Database Systems

Relational Design - Normalization

Prof. Dr. Agnès Voisard Muhammed-Ugur Karagülle

Institute of Computer Science, Databases and Information Systems Group

Fraunhofer FOKUS

2025

Notes

Notes

Outline

1 Motivation

2 Anomalies

3 Loss of Information

4 Decomposition

5 Normal Forms

Motivation

What can go wrong?

- Repetition of information
- Inability to represent some information
- Loss of information

Example

Banking example:

```
BRANCH(BranchName,Assets,BranchCity)
BORROW(BranchName,LoanNum,CName,Amount)
```

BRANCH

BranchName	Assets	BranchCity
Downtown	90000000	Brooklyn
Redwood	210000000	Palo Alto
Perryridge	170000000	Manhattan
Mianus	40000000	Manhattan
Round Hill	8000000000	Manhattan
Pownal	30000000	Bennington
North Town	3700000000	Rye
Brighton	7100000000	Brookyn

Reference Example (cont'd)

BORROW

BranchName	LoanNum	CName	Amount
Downtown	17	Jones	1000
Redwood	23	Smith	2000
Perryridge	15	Hayes	1500
Downtown	14	Jackson	1500
Mianus	93	Curry	500
Round Hill	11	Turner	900
Pownal	29	Williams	1200
North Town	16	Adams	1300
Downtown	18	Johnson	2000
Perryridge	25	Glenn	2500
Brighton	10	Brooks	2200

Example

Alternative design:

`LENDING(BranchName,Assets,BranchCity,LoanNum,CName,Amount)`

- Natural join of previous BRANCH and BORROW instances

BranchName	Assets	BranchCity	LoanNum	CName	Amount
Downtown	9000000	Brooklyn	17	Jones	1000
Redwood	21000000	Palo Alto	23	Smith	2000
Perryridge	17000000	Manhattan	15	Hayes	1500
Downtown	9000000	Brooklyn	14	Jackson	1500
Mianus	4000000	Manhattan	93	Turner	900
Round Hill	800000000	Bennington	29	Williams	1200

Anomalies

Representing, updating, adding, or deleting certain facts in the database may lead to problems
Problems: **Anomalies**

1 **Redundancy**: Waste of space

Example

Branchcity repeated for each BranchName

Update: Potential inconsistency

Example

Perryridge branch moves from Manhattan to Newtown
Update every tuple (costly), or
change it in one tuple and not in the other ones

3 **Insertion**: We need to have at least one loan in the branch to have information about it (if no null values)

Example

E.g., what if there is a branch without loans?

Deletion: If we delete all customers of BranchName we loose track of the information regarding Branches

Example

E.g., BranchCity

4

Loss of Information

Previous (bad) design $\Rightarrow$ Decompose a relation schema with many attributes into several schemas with fewer attributes

Example

BORROW(Branchname,LoanNum,CName,Amount)
decomposed into:

AMT(CName,Amount)
LOAN(BranchName,LoanNum,Amount)

AMT = $\Pi_{CName,Amount}$(BORROW)
LOAN = $\Pi_{BranchName,LoanNum,Amount}$(BORROW)

Reference Example - BORROW Relation Repeated

BORROW

BranchName	LoanNum	CName	Amount
Downtown	17	Jones	1000
Redwood	23	Smith	2000
Perryridge	15	Hayes	1500
Downtown	14	Jackson	1500
Mianus	93	Curry	500
Round Hill	11	Turner	900
Pownal	29	Williams	1200
North Town	16	Adams	1300
Downtown	18	Johnson	2000
Perryridge	25	Glenn	2500
Brighton	10	Brooks	2200

LOAN & AMT

LOAN

BranchName	LoanNum	Amount
Downtown	17	1000
Redwood	23	2000
Perryridge	15	1500
Downtown	14	1500
Mianus	93	500
Round Hill	11	900
Pownal	29	1200
North Town	16	1300
Downtown	18	2000
Perryridge	25	2500
Brighton	10	2200

AMT

CName	Amount
Jones	1000
Smith	2000
Hayes	1500
Jackson	1500
Curry	500
Turner	900
Williams	1200
Adams	1300
Johnson	2000
Glenn	2500
Brooks	2200

Loss of Information (cont'd)

Branches from which Jones has a loan?

