Part V

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Relational Database Design Theory

Educational Objective for Today ...

- Know how to refine the relational design

Educational Objective for Today . . .

- Know how to refine the relational design
- Understanding of normal forms

Educational Objective for Today . . .

- Know how to refine the relational design
- Understanding of normal forms
- Methodology and techniques for normalization

Target Model of the Logical Design

Relation Model

WINES

WineID	Name	Color	Vintage	Vineyard
1042	La Rose Grand Cru	Red	1998	Château La Rose
2168	Creek Shiraz	Red	2003	Creek
3456	Zinfandel	Red	2004	Helena
2171	Pinot Noir	Red	2001	Creek
3478	Pinot Noir	Red	1999	Helena
4711	Riesling Reserve	White	1999	Müller
4961	Chardonnay	White	2002	Bighorn

PRODUCER

Vineyard	District	Region
Creek	Barossa Valley	South Australia
Helena	Napa Valley	California
Château La Rose	Saint-Emilion	Bordeaux
Château La Pointe	Pomerol	Bordeaux
Müller	Rheingau	Hessen
Bighorn	Napa Valley	California

Terms of the Relational Model

Term	**Informal Meaning**
Attribute	Column of a table
Value domain	Possible values of an attribute
Attribute value	Element of a value domain
Relation schema	Set of attributes
Relation	Set of rows in a table
Tuple	Row in a table
Database schema	Set of relation schemas
Database	Set of relations (base relations)

Terms of the Relational Model /2

Term	Informal Meaning
Key	Minimal set of attributes, whose values uniquely identify a tuple in a table
Primary key	A key designated during database design
Foreign key	Set of attributes that are key in another relation
Foreign key constraint	All attribute values of the foreign key show up as keys in the other relation

Integrity Constraints

- Identifying set of attributes $K := \{B_1, \ldots, B_k\} \subseteq R$:

$$\forall t_1, t_2 \in r \left[t_1 \neq t_2 \quad \Longrightarrow \quad \exists B \in K : t_1(B) \neq t_2(B)\right]$$

- Key: is minimal identifying set of attributes
  - {`Name, Vintage, Vineyard`} and
  - {`WineID`} for `WINES`
- Prime attribute: element of a key
- Primary key: designated key
- Superkey: every superset of a key (= identifying set of attributes)
- Foreign key: $X(R_1) \rightarrow Y(R_2)$

$$\{t(X) | t \in r_1\} \subseteq \{t(Y) | t \in r_2\}$$

Relational Database Design

Relation with Redundancies

WINES

WineID	Name	...	Vineyard	District	Region
1042	La Rose Gr. Cru	...	Ch. La Rose	Saint-Emilion	Bordeaux
2168	Creek Shiraz	...	Creek	Barossa Valley	South Australia
3456	Zinfandel	...	Helena	Napa Valley	California
2171	Pinot Noir	...	Creek	Barossa Valley	South Australia
3478	Pinot Noir	...	Helena	Napa Valley	California
4711	Riesling Res.	...	Müller	Rheingau	Hessen
4961	Chardonnay	...	Bighorn	Napa Valley	California

Update Anomalies

- Insertion into the redundancy-containing relation `WINES`:

```
insert into WINES (WineID, Name, Color, Vintage,
     Vineyard, District, Region)
  values (4711, 'Chardonnay', 'White', 2004,
     'Helena', 'Rheingau', 'California')
```

  - WineID `4711` already assigned to another wine: violates FD `WineID` $\rightarrow$ `Name`
  - Up to now, vineyard `Helena` was located in Napa Valley: violates FD `Vineyard` $\rightarrow$ `District`
  - Rheingau is not located in California: violates FD `District` $\rightarrow$ `Region`
  - FD $=$ Functional Dependency (see next slides)
- Also: **`update`**- and **`delete`** anomalies

Functional Dependencies

- Functional dependency between two sets of attribute X and Y of a relation holds iff

> for each tuple of the relation, the attribute values of the X components determine the attribute values of the Y components.

