Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Ralf Möller

Databases
Relational Schema Design Theory (Part 2)


Relational schema design theory
Normal forms

Summary and outlook

1. **Introduction**
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. **Entity-relationship model (ER model)**
3. **Relational data model (RDM)**
   - From ER to RDM
4. **Relational schema design theory**
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages

Agreement on notation

- FDs that are fulfilled by every relation are called **trivial**
  Only FDs of the form $\alpha \longrightarrow \beta$ with $\beta \subseteq \alpha$ are trivial

- Attributes of a candidate key are called **prime**
  All other attribute of a relational schema are called non-prime

- Let $\mathcal{R}$ be a relational schema. Then the set of associated FDs is written as $F_{\mathcal{R}}$
  - If $\mathcal{R}$ is clear from context, we just write F

Avoid redundancies in data

Example:

$\mathcal{R}$ = {[A, B, C ,D]}, F = {A ⟶ B, D ⟶ ABCD}, candidate key: {D}

R			
A	B	C	D
3	4	5	1
3	4	6	2

„Do not represent the same fact twice“

General case: If $\alpha \longrightarrow \beta \in F$ then: α should be a super key or FD need to be trivial

If this is not the case: Decompose relation (see below: lossless and dependency-preserving)

Decomposition of relations

Correctness criteria for a decomposition :

- Losslessness

  The data contained in the original relation R with schema $\mathcal{R}$ must be reconstructible from new relations R_1, ..., R_n
  w.r.t. to the new schemas $\mathcal{R}_1$, .., $\mathcal{R}_n$

- Dependency preservation

  FDs $F_{\mathcal{R}}$ holding for $\mathcal{R}$ must be transferrable to schemas $\mathcal{R}_1$, ..., $\mathcal{R}_n$

Example: Beer drinkers

Beer drinkers		
Pub	*Guest*	*Beer*
Kowalski	Kemper	Pilsner
Kowalski	Eickler	Wheat beer
Innsteg	Kemper	Wheat beer

Beer drinkers		
Pub	*Guest*	*Beer*
Kowalski	Kemper	Pilsner
Kowalski	Eickler	Wheat beer
Innsteg	Kemper	Wheat beer

Visits	
Pub	*Guest*
Kowalski	Kemper
Kowalski	Eickler
Innsteg	Kemper

$\pi_{\ldots}$

$\bowtie$

Drinks	
Guest	*Beer*
Kemper	Pilsner
Eickler	Wheat beer
Kemper	Wheat beer

$\neq$

Visits ⋈ Drinks		
Pub	*Guest*	*Beer*
Kowalski	Kemper	Pilsner
Kowalski	Kemper	Wheat beer
Kowalski	Eickler	Wheat beer
Innsteg	Kemper	Pilsner
Innsteg	Kemper	Wheat beer

Decomposition can go wrong

Explanation

The Beer drinker example was a lossy decomposition. Only a non-trivial functional dependency holds

- {Pub, Guest} ⟶ {Beer}

However, non of the following losslessness-guaranteeing FDs hold:

- {Guest} ⟶ {Beer}
- {Guest} ⟶ {Pub}

In the context, people (in particular Kemper) drinks beer in different pubs, but in a certain pub always the same beer (such that the waiters are already instructed?)

Lossless decomposition

Parents		
Father	*Mother*	*Child*
Johann	Martha	Else
Johann	Maria	Theo
Heinz	Martha	Cleo

$\pi_{\text{Father, Child}}$

$\pi_{\text{Mother, Child}}$

Fathers	
Father	*Child*
Johann	Else
Johann	Theo
Heinz	Cleo

Mothers	
Mother	*Child*
Martha	Else
Maria	Theo
Martha	Cleo

Explanation

Parents: {[Father, Mother, Child]}
Fathers: {[Father, Child]}
Mothers: {[Mother, Child]}

Losslessness is guaranteed:
Not only one of the sufficient FDs hold, but even both

- {Child}⟶{Mother}
- {Child}⟶{Father}

{Kind} is naturally also a key for the relation Parents

Note: The decomposition of Parents is lossless, but otherwise rather useless because there are no redundancies (see normal forms)