$\Pi_{BranchName}\sigma_{CName="Jones"}$ (AMT ⋈ LOAN)

AMT ⋈ LOAN

BranchName	LoanNum	Amount	CName
Downtown	17	1000	Jones
Redwood	23	2000	Smith
Redwood	23	2000	Johnson
Perryridge	15	1500	Hayes
Perryridge	15	1500	Jackson
Downtown	14	1500	Hayes
Downtown	14	1500	Jackon
Mianus	93	500	Curry
Round Hill	11	900	Turner
Pownal	29	1200	Williams
North Town	16	1300	Adams
Downtown	18	2000	Smith
Downtown	18	2000	Johnson
Perryridge	25	2500	Glenn
Brighton	10	2200	Brooks

Loss of Information (cont'd)

Branches where Jackson has a loan?

$\Pi_{BranchName}\sigma_{CName=''Jackson''}$ (AMT $\bowtie$ LOAN):
Perryridge and Downtown

Some tuples in AMT $\bowtie$ LOAN are not in BORROW

New tuples created

Loss of Information (cont'd)

More tuples in AMT ⋈ LOAN but less information

We are not able to represent which customer borrows from which branch: We have lost the information

Lossy decomposition or **lossy-join** decomposition

Decomposition that is not a lossy join decomposition
is a **lossless-join** decomposition

Loss of Information (cont'd)

Why is it lossy?

Only attribute in common between `LOAN` and `AMT`: Amount
Only way we can express a relationship between `LOAN` and `AMT` is through Amount!

Not adequate:
Many customers may have loans in the same amount
and not necessarily from the the same branch

Loss of Information (cont'd)

LENDING decomposed into BRANCH and BORROW
BRANCH $\cap$ BORROW = {BranchName}
Relationship between CName and Assets is through BranchName

Difference with previous example
BranchName $\rightarrow$ Assets, BranchCity
Amount $\not\rightarrow$ BranchName

Let R be a relation schema

A set of relations schemas ρ: $\{R_1, R_2, ..., R_n\}$
is a **decomposition** of R if:

$R = R_1 \cup R_2 \cup ... \cup R_n$
(every attribute in R appears in at least one R_i, $1 \geq i \geq n$)

A decomposition $\{R_1, R_2, ..., R_n\}$ of a relation schema R is a lossless-join decomposition for R if for all relations r on schema R satisfying a set of dependencies FD:

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_n}(r)$$

Lossless join property: guarantees that any relation can be recovered from its projection onto relations

Theorem:
If $\rho = (R_1, R_2)$ is a decomposition of R and F a set of FD, then ρ has a lossless-join with respect to F if and only if:

$$R_1 \cap R_2 \rightarrow R_1 - R_2$$
or
$$R_1 \cap R_2 \rightarrow R_2 - R_1$$

belongs to F^+

Example

$R(A, B, C), FD_R = \{A \rightarrow B\}$

1. $R_1(AB), R_2(BC)$
   $R_1 \cap R_2 = B$
   $R_1 - R_2 = A$
   $R_2 - R_1 = C$
   Do we have $B \rightarrow A$ or $B \rightarrow C$?
2. $R_1(AB), R_2(AC)$
   $R_1 \cap R_2 = A$
   $R_1 - R_2 = B$
   $R_2 - R_1 = C$
   Do we have $A \rightarrow B$ or $A \rightarrow C$?

Decomposition that Preserves Dependencies

Other important property of a decomposition $\rho(R_1, \ldots, R_n)$:
Set of dependencies F implied by the projection of F onto the R_i's is preserved

Projection of F onto a set of attributes Z $\subseteq$ U:

$\pi_Z(F) = \{X \to Y \in F^+ \mid XY \subseteq Z\}$

Example

$R(A, B, C, D)$

$F = \{AB \to C,\ C \to A,\ A \to D\}$

$\pi_{ABC}(F) = \{AB \to C,\ C \to A\}$

Decomposition that Preserves Dependencies (cont'd)

Decomposition ρ **preserves** a set of dependencies F
if the union of all the dependencies in $\pi_{Ri}(F)$, for $i = 1, 2, ..., n$
logically implies all the dependencies in F

Example

$R(C, S, Z)$
$CS \rightarrow Z$
$Z \rightarrow C$

1. $\rho = (SZ, CZ)$ Lossless join?
2. Does it preserve dependencies?
   π_{SZ} = trivial dependencies
   $\pi_{CZ} = Z \rightarrow C$ + trivial dependencies

