Relational Database Design

- Finding database schemas with good properties
- Example:
  - Database for information on suppliers, parts supplied, and shipments
  - Information consists of:
    - S#: supplier number
    - SNAME: supplier name
    - SCITY: supplier city
    - P#: part number
    - PNAME: part name
    - PCITY: city where part is stored
    - QTY: quantity of shipment

The following FDs hold:
S# $\rightarrow$ SNAME SCITY
P# $\rightarrow$ PNAME PCITY
S#P# $\rightarrow$ QTY

Relational Database Design

- One possible database schema
  - BAD[S#, SNAME, SCITY, P#, PNAME, PCITY, QTY]
- Why BAD is bad:
  - redundancy
    - SCITY is determined by S#

    However, the same SCITY could appear many times with the same S# in this relation
    - Functional dependency: S# $\rightarrow$ SCITY
  - update anomalies
    - SCITY could be changed in one place but not in another, for the same S#, resulting in inconsistency

Relational Database Design

- Why BAD is bad:
  - insertion anomalies
    - cannot record supplier info unless that supplier supplies a part
  - deletion anomalies
    - by deleting parts we can lose supplier info
- Solution
  - New schema:

S[S#, SNAME, SCITY]	Key: S#
P[P#, PNAME, PCITY]	Key: P#
SP[S#, P#, QTY]	Key: S# P#

    - No other functional dependencies besides key dependencies
    - Normal forms: “nice” forms for schemas

Decomposing Relation Schemes

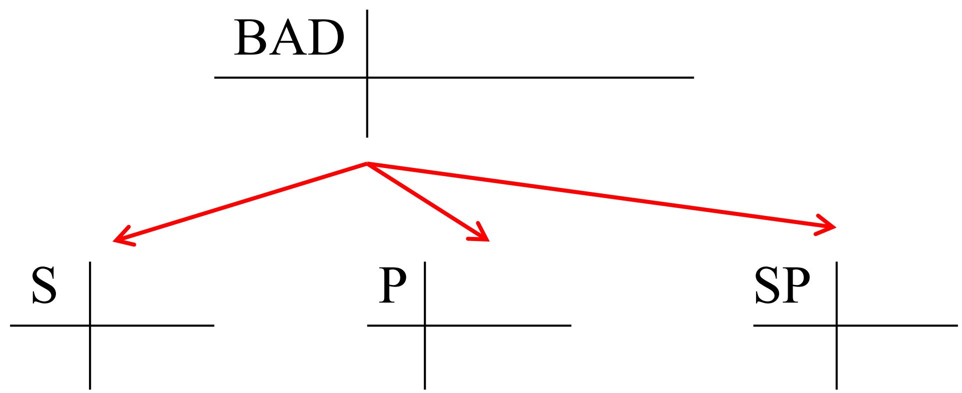
- Let R be a relation scheme
- A decomposition of R is a set $\rho = \{R_1, \ldots, R_k\}$ of relation schemas such that:

$$\mathbf{att(R) = \cup^{k}_{i = 1}\ att(R_i)}$$

- Example
  - {S, P, SP} is a decomposition of the relation schema BAD (see earlier example)

Conditions for a “Reasonable” Decomposition

- Do S, P, SP contain the same information as BAD?



$$BAD = \pi_S(BAD) \bowtie \pi_P(BAD) \bowtie \pi_{SP}(BAD) \ ?$$

Lossless join property

Conditions for a "Reasonable" Decomposition

Question: what is the connection between BAD and $\pi_S(BAD) \bowtie \pi_P(BAD) \bowtie \pi_{SP}(BAD)$?

A: no connection

B: $\pi_S(BAD) \bowtie \pi_P(BAD) \bowtie \pi_{SP}(BAD) \subseteq BAD$

C: $BAD \subseteq \pi_S(BAD) \bowtie \pi_P(BAD) \bowtie \pi_{SP}(BAD)$

Conditions for a “Reasonable” Decomposition

- It is always true that

$$\text{BAD} \subseteq \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD})$$

- The converse inclusion holds only because of the FDs

S# $\rightarrow$ SNAME SCITY
P# $\rightarrow$ PNAME PCITY
S#P# $\rightarrow$ QTY

Conditions for a “Reasonable” Decomposition

- Can the dependencies for BAD be enforced by “local” dependencies on S, P, SP?
  - The FDs for BAD:

    $S\# \rightarrow$ SNAME SCITY
    $P\# \rightarrow$ PNAME PCITY
    $S\#P\# \rightarrow$ QTY

  - Local FDs for S: $S\# \rightarrow$ SNAME SCITY
  - Local FDs for P: $P\# \rightarrow$ PNAME PCITY
  - Local FDs for SP: $S\#P\# \rightarrow$ QTY

  So nothing is lost: the local FDs can enforce the original FDs

Not always that simple.

