Design Theory for Relational Databases

Normalization

Guillaume Raschia — Polytech Nantes; Université de Nantes
Last update: November 16, 2020

Contents

[Source : J. Ullman, Stanford]

Bad Design

Relational Schema Design

Goal of relational schema design is to avoid **anomalies** and **redundancy**.

- Update anomaly: one occurrence of a fact is changed, but not all occurrences
- Insertion anomaly: related facts are required when a tuple is inserted
- Deletion anomaly: valid fact is lost when a tuple is deleted

Example of Bad Design

Drinkers(name, addr, beersLiked, brewery, favBeer)

name	addr	beersLiked	brewery	favBeer
Alice	Nantes	Trompe Souris	La Divatte	Titan
Alice	???	**Titan**	Bouffay	???
Bob	Rennes	**Titan**	???	Titan

Data is redundant, because each of the ???'s can be figured out by using the FD's **name** $\rightarrow$ addr favBeer and **beersLiked** $\rightarrow$ brewery

Bad Design: Update Anomalies

name	addr	beersLiked	brewery	favBeer
Alice	~~Nantes~~ **Vannes**	Trompe Souris	La Divatte Bouffay	Titan
Alice	Nantes	Titan	Bouffay	Titan
Bob	Rennes	Titan	Bouffay	Titan

- If Alice moves to Vannes, will we remember to change each of her tuples?

Bad Design: Deletion Anomalies

name	addr	beersLiked	brewery	favBeer
~~Alice~~	~~Nantes~~	~~Trompe Souris~~	~~La Divatte~~	~~Titan~~
Alice	Nantes	Titan	Bouffay	Titan
Bob	Rennes	Titan	Bouffay	Titan

- If nobody likes Trompe Souris anymore, we lose track of the fact that La Divatte brews Trompe Souris

Bad Design: Insertion Anomalies

name	addr	beersLiked	brewery	favBeer
Alice	Nantes	Trompe Souris	La Divatte	Titan
Alice	Nantes	Titan	Bouffay	Titan
Bob	Rennes	Titan	Bouffay	Titan
Charlie	Nantes	Mistral	Aerofab	Mistral

- If Charlie comes into play, one must know beers s/he likes and their breweries, otherwise `null` values

BCNF

Boyce-Codd Normal Form

Definition (BCNF)
We say a relation R is in BCNF if whenever $X \to Y$ is a nontrivial FD that holds in R, X is a superkey

- Remember: **nontrivial** means Y is not contained in X
- Remember: a **superkey** is any superset of a key (not necessarily a proper superset)

Example: BCNF

`Drinkers(name, addr, beersLiked, brewery, favBeer)`

- FD's:
  - name $\rightarrow$ addr favBeer
  - beersLiked $\rightarrow$ brewery
- Only key is {name, beersLiked}
- In each FD, the left-hand side is not a superkey
- Any one of these FD's shows `Drinkers` is not in BCNF

Another Example

`Beers(name, brewery, brewAddr)`

- FD's:
  - name $\rightarrow$ brewery
  - brewery $\rightarrow$ brewAddr
- Only key is {name}
- name$\rightarrow$ brewery does not violate BCNF, but brewery$\rightarrow$ brewAddr does

Decomposition into BCNF

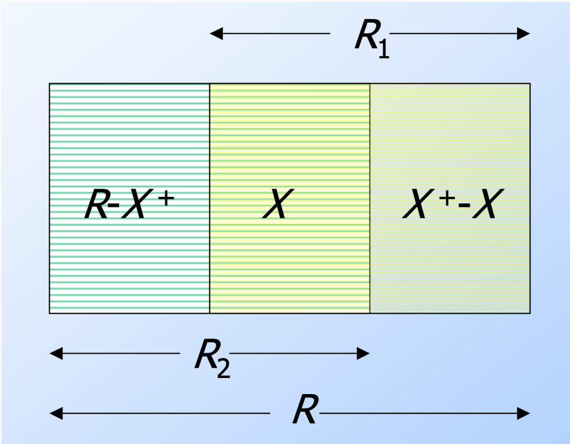
Given: relation R with FD's $\mathcal{F}$

1. Look among the given FD's for a BCNF violation $X \to Y$
   - If any FD following from $\mathcal{F}$ violates BCNF, then there will surely be an FD in $\mathcal{F}$ itself that violates BCNF
2. Compute X^+
   - Not all attributes, or else X is a superkey

Decompose R Using $X \to Y$

3. Replace R by relations with schemas:
   - $R_1 = X^+$
   - $R_2 = R - (X^+ - X)$
4. Project given FD's $\mathcal{F}$ onto the two new relations