- If two tuples have the same values for the X attributes, they also have the same values for all Y attributes.
- Notation for functional dependency (FD): $X \rightarrow Y$
- Example:

```
WineID   → Name, Vineyard
District → Region
```

- But not: `Vineyard → Name`

Keys as a Special Case

- For example on Slide 5-9

  `WineID` $\rightarrow$ `Name`, `Color`, `Vintage`, `Vineyard`, `District`, `Region`

- Always: `WineID` $\rightarrow$ `WineID`,
  then whole schema on the right side
- If left side minimal: Key
- Formally: X is key if FD $X \rightarrow R$ holds for relation schema R and X is minimal

Goal of database design: Transform all existing functional dependencies into "key dependencies", without losing semantic information

Deriving Functional Dependencies

Deriving FDs

r

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_1
a_4	b_1	c_1

- Satisfies $A \rightarrow B$ and $B \rightarrow C$
- Then $A \rightarrow C$ also holds
- Not derivable: $C \rightarrow A$ or $C \rightarrow B$

Deriving FDs /2

- If for f over R, it holds that $\textbf{SAT}_R(F) \subseteq \textbf{SAT}_R(f)$, then F implies the FD f (short: $F \models f$)
- Previous example:

$$F = \{A \rightarrow B, B \rightarrow C\} \models A \rightarrow C$$

- Computing the closure: Determine all functional dependencies that can be derived from a given set of FDs
- Closure $F_R^+ := \{f \mid (f \text{ FD over } R) \wedge F \models f\}$
- Example:

$$\begin{aligned} \{A \rightarrow B, B \rightarrow C\}^+ \quad = \quad & \{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C, A \rightarrow BC, \ldots, \\ & AB \rightarrow AB, \ldots\} \end{aligned}$$

Derivation Rules

F1	Reflexivity	$X \supseteq Y$	$\Longrightarrow$	$X \rightarrow Y$
F2	Augmentation	$\{X \rightarrow Y\}$	$\Longrightarrow$	$XZ \rightarrow YZ$ and $XZ \rightarrow Y$
F3	Transitivity	$\{X \rightarrow Y, Y \rightarrow Z\}$	$\Longrightarrow$	$X \rightarrow Z$
F4	Decomposition	$\{X \rightarrow YZ\}$	$\Longrightarrow$	$X \rightarrow Y$
F5	Union	$\{X \rightarrow Y, X \rightarrow Z\}$	$\Longrightarrow$	$X \rightarrow YZ$
F6	Pseudo-transitivity	$\{X \rightarrow Y, WY \rightarrow Z\}$	$\Longrightarrow$	$WX \rightarrow Z$

F1-**F3** known as Armstrong axioms (sound, complete)

- *Sound*: Rules do not derive FDs that are not logically implied
- *Complete*: All implied FDs are derived
- *Independent* (i.e., minimal w.r.t.[1] $\subseteq$): No rule can be omitted

[1] w.r.t. = with respect to

Alternative Set of Rules

- B-Axioms or RAP-rules

R Reflexivity $\{\} \implies X \rightarrow X$

A Accumulation $\{X \rightarrow YZ, Z \rightarrow AW\} \implies X \rightarrow YZA$

P Projectivity $\{X \rightarrow YZ\} \implies X \rightarrow Y$

- Rule set is complete because it allows to derive the Armstrong axioms

Membership Problem

Can a certain FD $X \rightarrow Y$ be derived from a given set F, i.e., is it implied by F?

Membership problem: "$X \rightarrow Y \in F^+$?"

- Closure over a set of attributes X w.r.t. F is $X_F^+ := \{A \mid X \rightarrow A \in F^+\}$
- Membership problem can be solved in linear time by solving the modified problem

Membership problem (2): "$Y \subseteq X_F^+$?"

Algorithm CLOSURE

- Compute X_F^+, the closure of X w.r.t. F

```
CLOSURE(F, X):
   X+ := X
   repeat
      X̄+ := X+  /* R-rule */
      forall FDs Y → Z ∈ F
         if Y ⊆ X+ then X+ := X+ ∪ Z  /* A-rule */
   until X+ = X̄+
   return X+

MEMBER(F, X → Y):   /* Test if X → Y ∈ F+ */
   return Y ⊆ CLOSURE(F, X)  /* P-rule */
```

- Example: $A \to C \in \{\underbrace{A \to B}_{f_1}, \underbrace{B \to C}_{f_2}\}^+$?

