

BAB III

ANALISIS PENANGANAN RELASI BITEMPORAL

Dalam mengembangkan sebuah API yang berfungsi untuk memberikan dukungan bagi penanganan relasi bitemporal, ada beberapa aspek yang perlu tersedia dalam API:

1. Dukungan bagi *query* temporal.
Pengguna dapat menuliskan *query* temporal dan mendapatkan hasil yang diinginkan. API bertugas untuk mengkonversi *query* temporal menjadi *query* yang dipahami oleh DBMS relasional. Hal ini juga mencakup dukungan bagi operator perbandingan interval yang hanya dikenal pada basis data temporal.
2. Penanganan integritas data berdasarkan konsep basis data temporal.
Integritas data bagi relasi bitemporal tidak dapat ditangani oleh *constraint* seperti *primary key* dan *foreign key* yang ada pada DBMS relasional, sehingga perlu ada penanganan khusus.
3. Dukungan bagi nilai waktu basis data temporal.
Pada basis data temporal, terdapat variabel dan konstanta waktu yang tidak dikenal oleh basis data relasional. API harus dapat merepresentasikan nilai waktu tersebut ke dalam nilai yang dikenali DBMS relasional, sambil tetap dapat menampilkan nilai tersebut kepada pengguna dalam bentuk nilai waktu basis data temporal.

Agar dapat menerapkan aspek tersebut, dilakukan analisis untuk hal-hal sebagai berikut:

1. Penerapan variabel dan konstanta waktu, yaitu implementasi penggunaan nilai waktu yang digunakan pada basis data temporal, seperti yang terdapat pada Subbab 2.6.
2. Penerapan operator perbandingan temporal, yaitu implementasi penanganan operator perbandingan nilai waktu yang digunakan pada basis data temporal, seperti yang terdapat pada Subbab 2.5.
3. Penanganan *primary key* dan *foreign key*, meliputi analisis implementasi *primary key* dan *foreign key* temporal pada basis data relasional. Analisis ini juga meliputi kasus yang mungkin terjadi saat operasi temporal dilakukan.
4. Konversi *query*, meliputi operasi *create table* dan manipulasi data, serta operasi *data retrieval*. Model konversi *query* dibuat berdasarkan model 1NF dengan *tuple stamping* seperti yang dapat dilihat pada subbab 2.2.2. Format waktu yang didukung adalah *time interval* dengan representasi *closed ended*. Operasi *data retrieval* meliputi analisis mengenai kondisi seleksi yang mungkin dilakukan, termasuk proses *coalescing*.

Batasan terhadap analisis yang dilakukan:

1. *Table constraint* yang ditangani hanya *primary key* dan *foreign key*.
2. Analisis penanganan integritas entitas hanya mencakup satu *key* pada relasi (hanya mencakup *primary key*, tetapi tidak mencakup *constraint unique* untuk atribut lain).
3. Analisis penanganan *foreign key* hanya mencakup aksi *restrict*.
4. Operasi seleksi tanpa *keyword* SNAPSHOT tidak mencakup *join* yang dilakukan terhadap lebih dari dua relasi.
5. Operasi seleksi tidak mencakup proyeksi nilai *valid time* dengan klausa `VALID` atau `VALID INTERSECT`.
6. Nilai *valid time* yang diterima pada klausa `VALID` operasi *insert*, *delete*, dan *update* adalah `PERIOD [<waktu_awal>,<waktu_akhir>]`.

3.1 Penerapan Variabel dan Konstanta Waktu

3.1.1 Kasus 'Now'

Nilai 'now' dipandang sebagai sebuah variabel yang berubah nilainya sesuai dengan perubahan waktu. Nilai 'now' di-*bind* dengan nilai waktu saat ini. Secara konsep, 'now' pada *valid time* sama dengan 'UC' pada *transaction time*. Akan tetapi, berbeda dengan 'UC' yang dianggap sebagai nilai terbesar dalam *transaction time*, nilai 'now' bukan merupakan nilai terbesar pada *valid time*. Penerapan konsep ini dapat menimbulkan masalah terkait dengan integritas entitas.

nama	gaji	V
Wiyanda Puspita	4000000	[1/1/07,now]

Gambar III-1 Contoh relasi yang mengandung nilai 'now'

Sebagai contoh, terdapat relasi seperti pada Gambar III-1 dengan *primary key* nama. Pada tanggal 7 November 2007, dilakukan *insert* terhadap relasi tersebut dengan nama "Wiyanda Puspita", gaji 4500000, dan keberlakuan *valid time* 1 Januari 2008 sampai dengan 31 Desember 2008. Secara konseptual, nilai tersebut dapat diterima oleh basis data karena tidak beririsan dengan waktu yang ada. 'Now' di-*bind* dengan nilai saat itu, yaitu 7 November 2007. Misalkan *insert* diterima dan menghasilkan nilai *tuple* seperti pada Gambar III-2.

nama	gaji	V
Wiyanda Puspita	4000000	[1/1/07,now]
Wiyanda Puspita	4500000	[1/1/08,31/12/08]

Gambar III-2 Contoh relasi dengan kasus 'now'

Saat *tuple* tersebut dimasukkan, kondisi tersebut tidak melanggar integritas data karena nilai *valid time* tidak beririsan. Akan tetapi, integritas data ini berkaitan dengan berjalannya waktu.

