PERTEMUAN 9

Penyempurnaan Skema dan Bentuk-bentuk Normal

Pokok Bahasan

- Persoalan-persoalan apa yang dapat ditimbulkan oleh adanya redundansi penyimpanan informasi?
- Apa yang dimaksud dengan *functional dependencies*?
- Apa yang dimaksud dengan bentuk-bentuk normal (*normal forms*) dan apa tujuannya?
- Apa manfaat dari BCNF dan 3NF?
- Apa pertimbangan dalam mendekomposisi relasi-relasi menjadi bentuk-bentuk normal?
- Dimana proses normalisasi dapat digunakan dalam proses desain basis data?
- Adakah bentuk kebergantungan (*dependency*) umum yang lebih bermanfaat dalam desain basis data?

Pengantar Penyempurnaan Skema: Persoalan yang Ditimbulkan oleh Redundansi

- *Redundansi ruang penyimpanan:* beberapa data disimpan secara berulang
- *Update anomaly:* Jika satu copy data terulang tsb diubah, inkonsistensi data dpt terjadi kecuali kalau semua copy dari data tsb diubah dengan cara yang sama
- *Insertion anomaly:* Mungkin dpt terjadi kesulitan utk menyisipkan data tertentu kecuali kalau beberapa data tidak terkait lainnya juga ikut disisipkan
- *Deletion anomaly:* Mungkin dpt terjadi kesulitan utk menghapus data tertentu tanpa harus kehilangan beberapa data tidak terkait lainnya

Persoalan yang Ditimbulkan oleh Redundansi: Contoh

SSN	Name	Lot	Rating	Wages	Hours
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Asumsi: nilai attribut wages ditentukan oleh nilai rating (utk satu nilai rating yang diberikan, hanya diperbolehkan terdapat satu nilai wages

- *Redundansi ruang penyimpanan:* nilai rating 8 yang berkorespondensi dg wages 10 diulang tiga kali
- *Update anomaly:* Nilai wages (yg terkait dengan nilai rating) dlm baris pertama dpt diubah tanpa membuat perubahan yg sama pada baris kedua dan kelima

Persoalan yang Ditimbulkan oleh Redundansi: Contoh

SSN	Name	Lot	Rating	Wages	Hours
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Asumsi: nilai attribut wages ditentukan oleh nilai rating (utk satu nilai rating yang diberikan, hanya diperbolehkan terdapat satu nilai wages

- *Insertion anomaly:* Kesulitan utk menyisipkan employee baru kecuali nilai wage untuk rating dari employee tsb sudah diketahui

- *Deletion anomaly:* Jika semua baris yang terkait dg nilai rating tertentu dihapus (misalnya baris utk employee 'Smethurst' dan 'Guldu' dihapus), maka kita akan kehilangan informasi ketergantungan antara nilai *rating* dan nilai *wages* yang diasosiasikan dengan nilai rating tsb (yaitu rating = 5 dan wages = 7)

Persoalan yang Ditimbulkan oleh Redundansi: Null Values

- Untuk kasus-kasus khusus, adanya nilai-nilai *null* yang berlebihan dalam suatu relasi dpt menimbulkan pemborosan pengunaan ruang penyimpanan.

- Hal ini terutama dpt terjadi pada suatu relasi dengan jumlah attribut yang besar dan jumlah baris yang juga besar, sehingga untuk kasus tertentu dapat terjadi banyak nilai-nilai kolom yang tidak memenuhi (*not applicable*) untuk sejumlah baris dalam relasi harus dibiarkan bernilai *null .*

- Sebagai contoh, utk relasi “Hourly Employees”, misalkan ditambah satu kolom baru (*OfficeLocCode*) utk mencatat kode lokasi kantor dari para pemimpin perusahaan. Jika misalnya terdapat ribuan employee, dan hanya ada sekitar 10% pemimpin, maka sebagian besar (90%) nilai kolom tersebut akan terisi dengan nilai *null* (pemborosan ruang penyimpan).

Pengantar Penyempurnaan Skema: Dekomposisi Skema Relasi

- *Proses Dekomposisi sebuah skema relasi R* berupa penggantian skema relasi menjadi dua (atau lebih) skema-skema baru yang masing-masing berisikan subset dari attribut-attribut relasi R dan kesemuanya memuat semua attribut yang ada dalam relasi R.

