CSE 344

MAY 18TH – LOSS AND VIEWS

ADMINISTRIVIA

- **HW7 Due Wednesday, May 23rd 11:30**
- **OQ6 Due Wednesday, May 23rd 11:00**
- **HW8 Out Wednesday, May 23rd**
  - Due Friday, June 1st

DATABASE DESIGN PROCESS

Conceptual Model:

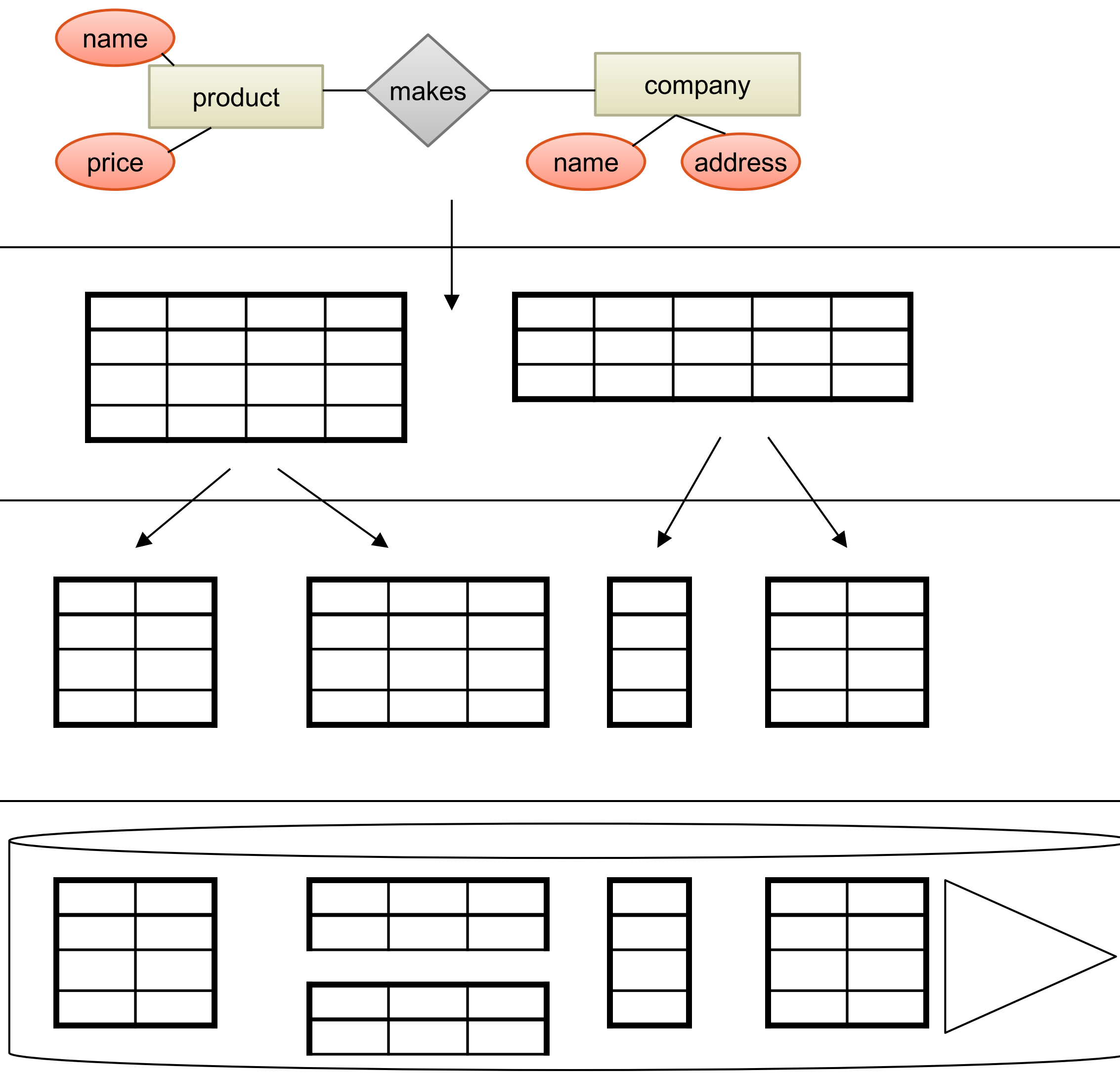
Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details