Setelah tanggal 1 Januari 2008, relasi tidak memenuhi integritas data karena saat itu terdapat dua nilai gaji untuk “Wiyanda Puspita”.

Karena itu, walaupun secara konseptual dimungkinkan untuk memasukkan *tuple* yang *primary key equivalent* dengan nilai *valid time* yang lebih besar dari ‘now’, kondisi tersebut tidak akan diterima dalam penerapannya di basis data. Berdasarkan hal ini, tidak dimungkinkan juga memasukkan *tuple* yang *primary key equivalent* dan mempunyai nilai *valid time* akhir ‘now’ jika telah ada *tuple* yang mempunyai nilai *valid time* lebih besar dari nilai waktu saat itu. Dengan membatasi hal ini, integritas entitas dapat ditangani dengan lebih mudah.

Selain berpotensi mendatangkan masalah bagi penanganan integritas entitas, nilai ‘now’ juga dapat bermasalah pada penanganan *referential integrity*. Sebagai contoh, terdapat relasi (a) seperti pada Gambar III-3. Relasi ini nilai atribut *id_dept*-nya diacu oleh atribut *id_dept* pada relasi (b) yang mempunyai atribut *nip* dan *id_dept*.

(a)

id dept	nama dept	V
1	Produksi	[1/1/07,31/12/07]
2	Keuangan	[1/1/07,31/12/07]

Gambar III-3 Contoh relasi acuan untuk kasus 'now'

Jika pada tanggal 7 November 2007 dimasukkan sebuah *tuple* ke relasi (b) seperti pada Gambar III-4, secara konseptual *tuple* tersebut dapat diterima karena nilai *valid time tuple* yang diacu pada relasi (a) mencakup nilai *valid time tuple* pada relasi (b), yaitu 1 Januari 2007 sampai *current date* (7 November 2007).

(b)

nip	id_dept	V
10030	1	[1/1/07,now]

Gambar III-4 Contoh relasi dengan kasus 'now' untuk referential integrity

Akan tetapi, setelah tanggal 1 Januari 2008, *referential integrity* tidak terjaga karena *valid time tuple* di relasi (a) tidak mencakup *valid time tuple* di relasi (b). Nilai *valid time* akhir *tuple* di relasi (b) telah melampaui nilai *valid time* akhir *tuple* di relasi (a), sehingga kondisi *tuple* seperti pada Gambar III-4 tidak akan diterima di basis data. Nilai *valid time* ‘now’ untuk *tuple* di relasi (b) hanya akan diterima jika *tuple* yang diacu di relasi (a) juga memiliki nilai *valid time* akhir ‘now’ atau ‘forever’ karena walaupun waktu terus berjalan, nilai *valid time* di *tuple* yang mengacu tidak akan pernah melampaui nilai *valid time* di *tuple* yang diacu.

3.1.2 Implementasi Nilai Waktu

Untuk mendukung nilai waktu pada basis data temporal, perlu dilakukan representasi terhadap nilai-nilai khusus yang tidak terdapat di basis data relasional, yaitu ‘beginning’, ‘forever’,

'now', dan 'UC'. Nilai 'beginning' dan 'forever' adalah nilai yang tidak akan berubah, sehingga nilainya dapat disimpan dengan nilai waktu terkecil dan terbesar dalam basis data. Untuk *chronon* hari, nilai 'beginning' dapat direpresentasikan dengan tanggal '01-01-0001' dan 'forever' dengan tanggal '31-12-9999'.

Nilai aktual 'now' dan 'UC' berubah sesuai dengan waktu. Pada dimensi *valid time*, nilai terkecil adalah 'beginning' dan nilai terbesar adalah 'forever', sehingga tidak ada satu nilai waktu di antara rentang tersebut yang dapat merepresentasikan 'now'. Jika nilai 'now' direpresentasikan oleh nilai waktu di antara rentang waktu tersebut, nilai tersebut harus di-*update* setiap perubahan nilai waktu. Berdasarkan kondisi tersebut, 'now' pada basis data dapat direpresentasikan dengan nilai *null*, yang merupakan nilai yang tidak terkait dengan satu nilai waktu di basis data. Ini dipilih dengan pertimbangan bahwa nilai *null* sendiri tidak akan pernah digunakan sebagai nilai kosong pada *valid time* karena basis data temporal tidak memperbolehkan hal tersebut. 'UC' merupakan nilai paling besar dalam dimensi *transaction time*, sehingga di basis data dapat direpresentasikan dengan nilai terbesar di basis data. 'UC' juga dapat direpresentasikan dengan *null* seperti pada nilai 'now'. Pada tugas akhir ini, 'UC' akan direpresentasikan dengan nilai *null*. Dengan adanya representasi nilai-nilai ini, dibutuhkan konversi untuk nilai waktu basis data temporal dan nilai representasi di basis data.

