Information Systems

WS 2005, JKU Linz
Course 4: Functional Dependencies and Normal Forms

Gábor Bodnár

URL: `http://www.risc.uni-linz.ac.at/education/courses/ws2005/is/`

Overview

- Integrity, constraints.
- Functional dependencies.
- Normal forms.

Integrity Constraints

Integrity constraints are used to express conditions the database have to fulfill in order the data it holds to be consistent.

We discuss briefly:

- domain constraints,
- referential integrity,
- assertions,
- triggers,
- functional dependencies.

Domain Constraints

Domain constraints restrict the possible values of attributes.

The valid values are selected by some predicate.

Example

Specifying the attribute `credit` in the `Credits` relation to be of numeric and take values from the range from 0 to 5:

$$\texttt{Credits.credit} \geq 0 \ \wedge \ \texttt{Credits.credit} \leq 5.$$

Remark

They are usually easy to check and thus provided by many DBMSes.

Referential Integrity

Let A and B be relations with describing attribute sets R and S respectively.

Then $P \subseteq R$ is a *foreign key* for a candidate key $L \subseteq S$ of B if for every tuple t in A there is a tuple u in B such that $\Pi_P t = \Pi_L u$ (or in other terms $\Pi_P A \subseteq \Pi_L B$).

Such a condition is a *referential integrity* constraint.

Example

Student

sid	fname	lname
0251563	Werner	Schmidt
0245654	Andrea	Ritter
0245675	Daniela	Schmidt

Attendance

sid	cid
0251563	327456
0251563	327564
0245654	327456

Course

cid	title
327456	Analysis I
327564	Algebra I

Referential Integrity and DB Modifications

Modifications in the database can destroy referential integrity; therefore

- when a tuple t is inserted into A, there must already be a tuple u in B with $\Pi_P t = \Pi_L u$,
- when deleting a tuple u from B and there is still a tuple t in A with $\Pi_P t = \Pi_L u$, then either abort the deletion or delete also t from A (depending on the application),
- when updating tuples in A analogous actions has to be taken as in the case of insertion and when updating B as in the case of deletion (to ensure referential integrity in both cases).

Assertions and Triggers

An *assertions* is a predicate the database should always satisfy.

The previous integrity constraints are special cases of assertion.

A *trigger* is a statement (in the query language) the DBMS executes automatically whenever a set of conditions becomes true.

Examples

Using the university database example, an assertion could be to require that no teacher has more than five courses in a semester.

In the same database, the deletion of a tuple from the `Student` relation could trigger deletion of referring tuples from the `Attendance` relation.

Functional Dependencies

Functional dependencies represent integrity constraints on sets of attributes in the database.

They are in strong correspondence with causal-, temporal-, etc. dependencies in the real world.

Let R be a set of attributes and let P and S be subsets R.

Then S *functionally depends* on P, denoted as $P \rightarrow S$, if for any legal relation A with R being its set of describing attributes, for any tuples t and u,

$$\Pi_P t = \Pi_P u \quad \text{implies} \quad \Pi_S t = \Pi_S u.$$

Functional Dependencies (continued)

A functional dependency is called *trivial* if it is satisfied by all relations.

We say that S *irreducibly (or fully) functionally depends* on P if $P \to S$ and we do not have $Q \to S$ for any proper subset of $Q \subset P$.

We call $P \to S$ *irreducible* if $Q \not\to S$ for every $Q \subset P$.

Remark

The fact that K is a superkey can be expressed as $K \to R$.

The fact that K is a candidate key can be expressed as $K \to R$ and the dependence is irreducible.

Example

$(\texttt{Student} \bowtie \texttt{Attendance}) \bowtie \texttt{Course}$

sid	fname	lname	cid	title
0251563	Werner	Schmidt	327456	Analysis I
0251563	Werner	Schmidt	327564	Algebra I
0245654	Andrea	Ritter	327456	Analysis I

A few functional dependencies in this relation are:

- $\{\texttt{sid}\} \rightarrow \{\texttt{fname}, \texttt{lname}\}$,
- $\{\texttt{cid}\} \rightarrow \{\texttt{title}\}$,
- $\{\texttt{sid}, \texttt{cid}\} \rightarrow \{\texttt{sid}, \texttt{fname}, \texttt{lname}, \texttt{cid}, \texttt{title}\}$.

