SWEN304: Normalization Algorithms

Dr. Dionysios Athanasopoulos

Lecturer

dionysios.athanasopoulos@vuw.ac.nz

Office: EA111, Easterfield building, Kelburn Campus

Normalization Algorithms

Given a universal relation and a set of functional dependencies, a normalization algorithm produces a set of relations whose desirable properties are:

- attribute preservation
- dependency preservation
- lossless join decomposition.

The most widely used normalization algorithms are:

- Synthesis algorithm (3NF Normalization)
- Decomposition algorithm (BCNF normalization)

Synthesis Algorithm: 3NF Normalization

Synthesis algorithm

The algorithm decomposes a universal relation U into a set of relations S that are in 3NF based on a set of functional dependencies F.

Properties:

- At least 3NF
- Attribute preservation
- Functional-dependency preservation

Synthesis Algorithm: 3NF Normalization

Steps of synthesis algorithm:

Input: U, FOutput: S

- 1: find a minimal cover G of F
- 2: **group** FDs from *G* according to the same left-hand side
- 3: **for all** groups of FDs **do** a new relation in S
- 4: **place** any remaining attributes in a single relation, along with a key of the universal relation, to ensure the attribute preservation property.

Minimal Cover

G is **minimal cover** of a set of functional dependencies F if

- every FD in G has a single attribute in its right-hand side (canonical form);
- we cannot remove any dependency from G and still have a set of dependencies that is equivalent to G.

Algorithm for Finding Minimal Cover

```
Input: U, F
Output: G
1: G \leftarrow F
\triangleright Check for canonical form.
```

- 2: **for all** $X \to \{A_1, A_2, ..., A_n\}$ in G **do replace** them with the dependencies $X \to A_1, X \to A_2, ..., X \to A_n$ (decomposition inference rule)
 - > Check for redundant attributes.
- 3: for all $X \rightarrow A$ in G do
- 4: **for all** attributes *B* that are elements of *X* **do**
- 5: **if** $\{G \{X \to A\}\} \cup \{(X \{B\}) \to A\}$ is equivalent to G **then** 6: **replace** $X \to A$ with $(X \{B\}) \to A$ in G
 - Check for redundant functional dependencies.
- 7: **for all** remaining functional dependencies $X \rightarrow A$ in G **do**
- 8: **if** $\{G \{X \rightarrow A\}\}\$ is equivalent to G then
- 9: **remove** $X \rightarrow A$ from G

Minimal Cover: Example

Minimal cover of $B \rightarrow A, D \rightarrow A, AB \rightarrow D$:

- Steps 1-2: completed
- Steps 3-6: check if there is any redundant attribute in $AB \rightarrow D$:
 - $B \rightarrow A \models BB \rightarrow AB$ (augmenting with B both sides) \models
 - $B \rightarrow AB \models B \rightarrow AB$ and $AB \rightarrow D \models B \rightarrow D$ (transitive rule).
 - Thus, $AB \rightarrow D$ can be replaced by $B \rightarrow D$.
- Steps 7-9:
 - $B \rightarrow D$ and $D \rightarrow A \models B \rightarrow A$ (transitive rule).
 - Thus, $B \rightarrow A$ is redundant.

Synthesis Algorithm: Example

Relation schema: Faculty

Functional dependencies *R*:

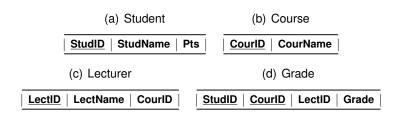
StudID → StudName, StudID → Pts, CourID → CourName, LectID →
 LectName, LectID → CourID, StudID+CourID → Grade, StudID+CourID −
 LectID

Synthesis algorithm:

- Step 1: completed $(G \equiv F)$
- Step 2: groups of functional dependencies:
 - {StudID → StudName, StudID → Pts}
 - {CourID → CourName}
 - {LectID → LectName, LectID → CourID}
 - $\bullet \ \{\mathit{StudID} + \mathit{CourID} \rightarrow \mathit{Grade}, \ \mathit{StudID} + \mathit{CourID} \rightarrow \mathit{LectID}\}$
- Step 3: creation of new relations:

Synthesis Algorithm: Example

• Step 3: creation of new relations:



Step 4: completed.

Observations:

- The above decomposition is attribute & dependency preserving
- The above decomposition is not in BCNF and hence, it is not free of update anomalies:
 - an update anomaly would arise between the relations Lecturer and Grade if a lecturer starts teaching another course.

