More Normalization Algorithms

CS157A

Chris Pollett

Nov. 28, 2005.

Outline

- 3NF Decomposition Algorithms
- BCNF Algorithms
- Multivalued Dependencies and 4NF

Dependency-Preserving Decomposition into 3NF

Input: A universal relation R and a set of FDs F on the attributes of R

Output: D -- a decomposition of R

Algorithm:

1. Find a minimal cover G for F.
2. For each left hand side X of an FD in G add a new relation schema Q in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where X--> $A_1, \ldots,$ X-->A_k are the only FDs in G with X as the left hand side. Make X the key of this schema.
3. Place any remaining attributes in a single relation to ensure the attribute preservation property.

Notice Step 2 guarantees dependencies will be preserved. The reason why the decomposition will be in 3NF is due to the use of a minimal cover. (If X--> Z and Z-->A are in G then X-->A won’t be in G.)

Lossless Join Decomposition into BCNF

Input: A universal relation R and a set of FDs F on the attributes of R

Output: D -- a decomposition of R

Algorithm:

```
Set D:={R};
While there is a relation schema Q in D that is not in BCNF
{
        1. Choose a relation Q in D not in BCNF
        2. Rind a FD X-->Y in Q that violates BCNF
        3. Replace Q in D by the two relation schemas (Q-Y) and (X ∪Y)
}
```

Since relations only have finitely many attributes this algorithm terminates with all the relations in BCNF. (3) ensures that our binary decomposition test for LJP will be passed.

Dependency-Preserving Lossless Join Decomposition into 3NF

Input: A universal relation R and a set of functional dependencies F on the attributes of R.

Output: D -- a decomposition of R

Algorithm:

1. Find a minimal cover G for F.
2. For each left hand side X of an FD in G add a new relation schema Q in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \to A_1, \ldots, X \to A_k$ are the only FDs in G with X as the left hand side. Make X the key of this schema.
3. If none of the schemas in D contains a key of R, then create one more relation schema that contains attributes which form a key of R.

Except for Step 3 this is like our earlier algorithm. The new step 3 still preserves attributes because a key needs to have all attributes not in any FD in G. Also, Step 3 in above ensures LJP.

Algorithm to Find a Key of R

In order to do Step 3 above we need an algorithm which can find a key for R. Here is how to do this:

Input: A universal relation R and a set of FDs F on the attributes of R.

Output: K -- a key

Algorithm:

```
1.    Set K:= R
2.    For each attribute A in K
      {
          Check $(K-A)^+$ == R.
          If Yes set K := K-A;
      }
```

Problems with Null Values and Dangling Tuples

- Our decomposition algorithms don’t really address how to handle nulls in our relation schemas.
- To see why this is hard consider our original COMPANY schema. It is in BCNF. Suppose we have employees with null values for their DNO. If we natural join EMPLOYEE with DEPARTMENT, these employees disappear. On the other hand using left outer joins we get a new fictitious department with null for all its attributes.
- Similarly, SUM and AVERAGE over columns with nulls must be carefully considered.
- We might try to come up with a decomposition property which further split EMPLOYEE into a table EMP1 which does not have DNO and a table EMP2(SSN,DNO). If we do not store in EMP2 employees with a NULL value for DNO then we have gotten rid of all NULLS
- Unfortunately, if we natural join EMP1 and EMP2 we might not get all the rows we had in EMPLOYEE. The tuples which were lost were in only one of the two relations (EMP1) of the decomposition. These are called **dangling tuples**.

Discussion of Normalization Algorithms

Some other issues which make it hard to totally rely on decomposition algorithms are:

1. It is hard to know in advance all the FDs that apply to the data.
2. The algorithms depend on certain nondeterministic choices which we have forced by how we ordered our FDs. For instance, there are usually many different minimal covers. We calculate one based on our initial ordering of our FDs. This might be suboptimal.

Notice also, we can’t in general decompose into BCNF and preserve dependencies. (Lots example last day is an example why not.)