Konversi yang dilakukan meliputi hal-hal berikut:

- Konversi untuk mempresentasikan nilai waktu, yaitu konversi nilai waktu di basis data ke nilai *valid time* yang akan ditampilkan saat operasi *select*. Nilai *null* dikonversi menjadi 'now', '01-01-0001' menjadi 'beginning', dan '31-12-9999' menjadi 'forever'. Konversi ini dilakukan dengan menggunakan klausa *CASE* pada *query* konversi. Untuk mengkonversi nilai *vs*, pada *query* digunakan klausa *CASE WHEN vs = '00010101' THEN 'beginning' ELSE vs END*. Sedangkan untuk mengkonversi nilai *ve*, digunakan klausa *CASE WHEN ve is null THEN 'now' WHEN ve = '99991231' THEN 'forever' ELSE ve END*. Untuk menghasilkan klausa ini, pada API digunakan fungsi `dbtotable(string): string` yang akan menghasilkan string konversi untuk nilai atribut *vs* dan *ve*.
- Konversi untuk menyimpan nilai waktu, yaitu konversi nilai waktu *valid time* dan yang dimasukkan pada *query* ke nilai di basis data. Nilai 'now' dikonversi menjadi *null*, 'beginning' menjadi '01-01-0001', dan 'forever' menjadi '31-12-9999'. Konversi ini dilakukan oleh fungsi `querytodb(string): string` yang terdapat pada API. Fungsi ini menerima nilai *valid time* awal dan akhir yang dimasukkan dalam *query* dan

mengkonversinya menjadi nilai yang bersesuaian pada basis data. Nilai yang dimasukkan pada *query* non-temporal adalah nilai konversi ini.

- Konversi untuk perhitungan waktu. Konversi ini mengubah nilai yang ada di basis data dan nilai yang ada pada *query* ke nilai aktualnya. Nilai *null* pada basis data dikonversi menjadi *current date*. Konversi ini dilakukan dengan menggunakan klausa `CASE WHEN ve is null THEN CURRENT_DATE ELSE ve END` pada *query* konversi. Untuk menghasilkan klausa *query* ini, digunakan fungsi `dbtoactual(string): string` pada API. Sedangkan untuk konversi nilai pada *query*, nilai 'now' dan 'UC' dikonversi menjadi *current date*, 'beginning' menjadi '01-01-0001', dan 'forever' menjadi '31-12-9999'. Konversi ini dilakukan oleh fungsi `querytoactual(string): string` yang terdapat pada API. Nilai yang dibandingkan pada *query* konversi adalah nilai yang didapatkan dari fungsi tersebut.

3.2 Penerapan Operator Perbandingan Temporal

Pada operasi yang melibatkan klausa kondisi, seperti *select*, *delete*, dan *update*, dibutuhkan penanganan bagi operator perbandingan temporal yang digunakan pada klausa tersebut. Operator perbandingan *valid time* dapat dilihat pada Tabel II-1. Dengan definisi yang sama, operator perbandingan ini juga dapat digunakan untuk perbandingan *transaction time*. Akan tetapi, secara konseptual nilai *transaction time* pada relasi bitemporal hanya digunakan untuk melihat *obsolete database* lewat operasi *select*. Operasi *delete* dan *update* berdasarkan kondisi *transaction time* tidak diperbolehkan. Karena itu, operator perbandingan temporal untuk *transaction time* hanya dapat digunakan pada operasi *select*.

Selain itu, pada API yang akan dibangun, nilai *transaction time* tidak akan ditampilkan pada saat operasi *select*. Operasi *select* berdasarkan kondisi *transaction time* hanya berlaku untuk satu titik *transaction time* tertentu. Karena itu, kondisi perbandingan *transaction time* yang diperbolehkan hanya `OVERLAPS`, `=`, `CONTAINS`, dan `MEETS` terhadap *event* tertentu. Perbandingan dengan operator `PRECEDES` untuk perbandingan *transaction time* tidak diterima karena dapat mengembalikan beberapa *tuple* yang terdapat pada keberlakuan waktu berbeda di basis data.

3.2.1 Konversi Operator Temporal

Penanganan operator perbandingan temporal akan dilakukan dengan membuat fungsi pada API yang akan melakukan konversi string kondisi perbandingan waktu berdasarkan masukan

empat titik waktu, yaitu waktu awal periode pertama, waktu akhir periode pertama, waktu awal periode kedua, dan waktu akhir periode kedua. Fungsi untuk tiap operator dapat dilihat pada Tabel III-1. Untuk perbandingan *valid time* "a <operator> b", vs1 dan ve1 adalah waktu awal dan akhir a, sedangkan vs2 dan ve2 adalah waktu awal dan akhir b. Jika operator perbandingan temporal pada Tabel III-1 ditemukan pada *query* temporal, string kondisi yang dimasukkan pada *query* non-temporal adalah hasil konversi dari fungsi yang bersesuaian. Semua periode waktu yang dibandingkan harus mempunyai nilai periode yang valid, yaitu $ve \geq vs$.

Tabel III-1 Fungsi untuk penanganan operator perbandingan temporal

Operator	Fungsi	Konversi
PRECEDES	sys_precedes(vs1,ve1,vs2,ve2)	$ve1 < vs2$
=	sys_equals(vs1,ve1,vs2,ve2)	$vs1 = vs2$ AND $ve1 = ve2$
OVERLAPS	sys_overlaps(vs1,ve1,vs2,ve2)	($vs1 \leq vs2$ AND $ve1 \geq vs2$) OR ($vs1 \leq ve2$ AND $ve1 \geq ve2$) OR ($vs2 < vs1$ AND $ve2 > ve1$)
CONTAINS	sys_contains(vs1,ve1,vs2,ve2)	$vs1 \leq vs2$ AND $ve1 \geq ve2$
MEETS	sys_meets(vs1,ve1,vs2,ve2)	$ve1 = vs2 - 1$ satuan <i>chronon</i>