Synthesis Algorithm: Exercise

Relation schema: U

Functional dependencies:

- lacktriangledown A
 ightarrow B
- $\bullet \ B \to C$

Is U in 3NF? If not, decompose it in 3NF.

BCNF Decomposition

BCNF decomposition

Given a set of functional dependencies, it decomposes a universal relation schema into a set of relation schemas that have the following properties:

- BCNF
- Attribute preservation
- Lossless join decomposition

Lossless join decomposition

A decomposition to a set of relations has the lossless (nonadditive) join property if the natural join of all the relations **does not lose tuples** and **does not give additional spurious tuples**.

BCNF Decomposition

```
Input: U, F
Output: S
```

- 1: $S \leftarrow U$
- 2: **while** there is a relation schema G in S that is not in BCNF **do**
- 3: **find** a functional dependency $X \rightarrow Y$ in G that violates BCNF
- 4: **replace** G by **two** new relation schemas, G Y and $X \cup Y$.

Relation schema: Faculty *U*

ne LectID LectName Grade	CourName	CourID	Pts	StudName	StudID
--------------------------------	----------	--------	-----	----------	--------

Functional dependencies R:

- StudID → StudName + Pts
- CourID → CourName
- LectID → LectName + CourID
- StudID + CourID → Grade + LectID

BCNF Decomposition: 1st Iteration

```
Output: S

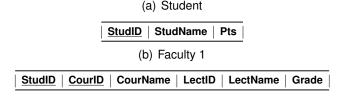
1: S \leftarrow U

2: while there is a relation schema G in S that is not in BCNF do

3: find a functional dependency X \rightarrow Y in G that violates BCNF

4: replace G by two new relation schemas, G - Y and X \cup Y.
```

- Given that the primary key is {StudID, CourID} and that there is the dependency StudID → StudName + Pts, Faculty is not in BCNF.
- So, we decompose *U* as follows:



Input: U, F

BCNF Decomposition: 2nd & 3rd Iterations

- Given that the primary key is {StudID, CourID} (recalculating it) and that there is the dependency, CourID → CourName (updating the dependencies), Faculty1 is not in BCNF.
- So, we decompose Faculty1 as follows:



- Given that the primary key is {StudID, CourID} (recalculating it) and that there is the dependency, LectID → LectName + CourID (updating the dependencies), Faculty2 is not in BCNF.
- So, we decompose Faculty2 as follows:



BCNF Decomposition: Final Result

- All produced relation schemas are in BCNF.
- But, we have lost the FD, $StudID + CourID \rightarrow Grade + LectID$.

Relation schema: Student

StudID | StudName | Pts

Relation schema: Course

CourlD CourName

Relation schema: Lecturer

| LectID | CourID | LectName

Relation schema: Grade

 $\underline{\text{StudID}} \mid \underline{\text{LectID}} \mid \text{Grade}$

BCNF Decomposition: Exercise



Functional dependencies:

- ullet $A \rightarrow B$
- $\bullet \ B \to C$
- $\bullet \ AC \to D$

Apply the BCNF-decomposition algorithm.

Synthesis Algorithm vs. BCNF Decomposition

Conclusions:

- Synthesis algorithm
 - may loss information
 - preserves functional dependencies.
- BCNF decomposition
 - may loss dependencies
 - guarantees nonlossy design.
- Nonlossy design is a must.
- Dependency preservation is desirable but not a must.
- How can we guarantee both conditions?

Dependency-preserving and Lossless Decomposition in 3NF

Input: U, FOutput: S

- 1: find a minimal cover G of F
- 2: **group** FDs from *G* according to the same left-hand side
- 3: for all each group of FDs do make a new relation in S
- 4: **if** none of the relation schemas in S contains a key of U **then make** one more relation schema in S that contains attributes that form a key of U
- 5: **eliminate** redundant relations from *S* (a.k.a. projections of existing relations).

Relation schema: Faculty

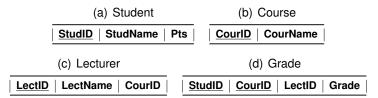
StudID StudName	Pts C	ourID Cour	Name LectID	LectName	Grade
-------------------	---------	--------------	---------------	----------	-------

Functional dependencies:

 StudID → StudName+Pts, CourID → CourName, LectID → LectName+ CourID, StudID + CourID → Grade + LectID

Dependency-preserving and Lossless Decomposition in 3NF

- Step 1: completed $(G \equiv F)$
- Step 2: groups of functional dependencies:
 - $\qquad \qquad \textbf{(StudID} \rightarrow \textbf{StudName, StudID} \rightarrow \textbf{Pts} \textbf{)}$
 - {CourID → CourName}
 - {LectID → LectName, LectID → CourID}
 - {StudID + CourID → Grade, StudID + CourID → LectID}
- Step 3: creation of new relations:



- Step 4: completed (there is relation schema that contains the key of *U*)
- Step 5: completed (no redundant relations).