Example for Algorithm CLOSURE

- Example: $A \rightarrow C \in \{\underbrace{A \rightarrow B}_{f_1}, \underbrace{B \rightarrow C}_{f_2}\}^+?$

Algorithm:

1. Initialize X as $\{A\}$
2. First run of the loop: $X = \{A, B\}$
3. Second run of the loop: $X = \{A, B, C\}$
4. Third run: no change - stop
5. Test whether C is in X

Application: Minimal Cover

... to minimize a set of FDs

forall FD $X \rightarrow Y \in F$ */* Left reduction */*
    forall $A \in X$ */* A superflous? */*
        if $Y \subseteq \textsc{Closure}(F, X - \{A\})$
        then replace $X \rightarrow Y$ with $(X - A) \rightarrow Y$ in F

forall remaining FD $X \rightarrow Y \in F$ */* Right reduction */*
    forall $B \in Y$ */* B superflous? */*
        if $B \subseteq \textsc{Closure}(F - \{X \rightarrow Y\} \cup \{X \rightarrow (Y - B)\}, X)$
        then replace $X \rightarrow Y$ with $X \rightarrow (Y - B)$

Eliminate FDs of the form $X \rightarrow \emptyset$
Combine FDs of the form $X \rightarrow Y_1, X \rightarrow Y_2, \ldots$ into $X \rightarrow Y_1 Y_2 \ldots$

Normal Forms

Normal Forms . . .

- . . . determine properties of relation schemata
- . . . forbid certain combinations of functional dependencies in relations
- . . . should prevent redundancies and anomalies

First Normal Form

- Allows only *atomic* attributes in relation schemas, i.e., only elements of standard datatypes, such as **`integer`** or **`string`**, are allowed as attribute values, but not **`array`** or **`set`**
- Not in 1NF:

Vineyard	**District**	**Region**	**WName**
Ch. La Rose	Saint-Emilion	Bordeaux	La Rose Grand Cru
Creek	Barossa Valley	South Australia	Creek Shiraz, Pinot Noir
Helena	Napa Valley	California	Zinfandel, Pinot Noir
MÃ¼ller	Rheingau	Hessen	Riesling Reserve
Bighorn	Napa Valley	California	Chardonnay

First Normal Form /2

- In first normal form:

Vineyard	District	Region	WName
Ch. La Rose	Saint-Emilion	Bordeaux	La Rose Grand Cru
Creek	Barossa Valley	South Australia	Creek Shiraz
Creek	Barossa Valley	South Australia	Pinot Noir
Helena	Napa Valley	California	Zinfandel
Helena	Napa Valley	California	Pinot Noir
MÃ¼ller	Rheingau	Hessen	Riesling Reserve
Bighorn	Napa Valley	California	Chardonnay

Second Normal Form

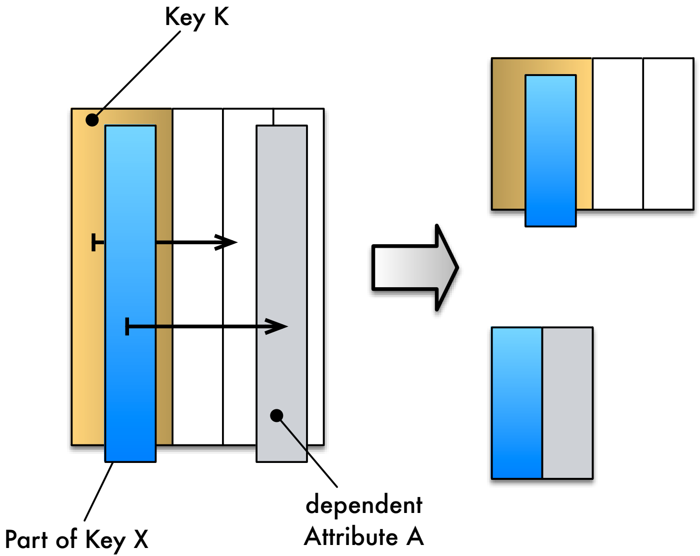
- Partial dependency: An attribute functionally depends on only part of the key

