Administrivia

- HW5 is due today.
- HW6 is due Tuesday March 24th.

Database Design

- How do we design a “good” database?
  - Want to ensure the integrity of the data.
  - Want to get good performance.
- Relational DBMS vs. NoSQL DBMS

Example

Students(studentId, courseId, room, grade, name, address)

<table>
<thead>
<tr>
<th>studentId</th>
<th>courseId</th>
<th>room</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>GHC 6115</td>
<td>A</td>
<td>Christos</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>B</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>15-415</td>
<td>GHC 6115</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>15-415</td>
<td>GHC 6115</td>
<td>A</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>
Redundancy Problems

• Update Anomalies
  – *If the room number changes, we need to make sure that we change all students records.*

• Insert Anomalies
  – *May not be possible to add a student unless they’re enrolled in a course.*

• Delete Anomalies
  – *If all the students enrolled in a course are deleted, then we lose the room number.*

Today’s Class

• Motivation
• Functional Dependencies
• Armstrong’s Axioms
• Closures
• Canonical Cover

Example

<table>
<thead>
<tr>
<th>STUDENTS</th>
<th>COURSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>studentId</td>
<td>name</td>
</tr>
<tr>
<td>123</td>
<td>Christos</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROOMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>courseId</td>
</tr>
<tr>
<td>15-415</td>
</tr>
<tr>
<td>15-721</td>
</tr>
</tbody>
</table>

This Week: why this decomposition is better and how to find it.

Functional Dependencies

• A form of a **constraint**:  
  – Part of the schema to define a valid instance.

• Definition: **X → Y**  
  – *The value of ‘X’ functionally defines the value of ‘Y’.*
Functional Dependencies

- **Formal Definition:**
  - $X \rightarrow Y \Rightarrow (t_1[x] = t_2[x] \Rightarrow t_1[y] = t_2[y])$
  - *If two tuples agree on the ‘X’ attribute, then they must agree on the ‘Y’ attribute too.*

<table>
<thead>
<tr>
<th>studentId</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Christos</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

studentId $\rightarrow$ name

- **Note that the two FDs $X \rightarrow Y$ and $X \rightarrow Z$ can be written in shorthand as $X \rightarrow YZ$.**
- **But $XY \rightarrow Z$ is not the same as the two FDs $X \rightarrow Z$ and $Y \rightarrow Z$.**

Defining FDs in SQL

```sql
CREATE ASSERTION student-name
CHECK (NOT EXISTS (SELECT * FROM students AS s1,
                   students AS s2
                   WHERE s1.studentId = s2.studentId
                        AND s1.name <> s2.name))
```

**FD: studentId $\rightarrow$ name**

Make sure that no two students ever have the same id without the same name.
Combining FDs in SQL

CREATE ASSERTION student-name-address
CHECK (NOT EXISTS
(SELECT * FROM students AS s1,
students AS s2
WHERE s1.studentId = s2.studentId
AND ((s1.name <> s2.name
OR (s1.address <> s2.address))));

FD₁: studentId → name
FD₂: studentId → address

Make sure that no two students ever have the same id without the same name and address.

SQL Assertions

• WARNING: No major DBMS supports SQL-92 assertions.

  4.30.4 Assertions

  An assertion is a named constraint that may relate to the content of
  individual rows of a table, to the entire contents of a table, or to a
  state required to exist among a number of tables.

  An assertion is described by an assertion descriptor. In addi-
  tion to the components of every constraint descriptor an assertion
  descriptor includes:
  - the <search condition>.
  - An assertion is satisfied if and only if the specified <search
  condition> is not false.

Defining FDs in IBM DB2

CREATE TABLE students (studentId INT PRIMARY KEY,
name VARCHAR(32),
:CONSTRAINT student_name
CHECK (name)
DETERMINED BY (studentId ));

FD: studentId → name

Why Should I Care?

• FDs seem important, but what can we actually do with them?
• They allow us to decide whether a database design is correct.
  – Note that this different then the question of whether it’s a good idea for performance…
Implied Dependencies

**Students** (studentId, courseId, grade, name, address)

<table>
<thead>
<tr>
<th>studentId</th>
<th>courseId</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>A</td>
<td>Christos</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>B</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>15-415</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>15-415</td>
<td>A</td>
<td>Waka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

**Provided FDs**
- studentId → name, address
- studentId, courseId → grade

**Implied FDs**
- studentId, courseId → studentId
- studentId, courseId → grade, name, address

These holds for any instance!

Another Example

**Product** (name, color, category, dept, price)

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
<th>category</th>
<th>dept</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Gadget</td>
<td>Toys</td>
<td>9.99</td>
</tr>
<tr>
<td>Widget</td>
<td>Black</td>
<td>Gadget</td>
<td>Toys</td>
<td>49.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Squirrels</td>
<td>Garden</td>
<td>19.99</td>
</tr>
</tbody>
</table>

**Provided FDs**
- name → color
- category → dept
- color, category → price

**Implied FDs**
- name, category → price

Armstrong’s Axioms – Reflexivity

- **Q:** Given a set of FDs \( \{f_1, \ldots, f_n\} \), how do we decide whether FD \( g \) holds?

