

# Normalisasi

## Bagian I

# First Normal Form (1NF)

- Domain disebut atomic bila elemen yang ada di dalamnya tidak dapat dibagi menjadi unit yang lebih kecil (*indivisible*)
- Sebuah skema relasi R berada dalam kondisi 1NF jika domain seluruh atribut dalam R atomic
- Nilai non-atomic membuat penyimpanan data menjadi rumit dan dapat mendorong terjadinya redundansi data. Contoh : himpunan account yang disimpan untuk tiap customer, dan himpunan pemilik yang disimpan untuk tiap account

# *Pitfalls* dalam Perancangan Basis Data Relasional

- Perancangan basis data relasional mensyaratkan penggunaan kumpulan relasi skema yang baik. Perancangan yang buruk dapat menyebabkan :
  - Pengulangan informasi
  - Ketidakmampuan dalam merepresentasikan informasi tertentu
- Tujuan perancangan :
  - Menghindari redundansi data
  - Memastikan keterhubungan antar atribut dapat direpresentasikan
  - Memfasilitasi pengecekan update yang menyebabkan pelanggaran integritas constraint basis data
- Contoh :  
lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)

<i>branch-name</i>	<i>branch-city</i>	<i>assets</i>	<i>customer-name</i>	<i>loan-number</i>	<i>amount</i>
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Ferryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

# *Pitfalls* dalam Perancangan Basis Data Relasional (lanj.)

- Permasalahan dalam instan lending-schema :
  - Redundansi
    - Data untuk branch-name, branch-city, dan assets diulangi untuk tiap loan yang dibuat
    - Tempat yang terbuang percuma
    - Proses update yang rumit, dapat menyebabkan inkonsistensi nilai atribut, contohnya untuk atribut assets
  - Nilai null
    - Tidak dapat menyimpan informasi tentang sebuah branch apabila tidak ada loan
    - Dapat diselesaikan dengan penggunaan nilai null, tetapi nilai null sulit ditangani

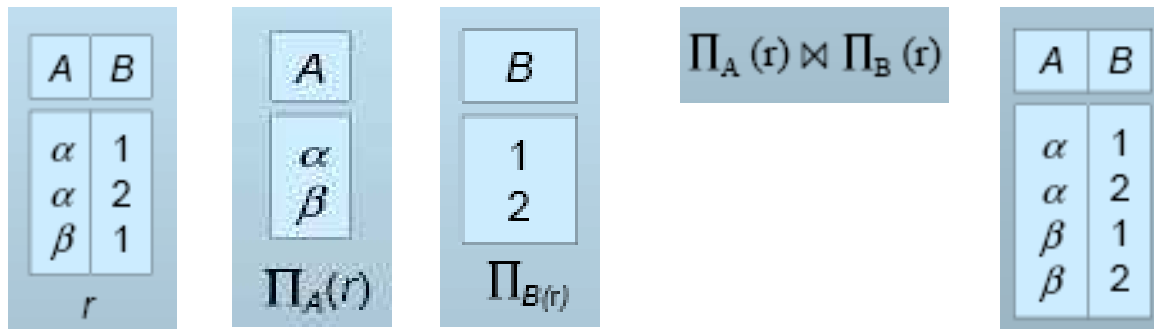
# *Decomposition*

- Skema relasi lending-schema didekomposisi menjadi :
  - Branch-schema = (branch-name, branch-city, assets)
  - Loan-info-schema = (customer-name, loan-number, branch-name, amount)
- Seluruh atribut dari skema asli (R) harus muncul di skema hasil dekomposisi ( $R_1, R_2$ ) :  
 $R = R_1 \cup R_2$
- *Lossless join decomposition*  
untuk semua kemungkinan relasi r dalam skema R

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

# Decomposition (lanj.)

- Contoh dekomposisi yang tidak memenuhi kondisi *lossless join decomposition*,  
 $R = (A,B)$  didekomposisi menjadi  $R_1=(A)$   
dan  $R_2=(B)$



# Tujuan Normalisasi

- Menentukan apakah relasi tertentu berada dalam “*good form*”
- Dalam kasus di mana relasi R tidak berada dalam “*good form*”, maka relasi tersebut didekomposisi menjadi himpunan relasi  $\{R_1, R_2, \dots, R_n\}$  di mana :
  - Setiap relasi berada dalam “*good form*”
  - Dekomposisi memenuhi syarat *lossless join decomposition*
- Normalisasi dilakukan berdasarkan :
  - *Functional dependencies*
  - *Multivalued dependencies*

