Introduction to Data Management
CSE 414

Unit 6: Conceptual Design

E/R Diagrams

Integrity Constraints

BCNF

(3 lectures)

Introduction to Data Management
CSE 414

Design Theory and BCNF

Relational Schema Design

Name	<u>SSN</u>	<u>PhoneNumber</u>	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

One person may have multiple phones, but lives in only one city

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Name	<u>SSN</u>	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Bellevue” (how ?)
- Easy to delete all Joe’s phone numbers (how ?)

Relational Schema Design (or Logical Design)

How do we do this systematically?

- Start with some relational schema
- Find out its ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

$A_1 \ldots A_n$ **determines** $B_1 .. B_m$

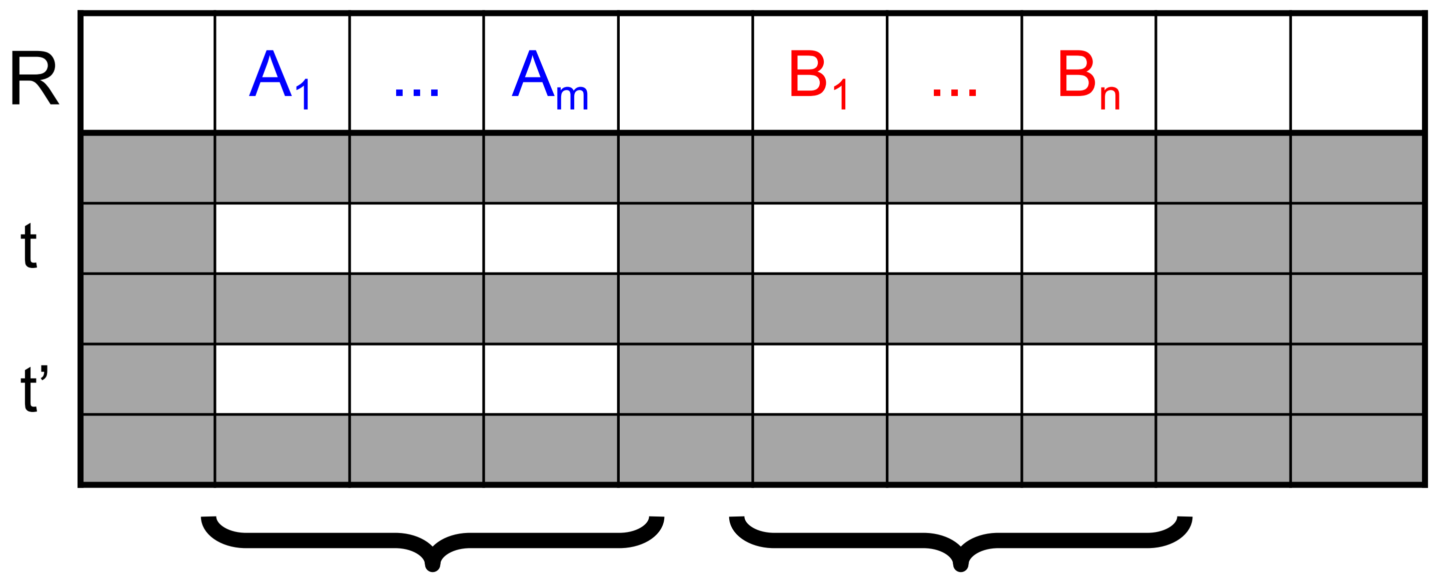
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

$\forall t, t' \in R,$

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



if t, t' agree here then t, t' agree here

Example

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

But not Phone → Position

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Do all the FDs hold on this instance?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	49
Gizmo	Stationary	Green	Office-supp.	59

What about this one ?

Buzzwords

- FD **holds** or **does not hold** on an instance
- If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**
- If we say that R satisfies an FD, we are **stating a constraint on R**

Why bother with FDs?

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number?