3.2.2 Konversi Klausa Kondisi

Fungsi pada Tabel III-1 akan digunakan jika operator perbandingan temporal ditemukan dalam klausa kondisi pada *query*. Pada operasi relasi bitemporal, klausa kondisi untuk operasi manipulasi data dan *data retrieval* berbeda berdasarkan ekspresi perbandingan temporal yang diterima. Pada operasi manipulasi data, hanya ekspresi perbandingan *valid time* yang boleh dilakukan. *Grammar* klausa kondisi yang diterima pada operasi *delete* dan *update* dapat dilihat pada Gambar III-5.

```

<condition> ::= <condition_expr> (<logical_operator>
                <condition_expr>)*
<condition_expr> ::= (not)? <comparison_expr> | <valid_expr> |
                (<condition>)
<logical_operator> ::= and | or
<valid_expr> ::= valid(<table_name>) <temporal_operator> <time_value>
<temporal_operator> ::= overlaps | contains | meets | precedes | =
<time_value> ::= period '['<date_start>,<date_end>]' | date
                <date_value>
<comparison_expr> ::= <expr> (<comparison_operator> <expr>)?
<comparison_operator> ::= < | <= | = | >= | > | <> | like | is (not)?
<expr> ::= (<column_name> | <expr_value>) (<operator> (<column_name>
                | <expr_value>))*
<operator> ::= + | - | div | / | mod | *

```

Gambar III-5 Grammar klausa kondisi pada *delete* dan *update*

Sebagai contoh, klausa kondisi yang diterima adalah `attr1>=10 and valid(t) overlaps period '[1 Jan 07, now]'`. Ekspresi perbandingan non-temporal seperti `attr1>=10` tidak perlu dikonversi, sedangkan ekspresi perbandingan *valid time* perlu dikonversi dengan fungsi `sys_overlaps()` pada Tabel III-1 untuk menghasilkan string *query* non-temporal yang bersesuaian. *Grammar* untuk konversi ekspresi perbandingan *valid time* dapat dilihat pada Gambar III-6.

```

<converted_valid_expr> ::= <temporal_function> (
                dbtoactual(<table_name>.vs),
                dbtoactual(<table_name>.ve),
                querytoactual(<date_start>),
                querytoactual(<date_end>))
<temporal_function> ::= sys_<temporal_operator>
<converted_condition_expr> ::= <comparison_expr> |
                <converted_valid_expr> | (<condition>)
<converted_condition> ::= <converted_condition_expr>
                (<logical_operator>
                <converted_condition_expr>)*

```

Gambar III-6 Grammar konversi ekspresi perbandingan *valid time*

Jika operator temporal yang digunakan adalah '=', fungsi yang digunakan adalah `sys_equals()`. Jika perbandingan dilakukan dengan nilai `date_value`, nilai `date_start` dan `date_end` adalah `date_value`. Pada *query* non-temporal, kondisi `attr1>=10 and valid(t) overlaps period '[1 Jan 07, now]'` akan dikonversi menjadi seperti pada Gambar III-7.

```
attr1>=10 and sys_overlaps(dbtoactual(t.vs), dbtoactual(t.ve),
querytoactual('1 Jan 07'), querytoactual('now'))
```

Gambar III-7 Contoh I hasil konversi klausa kondisi

Grammar klausa kondisi *select* sedikit berbeda dengan *grammar* pada Gambar III-5 karena menerima ekspresi perbandingan *transaction time*. Jika tidak terdapat ekspresi perbandingan *transaction time*, perbandingan yang dilakukan adalah `transaction(<table_name>) overlaps DATE CURRENT_DATE`. *Grammar* untuk klausa kondisi pada *select* dapat dilihat dari aturan produksi `<select_condition>` pada Gambar III-8. Aturan produksi `<logical_operator>`, `<valid_expr>`, dan `<comparison_expr>` sama dengan pada Gambar III-5. *Grammar* konversi klausa kondisinya dapat dilihat pada Gambar III-9.

```
<select_condition> : <select_condition_expr> (<logical_operator>
                    <select_condition_expr>)*
<select_condition_expr> ::= <comparison_expr> | <valid_expr> |
                          <transaction_expr> | (<select_condition>)
<transaction_time> ::= transaction(<table_name>)
                      <transaction_operator> date '<date_value>'
<transaction_operator> ::= overlaps | contains | meets | =
```

Gambar III-8 Grammar klausa kondisi pada operasi *select*

```
<converted_transaction_expr> ::= <temporal_function> (
                                dbtoactual(<table_name>.ts),
                                dbtoactual(<table_name>.te),
                                <date_value>, <date_value> )
<converted_condition_expr> ::= <comparison_expr> |
                              <converted_valid_expr> |
                              <converted_transaction_expr> |
                              (<condition>)
<converted_select_condition> ::= <converted_condition_expr>
                                (<logical_operator>
                                <converted_condition_expr>)*
```

Gambar III-9 Grammar konversi ekspresi perbandingan *transaction time*

Konversi yang dilakukan untuk ekspresi perbandingan *transaction time* mempunyai model yang sama dengan konversi ekspresi perbandingan *valid time*. Ekspresi yang melibatkan operator perbandingan temporal seperti `transaction(t) overlaps date '1 Jan 07'` akan dikonversi menjadi `sys_overlaps(dbtoactual(t.ts), dbtoactual(t.te), '1 Jan 07', '1 Jan 07')`.

