CPSC 421
Database Management Systems

Lecture 9:
Normalization and Physical Data Independence

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Today’s Agenda

• More on Database Design
  – Normalization
  – Views
Boyce-Codd Normal Form (BCNF)

- A relation is in "Boyce-Codd Normal Form" if all of its FDs are either
  - Trivial FDs (e.g., \( AB \rightarrow A \))
  - or
  - Key FDs

- Which (if any) of these relations is in BCNF?

EmpDept(EmplD, Name, Dept, DeptName)
Assigned(EmplID, JobID, EmpName, Percent)
Enrollment (StdntID, ClassID, Grade, InstrID, StdntName, InstrName)

BCNF and Redundancy

BCNF relations have no redundancy caused by FDs
- A relation has redundancy if there is an FD between attributes
- … and there can be repeated entries of data for those attributes

- For example, consider

<table>
<thead>
<tr>
<th>DeptID</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

- if the relation is in BCNF, then the FD must be a key FD, and so DeptID must be a key
- implying that any pair such as <12, CS> can appear only once!
Decomposition into BCNF

- An algorithm for decomposing a relation R with attributes A into a collection of BCNF relations

1. if R is not in BCNF and X → Y is a non-key FD then
2. decompose R into A – Y and X U Y
3. if A – Y and/or X U Y is not in BCNF then
4. recursively apply step 1 (to A – Y and/or X U Y)

Example

Enrollment(StdntID, ClassID, Grade, InstrID, StdntName)

- First use the non-key FD StdntID → StdntName
- … which gives the decomposition
  Enrollment(StdntID, ClassID, Grade, InstrID)
  Student(StdntID, StdntName)

- Now use the non-key FD ClassID → InstrID
- … which gives the decomposition
  Enrollment(StdntID, ClassID, Grade)
  ClassInstructor(ClassID, InstrID)
  Student(StdntID, StdntName)

- All relations are now in BCNF!
Another Example

- Given the schema:
  Loans(BranchID, LoanID, Amount, Assets, CustID, CustName)
- and assuming FDs
  BranchID → Assets
  CustID → CustName
- … lets Decompose it into BCNF relations

Answer:
Loans(BranchID, LoanID, Amount, CustID)
Customer(CustID, CustName)
Branch(BranchID, Assets)
  Loans.BranchID REFERENCES Branch.BranchID
  Loans.CustID REFERENCES Customer.CustID

Where are we?

- We have accomplished a lot!
  - we began with a relational schema
  - we identified problems with redundancy
  - we used FDs to eliminate those problems with decompositions into BCNF
  - along the way we learned how to identify keys using FDs
- There are two steps left …
  - ensure that the BCNF decompositions do not lose information (are not “lossy”)
  - it turns out that in some cases we may lose FDs, but there is a way to deal with this
Lossless Decomposition

Some decompositions may lose information content

• For example, lets say we decomposed:
  \[ \text{Enroll(StdntID, ClassID, Grade)} \]
  into:
  \[ \text{StudentGrade(StdntID, Grade)} \]
  \[ \text{ClassGrade(ClassID, Grade)} \]
  – a row (223, A) in StudentGrade implies student 223 received an A in some course
  – and a row (421, A) in ClassGrade means that some student received an A in course 421
  – but now we have no way to recreate the original table!
• This decomposition is “Lossy”

Lossless Decomposition

A decomposition of a schema with FDs F into attribute sets X and Y is “lossless” if for every instance R that satisfies F:

\[ R = \pi_X(R) \bowtie \pi_Y(R) \]

• That is, we can recover R from the natural join of the decomposed versions of R
### Example of a Lossless Decomposition

We have a relation $R$ which contains information about employees and their departments:

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

We define $X = \{\text{EID}, \text{Name}, \text{Dept}\}$ and $Y = \{\text{Dept}, \text{DeptName}\}$.

Then, $X = \pi_X(R) = \pi_X(R)$ and $Y = \pi_Y(R)$.

### Example of a Lossy Decomposition

We have a relation $R$ which contains information about students and their grades:

<table>
<thead>
<tr>
<th>StdntID</th>
<th>ClassID</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>CS223</td>
<td>A</td>
</tr>
<tr>
<td>456</td>
<td>CS421</td>
<td>A</td>
</tr>
</tbody>
</table>

We define $X = \{\text{StdntID}, \text{Grade}\}$ and $Y = \{\text{ClassID}, \text{Grade}\}$.

