Lecture 05: Normal Forms

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Normal Forms

- Now that we know how to derive FDs, we can:
  1. Search for "bad" FDs
  2. If they exist, then decompose them into two tables, repeat for sub-tables
  3. When done, the database schema is normalized
- A **normal form** is a characterization of a decomposition in terms of the properties that satisfied when putting the relations back together
- **Universal relation**: The joining of all tables → This is a BAD thing.
- Recall: Three properties:
  1. Lossless Joins: Information is not lost or bad information is not created when joining
  2. Dependency Preservation: FDs are not split across relations
  3. Redundancy avoidance: No repeated attributes in tuples
- **History**
  - Ted Codd introduced the concept of normalization and the first normal form
  - Codd Went on to define second and third normal form
  - Codd and Raymond Boyce later defined Boyce-Codd normal form
- The ith normal form is **more restrictive** than the (i-1)th normal form
- **Most common/important ones are the 3rd or Boyce-Codd normal Form**

It can be easily implemented in practice.

Types of Normal Forms → Theoretical guidance for db design

1. 1st normal form (1NF): All tables are flat
   - All types must be atomic
   - No repeating groups

   → No "list" as attribute. one more element to list → one more col. → Not a 1NF.
2. 2nd Normal form (2NF): "Good enough"
   - Must be in first normal form
   - Any non-key attributes fully depend on the candidate key

   → Functional dependency.

3. 3rd Normal form (3NF): Most common
   - Always preserves dependencies (unlike BCNF) but may have some anomalies

→ redundant information exists.

4. Boyce-Codd Normal form (BCNF): Most common
   - No redundancies and no lossless join
   - For any FD, if any left hand side attributes are not a super key, the relations are not in BCNF
   - **Some BCNF decompositions may lose dependencies when decomposed relations are joined back together**

(Dependency preservation may be violated)

F+.

Attr → Row

ex) A→B, B→C. R(A,B,C).
A→ABC (True)
B→ABC (False) → X.

Cannot reconstruct F from each F_is.

5. 4th and 5th Normal Forms: See textbooks
6. 6th Normal Form: Most (normal) people never need this

NoSQL

- The normal forms is usually not how people design databases
- Instead, people usually think in terms of object-oriented programming → ORM.
- Key tenants of the NoSQL movements
  1. Prior to early 2000s, few people needed high-performance DBMS. In modern day speed is very important
  2. Joins are slow, so we will denormalize tables
  3. Transactions are slow

Document data model blurs the seperation between physical layout and the logical layout.
⇒ one less abstraction layer : BAD

Conclusion

- You should know about normal forms, they exist
- There is no magic formula for determining the right amount of normalization for an application

Given a schema R and a set of FDs F, we can always decompose R into $\{R_1, \ldots, R_n\}$ such that
→ $\{R_1, \ldots, R_n\}$ are in BCNF
→ The decompositions are lossless.

BCNF DECOMPOSITION ALGORITHM

Given a relation R and a FD set F:
Step #1 – Compute F+
Step #2 – Result ← {R}
Step #3 – While $R_i \in$ Result not in BCNF, do:
→ **(a)** Choose $(X \to Y) \in F^+$ such that $(X \to Y)$ is covered by R_i and $X \nrightarrow R_i$
→ **(b)** Decompose R_i on $(X \to Y)$:

$R_{i,1} \leftarrow X \cup Y$ ← $R_{i,1}$ includes Y
$R_{i,2} \leftarrow R_i - Y$ ← $R_{i,2}$ does not include Y

$Result \leftarrow (Result - \{R_i\}) \cup \{R_{i,1}, R_{i,2}\}$

BCNF

A relation R with FD set F is in 3NF if for every $X \to Y$ in F^+:
→ $X \to Y$ is trivial, or
→ X is a super key, or
→ Y is part of a candidate key

3NF DECOMPOSITION ALGORITHM

Given a relation R and a FD set F:
Step #1: Compute F_c
Step #2: Result ← ∅
Step #3: For $(X \to Y) \in F_c$, add a relation $R_i(X,Y)$ to Result
Step #4: If Result is not lossless, add a relation with an appropriate key.

3NF.