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

\[
\forall t, t' \in R, \quad (t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land ... \land t.B_n)
\]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>( ... )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( ... )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
</tr>
<tr>
<td>( t' )</td>
<td>( t' )</td>
<td>( t' )</td>
<td>( t' )</td>
<td>( t' )</td>
<td>( t' )</td>
<td>( t' )</td>
</tr>
</tbody>
</table>

if \( t, t' \) agree here then \( t, t' \) agree here
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 \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

Closures:

\[ \text{name}^+ = \{\text{name}, \text{color}\} \]
\[ \{\text{name}, \text{category}\}^+ = \{\text{name}, \text{category}, \text{color}, \text{department}, \text{price}\} \]
\[ \text{color}^+ = \{\text{color}\} \]
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
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:

\[ \forall X, \text{ either } X^+ = X \text{ or } X^+ = [\text{all attributes}] \]
The only key is: \{SSN, PhoneNumber\}
Hence \(SSN \rightarrow \text{Name, City}\) is a “bad” dependency

In other words:
\(SSN^+ = \text{SSN, Name, City}\) and is neither \(SSN\) nor \(\text{All Attributes}\)
DECOMPOSITIONS IN GENERAL

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

\[ S_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ S_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
## LOSSLESS DECOMPOSITION

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

**Left Decomposition:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
</tbody>
</table>

**Middle Decomposition:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>
LOSSY DECOMPOSITION

What is lossy here?

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

- **Name**
  - Gizmo
  - OneClick
  - Gizmo

- **Category**
  - Gadget
  - Camera

- **Price**
  - 19.99
  - 24.99
  - 19.99
**LOSSY DECOMPOSITION**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

**Name**

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

**Price**

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
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:

$S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m$

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

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 $\Pi_{S_1}(R)$, $\Pi_{S_2}(R)$, $\Pi_{S_3}(R)$, ... Is it true that $S_1 \bowtie S_2 \bowtie S_3 \bowtie ... = R$?

That is true if we can show that:

$R \subseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie ...$ always holds (why?)

$R \supseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie ...$ neet to check
THE CHASE TEST FOR LOSSLESS JOIN

Example from textbook Ch. 3.4.2

R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)
R satisfies: A \rightarrow B, B \rightarrow C, CD \rightarrow 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
Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), \ S2 = \Pi_{AC}(R), \ S3 = \Pi_{BCD}(R), \]

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?
Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]
R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

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:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>
| a | b1| c1| d | Why ?
|   |   |   |   | \((a,d) \in S1 = \Pi_{AD}(R)\)
Example from textbook Ch. 3.4.2

**THE CHASE TEST FOR LOSSLESS JOIN**

\[
R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)
\]

\(R\) satisfies: \(A \rightarrow B, B \rightarrow C, \text{CD} \rightarrow A\)

\(S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R)\),

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:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
</tbody>
</table>

Why?

\((a,d) \in S1 = \Pi_{AD}(R)\)

\((a,c) \in S2 = \Pi_{BD}(R)\)
THE CHASE TEST FOR LOSSLESS JOIN

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D) \]
R satisfies: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

\[ S_1 = \Pi_{AD}(R), S_2 = \Pi_{AC}(R), S_3 = \Pi_{BCD}(R), \]

hence \( R \subseteq S_1 \bowtie S_2 \bowtie S_3 \)

Need to check: \( R \supseteq S_1 \bowtie S_2 \bowtie S_3 \)

Suppose \( (a,b,c,d) \in S_1 \bowtie S_2 \bowtie S_3 \) Is it also in \( R \)?