Name	Vineyard	Color	District	Region	Price
La Rose Grand Cru	Ch. La Rose	Red	Saint-Emilion	Bordeaux	39.00
Creek Shiraz	Creek	Red	Barossa Valley	South Australia	7.99
Pinot Noir	Creek	Red	Barossa Valley	South Australia	10.99
Zinfandel	Helena	Red	Napa Valley	California	5.99
Pinot Noir	Helena	Red	Napa Valley	California	19.99
Riesling Reserve	Müller	Weiš	Rheingau	Hessen	14.99
Chardonnay	Bighorn	Weiš	Napa Valley	California	9.90

f_1: Name, Vineyard → Price
f_2: Name → Color
f_3: Vineyard → District, Region
f_4: District → Region

- Second normal form eliminates such partial dependencies for non-key attributes

Elimination of Partial Dependencies



Second Normal Form /2

- Example relation in 2NF

  R1(<u>Name, Vineyard</u>, Price)

  R2(<u>Name</u>, Color)

  R3(<u>Vineyard</u>, District, Region)

Third Normal Form

- Eliminates transitive dependencies (in addition to the other kinds of dependencies)
- For instance, `Vineyard` $\rightarrow$ `District` and `District` $\rightarrow$ `Region` in relation on Slide 5-26
- Note: 3NF only considers non-key attributes as endpoints of transitive dependencies

Example Relation

Name	Vineyard	Color	District	Region	Price
La Rose Grand Cru	Ch. La Rose	Red	Saint-Emilion	Bordeaux	39.00
Creek Shiraz	Creek	Red	Barossa Valley	South Australia	7.99
Pinot Noir	Creek	Red	Barossa Valley	South Australia	10.99
Zinfandel	Helena	Red	Napa Valley	California	5.99
Pinot Noir	Helena	Red	Napa Valley	California	19.99
Riesling Reserve	Müller	Weiß	Rheingau	Hessen	14.99
Chardonnay	Bighorn	Weiß	Napa Valley	California	9.90

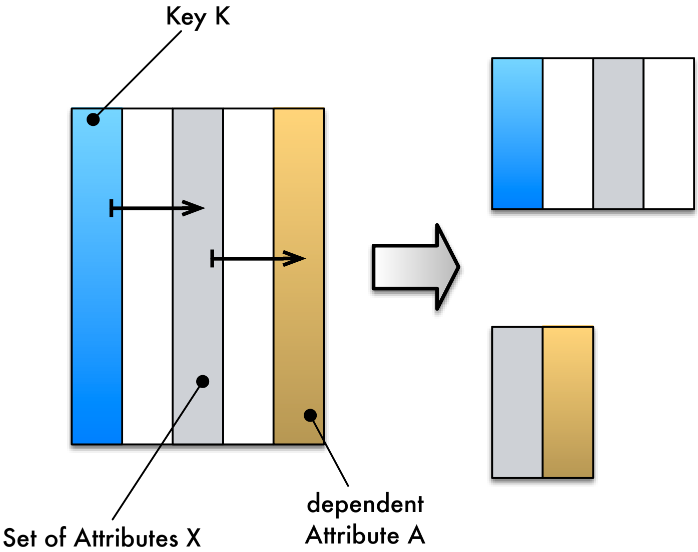
f_1: Name, Vineyard → Price

f_2: Name → Color

f_3: Vineyard → District, Region

f_4: District → Region

Elimination of Transitive Dependencies



Third Normal Form /2

- Result in 2NF
  - Example relation in 2NF

    ```
    R1(Name, Vineyard, Price)
    R2(Name, Color)
    R3(Vineyard, District, Region)
    ```
- Transitive dependency in `R3`, i.e., `R3` violates 3NF
- Example relations in 3NF

  ```
  R3_1(Vineyard, District)
  R3_2(District, Region)
  ```

Third Normal Form: Formally

- Relation schema R, $X \subseteq R$ and F is an FD set over R

- $A \in R$ is called transitively dependent on X w.r.t. F if and only if there is a $Y \subseteq R$ for which it holds that $X \rightarrow Y, Y \not\rightarrow X, Y \rightarrow A, A \notin XY$
- Extended relation schema $\mathcal{R} = (R, \mathcal{K})$ is in 3NF w.r.t. F if and only if

$$\nexists A \in R : \quad A \text{ is non-prime attribute in } R$$
$$\wedge\ A \text{ transitively dependent on a } K \in \mathcal{K} \text{ w.r.t. } F.$$

Boyce-Codd Normal Form

- Stronger version of 3NF: Elimination of transitive dependencies also between prime attributes

Name	Vineyard	Dealer	Price
La Rose Grand Cru	Château La Rose	Weinkontor	39.90
Creek Shiraz	Creek	Wein.de	7.99
Pinot Noir	Creek	Wein.de	10.99
Zinfandel	Helena	GreatWines.com	5.99
Pinot Noir	Helena	GreatWines.com	19.99
Riesling Reserve	MÃ¼ller	Weinkeller	19.99
Chardonnay	Bighorn	Wein-Dealer	9.90

- FDs:

$$\begin{aligned}\texttt{Name, Vineyard} &\rightarrow \texttt{Price}\\ \texttt{Vineyard} &\rightarrow \texttt{Dealer}\\ \texttt{Dealer} &\rightarrow \texttt{Vineyard}\end{aligned}$$

- Candidate keys: { `Name, Vineyard` } and { `Name, Dealer` }
- Example relation meets 3NF but not BCNF

Boyce-Codd-Normalform /2

- Extended relation schema $\mathcal{R} = (R, \mathcal{K})$, FD set F
- BCNF formally:

$$\nexists A \in R : A \text{ transitively depends on a } K \in \mathcal{K} \text{ w.r.t. } F.$$

- Schema in BCNF:

  WINES($\underline{\texttt{Name, Vineyard}}$, Price)
  WINE_TRADE($\underline{\texttt{Vineyard}}$, Dealer)

- However, BCNF may violate dependency preservation, therefore often stop at 3NF

Minimality

- Avoid global redundancies
- Meet other criteria (such as normal forms) with as few schemas as possible
- Example: Set of attributes ABC, set of FDs $\{A \rightarrow B, B \rightarrow C\}$
- Database schema in third normal form:

$$S = \{(AB, \{A\}), (BC, \{B\})\}$$

$$S' = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\}$$

Redundancies in S'

Schema Properties

Identifier	Schema Property	Key Points
	1NF	Only atomic attributes
	2NF	No non-prime attribute that partially depends on a key
S1	3NF	No non-prime attribute that transitively depends on a key
	BCNF	No attribute that transitively depends on a key
S2	Minimality	Minimal number of relation schemas that satisfies the other properties

Transformation Properties

Transformation Properties

- When decomposing a relation in multiple relations, care must be taken that ...
  1. ... only semantically sensible and consistent application data is presented (dependency preservation), and
  2. ... all application data can be derived from the base relations (lossless-join decomposition)

Dependency Preservation

- Dependency preservation: A set of dependencies can be transformed into an equivalent second set of dependencies
- More specifically: into the set of key dependencies because these can be validated efficiently by the database system
  - The set of dependencies shall be equivalent to the set of key constraints in the resulting database schema.
  - Equivalence ensures that, on a semantic level, the key dependencies express the exact same integrity constraints as the functional and other dependencies did before.

Dependency Preservation: Example

- Decomposition of the relation schema `WINES` (Slide 5-26) into 3NF:

  R1(<u>Name, Vineyard</u>, Price)
  R2(<u>Name</u>, Color)
  R3_1(<u>Vineyard</u>, District)
  R3_2(<u>District</u>, Region)

  with key dependencies

  $$\begin{aligned} \texttt{Name, Vineyard} &\rightarrow \texttt{Price} \\ \texttt{Name} &\rightarrow \texttt{Color} \\ \texttt{Vineyard} &\rightarrow \texttt{District} \\ \texttt{District} &\rightarrow \texttt{Region} \end{aligned}$$

- Equivalent to FDs $f_1 \ldots f_4$ (Slide 5-26) $\rightsquigarrow$ dependency-preserving