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies!
There could be more FDs implied by the ones we have.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$,
s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

1. name → color
2. category → department
3. color, category → price

Closures:

$name^+ = \{name, color\}$
$color^+ = \{color\}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$
$\{ name, category, \qquad \}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
  if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
  $B_1, \ldots, B_n$ are all in X
  then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$
{ name, category, color, }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
  if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
  $B_1, \ldots, B_n$ are all in X
  then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{\text{name}, \text{category}\}^+ =$
{ name, category, color, department }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{\text{name, category}\}^+ =$
{ name, category, color, department, price }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
  if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
  $B_1, \ldots, B_n$ are all in X
  then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{\text{name, category}\}^+ =$
{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ X = {A, B, }

Compute $\{A, F\}^+$ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

$$A, B \rightarrow C$$
$$A, D \rightarrow E$$
$$B \rightarrow D$$
$$A, F \rightarrow B$$

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

$$\begin{array}{l}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array}$$

Compute $\{A,B\}^+$ $\quad$ $X = \{A, B, C, D, E \qquad \}$

Compute $\{A, F\}^+$ $\quad$ $X = \{A, F, B, C, D, E \qquad \}$

Example

In class:

R(A,B,C,D,E,F)

$A, B \rightarrow C$
$A, D \rightarrow E$
$B \rightarrow D$
$A, F \rightarrow B$

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, B, C, D, E }

What is the key of R?

Practice at Home

Find all FD's implied by:

$$
\begin{aligned}
A, B &\rightarrow C \\
A, D &\rightarrow B \\
B &\rightarrow D
\end{aligned}
$$

Practice at Home

Find all FD's implied by:

A, B	$\rightarrow$	C
A, D	$\rightarrow$	B
B	$\rightarrow$	D

Step 1: Compute X^+, for every X:

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$,
$BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute– why ?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Step 2: Enumerate all FD's $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$

Keys

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$
- A **key** is a minimal superkey
  - A superkey and for which no subset is a superkey

Computing (Super)Keys

- For all sets X, compute X^+
- If X^+ = [all attributes], then X is a superkey
- Try reducing to the minimal X's to get the key

Example

Product(name, price, category, color)

$$\boxed{\begin{array}{l}\text{name, category} \rightarrow \text{price} \\ \text{category} \rightarrow \text{color}\end{array}}$$

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more distinct keys

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more distinct keys

$A \rightarrow B$
$B \rightarrow C$
$C \rightarrow A$

or

$AB \rightarrow C$
$BC \rightarrow A$

or

$A \rightarrow BC$
$B \rightarrow AC$

what are the keys here ?

Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key
- X → A is not OK otherwise
  - Need to decompose the table, but how?

