Databases

Lecture 6

Functional Dependencies, Normal Forms

Relational Algebra

Functional Dependencies, Normal Forms

- a dependency (simple, multi-valued) in a relation can be eliminated via decompositions (the original relation is decomposed into a collection of new relations)
- nevertheless, there are relations without such dependencies that can still contain redundant information, which can be a source of errors in the database

->

Example 13. Consider the relation FaPrCo [FacultyMember, Program, Course], storing the programs and courses for different faculty members; this relation has no functional dependencies; its key is {FacultyMember, Program, Course}

- consider the following data in the relation:

Fa	Pr	Co
F1	P1	C2
F1	P2	C1
F2	P1	C1
F1	P1	C1

- some associations appear in multiple records (redundant data):
  - faculty member F1 is teaching in program P1
  - faculty member F1 is teaching course C1
  - course C1 is taught in program P1

Fa	Pr	Co
F1	P1	C2
F1	P2	C1
F2	P1	C1
F1	P1	C1

- if some values in the relation are changed, e.g., "F1 will teach course C3 instead of course C1", several updates should be carried out, without knowing in how many records
- the same is true for the following changes: "in P1, course C3 should replace course C1", "F1 is switching from program P1 to P3"

- the previous relation cannot be decomposed into 2 relations (via projection), because new data would be introduced through the join
- this claim can be justified by considering the three possible projections on two attributes:

FaPr	Fa	Pr
	F1	P1
	F1	P2
	F2	P1

FaCo	Fa	Co
	F1	C2
	F1	C1
	F2	C1

PrCo	Pr	Co
	P1	C2
	P2	C1
	P1	C1

- when evaluating $\mathrm{FaPr} * \mathrm{PrCo}$, the following data is obtained:

$\mathbf{R' = FaPr * PrCo}$	**Fa**	**Pr**	**Co**
	F1	P1	C2
	F1	P1	C1
	F1	P2	C1
	F2	P1	C2
	F2	P1	C1

- this result set contains an extra tuple, which didn't exist in the original relation
- this is also true for the other join combinations: $\mathrm{FaPr} * \mathrm{FaCo}$ and $\mathrm{PrCo} * \mathrm{FaCo}$

- when evaluating $\mathrm{R}' * \mathrm{FaCo}$ (i.e., $\mathrm{FaPr} * \mathrm{PrCo} * \mathrm{FaCo}$), the original relation FaPrCo is obtained
- conclusion: FaPrCo cannot be decomposed into 2 projections, but it can be decomposed into 3 projections, i.e., FaPrCo is *3-decomposable*:

$$\mathrm{FaPrCo} = \mathrm{FaPr} * \mathrm{PrCo} * \mathrm{FaCo}, \text{ or } \mathrm{FaPrCo} = *\left(\mathrm{FaPr}, \mathrm{PrCo}, \mathrm{FaCo}\right)$$

- this conclusion (FaPrCo is 3-decomposable) is true for the data in the relation
- the 3-decomposability can be specified as a restriction that must be met by all the legal instances of the relation:

* *if (F1, P1) ∈ FaPr and (F1, C1) ∈ FaCo and (P1, C1) ∈ PrCo then (F1, P1, C1) ∈ FaPrCo*

- this restriction can be expressed on FaPrCo:

* *if (F1, P1, C2) ∈ $FaPrCo$ and (F1, P2, C1) ∈ $FaPrCo$ and (F2, P1, C1) ∈ $FaPrCo$ then (F1, P1, C1) ∈ $FaPrCo$*

- consider the following data in the relation

Fa	Pr	Co
F1	P1	C2
F1	P2	C1

- if the previous restriction is specified, then, if (F2, P1, C1) is added to the relation, (F1, P1, C1) must be also added:

Fa	Pr	Co
F1	P1	C2
F1	P2	C1
F2	**P1**	**C1**
F1	**P1**	**C1**

- if (F1, P1, C1) is removed from the relation, other data must be removed as well, at least (F2, P1, C1), for the restriction to be satisfied

Definition. Let R[A] be a relation and $R_i[\alpha_i]$, i=1,…,m, the projections of R on α_i. R satisfies the join dependency $* \{\alpha_1, \ldots, \alpha_m\}$ if R = $R_1 * \ldots * R_m$.

- FaPrCo has a join dependency because $\mathrm{FaPrCo} = \mathrm{FaPr} * \mathrm{PrCo} * \mathrm{FaCo}$

- JD $* \{\alpha_1, \ldots, \alpha_m\}$ is trivial if at least one of α_i is the set of all attributes in R.
- JD $* \{\alpha_1, \ldots, \alpha_m\}$ is implied by the candidate keys if each α_i is a superkey in R.

Definition. Relation R is in 5NF if every non-trivial JD is implied by the candidate keys in R.

- R[A] - a relation
- F - a set of functional dependencies
- α – a subset of attributes

- problems

I. compute the closure of F: F^+

II. compute the closure of a set of attributes under a set of functional dependencies, e.g., the closure of α under F: α^+

III. compute the minimal cover for a set of dependencies

- R[A] - a relation
- F - a set of functional dependencies
- problems

I. compute the closure of F: F^+

- the set F^+ contains all the functional dependencies implied by F
- F implies a functional dependency *f* if *f* holds on every relation that satisfies F
- the following 3 rules can be repeatedly applied to compute F^+ (Armstrong's axioms):
    - $\alpha, \beta, \gamma \subset A$

    1. reflexivity: if $\beta \subseteq \alpha$, then $\alpha \to \beta$
    2. augmentation: if $\alpha \to \beta$, then $\alpha\gamma \to \beta\gamma$
    3. transitivity: if $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$