Decomposition Picture



J. Ullman

Example: BCNF Decomposition

`Drinkers(name, addr, beersLiked, brewery, favBeer)`

$\mathcal{F}$ = {name→ addr, name → favBeer, beersLiked → brewery}

1. Pick BCNF violation name→ addr
2. Close the left side: $\{\text{name}\}^+$ = {name, addr, favBeer}
3. Decomposed relations:
   - `Drinkers1(name, addr, favBeer)`
   - `Drinkers2(name, beersLiked, brewery)`

Example (cont'd)

We are not done; we need to check `Drinkers1` and `Drinkers2` for BCNF

- Projecting FD's is easy here
- For `Drinkers1(name, addr, favBeer)`, relevant FD's are name→ addr and name→ favBeer
- Thus, {name} is the only key and `Drinkers1` is in BCNF

Example (cont'd)

- For `Drinkers2(name, beersLiked, brewery)`, the only FD is beersLiked→ brewery, and the only key is {name, beersLiked}
- Violation of BCNF
- beersLiked$^+$ = {beersLiked, brewery}, so we decompose `Drinkers2` into:
  - `Drinkers3(beersLiked, brewery)`
  - `Drinkers4(name, beersLiked)`

Example — Concluded

The resulting decomposition of `Drinkers`

- `Drinkers1(name, addr, favBeer)`
- `Drinkers3(beersLiked, brewery)`
- `Drinkers4(name, beersLiked)`

Notice: `Drinkers1` tells us about drinkers, `Drinkers3` tells us about beers, and `Drinkers4` tells us the relationship between drinkers and the beers they like

3NF

Third Normal Form — Motivation

There is one structure of FD's that causes trouble when we decompose

- AB $\rightarrow$ C and C$\rightarrow$ B
  - Example: A = street address, B = city, C = zip code
- There are two keys, {A,B} and {A,C}
- C$\rightarrow$ B is a BCNF violation, so we must decompose into AC, BC

We Cannot Enforce FD's

The problem is that if we use AC and BC as our database schema, we cannot enforce the FD AB $\rightarrow$ C by checking FD's in these decomposed relations

A = street, B = city, and C = zip

$R =$

street	zip
50 Otages	44000
50 Otages	44100

$S =$

zip	city
44000	Nantes
44100	Nantes

{street, zip} is key in R, {zip} is key in S

An Unenforceable FD

$R \bowtie S$ =

street	city	zip
50 Otages	Nantes	44000
50 Otages	Nantes	44100

$R \bowtie S$ joins tuples with equal zip codes

Although no FD's were violated in the decomposed relations, FD street city $\rightarrow$ zip is violated by the database as a whole

- 3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation

3rd Normal Form

Definition (3NF)
Every FD $X \rightarrow A$ satisfies one of those three conditions:

1. X$\rightarrow$ A is trivial
2. X is a superkey
3. A is prime (more flexible than BCNF)
    - An attribute is prime if it is a member of any key

In other words, a nontrivial FD $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime

Example: 3NF

- In our problem situation with FD's AB$\rightarrow$ C and C$\rightarrow$ B, we have keys AB and AC
- A, B and C are each prime
- Although C$\rightarrow$ B violates BCNF, it **does not violate 3NF** (B is prime)
- One can decide not to decompose to BCNF, still being 3NF

What 3NF and BCNF Give You?

Two important properties of a decomposition

1. **Lossless Join**: it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original
2. **Dependency Preservation**: it should be possible to check in the projected relations whether all the given FD's are satisfied

3NF and BCNF (cont'd)

- We can get **(1)** with a **BCNF** decomposition
- We can get both **(1) and (2)** with a **3NF** decomposition
- But we can't always get (1) and (2) with a BCNF decomposition
  - street-city-zip is an example

Testing for a Lossless Join

- If we project R onto R_1, R_2, ..., R_n, can we recover R by rejoining?
  - A projected fragment: $R_i = \pi_{X_i}(R)$
  - Does R equal to $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$?
- Any tuple in R can be recovered from its projected fragments
  - $R \subseteq R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ is obvious
- So the only question is: when we rejoin, do we ever get back something we didn't have originally?
  - $R \supseteq R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ must be proved

The Chase Test

- Suppose tuple t comes back in the join
- Then t is the join of projections of some tuples of R, one for each R_i of the decomposition
- Can we use the given FD's to show that one of these tuples must be t?