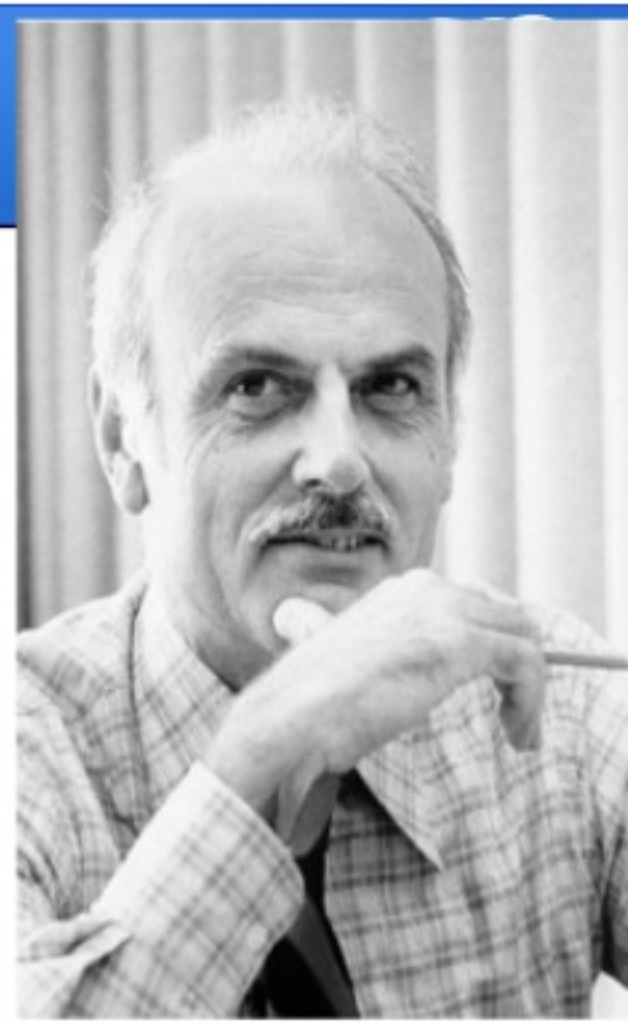
Boyce-Codd Normal Form

Boyce-Codd Normal Form

Dr. Raymond F. Boyce

Edgar Frank “Ted” Codd

"A Relational Model of Data for Large Shared Data Banks"



Boyce-Codd Normal Form

If there are no "bad" FDs:

Definition. A relation R is in BCNF if:

Whenever $X \rightarrow B$ is a non-trivial dependency, then X is a superkey.

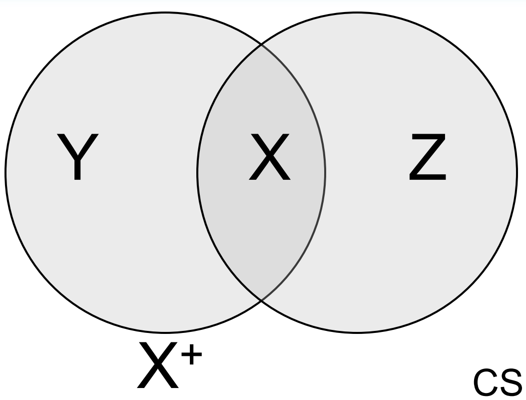
Equivalently:

Definition. A relation R is in BCNF if:

$\forall$ X, either $X^+ = X$ (i.e., X is not in any FDs)
or X^+ = [all attributes] (computed using FDs)

BCNF Decomposition Algorithm

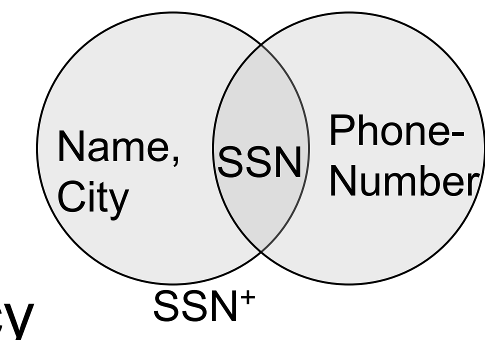
Normalize(R)
  find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]
  <u>if</u> (not found) **<u>then</u>** "R is in BCNF"
  <u>let</u> $Y = X^+ - X$; $Z =$ [all attributes] $- X^+$
  decompose R into R1($X \cup Y$) and R2($X \cup Z$)
  Normalize(R1); Normalize(R2);



Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

SSN → Name, City



The only key is: {SSN, PhoneNumber}

Hence SSN → Name, City is a “bad” dependency

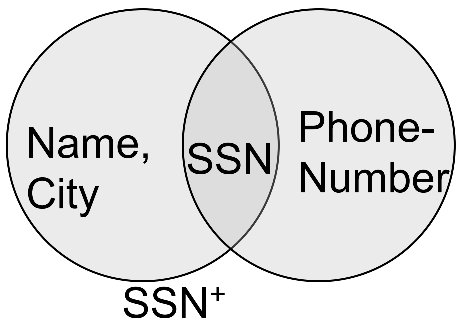
In other words:

SSN+ = SSN, Name, City and is neither SSN nor All Attributes

Example BCNF Decomposition

Name	**SSN**	**City**
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN → Name, City



SSN	**PhoneNumber**
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Find X s.t.: X ≠X^+ and X^+ ≠ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Find X s.t.: X $\neq X^+$ and $X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

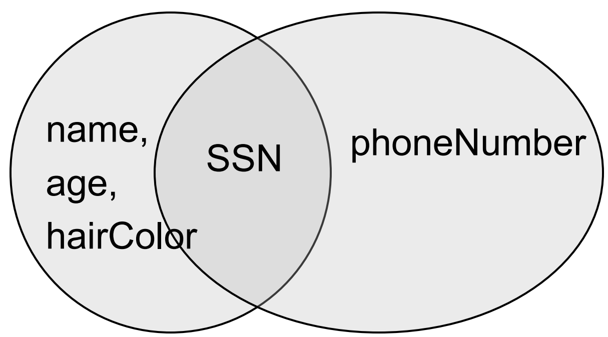
SSN → name, age

age → hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)

Phone(SSN, phoneNumber)



Find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

What are the keys ?

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: P: age+ = age, hairColor

Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Note the keys!

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)

Phone(SSN, phoneNumber)

Iteration 2: P: age+ = age, hairColor

Decompose: People(SSN, name, age)

Hair(age, hairColor)

Phone(SSN, phoneNumber)

R(A,B,C,D)

Example: BCNF

A → B
B → C

R(A,B,C,D)

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

Example: BCNF

Recall: find X s.t.
$X \subsetneq X^{+} \subsetneq$ [all-attrs]

R(A,B,C,D)

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

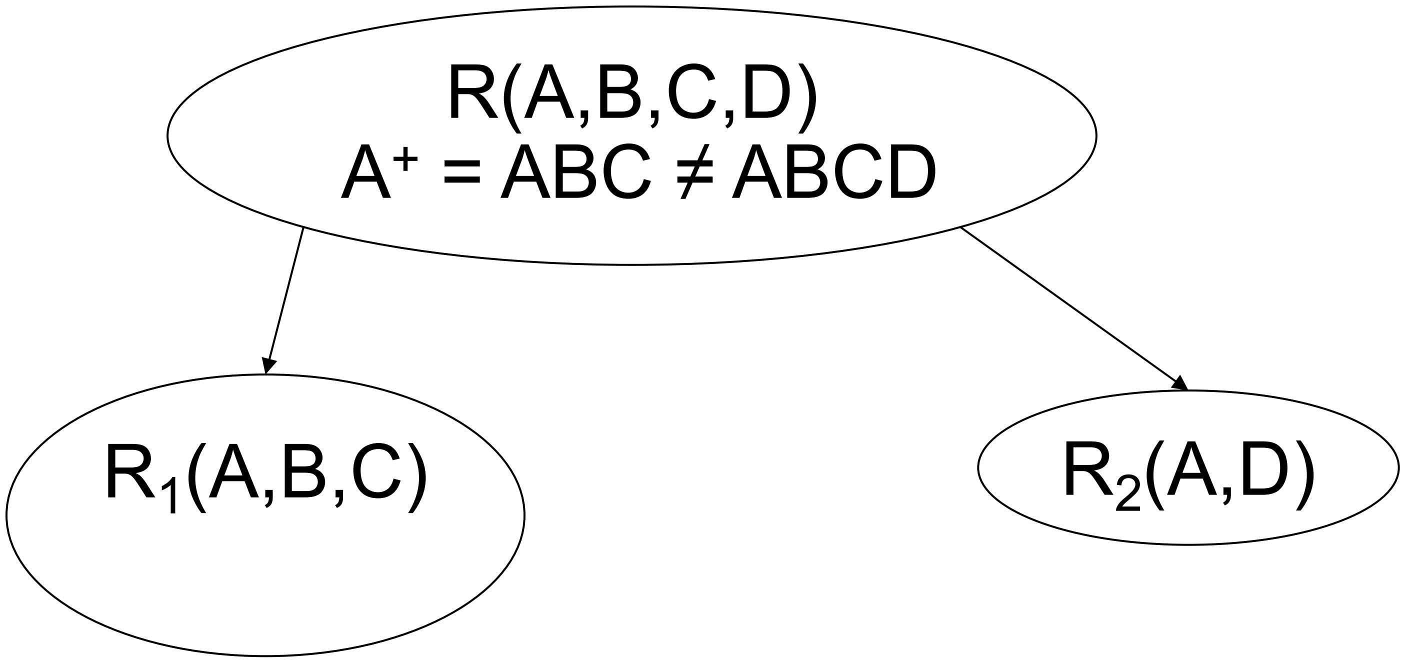
Example: BCNF

R(A,B,C,D)
