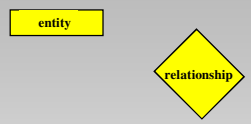
Conceptual Database Design
E-R Model

- Diagram based model
- Two primitives: entities & relationships



entity: a distinguishable object that exists on its own
  - Joe Doe, White House

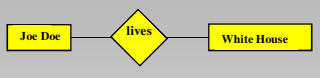
entity set: a collection of entities of the same type
  - PERSONs, HOUSEs, etc.- the notion of type is similar to that in a programming language

- entity sets may overlap
  - EMPLOYEEs and CUSTOMERs (Giant's employees are also customers)

- entities have attributes (properties that have a single value), i.e. employee name, soc.sec#
- if attributes do not have a single value, then we should model them as entities

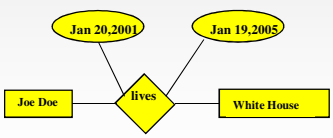
Relationships

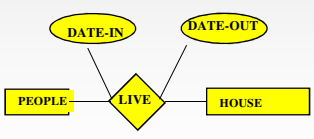
- relationship: an association among entities
  - Joe Doe lives in the White House



- relationship set: a collection of relationships of the same type
  - PEOPLE LIVE in HOUSEs

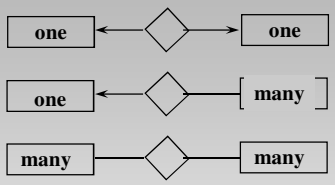
- relationships can also have attributes (properties with single values)
  - e.g. LIVE has an attribute DATE-IN to store the value 'Jan 20, 2001' the PERSON moved in the HOUSE and DATE-OUT 'Jan 19, 2005'





Mappings amongst relationships

- 1-1 (PERSONs and IRS-RECORD)
- 1-many (PERSON and ACCOUNTs)
- many-many (STUDENTs and COURSEs)

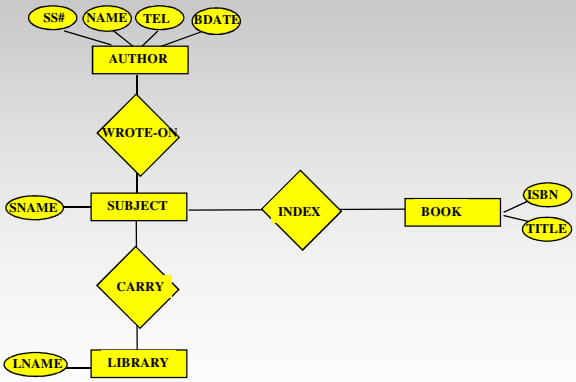


Note arrow points to the one

Our First Database Design

Application:

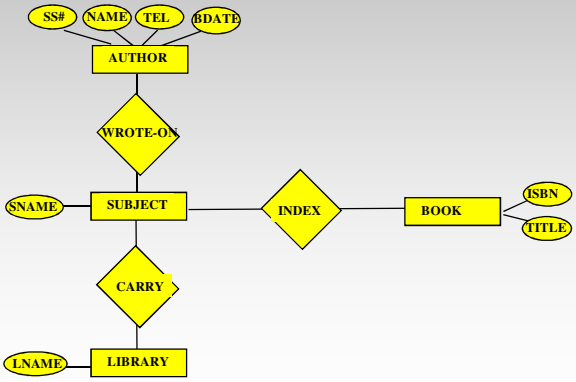
A library database that stores authors who have written books about various subjects. The database will also store info about libraries that carry books on these subjects.



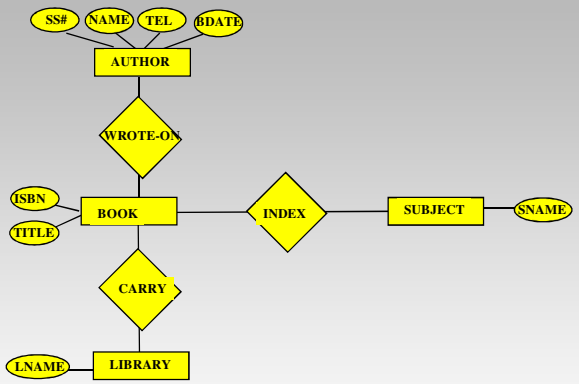
What's wrong?

Problems in our First Design

- Indirect way to find if a library carries books of a specific author
- Similarly not straight forward to find if a library carries a specific book
- Also not easy to find if an author has written a specific book



2nd Attempt to the Library Design



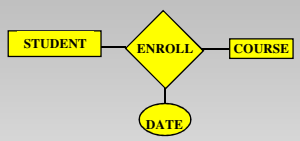
- Much better

Keys: For Entities (same as in relational model)

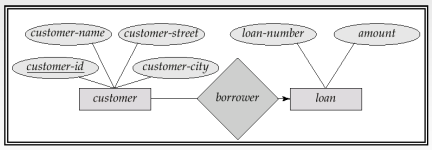
- ***superkey***: set of attributes whose values uniquely identify the entity
- ***candidate key***: a minimal superkey (or a minimal subset of a superkey whose values still uniquely identify the entity)
- ***primary key***: if more than one candidate key, one is chosen to be the primary used for the identification purpose

Keys: For Relationships (same as in relational model)

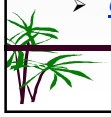
- ***many-many:*** the primary keys of involved entities plus possibly a subset of relationship attributes that required to further distinguish the relationship



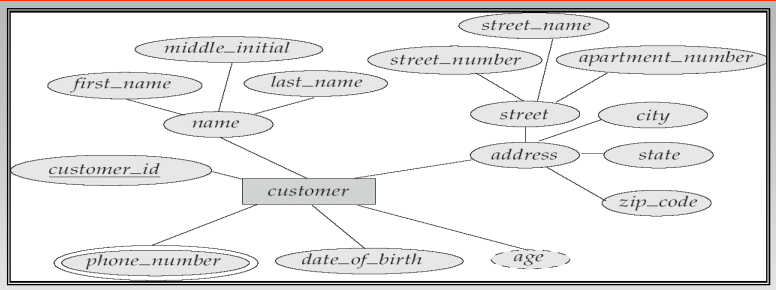
- ***one-many:*** the primary key of the "many" entity plus possibly a subset of relationship attributes that required to further distinguish the relationship
  - customer-id here is sufficient even when there are more than one borrowers in a loan because each borrower can only have at most 1 loan-