- **A:** Compute the closure using Armstrong’s Axioms (chapter 19.3):
  - Reflexivity
  - Augmentation
  - Transitivity

The set of all implied FDs
Armstrong’s Axioms

- **Augmentation**: If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \) for any \( Z \).
- Example: If \( \text{studentId} \rightarrow \text{name} \), then \( \text{studentId, grade} \rightarrow \text{name, grade} \)

Armstrong’s Axioms

- **Transitivity**: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \).
- Example: If \( \text{studentId} \rightarrow \text{address} \) and \( \text{address} \rightarrow \text{taxRate} \), then \( \text{studentId} \rightarrow \text{taxRate} \)

Additional Rules

- **Union**: \( (X \rightarrow Y) \land (X \rightarrow Z) \Rightarrow X \rightarrow YZ \)
- **Decomposition**: \( X \rightarrow YZ \Rightarrow (X \rightarrow Y) \land (X \rightarrow Z) \)
- **Pseudo-transitivity**: \( (X \rightarrow Y) \land (YW \rightarrow Z) \Rightarrow XW \rightarrow Z \)
Closures

- Given a set $F$ of FDs $\{f_1, \ldots, f_n\}$, we define the closure $F^+$ is the set of all implied FDs.

$\text{Students}(\text{studentId}, \text{courseId}, \text{grade}, \text{name}, \text{address})$

$F = \{\begin{align*}
\text{studentId, courseId} & \rightarrow \text{grade} \\
\text{studentId} & \rightarrow \text{name, address} \\
\text{studentId, name} & \rightarrow \text{studentId} \\
\text{studentId} & \rightarrow \text{name} \\
\text{studentId} & \rightarrow \text{address} \\
\text{studentId, grade} & \rightarrow \text{name, grade} \\
\text{studentId, courseId} & \rightarrow \text{grade, name} \\
\end{align*}\}$

$F^+$:

Students

Another Example

Product($\text{name, color, category, dept, price}$)

Provided FDs:

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

Implied FDs:

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

Why Do We Need the Closure?

- With closure we can find all FD’s easily.
- We can then compute the attribute closure
  - For a given attribute $X$, the attribute closure $X^+$ is the set of all attributes such that $X \rightarrow A$ can be inferred using the Armstrong Axioms.
- To check if $X \rightarrow A$,
  - Compute $X^+$
  - Check if $A \in X^+$

But Again, Why Should I Care?

- Maintaining the closure at runtime is expensive:
  - The DBMS has to check all the constraints for every insert, update, delete operation.
- We want a minimal set of FDs that was enough to ensure correctness.
Canonical Cover

- Given a set $F$ of FDs $\{f_1, \ldots, f_n\}$, we define the closure $F_c$ is the minimal set of all FDs.

\[
\begin{align*}
F & \quad F_c \\
\text{studentId, courseId} & \rightarrow \text{grade} \\
\text{studentId} & \rightarrow \text{name, address} \\
\text{studentId, name} & \rightarrow \text{name, address} \\
\text{studentId, courseId} & \rightarrow \text{grade, name}
\end{align*}
\]

Canonical Cover Definition

- Three properties for the canonical cover $F_c$:
  1. The RHS of every FD is a single attribute.
  2. The closure of $F_c$ is identical to the closure of $F$ (i.e., $F_c = F$ are equivalent).
  3. The $F_c$ is minimal (i.e., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated).

Computing the Canonical Cover

- Given a set $F$ of FDs, examine each FD:
  - Drop extraneous LHS or RHS attributes; or redundant FDs
  - Make sure that FDs have a single attribute in their RHS

- Repeat until no change

For #3, we need to eliminate all extraneous attributes from our set of FDs.
- An attribute is “extraneous” if the closure is the same, before and after its elimination.
Computing the Canonical Cover

$F$:  
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
A & \rightarrow BC \quad (2) \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]

$F_1$:  
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
A & \rightarrow B \quad (2') \\
A & \rightarrow C \quad (2'') \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]

Split (2)  

$F_2$:  
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
A & \rightarrow B \quad (2') \\
A & \rightarrow C \quad (2'') \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]

Eliminate (2')  

$F_3$:  
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]

Eliminate (2'')  

$F_4$:  
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]
Computing the Canonical Cover

F_4:
- B → C (3)
- A → B (4)

Eliminate (1')

F_5:
- B → C (3)
- A → B (4)

✓ Nothing is extraneous
✓ All RHS are single attributes
✓ Final & original set of FDs are equivalent (same closure)

No Really, Why Should I Care?

- The canonical cover is the minimum number of assertions that we need to implement to make sure that our database integrity is correct.
- Allows us to find the super key for a relation.

Relational Model: Keys

- **Super Key:**
  - Any set of attributes in a relation that functionally determines all attributes in the relation.
- **Candidate Key:**
  - Any super key such that the removal of any attribute leaves a set that does not functionally determine all attributes.

- **Super Key:**
  - Set of fields for which there are no two distinct tuples that have the same values for the attributes in this set.
- **Candidate Key:**
  - Set of fields that uniquely identifies a tuple according to a key constraint.
But Why Care About Super Keys?

• It is going to help us determine whether it’s okay to split a table into multiple sub-tables.
• Super keys ensure that we are able to recreate the original relation through joins.

Super Key Example

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
<th>category</th>
<th>dept</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Gadget</td>
<td>Toys</td>
<td>9.99</td>
</tr>
<tr>
<td>Widget</td>
<td>Black</td>
<td>Gadget</td>
<td>Toys</td>
<td>49.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Squirrels</td>
<td>Garden</td>
<td>19.99</td>
</tr>
</tbody>
</table>

Provided FDs

name → color

Provided FDs

name → price

Implied FDs

color, category → price

Summary

• How do we guarantee that F = F’?
  – Closures
• How do we find a minimal F’ for F?
  – Canonical Cover