**Relational Schema Design**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

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?

**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:**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

**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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

A1, A2, ..., An determines B1, B2, ..., Bm.

- **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 \( \rightarrow \) Name, Phone, Position
  Position \( \rightarrow \) Phone
  but not Phone \( \rightarrow \) Position

- **Example**
  - name \( \rightarrow \) color category department
    - Gizmo | Gadget | Green | Toys | 49
    - Tweaker | Gadget | Green | Toys | 99
  
  Do all the FDs hold on this instance?
Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-45-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

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:

\[
\text{name} \rightarrow \text{color, category} \rightarrow \text{department} \\
\text{color, category} \rightarrow \text{price}
\]

Then this FD also holds:

\[
\text{name, category} \rightarrow \text{price}
\]

An Interesting Observation

If all these FDs are true:

\[
\text{name} \rightarrow \text{color, category} \rightarrow \text{department} \\
\text{color, category} \rightarrow \text{price}
\]

Then this FD also holds:

\[
\text{name, category} \rightarrow \text{price}
\]

An Interesting Observation

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.
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:

Closures:
- \( \text{name}^+ = \{ \text{name}, \text{color} \} \)
- \( \text{color}^+ = \{ \text{color} \} \)

Closure Algorithm

\( X = \{A_1, \ldots, A_n\} \).

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 \( X \).

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

Example:

Hence:

\( \text{name, category} \rightarrow \text{color, department, price} \)
Compute \( \{A, B, C\} + X = \{A, B, C, D, E, F\} \)

Compute \( \{A, F\} + X = \{A, F, B, C, D, E\} \)

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

In class:
- \( A, B \rightarrow C \)
- \( A, D \rightarrow E \)
- \( B \rightarrow D \)
- \( A, F \rightarrow B \)

Compute \( (A,B)^* \) \( X = \{A, B, C, D, E\} \)

Compute \( (A, F)^* \) \( X = \{A, F, B, C, D, E\} \)

**Practice at Home**

Find all FD's implied by:
- \( A, B \rightarrow C \)
- \( A, D \rightarrow B \)
- \( B \rightarrow D \)

**Example**

**Example**

**Example**

**Example**

**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.

**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 $X^+ = \{\text{all attributes}\}$, then $X$ is a superkey
  - Try reducing to the minimal $X$'s to get the key

**Example**

Product(name, price, category, color)

$name, \text{category} \rightarrow \text{price}
\text{category} \rightarrow \text{color}$

What is the key?

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

Hence (name, category) is a key

**Example**

Product(name, price, category, color)

$name, \text{category} \rightarrow \text{price}
\text{category} \rightarrow \text{color}$

What is the key?

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

Hence (name, category) is a key

**Example**

Product(name, price, category, color)

$name, \text{category} \rightarrow \text{price}
\text{category} \rightarrow \text{color}$

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:

\begin{align*}
A &\rightarrow B \\
B &\rightarrow C \\
C &\rightarrow A \\
AB &\rightarrow C \\
BC &\rightarrow A \\
A &\rightarrow BC \\
B &\rightarrow AC
\end{align*}

what are the keys here?

Eliminating Anomalies

Main idea:

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

Boyce-Codd Normal Form

Boyce-Codd Normal Form

Dr. Raymond F. Boyce

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:

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

BCNF Decomposition Algorithm

Normalize(R)

find \( X \) s.t.: \( X \neq X^- \) and \( X^- \neq \text{[all attributes]} \)

**if** (not found) **then** "R is in BCNF"

**let** \( Y = X^- - X \); \( Z = \text{[all attributes]} - X^- \)

decompose R into R1(X \cup Y) and R2(X \cup Z)

Normalize(R1); Normalize(R2);
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

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)

Iteration 2: P: age+ = age, hairColor
Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Note the keys!
Recall: find \( X \) s.t. \( X \subseteq X^* \subseteq \text{[all-attrs]} \)

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

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

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

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

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
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>Houston</td>
<td>Doc1...</td>
<td>JPG1...</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Doc2...</td>
<td>JPG2...</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Doc3...</td>
<td>JPG3...</td>
</tr>
<tr>
<td>432432</td>
<td>Ann</td>
<td>Portland</td>
<td>Doc4...</td>
<td>JPG4...</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

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

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

<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>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</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

... . . . .

Customers (ssn, name, city)

SELECT name
FROM Customers
WHERE city = 'Seattle'

Which tables are inspected by the system?

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