- ***one-one:*** the primary key of any of the entities can be used

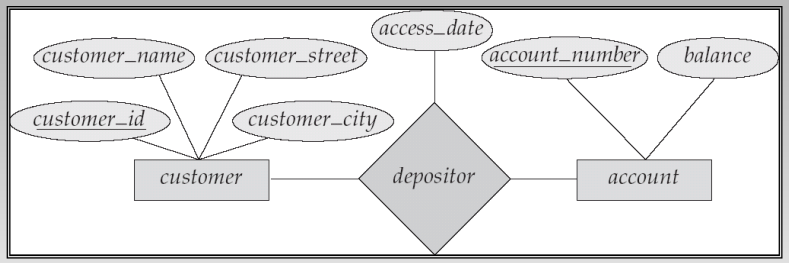


Notation of E-R Diagrams



- Rectangles represent entity sets.
- Lines link attributes to entity sets
- Ellipses represent attributes
  - Double ellipses represent multivalued attributes.
  - Dashed ellipses denote derived attributes.
  - Composite attributes are those with substructure
- Again, underlined attributes indicate the primary key

Notation of E-R Diagrams (cont)



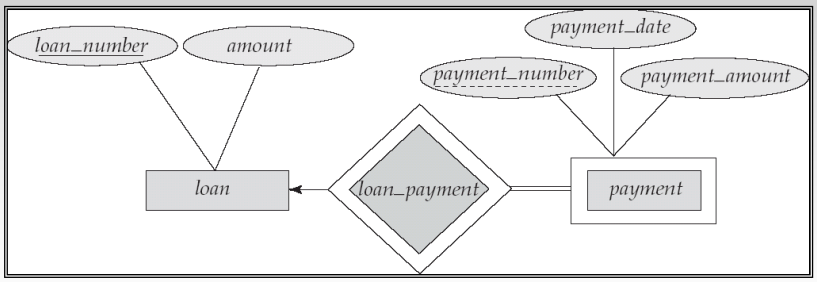
- Diamonds represent relationships.
- Lines link attributes to relationships.
- Ellipses represent attributes of relationships

Weak Entities

- An entity that does not have a primary key *on its own* is referred to as a ***weak entity.***
- The existence of a weak entity set depends on the existence of a identifying ***strong entity***
  - e.g. CHILDREN of EMPLOYEEs is a weak entity
  - a weak entity is "existent dependent" on a strong entity- when the strong entity gets deleted, so does the weak
- The ***discriminator*** (***or partial key)*** of a weak entity is the set of attributes that distinguishes among all the entities of a weak entity.
- The "primary key" of a weak entity is formed by the primary key of the strong entity it depends on and its discriminator.

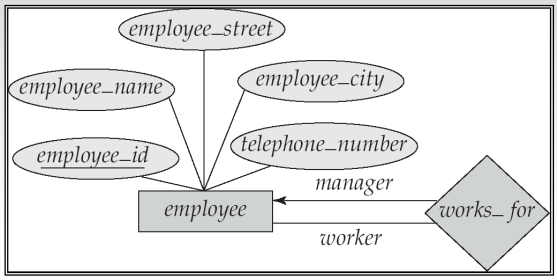
Weak Entities (Cont.)

- We depict a weak entity by double rectangles.
- One-to-Many relationship between the strong and the weak entity
  - If it were 1-1, we would model it as an attribute (e.g. ss# or gender)
- We underline the discriminator of a weak entity with a dashed line.
  payment_number is the discriminator of the *payment* entity
- Primary key for ***payment*** – (***loan_number, payment_number***)



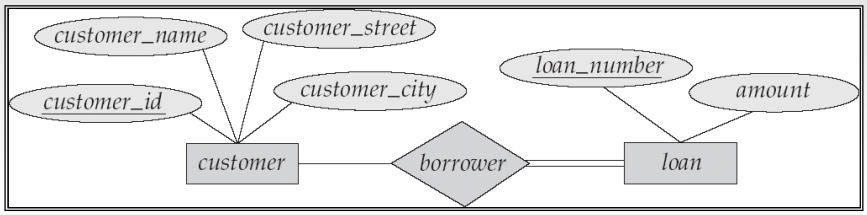
Roles

- **Entites of a relationship are not necessarily distinct**
- **The labels "manager" and "worker" are called *roles*; they specify how employee entities interact via the works_for relationship set.**
- **Roles are indicated in E-R diagrams by labeling the lines**
  - **Role labels are optional, and are used to clarify semantics of the relationship**



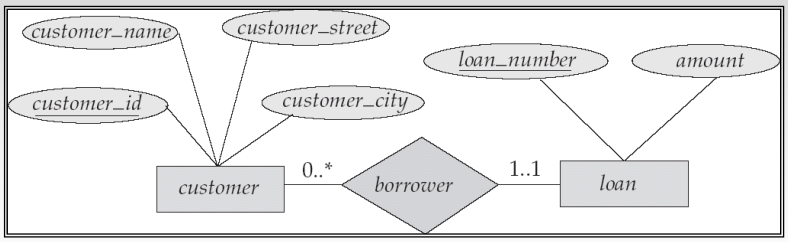
Participation of an Entity in a Relationship

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g. participation of loan in borrower is total
    - every loan must have a customer associated to it via borrower
- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of customer in borrower is partial

