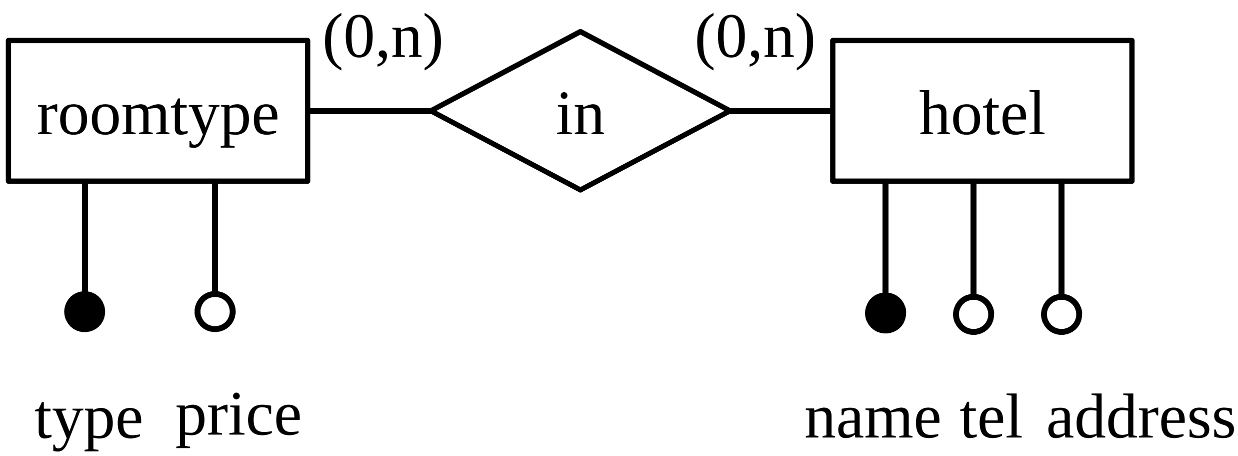
Normalization

Stéphane Bressan

National University of Singapore



We have the following functional dependencies:

`{type} → {price}`

`and`

`{name} → {tel, address}`

One Table

type	price	name	tel	address
superior	145	Sangria Clarke Quay	65166516	1 Clarke Quay
standard	75	Sangria Clarke Quay	65166516	1 Clarke Quay
suite	250	Sangria Clarke Quay	65166516	1 Clarke Quay
superior	145	Sangria Holland V	65166516	13 Holland Drive
standard	76	Sangria Holland V	65166516	13 Holland Drive
suite	250	Sangria Holland V	65165555	13 Holland Drive
junior suite	200	Sangria Holland V	65165555	13 Holland Drive
executive	175			

All the problems are caused by the functional dependencies. They can be solved using the functional dependencies.

The solution is to decompose into several tables

R1

name	tel	address
Sangria Clarke Quay	65166516	1 Clarke Quay
Sangria Holland V	65165555	13 Holland Drive

R2

type	price
superior	145
standard	75
suite	250
junior suite	200
executive	175

R3

type	name
superior	Sangria Clarke Quay
standard	Sangria Clarke Quay
suite	Sangria Clarke Quay
superior	Sangria Holland V
standard	Sangria Holland V
suite	Sangria Holland V
junior suite	Sangria Holland V

Decomposition

A decomposition of a relation scheme `R` is a set of relation scheme `Ri` such that:

$$\cup_i R_i = R$$

Namely, we have all the attributes.

The tables `Ri` are called 'fragments'

Lossless Decomposition

The decomposition is lossless if we can recover the initial table:

```
SELECT *
FROM (R1 NATURAL JOIN  R2 NATURAL JOIN R3)
```

Some attributes must be repeated in two fragments to allow a meaningful `JOIN`.

Otherwise it is lossy.

Lossless Decomposition

If we decompose a relation `R` into `R1` and `R2` where $R1 \cap R2 = X$ and $X \rightarrow R2$ then the decomposition is lossless.

Indeed, when we join `R1` and `R2` on `X`, for every tuple in `R1` there is only one tuple in `R2`.