Then, $X = \pi_X(R) = \pi_X(R)$ and $Y = \pi_Y(R)$, but

$$X \Join Y \neq R$$
Producing only Lossless Decompositions

We only want to produce lossless decompositions

• This is easy to guarantee:

• The decomposition of $R$ with respect to FDs $F$ into attributes sets $A_1$ and $A_2$ is lossless if and only if $A_1 \cap A_2$ contains a key for either $A_1$ or $A_2$
  – If they have a key in common, they can be joined back together
  – Note that $\{\text{StdntID, Grade}\} \cap \{\text{ClassID, Grade}\} = \{\text{Grade}\}$
  – See page 620 in the text

• This implies that the BCNF decomposition algorithm produces only lossless decompositions
  – In this case $F$ includes the FD $X \rightarrow Y$ and the decomposition is $A_1 = A - Y$ and $A_2 = XUY$
  – Therefore $A_1 \cap A_2 = X$ is a key for $XUA$

Producing only Lossless Decompositions

• Given the schema $R(S, C, G)$ with FD $SC \rightarrow G$

• Is the decomposition into $R_1(S, G)$ and $R_2(C, G)$ lossless or lossy? Why?
  – Take the intersection of the two sets $\{S, G\} \cap \{C, G\} = G$
  – Then determine if $G$ is a key for either table
  – That is, does $G \rightarrow C$?
    • NO
  – Does $G \rightarrow S$?
    • NO
  – Therefore, this decomposition is lossy!
Dependency Preserving Decompositions

Decompositions should also preserve FDs.

• For example
  \[ \text{Emp}(EID, \text{Addr}, \text{City}, \text{State}, \text{Zip}) \]
  \[ \rightarrow \text{Addr}, \text{City}, \text{State} \rightarrow \text{Zip} \]
  \[ \rightarrow \text{Zip} \rightarrow \text{State} \]

• Consider this decomposition
  \[ \text{Emp}(EID, \text{Addr}, \text{City}, \text{Zip}) \]
  \[ \text{ZipState}(\text{Zip}, \text{State}) \]

• Although this is BCNF, it does not preserve the FD
  \[ \rightarrow \text{Addr}, \text{City}, \text{State} \rightarrow \text{Zip} \]

• Here are some values
  \[ (123, 111 W 1^{st}, \text{Spokane}, 99999) \quad (99999, \text{WA}) \]
  \[ (456, 111 W 1^{st}, \text{Spokane}, 00000) \quad (00000, \text{WA}) \]

Dependency Preserving Decompositions

• Let \( R \) be a schema with FDs \( F \) and \( X, Y \) sets of attributes in \( R \)

• A dependency \( A \rightarrow B \) is in \( X \) if all attributes of \( A \) and all attributes of \( B \) are in \( X \)

• The projection \( F_X \) of dependencies \( F \) on attributes \( X \) is the closure of the FDs in \( X \)

• The decomposition of \( R \) into schemas with attributes \( X \) and \( Y \) is “dependency preserving” if \( (F_X \cup F_Y)^+ = F^+ \)
Example

Consider Emp(Addr, City, State, Zip) with
\[ F = \{ \text{Addr, City, State} \rightarrow \text{Zip}, \text{Zip} \rightarrow \text{State} \} \]

- If we decompose Emp so that \( X = \{ \text{Addr, City, Zip} \} \) and \( Y = \{ \text{Zip, State} \} \) what are the projections \( F_X \) and \( F_Y \)?
  \[ F_X = \emptyset \quad (\text{Addr, City, State} \rightarrow \text{Zip not in X, Zip} \rightarrow \text{State not in X}) \]
  \[ F_Y = \{ \text{Zip} \rightarrow \text{State} \} \quad (\text{Zip} \rightarrow \text{State} \text{ is in Y}) \]

- Is \( X, Y \) a dependency preserving decomposition?
  - No … \( (\text{Zip} \rightarrow \text{State})^+ \) does not contain \( \text{Addr, City, State} \rightarrow \text{Zip} \)
    and so it can never recreate \( F^+ \)

Third Normal Form (3NF)

Some schemas do not have both a lossless and dependency preserving composition into BCNF schemas