# *Lossless Join Decomposition*

- Untuk setiap kemungkinan relasi  $r$  dalam skema  $R$

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

- Sebuah dekomposisi  $R$  menjadi  $R_1$  dan  $R_2$  dikatakan *lossless join* jika dan hanya jika paling tidak satu dari *dependency* berikut berada dalam  $F^+$  :

- $R_1 \cap R_2 \rightarrow R_1$

- $R_1 \cap R_2 \rightarrow R_2$

Jika tidak, maka dekomposisi yang dihasilkan akan menjadi lossy



# Normalisasi Menggunakan *Functional Dependency*

- Saat kita melakukan dekomposisi sebuah skema relasi R dengan himpunan *functional dependency* F menjadi  $R_1, R_2, R_3, \dots, R_n$  kita menginginkan :
  - *Lossless join decomposition* : apabila kondisi ini tidak terpenuhi, dapat menyebabkan hilangnya informasi
  - Tidak terdapat redundansi : relasi  $R_i$  lebih disukai berada dalam kondisi Boyce-Codd Normal Form atau Third Normal Form
  - Dependency preservation : misal  $F_i$  adalah himpunan dependency  $F^+$  yang hanya mengandung atribut dalam  $R_i$ 
    - Lebih disukai dekomposisi memenuhi kondisi *dependency preserving*, yaitu :  $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$
    - Bila tidak memenuhi kondisi *dependency preserving*, untuk mengecek pelanggaran *functional dependency* harus dilakukan join

# Contoh

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$   
dapat didekomposisi menggunakan dua cara :
- $R_1 = (A, B), R_2 = (B, C)$ 
  - Lossless join decomposition  
 $R_1 \cap R_2 = \{B\}$  dan  $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), R_2 = (A, C)$ 
  - Lossless join decomposition  
 $R_1 \cap R_2 = \{A\}$  dan  $A \rightarrow AB$
  - Tidak *dependency preserving*  
(tidak dapat mencek  $B \rightarrow C$  tanpa menghitung  $R_1 \bowtie R_2$ )

# Test untuk Dependency Preservation

- Untuk mengecek apakah sebuah *dependency*  $\alpha \rightarrow \beta$  *preserved* pada dekomposisi R menjadi  $R_1, R_2, \dots, R_n$ , dapat menggunakan prosedur berikut :
  - result =  $\alpha$   
while (changes to result) do  
  for each  $R_i$  in decomposition  
     $t = (\text{result} \cap R_i)^+ \cap R_i$   
    result = result  $\cup$  t
  - Jika result mengandung semua atribut dalam  $\beta$ , maka *functional dependency*  $\alpha \rightarrow \beta$  *preserved*
- Test tersebut diterapkan pada semua dependency di F untuk mengecek apakah sebuah dekomposisi memenuhi syarat *dependency preserving*
- Prosedur tersebut memerlukan *polynomial time* dalam pemrosesannya

# Second Normal Form (2NF)

- Sebuah relasi berada dalam kondisi 2NF jika dan hanya jika relasi tersebut :
  - Berada dalam kondisi 1NF
  - Tidak ada atribut nonkey yang tergantung secara parsial dengan relasi dalam skema tersebut

# Third Normal Form (3NF)

- Sebuah relasi berada dalam kondisi 3NF jika dan hanya jika :
  - Berada dalam kondisi 2NF
  - Setiap atribut nonkey bersifat *nontransitively dependent* pada *primary key*
- Functional dependency  $X \rightarrow A$  pada  $F^+$  disebut transitive jika  $X \rightarrow Y$  dan  $Y \rightarrow A$  ada di  $F^+$ , dan  $Y \rightarrow X$  tidak ada di  $F^+$
- 3NF mengasumsikan relasi hanya mempunyai satu *candidate key*
- 3NF tidak cukup memecahkan persoalan pada kasus di mana sebuah relasi :
  - Mempunyai dua atau lebih *candidate key*, di mana
  - *Candidate key* tersebut komposit
  - Terdapat overlap

# Definisi Formal 3NF

- Sebuah skema relasi R berada dalam kondisi 3NF jika untuk semua  $\alpha \rightarrow \beta$  dalam  $F^+$  paling tidak satu kondisi berikut terpenuhi :
  - $\alpha \rightarrow \beta$  trivial (misal,  $\beta \in \alpha$ )
  - $\alpha$  adalah *superkey* untuk R
  - Setiap atribut A dalam  $\beta - \alpha$  terkandung dalam sebuah *candidate key* untuk R (tiap atribut dapat berada dalam *candidate key* yang berbeda)
- Contoh :
  - $R = (J,K,L), F = \{JK \rightarrow L, L \rightarrow K\}$
  - Dua *candidate key* : JK dan JL
  - R berada dalam kondisi 3NF :
    - $JK \rightarrow L$  JK adalah *superkey*
    - $L \rightarrow K$  K terkandung dalam sebuah *candidate key*