3.3 Penanganan *Primary Key* dan *Foreign Key*

Suatu relasi bitemporal memenuhi integritas entitas jika untuk setiap *tuple* yang berada pada masa keberlakuan *transaction time* dan *valid time* yang sama, tidak ada *tuple* dengan nilai atribut *key* yang sama. Pada relasi bitemporal, pelanggaran terhadap integritas data terjadi saat operasi *insert* dan *update* temporal, karena saat itu terdapat *tuple* baru yang dimasukkan ke dalam relasi dengan nilai atribut *key* dan *valid time* yang dapat beririsan dengan *tuple* yang sudah ada. Karena itu, penanganan integritas data harus dilakukan secepatnya saat operasi tersebut dijalankan, sehingga penanganan cukup dilakukan untuk data saat ini. Dengan kata lain, untuk setiap *tuple* yang masih berlaku saat ini, tidak boleh ada *tuple* bernilai atribut *key* yang sama dengan keberlakuan *valid time* yang beririsan.

Referential integrity pada relasi bitemporal akan terpenuhi jika untuk *tuple* t1 di relasi R1 yang diacu oleh *tuple* t2 di relasi R2, *valid time* t1 *contains valid time* t2 untuk setiap titik waktu. Dengan kata lain, selain nilai atribut *foreign key* harus terdapat pada *tuple* di relasi acuan, *tuple* juga harus berlaku pada titik waktu saat *tuple* acuan berlaku.

Integritas entitas akan ditangani oleh pendefinisian *primary key*, sedangkan *referential integrity* akan ditangani oleh pendefinisian *foreign key*. Penanganan *primary key* dan *foreign key* relasi bitemporal yang diterapkan pada DBMS relasional dapat dilakukan dengan beberapa cara, yaitu:

1. Pendefinisian *trigger* dan fungsi pada DBMS. Ini merupakan cara yang efisien untuk diterapkan karena semua proses penanganan *primary key* dan *foreign key* akan dilakukan oleh DBMS. Akan tetapi, fungsi yang didefinisikan bersifat spesifik terhadap skema relasi. Jika dilakukan perubahan terhadap skema relasi, seperti penambahan atribut atau perubahan *constraint*, fungsi yang didefinisikan untuk penanganan tersebut juga harus diubah.
2. Pembuatan tabel sistem yang menyimpan data mengenai *primary key* dan *foreign key* temporal yang didefinisikan pada basis data. Dengan cara ini, DBMS tidak melakukan penanganan integritas data. Penanganan terhadap *constraint* tersebut dilakukan oleh API dengan melihat data pada tabel sistem dan mengecek kesesuaian operasi yang dilakukan dengan *constraint* yang berkaitan. Pemeliharaannya bersifat sederhana karena hanya mencakup data pada tabel sistem. Jika dilakukan perubahan *constraint*, cukup dilakukan perubahan data pada tabel sistem.

Pada tugas akhir ini, penanganan *primary key* dan *foreign key* dilakukan dengan melakukan pendefinisian *trigger* dan fungsi. Pembuatan *trigger* dan fungsi di DBMS tidak dilakukan

oleh pengguna API, tetapi dilakukan oleh API saat *query create table* bitemporal diterima. Cara ini dipilih karena efisien untuk diterapkan dalam pembangunan API. API cukup mendefinisikan tiap *trigger* dan fungsi satu kali dan selanjutnya penanganan dilakukan oleh DBMS. Ini berbeda dengan cara penanganan lewat tabel sistem yang mengharuskan API untuk melakukan pengecekan setiap kali operasi temporal dilakukan. Akan tetapi, penanganan dengan *trigger* dan fungsi mempunyai konsekuensi, yaitu *trigger* dan fungsi yang terletak di DBMS ini dapat dimodifikasi oleh pengguna DBMS sehingga menyebabkan penanganan *primary key* dan *foreign key* tidak lagi dapat berjalan dengan baik. Dalam hal ini, diasumsikan bahwa keamanan DBMS cukup baik sehingga tidak ada yang akan mengubah *trigger* dan fungsi di basis data.

Masalah lain dalam melakukan penanganan dengan *trigger* dan fungsi adalah pemeliharannya. Informasi mengenai *constraint* dan fungsi yang menangani tiap *constraint* perlu disimpan. Karena itu, dalam pembangunan API pada tugas akhir ini didefinisikan juga tabel sistem pada basis data yang berisi data mengenai *primary key*, *foreign key*, dan fungsi yang menanganinya. Tabel sistem ini digunakan agar saat terjadi perubahan *constraint* dapat diketahui fungsi mana yang harus dihapus atau diperbaharui.

3.3.1 Primary Key

3.3.1.1 Konsep Primary Key Temporal

Primary key didefinisikan untuk menjaga integritas entitas pada suatu relasi. Pada basis data temporal, *primary key* menjaga suatu relasi agar menyimpan informasi yang konsisten di setiap titik waktu.