- Every schema has a lossless dependency preserving decomposition into 3NF …

- A schema \( R \) with FDs \( F \) is in 3NF if for every \( X \rightarrow Y \) in \( F \) either:
  - \( X \rightarrow Y \) is a trivial FD (i.e., \( X \supseteq Y \))
  - \( X \rightarrow Y \) is a key FD (i.e., \( X \) is a superkey)
  - \( Y \) is a part of some key for \( R \)

Definition of BCNF
Third Normal Form (3NF)

In other words, 3NF allows FDs that only partially (i.e., do not fully) depend on the key …

• For Emp(Addr, City, State, Zip) with
  \[ F = \{ \text{Addr, City, State} \rightarrow \text{Zip}, \text{Zip} \rightarrow \text{State} \} \]
  – the keys are: \((\text{Addr, City, State})\) and \((\text{Addr, City, Zip})\)

• Although there is no decomposition of this relation into BCNF …

• This relation is in 3NF!

Wrapping up

• Almost all schemas can be decomposed into BCNF schemas that preserve all FDs
  – But every once in a while we get a schema like the previous one

• So, if we do not have an ideal decomposition (lossless, dependency preserving) into BCNF, we can decompose into 3NF and have a lossless and dependency-preserving schema
  – But with some minor redundancy
Views

A “view” is a query that is stored in the database and that acts as a “virtual” table

• For example:
  
  ```sql
  CREATE VIEW astudents AS
  SELECT *
  FROM Students
  WHERE gpa > 3.0;
  ```

• Views can be used just like base tables within another query or in another view
  
  ```sql
  SELECT *
  FROM astudents
  WHERE age > 20;
  ```

Implementing Views

• The DBMS expands (i.e., rewrites) your query to include the view definition
  
  ```sql
  SELECT ClassID
  FROM astudent S, enrollment E
  WHERE S.StdntID = E.StdntID
  ```

• This query is expanded to
  
  ```sql
  SELECT ClassID
  FROM (SELECT * FROM student WHERE gpa >= 4.0) AS S, enrollment E
  WHERE S.StdntID = E.StdntID;
  ```
Views for Security

- For a base table:
  
  Student(StdntID, SSN, Name, Address, Telephone, Email, …)

- This view gives a “secure” version of the student relation

  CREATE VIEW sstudent AS
  SELECT StdntID, Name, Address
  FROM Student;

- Here, using the view we avoid exposing the SSN, Telephone, Email, etc.

Views for Integration

- Different companies might have different but similar “parts” databases

  PartsCo1(PartID, weight, …)
  PartsCo2(PartID, weight, …)

- We can combine these parts DBs into a single version using a view definition

- For instance, if company 1 uses pounds and company 2 uses kilograms for part weights:

  CREATE VIEW Part AS
  (SELECT PartID, 2.2066*weight, ...
  FROM PartsCo1)
  UNION
  (SELECT PartID, weight, ...
  FROM PartsCo2);
View Update Problem

• Views cannot always be updated unambiguously
• For example, for
  – Students(stdntid, gpa, deptid, …)
  – Department(deptid, dname, office, head, …)
• And views
  
  CREATE VIEW major_gpa AS
  SELECT major, AVG(gpa)
  FROM Students
  GROUP BY major

  CREATE VIEW std_dept AS
  SELECT stdntid, dname
  FROM Students JOIN Department
    USING (deptid)

• How do we change the GPA of CS majors from 3.5 to 3.6 using major_gpa?
• How do we delete a row (e.g., <jim, cpsc>) from std_dept?

View Update Problem

• A view can in general be updated if
  – It is defined over a single base table
  – It uses only selection and projection
  – It does not use aggregates, group by
  – It does not use DISTINCT
  – It does not use set operations (UNION, INTERSECT, MINUS)

• Different products provide different support for views, especially w.r.t updates
• Many more details not discussed here
Data Independence

- Multiple levels of abstraction support data independence
  - Changes isolated to their “levels”
  - This is very desirable since things change often!

**External View:**
What application programmers see

**Logical View:**
The conceptual or logical relations

**Physical View:**
Optimized/normalized relations including indexes

**Physical Storage (on disk(s) …)**

For Thursday

- Reading
  - Ch 19: 19.5-19.6 (up to 19.6.1)

- Be sure to understand:
  - Normalization, BCNF, 3NF, algorithms

- Homework
  - Homework 3 due on Thursday
  - Part 2 of the project coming soon (developing an ER model for your application data)