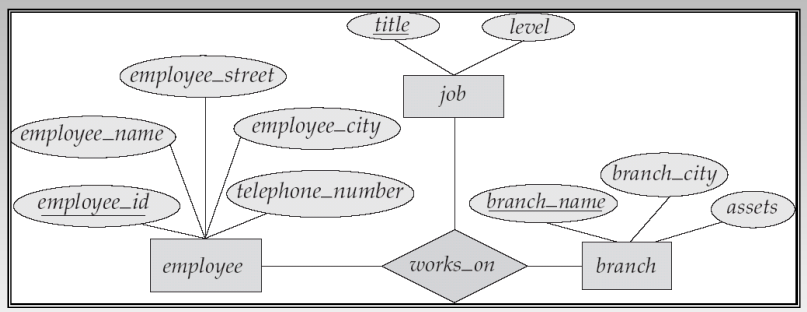


Alternative Notation for Cardinality Limits

- Cardinality limits can also express participation constraints
  - l..h can express minimum (l) or maximum (h) paricipation
  - l=1 is total participation (1..*) equivalent to double line
  - h=1 entity is in at most 1 relationship (1..1 means exactly one)
  - h=* entity has no limit on the participating relationship (0..*)

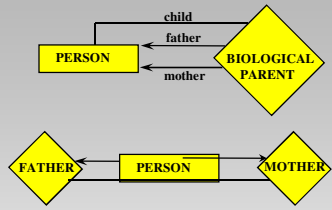


Ternary Relationship

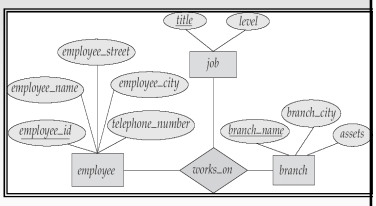


Binary Vs. Non-Binary Relationships

- Some relationships that appear to be non-binary *BIOLOGICAL-PARENT*
- May be better represented using binary relationships *FATHER* and *MOTHER*
  - *Note these relationships are between same entities*

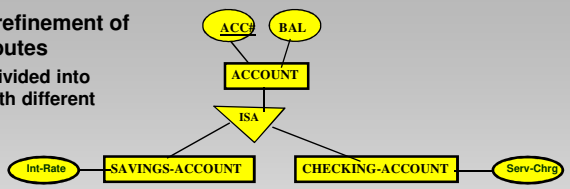


- But there are some relationships that are naturally non-binary E.g. *works_on*

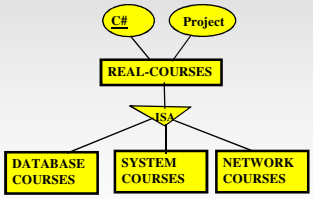


Extended E-R: Specialization-Generalization (ISA Hierarchy)

- specialization: top-down refinement of entities with distinct attributes
  - e.g. bank accounts are divided into checking and savings with different attributes

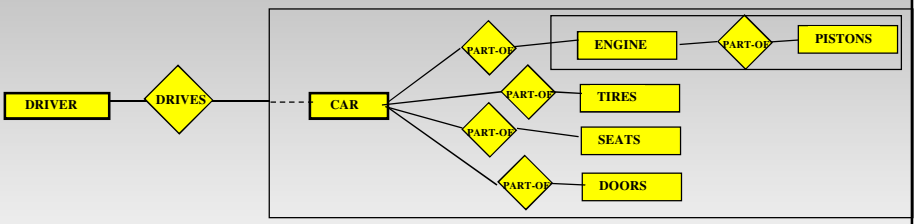


generalization: bottom-up abstraction of common attributes
  - e.g. from Database, System, and Network courses with common attribute Project we can abstract RealCourses
  - all other course attributes are included



Extended E-R: Aggregation
(Part-of Hierarchy)

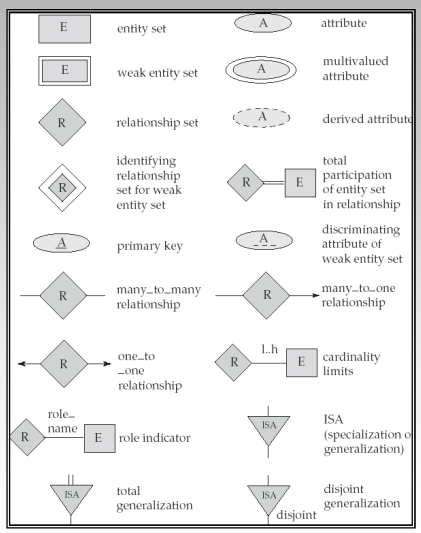
- relationships among relationships are not supported
- often we want to treat lower level relationships separately modeled



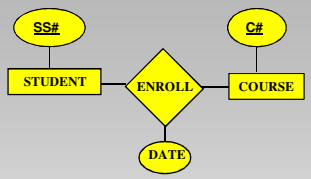
- e.g. parts and subparts in a car should not be each individually related to the driver

- Car is now like a high level entity with internal substructure (difficult to deal)

Summary of Symbols Used in E-R Notation



Mapping E-R to Relations



- **strong entities:**
  - STUDENT(SS#,SNAME,…)
  - COURSE(C#,CNAME,…)
- **relationships:**
  - ENROLL(SS#,C#,DATE)
- **weak entities:** COURSE-LAB with non-primary key and attributes EQUIPMENT and ROOM depending on strong entity COURSE will be mapped to
  - COURSE-LAB(C#,EQUIPMENT,ROOM)

Mapping E-R to Relations

- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: *instructor* with composite attribute *name= first_name* and *last_name* is mapped to

Instructor(ID, first_name, last_name)

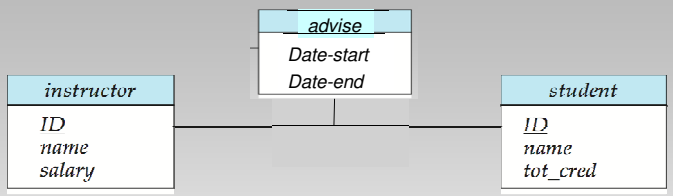
- Multi-valued attributes are mapped into a separate relations
  - Example: *instructor* with multi-valued attribute {phone-number} is mapped to

Instuctor_phone= (ID, phone_number)