  - Proses dekomposisi dilakukan dengan menggunakan konsep ketergantungan fungsional (functional dependencies)

- Contoh: skema relasi “Hourly_Employees” dpt didekomposisi menjadi:
  - Hourly_Emps2 (*ssn*, *name*, *lot*, *rating*, *hours*)
  - Wages (*rating*, *wages*)

Wages

R	W
8	10
5	7

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Dekomposisi Skema Relasi: Beberapa Persoalan Terkait

- Dekomposisi terhadap suatu skema relasi harus digunakan dengan penuh pertimbangan. Dua pertanyaan yang harus selalu dipertimbangkan:
  1. Adakah alasan untuk mendekomposisi suatu relasi ?
  2. Persoalan-persoalan apa saja (jika ada) yang akan diakibatkan oleh dekomposisi ?
- Jawaban thdp pertanyaan pertama dpt dibantu dengan bentuk-bentuk normal (*Normal Forms*/NF) terhadap relasi yang akan didekomposisi. Utk ini jika suatu skema relasi berada dlm salah satu NF, maka beberapa persoalan yang terkait dengan dekomposisi tidak akan muncul
- Untuk jawaban thdp pertanyaan kedua, dua sifat penting dari dekomposisi harus dipertimbangkan:
  - Sifat *lossless-join* yang memungkinkan untuk membentuk kembali (recover) nilai-nilai relasi yang didekomposisi dari relasi-relasi hasil dekomposisi
  - Sifat *dependency-preservation* yang memungkinkan untuk memaksa agar constraints yang berlaku pada relasi asal tetap berlaku pada sejumlah relasi-relasi yang lebih kecil

Functional Dependencies (FDs)

- Suatu functional dependency X $\rightarrow$Y dikatakan berlaku pada relasi R jika, utk setiap nilai *r* dari R yang diperbolehkan, berlaku keadaan:
  - $t1 \in r$, $t2 \in r$, πX ($t1$) = πX ($t2$) mengimplikasikan πY ($t1$) = πY ($t2$)
  - yaitu, jika diberikan dua tuples dalam *r*, jika nilai proyeksi X pada kedua tuples sama, maka nilai proyeksi Y pada kedua tuples juga sama. (X dan Y adalah *sets* dari attributes pada relasi yang sama.)

- Sebuah FD adalah pernyataan yang berlaku pada semua relasi-relasi yang dimungkinkan.
  - Harus diidentifikasi berdasarkan semantik dari aplikasi
  - Jika diberikan beberapa nilai *r1* dari R yang mungkin, kita dpt melakukan pengecekan apakah nilai tersebut melanggar beberapa FD *f*, tetapi kita tidak dapat mengatakan bahwa *f* berlaku pada R!

- Jika K adalah sebuah *candidate key* untuk R, maka berarti bahwa K $\rightarrow$ R
  - Tetapi, K $\rightarrow$ R tidak mengharuskan K terdiri dari satu set attribut yang *minimal* !

Contoh: Constraints pada Entity Set

- Perhatikan relasi Hourly_Emps berikut:
  - Hourly_Emps (*ssn, name, lot, rating, hrly_wages*, *hrs_worked*)

- *Notasi*: Utk penyederhaan penulisan, skema relasi tsb akan dinotasikan dengan menggabungkan singkatan dari attribut-attributnya: SNLRWH
  - Notasi ini menyatakan satu *set* attributes {S,N,L,R,W,H}.
  - Dalam beberapa kasus, nama sebuah relasi akan digunakan untuk mengacu ke semua attribut dari relasi tersebut. (contoh, Hourly_Emps untuk SNLRWH)

- Beberapa FD yang berlaku pada Hourly_Emps:
  - *ssn* adalah sebuah key: S $\rightarrow$ SNLRWH
  - *rating* menentukan *hrly_wages*: R $\rightarrow$ W

Contoh (Lanjutan)

- Beberapa persoalan akibat R → W :
  - *Update anomaly*: Dapatkah W diubah hanya pada tuple pertama dari SNLRWH ?
  - *Insertion anomaly*: Bgm jika diinginkan utk menyisipkan seorang *employee* tetapi *hourly wage* utk *rating* yang bersangkutan tidak diketahui ?
  - *Deletion anomaly*: Jika semua employee dengan rating 5 dihapus, maka informasi mengenai *hourly wage* utk *rating* 5 juga akan ikut terhapus

