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

Form/Spreadsheet

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Coffee	Kona	$8
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Coffee	Kona	$8

Problems

- (Supplier ID, item) appears to be the key, but Supplier ID is missing in many places.
- Addresses can be multi-valued
- Not a good table from database perspective

Normalization

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Coffee	Kona	$8
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Coffee	Kona	$8

The above is not a good table

- Going to proper set of tables is called "normalization"
  - avoid redundancy of data
  - capture the dependencies inherent in the data

Two approaches to Normalization

- We saw the ER modeling approach earlier. Create an ER model and then map to tables.
- Today, we will see another approach without going to the ER model.
  - State dependencies between attributes of tables
  - Map dependencies to tables. Can be done automatically!

1^{st} Normal Form - first step

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Coffee	Kona	$8
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Coffee	Kona	$8
2	Beanery	A3	Coffee	Kona	$8

- **Each value in table is single-valued.**
- **Each row contains all the relevant data**

We now have a relational table. Rows can be reordered, all rows independent

Redundancy and Errors

(Remember, we are trying to do this without using an ER)

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Coffee	Kona	$8
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Coffee	Kona	$8
2	Beanery	A3	Coffee	Kona	$8

Redundancy Problems

- Redundant storage
  - Supplier info unnecessarily stored multiple times
- Update anomalies
  - Change address of a supplier in one row leads to an inconsistency with other rows for the same supplier
- Insertion anomalies
  - Cannot easily insert a supplier without providing item info.
  - Another example: Can't add a course unless there is a student enrolled in the course
- Deletion anomalies
  - Deleting an item can cause loss of supplier info
  - Loss of info on one attribute because of deletion of other attributes. E.g., delete last item.

Dealing with Redundancy

- Redundancy arises when schema forces an unnatural association among attributes
- The new trick we will learn today is to use the notion of *functional dependencies*
- Main refinement technique: decomposition
  - replacing larger relation with smaller ones
- Decomposition should be used judiciously:
  - Normal forms: guarantees against (some) redundancy
  - But, there can be a performance hit in going to smaller tables

Functional Dependencies

FDs capture dependencies among attributes

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Coffee	Kona	$8
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Coffee	Kona	$8
2	Beanery	A3	Coffee	Kona	$8

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

FD: Definition

- Notation: a → b
- Read as: ‘a’ functionally determines ‘b’
- **Informally**: If you know ‘a’, there is only one ‘b’ to match.
- **Formally**: A form of Integrity Constraint

D: X → Y X and Y subsets of relation R’s attributes

$t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2) \Rightarrow \Pi_y(t1) = \Pi_y(t2)$

For example, (Supplier ID, item) → Price :

If the columns for Supplier ID, item are equal on some some rows, then Price column for those rows are also equal.

Functional Dependencies (FDs)

- $X \rightarrow Y$ is a form of Integrity Constraint
- Each FD is a statement about all allowable relations.
  - Based only on application semantics, not a table instance

Primary Key IC is a special case of FD

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

- Role of FDs in detecting problems. $X \rightarrow Y$.
- (X,Y) as (2, 2) twice: redundancy
- (X,Y) as (1,1) and (1,2): inconsistency

Basic Normalization

Write Keys -> attribute mappings.

One table for each

Supplier ID	Supplier Name	Item	Desc	Price
1	Acme	Coffee	Kona	$8
1	Acme	Paint	blue	$10
1	Acme	Flowers	pink	$3
2	Beanery	Coffee	Kona	$8
2	Beanery	Coffee	Kona	$8

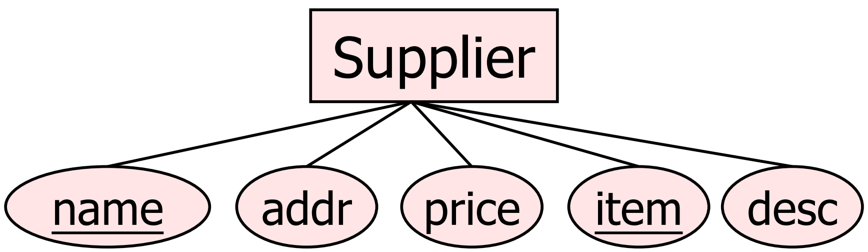
Supplier ID	Supplier Name
1	Acme
2	Beanery