- these rules are complete (they compute the closure) and sound (no erroneous functional dependencies can be derived)

- R[A] - a relation
- F - a set of functional dependencies
- problems

I. compute the closure of F: F^+

- the following rules can be derived from Armstrong's axioms:

4. union: if $\alpha \to \beta$ and $\alpha \to \gamma$, then $\alpha \to \beta\gamma$

$$\left.\begin{array}{l} \alpha \to \beta \underset{\text{augmentation}}{=>} \alpha\alpha \to \alpha\beta \\ \\ \alpha \to \gamma \underset{\text{augmentation}}{=>} \alpha\beta \to \beta\gamma \end{array}\right\} \underset{\text{transitivity}}{=>} \alpha \to \beta\gamma$$

5. decomposition: if $\alpha \to \beta\gamma$, then $\alpha \to \beta$ and $\alpha \to \gamma$

$$\left.\begin{array}{l} \alpha \to \beta\gamma \\ \beta\gamma \to \beta \text{ (reflexivity)} \end{array}\right\} \underset{\text{transitivity}}{=>} \alpha \to \beta \text{ (}\alpha \to \gamma \text{ can similarly be shown to hold)}$$

- R[A] - a relation
- F - a set of functional dependencies
- problems

I. compute the closure of F: F^+

- the following rules can be derived from Armstrong's axioms:

6. pseudo transitivity: if $\alpha \rightarrow \beta$ and $\beta\gamma \rightarrow \delta$, then $\alpha\gamma \rightarrow \delta$

$$\left.\begin{array}{l} \alpha \rightarrow \beta \text{ => } \alpha\gamma \rightarrow \beta\gamma \\ \beta\gamma \rightarrow \delta \end{array}\right\} \underset{\text{transitivity}}{\text{=>}} \quad \alpha\gamma \rightarrow \delta$$

- $\alpha, \beta, \gamma, \delta \subset A$

- R[A] - a relation
- F - a set of functional dependencies
- α – a subset of attributes
- problems

II. compute the closure of a set of attributes under a set of functional dependencies

- determine the closure of α under F, denoted by α^+
- α^+ - the set of attributes that are functionally dependent on attributes in α (under F)

- R[A] - a relation
- F - a set of functional dependencies
- α – a subset of attributes
- problems

II. compute the closure of a set of attributes under a set of functional dependencies

- algorithm:

```
closure := α;
repeat until there is no change
   for every functional dependency β → γ in F
      if β ⊆ closure
         then closure := closure ∪ γ;
```

- R[A] - a relation
- F - a set of functional dependencies
- problems

III. compute the minimal cover for a set of dependencies

Definition: F, G - two sets of functional dependencies; F and G are equivalent (notation $F \equiv G$) if $F^+ = G^+$.

->

- R[A] - a relation
- F - a set of functional dependencies
- problems

III. compute the minimal cover for a set of dependencies

Definition: F - set of functional dependencies; a minimal cover for F is a set F_M of functional dependencies such that:

1. $F_M \equiv F$
2. the right side of every dependency in F_M has a single attribute;
3. the left side of every dependency in F_M is irreducible (i.e., no attribute can be removed from the determinant of a dependency in F_M without changing F_M's closure);
4. no dependency *f* in F_M is redundant (no dependency can be discarded without changing F_M's closure).

- see lecture problems
  - R - a relation, F - a set of functional dependencies, f - a functional dependency
    - show that f is in F^+
  - R - a relation, F - a set of functional dependencies, α - a subset of the attributes of R
    - compute α^+
  - R - a relation, F - a set of functional dependencies
    - compute F_M - minimal cover for F

Relational Algebra

- query languages in the relational model
  - relational algebra and calculus - formal query languages with a significant influence on SQL
    - relational algebra
      - queries are specified in an operational manner
    - relational calculus
      - queries describe the desired answer, without specifying how it will be computed (declarative)
  - not expected to be Turing complete
  - not intended for complex calculations
  - provide smooth, efficient access to large datasets
  - allow optimizations

- relational algebra
  - used by DBMSs to represent query execution plans
  - a relational algebra query:
    - is built using a collection of operators
    - describes a step-by-step procedure for computing the result set
    - is evaluated on the input relations' instances
    - produces an instance of the output relation
  - every operation returns a relation, hence operators can be composed; the algebra is closed
  - operating on sets of tuples; extension - duplicates are not eliminated

Conditions

- conditions that can be used in several algebraic operators
- similar to the SELECT filter conditions

1. *attribute_name relational_operator value*

- *value* - attribute name, expression

2. *attribute_name IS [NOT] IN single_column_relation*

- a relation with one column can be considered a set
- the condition tests whether a value belongs to a set

3. *relation {IS [NOT] IN | = | <>} relation*

- the relations in the condition should be union-compatible

Conditions

4. *(condition)*
NOT condition
$condition_1$ *AND* $condition_2$
$condition_1$ *OR* $condition_2$,

where *condition*, $condition_1$, $condition_2$ are conditions of type 1-4.

Operators in the Algebra

- equivalent SELECT statements can be specified for the relational algebra expressions
- *selection*
  - unary operator
  - notation: $\sigma_C(R)$
  - resulting relation:
    - schema: R's schema
    - tuples: records in R that meet condition C
  - equivalent SELECT statement
    - `SELECT * FROM R WHERE C`

- *projection*
    - unary operator
    - notation: $\Pi_\alpha(R)$
    - resulting relation:
        - schema: attributes in α
        - tuples: every record in R is projected on α
    - α can be extended to a set of expressions, specifying the columns of the relation being computed
    - equivalent SELECT statement
        - `SELECT` α `FROM R`

References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html