

Databases Illuminated

Chapter 5

Normalization

Objectives of Normalization

- Develop a good description of the data, its relationships and constraints
- Produce a stable set of relations that
 - Is a faithful model of the enterprise
 - Is highly flexible
 - Reduces redundancy-saves space and reduces inconsistency in data
 - Is free of **update, insertion and deletion anomalies**

Anomalies

- An anomaly is an inconsistent, incomplete, or contradictory state of the database
 - **Insertion** anomaly – user is unable to insert a new record when it should be possible to do so
 - **Deletion** anomaly – when a record is deleted, other information that is tied to it is also deleted
 - **Update** anomaly – a record is updated, but other appearances of the same items are not updated

Anomaly Examples: NewClass Table

See **Figure 5.1**

NewClass(courseNo, stuld, stuLastName, fID, schedule, room, grade)

Update anomaly: If schedule of ART103A is updated in first record, and not in second and third – inconsistent data

Deletion anomaly: If record of student S1001 is deleted, information about HST205A class is lost also

Insertion anomaly: It is not possible to add a new class, for MTH101A , even if its teacher, schedule, and room are known, unless there is a student registered for it, because the key contains stuld

Normal Forms

- First normal form -1NF
- Second normal form-2NF
- Third normal form-3NF
- Boyce-Codd normal form-BCNF
- Fourth normal form-4NF
- Fifth normal form-5NF
- Domain/Key normal form-DKNF

Each is contained within the previous form – each has stricter rules than the previous form

Functional Dependency

- A **functional dependency** (FD) is a type of relationship between attributes
- If A and B are sets of attributes of relation R, say B is functionally dependent on A if each A value in R has associated with it exactly one value of B in R.
- Alternatively, if two tuples have the same A values, they must also have the same B values
- Write **A ? B**, read **A functionally determines B**, or B functionally dependent on A
- FD is actually a many-to-one relationship between A and B

Example of FDs

- Let R be

NewStudent(stuld, lastName, major, credits, status, socSecNo)

- FDs in R include

{stuld}? {lastName}, but not the reverse

{stuld} ? {lastName, major, credits, status, socSecNo, stuld}

{socSecNo} ? {stuld, lastName, major, credits, status, socSecNo}

{credits}? {status}, but not {status}? {credits}

Trivial Functional Dependency

- The FD $X \twoheadrightarrow Y$ is **trivial** if set $\{Y\}$ is a subset of set $\{X\}$

Examples: If A and B are attributes of R,

$\{A\} \twoheadrightarrow \{A\}$

$\{A,B\} \twoheadrightarrow \{A\}$

$\{A,B\} \twoheadrightarrow \{B\}$

$\{A,B\} \twoheadrightarrow \{A,B\}$

are all trivial FDs

Keys

- **Superkey** – functionally determines all attributes in a relation
- **Candidate key** - superkey that is a minimal identifier (no extraneous attributes)
- **Primary key** - candidate key actually used
- Primary key has **no-null** constraint and **uniqueness** constraint
- Should also enforce uniqueness and no-null rule for candidate keys

First Normal Form

- A relation is in **1NF** if every attribute is single-valued for each tuple
- Each cell of the table has only one value in it
- Domains of attributes are **atomic**: no sets, lists, repeating fields or groups allowed in domains

Counter-Example for 1NF

See Figure 5.4(a)

NewStu(Stuld, lastName, major, credits, status, socSecNo) – Assume students can have more than one major

The *major* attribute is not single-valued for each tuple

Ensuring 1NF

- Best solution: For each multi-valued attribute, create a new table, in which you place the key of the original table and the multi-valued attribute. Keep the original table, with its key

Ex. NewStu2(stuld, lastName, credits, status, socSecNo)

Majors(stuld, major)

See **Figure 5.4(b)**

Another method for 1NF

- “Flatten” the original table by making the multi-valued attribute part of the key

Student(stuld, lastName, major, credits, status, socSecNo)

See **Figure 5.4(d)**

Yet Another Method

- If the number of repeats is limited, make additional columns for multiple values

Student(stuld, lastName, major1, major2, credits, status, socSecNo)

See **Figure 5.4(c)**

Full Functional Dependency

- In relation R , set of attributes B is **fully functionally dependent** on set of attributes A of R if B is functionally dependent on A but not functionally dependent on any proper subset of A
- This means every attribute in A is needed to functionally determine B

Partial Functional Dependency Example

NewClass(courseNo, stuld, stuLastName, facId, schedule, room, grade)

FDs:

{courseNo, stuld} ? {lastName}

{courseNo, stuld} ? {facId}

{courseNo, stuld} ? {schedule}

{courseNo, stuld} ? {room}

{courseNo, stuld} ? {grade}

courseNo ? facId **partial FD

courseNo ? schedule **partial FD

courseNo ? room ** partial FD

stuld ? lastName ** partial FD

...plus trivial FDs that are partial...