Contoh relasi bitemporal yang memenuhi integritas entitas dapat dilihat pada Gambar III-10. Sedangkan relasi bitemporal yang tidak memenuhi integritas entitas dapat dilihat pada Gambar III-11. Relasi bitemporal pada kedua gambar tersebut mempunyai atribut *key* "nip".

nip	gaji	vs	ve	ts	te
10030	2500000	1/1/07	31/5/07	1/1/07	UC
10030	3000000	1/6/07	30/6/07	1/6/07	UC

Gambar III-10 Contoh relasi bitemporal yang memenuhi integritas data

nip	gaji	vs	ve	ts	te
10030	2500000	1/1/07	30/6/07	1/1/07	UC
10030	3000000	1/6/07	30/6/07	1/6/07	UC

Gambar III-11 Contoh I relasi bitemporal yang tidak memenuhi integritas data

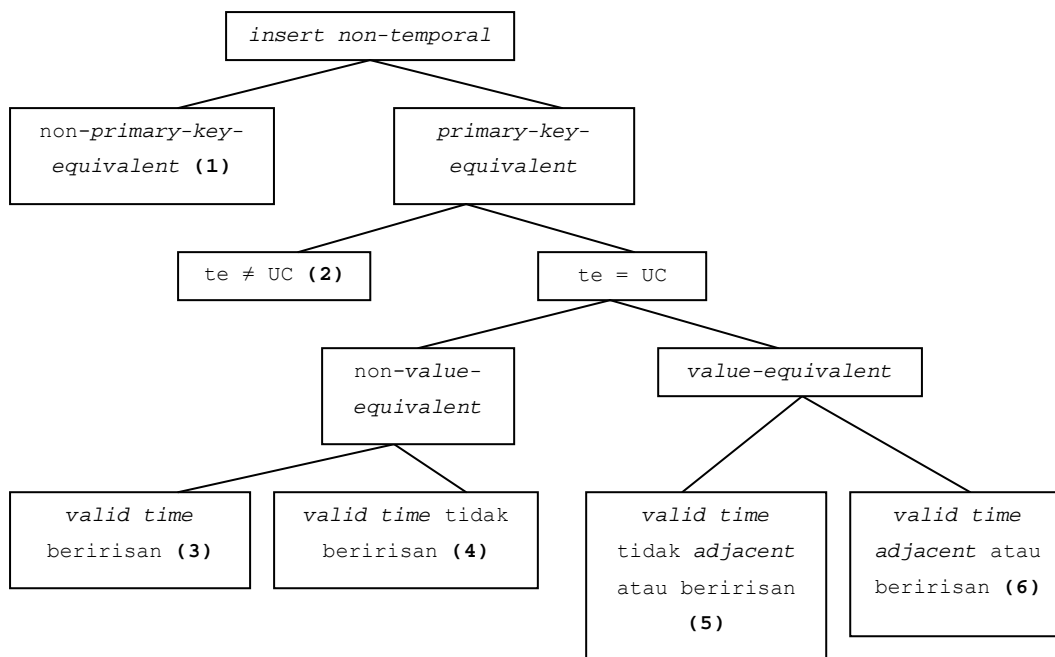
Pada Gambar III-10, dalam masa *transaction time* yang sama, yaitu 1 Juni 2007 sampai saat ini, terdapat dua *tuple* dengan nilai “nip” yang sama dan nilai gaji yang berbeda. Ini tidak menjadi masalah karena nilai *valid time* dari kedua *tuple* tersebut tidak beririsan, sehingga untuk setiap *chronon* dalam *valid time* hanya terdapat satu *tuple* dengan nilai nip ‘10030’. Relasi pada Gambar III-11 tidak memenuhi integritas data karena pada *transaction time* yang sama, terdapat dua *tuple* dengan nilai atribut *key* yang sama dan *valid time* yang beririsan. Pada masa keberlakuan *valid time* 1 Juni 2007 sampai 30 Juni 2007, terdapat dua nilai gaji bagi nip ‘10031’. Ini berarti informasi mengenai gaji seseorang dengan nip ‘10031’ tidak konsisten.

3.3.1.2 Kasus Integritas Entitas

Pelanggaran terhadap integritas entitas dapat terjadi pada operasi yang memasukkan *tuple* dengan nilai baru, yaitu *insert* dan *update*. Pada relasi bitemporal, operasi *update* tidak dilakukan seperti *update* pada relasi relasional, tetapi dilakukan dengan mengakhiri keberlakuan *tuple* lama dan melakukan *insert tuple* dengan nilai yang baru. Karena itu, penanganan integritas data dilakukan saat terjadi operasi *insert* non-temporal pada relasi, baik pada operasi *insert* temporal atau *update* temporal. Jika nilai *tuple-tuple* yang dimasukkan pada operasi *insert* atau *update* temporal melanggar integritas entitas, operasi temporal tersebut tidak jadi dilakukan. Hal ini dapat ditangani dengan mendefinisikan *trigger* pada kondisi *before insert* untuk relasi yang akan ditangani. Fungsi pada *trigger* ini memeriksa setiap *tuple* yang akan dimasukkan ke dalam relasi. Implementasi *trigger* dan fungsi ini dibahas pada Subbab 3.3.3.

Pada operasi *insert* non-temporal ini juga perlu dilakukan proses *coalescing*, yaitu penggabungan *tuple-tuple* yang *value-equivalent* dengan nilai waktu yang *adjacent* (mempunyai rentang yang berurutan tanpa ada *event* di antaranya) atau *overlaps* menjadi satu *tuple* yang mempunyai nilai waktu yang mencakup seluruh nilai waktu tersebut. Ini dilakukan agar konsep *referential integrity* pada relasi bitemporal tetap terjaga saat operasi temporal dilakukan. Sebagai contoh, jika relasi R1 mempunyai *tuple* yang mengacu relasi R2, dan pada R2 terdapat dua *tuple value-equivalent* dengan *valid time* berurutan, *tuple* pada R1 hanya dapat mengacu salah satu *tuple*. Padahal, secara konseptual, keberlakuan satu fakta yang dicakup oleh dua *tuple* pada R2 tersebut mempunyai rentang *valid time* yang lebih luas. *Tuple* di R1 seharusnya dapat mengacu *tuple* di R2 jika *tuple* pada R1 tercakup dalam rentang *valid time* tersebut.