Mapping E-R to Relations

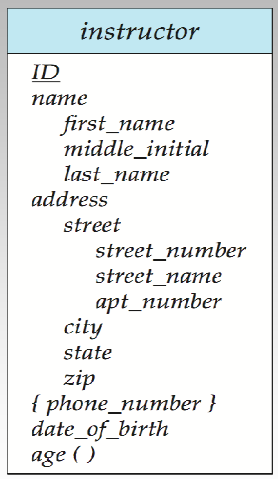
- ISA relationships
  - case 1: map only the supper class entity with an additional attribute that distinguishes the two subclasses (e.g. savings from checking accounts)
  - case 2: map the two subclasses to a separate relation and ignore the super class (this is good when the subclasses are disjoint and span the whole superclass)
- Aggregate relationships map the entities and relationships the same way- just treat the aggregate as an entity

UML Variation of E-R Diagrams
Unified Modeling Language

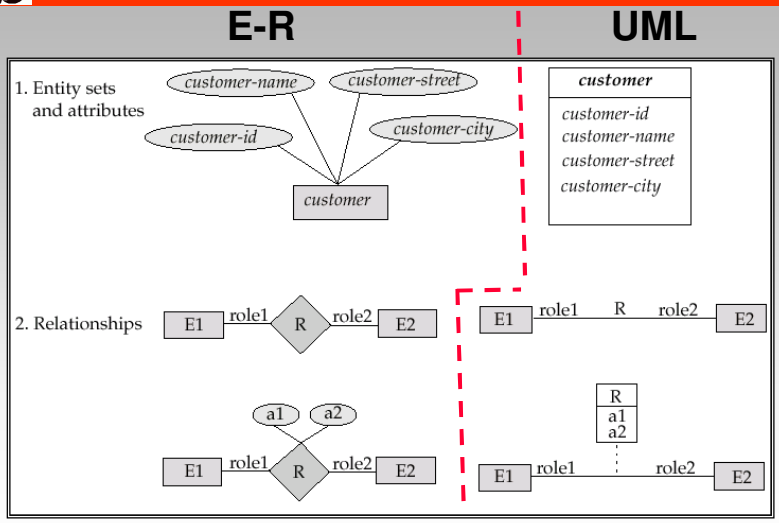


- Rectangles represent entity sets.
- Edge labels represent relationship sets.
- Attributes listed inside (entity/relationship) rectangles
- Underline indicates primary key attributes

Entity With Composite, Multivalued, and Derived Attributes

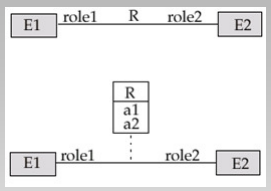


E-R & UML Diagram Notations



UML Class Diagrams (Cont.)

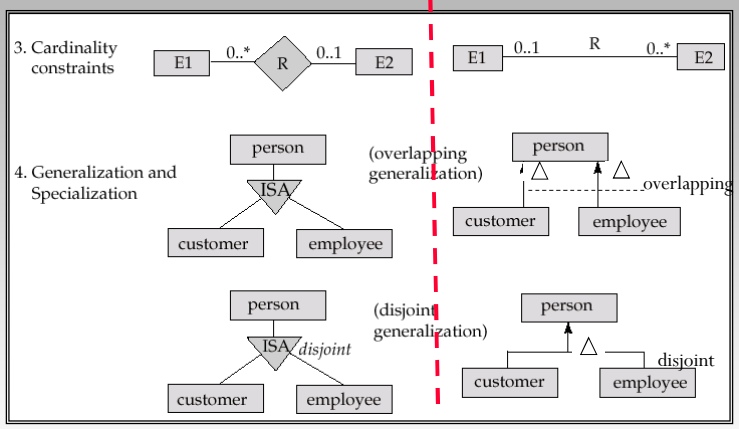
- **Binary relationship: R with roles**



- **Binary relationship: R with roles and Attributes a1 and a2**

- **Non-binary relationships drawn using diamonds, just as in ER diagrams**

UML Class Diagram Notation (Cont.)

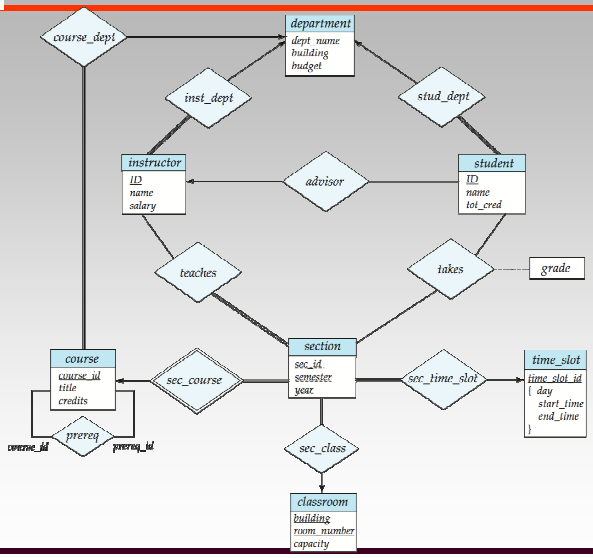


*Note reversal of position in cardinality constraint depiction

UML Class Diagrams (Contd.)

- Cardinality constraints are specified in the form *l..h*, where *l* denotes the minimum and *h* the maximum number of relationships an entity can participate in.
- Beware: the positioning of the constraints is exactly the reverse of the positioning of constraints in E-R diagrams.
- The constraint 0..* on the *E*2 side and 0..1 on the *E*1 side means that each *E*2 entity can participate in at most one relationship, whereas each *E*1 entity can participate in many relationships; in other words, the relationship is many to one from *E*2 to *E*1.
- Single values, such as 1 or * may be written on edges; The single value 1 on an edge is treated as equivalent to 1..1, while * is equivalent to 0..*.