Second Normal Form

- A relation is in **second normal form** (2NF) if it is in first normal form and all the non-key attributes are **fully** functionally dependent on the key.
- No non-key attribute is FD on just part of the key
- If key has only one attribute, and R is 1NF, R is automatically 2NF

Converting to 2NF

- Identify each partial FD
- Remove the attributes that depend on each of the determinants so identified
- Place these determinants in separate relations along with their dependent attributes
- In original relation keep the composite key and any attributes that are fully functionally dependent on all of it
- Even if the composite key has no dependent attributes, keep that relation to connect logically the others

2NF Example

NewClass(courseNo, stuld, stuLastName, facId, schedule, room, grade)

FDs grouped by determinant:

{courseNo} ? {courseNo, facId, schedule, room}

{stuld} ? {stuld, lastName}

{courseNo, stuld} ? {courseNo, stuld, facId, schedule,
room, lastName, grade}

Create tables grouped by determinants:

Course(courseNo, facId, schedule, room)

Stu(stuld, lastName)

Keep relation with original composite key, with attributes FD on it, if any

NewStu2(courseNo, stuld, grade)

Transitive Dependency

- If A, B, and C are attributes of relation R, such that $A \twoheadrightarrow B$, and $B \twoheadrightarrow C$, then C is **transitively dependent** on A

Example:

NewStudent (stuld, lastName, major, credits, status)

FD:

credits? status

By transitivity:

stuld? credits ? credits? status implies stuld? status

Transitive dependencies cause update, insertion, deletion anomalies.

Third Normal Form

- A relation is in **third normal form (3NF)** if whenever a non-trivial functional dependency $X \twoheadrightarrow A$ exists, then either X is a superkey or A is a member of some candidate key
- To be 3NF, relation must be 2NF and have no transitive dependencies
- No non-key attribute determines another non-key attribute. Here key includes “candidate key”

Making a relation 3NF

- For example,
NewStudent (stuld, lastName, major, credits, status)
with FD credits? status
- Remove the dependent attribute, *status*, from the relation
- Create a new table with the dependent attribute and its determinant, *credits*
- Keep the determinant in the original table

NewStu2 (stuld, lastName, major, credits)

Stats (credits, status)

Boyce-Codd Normal Form

- A relation is in Boyce/Codd Normal Form (BCNF) if whenever a non-trivial functional dependency $X \twoheadrightarrow A$ exists, then X is a superkey
- Stricter than 3NF, which allows A to be part of a candidate key
- If there is just one single candidate key, the forms are equivalent

Example

NewFac (facName, dept, office, rank, dateHired)

FDs:

office ? dept

facName,dept ? office, rank, dateHired

facName,office ? dept, rank, dateHired

- NewFac is not BCNF because office is not a superkey
- To make it BCNF, remove the dependent attributes to a new relation, with the determinant as the key
- Project into

Fac1 (office, dept)

Fac2 (facName, office, rank, dateHired)

Note we have lost a functional dependency in Fac2 – no longer able to see that {facName, dept} is a determinant, since they are in different relations

Properties of Decompositions

- Starting with a universal relation that contains all the attributes, we can decompose into relations by projection
- A **decomposition** of a relation R is a set of relations $\{R_1, R_2, \dots, R_n\}$ such that each R_i is a subset of R and the union of all of the R_i is R .
- Desirable properties of decompositions
 - **Attribute preservation** - every attribute is in some relation
 - **Dependency preservation** - see previous example
 - **Lossless decomposition** - discussed later

Dependency Preservation

- If R is decomposed into $\{R_1, R_2, \dots, R_n\}$ so that for each functional dependency $X \twoheadrightarrow Y$ all the attributes in $X \twoheadrightarrow Y$ appear in the same relation, R_i , then all FDs are preserved
- Allows DBMS to check each FD constraint by checking just one table for each

Multi-valued Dependency

- In $R(A,B,C)$ if each A values has associated with it a set of B values and a set of C values such that the B and C values are independent of each other, then **A multi-determines B** and **A multi-determines C**
- Multi-valued dependencies occur in pairs
- Example: JointAppoint(facId, dept, committee)
assuming a faculty member can belong to more than one department and belong to more than one committee
- Table must list all combinations of values of department and committee for each facId

4NF

- A table is **4NF** if it is BCNF and has **no multi-valued dependencies**
- Example: remove MVDs in JointAppoint
Appoint1 (facId,dept)
Appoint2(facId,committee)

Example of Lossy Projection

Original EmpRoleProj table:

<u>EmpName</u>	<u>role</u>	<u>projName</u>
Smith	designer	Nile
Smith	programmer	Amazon
Smith	designer	Amazon
Jones	designer	Amazon