Dependency Preservation: Example /2

- Zip code (a.k.a. postal code) structure of the Deutsche Post

  `ADDRESS(ZIP (Z), City (C), Street (S), Street Number (N))`

  and functional dependencies F

  $$\mathtt{CSN} \rightarrow \mathtt{Z},\ \mathtt{Z} \rightarrow \mathtt{C}$$

- Candidate keys: `CSN` and `ZSN` $\rightsquigarrow$ 3NF
- Does not meet BCNF (because $\mathtt{ZSN} \rightarrow \mathtt{Z} \rightarrow \mathtt{C}$): therefore decomposition of `ADDRESS`
- But: every decomposition would destroy $\mathtt{CSN} \rightarrow \mathtt{Z}$
- Set of resulting FDs is not equivalent to F, the decomposition is therefore not dependency-preserving

Dependency Preservation: Formally

- Locally extended database schema $S = \{(R_1, \mathcal{K}_1), \ldots, (R_p, \mathcal{K}_p)\}$; a set F of local dependencies

S fully characterizes F (or: is dependency-preserving w.r.t. F) if and only if

$$F \equiv \{K \rightarrow R \mid (R, \mathcal{K}) \in S, K \in \mathcal{K}\}$$

Lossless-Join Decomposition

- In order to satisfy the criteria of the normal forms, relation schemas sometimes have to be decomposed into smaller relation schemas
- In order to restrict to “sensible” decomposition, require that the original relation can be recreated from the decomposed relations using a natural join
  $\leadsto$ lossless-join decomposition

Lossless-Join Decomposition: Examples

- Decompose the relation schema $R = ABC$ into

$$R_1 = AB \text{ and } R_2 = BC$$

- Decomposition is not join-lossless given the dependencies

$$F = \{A \to B, C \to B\}$$

- In contrast, the decomposition is join-lossless given the dependencies

$$F' = \{A \to B, B \to C\}$$

Lossless-Join Decomposition

- Original relation:

A	B	C
1	2	3
4	2	3

- Decomposition:

A	B
1	2
4	2

B	C
2	3

- Join (join-lossless):

A	B	C
1	2	3
4	2	3

Non-Join-Lossless Decomposition

- Original relation:

A	B	C
1	2	3
4	2	5

- Decomposition:

A	B
1	2
4	2

B	C
2	3
2	5

- Join (not join-lossless):

A	B	C
1	2	3
4	2	5
1	2	5
4	2	3

Lossless-Join Decomposition: Formally

The decomposition of a set of attributes X in $X_1, \ldots, X_p$ with $X = \bigcup_{i=1}^{p} X_i$ is called a lossless-join decomposition under a set of dependencies F over X if and only if

$$\forall r \in \mathbf{SAT}_X(F) : \pi_{X_1}(r) \bowtie \cdots \bowtie \pi_{X_p}(r) = r$$

holds.

- Simple criterion for a join-lossless decomposition into two relation schemas: Decomposition of X into X_1 and X_2 is join-lossless under F, if $X_1 \cap X_2 \rightarrow X_1 \in F^+$ or $X_1 \cap X_2 \rightarrow X_2 \in F^+$

Transformation Properties

Identifier	Transformation Property	Key Points
T1	Dependency Preservation	All given dependencies are represented by keys
T2	Lossless-Join Decomposition	Original relations can be recreated by joining base relations

Design Methods and Decomposition

Design Methods: Goals

- Given: Universe $\mathcal{U}$ and set of FDs F
- Locally extended database schema $S = \{(R_1, \mathcal{K}_1), \ldots, (R_p, \mathcal{K}_p)\}$ compute with
  - **T1**: Dependency Preservation (S fully characterizes F)
  - **S1**: S is in 3NF under F
  - **T2**: Lossless-Join Decomposition
  - **S2**: Minimality, i.e.,
    $\nexists S' : S'$ satisfies **T1**, **S1**, **T2** and $|S'| < |S|$

Design Methods: Example

- Database schemas badly designed if only one of these four criteria is not fulfilled
- Example: $S = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\}$ fulfills $\mathbf{T1}$, $\mathbf{S1}$ and $\mathbf{T2}$ under $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
  in third relation AC tuple redundant or inconsistent
- Correct: $S' = \{(AB, \{A\}), (BC, \{B\})\}$