Example: R | A B C $\quad A \rightarrow B, A \rightarrow C, B \rightarrow C$

P | A B $\qquad$ Q | B C

$A \rightarrow B \qquad B \rightarrow C$

Local FDs: $A \rightarrow B$, $B \rightarrow C$ not the same as original FDs

Enough to enforce original FDs because $A \rightarrow B$, $B \rightarrow C$ imply $A \rightarrow C$

The decomposition is dependency preserving

Lossless Join

- Let R be a relation schema and F a set of FDs over R. A decomposition $\rho=\{R_1, \ldots, R_k\}$ of R has lossless join wrt F iff, for every relation R satisfying F,

$$R = \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rk}(R)$$

- Checking lossless join:
  Idea: minimize the conjunctive query

$$\pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rk}(R)$$

  knowing that R satisfies the FDs F.
  The result must be query returning just R itself.

Testing Lossless Join

Example: R = BAD $\rho = \{S, P, SP\}$

SQL query for $\pi_S(BAD) \bowtie \pi_P(BAD) \bowtie \pi_{SP}(BAD)$:

```
SELECT  s.S#, s.SNAME, s.SCITY,
        p.P#, p.PNAME, p.PCITY, sp.QTY
FROM    BAD s, BAD p, BAD sp
WHERE   s.S# = sp.S# and p.P# = sp.P#
```

Testing Lossless Join

```
SELECT  s.S#, s.SNAME, s.SCITY,
        p.P#, p.PNAME, p.PCITY, sp.QTY
FROM    BAD s, BAD p, BAD sp
WHERE   s.S# = sp.S# and p.P# = sp.P#
```

Corresponding pattern:

	S#	SNAME	SCITY	P#	PNAME	PCITY	QTY
s	a_1	a_2	a_3	--	--	--	--
p	--	--	--	a_4	a_5	a_6	--
sp	a_1	--	--	a_4	--	--	a_7
answer	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Testing Lossless Join

1. Compute pattern for $Q = \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rk}(R)$
2. Compute $\mathrm{CHASE}_F(Q)$
3. Minimized $\mathrm{CHASE}_F(Q)$ must be R

Example: R = BAD $\rho = \{S, P, SP\}$

Pattern for select * from BAD:

S#	SNAME	SCITY	P#	PNAME	PCITY	QTY	
a_1	a_2	a_3	a_4	a_5	a_6	a_7	
a_1	a_2	a_3	a_4	a_5	a_6	a_7	$\leftarrow$ answer

Testing Lossless Join

Example: R = BAD ρ = {S, P, SP}

F = {S#→SCITY SNAME
P#→PNAME PCITY
P#S# →QTY}

Pattern Q

S#	SNAME	SCITY	P#	PNAME	PCITY	QTY
a_1	a_2	a_3	--	--	--	--
--	--	--	a_4	a_5	a_6	--
a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1	a_2	a_3	a_4	a_5	a_6	a_7

← answer

$CHASE_F(Q)$ makes third row equal to $<a_1, \ldots, a_7>$ so the minimized tableau is the previous pattern for BAD

Testing Lossless Join

Example: R = BAD $\rho = \{S, P, SP\}$

$F = \{S\# \rightarrow SCITY\ SNAME$
$P\# \rightarrow PNAME\ PCITY$
$P\#S\# \rightarrow QTY\}$

Pattern Q

S#	SNAME	SCITY	P#	PNAME	PCITY	QTY
a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1	a_2	a_3	a_4	a_5	a_6	a_7

answer

So BAD has lossless join with respect to {S, P, SP}

Testing Lossless Join

More concise algorithm

Input: R, decomposition $\{R_1, \ldots, R_k\}$, set F of FDs

1. construct a pattern using variable *a* for each attribute A:

> for each R_i the pattern has one row with variable *a* for each attribute A of R_i and wildcards everywhere else

3. Chase the pattern with F
4. Output YES iff the resulting pattern has an entire row of variables

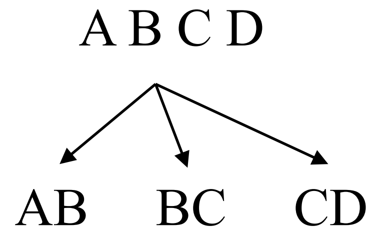
Example

S: S# SNAME SCITY
P: P# PNAME PCITY
SP: S# P# QTY

F = {S#→SCITY SNAME
P#→PNAME PCITY
P#S# →QTY}

	S#	SNAME	SCITY	P#	PNAME	PCITY	QTY
S	a_1	a_2	a_3	--	--	--	--
P	--	--	--	a_4	a_5	a_6	--
SP	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Another example



$B \rightarrow A$

$C \rightarrow B$

Towards dependency preservation: FD implication

- Given FDs can imply additional FDs

Example: $A \to B$ and $B \to C$ imply $A \to C$

Every relation that satisfies $A \to B$ and $B \to C$ also satisfies $A \to C$

Another example:

$A \to C$, $BC \to D$, $AD \to E$ imply $AB \to E$

Definition: A set F of FDs implies another FD $X \rightarrow Y$ if every relation satisfying F also satisfies $X \rightarrow Y$

Notation for "F implies $X \rightarrow A$": $F \vDash X \rightarrow A$

Everything that F implies:
$F^+ = \{X \rightarrow Y \mid F \vDash X \rightarrow Y\}$

Example: $\{A \rightarrow B,\ B \rightarrow C\}^+$ includes the following FDs:
$A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C,\ AB \rightarrow C$

also "trivial" FDs (that are always true):
$A \rightarrow A,\ AB \rightarrow A,\ ABC \rightarrow A,\ B \rightarrow B,\ AB \rightarrow B$, etc

Checking if $X \to A$ is implied by a set F of FDs

Needed to compute keys, check dependency preservation, check satisfaction of normal forms, etc.

Useful to think of the closure of X: the set of attributes "determined" by X

Definition: The closure of a set of attributes X with respect to a set F of FDs is

$$X^+ = \{A \mid F \models X \to A\}$$

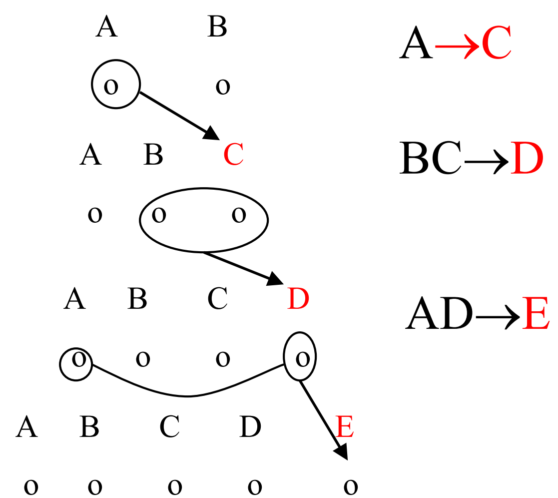
By the definition, $X \to A \in F^+$ iff $A \in X^+$

Example of Closure Computation

- $R = ABCDEF$
- $F = \{A \rightarrow C,\ BC \rightarrow D\ AD \rightarrow E\}$
- $X = AB$

Computing X^+ :

- $X^{(0)} = AB$
- $X^{(1)} = ABC$
- $X^{(2)} = ABCD$
- $X^{(3)} = ABCDE$
- $X^{(4)} = X^{(3)}$
- $X^+ = ABCDE$



To check if X is a superkey in R: $X^+ = att(R)$

Computing the closure of a set of attributes wrt a set of FD's

- Let F be a set of FD's over R, and $X \subseteq R$. The following computes the closure X^+ of X wrt F
  1. $X^{(0)} \leftarrow X$
  2. $X^{(i+1)} \leftarrow X^{(i)} \cup \bigcup\{Z \mid V \rightarrow Z \in F, V \subseteq X^{(i)}\}$

  We get: $X^{(0)} \subseteq X^{(1)} \subseteq \ldots \subseteq X^{(i)} \subseteq X^{(i+1)} \subseteq \ldots \subseteq att(R)$

  Since att(R) is finite, there must exist a $k \geq 0$ such that $X^{(k)} = X^{(k+1)}$
  Then, $X^+ = X^{(k)}$

Dependency Preserving Decompositions

- "Local" FDs

> if F is a set of FD's over R and $X \subseteq$ attributes(R) then
> $\pi_X(F^+) = \{V \rightarrow W \in F^+ \mid V \subseteq X \text{ and } W \subseteq X\}$

These are the FDs implied by F
that apply to the set of attributes X ("local" to X)

