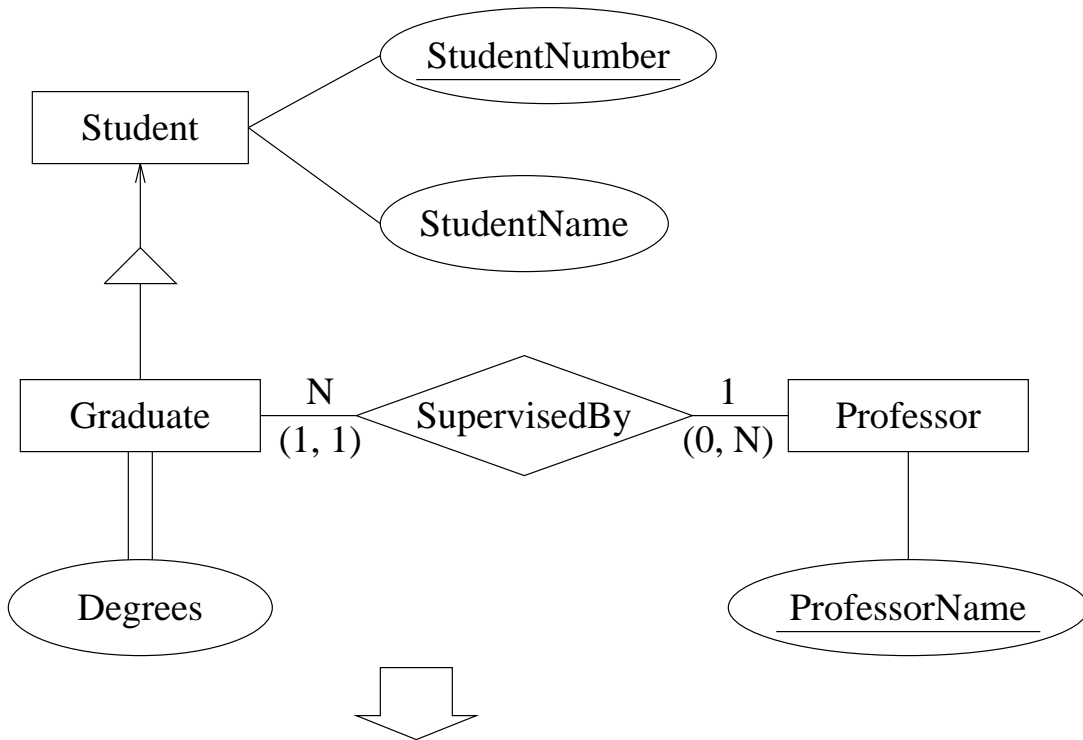
CourseAccount

<u>UserId</u>	StudentNum	CourseNum	ExpirationDate
---------------	------------	-----------	----------------

## Representing Specialization

Create table for higher-level entity set, and treat specialized entity subsets like weak entity sets

Ex.