R must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

- \( (a,d) \in S_1 = \Pi_{AD}(R) \)
- \( (a,c) \in S_2 = \Pi_{BD}(R) \)
- \( (b,c,d) \in S_3 = \Pi_{BCD}(R) \)
Example from textbook Ch. 3.4.2

THE CHASE TEST FOR LOSSLESS JOIN

\[ R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D) \]
R satisfies: \( A \Rightarrow B, B \Rightarrow C, CD \Rightarrow A \)

\( S_1 = \Pi_{AD}(R), S_2 = \Pi_{AC}(R), S_3 = \Pi_{BCD}(R), \)

hence \( R \subseteq S_1 \bowtie S_2 \bowtie S_3 \)

Need to check: \( R \supseteq S_1 \bowtie S_2 \bowtie S_3 \)

Suppose \((a,b,c,d) \in S_1 \bowtie S_2 \bowtie S_3\) Is it also in \(R\)?

\( R \) must contain the following tuples:

“Chase” them (apply FDs):

\[
\begin{array}{cccc}
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b_1</td>
<td>c_1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b_2</td>
<td>c</td>
<td>d_2</td>
</tr>
<tr>
<td>a_3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
\end{array}
\]

Why?

\((a,d) \in S_1 = \Pi_{AD}(R)\)
\((a,c) \in S_2 = \Pi_{BD}(R)\)
\((b,c,d) \in S_3 = \Pi_{BCD}(R)\)
Example from textbook Ch. 3.4.2

**THE CHASE TEST FOR LOSSLESS JOIN**

\[ R(A,B,C,D) = S1(A,D) \Join S2(A,C) \Join S3(B,C,D) \]

\[ R \text{ satisfies: } A \rightarrow B, \ B \rightarrow C, \ CD \rightarrow A \]

\( S1 = \Pi_{AD}(R), \ S2 = \Pi_{AC}(R), \ S3 = \Pi_{BCD}(R), \)

\[ \text{hence } R \subseteq S1 \Join S2 \Join S3 \]

Need to check: \( R \supseteq S1 \Join S2 \Join S3 \)

Suppose \((a,b,c,d) \in S1 \Join S2 \Join S3\) Is it also in \(R\)?

\( R \) must contain the following tuples:

“Chase” them (apply FDs):

\[ A \rightarrow B \]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c1 & d \\
a & b1 & c & d2 \\
a3 & b & c & d \\
\end{array}
\]

\[ B \rightarrow C \]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c & d \\
a & b1 & c & d2 \\
a3 & b & c & d \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c1 & d \\
(a,d) \in S1 = \Pi_{AD}(R) \\
(a,c) \in S2 = \Pi_{BD}(R) \\
(b,c,d) \in S3 = \Pi_{BCD}(R) \\
\end{array}
\]
THE CHASE TEST FOR LOSSLESS JOIN

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R), \]

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:

“Chase” them (apply FDs):

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c1 & d \\
a & b2 & c & d2 \\
a3 & b & c & d \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c & d \\
a & b2 & c & d2 \\
a3 & b & c & d \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c1 & d \\
a & b2 & c & d2 \\
a3 & b & c & d \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c & d \\
a & b1 & c & d2 \\
a & b1 & c & d \\
\end{array}
\]

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
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
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)
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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
<td>Clob1...</td>
<td>Blob1...</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Clob2...</td>
<td>Blob2...</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Clob3...</td>
<td>Blob3...</td>
</tr>
<tr>
<td>432432</td>
<td>Ann</td>
<td>Portland</td>
<td>Clob4...</td>
<td>Blob4...</td>
</tr>
</tbody>
</table>

T2.SSN is a key *and* a foreign key to T1.SSN. Same for T3.SSN.
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
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'
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'

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

Original query:
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'

Final query:
SELECT T1.address
FROM T1
WHERE T1.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
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**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
</tr>
<tr>
<td>--</td>
<td>Frank</td>
<td>Calgary</td>
</tr>
<tr>
<td>--</td>
<td>Jean</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

**CustomersInHouston**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
</tr>
</tbody>
</table>

**CustomersInSeattle**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

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

CustomersInHouston(ssn, name, city)
CustomersInSeattle(ssn, name, city)
HORIZONTAL PARTITIONING

SELECT name
FROM Customers
WHERE city = 'Seattle'

Which tables are inspected by the system?
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

...
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