Relational Database Design: The Good, The Bad and The Ugly

- Bad design & anomalies
- Normal forms
- Universal Relational Approach
- Decomposition – normalization

Relational Database Design

- One single, large table
- Simple ?
- Good ? or Bad? Or just Ugly?

Goal

- Design 'good' tables
  - define what 'good' means
  - fix 'bad' tables
- in short: "we want tables where the attributes depend on the primary key, on the whole key, and nothing but the key"
- Let's see why, and how:

Bad Design

- Relation Schema:

  takes1 (ssn, cid, grade, name, address)
- 'Bad' - why?
- because: ssn $\rightarrow$ (name, address)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Pitfalls

- Redundancy
  - Waste of space (Repeated values, NULL)
  - Update anomalies (inconsistencies)

ssn	cid	grade	name	address
123	413	A	smith	Elm
123	415	B	smith	Main
123	211	A	smith	Main

- Insertion & Deletion anomalies

Insertion Anomaly

- "jones" registers, but takes no class
- Where do you store his address!?!

ssn	cid	grade	name	address
123	413	A	smith	Main
…	…	…	…	…
234	null	null	jones	Forbes

Deletion Anomaly

- delete the last class record of 'smith'
- What about the address !?!
  - (we lose his address!)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Solution: decomposition

- Split offending table in two (or more)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

? ?

Functional dependencies

- How do we split a bad table?
- How do we recognize a bad table?
- Set of rules: *normal forms*
- **Functional Dependencies** (FD)
- Let $R=(A_1, A_2,\ldots, A_n)$ and $X\subseteq R$ and $Y\subseteq R$

  $X \rightarrow Y$ if the value of X **uniquely** determines a value of Y
  - X functionaly determines Y
  - Y is functionally dependent on X

Functional Dependency Examples

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b2	c2	d3
a3	b3	c2	d4

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

- Loans (loan_number, branch, customer, amount)
  - loan_number → amount is satisfied
  - loan_number → branch is satisfied
  - amount → branch is not satisfied

Decompositions

- There are careless, 'bad' decompositions
- There are correct decompositions
  - lossless
  - dependency preserving

Decomposition: lossless

- Non-lossy decomposition is called lossless or non-additive decompositions
- Definition:

  Let R schema R, with FD **'F'**.

  R1, R2 is a lossless decomposition of R if for all relations r(R), r1(R1) and r2(R2)

  $$r1 \bowtie r2 = r$$

Decomposition: lossy

R1(ssn, grade, name, address)

ssn	grade	name	address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

R2(cid, grade)

cid	grade
413	A
415	B
211	A

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn → (name, address)

(ssn, cid) → grade ✘

can not recover original table with a join!

Decomposition: lossless…

- What about an easier criterion?
- Theorem:
  A decomposition is lossless if the joining attribute is a *superkey* in at least one of the new tables
- Formally:

$$R1 \cap R2 \rightarrow R1 \; or$$
$$R1 \cap R2 \rightarrow R2$$

Example: Loseless Decomposition

R1

ssn	cid	grade
123	413	A
123	415	B
234	211	A

(ssn, cid) → grade

R2

ssn	name	address
123	smith	Main
234	jones	Forbes

ssn → (name, address)

ssn	cid	grade	name	address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn → (name, address)

(ssn, cid) → grade

Dependency Preservation & Canonical Cover

- We don't want the original FDs to span two tables.
- More specifically: … the FDs of the **canonical cover**
- **Canonical Cover** is the minimum set of FDs without any trivial, extraneous and implied FDs
- Example:
  fd.1 A → B
  fd.2 B → C
  fd.3 AB → C
  - What is the canonical cover?
  Keep only fd.1 & fd.2: A → B & B → C

Rules of Inference: Amstrong's Axions

- Reflexivity rule:

  if $B \subset A$, then $A \rightarrow B$
- Augmentation rule:

  if $A \rightarrow B$, then $AC \rightarrow BC$
- Transitive rule:

  if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

More Rules of Inference

- Self-determination:

  $A \rightarrow A$
- Union rule:

  if $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow BC$
- Decomposition rule:

  if $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$
- Pseudotransitivity, Composition:

  if $A \rightarrow B$ and $CB \rightarrow D$ implies $CA \rightarrow D$

  if $A \rightarrow B$ and $C \rightarrow D$, then $AC \rightarrow BD$

Decomposition: non-dependency preserving

- Dependency preserving: the original FDs should not span across tables
- Non-dependency preserving example:

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh

S#	status
123	E
125	E
234	A

S# → address **S# → status**

quiz: is it lossless?

Dependency Preserving Decomposition

- Example

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh

address	status
London	E
Paris	E
Pgh	A

S#→address address→status

but: S# → status ?

- why is dependency preservation good?

Why Dependency Preservation?