Kondisi yang mungkin terjadi saat *tuple* baru dimasukkan dapat dilihat pada Gambar III-12.



Gambar III-12 Kemungkinan yang terjadi saat *insert tuple* dilakukan

Penjelasan kasus pada Gambar III-12 adalah sebagai berikut:

- (1) Semua *tuple* dalam relasi tidak ada yang mempunyai nilai *primary key* sama dengan *tuple* yang dimasukkan.
- (2) Ada *tuple* yang mempunyai nilai *primary key* yang sama dengan *tuple* yang dimasukkan, tetapi *tuple* sudah tidak berlaku di basis data sehingga nilai *te* pada *tuple* \neq UC.
- (3) Ada *tuple* yang *primary key equivalent* tetapi tidak *value-equivalent*, masih berlaku di basis data, dan *valid time* beririsan.
- (4) Ada *tuple* yang *primary key equivalent* tetapi tidak *value-equivalent*, masih berlaku di basis data, dan *valid time* tidak beririsan.
- (5) Ada *tuple* yang *value-equivalent* dengan *tuple* yang dimasukkan, masih berlaku di basis data, dan *valid time adjacent* atau beririsan.
- (6) Ada *tuple* yang *value-equivalent* dengan *tuple* yang dimasukkan, masih berlaku di basis data, dan *valid time* tidak *adjacent* atau beririsan.

Tuple yang akan menyebabkan pelanggaran integritas entitas adalah *tuple* yang mempunyai kondisi (3), sehingga *tuple* ini tidak dapat diterima. *Tuple* yang mempunyai kondisi (1), (2), (4), dan (5) dapat langsung diterima. Sedangkan *tuple* dengan kondisi (6) diterima dengan melakukan proses *coalescing*. *Tuple* yang *value-equivalent* dan mempunyai nilai *valid time adjacent* atau beririsan dengan *valid time* pada *tuple* yang dimasukkan akan diakhiri keberlakuannya di basis data. *Tuple* yang dimasukkan akan mempunyai nilai *valid time* awal

atau akhir yang baru, yang mencakup keberlakuan *valid time tuple value-equivalent* yang telah ada di basis data.

3.3.2 Foreign Key

3.3.2.1 Konsep Foreign Key Temporal

Foreign key didefinisikan untuk menjaga *referential integrity* pada relasi. Pada basis data temporal, *foreign key* menjaga agar untuk setiap *tuple* t_1 yang diacu oleh *tuple* t_2 , keberlakuan *valid time* t_1 contains *valid time* t_2 .

Contoh relasi bitemporal yang tidak memenuhi *referential integrity* dapat dilihat pada Gambar III-13. Relasi (a) mempunyai atribut *nip* sebagai *key* dan atribut *id_dept* yang mengacu pada nilai *id_dept* pada relasi (b). Relasi (b) mempunyai atribut *id_dept* sebagai *key*. Nilai *id_dept* pada relasi (a) tidak sama keberlakuannya dengan nilai *id_dept* pada relasi yang (b). Pada relasi yang (b), *tuple* dengan nilai *id_dept* = '1' telah berakhir keberlakuannya di basis data pada tanggal 31 Desember 2006, sehingga nilai *id_dept* = '1' seharusnya tidak dapat dimasukkan pada relasi (a). Ini mengakibatkan nilai "id_dept" pada relasi (a) tidak valid.

(a)	nip	id_dept	vs	ve	ts	te
	10030	1	1/1/07	31/5/07	1/1/07	UC

(b)	id_dept	nama_dept	vs	ve	ts	te
	1	Produksi	1/1/06	now	1/1/06	31/12/06
	2	Keuangan	1/1/06	now	1/1/06	UC

Gambar III-13 Contoh II relasi bitemporal yang tidak memenuhi integritas data

Contoh lain relasi bitemporal yang tidak memenuhi integritas data dapat dilihat pada Gambar III-14. *Tuple* pada relasi (a) dan *tuple* yang diacu pada relasi (b) berlaku di basis data pada setiap waktu keberlakuan *transaction time tuple* di relasi (a). Dengan kata lain, nilai *transaction time* pada *tuple* di relasi (b) mencakup nilai *transaction time tuple* di relasi (a). Akan tetapi, *tuple* di relasi (a) dan *tuple* di relasi (b) tidak berlaku secara bersamaan dalam setiap titik *valid time tuple* pada relasi (a). *Valid time tuple* pada relasi (b) bernilai 1 Januari 2006 sampai dengan 31 Mei 2007. Nilai ini tidak mencakup nilai *valid time tuple* pada relasi (a), yaitu 1 Januari 2007 sampai dengan 31 Desember 2007. Untuk keberlakuan *valid time* 1 Juni 2007 sampai dengan 31 Desember 2007, nilai *id_dept* = '1' pada *tuple* di relasi (a) tidak mempunyai acuan di basis data.