The decomposition is lossless

{type} → {price}

and {name} →{tel, address}

R1

name	tel	address
Sangria Clarke Quay	65166516	1 Clarke Quay
Sangria Holland V	65165555	13 Holland Drive

R2

type	price
superior	145
standard	75
suite	250
junior suite	200
executive	175

R3

type	name
superior	Sangria Clarke Quay
standard	Sangria Clarke Quay
suite	Sangria Clarke Quay
superior	Sangria Holland V
standard	Sangria Holland V
suite	Sangria Holland V
junior suite	Sangria Holland V

The decomposition is lossy

Flight Number	Departure time	Arrival time	Origin	Destination
SG12	12h00	13h00+	SIN	CDG
TG414	15h50	16h30	SIN	JKT
TG415	12h00	14h20	BKK	SIN

Flight Number	Departure time	Origin
SG12	12h00	SIN
TG414	15h50	SIN
TG415	12h00	BKK

Departure time	Arrival time	Destination
12h00	13h00+	CDG
15h50	16h30	JKT
12h00	14h20	SIN

Dependency Preserving Decomposition

`R` with `F` is decomposed into a set of relation schemes `Ri`.

Each `Ri` inherits of a set of functional dependencies `Fi`.

The decomposition is dependency preserving the set of functional dependencies on the new scheme is equivalent to the original set of functional dependencies.

$$(\cup_i Fi)^+ = F^+$$

A Note on Projecting Functional Dependencies

`R` with `F` is decomposed into a set of relation schemes `Ri`.

Each `Ri` inherits a set of functional dependencies `Fi`. The functional dependencies in `Fi` are called projected functional dependencies.

We need to choose a cover (preferably minimal) of the set of dependencies in `F+ (it may not always work to look at F only)` that only involve attributes of `Ri`.

Example of Projected Functional Dependencies (1)

R = {A, B, C, D}

F = {{D} → {B, C}, {C} → {D}, {B} → {A}}

We decompose into

R1 = {A, B}

F1 = {{B} → {A}}

R2 = {B, C, D}

F2 = {{D} → {B, C}, {C} → {D}}

(F1 ∪ F2)+ = F+

(by the way, the decomposition is lossless)

Example of Projected Functional Dependencies (2)

R = {A, B, C, D}

F = {{A, B} → {C}, {C} → {A}}

We decompose into

R1 = {A, C}

F1 = {{C} → {A}}

R2 = {B, C, D}

F2 = ∅

We lose {A, B} → {C}

$(F1 \cup F2)^+ \neq F^+$

(by the way, the decomposition is lossless)

Example of Projected Functional Dependencies (3)

R = {A, B, C, D}

F = {{A} → {B}, {B} → {C}, {C} → {D}, {D} → {A}}

We decompose into

R1 = {A, D}

F1 = {{D} → {A}, {A} → {D}}

R2 = {A, B, C}

F2 = {{A} → {B}, {B} → {C}, {C} → {A}}

Some functional dependencies are in F+ but not in F. It seems that we lost {C} → {D} and {D} → {A} but we did not.

They can be found in (F1 ∪ F2)+.

(F1 ∪ F2)+ = F+.

(by the way, the decomposition is lossless)

Too Much Decomposition

It might be tempting to decompose to the extreme

Evaluation of queries may be inefficient since it will involve combining several relations

Flight Number	Departure time	Arrival time	Origin	Destination
SG12	12h00	13h00+	SIN	CDG
TG414	15h50	16h30	SIN	JKT
TG415	12h00	14h20	BKK	SIN

Flight Number	Departure time	Origin
SG12	12h00	SIN
TG414	15h50	SIN
TG415	12h00	BKK

Flight Number	Arrival time	Destination
SG12	13h00+	CDG
TG414	16h30	JKT
TG415	14h20	SIN

Objectives

We want to decompose into a lossless, dependency preserving decomposition in BCNF if possible or into a lossless, dependency preserving decomposition in 3NF, otherwise.

It is always possible to decompose into a lossless, decomposition in BCNF. We will learn a decomposition algorithm to do so. However, there may not always exist a dependency preserving decomposition.

It is always possible to decompose into a lossless, dependency preserving decomposition in 3NF. We will learn a synthesis algorithm to do so. Mot of the time it will give us a lossless, dependency preserving decomposition in BCNF.

Looking for a "Good" Design

The designer needs guidelines:

- **Normalization theory**
  - **Minimal redundancy and no anomalies**
  - **Lossless decompositions**
  - **Dependency preserving decompositions**

But ultimately the designer needs to look at the workload (the queries and their efficiency requirement)

Learning Objectives

Be able to decompose and synthesize a schema into a lossless and dependency preserving (if possible) BCNF and 3NF decomposition.