Decomposition

- Given: Initial universal relation schema $\mathcal{R} = (\mathcal{U}, \mathcal{K}(F))$ with all attributes and a set of implied keys implied by FDs F over R
  - Set of attributes $\mathcal{U}$ and set of FDs F
  - Find all $K \rightarrow \mathcal{U}$ with K minimal, for which $K \rightarrow \mathcal{U} \in F^+$ ($\mathcal{K}(F)$)
- Wanted: Decomposition into $D = \{\mathcal{R}_1, \mathcal{R}_2, \dots\}$ of 3NF-relation schemas

Decomposition: Algorithm

DECOMPOSE($\mathcal{R}$)
  Set $D := \{\mathcal{R}\}$
  while $\mathcal{R}' \in D$, does not meet 3NF
    / Find attribute A that is transitively dependent on K */*
    if Key K with $K \rightarrow Y, Y \not\rightarrow K, Y \rightarrow A, A \notin KY$
    then
      / Decompose relation schema R w.r.t. A */*
      $R_1 := R - A$, $R_2 := YA$
      $\mathcal{R}_1 := (R_1, \mathcal{K})$, $\mathcal{R}_2 := (R_2, \mathcal{K}_2 = \{Y\})$
      $D := (D - \mathcal{R}') \cup \{\mathcal{R}_1\} \cup \{\mathcal{R}_2\}$
    end if
  end while
  return D

Decomposition: Example

- Initial relation schema $R = ABC$
- Functional dependencies $F = \{A \rightarrow B, B \rightarrow C\}$
- Keys $K = A$

Decomposition: Example /2

- Initial relation schema R with `Name`, `Vineyard`, `Price`, `Color`, `District`, `Region`
- Functional dependencies

f_1: `Name, Vineyard` $\rightarrow$ `Price`
f_2: `Name, Vineyard` $\rightarrow$ `Vineyard`
f_3: `Name, Vineyard` $\rightarrow$ `Name`
f_4: `Name` $\rightarrow$ `Color`
f_5: `Vineyard` $\rightarrow$ `District, Region`
f_6: `District` $\rightarrow$ `Region`

- ... results in 4 relations, all in 3NF

Decomposition: Assessment

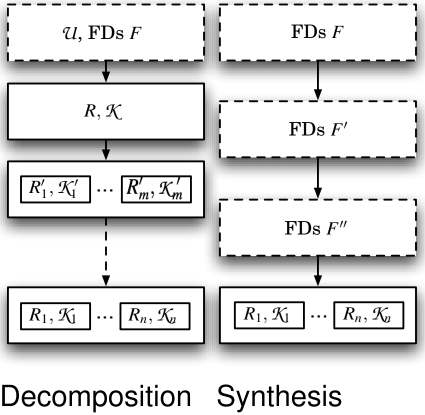
- Advantages: 3NF, lossless-join decomposition
- Disadvantages: other criteria not fulfilled, depends on order, NP-hard (search for keys)

Synthesis Algorithm

Synthesis Method

- Principle: Synthesis transforms original set of FDs F into a resulting set of key dependencies G such that $F \equiv G$
- “Dependency Preservation” built into the method
- 3NF and minimality also achieved, independent of order
- Computational complexity: quadratic

Comparison Decomposition — Synthesis



Synthesis Method: Algorithm

- Given: Relation schema R mit FDs F
- Wanted: Join-lossless and dependency-preserving decomposition into $R_1, \ldots R_n$ where all R_i are in 3NF
- Algorithm:

SYNTHESIZE(F):

$\hat{F} :=$ **MINIMALCOVER**(F) */* Determine minimal cover */*

Compute equivalence classes C_i of FDs from $\hat{F}$ with equal or equivalent left sides, i.e., $C_i = \{X_i \rightarrow A_{i1}, X_i \rightarrow A_{i2}, \ldots\}$

