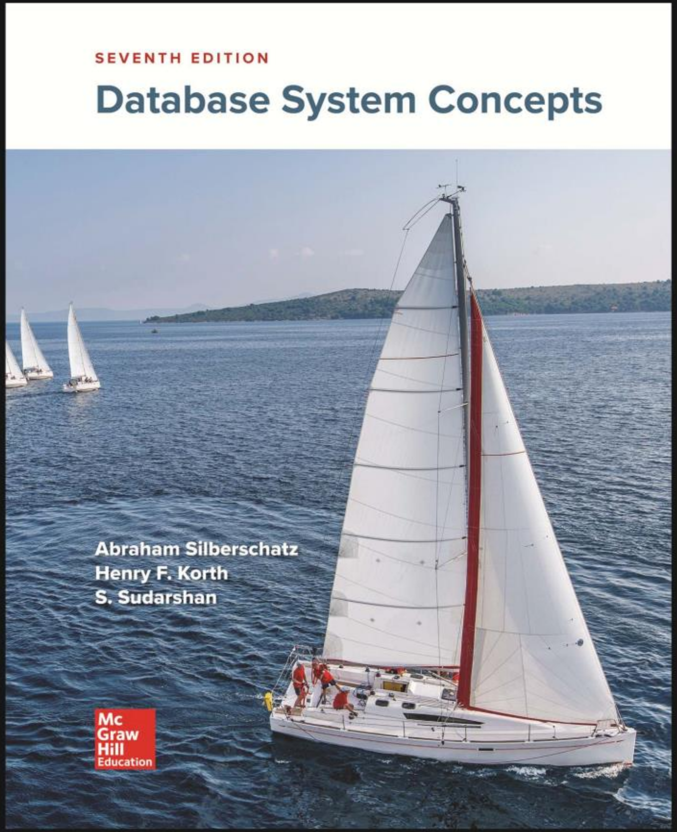
IF2240 – Basis Data

Relational Database Design



Sumber

- Silberschatz, Korth, Sudarshan: "Database System Concepts", 7th Edition
  - Chapter 7: Relational Database Design

Capaian

- Mahasiswa dapat menghasilkan desain basis data relasional yang baik berdasarkan prinsip-prinsip yang diberikan

Outline

- Features of Good Relational Design
- Functional Dependencies
- Decomposition Using Functional Dependencies
- Normal Forms
- Functional Dependency Theory
- Algorithms for Decomposition using Functional Dependencies
- Database-Design Process

Algorithm for Decomposition Using Functional Dependencies

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF: verify that α is a superkey
- **Simplified test**: To check if *R* is in BCNF, it suffices to check only the dependencies in *F*.
- However, **simplified test** using only F is incorrect when testing a relation in a decomposition of R

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF: verify that α is a superkey
- **Simplified test**: To check if R is in BCNF, it suffices to check only the dependencies in F.
- However, **simplified test** using only F is incorrect when testing a relation in a decomposition of R

- Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$
- Decompose R into $R_1 = (A,B)$ and $R_2 = (A,C, D, E)$
- Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be misled into thinking R_2 satisfies BCNF.
- In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

To check if a relation R_i in a decomposition of R is in BCNF

Either

test R_i for BCNF with respect to the **restriction** of F^+ to R_i (that is, all FDs in F^+ that contain only attributes from R_i)

use the original set of dependencies *F* that hold on *R*, but with additional test

Testing Decomposition for BCNF

To check if a relation R_i in a decomposition of R is in BCNF

use the original set of dependencies F that hold on R, but with additional test

For every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$ or includes all attributes of R_i.

If the condition is violated by some $\alpha \rightarrow \beta$ in F^+, the dependency

$$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$

can be shown to hold on R_i, and R_i violates BCNF.