Student

<u>StudentNumber</u>	StudentName
----------------------	-------------

Professor

<u>ProfessorName</u>
----------------------

Graduate

<u>StudentNumber</u>	ProfessorName
----------------------	---------------

Degree

<u>StudentNumber</u>	Degree
----------------------	--------

## Representing Generalization

Create a table for each lower-level entity set only

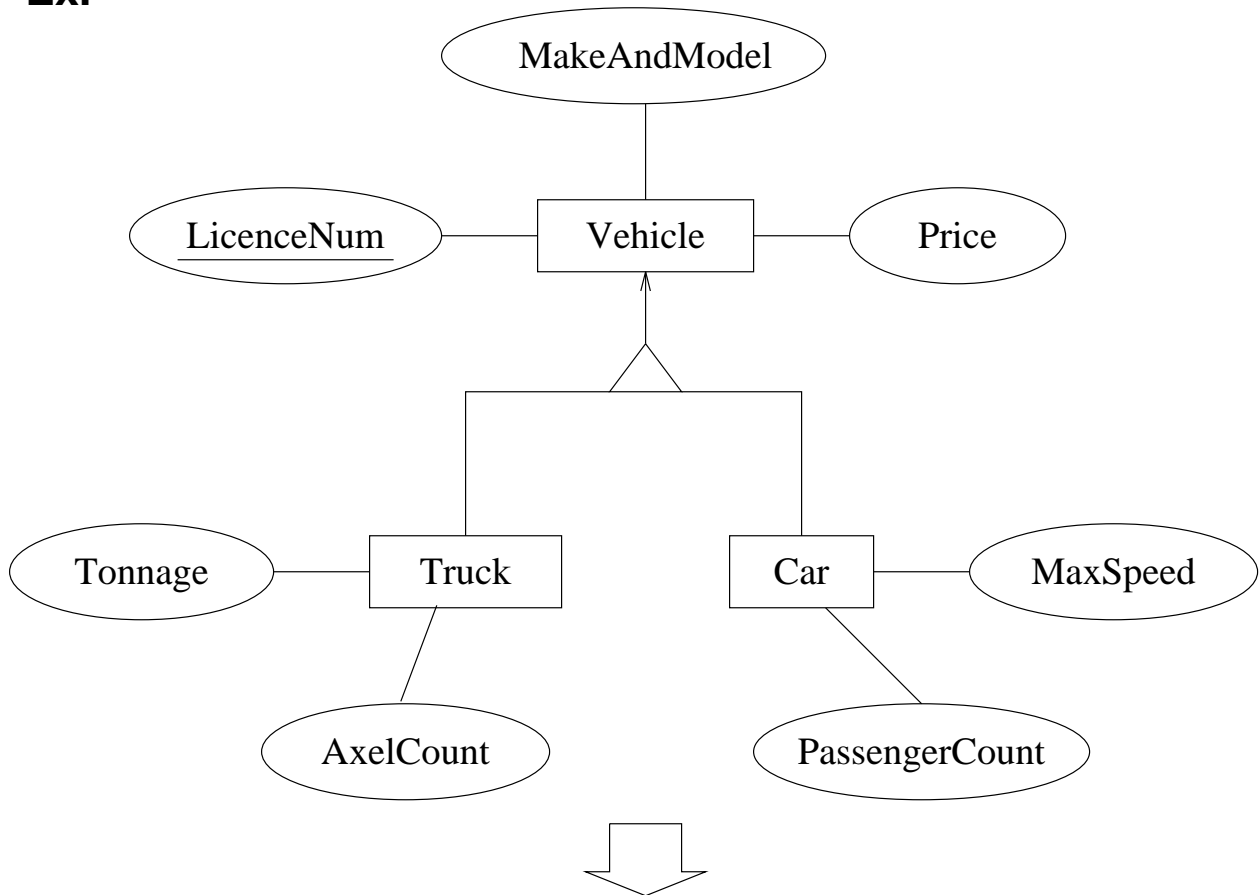
Columns of new tables should include

- Attributes of lower level entity set
- Attributes of the superset

The higher-level entity set can be defined as a view on the tables for the lower-level entity sets

## Representing Generalization (cont'd)

Ex.



Truck

LicenceNum	MakeAndModel	Price	Tonnage	AxelCount
------------	--------------	-------	---------	-----------

Car

LicenceNum	MakeAndModel	Price	MaxSpeed	PassengerCount
------------	--------------	-------	----------	----------------

## Normal Forms

What is a good relational database schema?

How can we evaluate an existing relational schema?

Goals:

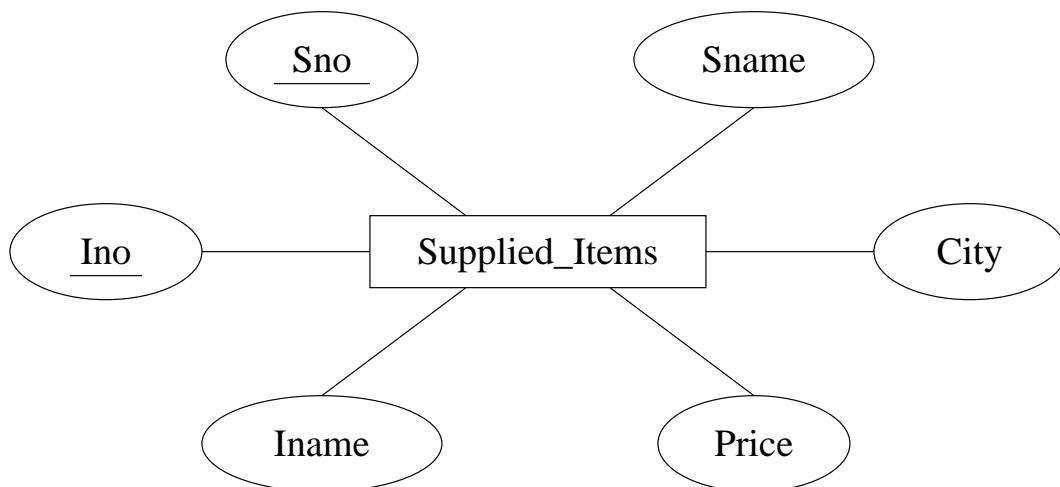
- Intuitive and straightforward changes
- Nonredundant storage of data

