Functional Dependencies and Normalization

CS157A

Chris Pollett

Nov. 14, 2005.

Outline

- Informal Design Guidelines for Relation Schemas
- Functional Dependencies

Introduction

- To this point we've heuristically learned how to come up with a choice of relation schemas for a database based on ER modeling.
- There are informally two levels at which we can judge the resulting tables:
  (1) The logical level --do the tables make sense to DB users.
  (2) The implementation level -- are the tables physically stored with indexes, etc appropriate for their typical use.
- Today, we will be interested in formal approaches to saying if our schemas are good.

Schema quality: What should we optimize?

- Semantics of attributes
- Reducing the number of redundant values in tuples
- Reducing the null values in tuples
- Disallowing the possibility of generating spurious tuples.

Semantics of the Relation Attributes

- We should be grouping our attributes into relations so that they have some real-world meaning.
- This meaning (aka **semantics**) tells us how to interpret the attributes values stored in some tuple.
- For instance, a BDate value in a tuple in the EMPLOYEE table should be interpreted as someones birthday.
- The ease with which we can explain each attribute in a table is an informal measure of how well the relation is designed.
- So a first guideline on relation schema design is to make sure that each schema has some easy to explain meaning. One should not combine attributes from multiple entity types and relationship types into a single relation.

Redundant Information

- Another goal in relation schema design is to reduce the storage space used by the base relations.
- As an example, suppose we combined the attributes of EMPLOYEE and DEPARTMENT into one table EMP_DEPT(EName, SSN, BDate, Address, DNumber, DName, DMGRSSN)
- Then the same values for DNumber, DName, DMGRSSN are repeated for each employee who works for a given department.
- This could be minimized if used DNumber as a foreign key reference to a different DEPARTMENT table.
- Further, such a table is prone to several kinds of update anomalies which we'll discuss on the next slide.

Update Anomalies

- Insertion Anomalies -- to inset into EMP_DEPT a new employee involves padding a tuple with nulls until that employee is assigned a department. Need to also check that for each row insert the values are consistent with a given department. It is unclear how one would insert a new department at all as SSN is the key for EMP_DEPT.
- Deletion Anomalies -- deleting the last employee who works for a department deletes the department.
- Modification Anomalies - changing the manager of a department involves changing many rows.

Thus, another design guideline should be to design tables so that Insertion, Deletion, and Modification anomalies are not possible.

Null Values in Tuples

- As we've said before null values can have several interpretations:
  - The attribute does not apply
  - The attribute value for this tuple is unknown
  - The value is known but absent; that is, it has not been recorded yet.
- Therefore, it is good to have another guideline which says as far as possible avoid placing attributes in a base relation whose values frequently be null.
- As an example, if only 10% of employees have offices rather than add an office number attribute to employee make a separate table EMP_OFFICES(ESSN, OfficeNo).

Generation of Spurious Tuples

- Consider the tables
  - EMP_LOCS(EName, PLocation)
  - EMP_PROJ1(SSN, PNumber, Hours, PName, PLocation)
- versus the table
  - EMP_PROJ(SSN, PNumber, Hours, EName, PName, PLocation)
- If we use the former as our base tables then we cannot recover all the information of the latter because trying to natural join the two tables will produce many rows not in EMP_PROJ.
- These extra rows are called **spurious tuples**.
- Another design guideline is that relation schemas should be designed so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way such that no spurious tuples are generated.

Functional Dependencies

- We now try to come up with formal ways to ensure some of the guidelines we've listed above.
- **Definition:** A functional dependency

  X-->Y, between two sets of attributes of some schema R indicates that whenever s,t are tuples in R it holds that s[X]=t[X] implies s[Y] = t[Y].
- We abbreviate functional dependency as FD.
- If X-->Y we say Y is functionally dependent of X.
- If X is a superkey of R then X-->Y for any subset of attributes Y of R.

Inference and Functional Dependencies

- Let F denote some set of functional dependencies that hold on some schema.
- Some FDs are obvious and can be specified by the DB designer.
- Often other dependencies can be inferred from these dependencies.
- For example if A-->B and B-->C then we can infer A--> C.
- **Definition:** The closure of F, denoted F^+, consists of all the FDs in F together with all FDs which can be inferred from F.

Rules for Inferring FDs

IR1 (reflexive rule): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (augmentation rule): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

IR4 (decomposition rule): $\{X \rightarrow YZ\} \models X \rightarrow Y$

IR5 (union rule): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

IR1-3 imply IR4-6.

IR1-3 are called Armstrong's inference rules.

They can be shown to be complete in that any inferable FD in F^+ can be derived from F and these rules.