Dependency Preserving Decompositions

<u>Example:</u> $F = \{A \rightarrow C, BC \rightarrow D, AD \rightarrow E\}$

The FDs in F^+ that are

- <u>local to AC</u>: $A \rightarrow C$ $A^+ = AC$ $C^+ = C$

- <u>local to ABD</u>: $AB \rightarrow D$

  $A^+ = AC$ $B^+ = B$ $D^+ = D$

  $AB^+ = ABCDE$ $AD^+ = ADCE$ $BD^+ = BD$

- <u>local to ABCE</u>: $A \rightarrow C$ $AB \rightarrow CE$ $AE \rightarrow C$

  $ABC \rightarrow E$ $ABE \rightarrow C$

  $A^+ = AC$ $B^+ = B$ $C^+ = C$ $E^+ = E$

  $AB^+ = ABCDE$ $AC^+ = AC$ $AE^+ = AEC$

  $BC^+ = BCD$ $BE^+ = BE$ $CE^+ = CE$

  $ABC^+ = ABCDE$ $ABE^+ = ABECD$ $BCE^+ = BCED$

Dependency Preserving Decompositions

- Definition:
  - Let $\rho = (R_1, \ldots, R_k)$ be a decomposition for R and F a set of FD's over R. Then ρ preserves F iff

The set of local FDs $\cup^k_{i=1}\pi_{Ri}(F^+)$ is equivalent to F

In other words, all FDs in F are implied by the local FDs

Testing Preservation of Dependencies

- Naïve method:
  1. Compute F^+
  2. Compute $G = \cup^k_{i=1} \pi_{Ri}(F^+)$ `%local fds`
  3. Check that $F \subseteq G^+$

  Drawback: impractical, since the size of F^+ can be exponential in the size of F.

- Improved method:

  avoid computing all of G

  idea: $X \rightarrow A$ is local to R_i iff $AX \subseteq att(R_i)$ and $A \in X^+$

Example

R = {A B C D} ρ = {AB, BC, CD}

F = {A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D, D $\rightarrow$ A}

Does ρ preserve D $\rightarrow$ A? (Is it implied by the local FDs ?)

Decomposition: $\overset{*}{A}\overset{*}{B}$ $\overset{*}{B}\overset{*}{C}$ $\overset{*}{C}\overset{*}{D}$

Start with D

D^+ = DABC so D $\rightarrow$ C is local to CD, add C

C^+ = CDAB so C $\rightarrow$ B is local to BC, add B

B^+ = BCDA so B $\rightarrow$ A is local to AB, add A $\leftarrow$ success !

D $\rightarrow$ A is implied by local FDs

Testing Preservation of Dependencies

- Naïve method:
  1. Compute F^+
  2. Compute $G = \cup^k_{i=1} \pi_{Ri}(F^+)$ `%local fds`
  3. Check that $F \subseteq G^+$

  Drawback: impractical, since the size of F^+ can be exponential in the size of F.

- Improved method:

  for each $X \rightarrow Y$ in F do
      $Z := X$
      while changes occur in Z do
        for $i := 1$ to k do
        $Z := Z \cup ((Z \cap R_i)^+ \cap R_i)$
      if $Y \not\subset Z$ output "no" and stop
  output "yes"

  This computes the closure of Z wrt G

  (+ is wrt F)

Full example

$R = \{A\ B\ C\ D\}$ $\rho = \{AB, BC, CD\}$

$F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

Does ρ preserve F?

Clearly, ρ preserves $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$ (they are local FDs).

Does ρ preserve $D \rightarrow A$?

Decomposition: $\overset{*}{A}\overset{*}{B}$ $\overset{*}{B}\overset{*}{C}$ $\overset{*}{C}\overset{*}{D}$

Start with D

$D^+ = DABC$ so $D \rightarrow C$ is local to CD, add C

$C^+ = CDAB$ so $C \rightarrow B$ is local to BC, add B

$B^+ = BCDA$ so $B \rightarrow A$ is local to AB, add A ← success !

$D \rightarrow A$ is preserved

Another example

$$Att(R) = ABCDE$$

$$\rho = \{ABC, ADE, CE\}$$

$$F = \{AB \rightarrow C, C \rightarrow E, E \rightarrow C, C \rightarrow D, AB \rightarrow E\}$$

Normal Forms

- Terminology recall:

Let R be a relation schema and F a set of fd's over attributes(R).

- Superkey: $X \subseteq att(R)$ such that $X \rightarrow att(R) \in F^+$.

  X determines all attributes of R

- Key: $X \subseteq att(R)$ such that $X \rightarrow att(R) \in F^+$
  and there is no $Y \subset X$ such that $Y \rightarrow att(R) \in F^+$.