E-R Diagram for a University Database



Relational Database Design

- goal to have a schema that
  - makes queries simpler
  - avoids redundancies and update anomalies

Example:

EMP(eno,ename,byr,sal,dno)
DEPT(dno,dname,floor,mgr)

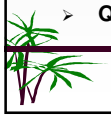
Q1: find all employees that make more than their manager

```
select e.ename
from EMP e, EMP m, DEPT d
where e.dno=d.dno and d.mgr=m.eno and e.sal > m.sal
```

Q2: For each department, find its employees' highest salary

```
select  d.dname, max(e.sal)
from  EMP e, DEPT d
 where e.dno=d.dno
 group by(d.dno)
```

- Q1 requires 2 joins
- Q2 1 join and a group by



Simplicity of DB Design & Anomalies

EMP⋈DEPT=ED(eno,ename,byr,sal,dno,dname,floor,mgr)

Q1: find all employees that make more than their manager

```
select e.ename
from ED e, ED m
where   e.mgr=m.eno and e.sal > m.sal
```

Q2: For each department, find the maximum salary

```
select dname, max(sal)
from ED
groupby(dno)
```

- Q1 now requires 1 join
- Q2 no join
- Simpler and easier queries
- should we then pre-compute the joins and keep bigger relations?
- Perhaps create the ***universal relation*** with all attributes in and NULLs for those values that make no sense. Right or Wrong?

Combining Schemas?

- Suppose we combine

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
69987	White	null	null

instructor

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000
null	Taylor	null

department

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Key?

Repetition of information

A Combined Schema Without Repetition

- Consider combining relations

  sec_classroom(<u>course_id</u>, <u>sec_id</u>, building, room_number)
  section(<u>course_id</u>, <u>sec_id</u>, semester, year)

  section(<u>course_id</u>, <u>sec_id</u>, semester, year, building, room_number)

- No repetition in this case.
- When do we have repetition and when not?

Update Anomalies

- Redundancy (repetition of info)
  - each department location and its budget is repeated for each instructor of it
  - there is a potential for inconsistencies (update anomalies) where the location of the department is changed in some tuples but not in others
- No independence existence (inability to represent certain info)
  - ***insertion anomaly***: a new department "PokemonGo Sci" to be located in the Irebe building cannot be stored until the first instructor is hired
  - ***deletion anomaly***: a department can no longer exist when the last instructor "Crick" is deleted
- if we discover such anomalies we decompose the relations until the problems go away.

Fixing Update Anomalies

- Suppose we had started with *inst_dept*

  Inst_dept(ID, name, salary, dept_name, building, budget)

  How do we know to split up the relation to avoid redundancies?
- Because *dept_name* is not a candidate key
  - the building and budget of a department will be repeated for every instructor of the same department
- We also know, that every department is located in exactly one building- Denote this as a *functional dependency (FD):*

  dept_name → building, budget
- We can use this FD to decompose Inst_dept into

  department(dept_name, building, budget)

  Instructor(ID,name,salary, dept_name)
- This process is called ***NORMALIZATION*** *(see later)*

Objectives of Database Design

Obtain relations with these properties:

- no redundancy
  - avoids repeated updates
  - better space efficiency
- semantic clarity
  - easier understood
- linguistic efficiency
  - the simpler the queries the better for the app programmer
  - also better for the query optimizer)
- Solution? Decompose all relations to Binary ones. NO repetition
- Wrong. Why?
- Performance issues
  - the smaller the relations the more joins (expensive)

Functional Dependencies

- set of attributes whose values uniquely determine the values of the remaining attributes e.g. a key defines an FD

  e.g. in EMP(eno,ename,sal) — key FDs: eno → ename; eno → sal
  DEPT(dno,dname,floor)
  WORKS-IN(eno,dno,hours) — other FDs: {eno,dno} → hours

  every pair of values of eno,dno there exists one exactly value for hours

- in general if $\alpha \subseteq R$ and $\beta \subseteq R$, then $\alpha \rightarrow \beta$ holds in the extension r(R) of R iff for any pair t1 and t2 tuples of r(R) such that t1(α)=t2(α), then it is also true that t1(β)=t2(β) (uniqueness of β values)
- we can use the FDs as
  - constraints that we want to enforce (e.g. keys)
  - for checking if the FDs are satisfied in the database

$A \rightarrow B$ satisfied?
$A \rightarrow C$ satisfied?
$C \rightarrow A$ satisfied?
$AB \rightarrow D$ satisfied?

R(A B C D)
1 1 1 1
1 2 1 2
2 2 2 2
2 3 2 3
3 3 2 4

FD Formalization

- trivial dependencies: $\alpha \rightarrow \alpha$
  $\alpha \rightarrow \beta$ if $\beta \subseteq \alpha$
- closure
  - need all FDs
  - some logically implied by others e.g. if $A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$ is implied
- given F = set of FDs,
  find F+ (the closure) of all logically implied by F

Amstrong's axioms

- reflexivity: if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ (trivial FD)
- augmentation: if $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$
- transitivity: if $\alpha \rightarrow \beta$ & $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

More FD Rules

- union rule: if $\alpha \rightarrow \beta$ & $\alpha \rightarrow \gamma$ then $\alpha \rightarrow \beta\gamma$
- decomposition rule: if $\alpha \rightarrow \beta\gamma$ then $\alpha \rightarrow \beta$ & $\alpha \rightarrow \gamma$
- pseudotransitivity rule: if $\alpha \rightarrow \beta$ & $\gamma\beta \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$

Example: R(A,B,C,G,H,I)
F={ $A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H$ }
F+= { $A \rightarrow H$ /* $A \rightarrow B \rightarrow H$ transitivity
$CG \rightarrow HI$ /* $CG \rightarrow H$, $CG \rightarrow I$ union rule
$AG \rightarrow I$ /* $A \rightarrow C$ augmentation $AG \rightarrow CG \rightarrow I$
$AG \rightarrow H$ } /* $A \rightarrow C$ augmentation $AG \rightarrow CG \rightarrow H$

- there is a non-trivial (exponential) algorithm for computing F+

Closure of Attribute Sets

- the closure α^+ of a set of attributes α under F is the set of all attributes that are functionally determined by α
- there is an algorithm that computes the closure

Example:

R(A,B,C,G,H,I) F={ $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$ }

Algorithm to Compute (AG)+

start with		result=(AG)
$A \rightarrow B$	expands	result=(AGB)
$A \rightarrow C$	expands	result=(AGBC)
$CG \rightarrow H$	"-"	result=(AGBCH)
$CG \rightarrow I$	"-"	result=(AGBCHI)
$B \rightarrow H$	no more expansion	

Note that since G is not on any right hand side, no subset of the attributes can be a superkey unless it contains G for there is no FD to generate it.

