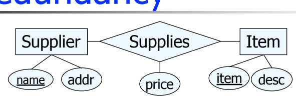
Normalization

Chapter 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

Redundancy



- What is a good relational schema?
- Suppose we did the following…

Name	Item	Desc	Addr	Price
Acme	Dynamite	boom	A2	$8
Acme	Paint	blue	A2	$10
Acme	Flowers	pink	A2	$3

Redundant storage problem: Supplier information stored once per item it supplies!

Redundancy

Name	Item	Desc	Addr	Price
Acme	Dynamite	boom	A2	$8
Acme	Paint	blue	A2	$10
Acme	Flowers	pink	A2	$3

- Redundancy Problems:
  - Redundant storage
    - A supplier supplies multiple items
  - Update anomalies
    - Change address of a supplier
  - Insertion anomalies
    - Insert a supplier (nulls?)
    - What if the supplier doesn't supply any items?
  - Deletion anomalies
    - What if we want to delete the last item tuple?

Better Schema:

S(name,addr)

SP(name,item,price)

I(item,desc)

Dealing with Redundancy

- Redundancy arises when schema forces an unnatural association among attributes
  - Resolve with functional dependencies
- Main refinement technique: decomposition
  - replacing larger relation with smaller ones
- Decomposition should be used judiciously:
  - Normal forms: guarantees against (some) redundancy

Functional Dependencies (FDs)

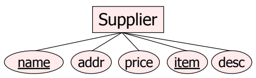
- A form of IC
- D: $X \rightarrow Y$ X and Y subsets of relation R's attributes
  $t1 \in r$, $t2 \in r$, $\prod_X(t1) = \prod_X(t2) \Rightarrow \prod_Y(t1) = \prod_Y(t2)$
- An FD is a statement about all allowable relations.
  - Based only on application semantics, can't deduce from instances
  - Can simply check if an instance violates FD (and other ICs)
- Consider, $(X,Y) \rightarrow Z$. Does this imply (X,Y) is a key?

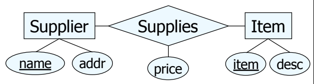
X	Y	Z	K
1	1	11	A
1	2	12	A
2	2	22	A
2	2	22	B

Primary Key IC is a special case of FD

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

Example: Constraints on Entity Set





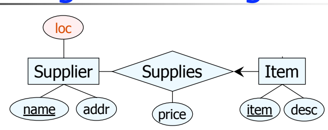
- S(<u>name, item</u>, desc, addr, price)
- FD: {n,i} → {n,i,d,a,p}
- FD: {n} → {a}
- FD: {i} → {d}
- Decompose to: <u>N</u>A, <u>I</u>D, <u>IN</u>P

- Spl(<u>name, item</u>, 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 (<u>item</u>, name, desc, loc, price)
  S (<u>name</u>, addr)
- A supplier keeps all items in the same location
  FD: name → loc
- Solution:
  IS (<u>item</u>, name, desc, price)
  Loc (<u>name</u>, addr, loc) <- New ER diagram

Inference Rules

- ename → ejob, ejob → esal; ⇒ ename → esal
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If X ⊇ Y, 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^+
- F+ obtained by repeatedly applying Armstrong's Axioms

Decomposition

- Replace a relation with two or more relations
- Problems with decomposition
  - Some queries become more expensive. (more joins)
  - Lossless Join: Can we reconstruct the original relation from instances of the decomposed relations?
  - 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:
  $\prod_X(r) \bowtie \prod_Y(r) = r$ for **every** instance r of R
- Note, $r \subseteq \prod_X(r) \bowtie \prod_Y(r)$ is always true, not vice versa, unless the join is lossless
- Can generalize to three or 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 (cont.)

- Relation R, FDs F: Decomposed to X, Y
  - Test: lossless-join w.r.t. F if and only if 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

Lossless join decomposition is always required!

Dependency Preserving Decomposition

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

Dependency preserving decomposition is desired!

Normal Forms

- Certain kind of decomposition
- Guarantees that certain problems won't occur
  - 1 NF : No set-valued attrs
  - 2 NF : Historical
  - 3 NF : …
  - BCNF : Boyce-Codd Normal Form

More Restrictive

Boyce-Codd Normal Form (BCNF)

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

  i.e. all non-trivial FDs over R are keys.
- **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 → A, two tuples have the same X value
  - Can the A values be the same (i.e. redundant)?
  - **NO!** X is a key, ⇒ y1 = y2.

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, CreditCard)
  - 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.

Exercise

- Relation R=(A,B,C,D,E)
- FDs:

  A → BC

  CD → E

  B → D

  E → A
- Is R in BCNF?
- Is R in 3NF?

Hint: Use Armstrong's Axioms

Normalization

- Bad schemas lead to redundancy
  - Redundant storage, update, insert, and delete anomaly
- To "correct" bad schemas: decompose relations
  - Must be a lossless-join decomposition
  - Would like dependency preserving decompositions
- Desired Normal Forms
  - BCNF: allow only super-key functional dependencies
  - 3NF: allow dependencies with prime attributes on the RHS
    - Allows a limited form of redundancy
    - Trades off performance (avoid joins) for redundancy

Announcements

- Optional Exercises: 19.1, 19.3, 19.5, 19.7, 19.11