Physical Schema

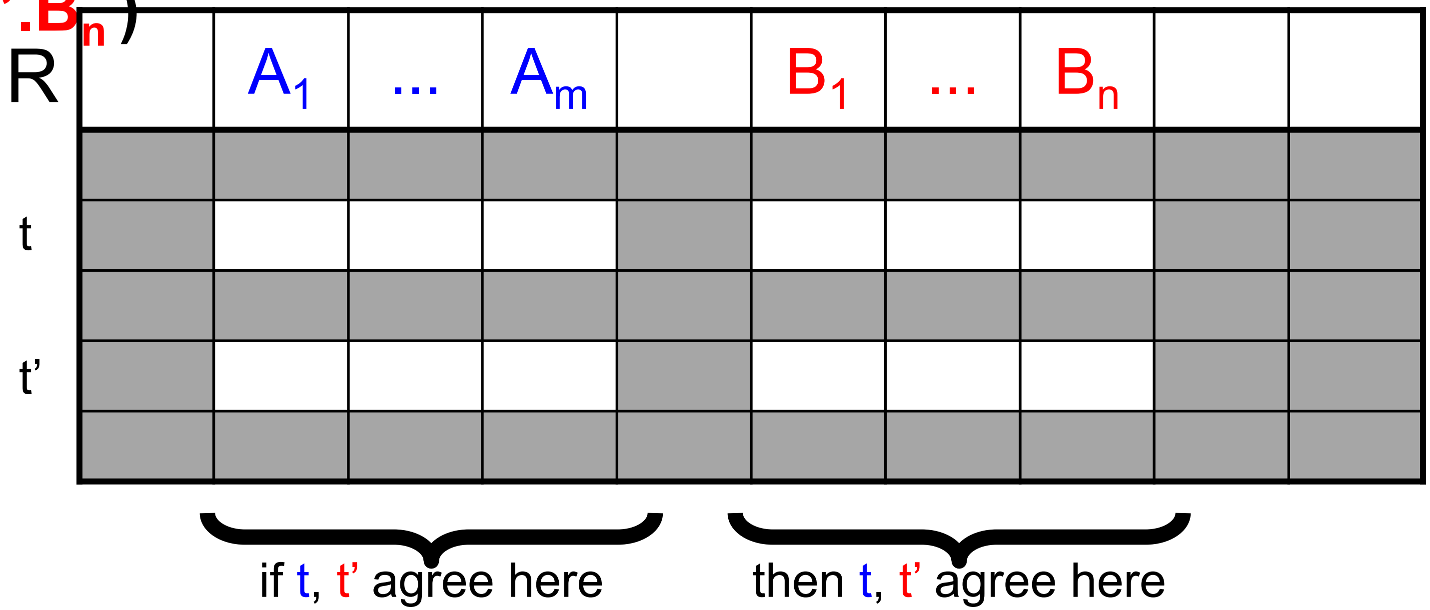


FUNCTIONAL DEPENDENCIES (FDS)

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

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

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



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}

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

color$^+$ = {color}

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

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

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:

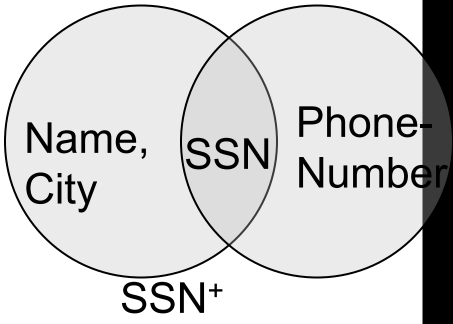
Definition. A relation R is in BCNF if:

$\forall$ X, either $X^+ = X$ or $X^+ =$ [all attributes]

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

DECOMPOSITIONS IN GENERAL

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$S_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$S_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

S_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$
S_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

LOSSLESS DECOMPOSITION

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
~~Gizmo~~	~~19.99~~

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

LOSSY DECOMPOSITION

What is lossy here?

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

LOSSY DECOMPOSITION

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

DECOMPOSITION IN GENERAL

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$S_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$S_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

Let: S_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

S_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

The decomposition is called *lossless* if $R = S_1 \bowtie S_2$

Fact: If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ then the decomposition is lossless

It follows that every BCNF decomposition is lossless

IS THIS LOSSLESS?

If we decompose R into $\Pi_{S1}(R)$, $\Pi_{S2}(R)$, $\Pi_{S3}(R)$, …
Is it true that $S1 \bowtie S2 \bowtie S3 \bowtie \ldots = R$?

That is true if we can show that:

$R \subseteq S1 \bowtie S2 \bowtie S3 \bowtie \ldots$ always holds (why?)