- Record the status for Athens to be A:

S#	address	status
123	London	E
125	Paris	E
234	Pgh	A

S# → (address, status)

address → status

S#	address
123	London
125	Paris
234	Pgh
333	Athens

S#→address

address	status
London	E
Paris	E
Pgh	A
Athens	A

address→status

Decomposition - conclusions

- decompositions should always be lossless
  - joining attribute -> superkey
- Decompositions should be dependency preserving whenever possible

Testing for Loseless Decomposition

- Given
  - relation schema R (A1, A2, …, An),
  - a set of functional dependencies F and
  - decomposition p = {R1, R2,…, Rm}
- Steps:

1. Construct an *m* x *n* table S,

   with a column for each of the *n* attributes in R and

   row for each of the *m* relations in the decomposition.

Testing for Loseless Decomposition…

2. For each cell S(i,j) of S,

   If the attribute for the column Aj is in the relation for the row Ri,

   then set S(i,j) = k(j) to indicate *known* value

   else set S(i,j) = u(i,j) to indicate *unknown* value

3. Consider each FD, X → Y F until no more changes can be made to S.
   - Look for rows whose X-column agrees
   - EQUATE Y-column
4. After all possible changes have been made to S,
   - If a row is made up entirely of symbols k(1), k(2), .., k(n) the join is lossless.
   - Otherwise, if there is no such row, the join is lossy.

Example

- R (A,B,C,D,E)
- Decomposition: R1(A,C), R2(A,B,D), R3(D,E)
- FDs: A $\rightarrow$ C, AB $\rightarrow$ D, D $\rightarrow$ E

	A	B	C	D	E
R1(A,C)					
R2(A,B,D)					
R3(D,E)					

Relational Database Design

- One single, large table
- Simple ?
- Good ? or Bad? Or just Ugly?

Normal Forms

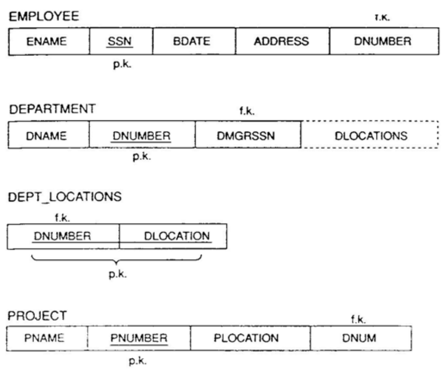
- We discussed how to fix 'bad' schemas
- but what is a 'good' schema?
- Informally: "we want tables where the attributes depend on the primary key, on the whole key, and nothing but the key"
- Formally: 'good', if it obeys a 'normal form'
- Typically: Boyce-Codd Normal form or the 3NF
- Normal forms are defined in terms of FDs

Functional dependency

- Definition: Let $R=(A_1, A_2,\ldots, A_n)$ and $X \subseteq R$ and $Y \subseteq R$
  $X \rightarrow Y$ if the value of X **uniquely** determines a value of Y
- A functional dependency is a property of the meaning or semantic of the attributes in a relation schema.
- We use our understanding of the semantics of the attributes of R – that is, how they relate to one another – to specify the FD that should hold an all relational instances.
- Functional dependence is a semantic notion.
  - Recognizing the FDs is part of the process of understanding what data means.

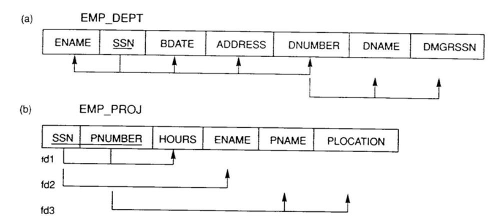
Good Database Schema

- Relations should have simple meaning



Bad Database Schema

- Relations should not have multiple meanings



Types of Functional Dependencies

- Trivial dependency: X→Y is *trivial* if it is true for any X and Y of any relation, regardless of X and Y semantics.
  - Ex1: A -> A
  - Ex2: If {A,B} a Key, then {A,B} → A (Y⊆X, X→Y is trivial)
- Partial dependency: X→Y is *partial* if there is an attribute A in X that can be removed from X and the dependency can still hold: X–{A} → Y
  - E.g., SUPPLY (SID,PID,DID, SCity, DCity, Qty)

    {SID, PID, DID} → Scity SID → SCity

    {SID, PID, DID} → Dcity DID → Dcity
- Full dependency: ??

Types of Functional Dependencies…

- Transitive dependency: X→Y is transitive in R if there is a set of attributes Z that is not a subset of any key of R and both X → Z and Z → Y hold
  - E.g., EMP (SSN, EName, DeptID, MGRSSN)

    (fd.1) SSN → DeptId

    (fd.2) DeptId → MGRSSN

    (from fd.1 & fd.2) SSN → MGRSSN
- Multivalued dependency : X→Y is multivalued dependency in R if X is a key, Z in R and Z→Y
  - E.g., DJP (DeptID, ProjectID, part)

    (fd.1) DeptId → part

    (fd.2) ProjectID → part