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of *R(A,B,C,D)*.

Example: AC→B, B→C, A→D , D→C

Is A a superkey?

A+={ADCB} YES

Is CB a superkey?

CB+={BC} NO

- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.

Example: Does B→D?

B+={BC}

D is not in B+ NO

Most Important Use of Attribute Closure

- **Find some or all candidate keys**
  - For each subset of the attributes, check if it is a superkey
    - 1-attribute keys A, B, C, D, E
    - 2-attribute keys AB, AC, AD, AE, BC, BD, BE, CD, CE, DE
    - 3-attribute keys ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE
    - 4-attribute key ABCD, ABCE, ABDE, ACDE, BCDE
    - All-attribute key ABCDE

Example: AB→D , A→E, E→C

- **Instead of the brute-force CPU method, we can use our BPU:**
  - **➔ observe A and B are not in the RHS of any FD**
  - **➔ they *both* have to be in any candidate key and,**
  - **➔ eliminates the need to check 1-attribute keys**
- **2-attribute keys AB+={ABDEC} is a SK**
  - **➔ any other 2-attr combination cannot be because it does not contain *both* A and B**
- **3-attr keys**
  - **➔ any 3-atribute combination containing both A and B is not a candidate key**
  - **➔ 4- and 5-atribute cannot be for the same reason**
- **Therefore, in this example AB is the ONLY candidate key.**

Testing for Extraneous Attributes in FDs

- **An extraneous attribute can be removed without affecting F^+**

Example: F={AB→CD, A→E , E→C}

Is C extraneous in AB→CD? If so, we can remove it obtaining F'={AB→D , A→E, E→C} and see if AB➔D now can regenerate C. Note that C is on the RHS

Compute AB+ using F'={AB→D , A→E, E→C}
AB+={ABDEC}

YES because it includes C and D

Is D extraneous in AB→CD?
Compute AB+ using F''= {AB→C , A→E, E→C}
AB+={ABCE}

NO because it does not include both C and D

Is B extraneous in AB→CD? Note that B is on the LHS
Compute A+ using F={AB→CD, A→E , E→C}
A+={AEC}

NO because it does not include both C and D

Extraneous Attributes in FDs (cont)

Another set of FDs G= {AB→CD, A→E , E→D, D→C}
Is B extraneous in AB→CD? (again B is on the LHS)

Compute A+ using G= {AB→CD, A→E , E→D, D→C}
A+={AEDC}

YES because it includes both C & D

Canonical Cover

- The canonical cover of a set of FDs F is the set of FDs obtained by removing all extraneous attributes
  - The extraneous attributes in F={AB→CD, A→E , E→C} is C
  - The canonical cover of F is: F'={AB→D, A→E , E→C}

Return to Decompositions

- If relations have update anomalies we decompose them
- How? We will use the FDs to make it formal and correct
- A decomposition is correct if it is reversible (lossless)
- A lossless decomposition can be re-composed by a join
- not all decompositions are reversible (lossy)

Example of Lossy Join Decompositions

SHIPMENT(S# P# J#)

S#	P#	J#
s1	p1	j1
s2	p2	j1
s2	p3	j2
s3	p3	j3
s4	p4	j3

======>>

SP(S# P#)

S#	P#
s1	p1
s2	p2
s2	p3
s3	p3
s4	p4

PJ(P# J#)

P#	J#
p1	j1
p2	j1
p3	j2
p3	j3
p4	j3

if we join again SP and PJ

SP-PJ(S# P# P# J#)

S#	P#	P#	J#
s1	p1	p1	j1
s2	p2	p2	j1
s2	p3	p3	j2
s2	p3	p3	j3
s3	p3	p3	j2
s3	p3	p3	j3
s4	p4	p4	j3

from the joined tuples we cannot deduce which were actually shipped

This is called the ***connection trap*** and the decomposition ***lossy***

Lossless Decompositions

Lossless if two conditions are satisfied:

Condition 1: **All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):**

$$R = R_1 \cup R_2$$

Condition 2: **At least one of the following dependencies is in F^+:**

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$
- **anyone of these two FDs avoids the connection trap because it guarantees uniqueness in the mapping**

EXAMPLE: R(A,B,C,D) A→B B→CD

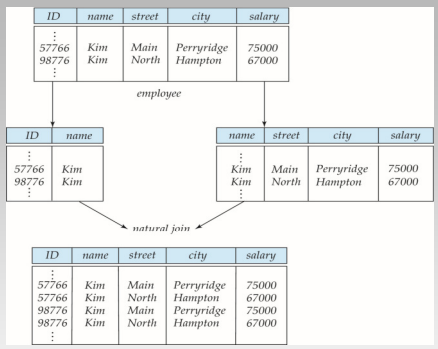
/ \

and R_1(A,B) R_2(B,C,D)

Condition 1: $R_1 \cup R_2$ = (A,B,C,D) = R

Condition 2: $R_1 \cap R_2$ = B and because of B→CD implies B → BCD = R_2

Another Example of a Lossy Join Decomposition



Condition 1: All attributes in the decomposition are present ($R = R_1 \cup R_2$)

Condition 2: It is neither $R_1 \cap R_2$ = name → R_1

nor $R_1 \cap R_2$ = name → R_2

Dependency Preservation

In a decomposition $R = R_1 \cup R_2$

- **dependencies are preserved if**
  - **No FD crosses over the two relations, i.e. all FDs in F+ have attributes that are within one of R_1 or R_2**
  - **Dependency preservation ensures that FDs remain intra-relational constraints**
  - **If an FD is not preserved (crosses over two relations), we need to join R_1 and R_2 in order to enforce FDs**
  - **In this case the FD becomes an inter-relational constraint**
- **to check if a decomposition is dependency preserving**
  - **we need to examine all FDs in F+**
  - **there is an algorithm for testing dependency preservation**

BCNF Relations

Boyce-Codd Normal Form

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

i. $\alpha \rightarrow \beta$ **is trivial (i.e., $\beta \subseteq \alpha$)**

ii. α **is a superkey for R**

Example: $R = (A, B, C)$ $F = \{A \rightarrow B \quad B \rightarrow C\}$

Is R in BCNF?