Decomposition into BCNF

```
Let S be the initial set of schemes Ri with Fi
Until all relation schemes in S are in BCNF
for each Ri in S
      if X → Y in F+ violates BCNF for Ri
      then let S be
            (S - {Ri}) ∪ {X+, (R-X+) ∪ X}
      endfor
enduntil
```

Decomposition into BCNF

```
R = {A,B,C,D,E}
{C, D} → {E} violates BCNF
and {C, D}+ = {C,D,E,F}
```

A	B	C	D	E	F

```
{C, D}+
=
{C,D,E,F}
```

C	D	E	F

```
R - {C, D}+ ∪ {C,D}
=
{A,B,C,D}
```

A	B	C	D

Decomposition into BCNF

The different possible orders* in which we may consider the dependencies violating BCNF in the algorithm application may lead to different decompositions

orders in which we consider the constraints violating the BCNF condition

An Example

`R = {A, B, C, D, E}`

`F = {{A, B}` → `{C}, {C}` → `{D}, {D}` →`{E}}`

The candidate key `is {A, B}`

`{C}` → `{D}` is not trivial. {C} is not a superkey.

`R` with `F` is not in BCNF.

Let us decompose `R` using `{C}` → `{D}.`

(we could also use `{D}` →`{E}`)

(Cont.)

$R1 = \{C\}+ = \{C, D\}$

$F1 = \{\{C\} \rightarrow \{D\}\}$

The candidate key is $\{C\}$

$R1$ with $F1$ is in BCNF

$R2 = R - \{C\}+ \cup \{C\} = \{A, B, C, E\}$

$F2 = \{\{A, B\} \rightarrow \{C\}, \{C\} \rightarrow \{E\}\}$

The candidate key is $\{A, B\}$

$R2$ with $F2$ is in not in BCNF because $\{C\} \rightarrow \{E\}$ is not trivial and {C} is not a superkey.

We could continue but we already lost $\{D\} \rightarrow \{E\}$...

(Cont.)

Let us try again.

Let us decompose `R` using $\{D\} \rightarrow \{E\}$.

$R1 = \{D\}+ = \{D, E\}$

$F1 = \{\{D\} \rightarrow \{E\}\}$

The candidate key is $\{D\}$

`R1` with `F1` is in BCNF

$R2 = R - \{D\}+ \cup \{D\} = \{A, B, C, D\}$

$F2 = \{\{A, B\} \rightarrow \{C\}, \{C\} \rightarrow \{D\}\}$

The candidate key is $\{A, B\}$

`R2` with `F2` is in not in BCNF because $\{C\} \rightarrow \{D\}$ is not trivial and {C} is not a superkey. We continue to decompose `R2`.

(Cont.)

We keep `R1`, which is in BCNF.

We decompose `R2`

`R2.1 = {C}+ = {C, D}`

`F2.1 = {{C} → {D}}`

The candidate key `is {C}`

`R2.1` with `F2.1` is in BCNF

`R2.2 = R2 – {C}+ ∪ {C} = {A, B, C}`

`F2 = {{A, B} → {C}}`

The candidate key `is {A, B}`

`R2.2` with `F2.2` is in BCNF.

(Cont.)

We are DONE!

`R1 = {D, E}` with `F1 = {{D} → {E}}` is in BCNF

`R2.1 = {C, D}` with `F2.1 = {{C} → {D}}` is in BCNF

`R2.2 ={A, B, C}` with `F2 = {{A, B} → {C}}` is in BCNF

We have a lossless, dependency preserving decomposition in BCNF.

Decomposition

name	tel	address
Sangria Clarke Quay	65166516	1 Clarke Quay
Sangria Holland V	65165555	13 Holland Drive

type	price
superior	145
standard	75
suite	250
junior suite	200
executive	175

type	name
superior	Sangria Clarke Quay
standard	Sangria Clarke Quay
suite	Sangria Clarke Quay
superior	Sangria Holland V
standard	Sangria Holland V
suite	Sangria Holland V
junior suite	Sangria Holland V

Another Example

$R = \{A, B, C, D\}$

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

The candidate keys are $\{A, D\}$ and $\{B, D\}$

$\{B\} \rightarrow \{A\}$ is **not trivial**. $\{B\}$ is **not a superkey**.