The Chase – (cont'd)

Procedure

1. Start by assuming $t = abc\ldots$
2. For each i, there is a tuple s_i of R that has a, b, c, ... in the attributes of R_i
3. s_i can have any values in other attributes
4. We'll use the same letter as in t, but with a subscript, for these components

Example: The Chase

- Let $R(A, B, C, D)$ and the decomposition be $R_1(A, B)$, $R_2(B, C)$ and $R_3(C, D)$
- Let the given FD's be $\mathcal{F} = \{C \to D, B \to A\}$
- Suppose the tuple $t = abcd$ is the join of tuples projected onto AB, BC, CD

The Tableau

Let's build an instance of R from tuple $t = abcd$ in $\pi_{AB}(R) \bowtie \pi_{BC}(R) \bowtie \pi_{CD}(R)$

R	A	B	C	D
from AB part of t	a	b	c_1	d_1
from BC part of t	$\cancel{a_2}$ a	b	c	$\cancel{d_2}$ d
from CD part of t	a_3	b_3	c	d

- Use B$\to$ A to state a_2 must be a
- Use C$\to$ D to state d_2 must be d

We've proved the second tuple must be $t = abcd$; then $(\pi_{AB}(R) \bowtie \pi_{BC}(R) \bowtie \pi_{CD}(R)) \subseteq R$!

Summary of the Chase

Build the Tableau, then

1. If two rows agree in the left side of a FD, make their right sides agree too
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original table (the join is **lossless**)
4. Otherwise, the final tableau is a **counterexample**

Example: Lossy Join

- Same relation $R(A, B, C, D)$ and same decomposition AB, BC, CD
- But with only the FD C$\rightarrow$ D

The Tableau

R	A	B	C	D
$t_1 =$	a	b	c_1	d_1
$t_2 =$	a_2	b	c	$\cancel{d_2}$ d
$t_3 =$	a_3	b_3	c	d

- Use C$\rightarrow$ D to state d_2 must be d, and that's all

These three tuples are an example of R that shows **the join is lossy**:

- $abcd$ is not in R, but we can project and rejoin to get $t = abcd$

$$t_1[AB] \bowtie t_2[BC] \bowtie t_3[CD] = abcd$$

3NF Decomposition

There is always a **lossless-join and dependency-preserving 3NF** decomposition

How to achieve 3NF?

1. Perform the iterative binary decomposition process up to 3NF only
2. Use the **3NF Synthesis**
   - Need a minimal basis for the FD's

Minimal Cover and 3NF Synthesis

Minimal Cover

also known as **minimal basis** or even **canonical cover**

A set of FD's is a minimal cover iff

1. *rhs*'s are single attributes
2. No **redundant FD**, ie FD that can be discarded
3. No **extraneous attribute in *lhs***, ie that can be removed from the *lhs*

Constructing a Minimal Cover

Given a set of FD's $\mathcal{F}$

Finding a minimal cover $\mathcal{F}_{\min}$ requires:

1. Split *rhs*'s
2. Repeatedly try to remove an FD X $\rightarrow$ A and see if the remaining FD's are equivalent to the original
3. Repeatedly try to remove an attribute B from a *lhs* BX$\rightarrow$ A and see if the resulting FD's are equivalent to the original
4. Iterate 2-3 up to stable

Constructing a Minimal Cover — for Real

How to achieve Step 2: $\mathcal{F} - \{X \rightarrow A\} \equiv \mathcal{F}$?

$\Leftrightarrow$ $\mathcal{F} - \{X \rightarrow A\} \models X \rightarrow A$? (to prove $\mathcal{F} - \{X \rightarrow A\}$ implies $\mathcal{F}$)

$\Leftrightarrow$ Check for $A \in X^+$ wrt $\mathcal{F} - \{X \rightarrow A\}$

How to achieve Step 3: $(\mathcal{F} - \{BX \rightarrow A\} \cup \{X \rightarrow A\}) \equiv \mathcal{F}$?

$\Leftrightarrow$ $\mathcal{F}$ implies $(\mathcal{F} - \{BX \rightarrow A\} \cup \{X \rightarrow A\})$

- The other way round is obvious since $X \rightarrow A \models BX \rightarrow A$

$\Leftrightarrow$ $\mathcal{F} \models X \rightarrow A$

$\Leftrightarrow$ Check for $A \in X^+$ wrt $\mathcal{F}$

Example: Minimal Cover

Given the FD's $\mathcal{F} = \{ABC \to CD, A \to B, C \to A\}$

- Step 1: $\mathcal{F}_1 = \{ABC \to C, ABC \to D, A \to B, C \to A\}$
- Step2:
  - remove (trivial) $ABC \to C$ to get $\mathcal{F}_2 = \mathcal{F}_1 - \{ABC \to C\}$
  - cannot remove $ABC \to D$, since $D \notin ABC^+_{\mathcal{F}_2 - \{ABC \to D\}} = ABC$
  - cannot remove $A \to B$, since $B \notin A^+_{\mathcal{F}_2 - \{A \to B\}} = A$
  - cannot remove $C \to A$, since $A \notin C^+_{\mathcal{F}_2 - \{C \to A\}} = C$