No. *$B \rightarrow C$ and B is not a superkey*

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ violates BCNF
  1. compute the attribute closure of the LHS α^+ and
  2. verify that α^+ includes all attributes of *R*, i.e., it is a superkey of *R*.

- Test: For a relation *R* with a given set of FDs F is in BCNF, it suffices to check only the FDs in the given set *F* for violation
  - NO NEED to check all FDs in F^+
- Lemma: if none of the FDs in *F* causes a BCNF violation, then none of the FDs in F^+ will.
- Unfortunately, this test applies to R only and not any decomposition R_i of R
  - E.g. Consider $R(A, B, C, D)$ with $F = \{ A \rightarrow B, B \rightarrow C\}$
    And
    $R_1(A,B)$ $R_2(A,C,D)$
  - None of the FDs in *F* have attributes in $R_2(A,C,D)$ so we might think R_2 satisfies BCNF
  - It is not the case because $A \rightarrow C$ is in *F+* and A+ not a SK, thus, R_2 violates BCNF

Alternative Simplified Test Without F+

- In a decomposition we may get a relation R_i for which we need to check BCNF violation
- Test: For every subset α of R_i begin {
  let $X = R_i - \alpha$
  compute $\alpha+$ under F (not F+)
  if either $\alpha+$ contains SOME attributes of $X(=R_i-\alpha)$ but not ALL
  then R_i violates BCNF and exit }
  /* else if $\alpha+$ includes ALL or NO attributes of X, R_i does not violate BCNF */
  If no subset violates BCNF, test succeeds R_i is in BCNF
- Retry previous example $R(A, B, C, D)$ with $F = \{ A \rightarrow B, B \rightarrow C\}$
  $R_1(A,B)$ $R_2(A,C,D)$
  In $R_2(A,C,D)$ above let's try a subset $\alpha = \{A\}$; $X = R_2 - \{A\} = \{CD\}$
  Compute $A+ = \{ABC\}$
  - A+ contains SOME (C) but not ALL attribute of X
    Thus R_2 violates BCNF (Test fails). No need to test other subsets of R_2.
- When a test fails, the FD that violates BCNF is: $\alpha \rightarrow (\alpha+ - \alpha) \cap R_i$
  in our example: $A \rightarrow [\{ABC\}-\{A\}=\{BC\}] \cap (ACD)=C$ or $A \rightarrow C$
  This is the FD deduced in F+ by transitivity and we discovered it without having to compute F+

3NF Motivation

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
j_4	l_2	k_2

J	L
j_1	l_1
j_2	l_1
j_3	l_1
j_4	l_2

L	K
l_1	k_1
l_2	k_2

Remember our goal is to remove redundancy

Relation JLK has $F = \{JK \rightarrow L, L \rightarrow K\}$
$L \rightarrow K$ violates BCNF for L+={LK} ← not SK
Decompose to (LK) & (JL) both in BCNF

- **The BCNF decomposition may not preserve dependencies**
- **On updates it is not efficient checking for FD violation across relations**
  - ***Checking JK → L requires first a join of JL & LK***
- **Solution: use the weaker 3NF**

3NF Relations

- **A relation schema *R* is in third normal form (3NF) if *for each* functional dependency $\alpha \rightarrow \beta$ in F^+ at least one of the following holds:**
  - **i. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)**
  - **ii. α is a superkey for *R***
  - **iii. each attribute *A* in $\beta - \alpha$ is contained in some candidate key for *R***

  ***(NOTE:* each attribute in $\beta - \alpha$ may be in a different candidate key)**

- **If a relation is in BCNF it is in 3NF**
  - **Because condition (ii) above holds**
- **Property (iii) is a relaxation of BCNF that allows dependency preservation**

3NF (Cont.)

- **Example**

R (

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
j_4	l_2	k_2

)

$F = \{JK \rightarrow L, L \rightarrow K\}$

- R is not in BCNF
  - L+=LK not a superkey
- Is R in 3NF?
- L is not a superkey therefore check property (iii) attributes in $\beta - \alpha$ *are contained in a candidate key*
- Find candidate keys:

  JK+=JKL *JK* is a superkey

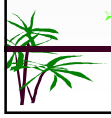
  JL+=JLK JL is a superkey
- *{K}-{L}={K} in the candidate key JK.* *Thus R is a 3NF*

Testing for 3NF

- Need to check only FDs in *F*, no need for F^+.
- Use attribute closure to check, for each dependency $\alpha \rightarrow \beta$, to find out whether α is a superkey.
- If α is not a superkey, we have to verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R*
  - this test requires finding **all** candidate keys
  - testing for 3NF has been shown to be NP-hard

Testing for BCNF & 3NF

- R(A,B,C,D) A → B C → D B → C
  - BCNF? C+={CD} not a SK (violation)
  - 3NF? For C➔D we need to find all candidate keys
    Since A is not in any RHS it has to be in every key
    A+={ABCD} superkey thus A is a candidate key
    Since B,C, and D are not A is the only candidate key. Any combination wold have to include A and would not be a candidate key
    {D}-{C}={D} is not in any candidate key. Therefore R is not 3NF
- M(T,F,S,A) T→ F T→ S TA → S
  - BCNF? T+={FS} not a SK (violation)
  - 3NF?
  - Since T+ is not a SK we have to test if {F}-{T}=F is in a candidate key.
  - To do so, we need to generate all candidate keys: T and A are not in any RHS they have to be in every key: TA+={TAFS} - this is the only candidate key
  - Since F is in the candidate key, no violation of 3NF
  - However, for T→ S, {S}-{T}={S} and S is NOT in a candidate key. *Thus R is not 3NF*



