CS 405G: Introduction to Database Systems

Database Normalization

Database Normalization

- Database normalization relates to the level of redundancy in a relational database's structure.
- The key idea is to reduce the chance of having multiple different version of the same data.
- Well-normalized databases have a schema that reflects the true dependencies between tracked quantities.
- Any increase in normalization generally involves splitting existing tables into multiple ones, which must be re-joined each time a query is issued.

Normalization

- A *normalization* is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency.

- A *normal form* is a certification that tells whether a relation schema is in a particular state

Normal Forms

- Edgar F. Codd originally established three normal forms: 1NF, 2NF and 3NF.
- 3NF is widely considered to be sufficient.
- Normalizing beyond 3NF can be tricky with current SQL technology as of 2005
- Full normalization is considered a good exercise to help discover all potential internal database consistency problems.

First Normal Form (1NF)

http://en.wikipedia.org/wiki/First_normal_form

"What is your favorite color?"
"What food will you not eat?"

TABLE 1

Person	Favorite Color
Bob	blue
Jane	green

TABLE 2

Person	Foods Not Eaten
Bob	okra
Bob	brussel sprouts
Jane	peas

More examples.

- http://en.wikipedia.org/wiki/First_normal_form

2^{nd} Normal Form

- An attribute *A* of a relation *R* is a *nonprimary attribute* if it is not part of any key in *R,* otherwise*, A* is a *primary attribute.*
- *R* is in (general) 2^{nd} normal form if every nonprimary attribute *A* in *R* is not partially functionally dependent on *any* key of *R*

X	Y	Z	W
a	b	c	e
b	b	c	f
c	b	c	g

X , Y -> Z, W $\Rightarrow$ (X , Y , W)

Y -> Z $\quad$ (Y, Z)

2nd Normal Form

- Note about 2nd Normal Form
  - by definition, every nonprimary attribute is functionally dependent on every key of *R*
  - In other words, *R* is in its 2nd normal form if we could not find a partial dependency of a nonprimary key to a key in *R*.

  - 2NF prescribes full functional dependency on the primary key.
  - It most commonly applies to tables that have composite primary keys, where two or more attributes comprise the primary key.

2NF Example

PART_NUMBER (PRIMARY KEY)
SUPPLIER_NAME (PRIMARY KEY)
PRICE
SUPPLIER_ADDRESS

- The PART_NUMBER and SUPPLIER_NAME form the composite primary key.
- SUPPLIER_ADDRESS is only dependent on the SUPPLIER_NAME, and therefore this table breaks 2NF.

Decomposition

EID	PID	Ename	email	Pname	Hours
1234	10	John Smith	jsmith@ac.com	B2B platform	10
1123	9	Ben Liu	bliu@ac.com	CRM	40
1234	9	John Smith	jsmith@ac.com	CRM	30
1023	10	Susan Sidhuk	ssidhuk@ac.com	B2B platform	40

Decomposition

Foreign key

EID	Ename	email
1234	John Smith	jsmith@ac.com
1123	Ben Liu	bliu@ac.com
1023	Susan Sidhuk	ssidhuk@ac.com

EID	PID	Pname	Hours
1234	10	B2B platform	10
1123	9	CRM	40
1234	9	CRM	30
1023	10	B2B platform	40

- Decomposition eliminates redundancy
- To get back to the original relation: ∗

Decomposition

- Decomposition may be applied recursively

EID	PID	Pname	Hours
1234	10	B2B platform	10
1123	9	CRM	40
1234	9	CRM	30
1023	10	B2B platform	40

PID	Pname
10	B2B platform
9	CRM

EID	PID	Hours
1234	10	10
1123	9	40
1234	9	30
1023	10	40

Unnecessary decomposition

EID	Ename	email
1234	John Smith	jsmith@ac.com
1123	Ben Liu	bliu@ac.com
1023	Susan Sidhuk	ssidhuk@ac.com

EID	Ename
1234	John Smith
1123	Ben Liu
1023	Susan Sidhuk

EID	email
1234	jsmith@ac.com
1123	bliu@ac.com
1023	ssidhuk@ac.com

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed, and now *EID* is stored twice->

Bad decomposition

EID	PID	Hours
1234	10	10
1123	9	40
1234	9	30
1023	10	40

EID	PID
1234	10
1123	9
1234	9
1023	10

EID	Hours
1234	10
1123	40
1234	30
1023	40

- Association between *PID* and *hours* is lost
- Join returns more rows than the original relation

Lossless join decomposition

- Decompose relation R into relations S and T
  - $attrs(R) = attrs(S) \cup attrs(T)$
  - $S = \pi_{attrs(S)} (R)$
  - $T = \pi_{attrs(T)} (R)$
- The decomposition is a lossless join decomposition if, given known *constraints* such as FD's, we can guarantee that $R = S * T$
- Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  - A *lossy* decomposition is one with $R \subset S \bowtie T$

Loss? But I got more rows->

- "Loss" refers not to the loss of tuples, but to the loss of information
  - Or, the ability to distinguish different original tuples

EID	PID	Hours
1234	10	10
1123	9	40
1234	9	30
1023	10	40

EID	PID
1234	10
1123	9
1234	9
1023	10

EID	Hours
1234	10
1123	40
1234	30
1023	40

Questions about decomposition

- When to decompose
- How to come up with a correct decomposition (i.e., lossless join decomposition)

Third normal form

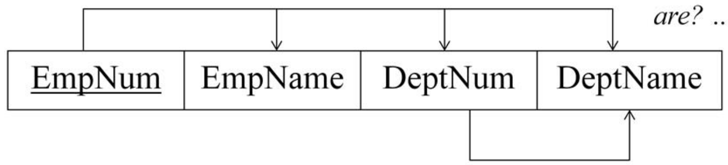
- **3NF** requires that there are no non-trivial functional dependencies of non-key attributes on something other than a superset of a candidate key.