Example: Minimal Cover (cont'd)

Given the FD's $\mathcal{F} = \{ABC \to CD, A \to B, C \to A\}$

Step 1-2 yields to $\mathcal{F}_2 = \{ABC \to D, A \to B, C \to A\}$

- Step 3:
  - remove A in $ABC \to D$ since $D \in BC^+_{\mathcal{F}_2} = BCAD$. Define $\mathcal{F}_3$
  - remove B in $BC \to D$ since $D \in C^+_{\mathcal{F}_3} = CABD$. Define $\mathcal{F}_4$
  - cannot remove C in $C \to D$ (singleton in *lhs*)
  - nothing to do for $A \to B$ and $C \to A$ (singletons in *lhs*)

Finally, $\mathcal{F}_{\min} = \mathcal{F}_4 = \{C \to D, A \to B, C \to A\}$

Properties of Minimal Cover

Given a set of FD's $\mathcal{F}$

- FD's in $\mathcal{F}_{\text{min}}$ are irreducibles
- $\mathcal{F}_{\text{min}} \equiv \mathcal{F}$ that is equivalent to $\mathcal{F}_{\text{min}}^{+} = \mathcal{F}^{+}$
- $\mathcal{F}_{\text{min}}$ is **not unique**
  - depends on the nondeterministic choices in Steps 2-3
- $\mathcal{F}_{\text{min}}$ is required for the 3NF synthetis algorithm

3NF Synthesis

1. Create one relation for each ***lhs*** of FD's in the minimal cover
   - Schema is the union of ***lhs*** and set of ***rhs***'s
2. Discard relation $R(X)$ if $S(XY)$ exists
3. If no key is contained in an FD, then add one relation whose schema is some key

Example: 3NF Synthesis

Relation $R(A, B, C, D, E)$ with FD's $\mathcal{F} = \{AB \to C, AB \to D, C \to B, E \to B\}$

Assume $\mathcal{F}$ is a **minimal cover**

Decomposition

1. ABCD, CB and EB
2. Then, remove CB since CB $\subseteq$ ABCD
3. And add AE for a key

Example: 3NF Synthesis (cont'd)

Resulting decomposition is $R_1(\underline{AB}CD)$, $R_2(\underline{E}B)$ and $R_3(\underline{AE})$

- R_1 R_2 and R_3 are 3NF
- R_1 is not BCNF by C$\to$ B

Why It Works?

3NF Synthesis

- **Preserves dependencies**: each FD from a minimal cover is contained in a relation, thus preserved
- **Lossless Join**: use the chase to show that the row for the relation that contains a key can be made all unsubscripted variables
- **3NF**: hard part, a property of minimal covers

The Many Other Normal Forms

First Normal Form

The very baseline of Relational Database Design

1NF

Relation has (a) a **key** and (b) **atomic** columns and (c) **no repeating groups** of columns

- Sets or tuples or tables are not allowed as attribute values
- (beersLiked$_1$, beersLiked$_2$, BeersLiked$_3$) is not allowed as a subset of columns

Second Normal Form

2NF

1NF and every non-prime attribute is **fully** functionally dependent on keys

- Remind: non-prime attributes are not key attributes
- FD X$\rightarrow$ A is full *iff* A doesn't depend on a proper subset of X
- 2NF is not so relevant in DB design

Example: 2NF

`Drinkers(name, addr, beersLiked, brewery, favBeer)`

FD's are name → addr favBeer and beersLiked → brewery

- `Drinkers` is 1NF: (a) key, (b) atomic values (c) non-repeating attributes
- `Drinkers` is not 2NF
  - name beersLiked → addr is not full, since name → addr holds
- `Drinkers` cannot be 3NF either

Normal Forms: Best Practices

1NF < 2NF < 3NF < BCNF

- Always try to decompose up to BCNF
- When one cannot get dependency preserving BCNF, may decide to stop to 3NF
- Denormalize below 3NF only for good reason (performance) or data model shift[1]

[1] nested relations, document db, key-value stores, ...

More Stringent Normal Forms

Beyond Functional Dependencies

- Multi-Valued Dependencies: 4NF
- FD + Join Dependencies: ETNF (H. Darwen et al., 2012)
- JD: 5NF, 6NF
- Domain and Key constraints: DKNF
- ...