Exercise

What are the nontrivial irreducible functional dependencies of the following relation scheme describing relationships between borrowers and books.

`mid` library member ID

`name` library member name

`bid` book (instance) ID

`isbn` book ISBN number

`author` name of the author

`title` title of the book.

Armstrong's Axioms

The following axiom system describes functional dependencies.

For nonempty $P, Q, S, T \subseteq R$ we have

- $Q \subseteq P$ implies $P \to Q$ (reflexivity),
- $P \to Q$ implies $P \cup S \to Q \cup S$ (augmentation),
- $P \to Q$ and $Q \to S$ implies $P \to S$ (transitivity).

These axioms are consistent and complete.

Additional Properties of FDs

- $P \rightarrow P$ (self-determination),
- $P \rightarrow Q \cup S$ implies $P \rightarrow Q$ and $P \rightarrow S$ (decomposition),
- $P \rightarrow Q$ and $P \rightarrow S$ implies $P \rightarrow Q \cup S$ (union),
- $P \rightarrow Q$ and $S \rightarrow T$ implies $P \cup S \rightarrow Q \cup T$ (composition),
- $P \rightarrow Q$ and $Q \cup S \rightarrow T$ implies $P \cup S \rightarrow T$ (pseudotransitivity).

If F is a set of functional dependencies the set of all dependencies it implies is called the *closure of* F, denoted with F^+.

The set of attributes of R determined by $P \subseteq R$ under F is called the *closure of* P, denoted as P^+

Equivalence and Irreducibility of sets of FDs

Let F and G be two sets of functional dependencies, if $F^+ \subseteq G^+$ then we say that G *covers* F. If F and G mutually cover each other, they are called *equivalent*.

A set of dependencies is called *irreducible* if

- the right hand side of every functional dependency in F is a singleton,
- no attribute can be discarded from the left hand side of any functional dependency in F without changing F^+,
- no functional dependency in F can be discarded without changing F^+.

Example

Let $F = \{F_1, F_2, F_3\}$. A few functional dependencies in F^+ are:

$\{\texttt{sid}\}$	$\rightarrow$	$\{\texttt{fname,lname}\}$	F_1
$\{\texttt{cid}\}$	$\rightarrow$	$\{\texttt{title}\}$	F_2
$\{\texttt{sid,cid}\}$	$\rightarrow$	$\{\texttt{sid,fname,lname,cid,title}\}$	F_3
$\{\texttt{sid}\}$	$\rightarrow$	$\{\texttt{fname}\}$	// decmp.
$\{\texttt{sid}\}$	$\rightarrow$	$\{\texttt{lname}\}$	// decmp.
$\{\texttt{sid,cid}\}$	$\rightarrow$	$\{\texttt{sid,cid}\}$	// refl.
$\{\texttt{sid,cid}\}$	$\rightarrow$	$\{\texttt{fname,lname,title}\}$	// comp.

Furthermore F cover both $\{F_1\}$ and $\{F_2\}$, and $\{F_1, F_2\}$ is equivalent to F. An equivalent irreducible set of FDs is

$$\{\{\texttt{sid}\} \rightarrow \{\texttt{fname}\}, \quad \{\texttt{sid}\} \rightarrow \{\texttt{lname}\}, \quad \{\texttt{cid}\} \rightarrow \{\texttt{title}\}\}.$$

Normalization

Goals:

- Force better database design.
- Eliminate data redundancy.
- Make data retrieval more efficient.

We decompose the initial relation schemes so that the resulting schemes:

- satisfy certain functional dependency constraints,
- allow the reconstruction of the original relations without data loss.

Lossless Join Decomposition

Let R be a relation scheme; a set of relation schemes $R_1, \ldots, R_n$ is a *decomposition* of R if $R = R_1 \cup \cdots \cup R_n$.

We call $R_1, \ldots, R_n$ a *lossless join decomposition* of R if for all legal relation A on R we have

$$A = \Pi_{R_1}(A) \bowtie \cdots \bowtie \Pi_{R_n}(A).$$

A decomposition $P \cup S = R$, with respect to F, is a lossless join decomposition if $P \cap S \rightarrow P$ or $P \cap S \rightarrow S$ is in F^+ (Heath).