$A^+ = ABC \neq ABCD$

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

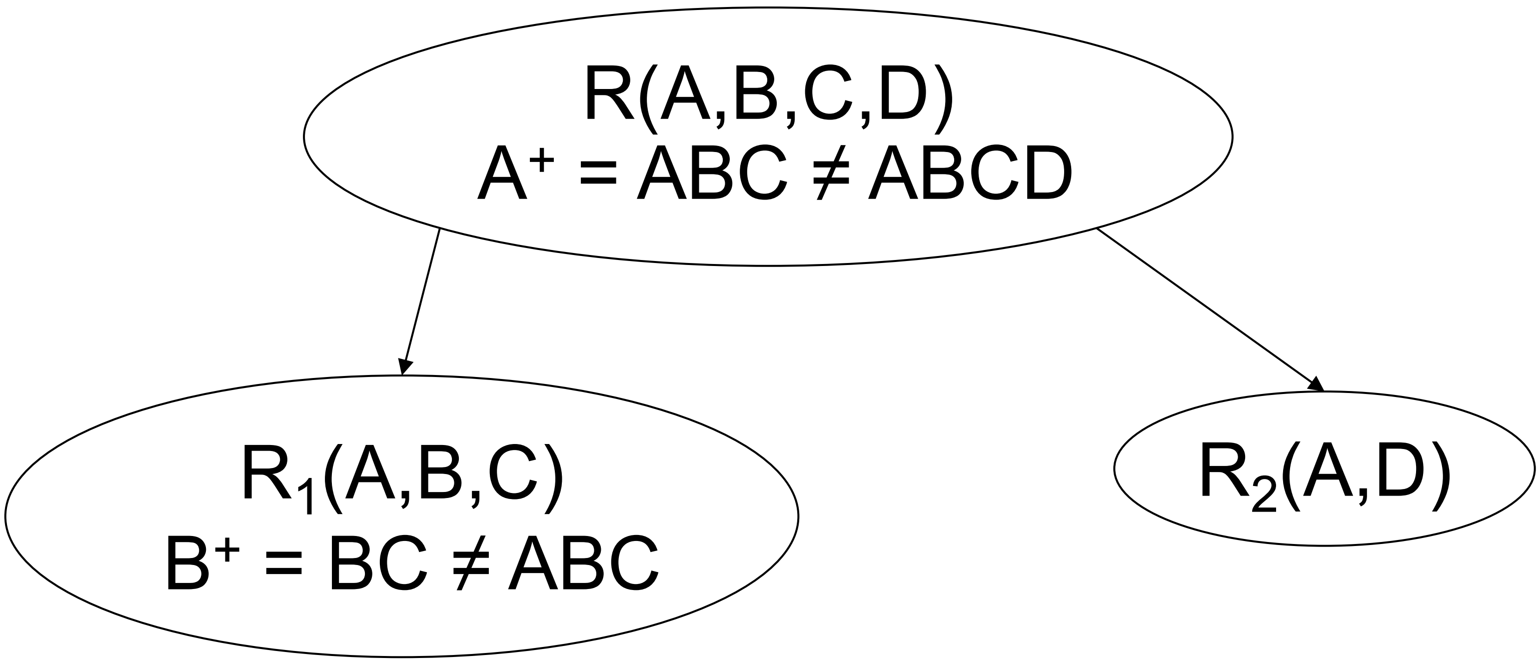
Example: BCNF



R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

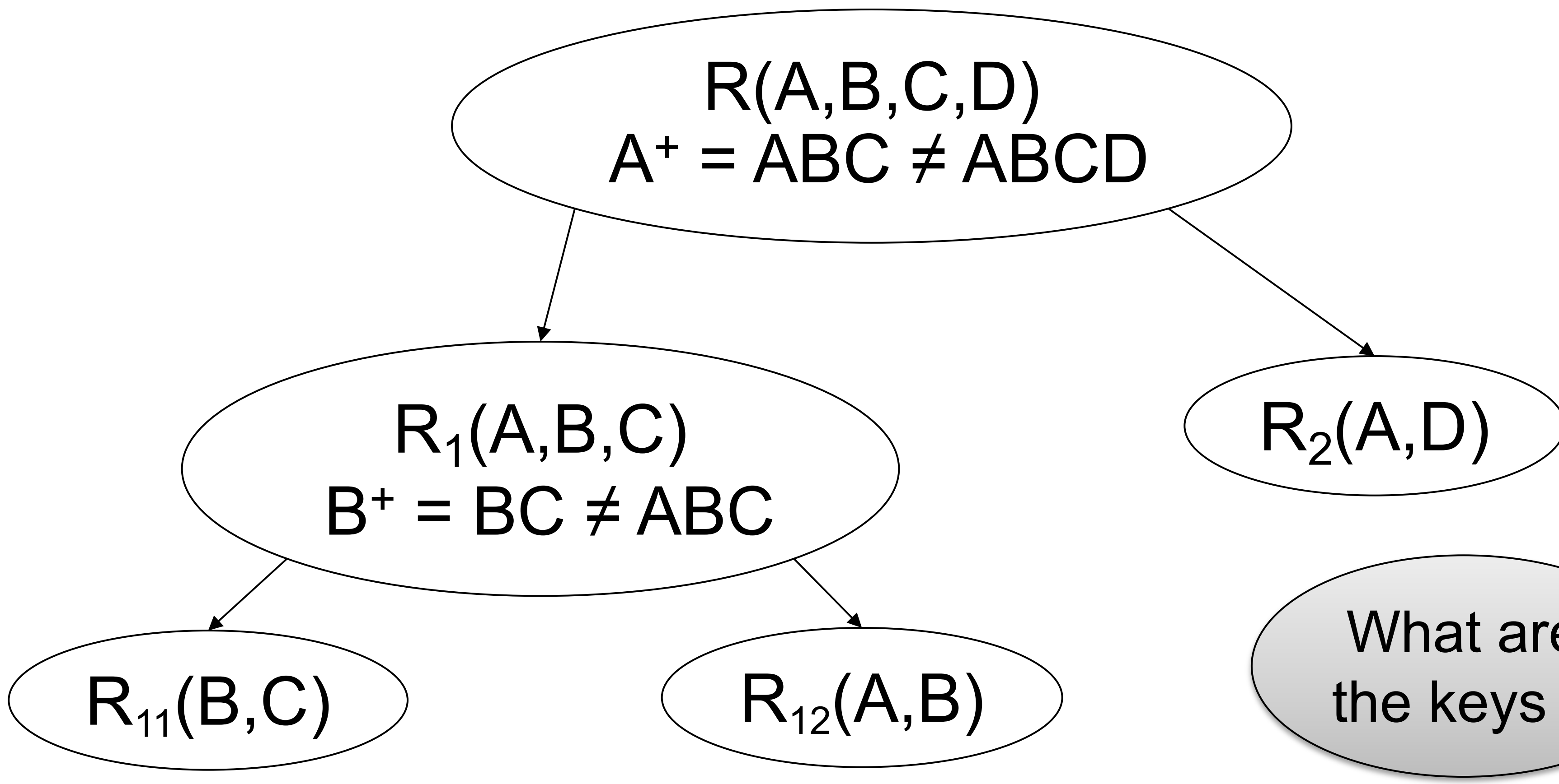
Example: BCNF



R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

Example: BCNF



What are the keys ?

What happens if in R we first pick B^+ ? Or AB^+ ?

Getting Practical

How to implement normalization in SQL

Motivation

- We learned about how to normalize tables to avoid anomalies
- How can we implement normalization in SQL if we can’t modify existing tables?
  - This might be due to legacy applications that rely on previous schemas to run

Views

- A view in SQL =
  - A table computed from other tables, s.t., whenever the base tables are updated, the view is updated too
- More generally:
  - A view is derived data that keeps track of changes in the original data
- Compare:
  - A function computes a value from other values, but does not keep track of changes to the inputs

Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

A Simple View

Create a view that returns for each store
the prices of products purchased at that store

```
CREATE VIEW  StorePrice AS
  SELECT DISTINCT x.store, y.price
  FROM  Purchase x, Product y
  WHERE x.product = y.pname
```

This is like a new table
StorePrice(store,price)

Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

We Use a View Like Any Table

- A "high end" store is a store that sell some products over 1000.
- For each customer, return all the high end stores that they visit.

```
SELECT DISTINCT u.customer, u.store
FROM Purchase u, StorePrice v
WHERE u.store = v.store
   AND v.price > 1000
```

Types of Views

- Virtual views
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Pre-computed offline – fast at runtime
  - May have stale data (must recompute or update)
  - Indexes *are* materialized views
- A key component of physical tuning of databases is the selection of materialized views and indexes

Vertical Partitioning

Resumes

<u>**SSN**</u>	**Name**	**Address**	**Resume**	**Picture**
234234	Mary	Houston	Doc1…	JPG1…
345345	Sue	Seattle	Doc2…	JPG2…
345343	Joan	Seattle	Doc3…	JPG3…
432432	Ann	Portland	Doc4…	JPG4…

T1

<u>**SSN**</u>	**Name**	**Address**
234234	Mary	Houston
345345	Sue	Seattle
. . .		

T2

<u>**SSN**</u>	**Resume**
234234	Doc1…
345345	Doc2…

T3

<u>**SSN**</u>	**Picture**
234234	JPG1…
345345	JPG2…

T2.SSN is a key *<u>and</u>* a foreign key to **T1**.SSN. Same for **T3**.SSN

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE  T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE  T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM    Resumes
WHERE   name = 'Sue'
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE   T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

Original query:

```
SELECT T1.address
FROM T1, T2, T3
WHERE T1.name = 'Sue'
    AND T1.SSN=T2.SSN
    AND T1.SSN = T3.SSN
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE  T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM    Resumes
WHERE   name = 'Sue'
```

Modified query:

SELECT T1.address
FROM T1, ~~T2, T3~~
WHERE T1.name = 'Sue'
~~AND T1.SSN=T2.SSN~~
~~AND T1.SSN = T3.SSN~~

Final query:

```
SELECT T1.address
FROM T1
WHERE T1.name = 'Sue'
```

Vertical Partitioning Applications

- Advantages
  - Speeds up queries that touch only a small fraction of columns
  - Single column can be compressed effectively, reducing disk I/O
- Disadvantages
  - Updates are expensive!
  - Need many joins to access many columns
  - Repeated key columns add overhead

Horizontal Partitioning

Customers

SSN	Name	City
234234	Mary	Houston
345345	Sue	Seattle
345343	Joan	Seattle
234234	Ann	Portland
--	Frank	Calgary
--	Jean	Montreal

CustomersInHouston

SSN	Name	City
234234	Mary	Houston

CustomersInSeattle

SSN	Name	City
345345	Sue	Seattle
345343	Joan	Seattle

. . . . .

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
CREATE VIEW Customers AS
    CustomersInHouston
      UNION ALL
    CustomersInSeattle
      UNION ALL
     . . .
```

CustomersInHouston(<u>ssn</u>,name,city)
CustomersInSeattle(<u>ssn</u>,name,city)
. . . . .

Customers(<u>ssn</u>,name,city)

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = ‘Seattle’
```

Which tables are inspected by the system ?

CustomersInHouston(<u>ssn</u>,name,city)
CustomersInSeattle(<u>ssn</u>,name,city)
. . . . .

Customers(<u>ssn</u>,name,city)

Horizontal Partitioning

Better: remove CustomerInHouston.city etc

```
CREATE VIEW  Customers  AS
    (SELECT SSN, name, 'Houston' as city
     FROM CustomersInHouston)
       UNION ALL
    (SELECT SSN, name, 'Seattle' as city
     FROM CustomersInSeattle)
       UNION ALL
     . . .
```

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT  name
FROM    Customers
WHERE   city = 'Seattle'
```

```
SELECT name
FROM   CustomersInSeattle
```

Horizontal Partitioning Applications

- Performance optimization
  - Especially for data warehousing
  - E.g., one partition per month
  - E.g., archived applications and active applications
- Distributed and parallel databases
- Data integration

Conclusion

- Poor schemas can lead to performance inefficiencies
- E/R diagrams are means to structurally visualize and design relational schemas
- Normalization is a principled way of converting schemas into a form that avoid such problems
- BCNF is one of the most widely used normalized form in practice