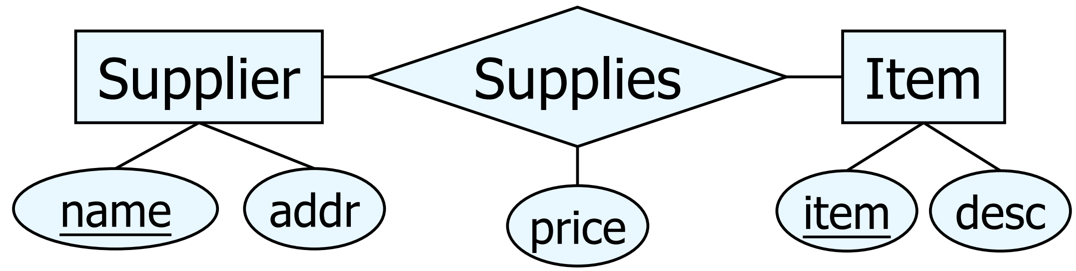
Item	Desc
Coffee	Kona
Paint	blue
Flowers	pink

Supplier ID	Item	Price
1	Coffee	$8
1	Paint	$10
1	Flowers	$3
2	Coffee	$8

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

Example: Constraints on Entity Set



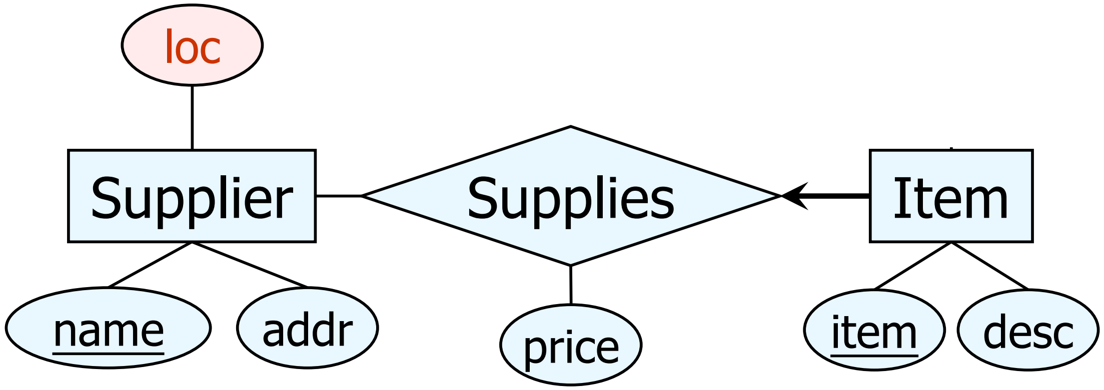


- S(name, item, desc, addr, price)
- FD: {n,i} → {n,i,d,a,p}
- FD: {n} → {a}
- FD: {i} → {d}
- Decompose to: NA, ID, INP

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

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

Refining an ER Diagram



- IS (item, name, desc, loc, price)
  S (name, addr)
- A supplier keeps all items of the same name in the same location

  FD: name → loc
- Solution:
  IS (item, name, desc, price)
  Loc (name, addr, loc)

  <- New ER diagram

Armstrong's Inference Axioms

- Axiom#1: Reflexive Property:
  - (Supplier ID, Item#) -> Item#
  - Obviously, if we know (Supplier ID and item#) pair, we know the item#.
- In general, given attribute sets X and Y
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
- In the above example, Y is [Item#]. X is [Supplier ID, Item#].
- This is called a *trivial dependency.*

Armstrong's Axioms (contd)