SSN	Name	Lot	Rating	Wages	Hours
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Wages

R	W
8	10
5	7

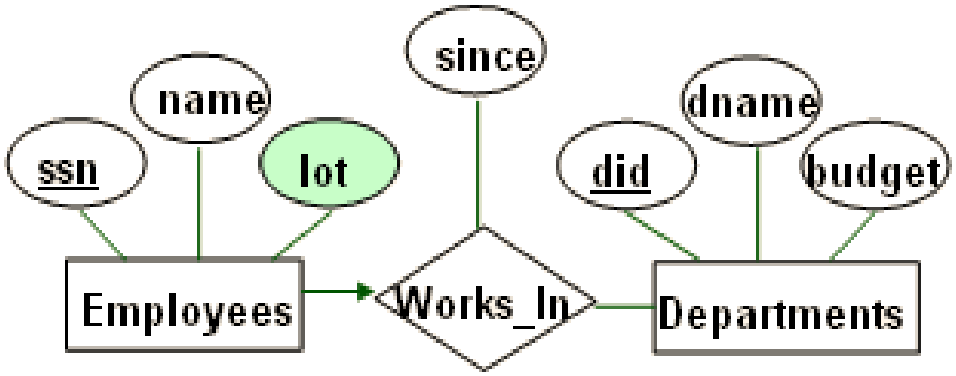
Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

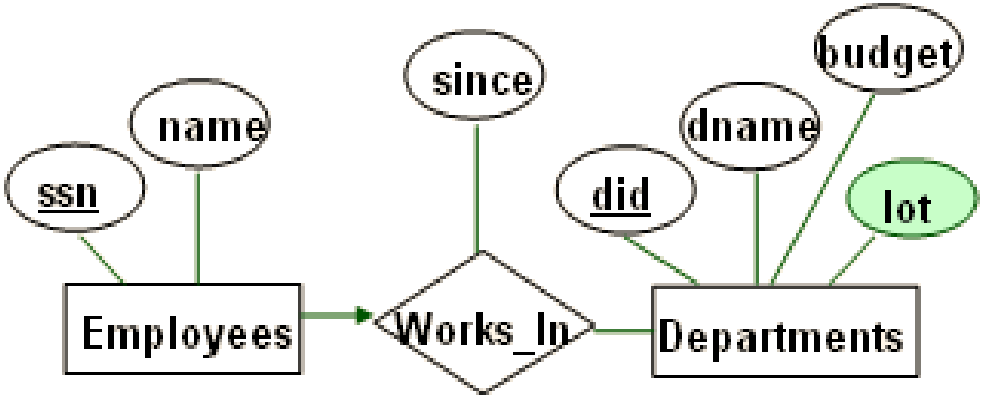
Penyempurnaan ER Diagram

- Pemetaan diagram pertama:
  Workers(S,N,L,D,S)
  Departments(D,M,B)
  - Lots diasosiasikan dengan relasi workers.
- Jika diasumsikan bhw semua workers dlm sebuah dept ditentukan sebuah lot yang sama, maka D → L (Redundansi)
- Redundansi dpt diatasi dg:
  Workers2(S,N,D,S)
  Dept_Lots(D,L)
  Departments(D,M,B)
- Dapat disempurnakan (*fine-tune*) menjadi:
  Workers2(S,N,D,S)
  Departments(D,M,B,L)

Sebelum (Workers):



Sesudah (Workers2):



Alasan Mengenai FD

- Jika diberikan satu set FDs, kita dapat menurunkan (*infer*) tambahan FDs: *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* mengimplikasikan *ssn* $\rightarrow$ *lot*

- Sebuah FD *f* dikatakan dapat diimplikasikan oleh (*implied by*) satu set FDs *F* jika *f* berlaku bilamana semua FDs dalam *F* berlaku.
  - *F*+ (*closure of F*) adalah set dari semua FDs yang diimplikasikan oleh *F*.