Criteria for a lossless decomposition

$\mathcal{R} = \mathcal{R}1 \cup \mathcal{R}2$

- $R1 := \Pi_{\mathcal{R}1}(R)$
- $R2 := \Pi_{\mathcal{R}2}(R)$

Decomposition of $\mathcal{R}$ into $\mathcal{R}1$ and $\mathcal{R}2$ is lossless if for all instances R of $\mathcal{R}$:

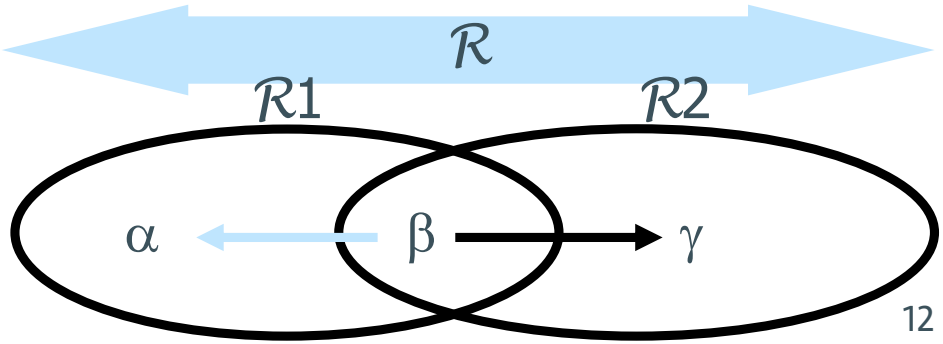
- $R = R1 \bowtie R2$

Sufficient condition of a lossless decomposition possibility:

One FD of the form

- $(\mathcal{R}1 \cap \mathcal{R}2) \rightarrow \mathcal{R}1$ or
- $(\mathcal{R}1 \cap \mathcal{R}2) \rightarrow \mathcal{R}2$

must hold



Dependency preservation

$\mathcal{R}$ decomposed into $\mathcal{R}_1$, ..., $\mathcal{R}_n$
$F_{\mathcal{R}} = (F_{\mathcal{R}1} \cup ... \cup F_{\mathcal{R}n})$ resp. $F_{\mathcal{R}}^+ = (F_{\mathcal{R}1} \cup ... \cup F_{\mathcal{R}n})^+$

Example for loss of dependencies
- Schema Post code directory: {[Street, Place, FState, Post code]}

Associated FDs
1. {Post code} ⟶ {Place, FState}
2. {Street, Place, FState} ⟶ {Post code}

Consition decompositions
- FD_1 Streets: {[Post code, Street]}
- FD_2 Places: {[Post code, Place, FState]}

FD_2 can neither be checked according to Streets nor to Places (both relational schemas contain two few attributes)

Decomposition of Post code directory

Post code directory

Place	***FState***	***Street***	*Post code*
Frankfurt	Hessen	Goethestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234

$\pi_{\text{Post code,Street}}$ $\pi_{\text{Place,PState,Post code}}$

Streets

Post code	***Streets***
15234	Goethestraße
60313	Goethestraße
60437	Galgenstraße

Places

Place	*FState*	***Post code***
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234

The FD {Street, Place, FState} ⟶ {Post ode} is no longer to be associated with any decompositon⟶ Insertion of inconsistent tuples is possible

Post code directory			
Place	**FState**	**Street**	Post code
Frankfurt	Hessen	Goethestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234

$\pi_{\text{Post code,Street}}$

$\pi_{\text{Place,PState,Post code}}$

Insertion of two tuples violoating FD
Place, FState, Street
⟶
Post code

Streets	
Post code	**Streets**
15234	Goethestraße
60313	Goethestraße
60437	Galgenstraße
15235	Goethestraße

Places		
Place	FState	**Post code**
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234
Frankfurt	Brandenburg	15235

Post code directory			
Place	***FState***	***Street***	*Post code*
Frankfurt	Hessen	Goethestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234
Frankfurt	Brandenburg	Goethestraße	15235