- All non-key attributes are mutually independent.

3NF - Example

Consider this **Employee** relation

Candidate keys are? ...

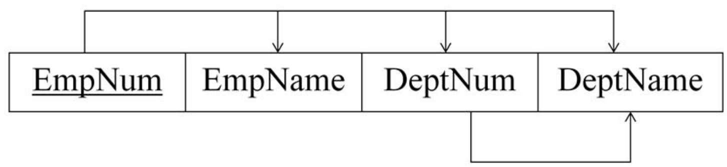


EmpName, DeptNum, and DeptName are non-key attributes.

DeptNum determines DeptName, a non-key attribute, and DeptNum is not a candidate key.

Is the relation in 3NF? … no

Is the relation in 2NF? … yes



We correct the situation by decomposing the original relation into two 3NF relations. Note the decomposition is *lossless*.

EmpNum	EmpName	DeptNum

DeptNum	DeptName

Verify these two relations are in 3NF.

Boyce-Codd normal form (BCNF)

- **BCNF** requires that there are no non-trivial functional dependencies of attributes on something other than a superset of a candidate key (called a superkey).
- All attributes are dependent on a key, a whole key and nothing but a key (excluding trivial dependencies, like A->A).

• A table is said to be in the BCNF if and only if it is in the 3NF and every non-trivial, left-irreducible functional dependency has a candidate key as its determinant.

• In more informal terms, a table is in BCNF if it is in 3NF and the only determinants are the candidate keys.

Non-key FD's

- Consider a non-trivial FD X -> Y where X is not a super key
  - Since X is not a super key, there are some attributes (say Z) that are not functionally determined by X

X	Y	Z
a	b	c
a	b	d

That b is always associated with a is recorded by multiple rows: redundancy, update anomaly, deletion anomaly

Dealing with Nonkey Dependency: BCNF

- A relation *R* is in Boyce-Codd Normal Form if
  - For every non-trivial FD *X* -> *Y* in *R*, *X* is a super key
  - That is, all FDs follow from “key -> other attributes”
- When to decompose
  - As long as some relation is not in BCNF
- How to come up with a correct decomposition
  - Always decompose on a BCNF violation (details next)
  - ☞ Then it is guaranteed to be a lossless join decomposition

BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD X -> Y in R where X is not a super key of R
- Decompose R into R_1 and R_2, where
  - R_1 has attributes $X \cup Y$
  - R_2 has attributes $X \cup Z$, where Z contains all attributes of R that are in neither X nor Y (i.e. $Z = attr(R) - X - Y$)
- Repeat until all relations are in BCNF

BCNF decomposition example

WorkOn (*EID*, *Ename*, *email*, *PID*, *hours*)

BCNF violation: *EID* -> *Ename*, *email*

Student (*EID*, *Ename*, *email*)

BCNF

Grade (*EID*, *PID*, *hours*)

BCNF

Another example

WorkOn (*EID*, *Ename*, *email*, *PID*, *hours*)

BCNF violation: *email* -> *EID*

StudentID (*email*, *EID*)

BCNF

StudentGrade' (*email*, *Ename*, *PID*, *hours*)

BCNF violation: *email* -> *Ename*

StudentName (*email*, *Ename*)

BCNF

Grade (*email*, *PID*, *hours*)

BCNF

Exercise

- *Property(Property_id#, County_name, Lot#, Area, Price, Tax_rate*)
  - Property_id#-> *County_name, Lot#, Area, Price, Tax_rate*
  - *County_name, Lot#* -> Property_id#*, Area, Price, Tax_rate*
  - *County_name* -> *Tax_rate*
  - *area* -> *Price*

Exercise

Property(Property_id#, County_name, Lot#, Area, Price, Tax_rate)

BCNF violation: County_name -> *Tax_rate*

LOTS1 (*County_name, Tax_rate*)

BCNF

LOTS2 (*Property_id#, County_name, Lot#, Area, Price*)

BCNF violation: Area -> *Price*

LOTS2A (*Area, Price*)

BCNF

LOTS2B (*Property_id#, County_name, Lot#, Area*)

BCNF

Why is BCNF decomposition lossless

Given non-trivial $X \rightarrow Y$ in R where X is not a super key of R, need to prove:

- Anything we project always comes back in the join:
  $R \subseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$
  - Sure; and it doesn't depend on the FD
- Anything that comes back in the join must be in the original relation:
  $R \supseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$
  - Proof makes use of the fact that $X \rightarrow Y$

Recap

- Functional dependencies: a generalization of the key concept
- Partial dependencies: a source of redundancy
  - Use 2nd Normal form to remove partial dependency
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method for removing ALL functional dependency related redundancies
  - Plus, BNCF decomposition is a lossless join decomposition