- Aksioma Armstrong (X, Y, Z adalah sets dari attributes):
  - *Reflexivity*: Jika X $\supseteq$ Y, maka X $\rightarrow$ Y
  - *Augmentation*: Jika X $\rightarrow$ Y, maka XZ $\rightarrow$ YZ utk sembarang Z
  - *Transitivity*: Jika X $\rightarrow$ Y dan Y $\rightarrow$ Z, maka X $\rightarrow$ Z

- Aksioma di atas merupakan aturan-aturan penyimpulan (*inference rules*) untuk FDs yang *logis (sounds)* dan *lengkap (complete)* !
- Dua aturan tambahan yang menyertai Aksioma Armstrong:
  - *Union*: Jika X $\rightarrow$ Y dan X $\rightarrow$ Z, maka X $\rightarrow$ YZ
  - *Decomposition*: Jika X $\rightarrow$ YZ, maka X $\rightarrow$ Y dan X $\rightarrow$ Z

Alasan Mengenai FD (Lanjutan)

- Suatu FD disebut *trivial* jika sisi kanan dari FD hanya terdiri dari attribut yang juga muncul di sisi kiri dari FD (akibat rumus *reflexivity*). Selain trivial FDs, selebihnya disebut *nontrivial* FDs.
  - Dengan menggunakan rumus reflexivity, semua trivial dependencies dapat diturunkan.

- Contoh: Contracts(*cid,sid,jid,did,pid,qty,value*), dan:
  - C adalah key: $C \rightarrow CSJDPQV$
  - Project membeli setiap part menggunakan contract tunggal: $JP \rightarrow C$
  - Dept membeli paling banyak satu part dari sebuah supplier: $SD \rightarrow P$

- Nontrivial FDs yang dapat diperoleh dari relasi 'Contracts':
  - $JP \rightarrow C$, $C \rightarrow CSJDPQV$ mengimplikasikan $JP \rightarrow CSJDPQV$
  - $SD \rightarrow P$ mengimplikasikan $SDJ \rightarrow JP$
  - $SDJ \rightarrow JP$, $JP \rightarrow CSJDPQV$ mengimplikasikan $SDJ \rightarrow CSJDPQV$

Alasan Mengenai FD: Attribute Closure

- Hanya untuk mengecek apakah sebuah dependensi, misalnya $X \rightarrow Y$ terdapat dlm closure dari satu set FDs *F*, kita dapat melakukannya secara efisien tanpa harus menghitung *F*+:
  - Hitung attribute closure X+ dg mengacu pada F, yaitu hitung satu attributes A sehingga $X \rightarrow A$ dpt diturunkan menggunakan Aksioma Armstrong (Algoritma utk menghitung attribute closure X+ dari set attribut X dg mengacu pada satu set FDs F):

    closure = *X*;
    REPEAT
    {
    IF there is an FD $U \rightarrow V$ in *F* such that $U \subseteq$ closure THEN
        set *closure* = *closure* $\cup$ *V*
    } UNTIL there is no change;
  - Kemudian cek apakah Y ada dalam X+
- Latihan: Apakah F = {$A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow E$} mengimplikasikan $A \rightarrow E$?
- Algoritma di atas dpt dimodifikasi utk memperoleh “keys” dari suatu skema relasi dengan cara memulai dengan set X yang terdiri dari satu attribut tunggal dan berhenti begitu *closure* berisikan semua attribut dari skema relasi.

Bentuk-Bentuk Normal (Normal Forms)

- Normal Forms (NF) digunakan utk membantu kita utk memutuskan apakah suatu skema relasi sudah merupakan hasil desain yang “baik” atau masih perlu didekomposisi menjadi relasi-relasi yang lebih kecil.

- Jika suatu relasi skema sudah berada dalam salah satu NF, berarti bhw beberapa jenis persoalan redundansi dapat dihindari/diminimalkan.

- NF yang didasarkan pada FDs: 1NF, 2NF, 3NF dan Boyce-Codd NF (BCNF):
  - Setiap relasi dlm BCNF juga berada dlm 3 NF
  - Setiap relasi dlm 3 NF juga berada dlm 2 NF, dan
  - Setiap relasi dlm 2 NF juga berada dlm 1NF

Bentuk-Bentuk Normal (Normal Forms)

- Setiap relasi yang berada dlm 1NF berlaku constraint bhw setiap field hanya berisikan nilai-nilai atomic (tidak boleh berisikan lists atau sets).
  - Dlm perkuliahan, constraint ini dianggap berlaku sebelum dilakukan proses normalisasi

- Oleh karena 2NF dibuat atas dasar sejarah perkembangan database (dari network model ke relational model), maka pembahasan hanya ditekankan pada proses pembentukan 3NF dan BCNF yang merupakan langkah penting dalam proses desain database.