BCNF Decomposition Algorithm

- BCNF is a desirable property (avoids redundancy & update anomalies)
- If a relation is not in BCNF, we decompose it to obtain BCNF relations

```
result := {R};
done := false;
compute F+;
while (not done) do
   if (there is a schema Ri in result that is not in BCNF)
      then begin
            let α → β be a nontrivial FD in Ri
                such that α → Ri and α+ is not a SK;
                result := (result – Ri) ∪ (Ri – β) ∪ (α, β);
      end
      else done := true;
```

By construction, each relation (α, β) added to the result guarantees it is a lossles decomposition

Example of BCNF Decomposition

- R(A,B,C,D) A → B C → D B → C
  - BCNF?
    A → B does not violate BCNF because A+={ABCD} is a SK
    C → D violates BCNF because C+={CD} is not a SK
  - Decompose R using C → D into: R1(CD) & R2(CAB)
    R1 is binary- always in BCNF

  Is R2 in BCNF? Apply the test without F+
  Let α = A, α+=A+={ABCD} and X=R2-α={ABC}-A={BC}
  α+ contains ALL attributes of X –No violation
  Let α = B, α+=B+={BCD} and X=R2-α={ABC}-B={AC}
  α+ contains SOME attributes of X not ALL – BCNF violation
  what's the violating FD? α → (α+ - α)∩ R2 or B → [{BCD}-{B}]∩R2 ={CD}∩ABC}={C}
  B →C
  - Decompose R2 using B →C into R21(BC) & R22(BA)
    both binary thus in BCNF

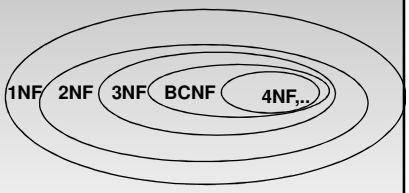
- The decomposition of R into BCNF is: R1(CD) R21(BC) R22(BA)
- It is dependency preserving – no inter-relational constraints

3NF Decomposition and Comparison of Algorithms

- There is a complicated algorithm for obtaining 3NF
  - Requires generation of F+
  - We will not cover it
- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

The Normal Forms

- 1NF: every attribute has an atomic value (not a set value)
- 2NF: we will not be concerned in this course
- 3NF: if for each FD X→Y either
  - it is trivial or
  - X is a superkey
  - Y-X is a proper subset of a candidate key
- BCNF: if for each FD X→Y either
  - it is trivial or
  - X is a superkey
- 4NF,…: we are not concerned in this course.



Goals of Normalization Using FDs

Transform relations is in "desirable" normal form: BCNF or 3NF
If a relation R is not such, decompose R into a set of relations
$\{R_1, R_2, ..., R_n\}$ such that

- The decomposition is lossless
- Relations R_i *are* in either BCNF or 3NF
- Preferably the decomposition preserves dependency

BCNF and Over-normalization

- We have to look at the meaning too
- 3NF relation has redudancy anomalies: TEACH(student,teacher,subject)
  - insertion: cannot insert a teacher until we had a student taking his subject
  - deletion: if I delete the last student of a teacher, then I loose the subject he teaches
- What is really the problem? schema *overload.* We are trying to capture two meanings:
  1. subject X is (or can be) taught by teacher Y
  2. student Z takes subject W from teacher V
- it makes no sense to say we loose the subject he teaches when he does not have a student! Who does he teach to?
- normalizing it to BCNF cannot preserve dependencies. Therefore, it is better to stay with the 3NF TEACH and another relation SUBJECT_TAUGHT:

TEACH(student,teacher,subject) **3NF**

SUBJECT-TAUGHT(teacher,subject) **BCNF**

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization (most are in BCNF)
- In a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example:
    employee(ename, *department_name , building*)
    and
    department_name→ building
  - Good design would have made department a separate entity

Denormalization for Performance

- After all the trouble to normalize, we may go to a non-normalized schema for performance

Example:

frequently displaying (*prereqs, course_id, title)*

requires to compute *course*⋈ *prereq*

- Alternative 1: Use denormalized relations to get
  - faster lookup, but
  - pay extra space and execution time for updates
  - extra coding for programmer (possibility of error in extra code)
- Alternative 2: use a materialized view defined as: *course* ⋈ *prereq*
  - faster lookup, but
  - pay extra space and execution time for updates
  - no extra coding

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

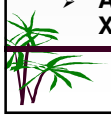
  earnings_2004 (*company_id, year, amount)*
  earnings_2005 (*company_id, year, amount)*
  earnings_2006 (*company_id, year, amount)*
  - BCNF, but
  - querying across years difficult
  - needs new table each year
- Another bad database design

  company_year (*company_id, earnings_2004, earnings_2005, earnings_2006*)
  - Also BCNF, but
  - querying across years difficult
  - requires new attribute each year

  This is a crosstab, where values of an attribute become column names - Used in spreadsheets, and in data analysis tools

Modeling Temporal Data

- Temporal data have an association time interval during which the data are *valid*
- A snapshot is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like
  ID → *street, city*
  does not hold, because the address varies over time
- A temporal functional dependency X → Y holds on schema *R* if the FD X → Y holds on all snapshots for all legal instances r (*R*).



Modeling Temporal Data (Cont.)

- Database designers may add
  - start time
  - end time

  attributes to relations

  E.g., *course*(*course_id, course_title, start_time, end_time*)
  - Constraint: no two tuples can have overlapping valid times
  - Hard to enforce efficiently
- Foreign key references may be
  - to current version of data or
  - to data at a point in time
  - E.g., in enroll the foreign keys to StudentID, CourseID, etc. reference StudentID and CourseID at the time the course was taken. ■