**Introduction to Database Systems**

**CSE 444**

**Lecture #8**

Jan 29 2001

---

**Announcements**

- **Mid Term Syllabus**
  - Material in lectures
  - Textbook
    - Chapter 1.1, Chapter 2 (except 2.1 and ODL), Chapter 3 (except 3.2, 3.8), Chapter 4.1, 4.5, 4.6, Chapter 5 (except 5.10), Chapter 6.1, 6.2, 7.1, 7.3

- **Mid Term will be in class closed book exam**

- **Key Focus: Schema Design and SQL**
  - Yes/No; Short answer; Multi-part question

- **Extended Office Hours (this week only):**
  - Surajit M, W: 4.50-5.50
  - Yana Thu 4.30-5.30 and usual hours

---

**Functional Dependencies**

Reading: Chapter 3.5, 3.6, 3.7

---

**Mapping ER Diagram to Relations**

- Entity mapped to a relation
- Many-many relationship mapped to a relation
- Some columns will be NULL-able
- May be possible to combine relations
  - Many-to-one relationships
  - Danger of redundancy: delete/update inconsistencies

---

**Example**

- Drinker(name, addr) and Favorite(drinker, beer) combined as:
  - Drinker_info(name, addr, choice_beer)
- Can you combine Drinker(name, addr) and Likes(drinker, beer)?
  - Combination introduces redundancy

---

**Need for Schema Refinement**

- Resulting schema may have redundancy
  - Inaccurate E-R modeling
  - Inappropriate combination of relations during mapping
- Functional Dependency provides a mathematical tool to detect redundancy
- Decomposition to ensure that schema does not suffer from redundancy
**Example**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E1847</td>
<td>John</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>lawyer</td>
</tr>
</tbody>
</table>

**Functional Dependencies**

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:

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

Motivating example for the study of functional dependencies:

<table>
<thead>
<tr>
<th>Name</th>
<th>Social Security Number</th>
<th>Phone Number</th>
</tr>
</thead>
</table>

**A Property of Functional Dependency**

**Splitting/Combining Lemma**

\[ \rightarrow \]

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

Is equivalent to

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

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

\[ \ldots \]

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

**Inference of Implied FD**

- **Important for**
  - Schema Redesign
  - Identifying key
- **Armstrong’s axioms**
  - Reflexivity If \( Y \subset X \) then \( X \rightarrow Y \)
  - Augmentation \( A \rightarrow B \) implies \( AX \rightarrow BX \)
  - Transitivity \( A \rightarrow B \) and \( B \rightarrow C \) implies \( A \rightarrow C \)
- **A sound and complete inference system to obtain all implied functional dependencies**

**Example**

\[ A \rightarrow B \]

\[ B \rightarrow D \]

\[ C \rightarrow D \]

Example

\[ A \rightarrow C \]

\[ A \rightarrow E \]

\[ B \rightarrow D \]

\[ A \rightarrow B \]

Let’s infer a few FD-s...
Closure of a set of Attributes

Given a set of attributes \( A = \{A_1, \ldots, A_n\} \) and a set of dependencies \( S \).

Closure(\( A \)) is the set of all attributes \( B \) such that: any relation which satisfies \( S \) also satisfies:
\[
A_1, \ldots, A_n \rightarrow B
\]

1. Closure(\( A \)) is a subset of all FDs implied
2. For a relation \( R(\( A \) \)) and a key \( B \) of \( R(\( A \)) \):
   - What is the relationship between closure (\( B \)) and \( A \)?

Keys and SuperKeys

Product: name → price, manufacturer
Person: ssn → name, age
Company: name → stock price, president

Key of a relation is a set of attributes that:
- functionally determines all the attributes of the relation
- none of its subsets determines all the attributes.

Superkey: a set of attributes that contains a key

Example

Drinkers (name, addr, likesbeer, manuf, favbeer)

What are the Keys?
Superkeys?

Closure Algorithm

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

Repeat until \( X \) doesn't change do:

if \( B_1, B_2, \ldots, B_n \rightarrow C \) is in \( S \), and \( B_1, B_2, \ldots, B_n \) are all in \( X \), and \( C \) is not in \( X \) then
add \( C \) to \( X \).

Example

\[
\begin{align*}
A & \rightarrow C \\
A & \rightarrow E \\
B & \rightarrow D \\
A & \rightarrow F
\end{align*}
\]

Closure of \( \{A, B\} \): \( X = \{A, B, C\} \)
Closure of \( \{A, F\} \): \( X = \{A, F, C\} \)

Example

\( X \)\( AB \) → \( C \), \( C \rightarrow D \), \( D \rightarrow A \)

Any "interesting" consequences?
Why Is the Algorithm Correct?