For each equivalence class C_i create a schema of the form $R_{C_i} = \{X_i \cup \{A_{i1}\} \cup \{A_{i2}\} \cup \ldots\}$

if none of the schemas R_{C_i} contains a key from R

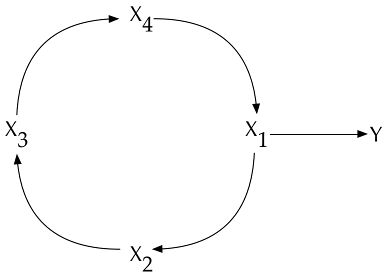
then create additional relation schema R_K with attributes from R, which form the key

return $\{R_K, R_{C_1}, R_{C_2}, \ldots\}$

Equivalence Classes

- Class of FDs whose left sides are equal or equivalent
- Left sides are equivalent if they determine each other functionally
- Relation schema R with $X_i, Y \subset R$, set of FDs
  $X_i \rightarrow X_j$ and $X_i \rightarrow Y$ with $1 \leq i, j \leq n$ can be expressed as

$$(X_1, X_2, \ldots, X_n) \rightarrow Y$$



Equivalence Classes: Example

- Set of FDs

$$F = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C, B \rightarrow A, C \rightarrow E\}$$

- Minimal cover

$$\hat{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow A, C \rightarrow E\}$$

- Aggregation into equivalence classes

$$\begin{aligned} C_1 &= \{A \rightarrow B, B \rightarrow C, B \rightarrow A\} \\ C_2 &= \{C \rightarrow E\} \end{aligned}$$

- Result of synthesis

$$(ABC, \{\{A\}, \{B\}\}), (CE, \{C\})$$

Achieving a Lossless-Join Decomposition

- Achieve a lossless-join decomposition by a simple “trick”:
  - Extend the original set of FDs F with $\mathcal{U} \rightarrow \delta$, where δ is a dummy attribute
  - δ is removed after synthesis
- Example: $\{A \rightarrow B, C \rightarrow E\}$
  - Result of synthesis $(AB, \{A\}), (CE, \{C\})$ is not lossless, because the universal key is not part of any schema
  - Dummy-FD $ABCE \rightarrow \delta$; reduced to $AC \rightarrow \delta$
  - Yields third relation schema

$$(AC, \{AC\})$$

Synthesis: Example

- Relation schema and set of FDs from Slide 5-56
- Steps
  1. Minimal cover: removal of f_2, f_3 as well as `Region` in f_5
  2. Equivalence classes:

$$
\begin{aligned}
C_1 &= \{\texttt{Name}, \texttt{Vineyard} \rightarrow \texttt{Price}\} \\
C_2 &= \{\texttt{Name} \rightarrow \texttt{Color}\} \\
C_3 &= \{\texttt{Vineyard} \rightarrow \texttt{District}\} \\
C_4 &= \{\texttt{District} \rightarrow \texttt{Region}\}
\end{aligned}
$$

  3. Derivation of relation schemas

Example for Synthesis

- Initial relation schema R with `Name`, `Vineyard`, `Price`, `Color`, `District`, `Region`
- Functional dependencies

$$
\begin{array}{lll}
f_1: & \texttt{Name, Vineyard} & \rightarrow \texttt{Price} \\
f_2: & \texttt{Name, Vineyard} & \rightarrow \texttt{Vineyard} \\
f_3: & \texttt{Name, Vineyard} & \rightarrow \texttt{Name} \\
f_4: & \texttt{Name} & \rightarrow \texttt{Color} \\
f_5: & \texttt{Vineyard} & \rightarrow \texttt{District, Region} \\
f_6: & \texttt{District} & \rightarrow \texttt{Region}
\end{array}
$$

- Resulting equivalence classes
  - { `Name`, `Vineyard` }
  - { `Name` }
  - { `Vineyard` }
  - { `District` }
- Same result as for the decomposition

Summary

- Functional dependencies
- Normal forms (1NF – 3NF, BCNF)
- Dependency preservation and lossless-join decomposition
- Design methods

Control Questions

- What is the goal of normalizing relational schemas?

Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?

Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?
- What is the difference between 3NF and BCNF?

Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?
- What is the difference between 3NF and BCNF?
- What does it mean for a decomposition to be *dependency-preserving?*
  What is a *lossless-join decomposition?*