**We review:**

- Boyce-Codd Normal Form (BCNF)
- Third Normal Form (3NF)

## Change Anomalies

Assume we are given the E-R diagram





## Change Anomalies (cont'd)

This maps to

Supplied\_Items

<u>Sno</u>	Sname	City	<u>Ino</u>	Iname	Price
S1	Magna	Ajax	I1	Bolt	0.50
S1	Magna	Ajax	I2	Nut	0.25
S1	Magna	Ajax	I3	Screw	0.30
S2	Budd	Hull	I3	Screw	0.40

### Problems

1. Update problems (e.g. changing name of supplier)
2. Insert problems (e.g. add a new item)
3. Delete problems (e.g. Budd no longer supplies screws)
4. Likely increase in space requirements

## Change Anomalies (cont'd)

Now compare to

Supplier

<u>Sno</u>	Sname	City
S1	Magna	Ajax
S2	Budd	Hull

Item

<u>Ino</u>	Iname
I1	Bolt
I2	Nut
I3	Screw

Supplies

<u>Sno</u>	<u>Ino</u>	Price
S1	I1	0.50
S1	I2	0.25
S1	I3	0.30
S2	I3	0.40

## Change Anomalies (cont'd)

But other extreme is also undesirable (information about relationships is lost)

Snos	Snames	Cities
<u>Sno</u>	<u>Sname</u>	<u>City</u>
S1	Magna	Ajax
S2	Budd	Hull

Inums	Inames	Prices
<u>Inum</u>	<u>Iname</u>	<u>Price</u>
I1	Bolt	0.50
I2	Nut	0.25
I3	Screw	0.30
		0.40

## **Good Database Design**

What is a “good” relational database schema?

Rule of thumb: Independent facts in separate tables

or: Each relation schema should consist of a primary key and a set of mutually independent attributes

## Functional Dependencies

Generalizes notion of superkey

Used to characterize BCNF and 3NF

Some notation: Allow projection operation on tuples

**Ex.** when  $t$  is the first tuple in Supplier,

- $t[\text{Sno}] = (\text{S1})$
- $t[\text{Sname, City}] = (\text{Magna, Ajax})$

## Examples of Functional Dependencies

Consider

EmpProj

<u>SIN</u>	<u>PNum</u>	Hours	EName	PName	PLoc	Allowance
------------	-------------	-------	-------	-------	------	-----------

Key constraint forbids two different rows  $t$  and  $u$  in EmpProj with  $t[\text{SIN}, \text{PNum}] = u[\text{SIN}, \text{PNum}]$

Also disallowed

- one SIN with two employee names
- one project number with two project names or two locations
- different allowances for the same number of hours at the same location

Notation

- $\text{SIN} \rightarrow \text{EName}$
- $\text{PNum} \rightarrow \text{PName}, \text{PLoc}$
- $\text{PLoc}, \text{Hours} \rightarrow \text{Allowance}$

## Formal Definitions

Let  $R$  be a relation schema, and  $X, Y \subseteq R$ .

The **functional dependency**

$$X \rightarrow Y$$

holds on  $R$  if no legal instance of  $R$  contains two tuples  $t$  and  $u$  with  $t[X] = u[X]$  and  $t[Y] \neq u[Y]$

$X$  *functionally determines*  $Y$ ,  
 $Y$  is *functionally dependent* on  $X$

**Keys again:**  $K \subseteq R$  is a superkey for relation schema  $R$  if dependency  $K \rightarrow R$  holds on  $R$

Functional dependencies are constraints on *all* instances of a schema; they are design decisions based on the semantics of the attributes

---

---

A relation can confirm that a functional dependency does not hold; the relation cannot confirm that a functional dependency must always hold.

---

---

## Formal Definitions (cont'd)

**Ex.**

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Al-Nour
Hall	Compilers	Hoffman
Brown	Data Structures	Augenthaler

Let  $F$  denote a set of functional dependencies over  $R$ .

The **closure of  $F$** , denoted by  $F^+$  is the set of all functional dependencies that are satisfied by every relation instance that satisfies  $F$

(= *logical implications* of  $F$ )

**Note:**  $F \subseteq F^+$



## Reasoning About Functional Dependencies

Need to be able to

- find superkeys
- decide membership in  $F^+$
- find minimal covers

---

---

There are different sets of functional dependencies that have the same logical implications. Small simple sets are desirable, and are called **minimal covers**.

---

---

## Reasoning About Functional Dependencies (cont'd)

Logical implications can be derived by using inference rules called **Armstrong's axioms**

- (reflexivity)  $Y \subseteq X \Rightarrow X \rightarrow Y$
- (augmentation)  $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- (transitivity)  $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

Additional rules can be derived

- (union)  $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
- (decomposition)  $X \rightarrow YZ \Rightarrow X \rightarrow Y$

The axioms are

- sound (anything derived from  $F$  is in  $F^+$ )
- complete (anything in  $F^+$  can be derived)

## Reasoning (cont'd)

**Ex.** Let  $F$  consist of

$SIN, PNum \rightarrow Hours$   
 $SIN \rightarrow EName$   
 $PNum \rightarrow PName, PLoc$   
 $PLoc, Hours \rightarrow Allowance$

A derivation of:  $SIN, PNum \rightarrow Allowance$

1.  $SIN, PNum \rightarrow Hours$  ( $\in F$ )
2.  $PNum \rightarrow PName, PLoc$  ( $\in F$ )
3.  $PLoc, Hours \rightarrow Allowance$  ( $\in F$ )
4.  $SIN, PNum \rightarrow PNum$  (reflexivity)
5.  $SIN, PNum \rightarrow PName, PLoc$  (transitivity, 4 and 2)
6.  $SIN, PNum \rightarrow PLoc$  (decomposition, 5)
7.  $SIN, PNum \rightarrow PLoc, Hours$  (union, 6, 1)
8.  $SIN, PNum \rightarrow Allowance$  (transitivity, 7 and 3)

## Reasoning (cont'd)

There is a more efficient way of using Armstrong's axioms

```
function ComputeX+(X, F)  
begin  
  X+ := X;  
  while true do  
    if there exists (Y → Z) ∈ F such that  
      (1) Y ⊆ X+, and  
      (2) Z ⊄ X+  
    then X+ := X+ ∪ Z  
    else exit;  
  return X+;  
end
```

Let  $R$  be a relational schema and  $F$  a set of functional dependencies on  $R$ . Then

**Theorem:**  $X$  is a superkey of  $R$  if and only if

$$\text{ComputeX}^+(X, F) = R$$

**Theorem:**  $X \rightarrow Y \in F^+$  if and only if

$$Y \subseteq \text{ComputeX}^+(X, F)$$

## Finding Minimal Covers

A minimal cover for  $F$  can be computed in four steps. Note that each step must be repeated until it no longer succeeds in updating  $F$ .

**Step 1.** Replace  $X \rightarrow YZ$  with the pair  $X \rightarrow Y$  and  $X \rightarrow Z$ .

**Step 2.** Remove  $X \rightarrow A$  from  $F$  if  $A \in \text{Compute}X^+(X, F - \{X \rightarrow A\})$ .

**Step 3.** Remove  $A$  from the left-hand-side of  $X \rightarrow B$  in  $F$  if  $B$  is in  $\text{Compute}X^+(X - \{A\}, F)$ .

**Step 4.** Replace  $X \rightarrow Y$  and  $X \rightarrow Z$  in  $F$  by  $X \rightarrow YZ$ .

## Boyce-Codd Normal Form (BCNF)

Formalization of the goal that independent relationships are stored in separate tables

Let  $R$  be a relation schema and  $F$  a set of functional dependencies. A functional dependency  $X \rightarrow Y$  is **trivial** if  $Y \subseteq X$ .

Schema  $R$  is in **BCNF** if and only if whenever  $(X \rightarrow Y) \in F^+$  and  $XY \subseteq R$ , then either

- $(X \rightarrow Y)$  is trivial, or
- $X$  is a superkey of  $R$

A database schema  $\{R_1, \dots, R_n\}$  is in BCNF if each relation schema  $R_i$  is in BCNF

## BCNF (cont'd)

Why does BCNF avoid redundancy?

For schema Supplied\_Items we had

FD: Sno  $\rightarrow$  Sname, City

Implies: supplier name “Magna” and city “Ajax” must be repeated for each item supplied by supplier S1.

Assume FD holds over a schema  $R$  that is in BCNF. This implies

- Sno is a superkey for  $R$
- each Sno value appears on one row only
- no need to repeat Sname and City values

(Cf. the original Supplier table)

## Computing a Normal Form

What to do if a given relational schema is not in BCNF?

Strategy: identify undesirable dependencies,  
*decompose* schema

Let  $R$  be a relation schema (= set of attributes). Collection  $\{R_1, \dots, R_n\}$  of relation schemas is a **decomposition** of  $R$  if

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

A good decomposition does not

- lose information
- complicate checking of constraints



## Lossless-Join Decompositions

We should be able to construct the original table from its decomposition

**Ex.** Consider replacing

Marks

<u>Student</u>	<u>Assignment</u>	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Bob	A1	G2	60

by decomposing (i.e. projecting) into two tables

SGM

<u>Student</u>	<u>Group</u>	<u>Mark</u>
Ann	G1	80
Ann	G3	60
Bob	G2	60

AM

<u>Assignment</u>	<u>Mark</u>
A1	80
A2	60
A1	60

## Lossless-Join Decompositions (cont'd)

But computing the natural join of SGM and AM produces

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Ann	A1	G3	60 !
Bob	A2	G2	60 !
Bob	A1	G2	60

We get extra data, **spurious tuples**, and would therefore lose information if we were to replace Marks by SGM and AM.

If converse is true, if re-joining SGM and AM would **always** produce exactly the tuples in Marks, then we call SGM and AM a **lossless-join decomposition**.

## Identifying Lossless-Join Decompositions

A decomposition  $\{R_1, R_2\}$  of  $R$  is lossless if and only if the common attributes of  $R_1$  and  $R_2$  form a superkey for either schema, that is

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2$$

**Ex.** In the previous example we had

$$\begin{aligned} R &= \{\text{Student, Assignment, Group, Mark}\} , \\ F &= \{(\text{Student, Assignment} \rightarrow \text{Group, Mark})\} , \\ R_1 &= \{\text{Student, Group, Mark}\} , \\ R_2 &= \{\text{Assignment, Mark}\} \end{aligned}$$

Decomposition  $\{R_1, R_2\}$  is lossy because  $R_1 \cap R_2 (= \{M\})$  is not a superkey of either SGM or AM

## Dependency Preservation

Goal: efficient testing of constraints on the decomposed schema

**Ex.** A table for a company database could be

R

Proj	Dept	Div
------	------	-----

with functional dependencies

FD1: Proj  $\rightarrow$  Dept,  
FD2: Dept  $\rightarrow$  Div, and  
FD3: Proj  $\rightarrow$  Div

Consider two decompositions

$D_1 = \{R1[Proj, Dept], R2[Dept, Div]\}$

$D_2 = \{R1[Proj, Dept], R3[Proj, Div]\}$

Both are lossless. (Why?)

## Dependency Preservation (cont'd)

Decomposition  $D_1$  lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied

In decomposition  $D_2$  we can test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an **interrelational constraint**: testing it requires joining tables R1 and R3

Let  $R$  be a relation schema and  $F$  a set of functional dependencies on  $R$ .

A decomposition  $D = \{R_1, \dots, R_n\}$  of  $R$  is **dependency preserving** if there is an equivalent set  $F'$  of functional dependencies, none of which is interrelational in  $D$

## Computing a Lossless-Join BCNF Decomposition

```
function ComputeBCNF( $R, F$ )  
begin  
  Result := { $R$ };  
  while some  $R_i \in$  Result and  $(X \rightarrow Y) \in F^+$   
    violate the BCNF condition do begin  
    Replace  $R_i$  by  $R_i - (Y - X)$ ;  
    Add { $X, Y$ } to Result;  
  end;  
  return Result;  
end
```

No efficient procedure to do this exists.

It is possible that no dependency preserving BCNF decomposition exists.

**Ex.** Consider  $R = \{A, B, C\}$  and  $F = \{AB \rightarrow C, C \rightarrow B\}$ .

## Third Normal Form (3NF)

Let  $R$  be a relation schema and  $F$  a set of functional dependencies.

Schema  $R$  is in **3NF** if and only if whenever  $(X \rightarrow Y) \in F^+$  and  $XY \subseteq R$ , then either

- $(X \rightarrow Y)$  is trivial, or
- $X$  is a superkey of  $R$ , or
- each attribute of  $Y$  is contained in a candidate key of  $R$

A database schema  $\{R_1, \dots, R_n\}$  is in 3NF if each relation schema  $R_i$  is in 3NF

Because 3NF is looser than BCNF, it allows more redundancy

A lossless-join, dependency preserving decomposition into 3NF relation schemas always exists.

## Computing a 3NF Decomposition

A lossless-join 3NF decomposition that is dependency preserving can be efficiently computed

```
function Compute3NF(R, F)  
begin  
  Result :=  $\emptyset$ ;  
  F' := a minimal cover for F;  
  for each (X → Y) ∈ F' do  
    Result := Result ∪ {XY};  
  if there is no Ri ∈ Result such that  
    Ri contains a candidate key for R then begin  
    compute a candidate key K for R;  
    Result := Result ∪ {K};  
  end;  
  return Result;  
end
```