Show the following by induction:

For every $B$ in $X$:
- $A_1, ..., A_n \rightarrow B$

Initially $X = \{A_1, ..., A_n\}$ holds

Induction step: $B_1, ..., B_m$ in $X$
- $A_1, ..., A_n \rightarrow B_1, ..., B_m$
- We also have $B_1, ..., B_m \rightarrow C$
- By transitivity we have $A_1, ..., A_n \rightarrow C$

This shows that the algorithm is sound; need to show it is complete.

Relational Schema Design

Main idea:
- Start with initial relational schema
- Find out implied FD-s
- Use them to design a better relational schema

Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E1847</td>
<td>John</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>lawyer</td>
</tr>
</tbody>
</table>

Example (Contd)

- What if:
  - All current salespersons resign
  - Can I update Smith’s phone?
  - Can I add a salesperson Roy with phone 6923?

Boyce-Codd Normal Form

A relation $R$ is in BCNF if and only if:

Whenever there is a nontrivial dependency
for $R$, it is the case that $A_1, A_2, ..., A_n \rightarrow A_j, A_k, ... A_l \rightarrow B$
a super-key for $R$.

In English (though a bit vague):

Whenever a set of attributes of $R$ is determining another attribute, should determine all the attributes of $R$.

What is interesting about BCNF?

- No redundancy due to FD-s
- No update anomalies
  - Only one (unique) occurrence of a fact is updated
- No deletion anomalies
Relational Schema Design

Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>

Problems:
- redundancy
- update anomalies
- deletion anomalies

Note: SSN is NOT a key here

Decompositions in General

Let \( R \) be a relation with attributes \( A_1, A_2, \ldots, A_n \)

Create two relations \( R_1 \) and \( R_2 \) with attributes

\[
B_1, B_2, \ldots, B_m \quad C_1, C_2, \ldots, C_l
\]

Such that:

\[
B_1, B_2, \ldots, B_m \cup C_1, C_2, \ldots, C_l = A_1, A_2, \ldots, A_n
\]

And
- \( R_1 \) is the projection of \( R \) on \( B_1, B_2, \ldots, B_m \)
- \( R_2 \) is the projection of \( R \) on \( C_1, C_2, \ldots, C_l \)

Incorrect Decomposition

Sometimes it is incorrect:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Games</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>29.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Decompose on: Name, Category and Price, Category

Incorrect Decomposition

When we put it back:
Cannot recover information

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>

What are the dependencies?
What are the keys?
Is it in BCNF?

And Now?

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-321-99</td>
<td>Fred</td>
</tr>
<tr>
<td>909-438-44</td>
<td>Joe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>
What About This?

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>gadgets</td>
</tr>
</tbody>
</table>

Question:
Find an example of a 2-attribute relation that is not in BCNF.

---

Decomposition Strategy for BCNF

Find a FD that violates the BCNF condition:

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

---

Example

\* Movie (title, year, studio, president, pres_addr)
  \* Title, year \rightarrow studio, studio \rightarrow president,
    president \rightarrow pres_addr
  \* studio \rightarrow president, pres_addr
  \* Decompose: Studio(studio, president, pres_addr), Movie(title, year, studio)
  \* Decompose again?

---

Projecting FD

\* Given F over R, what is the FD that must hold over S, where S is obtained by decomposition?
  \* Compute closure(X) for each subset X of S
  \* X \rightarrow B holds in S if
    \* B in S
    \* B in closure(X)
    \* B not in X
  \* See Examples 3.39 and 3.40 in text

---

Decomposition Based on BCNF is Information Preserving

Attributes A, B, C. FD: \( A \rightarrow C \)
Relations R1[A,B] R2[A,C]
Tuples in R1: (a,b), (a,b')
Tuples in R2: (a,c), (a,c')
Tuples in the join of R1 and R2: (a,b,c), (a,b,c'), (a,b',c), (a,b',c')
Can (a,b,c') be a bogus tuple? What about (a,b',c')?

---

Problem with BCNF

\* Street, city \rightarrow zip, zip \rightarrow city
\* Keys?
\* Consider (Street, city) and (city, zip)
  \* How to check street, city \rightarrow zip?
\* 3NF
  \* Allow FD if LHS is part of a key (prime)
Problems with Decompositions

There are three potential problems to consider:

1. Some queries become more expensive.
   - e.g., find employee and department names
2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
3. Checking some dependencies may require joining the instances of the decomposed relations.

Tradeoff: Must consider these issues vs. redundancy.

Summary of Schema Refinement

If a relation is in BCNF, it is free of redundancies that can be detected using FDs.

If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations:

- Lossless-join, dependency preserving decomposition into BCNF is not always possible
- Lossless-join decomposition into BCNF is always possible
- Lossless-join, dependency preserving decomposition into 3NF is always possible

Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.

Various decompositions of a single schema are possible.