We use above dependency to decompose R_i

BCNF Decomposition Algorithm

Let F_c be a canonical cover for F *that constraints R.*

```
result := {R };
done := false;
while (not done) do
      if (there is a schema Ri in result that is not in BCNF)
         then begin
               let α → β  be a nontrivial functional dependency that
                     holds on Ri  such that α → Ri is not in F+,
                     and α ∩ β  = ∅;
            result := (result – Ri ) ∪ (Ri – β) ∪ (α, β );
               end
         else done := true;
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

BCNF Decomposition Algorithm Example

class (*course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id*)

Functional dependencies:

- *course_id*→ *title, dept_name, credits*
- *building, room_number*→*capacity*
- *course_id, sec_id, semester, year*→*building, room_number, time_slot_id*

A candidate key {*course_id, sec_id, semester, year*}

BCNF Decomposition:

- *course_id*→ *title, dept_name, credits* holds
  - but *course_id* is not a superkey.
- We replace *class* by:
  - *course*(*course_id, title, dept_name, credits*) BCNF
  - *class-1* (*course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id*)
- *building, room_number*→*capacity* holds on *class-1*
  - but {*building, room_number*} is not a superkey for *class-1*.
- We replace *class-1* by:
  - *classroom* (*building, room_number, capacity*) BCNF
  - *section* (*course_id, sec_id, semester, year, building, room_number, time_slot_id*) BCNF

Testing for 3NF

Need to check only FDs in *F*, need not check all FDs in F^+.

Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.

If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*

This test is rather more expensive, since it involve finding candidate keys

- Testing for 3NF has been shown to be NP-hard

Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

3NF Decomposition Algorithm

Let F_c be a canonical cover for F *that constraints R.*

```
i := 0;
for each functional dependency α → β in Fc do
    if none of the schemas Rj, 1 ≤ j ≤ i contains α β
        then begin
            i := i + 1;
            Ri := α β
        end

if none of the schemas Rj, 1 ≤ j ≤ i contains a candidate key for R
    then begin
        i := i + 1;
        Ri := any candidate key for R;
    end

/* Optionally, remove redundant relations */

repeat
    if any schema Rj is contained in another schema Rk
      then /* delete Rj */
        Rj = Ri;
        i=i-1;