  X is a minimal superkey

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

Superkeys: A, AB, AC, ABC

Key: A (the only one)

Purpose of normal forms

- Eliminate problems of redundancy and anomalies.
- Normal forms:
  - Boyce-Codd, third, fourth, ….
- <u>Boyce-Codd Normal Form</u> (BCNF)

A relation scheme R is in BCNF wrt a set of FD's F over R, iff whenever $X \rightarrow A \in F^+$ and $A \notin X$, X is a superkey for R

The only nontrivial FDs are those induced by keys

Example

- BAD(S#, P#, SNAME, PNAME, SCITY, PCITY, QTY)
  - not in BCNF wrt

    F = S#→SNAME SCITY
    P#→PNAME PCITY
    S# P#→QTY

  - S(S#, SCITY, SNAME) is in BCNF wrt S#→SNAME SCITY
  - P(P#, PCITY, PNAME) is in BCNF wrt P#→PNAME PCITY
  - SP(S# P# QTY) is in BCNF wrt S# P# → QTY

Decomposition of a relation schema into BCNF relation schemas, with lossless join

- Every relation scheme has a decomposition into BCNF relation schemes, with lossless join

  not always dependency preserving

- Example

  R = CITY ZIP ST

  F = CITY ST $\rightarrow$ ZIP, ZIP $\rightarrow$ CITY

  – R has no decomposition into BCNF schemas, which preserves F

  (CITY ST $\rightarrow$ ZIP is never preserved)

Algorithm to obtain a lossless join decomposition of a given R wrt F

<u>Basic step (example):</u>

Suppose S = ABC only local FD: $A \rightarrow B$

This violates BCNF because A is not a superkey in S

Decomposition step to eliminate violation:

ABC

AB AC

This has lossless join:
apply the test using the FD $A \rightarrow B$

A	B	C
a	b	-
a	-	c
a	b	c

Algorithm to obtain a lossless join decomposition of a given R wrt F

Start with $\rho = \{R\}$

Apply recursively the following procedure:

for each S in ρ not in BCNF,

$$\left.\begin{array}{l} \text{let } X \rightarrow A \in F^+ \text{ be such that } XA \subseteq S,\ A \notin X, \\ \qquad\qquad\qquad X \text{ is not a superkey of } S \end{array}\right\} \text{violation of BCNF}$$

replace S by S_1 and S_2, where

$S_1 = XA$

$S_2 = X(S - A)$

until all relation schemes are in BCNF

Note:

$$\begin{tikzcd}[column sep=small] & S \arrow[dl, dash] \arrow[dr, dash] & \\ S_1 & & S_2 \end{tikzcd}$$

this decomposition has lossless join:

$X \rightarrow A \in F^+$

X is a key for S_1

Example

R = C T H R S G

- C = course
- T = teacher
- H = hour
- R = room
- S = student
- G = grade

F: $C \rightarrow T$

$HR \rightarrow C$

$HT \rightarrow R$

$CS \rightarrow G$

$HS \rightarrow R$

What are the keys?

only key: HS

Example

FDs: C→T, CS →G, HR →C, HS →R, TH →R

CTHRSG

violation: CS → G   (CS)+ = CSTG

CSG

no violation: C+ = CT,
S+ = S, G+ = G

CTHRS

violation: C → T   C+ = CT

CT

no violation: 2 attributes

CHRS

violation: CH → R   (CH)+ = CHTR

CHR

CHS

no violation: C+ = CT, H+ = H, R+ = R, S+ = S

Resulting decomposition: CSG, CT, CHR, CHS

Remarks (drawbacks)

- Decomposition is not unique
  - CHRS could be decomposed into CHR and CHS <u>or</u> CHR and RHS
- The decomposition does not preserve TH→R

run dependency preservation algorithm

F = C→T CS →G, HR →C, HS →R, TH →R

for the previous BCNF decomposition:

CSG   CT   CHR   CHS

the local closure of TH is TH

so TH →R is not preserved

- Is there an efficient algorithm for BCNF decomposition?
  – Most likely NO:

It is NP-complete to decide whether a relation R is in BCNF with respect to a set of FDs.

$$\Downarrow$$

Any algorithm for BCNF is most likely exponential

- Problem with BCNF:
  - Not every relation schema can be decomposed into BCNF relation schemas which have lossless join *and* preserve the dependencies.

- <u>Third Normal Form (3NF)</u>
  - A relation scheme R is in Third Normal Form wrt a set F of fd's over R, if whenever $X \rightarrow A$ holds in R and $A \notin X$ then either X is a superkey <u>or A is prime</u>

$A \in att(R)$ is prime: $A \in K$ for some key K

3NF is weaker than BCNF

Decomposition of a relation schema into BCNF relation schemas, with lossless join

- Example

  R = CITY ZIP ST

  F = CITY ST $\rightarrow$ ZIP, ZIP $\rightarrow$ CITY

R is in 3NF but not in BCNF

Violation of BCNF: ZIP $\rightarrow$ CITY

However, CITY belongs to the key CITY ST so this satisfies 3NF

Computing a 3NF decomposition

Three main steps:

1. Simplify the set of FDs (eliminate redundancies)
2. Construct a first-cut decomposition from remaining FDs

   e.g.: from A $\rightarrow$ B make a relation AB

   dependency preserving, not always lossless join
3. If needed, modify decomposition to ensure lossless join

Eliminating redundancies in the FDs

1. Rewrite the FDs with single attributes on RHS

   ex: replace $AB \rightarrow CD$ with $AB \rightarrow C$ and $AB \rightarrow D$

2. Eliminate redundant FDs

   ex: $F = \{A \rightarrow C,\ A \rightarrow B,\ B \rightarrow C\}$

   $A \rightarrow C$ is redundant (it is implied by $A \rightarrow B$ and $B \rightarrow C$)

3. Eliminate redundant attributes from LHS of FDs

   ex: $F = \{A \rightarrow B,\ AB \rightarrow C\}$

   B is redundant in $AB \rightarrow C$ because A by itself determines C ($A \rightarrow C$ is implied by F)

Example minimization

Attributes: ABCDEI
FDs: A $\rightarrow$ C, AB $\rightarrow$ C, C $\rightarrow$ DI, CD $\rightarrow$ I, EC $\rightarrow$ AB, EI $\rightarrow$ C

First-cut 3NF decomposition

R a schema

F <u>minimal</u> set of FD's over R (no redundancies)

$\rho = \{XA_1 \ldots A_m \mid X \to A_i \in F\} \cup \{B \in R \mid B \text{ does not occur in } F\}$

- Example
  - R = CTHRSG (see earlier example)
  - F = $C \to T$ $CS \to G$
    $HR \to C$ $HS \to R$ (minimal)
    $HT \to R$
  - Then ρ = {CT, HRC, HTR, CSG, HRS}

First-cut 3NF decomposition

- <u>Theorem</u>. ρ preserves F and each R_i in ρ is in 3NF wrt $\pi_{Ri}(F^+)$.

Why is each relation in 3NF?

Proof idea: Suppose XA is constructed from X → A in F

Let Y → B be local to XA. Two cases:

1. B = A. Then Y = X and Y is a superkey (if $Y \subset X$ then attributes in X – Y are redundant)
2. $B \neq A$, so $B \in X$ and B is prime (X must be a key, otherwise X → A has redundant attributes on LHS).

Second cut: ensure lossless join

- To obtain decomposition that also has lossless join:
  - add to ρ a set K where K is a key for R

    …unless there already is a “piece” in the decomposition whose attributes form a superkey for R

- <u>Theorem</u> $\rho \cup \{K\}$ is a 3NF decomposition which is dependency preserving and has lossless join.

  Proof idea: chasing the pattern for $\rho \cup \{K\}$ produces answer variables in the entire row for K. Apply the FDs in the same order used in the computation of K^+

Example

- R: ABCDE F = $\{A \to C, BC \to D\ AD \to E\}$

  $\rho = \{AC, BCD, ADE\}$

- None of AC, BCD, ADE is a superkey:

  $AC^+ = AC$, $BCD^+ = BCD$, $ADE^+ = ADEC$

- AB is a key, add it to ρ

- Claim: $\rho \cup \{AB\}$ has lossless join (see next slide)

Test for lossless join

F = $\{A \to C, BC \to D\ AD \to E\}$

Pattern for $\{AC, BCD, ADE, AB\}$:

A	B	C	D	E
a	-	c	-	-
-	b	c	d	-
a	-	-	d	e
a	b	c	d	e

Chase with A → C, BC → D, AD → E (in order)

FDs used in computation of AB^+ (in order):

A → C, BC → D, AD → E

AB ABC ABCD ABCDE

A (simplified) overview of schema design using FDs

1. Choose attributes in R

2. Specify dependencies F
   (tool: "Armstrong relations" – relations satisfying exactly F^+)

3. Find a lossless-join, dependency preserving decomposition into Normal Form schemas
   – BCNF, if possible
   – 3NF, if not BCNF

Attributes vs. data

Example: keep information about employee names and the departments in which they work
Departments: PCs, Toys, Candy
Two options:

1

Emp	name	dept

2

Emp	name	PCs	Toys	Candy

Some criteria:

- Are departments stable ?
- Sparse vs. dense tables
- Ease of querying

Find the departments in which Joe works

A: easier with 1
B: easier with 2

Find the employees who work in every department

A: easier with 1
B: easier with 2

Armstrong relation example

Schedule	theater	title

$F = \{theater \rightarrow title\}$

Armstrong relation for F: satisfies exactly the FDs in F^+

Schedule	theater	title
	Paloma	Casablanca
	Hillcrest	Casablanca

- satisfies theater → title
- violates title → theater

Full normalization example

Attributes: broker, office, investor, stock, quantity, dividend

B O I S Q D

FDs: $S \rightarrow D$, $I \rightarrow B$, $IS \rightarrow Q$, $B \rightarrow O$

Attributes: broker, office, investor, stock, quantity, dividend

B O I S Q D

FDs: $S \rightarrow D$, $I \rightarrow B$, $IS \rightarrow Q$, $B \rightarrow O$

Attributes: broker, office, investor, stock, quantity, dividend

B O I S Q D

FDs: $S \rightarrow D$, $I \rightarrow B$, $IS \rightarrow Q$, $B \rightarrow O$

A glimpse beyond FDs

- Example

Movie	title	director	actor

  - Suppose each movie may have several directors and several actors
  - then there are no fds (so this is in BCNF)
  - redundancy still exists

Directors and actors are independent info for same title

Better design:

Directors	title	director

Actors	title	actor

Captured by "multi-valued dependencies"

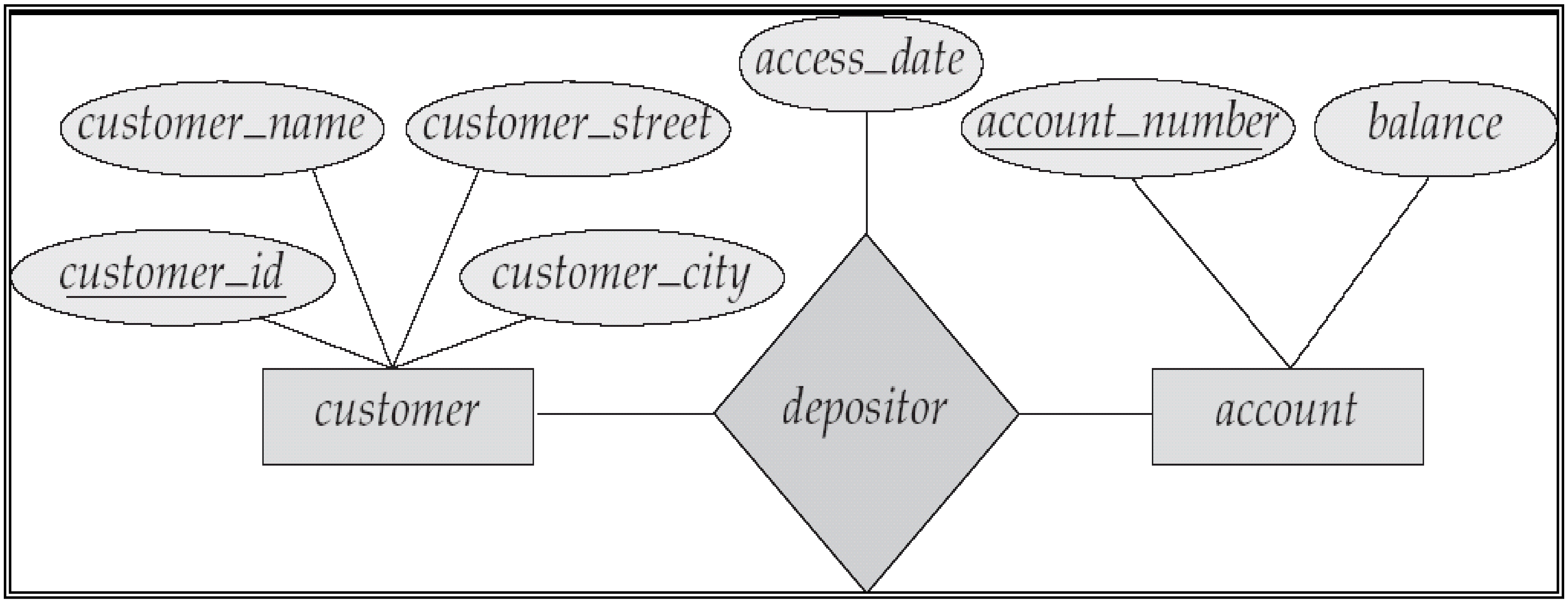
Directors and actors are independent info for same title