- Axiom #2: Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
  - ename → ejob, ejob → esal; ⇒ ename → esal
- Axiom #3: Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - If ename → ejob, then ename, salary → ejob, salary

- Additional useful rules (derivable from above axioms):
  - Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Given a set F of FDs, its closure is denoted as F^+
  - F+ obtained by repeatedly applying Armstrong's Axioms

Deriving Union Rule from Axioms

- To prove: if X $\rightarrow$ Y and X $\rightarrow$ Z then X $\rightarrow$ YZ.
- Proof:

1. X $\rightarrow$ Y (given)
2. X $\rightarrow$ Z (given)
3. XX $\rightarrow$ XZ or X $\rightarrow$ XZ (augmentation of 2.)
4. XZ $\rightarrow$ YZ (augmentation of 1.)
5. X $\rightarrow$ YZ (transitivity of 3. and 4.)

Possible to derive the decomposition rule from the basic Armstrong rules.

Decomposition

- Decompose relations to eliminate redundancy and anomalies. Two key properties:
  - Lossless join: Be able to reconstruct the original relation from instances of the decomposed relation. (You did that in Project 1, Part 4)
  - Dependency preservation: Preserve original functional dependencies
- Potential tradeoff:
  - More joins required to answer some queries
  - But, eliminating redundancy and anomalies usually worth it

Lossless Join Decompositions

- Given relation R, FDs F: R decomposed to X, Y is a 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

(Can generalize to decomposing into n 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

How to test for ossless Join?

- Given a relation R, FDs F,
- Test: The decomposition of R into X, Y is a 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 must contain a key for either X or Y

Lossless join decomposition is always required for a valid decomposition !

Dependency Preserving Decomposition

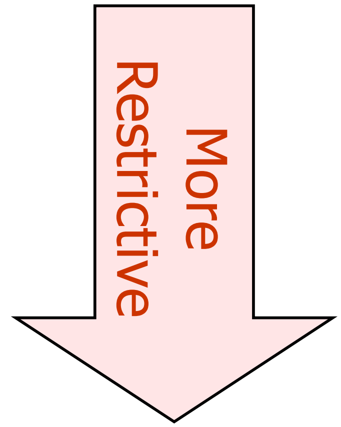
- R (sailor, boat, date)
- FD: {D → S, D → B}
- Consider decomposition to

  X (sailor, boat) Fx = {}
  Y (boat, date) Fy = {D → B}
- To enforce D → S, must join X and Y (expensive). This dependency is lost.
- Dependency preserving Rule:
  - F+ = (Fx ∪ Fy)+

The above decomposition is not dependency preserving

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



Boyce-Codd Normal Form (BCNF)

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

  i.e. all non-trivial FDs over R are due to keys.
- **No redundancy in R** (at least none that FDs detect)
- Most desirable normal form

X	Y	A
x	y1	a
x	?	a

- Consider a relation in BCNF and FD: $X \rightarrow A$, two tuples have the same X value
  - Can the y values be different?
  - **NO! non-trivial dependency**
    - => X is a (super) key $\Rightarrow$ the '?' must be y1.

3NF

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

Exercise 1: FDs & Normal Forms

Suppose you are given the following relation R: ABCDEF

BC -> D

CD -> B

D -> E

ACD -> F

1. List all of the above FDs that violate BCNF
2. List all of the above FDs that violate 3NF

Exercise 1: Solution

All possible keys:

ACD, ABC

1. Violates BCNF:

   BC->D, CD->B, D->E

2. Violates 3NF

   D->E

== FDs: ==

BC -> D

CD -> B

D -> E

ACD -> F

Exercise 2

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

  $A \rightarrow BC$
  $CD \rightarrow E$
  $B \rightarrow D$
  $E \rightarrow A$

- Identify key(s)
- Is R in BCNF?
- Is R in 3NF?