return (R1, R2, ..., Ri)
```

3NF Decomposition Algorithm Example

- Relation schema:

  cust_banker_branch = (*<u>customer_id, employee_id</u>, branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *employee_id*

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id → *type*
    employee_id → *branch_name*
    customer_id, branch_name → *employee_id*

3NF Decomposition Algorithm Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id*, *branch_name, type*)

- The functional dependencies for this relation schema are:

  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *employee_id*

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id → *type*
    employee_id → *branch_name*
    customer_id, branch_name → *employee_id*

- The **for** loop generates following 3NF schema:
  - (*customer_id, employee_id, type*)
  - (*employee_id, branch_name*)
  - (*customer_id, branch_name, employee_id*)
- Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs to be added
- Detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:
  - (*customer_id, employee_id, type*)
  - (*customer_id, branch_name, employee_id*)

Comparison of BCNF and 3NF

It is always possible to decompose a relation into a set of relations that are in 3NF such that:

- The decomposition is lossless
- The dependencies are preserved

It is always possible to decompose a relation into a set of relations that are in BCNF such that:

- The decomposition is lossless
- It may not be possible to preserve dependencies.

Design Goals

Goal for a relational database design is:

- BCNF.
- Lossless join.
- Dependency preservation.

If we cannot achieve this, we accept one of

- Lack of dependency preservation
- Redundancy due to use of 3NF

FDs in SQL?

- Only using superkeys
- Can specify using assertions, but they are expensive

Latihan

Penduduk = (NIK, Nama, TanggalLahir, Alamat, KodeKelurahan, KodeKecamatan, KodeKabKota, KodeProvinsi, KodePos)

- NIK bernilai unik untuk setiap penduduk.
- Setiap kelurahan hanya memiliki sebuah kode pos dan beberapa kelurahan di kecamatan yang sama dapat memiliki kode pos yang sama.

Lakukan normalisasi terhadap Relasi Penduduk hingga menghasilkan relasi-relasi BCNF. Apakah skema yang dihasilkan dependency preserving?

Solusi Latihan

Penduduk = (NIK, Nama, TanggalLahir, Alamat, KodeKelurahan, KodeKecamatan, KodeKabKota, KodeProvinsi, KodePos)

- NIK bernilai unik untuk setiap penduduk.
- Setiap kelurahan hanya memiliki sebuah kode pos dan beberapa kelurahan di kecamatan yang sama dapat memiliki kode pos yang sama.

Lakukan normalisasi terhadap Relasi Penduduk hingga menghasilkan relasi-relasi BCNF. Apakah skema yang dihasilkan dependency preserving?

Tentukan FD yang berlaku pada Relasi Penduduk

- NIK bernilai unik untuk setiap penduduk

  NIK → Nama TanggalLahir Alamat KodeKelurahan ~~KodeKecamatan~~ ~~KodeKabKota~~ ~~KodeProvinsi~~ ~~KodePos~~

- Setiap kelurahan hanya memiliki sebuah kode pos dan beberapa kelurahan di kecamatan yang sama dapat memiliki kode pos yang sama

  KodeKelurahan → KodePos

  KodePos → KodeKecamatan

- Beberapa aturan yang telah berlaku umum tentang wilayah

  ~~KodeKelurahan → KodeKecamatan~~

  KodeKecamatan → KodeKabKota

  KodeKabKota → KodeProvinsi

Alternatif Solusi Latihan

Penduduk = (NIK, Nama, TanggalLahir, Alamat, KodeKelurahan, KodeKecamatan, KodeKabKota, KodeProvinsi, KodePos)

F_c={

[1]NIK → Nama TanggalLahir Alamat KodeKelurahan

[2]KodeKelurahan → KodePos

[3]KodePos → KodeKecamatan

[4]KodeKecamatan → KodeKabKota

[5]KodeKabKota → KodeProvinsi }

Candidate Key = {NIK}

Terapkan Algoritma Dekomposisi Menghasilkan Relasi 3NF

- Setiap f pada F_c membentuk sebuah relasi baru.
  - R1=(<u>NIK</u>, Nama, TanggalLahir, Alamat, KodeKelurahan)
  - R2=(<u>KodeKelurahan</u>, KodePos)
  - R3=(<u>KodePos</u>, KodeKecamatan)
  - R4=(<u>KodeKecamatan</u>, KodeKabKota)
  - R5=(<u>KodeKabKota</u>, KodeProvinsi)
- R1 sudah mengandung Candidate Key Relasi Penduduk
- Tidak ada relasi hasil dekomposisi yang merupakan subset dari relasi lainnya

Periksa apakah semua relasi sudah BCNF

- Pada R1 terdefinisi FD1, pada R2 FD2, dst.
- Pada setiap relasi, lhs dari FD adalah super key
- R1-R5 sudah BCNF

KNOWLEDGE & SOFTWARE ENGINEERING

Overall Database Design Process

Overall Database Design Process

R could have been generated when converting E-R diagram to a set of tables.

R could have been a single relation containing *all* attributes that are of interest (called **universal relation**).

R could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

When an E-R diagram is carefully designed, the tables generated from the E-R diagram should not need further normalization.

However, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity

Example: an *employee* entity with attributes *department_name* and *building*,

functional dependency *department_name*$\rightarrow$ *building*

Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

Example, displaying *prereqs* along with *course_id* and *title* requires join of *course* with *prereq*

Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes

- faster lookup
- extra space and extra execution time for updates
- extra coding work for programmer and possibility of error in extra code

Alternative 2: use a materialized view defined a *course* ⋈ *prereq*

- Benefits and drawbacks same as above
- except no extra coding work for programmer and avoids possible errors

Other Design Issues

Examples of bad database design

Having several relations *earnings_2014*, *earnings_2015*, *earnings_2016*, etc., all with the schema (*company_id, earnings*).

- Above are in BCNF, but make querying across years difficult and needs new table each year

Having one relation:
company_year (*company_id, earnings_2014, earnings_2015, earnings_2016*)

- Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
- Is an exa ... mes
- Used in s

Earnings (*company_id, year, amount*)

End of Topic