Insertion of two tuples violoating FD Place, FState, Street → Post code

Streets	
Post code	***Street***
15234	Goethestraße
60313	Goethestraße
60437	Galgenstraße
15235	Goethestraße

Places		
Place	*FState*	***Post code***
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234
Frankfurt	Brandenburg	15235

Evaluation criteria for relational schemas

- No redundancies in data
- If possible, check of FDs associated with relational schema realized with primary keys only (and not with computation-intensive joins)

→ Normal forms

First normal form (1 NF): "simple domains"

Example:

Parents		
Father	*Mother*	*Children*
Johann	Martha	{Else, Lucie}
Johann	Maria	{Theo, Josef}
Heinz	Martha	{Cleo}

1 NF

Parents		
Father	*Mother*	*Child*
Johann	Martha	Else
Johann	Martha	Lucie
Johann	Maria	Theo
Johann	Maria	Josef
Heinz	Martha	Cleo

Second normal form

A relational schema $\mathcal{R}$ with associated FDs $F_{\mathcal{R}}$ is in second normal form (2 NF) if every non-key attribute $A \in \mathcal{R}$ is fully functionally dependent from every candidate key of a relation with schema $\mathcal{R}$

Students and lectures is not in 2 NF

- {StudID} ⟶ {Name}
- {StudID} ⟶ {Semester}

Students and lectures			
StudID	***LectID***	*Name*	*Semester*
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

Again: Avoid redundancies in data

Example:

$\mathcal{R}$ = {[A, B, C ,D]}, F = {A ⟶ B, D ⟶ ABCD}, candidate key: {D}

R			
A	B	C	D
3	4	5	1
3	4	6	2

„Do not represent the same fact twice“

General case: If $\alpha \longrightarrow \beta \in F$ then: α should be a super key or FD need to be trivial

If this is not the case: Decompose relation (see below: lossless and dependency-preserving)


Functional dependencies
External checks?

Boyce-Codd Normal form (BCNF)

A Relational schema $\mathcal{R}$ is in BCNF if

- in addition to 2 NF
- for all FDs $\alpha \longrightarrow \beta$ associated with $\mathcal{R}$ at least one of the following conditions holds:
  - $\beta \subseteq \alpha$, i.e., that dependency is trivial or
  - α is super key of $\mathcal{R}$

Beispiel:

- Given is a relational schema with associated FDs:

Cities: {[Place, FState, Prime minister, Inhabitants]}

FDs:

{FState} ⟶ {Prime minister}
{Place, FState} ⟶ {Inhabitants}
{Prime Minister} ⟶ {FState}

- 2 NF?

Beispiel:

- Given is a relational schema with associated FDs:

Cities: {[Place, FState, Prime minister, Inhabitants]}

FDs:

{FState} ⟶ {Prima minister}
{Place, FState} ⟶ {Inhabitants}
{Prima Minister} ⟶ {FState}

- 2 NF? Yes
- BCNF?

Beispiel:

- Given is a relational schema with associated FDs:

Cities: {[Place, FState, Prime minister, Inhabitants]}

FDs:

{FState} ⟶ {Prima minister}
{Place, FState} ⟶ {Inhabitants}
{Prima Minister} ⟶ {FState}

- 2 NF? Yes
- BCNF? No, decomposition required

Example: Decomposition of the relation *Cities*

Cities: {[Place, FState, Prime minister, Inhabitants]}
FDs:

- {FState} ⟶ {Prime minister}
- {Place, FState} ⟶ {Inhabitants}
- {Prime minister} ⟶ {FState}

$\mathcal{R}_1$:

- **Governments:** {[FState, Prime minister]}
- **FDs:** { {FState} ⟶ {Prime minister}, {Prime minister} ⟶ {FState} }
- **Candidate keys:** κ = { {FState}, {Prime minister} }

$\mathcal{R}_2$:

- **Cities:** {[Place, FState, Inhabitants]}
- **FDs:** { {Place, FState} ⟶ {Inhabitants} }
- **Candidate keys :** κ = { {Place, FState} }

Summary and outlook

1. Introduction
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. Entity-relationship model (ER model)
3. Relational data model (RDM)
   - From ER to RDM
4. Relational schema design theory
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages