Fall 2013

SCHEMA REFINEMENT AND NORMAL FORMS
[CH 19]

Database Design: The Story so Far

- Requirements Analysis
  - Data stored, operations, apps, …
- Conceptual Database Design
  - Model high-level description of the data, constraints, ER model
- Logical Database Design
  - Choose a DBMS and design a database schema
- Schema Refinement
  - Normalize relations, avoid redundancy, anomalies …
- Physical Database Design
  - Examine physical database structures like indices, restructure …
- Security Design

Normalization

What is a good relational schema? How can we improve it?

- e.g.: Suppliers (<u>name, item</u>, desc, addr, price)

  <u>Redundancy</u> Problems:

  1. A supplier supplies two items: Redundant Storage
  2. Change address of a supplier: Update Anomaly
  3. Insert a supplier: Insertion Anomaly
     - What if the supplier does not supply any items (nulls?)
     - Record desc for an item that is not supplied by any supplier
  4. Delete the only supplier tuple: Delete Anomaly
     - Use nulls?
     - Delete the last item tuple. Can't make name null. Why?

Alternative:

Dealing with Redundancy

- Identify “bad” schemas
  - functional dependencies
- Main refinement technique: decomposition
  - replacing larger relation with smaller ones
- Decomposition should be used judiciously:
  - Is there a reason to decompose a relation?
    - Normal forms: guarantees against (some) redundancy
  - Does decomposition cause any problems?
    - Lossless join
    - Dependency preservation
    - Performance (must join decomposed relations)

Functional Dependencies (FDs)

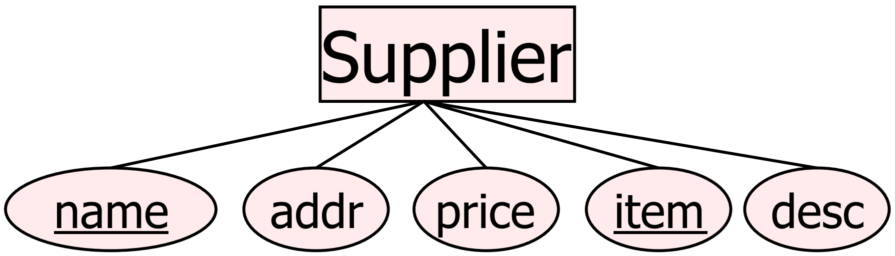
- A form of IC
- D: $X \rightarrow Y$ X and Y subsets of relation R's attributes

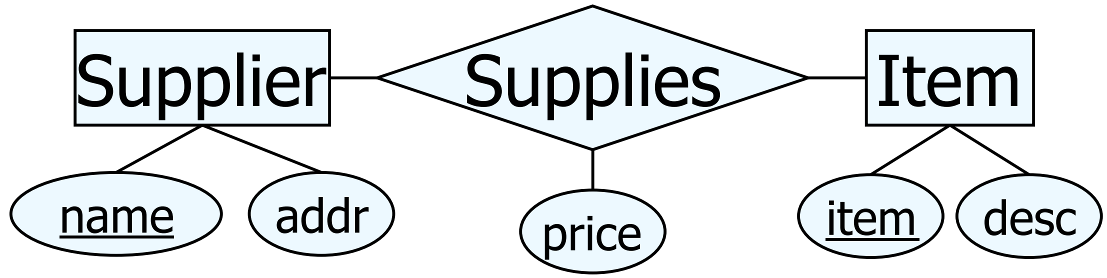
  $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2) \Rightarrow \Pi_y(t1) = \Pi_y(t2)$
- An FD is a statement about all allowable relations.
  - Based only on application semantics, can't deduce from instances
  - Can simply check if an instance violates FD (and other ICs)
- Consider, $(X,Y) \rightarrow Z$. Does this imply (X,Y) is a key?

X	Y	Z	K
1	1	11	A
1	2	12	A
2	2	22	A
2	2	22	B

Primary Key IC is a special case of FD

Example: Constraints on Entity Set





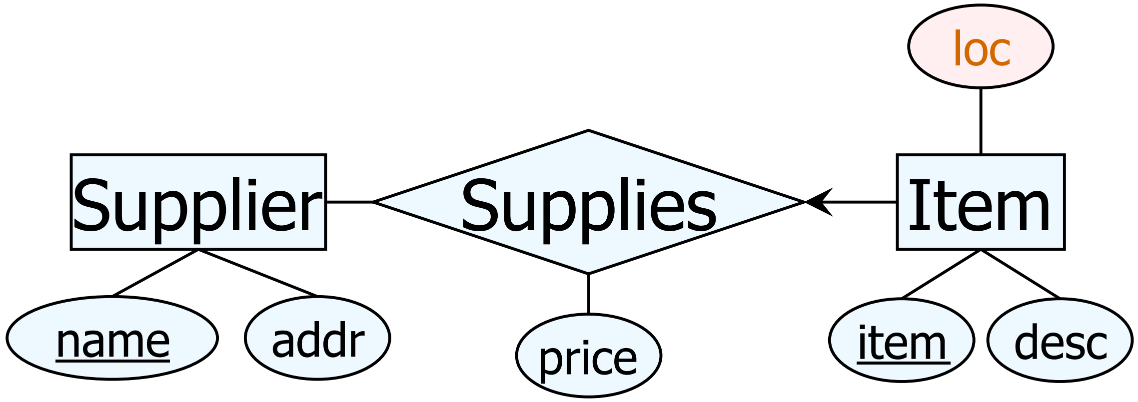
- S(name, item, desc, addr, price)
- FD: {n,i} → {n,i,d,a,p}
- FD: {n} → {a}
- FD: {i} → {d}
- Decompose to: NA, ID, INP

- Spl(name, item, price)
  - FD: {n,i} → {n, i, p}
- Sup(name, addr)
  - FD: {n} → {n, a}
- Item (item, desc)
  - FD: {i} → {i, d}

ER design is subjective and can have many E + Rs
FDs: sanity checks + deeper understanding of schema

Same situation could happen with a relationship set

Refining an ER Diagram



- IS (item, name, desc, loc, price)
  S (name, addr)
- A supplier keeps all items in the same location
  FD: name $\rightarrow$ loc
- Solution:

Inferring FD

- ename → ejob, ejob → esal; ⇒ ename → esal
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If $Y \subseteq X$, then X → Y
  - Augmentation: If X → Y, then XZ → YZ for any Z
  - Transitivity: If X → Y and Y → Z, then X → Z
- Additional rules (derivable):
  - Union: If X → Y and X → Z, then X → YZ
  - Decomposition: If X → YZ, then X → Y and X → Z
- Set of all FD = closure of F, denoted as F^+
- AA sound: only generates FD in F^+
- AA complete: repeated application generates all FD in F^+

Decomposition

- Replace a relation with two or more relations
- Problems with decomposition

1. Some queries become more expensive. (more joins)
2. **Lossless Join**: Can we reconstruct the original relation from instances of the decomposed relations?
3. **Dependency Preservation**: Checking some dependencies may require joining the instances of the decomposed relations.

Lossless Join Decompositions

- Relation R, FDs F: Decomposed to X, Y
- Lossless-Join decomposition if:

  $\Pi_X(r) \bowtie \Pi_Y(r) = r$ for **every** instance r of R
- Note, $r \subseteq \Pi_X(r) \bowtie \Pi_Y(r)$ is always true,

  not vice versa, unless the join is lossless
- Can generalize to three more relations

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Join …

- Relation R, FDs F: Decomposed to X, Y
  - Test: lossless-join w.r.t. F if and only if the closure of F contains:
    - $X \cap Y \rightarrow X$, or
    - $X \cap Y \rightarrow Y$

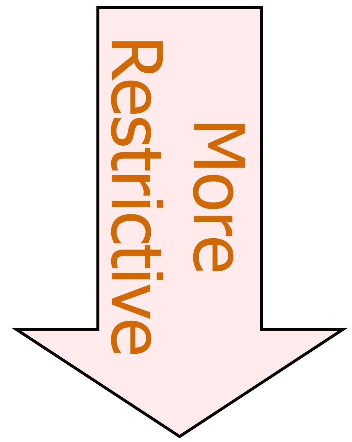
    i.e. attributes common to X and Y contain a key for either X or Y
  - Also, given FD: $X \rightarrow Y$ and $X \cap Y = \emptyset$, the decomposition into R-Y and XY is lossless
    - **X is a key in XY, and appears in both**

Dependency Preserving Decomposition

- R (sailor, boat, date) {**D → S**, D → B}
  → X (sailor, boat)
  Y (boat, date) {D → B}
- To check D → S need to join R1 and R2 (expensive)
- Dependency preserving:
  - R → X, Y $F^+ = (F_x \cup F_y)^+$
    - Note: F not necessarily = $F_x \cup F_y$

Normal Forms

- Is any refinement is needed!
- Normal Forms: guarantees that certain kinds of problems won't occur
  - 1 NF : Atomic values
  - 2 NF : Historical
  - 3 NF : …
  - BCNF : Boyce-Codd Normal Form



- Role of FDs in detecting redundancy:
  - Relation R with 3 attributes, ABC.
    - No ICs (FDs) hold ⇒ no redundancy.
    - $A \rightarrow B$ ⇒ 2 or more tuples with the same A value, redundantly have the same B value!

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for all X $\rightarrow$ A in F^+
  - A $\in$ X (trivial FD), or
  - X is a super key

i.e. all non-trivial FDs over R are key constraints.

- **No redundancy in R** (at least none that FDs detect)
- Most desirable normal form

X	Y	A
x	y1	a
x	y2	?

- Consider a relation in BCNF and FD: X $\rightarrow$ A, two tuples have the same X value
  - Can the A values be the same (i.e. redundant)?
  - **NO!** X is a key, $\Rightarrow$ y1 = y2. Not a set!

3NF

- Relation R with FDs F is in 3NF if, for all X → A in F^+
  - $A \in X$ or
  - X is a super key or
  - A is part of some key for R **(prime attribute)**
    - Minimality of a key (not superkey) is crucial!
- BCNF implies 3NF
- e.g.: Sailor (Sailor, Boat, Date, CreditCrd)
  - SBD -> SBDC, S -> C **(not 3NF)**
  - If C -> S, then CBD -> SBDC (i.e. CBD is also a key). Now in 3NF!
  - Note redundancy in (S, C); 3NF permits this
  - Compromise used when BCNF not achievable, or perf. Consideration
- Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.