Bentuk-Bentuk Normal (Normal Forms)

- Peran FD dalam mendeteksi redundansi: Perhatikan sebuah relasi R dengan 3 attributes ABC.
  - Jika tidak ada FD yang berlaku pada relasi R, maka dapat dipastikan tidak akan terdapat persoalan redundansi.
  - Namun, jika dimisalkan berlaku A $\rightarrow$ B, maka jika terdapat beberapa tuples yang mempunyai nilai A yang sama maka baris-baris tersebut juga harus mempunyai nilai B yang sama. Untuk ini, potensi terjadinya redundansi dapat diperkirakan dengan menggunakan informasi FDs

Boyce-Codd Normal Form (BCNF)

- Relasi R dg FDs *F* dikatakan berada dalam BCNF jika, utk semua FD $X \rightarrow A$ dalam F, salah satu dari pernyataan berikut harus berlaku:
  - $A \in X$ (disebut *trivial* FD), atau
  - X adalah key dari R.

- Dengan kata lain, R dikatakan berada dalam BCNF jika non-trivial FDs yang berlaku pada R hanya berupa key constraints.
  - Tidak ada redundansi yang dpt diprediksi hanya dengan menggunakan FDs saja
  - Jika terdapat dua tuples yang mempunyai nilai X yang sama, maka kita tidak dapat menyimpulkan nilai A dalam satu tuple dari nilai A dalam tuple lainnya
  - Namun, jika relasi contoh berada dalam BCNF, maka kedua tuples harus identik (karena X adalah sebuah key).

X	Y	A
x	y1	a
x	y2	?

Third Normal Form (3NF)

- Relasi R dg FDs *F* dikatakan berada dlm 3NF jika, untuk semua FD $X \rightarrow A$ dalam F, salah satu dari pernyataan berikut harus berlaku:
  - $A \in X$ (disebut *trivial* FD), atau
  - X adalah key dari R, atau
  - A adalah bagian dari beberapa key dari R (A adalah *prime attribute*)

- *Minimality* dari key dalam kondisi ketiga di atas menjadi sangat penting !

- Jika R berada dlm BCNF, sudah tentu juga berada dlm 3NF

- Jika R berada dlm 3NF, beberapa redundansi masih mungkin terjadi.
  - Bentuk 3NF dapat dipakai sebagai bentuk yang kompromistis dan digunakan bilamana BCNF tidak dapat diupayakan (misalnya karena tidak ada dekomposisi yang “baik”, atau karena alasan pertimbangan kinerja dari database)

Apa yang Dapat Dicapai oleh 3NF?

- Jika depedensi X $\rightarrow$ A menyebabkan pelanggaran dari 3NF, maka salah satu kasus di bawah ini akan terjadi:
  - X adalah subset dari beberapa key K (*partial dependency*)
    - Pasangan nilai (X, A) yang sama akan tersimpan secara redundan
  - X bukan subset dari sembarang key K (*transitive dependency*)
    - Terdapat mata rantai FDs K $\rightarrow$ X $\rightarrow$ A, yang berarti bhw kita tdk dpt mengasosiasikan sebuah nilai X dengan sebuah nilai K kecuali kalau kita juga mengasosiasikan sebuah nilai A dengan sebuah nilai X

- Namun demikian, walaupun seandainya relasi berada dalam 3NF, persoalan-persoalan berikut masih dpt terjadi:
  - Contoh: relasi Reserves SBDC (C=Credit Card ID), S $\rightarrow$ C, C $\rightarrow$ S berada dalam 3NF, tetapi utk setiap reservasi dari sailor S, pasangan nilai (S, C) yang sama akan tersimpan dalam database.

- Dengan demikian, 3NF memang merupakan bentuk normal yang relatif kompromistis terhadap BCNF.