Observations:

- The decomposition is **lossless** and **dependency preserving**.
- It is not in BCNF and hence, it is not free of update anomalies.

Lossless Join Decomposition

Lossless decomposition

A decomposition $D = \{R_1, R_2, ..., R_m\}$ of R has the lossless (non-additive) join property w.r.t. a set of dependencies F on R if for every relation state r of R that satisfies F, the following holds:

• join $(\pi_{R_1}(r), ..., \pi_{R_m}(r)) = r$.

Lossless decomposition in two relations

A decomposition D = $\{R_1, R_2\}$ of R is lossless if $R_1 \cap R_2$ contains a key of R_1 or a key of R_2 ;

this condition is not generalizable for many relations.

Algorithm for Testing Lossless Join Decomposition

Algorithm

Build a universal relation schema using the following algorithm.

Steps of algorithm for testing lossless join decomposition:

```
    Input: F, D
    Output: U
    1: U ← ∅
    2: pick up two schemas, D<sub>1</sub> and D<sub>2</sub>: D<sub>1</sub> ∩ D<sub>2</sub> contains the key K<sub>1</sub> or K<sub>2</sub>
    3: remove D<sub>1</sub> and D<sub>2</sub> from D
    4: add D<sub>1</sub> and D<sub>2</sub> in U and set the key of U as K<sub>1</sub> or K<sub>2</sub>
    5: repeat
    6: pick up a schema, D<sub>3</sub>: U ∩ D<sub>3</sub> contains the key of U or K<sub>3</sub>
    7: remove the schema D<sub>3</sub> from D
    8: add the schema D<sub>3</sub> in U and keep the same key for U
    9: until no more relations D<sub>i</sub> can be added in U
```

Lossless Join Decomposition: Example

Assuming that the universal relation schema key is *AC*, is the following decomposition **lossless**?

- $D_1(\underline{A}, B)$
- $D_2(\underline{B}, D)$
- D₃(B, <u>C</u>)
- 1st execution:
 - a. $D_1(\underline{A}, B) \cap D_2(\underline{B}, D)$ contains the key of D_2
 - b. $U(A, \underline{B}, D) \cap D_3(B, \underline{C})$ does not contain the key of D_3

The relation schema D_3 is left in D.

- 2nd execution:
 - a. $D_2(\underline{B}, D) \cap D_3(B, \underline{C})$ contains the key of D_2
 - b. $U(\underline{B}, C, D) \cap D_1(\underline{A}, B)$ does not contain the key of D_1

The relation schema D_1 is left in D.

- 3rd execution:
 - $D_1(\underline{A}, B) \cap D_3(B, \underline{C})$ does not contain the key of D_1 or D_3 .

The relation schemas D_1 , D_2 , and D_3 are left in D.

Algorithm for Testing Lossless Join Decomposition

Checking the algorithm output

- If there are some relation schemas left in *D*, we apply the previous algorithm starting with another relation schema from *D*.
- If there are no relation schemas left in D and the constructed relation schema U contains the provided universal key, then D is lossless join decomposition.
- If after any execution of the algorithm, we have not built a universal relation schema that includes all the relation schemas in D, then D is a lossy join decomposition.

Lossless Join Decomposition: Exercises 1-2

Exercise 1:

Assuming that the universal relation schema key is *AC*, is the following decomposition **lossless**?

- $D_1(A, B)$
- $D_2(\underline{B}, D)$
- D₃(B, <u>C</u>)
- $D_4(A, C)$

Exercise 2:

Functional dependencies *F*:

ullet B o C

Is the decomposition $D = \{D_1, D_2\}$

- $D_1 = \{A, B\}, F_1 = \{\}$
- $D_2 = \{B, C\}, F_2 = \{B \rightarrow C\}$

lossless?

References

- ElMasri, Navathe, Fundamentals of Database Systems, 6th Edition, Addison Wesley.
- Hui Ma & Pavle Mogin, SWEN304 Lecture Slides, 2016 https://ecs.victoria.ac.nz/Courses/SWEN304_2016T2/LectureSchedule