# Testing untuk 3NF

- Optimization : hanya perlu mencek *functional dependency* dalam F, tidak perlu mencek seluruh *functional dependency* dalam F<sup>+</sup>
- Menggunakan *closure* atribut untuk mencek setiap *dependency*  $\alpha \rightarrow \beta$ , apakah  $\alpha$  superkey
- Jika  $\alpha$  bukan superkey, harus dilakukan verifikasi untuk tiap atribut dalam  $\beta$  apakah terkandung dalam sebuah *candidate key* R

# Tujuan Perancangan

- Tujuan dalam perancangan basis data relasional :
  - Lossless join
  - Dependency preservation
- Bila tujuan tersebut tidak dapat tercapai, akan terjadi :
  - Tidak mempunyai dependency preservation
  - redundansi
- SQL tidak menyediakan cara untuk menyatakan functional dependency selain penggunaan superkey. Bisa diterapkan menggunakan assertion, tetapi biayanya mahal.
- Bahkan jika hasil dekomposisi yang kita peroleh memenuhi syarat dependency preserving, penggunaan SQL tidak mampu untuk mentest secara efisien sebuah functional dependency yang sisi kirinya bukan merupakan key



# Proses Perancangan Basis Data Secara Umum

- Kita mengasumsikan sebuah skema R yang tersedia :
  - R dapat merupakan hasil saat mengubah diagram ER menjadi himpunan tabel
  - R dapat merupakan sebuah relasi tunggal yang mengandung semua atribut yang terhubung (disebut sebagai relasi universal)
  - Normalisasi memecah R menjadi beberapa relasi yang lebih kecil
  - R dapat merupakan hasil dari beberapa perancangan ad hoc relasi, yang kemudian ditest/diubah menjadi bentuk normal

# ER Model dan Normalisasi

- Saat sebuah diagram ER dirancang secara hati-hati, semua entitas didientifikasi secara benar, tabel yang di-generate dari diagram ER tersebut seharusnya tidak memerlukan proses normalisasi lagi
- Bagaimanapun, dalam kenyataannya perancangan dapat mengandung FD dari atribut nonkey ke atribut lainnya
- Contoh : entitas employee dengan atribut department-number dan department-address, dan sebuah FD department-number -> department-address
  - Perancangan yang baik akan membuat department sebagai sebuah entitas
- FD dari atribut nonkey dimungkinkan, tetapi jarang, sebagian besar relationship adalah binary

# Denormalisasi untuk Performansi

- Dalam beberapa kondisi, kita mungkin menggunakan skema non-normalized untuk meningkatkan performansi
- Contoh : untuk menampilkan customer-name bersama account-number dan balance membutuhkan join antara entitas account dan depositor
- Alternatif 1 : menggunakan denormalized relation yang mengandung atribut dari account dan juga dari depositor
  - Lookup lebih cepat
  - Membutuhkan tempat dan waktu ekstra pada update
  - Membutuhkan coding ekstra serta kemungkinan munculnya error
- Alternatif 2 : menggunakan materialized view yang didefinisikan dengan `account ⋈ depositor`
  - Manfaat dan kekurangan sama dengan alternatif 1, selain tanpa ekstra coding dan menghindari error yang mungkin

# Persoalan Perncangan Lainnya

- Beberapa aspek dalam perancangan basis data tidak ditangani dengan normalisasi
- Contoh : perancangan basis data yang buruk, yang harus dihindari :
- Daripada menggunakan earnings (company-id, year, amount), gunakanlah :
  - Earnings-2000, earnings-2001, earnings-2002, etc, semuanya dalam skema (company-id, earnings)
    - Seluruh relasi tersebut dalam BCNF, tetapi sulit membuat query yang meliputi beberapa year, dan jumlah tabel semakin bertambah sesuai dengan tahun
  - Company-year(company-id, earnings-2000, earnings-2001, earnings-2002)
    - Dalam BCNF juga, tetapi query yang melibatkan tahun yang berbeda juga sulit, dan untuk setiap tahun diperlukan atribut tambahan baru
    - Merupakan contoh dari crosstab, di mana nilai untuk sebuah atribut menjadi nama kolom
    - Digunakan dalam spreadsheet, dan dalam tools analisis data