Project into

Table a

<u>EmpName</u>	<u>role</u>
Smith	designer
Smith	programmer
Jones	designer

Table b

<u>role</u>	<u>projName</u>
designer	Nile
programmer	Amazon
designer	Amazon

Joining Table a and Table b produces

<u>EmpName</u>	<u>role</u>	<u>projName</u>
Smith	designer	Nile
Smith	designer	Amazon
Smith	programmer	Amazon
Jones	designer	Nile
Jones	designer	Amazon

 **spurious tuple**

Lossless Decomposition

- A decomposition of R into $\{R_1, R_2, \dots, R_n\}$ is **lossless** if the natural join of R_1, R_2, \dots, R_n produces exactly the relation R
- No **spurious tuples** are created when the projections are joined.
- always possible to find a BCNF decomposition that is lossless

Lossless Projections

- Lossless property guaranteed if for each pair of relations that will be joined, the set of common attributes is a superkey of one of the relations
- Binary decomposition of R into $\{R_1, R_2\}$ lossless iff one of these holds

$$R_1 \cap R_2 \supseteq R_1 - R_2$$

or

$$R_1 \cap R_2 \supseteq R_2 - R_1$$

- For n relations in decomposition, test by general *Algorithm to Test for Lossless Join*, Section 5.6.3
- If projection is done by successive binary projections, can apply binary decomposition test repeatedly

5NF and DKNF

- A relation is **5NF** if there are no remaining non-trivial lossless projections
- A relation is in **Domain-Key Normal Form (DKNF)** if every constraint is a logical consequence of domain constraints or key constraints

De-normalization

- When to stop the normalization process
 - When applications require too many joins
 - When you cannot get a non-loss decomposition that preserves dependencies

Inference Rules for FDs

- Armstrong's Axioms

- **Reflexivity** If B is a subset of A , then $A \twoheadrightarrow B$.
- **Augmentation** If $A \twoheadrightarrow B$, then $AC \twoheadrightarrow BC$.
- **Transitivity** If $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$, then $A \twoheadrightarrow C$

Additional rules that follow:

- **Additivity** If $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$, then $A \twoheadrightarrow BC$
- **Projectivity** If $A \twoheadrightarrow BC$, then $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$
- **Pseudotransitivity** If $A \twoheadrightarrow B$ and $CB \twoheadrightarrow D$, then $AC \twoheadrightarrow D$

Closure of Set of FDs

- If F is a set of functional dependencies for a relation R , then the set of all functional dependencies that can be derived from F , F^+ , is called the **closure of F**
- Could compute closure by applying Armstrong's Axioms repeatedly

Closure of an Attribute

- If A is an attribute or set of attributes of relation R , all the attributes in R that are functionally dependent on A in R form the **closure of A , A^+**
- Computed by Closure Algorithm for A , Section 5.7.3

result ? A;

while (result changes) do

for each functional dependency $B \rightarrow C$ in F

if B is contained in result then result ? result ? C;

end;

$A^+ ?$ result;

Uses of Attribute Closure

- Can determine if A is a superkey-if every attribute in R functionally dependent on A
- Can determine whether a given FD $X \rightarrow Y$ is in the closure of the set of FDs. (Find X^+ , see if it includes Y)

Redundant FDs and Covers

- Given a set of FDs, can determine if any of them is **redundant**, i.e. can be derived from the remaining FDs, by a simple algorithm – see Section 5.7.4
- If a relation R has two sets of FDs, F and G
 - then F is a **cover** for G if every FD in G is also in F^+
 - F and G are equivalent if F is a cover for G and G is a cover for F (i.e. $F^+ = G^+$)

Minimal Set of FDs

- Set of FDs, F is **minimal** if
 - The right side of every FD in F has a single attribute (called standard or canonical form)
 - No attribute in the left side of any FD is extraneous
 - F has no redundant FDs

Minimal Cover for Set of FDs

- A minimal cover for a set of FDs is a cover such that no proper subset of itself is also a cover
- A set of FDs may have several minimal covers
- See *Algorithm for Finding a Minimal Cover*, Section 5.7.7

Decomposition Algorithm for BCNF

- Can always find a lossless decomposition that is BCNF
 - Find a FD that is a violation of BCNF and remove it by decomposition process
 - Repeat this process until all violations are removed
 - See algorithm, Section 5.7.8
- No need to go through 1NF, 2NF, 3NF process
- Not always possible to preserve all FDs

Synthesis Algorithm for 3NF

- Can always find 3NF decomposition that is lossless **and that preserves all FDs**
- 3NF Algorithm uses synthesis
 - Begin with universal relation and set of FDs, G
 - Find a minimal cover for G
 - Combine FDs that have the same determinant
 - Include a relation with a key of R
 - See algorithm, Section 5.7.9