Taking into account MVDs results in an extension of BCNF called 4th Normal Form

Alternative approach

- Start with an Entity-Relationship (ER) diagram
- Translate into corresponding relational schema
- Identify functional dependencies
- Use design theory to further improve and bring to BCNF or 3NF or 4NF

Example: piece of an E-R diagram



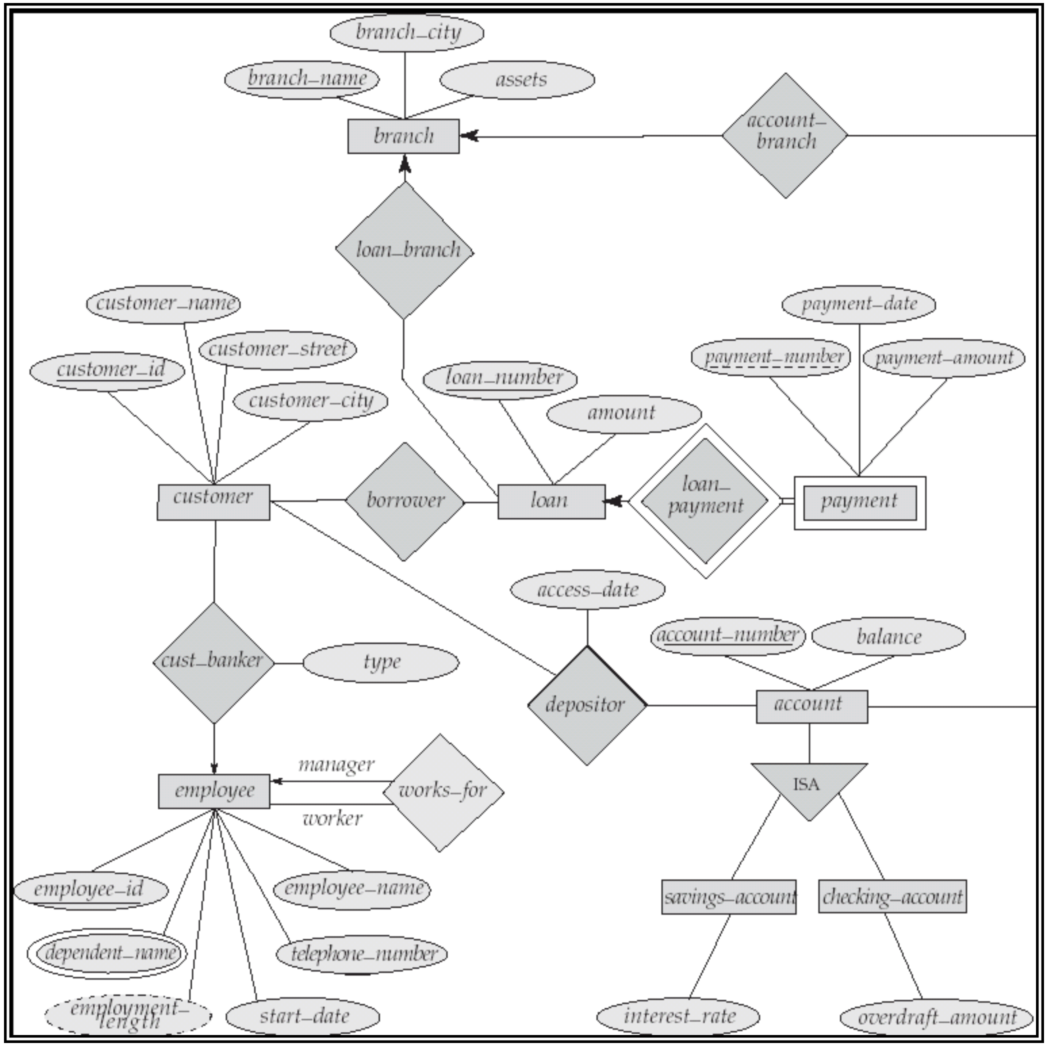
Corresponding relational schema

customer	customer-id customer-name customer-zip customer-city

account	account-number balance

depositor	customer-id account-number access date

A more realistic E-R diagram



First-cut relational schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*. *employee_name*, *telephone_number*, *start_date*)
- *dependent_name* = (*employee_id*, *dname*)
- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *borrower* = (*customer_id*, *loan_number*)
- *depositor* = (*customer_id*, *account_number*)
- *cust_banker* = (*customer_id*, *employee_id*, *type*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)
- *payment* = (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)
- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)