Remark:
Decomposition may have a lossless join with respect to F,
yet not preserve F, and may preserve F and have a lossless join

Normal forms for relations schemes with dependencies
"Good" database design **Most of anomalies do not occur**

First Normal Form (**1NF**)
What we called relation so far
All attributes have an atomic domain

Non First Normal Forms (N1NF): Nested relations where attributes are relations

Notion of **prime** attribute
Attribute A in relation scheme R is a prime attribute if A is a member of any key for R (otherwise non-prime)

Example

$R(A, B, C, D)$, $F_R = \{AB \rightarrow C,\ B \rightarrow D,\ BC \rightarrow A\}$
Keys: AB, BC
A, B, C prime
D non-prime

2nd Normal Form (2NF)

Definition:

Second Normal Form **2NF**
A relation schema is in 2NF if and only if it is in 1NF and no non-prime attribute is dependent on a subset of any candidate key of the table

I.e., a relation schema mis in 2NF if and only if, it is in 1NF and every non-prime attribute of the table is either dependent on the whole of a candidate key, or on another non-prime attribute

For each $(X \rightarrow A)$ in $F+$ such that A is not in X, then X is not a strict subset of any superkey or A belongs to a key

Definition:

A relation schema R is in 3NF if,
whenever a FD ($X \rightarrow A$) holds in R, either:

X is a superkey of R, or
A is a prime attribute of R

Example

$R(C, S, Z)$
$F_R = \{CS \rightarrow Z,\ Z \rightarrow C\}$
Keys: CS, SZ
CS is a key
$C \subseteq CS$, $C \in$ a key
$\Rightarrow$ R in 3NF

Theorem:
Every 1NF relation has a "well-behaved" decomposition,
i.e., in 3NF, with lossless join that preserves dependencies

“Well-behaved” 3NF Decomposition

Assume F is a **minimal cover**

Algorithm to achieve a well-behaved 3NF decomposition

1. For each FD $(X \rightarrow A)$ in F create a relation with schema (XA)
2. If none of the keys appears in one of the schemas of 1 then add a relation with schema Y, with Y a key
3. If for relations created in 1 there exists a relation R_1 whose schema is included in the schema of another relation, then remove R_1
4. Replace relations (XA_1), ..., (XA_k) with a single relation $(XA_1...A_k)$

“Well-behaved” 3NF Decomposition (cont’d)

$R(A, B, C, D),\ F_R = \{AB \to C,\ B \to D,\ C \to A\}$ is not in 3NF
Keys: AB, BC

1. $R_1(A, B, C),\ R_2(B, D),\ R_3(C, A)$
2. Not necessary to add a relation with AB: AB is in R_1
3. Remove R_3: $CA \subseteq ABC$

“Good” decomposition: $R_1(A, B, C),\ F_{R_1} = \{AB \to C,\ C \to A\}$
$R_2(B, D),\ F_{R_2} = \{B \to D\}$
Check that R_1 and R_2 are in 3NF

Some anomalies in the Third Normal Form (Redundancy)

Example

$R(C, S, Z)$
$F = \{CS \rightarrow Z,\ Z \rightarrow C\}$
Keys: CS, SZ
Redundancy: $C \leftrightarrow Z$
Boyce-Codd Normal Form (BCNF)

Definition:
A relation is in BCNF if for each $X \rightarrow A$ in F^+,
X is a superkey
Theorem:
$BCNF \Rightarrow 3NF \Rightarrow 2NF$

Goal for Relational DB Design

- BCNF
- Lossless join
- Dependency preservation

If it cannot be achieved:

- 3NF
- Lossless join
- Dependency preservation

Questions?

What will come next?

1 Welcome to Database Systems
2 Introduction to Database Systems
3 Entity Relationship Design Diagram (ERM)
4 Relational Model
5 Relational Algebra
6 Structured Query Language (SQL)
7 Relational Database Design - Functional Dependencies
8 Relational Database Design - Normalization
9 Online Analytical Processing + Embedded SQL
10 Physical Representation - Storage and File Structure
11 Physical Representation - Indexing and Hashing
12 Transactions
13 Concurrency Control Techniques
14 Recovery Techniques
15 Query Processing and Optimization