Proses Dekomposisi dari sebuah Skema Relasi

- Asumsikan relasi R terdiri dari attributes *A1 … An.* Proses *dekomposisi* dari R meliputi penggantian R oleh dua atau lebih relasi sehingga :
  - Setiap skema relasi yang baru terdiri dari subset attribut dari R (dan tidak satupun attribut yang tidak muncul dalam R), dan
  - Setiap attribut dari R muncul sebagai sebuah attribut dari salah satu relasi-relasi yang baru

- Secara intuitif, pendekomposisian R berarti bahwa kita akan menyimpan nilai-nilai dari skema-skema relasi yang dihasilkan oleh proses dekomposisi, bukan nilai-nilai dari relasi R

- Contoh, relasi SNLRWH dapat didekomposisi menjadi SNLRH dan RW (lihat slide berikutnya).

Contoh Dekomposisi

- Proses dekomposisi sebaiknya digunakan hanya bilamana diperlukan.
  - SNLRWH mempunyai FDs $S \rightarrow$ SNLRWH dan $R \rightarrow W$
  - FD kedua menimbulkan pelanggaran 3NF; nilai-nilai W secara berulang diasosiasikan dg nilai-nilai R. Cara yang termudah utk memperbaiki ini adalah menciptakan relasi baru RW utk menyimpan asosiasi-asosiasi tersebut, dan untuk menghapus W dari skema utama, yaitu:
    - Kita dekomposisi SNLRWH menjadi SNLRH dan RW

- Informasi yang akan disimpan terdiri dari SNLRWH tuples. Jika kita hanya menyimpan proyeksi dari tuples ini pada SNLRH dan RW, adakah persoalan-persoalan potensial lain yang perlu dipertimbangkan? (lihat slide berikutnya)

Persoalan-persoalan yang Dapat Ditimbulkan oleh Dekomposisi

- Terdapat 3 persoalan potensial yang perlu diperhatikan:
  - Beberapa queries menjadi lebih mahal.
    - Contoh, Brp gaji yang diterima oleh Joe? (gaji = W*H)
  - Untuk nilai-nilai relasi hasil dekomposisi, mungkin kita tidak dapat merekonstruksi nilai-nilai relasi asal yang bersesuaian (*losless joins*) !
    - Kebetulan tidak terjadi pada contoh relasi SNLRWH
  - Pengecekan beberapa dependensi bisa jadi membutuhkan penggabungan (*joining*) nilai-nilai relasi hasil dekomposisi (*dependency preservation*) !
    - Kebetulan tidak terjadi pada contoh relasi SNLRWH

- *Tradeoff*: Harus mempertimbangkan issue ini, selain issue redundansi.

Dekomposisi yang Bersifat Lossless Join

- Dekomposisi R menjadi X dan Y disebut *lossless-join* dg mengacu pada satu set FDs F jika, untuk setiap instance *r* yang memenuhi F, berlaku:
  - $\pi X\ (r) \bowtie \pi Y\ (r) = r$

- Keadaan yang selalu harus benar: $r \subseteq \pi X\ (r) \bowtie \pi Y\ (r)$
  - Secara umum, arah sebaliknya tidak berlaku! Jika berlaku, maka dekomposisi bersifat *lossless-join*.