R with F is **not in BCNF**.

Let us **decompose** it using $\{B\} \rightarrow \{A\}$.

(Cont.)

$R1 = \{B\}^+ = \{B, A\}$

$F1 = \{\{B\} \rightarrow \{A\}\}$

The candidate key is $\{B\}$.

$R1$ with $F1$ is in BCNF.

$R2 = R - \{B\}^+ \cup \{B\} = \{B, C, D\}$

$F2 = \emptyset$

The candidate keys is $\{B, C, D\}$

$R2$ with $F2$ is in BCNF.

But we lost $\{A, D\} \rightarrow \{B, C\}$!

The decomposition is not dependency preserving.

(Cont.)

R = {A, B, C, D}

F = {{A, D} → {B, C}, {B} → {A}}

The candidate keys are {A, D} and {B, D}

{B} → {A} is not trivial. {B} is not a superkey.

However {A} is a prime attribute.

R with F is in 3NF. We do not decompose.

What if it is not even in 3NF in the first place?

Decomposition into BCNF

```
Let R be a relation scheme;
Let F be a set of functional dependencies;
S = ∅;
compute the minimal cover F'
for each X → Y in F'
      if no relation in S contains X ∪ Y
      then create relation with scheme X ∪ Y
      if no relation in S contains a candidate
          key for R
      then create a relation
         with scheme any candidate key for Ri
endfor
```

An Example

`R = {A, B, C, D}`

`F = {{A, D}` → `{B, C}, {B}` → `{A}}`

The candidate keys `are {A, D}` and `{B, D}`

`F` is an extended minimal cover (otherwise compute it…)

- `{A, D}` → `{B, C}` gives `R1 = {A, B, C, D},`
  it is in 3NF by construction. It is not in BCNF.
- `{B}` → `{A}` should give `R2 = {A, B},`
  but it is included in `R1`.
  We do not create it.
  it is in 3NF by construction.
- `{A, B, C, D}` already contains a candidate key.
- The decomposition is dependency preserving by construction.

Another Example

`R = {A, B, C, D, E}`

`F = {{A, B} → {C}, {C} → {D}, {D} →{E}}`

The candidate key is `{A, B}`

`F` is an extended minimal cover (otherwise compute it…)

- `{A, B} → {C}` gives `R1 = {A, B, C},`
  it is in 3NF by construction. It is in BCNF.
- `{C} → {D}` gives `R2 = {C, D},`
  it is in 3NF by construction. It is in BCNF.
- `{D} → {E}` gives `R1 = {D, E},`
  it is in 3NF by construction. It is in BCNF.
  but it is included in `R1`.
- `{A, B}` already contains a candidate key.
- The decomposition is dependency preserving by construction.
- The decomposition is in BCNF by chance (by odds are very good).

Another Example

R = {A, B, C, D, E}

F = {{A} → {B, C}, {D} →{E}}

The candidate key is {A, D}

F is an extended minimal cover (otherwise compute it…)

- **{A} → {B, C} gives R1 = {A, B, C},**

  it is in 3NF by construction. It is in BCNF.

- **{D} → {E} gives R2 = {D, E},**

  it is in 3NF by construction. It is in BCNF.

- **There is no fragment containing the key. We create a fragment with a candidate key .**

  {A, D} gives R2 = {A, D},

  it is in 3NF by construction. It is in BCNF.

- **The decomposition is dependency preserving by construction.**
- **The decomposition is in BCNF by chance (by odds are very good).**

Synthesis

name	tel	address
Sangria Clarke Quay	65166516	1 Clarke Quay
Sangria Holland V	65165555	13 Holland Drive

type	name
superior	Sangria Clarke Quay
standard	Sangria Clarke Quay
suite	Sangria Clarke Quay
superior	Sangria Holland V
standard	Sangria Holland V
suite	Sangria Holland V
junior suite	Sangria Holland V

type	price
superior	145
standard	75
suite	250
junior suite	200
executive	175

Credits

The content of this lecture is based on the book "Introduction to database Systems"
By
S. Bressan and B. Catania, McGraw Hill publisher

Images and clips used in this presentation are licensed from Microsoft Office Online Clipart and Media

For questions about the content of this course and about copyrights, please contact Stéphane Bressan

steph@nus.edu.sg