If not, list violating FDs

Use Armstrong's Axioms

Decomposition into BCNF

- Relation R with FDs F.
- Identify if any FDs violate BCNF (How?)
  - If $X \rightarrow Y$ violates BCNF, decompose R into R - Y and XY.
- Repeated application of this idea give us a collection of relations that are in BCNF.
- Does this algorithm provide a lossless join decomposition?
  - Yes! Notice that X is a key for the relation XY.
- Several dependencies may cause violation of BCNF. The order in which we "deal with" them could lead to very different sets of relations!

Algorithm for BCNF (relation R, FDs F)

done = false;
result = {R};
compute F^+;
while (not done) do
    if $\exists R_i \in$ result and R_i is not in BCNF
        let $\alpha \rightarrow \beta$ be a nontrivial FD that holds in R_i
            such that $(\alpha \rightarrow R_i) \notin F^+$ and $\alpha \cap \beta = \emptyset$;
        result=$(\text{result} - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    else done=true ;

Exercise 2 decomposition

Relation R=(A,B,C,D,E)

FDs: $A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$, $E \rightarrow A$

Solution:

- Keys: A, BC, CD, E
- $B \rightarrow D$ violates BCNF
- Decompose R into R1=(A,B,C,E) R2=(B,D)
- Is this decomposition lossless join?
  - Yes! $R1 \cap R2 = B$ and $B \rightarrow R2$
- Is it dependency preserving?
  - F1: $A \rightarrow BC$, $E \rightarrow A$
  - F2: $B \rightarrow D$
  - No! $CD \rightarrow E$ is not in $(F1 \cup F2)^+$

➢ In this case, leave it in 3NF

Exercise 3

Suppose you are given the following relation
R: ABCDEF

- **BC -> D**
- **CD -> B**
- **D -> E**
- **ACD -> F**

Do the following decompositions satisfy lossless join property?

- **R1: ACDF, R2: ABCDE**
- **R1: BCD, R2: ABEF**

Exercise 4: Solution

- R1: ACDF, R2: ABCDE

  It is lossless

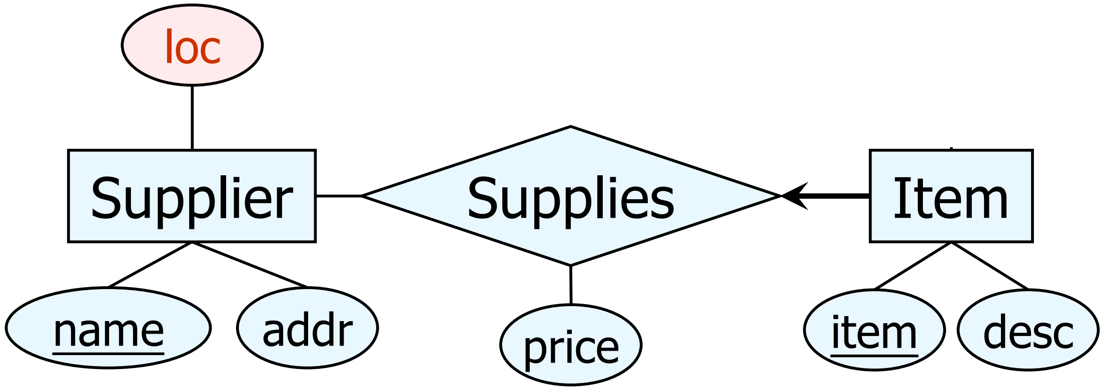
  Attributes common: ACD - it is a key

- R1: BCD, R2: ABEF

  It is not lossless

  Attributes common: B - not a key

Refining an ER Diagram



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

Normalization Example

- IS (<u>item</u>, name, desc, loc, price)
  S (<u>name</u>, addr)
- FDs = {i → ndlp, n → la}
- IS is not in BCNF, due to n → l
- Break it up: IS(<u>i</u>,n,d,p), Loc(<u>n</u>,l).
- S(<u>n</u>,a) remains unchanged.
- Now notice same key for S and Loc, so merge.
- Loc (<u>name</u>, addr, loc)

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

Exercises

- 19.1, 19.5, 19.25