The *restriction* of F to R_i, denoted as F_i, is the set of all functional dependencies of F^+ that contain only attributes in R_i.

If $F^+ = (F_1 \cup \cdots \cup F_n)^+$, we call $R_1, \ldots, R_n$ a *dependency preserving decomposition* of R.

Example

For the relation scheme

$$\{\texttt{sid},\ \texttt{fname},\ \texttt{lname},\ \texttt{cid},\ \texttt{title}\}$$

the decomposition

$$\{\texttt{sid},\ \texttt{fname},\ \texttt{lname}\} \cup \{\texttt{sid},\ \texttt{cid},\ \texttt{title}\}$$

is a lossless join decomposition, where

$$\{\texttt{sid},\ \texttt{cid}\} \cup \{\texttt{cid},\ \texttt{title}\}$$

is a lossless join decomposition for the latter.

This is also a dependency preserving decomposition.

First Normal Form (1NF)

A relation scheme is in *first normal form (1NF)*, if all the attribute values in it are atomic.

Remark

By our conventions about atomicity, every relation is in 1NF.

For defining 2NF and 3NF we assume that the relation scheme has only one candidate key which is the primary key.

An attribute not member of any candidate key is referred to as *non-key attribute*.

Second Normal Form (2NF)

A relation scheme is in *second normal form (2NF)* (with respect to a set of functional dependencies F), if it is 1NF and every non-key attribute irreducibly functionally depends on the primary key.

Example

The relation scheme $\{\texttt{sid}, \texttt{fname}, \texttt{lname}, \texttt{cid}, \texttt{title}\}$ with primary key $\{\texttt{sid}, \texttt{cid}\}$, with respect to the set of functional dependencies implied by

$$\{\{\texttt{sid}\} \to \{\texttt{fname}\}, \quad \{\texttt{sid}\} \to \{\texttt{lname}\}, \quad \{\texttt{cid}\} \to \{\texttt{title}\}\}.$$

is in 1NF but not in 2NF. For instance $\{\texttt{fname}\}$ functionally depends on $\{\texttt{sid}\}$ which is a proper subset of the primary key.

Third Normal Form (3NF)

A functional dependence $P \rightarrow S$ in F^+ is called *transitive* (via Q) if there exist functional dependencies $P \rightarrow Q$ and $Q \rightarrow S$ with $S \not\rightarrow Q$ and $Q \not\rightarrow P$.

A relation scheme is in *third normal form (3NF)* (with respect to a set of functional dependencies F), if it is 2NF and every non-key attribute non-transitively depends on the primary key.

Example

Let us consider then the relation scheme

$$\{\texttt{sid}, \texttt{pts1}, \texttt{pts2}, \texttt{grade}\}.$$

The primary key is $\{\texttt{sid}\}$. This scheme is in 2NF, but not in 3NF because of $\{\texttt{pts1}, \texttt{pts2}\} \rightarrow \{\texttt{grade}\}$.

Boyce-Codd Normal Form (BCNF)

A relation scheme is in *Boyce-Codd normal form (BCNF)* (with respect to a set of functional dependencies F), if the left hand side of every nontrivial irreducible functional dependency is a candidate key.

Example

Relation scheme: {sid, cid, title, grade}.

Let the candidate keys be {sid,cid}, {sid, title}.

{sid,cid} → {title}
{sid,cid} → {grade}
{sid,title} → {cid}
{sid,title} → {grade}
{cid} → {title}
{title} → {cid}.

Remark

There are also higher normal forms denoted by 4NF and 5NF (not treated in this lecture).

The goal of database design is to reach BCNF, or if it seems to be too much effort for too little gain, to reach 3NF.

Too much normalization can also decrease system performance, therefore sometimes *denormalization* is applied after reaching the higher normal forms.

Summary

- Integrity constraints: domain constraints, referential integrity, assertions, triggers.
- Functional dependencies (FDs), irreducibility.
- Armstrong's axioms, closure of a set of FDs.
- Equivalence sets of FDs, irreducible sets of FDs.
- Normalization, lossless join decompositions.
- 1NF, 2NF, 3NF, BCNF.