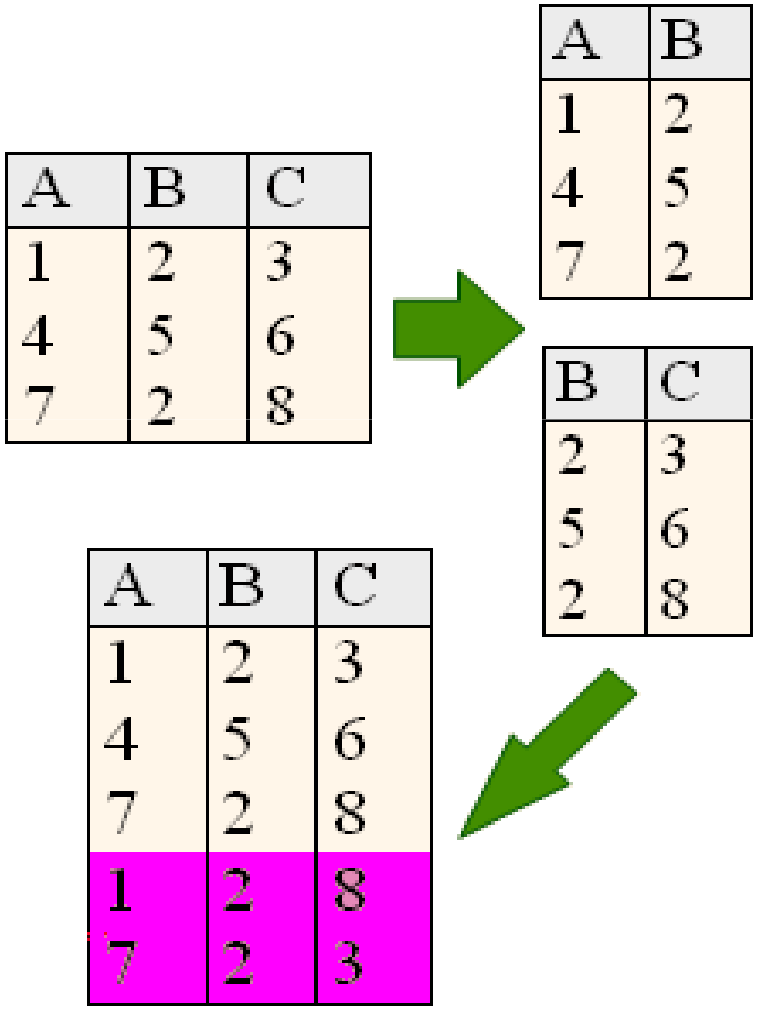
- Definisi di atas dapat secara mudah diperluas utk proses dekomposisi menjadi 3 relasi atau lebih

- *Penting untuk diperhatikan bhw semua jenis dekomposisi yang digunakan untuk menangani redundansi harus bersifat lossless! (Hindari persoalan ke-2)*

Lossless Join (Lanjutan)

- Dekomposisi R menjadi X dan Y bersifat lossless-join dg mengacu pada FDs F, jika dan hanya jika closure dari F (F+) berisikan:
  - $X \cap Y \rightarrow X$, atau
  - $X \cap Y \rightarrow Y$

- Secara umum, dekomposisi R menjadi UV dan R - V bersifat lossless-join jika $U \rightarrow V$ berlaku pada R dan $U \cap V = \varnothing$.



Dekomposisi yang Mempertahankan Dependensi

- Perhatikan CSJDPQV, C adalah key, JP $\rightarrow$ C dan SD $\rightarrow$ P.
  - Dekomposisi BCNF : CSJDQV dan SDP
  - Persoalan: Utk mengecek JP $\rightarrow$ C diperlukan operasi join!

- Dekomposisi yg mempertahankan dependensi (Intuitif):
  - Jika R didekomposisi menjadi X, Y dan Z, dan kita memaksa agar FDs tetap berlaku pada X, Y dan Z, maka semua FDs yang diberikan utk berlaku pada R hrs juga tetap berlaku. *(Menghindari persoalan ke-3)*

- *Projection dari set FDs F*: Jika R didekomposisi menjadi X, ... projection dari F pada X (disimbolkan FX) adalah set dari FDs U $\rightarrow$ V dalam F+ (*closure of F*) sedemikian rupa sehingga U, V ada dalam X.

Dekomposisi yang Mempertahankan Dependensi (Lanjutan)

- Dekomposisi R menjadi X dan Y bersifat *mempertahankan dependensi (dependency preserving)* jika $(FX \cup FY)+ = F+$, yaitu:
  - Jika kita hanya memperhatikan dependensi dalam F + yang dapat dicek dalam X tanpa memperhatikan Y, dan dalam Y tanpa memperhatikan X, maka hal ini mengimplikasikan bahwa semua dependensi ada dalam F +.

- Penting utk memperhatikan F +, BUKAN F, dalam definisi ini:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, didekomposisi menjadi AB dan BC.
  - Apakah bersifat *dependency preserving*? apakah $C \rightarrow A$ dipertahankan??

- *Dependency preserving* tidak mengimplikasikan *lossless join*:
  - ABC, $A \rightarrow B$, didekomposisi menjadi AB dan BC.