$R \supseteq S1 \bowtie S2 \bowtie S3 \bowtie \ldots$ neet to check

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D
a	b1	c1	d

Why ?
(a,d) ∈ S1 = Π_{AD}(R)

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	(a,d) ∈S1 = Π_{AD}(R)
a	b2	c	d2	(a,c) ∈S2 = Π_{BD}(R)

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	(a,d) ∈ S1 = Π_{AD}(R)
a	b2	c	d2	(a,c) ∈ S2 = Π_{BD}(R)
a3	b	c	d	(b,c,d) ∈ S3 = Π_{BCD}(R)

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R ⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	(a,d) ∈S1 = Π_{AD}(R)
a	b2	c	d2	(a,c) ∈S2 = Π_{BD}(R)
a3	b	c	d	(b,c,d) ∈S3 = Π_{BCD}(R)

"Chase" them (apply FDs):

A→B

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R⊆ S1 ⋈ S2 ⋈ S3
Need to check: R ⊇ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) ∈ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	(a,d) ∈S1 = Π_{AD}(R)
a	b2	c	d2	(a,c) ∈S2 = Π_{BD}(R)
a3	b	c	d	(b,c,d) ∈S3 = Π_{BCD}(R)

"Chase" them (apply FDs):

A→B

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

B→C

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

Why ?
$(a,d) \in S1 = \Pi_{AD}(R)$
$(a,c) \in S2 = \Pi_{BD}(R)$
$(b,c,d) \in S3 = \Pi_{BCD}(R)$

"Chase" them (apply FDs):

A→B

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

B→C

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

CD→A

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d

Hence R contains (a,b,c,d)

SCHEMA REFINEMENTS = NORMAL FORMS

- **1st Normal Form = all tables are flat**
- **2nd Normal Form = no FD with ”non-prime” attributes**
  - *Obselete*
  - Prime attributes: attributes part of a key
- **Boyce Codd Normal Form = no “bad” FDs**
  - Are there problems with BCNF?

DEPENDENCY PRESERVATION

- **Bookings(title,theatre,city)**
  - FD:
    - theatre -> city
    - title,city -> theatre
- What are the keys?

DEPENDENCY PRESERVATION

- **Bookings(title,theatre,city)**
  - FD:
    - theatre -> city
    - title,city -> theatre
- What are the keys?
  - None of the single attributes
  - {title,city},{theatre,title}
- **BCNF?**

DEPENDENCY PRESERVATION

- **Bookings(title,theatre,city)**
  - FD:
    - theatre -> city
    - title,city -> theatre
- What are the keys?
  - None of the single attributes
  - {title,city},{theatre,title}
- **BCNF?**
  - No, {theatre} is neither a trivial dependency nor a superkey
  - Decompose?

DEPENDENCY PRESERVATION

- **Bookings(title,theatre,city)**
  - FD:
    - theatre -> city
    - title,city -> theatre
- What are the keys?
  - None of the single attributes
  - {title,city},{theatre,title}
- **BCNF?**
  - No, {theatre} is neither a trivial dependency nor a superkey
  - Decompose? R1(theatre,city) R2(theatre,title)
  - What’s wrong? *(think of FDs)*

DEPENDENCY PRESERVATION

- **Bookings(title,theatre,city)**
  - FD:
    - theatre -> city
    - title,city -> theatre
- What are the keys?
  - None of the single attributes
  - {title,city},{theatre,title}
- **BCNF?**
  - No, {theatre} is neither a trivial dependency nor a superkey
  - Decompose? R1(theatre,city) R2(theatre,title)
  - What’s wrong? *(think of FDs)*
  - We can’t guarantee title,city -> theatre with simple constraints if we join

NORMAL FORMS

- **3rd Normal form**
  - Allows tables with BCNF violations if a decomposition separates an FD
  - Can result in redundancy
- **4th Normal form**
  - Multi-valued dependencies
    - Incorporate info about attributes in neither A nor B
    - All MVDs are also FDs
  - Apply BCNF alg with for MVD and FD

NORMAL FORMS

- **5th Normal Form**
  - Join dependency
    - Lossless/exact joining
    - Join independent Tables
- **6th Normal Form**
  - Only allow trivial join dependencies
  - Only need key/tuple constraints to represent all constraints

FORMS/DECOMPOSITION

- Produce and verify FDs, superkeys, keys
- Be able to decompose a table into BCNF
- Flaws of 1NF/BCNF
- Identify loss and be able to apply the chase test

IMPLEMENTATION

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

MATERIALIZED VIEWS

```
CREATE MATERIALIZED VIEW View_name
BUILD [IMMEDIATE/DEFERRED]
REFRESH [FAST/COMPLETE/FORCE]
ON [COMMIT/DEMAND]
AS Sql_query
```

- **Immediate v deferred**
  - Build immediately, or after a query
- **Fast v. Complete v. Force**
  - Level of refresh – log based v. complete rebuild
- **Commit v. Demand**
  - Commit: after data is added
  - Demand: after conditions are set (time is common)

VERTICAL PARTITIONING

Resumes

SSN	Name	Address	Resume	Picture
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
432432	Ann	Portland	Clob4…	Blob4…

T1

SSN	Name	Address
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	Resume
234234	Clob1…
345345	Clob2…

T3

SSN	Picture
234234	Blob1…
345345	Blob2…

T2.SSN is a key *and* 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(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

HORIZONTAL PARTITIONING

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

Which tables are inspected by the system ?

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

Customers(ssn,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