- Demikian juga sebaliknya, sifat *lossless-join* tidak mengimplikasikan *dependency preserving*

Dekomposisi menjadi BCNF

- Perhatikan relasi R dengan FDs F. Jika $X \rightarrow Y$ melanggar BCNF, lakukan dekomposisi R menjadi R - Y and XY.
  - Penggunaan secara berulang dari ide ini akan menghasilkan sekumpulan relasi yang berada dalam BCNF dan lossless join decomposition, dan dijamin utk mengalami terminasi.
  - Contoh, CSJDPQV, key C, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - Utk menangani $SD \rightarrow P$, dekomposisi menjadi SDP, CSJDQV.
  - Utk menangani $J \rightarrow S$, dekomposisi CSJDQV menjadi JS dan CJDQV

- Secara umum, beberapa dependensi yang diberikan dapat menimbulkan pelanggaran BCNF. Ingat, urutan "penanganan" dekomposisi seperti di atas dapat memberikan relasi hasil dekomposisi yang berbeda !

BCNF dan Dependency Preservation

- Secara umum, dimungkinkan dekomposisi menjadi BCNF tidak mempertahankan dependensi.
  - Contoh, CSZ, CS $\rightarrow$ Z, Z $\rightarrow$ C
  - Tdk dapat didekomposisi utk mempertahankan FD pertama (tidak dpt dilakukan dekomposisi BCNF).

- Dengan cara yang sama, dekomposisi CSJDQV menjadi SDP, JS dan CJDQV tidak mempertahankan dependensi (dengan mengacu ke FDs JP $\rightarrow$ C, SD $\rightarrow$ P dan J $\rightarrow$ S).
  - Namun demikian, dekomposisi di atas bersifat lossless-join.
  - Dalam kasus ini, penambahan JPC pada kumpulan relasi akan mememberikan dekomposisi yang dpt mempertahankan dependensi.
    - Penyimpanan tuples JPC hanya untuk tujuan pengecekan FD! (*Persoalan Redundansi!*)

Dekomposisi menjadi 3NF

- Algoritma untuk *lossless join decomposition* menjadi BCNF dapat digunakan utk memperoleh *lossless join decomposition* menjadi 3NF (dapat berhenti lebih awal).

- Untuk menjamin *dependency preservation*, suatu ide:
  - Jika $X \rightarrow Y$ tdk dipertahankan, tambahkan relasi XY.
  - Persoalan yang timbul adalah XY dpt melanggar 3NF! Contoh, perhatikan penambahan CJP utk mempertahankan $JP \rightarrow C$. Apa yang terjadi jika juga berlaku $J \rightarrow C$?

- Penyempurnaan: Sebagai pengganti set dari FDs F, gunakan *minimal cover dari F*.

Minimal Cover untuk Set dari FDs

- *Minimal cover* G utk sebuah set dari FDs F:
  - Closure dari F = closure dari G.
  - Bagian sisi kanan dari setiap FD dalam G berupa sebuah attribut tunggal.
  - Jika G diubah dengan menghapus sebuah FD atau dengan menghapus beberapa attributes dari sebuah FD dalam G, maka closure akan berubah.

- Secara intuitif, setiap FD dalam G diperlukan, dan harus *seminimal mungkin* untuk memperoleh closure yang sama seperti F.
- Contoh, $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ mempunyai *minimal cover* berikut:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ dan $EF \rightarrow H$

- *Minimal Cover* dapat menghasilkan dekomposisi yang bersifat *Lossless-Join* dan *Dependency Preserving*. Decomp !!

Rangkuman

- Jika sebuah relasi berada dalam BCNF, maka relasi tersebut bebas dari redundansi yang dapat dideteksi dengan menggunakan FDs.
  - Dengan demikian, upaua untuk menjamin bhw semua relasi berada dalam BCNF merupakan upaya heuristik yang baik.

- Jika sebuah relasi tidak berada dalam BCNF, coba lakukan dekomposisi menjadi sekumpulan relasi-relasi BCNF.
  - Harus mempertimbangkan apakah semua FDs dipertahankan. Jika dekomposisi menjadi BCNF yang bersifat lossless-join dan dependency preserving tidak dimungkinkan (atau tidak cocok, untuk beberapa queries yang tipikal), pertimbangkan dekomposisi menjadi 3NF.
  - Dekomposisi sebaiknya dilakukan dan/atau diperiksa kembali dengan mempertimbangkan *performance requirements* yang diinginkan.