(a)	nip	id_dept	vs	ve	ts	te
	10030	1	1/1/07	31/12/07	1/1/07	UC

(b)	id_dept	nama_dept	vs	ve	ts	te
	1	Produksi	1/1/06	31/5/07	1/1/06	UC

Gambar III-14 Contoh III relasi bitemporal yang tidak memenuhi integritas data

3.3.2.2 Kasus Referential Integrity

Penanganan *referential integrity* pada basis data temporal perlu dilakukan pada dua kondisi:

1. Operasi *insert* atau *update* temporal yang dilakukan terhadap relasi yang memiliki atribut *foreign key*.

Pada operasi temporal ini dilakukan *insert tuple* pada relasi. Pelanggaran terjadi jika *tuple* yang dimasukkan tidak mempunyai *tuple* acuan di relasi lain karena nilai atribut *foreign key* yang dimasukkan tidak valid atau nilai *valid time* tidak tercakup.

2. Operasi *delete* atau *update* temporal yang dilakukan terhadap relasi yang diacu oleh *constraint foreign key* relasi lain.

Pada operasi ini, nilai suatu *tuple* dapat berubah dan keberlakuan *tuple* di basis data dapat diakhiri. Ini mengakibatkan timbulnya kemungkinan bahwa *tuple* di relasi lain tidak lagi mempunyai *tuple* acuan di relasi tersebut.

Penanganan untuk kedua kondisi ini dapat diimplementasikan dengan fungsi di DBMS yang memeriksa kondisi relasi setelah *query* konversi dilakukan. Fungsi ini akan dijelaskan pada Subbab 3.3.3.

3.3.3 Implementasi Fungsi dan Tabel Sistem

Berdasarkan analisis pada Subbab 3.3.1.2 dan Subbab 3.3.2.2, penanganan *primary key* dan *foreign key* temporal akan diimplementasikan dalam dua jenis fungsi, yaitu:

1. Fungsi untuk penanganan *primary key* dan *coalescing*. Fungsi ini spesifik terhadap tiap relasi dan akan dieksekusi oleh *trigger* pada kondisi *before insert*. Jika *primary key* tidak didefinisikan pada relasi, fungsi ini tetap dibuat untuk melakukan penanganan *coalescing*.
2. Fungsi untuk penanganan *foreign key*. Fungsi ini akan dieksekusi setelah operasi *insert* atau *update* temporal pada relasi yang mempunyai atribut *foreign key* dan setelah operasi *delete* atau *update* temporal pada relasi yang menjadi acuan pada *constraint foreign key*. Jika relasi terkait dengan beberapa *constraint foreign key*, beberapa fungsi akan dieksekusi. Masing-masing fungsi melakukan pengecekan untuk sebuah *constraint foreign key*.

Fungsi yang dieksekusi pada *trigger before insert* untuk tiap relasi dapat dilihat pada Gambar III-15. Fungsi ini ditulis dalam sintaks pl/pgsql. Pada fungsi tersebut dilakukan langkah-langkah sebagai berikut:

1. Pengecekan untuk kasus 'now' seperti pada Subbab 3.1.1. Jika nilai *valid time* akhir pada *tuple* yang akan dimasukkan adalah 'now', pada relasi tidak boleh ada *tuple* yang *primary key equivalent* dengan nilai *valid time* akhir lebih dari *current date*. Jika nilai *valid time* akhir pada *tuple* yang akan dimasukkan lebih dari *current date*, pada relasi tidak boleh ada *tuple* yang *primary key equivalent* dengan nilai *valid time* akhir 'now'. Pengecekan ini tidak akan dilakukan jika tidak ada *primary key* pada relasi.
2. Pengecekan *primary key*. Jika ada *tuple* dalam relasi yang *primary key equivalent* dengan *tuple* yang akan dimasukkan, tetapi tidak *value-equivalent* dan mempunyai *valid time* yang beririsan, *insert tuple* baru akan melanggar *constraint primary key*. Pengecekan ini tidak akan dilakukan jika *primary key* mencakup semua atribut *non-timestamp*.
3. Pengecekan untuk proses *coalescing* di basis data. Jika ada *tuple-tuple* yang *value-equivalent* dengan *tuple* yang akan dimasukkan dan mempunyai nilai *valid time* berurutan atau beririsan, *tuple-tuple* tersebut diakhiri keberlakuannya di basis data. Pada *tuple* yang akan dimasukkan, nilai *valid time* diperbaharui sehingga mencakup nilai *valid time tuple-tuple* yang *value equivalent* tersebut.

Pada Gambar III-15, <table_name> adalah nama relasi yang akan menerima *tuple* baru, <pkey> adalah nama atribut *primary key*, <nonpkey> adalah atribut *non-timestamp* yang bukan *primary key*, dan <attr> adalah atribut *non-timestamp* pada relasi. Nilai-nilai ini didapatkan dari *query create table*. *Exception* yang dihasilkan pada fungsi ini mengindikasikan pelanggaran *constraint* sehingga operasi temporal tidak dapat dilakukan.

```

DECLARE max_ve DATE;
        min_vs DATE;
        i int;
BEGIN

/* primary key tidak boleh null */
IF (new.<pkey> is null [OR ...])
THEN RAISE EXCEPTION 'primary key violation';
END IF;

/* pengecekan kasus 'now' */
IF (new.ve is null)
THEN
BEGIN

```

```

SELECT INTO i count(*) FROM <table_name>
WHERE <pkey> = new.<pkey> [AND ...]
      AND te is null AND ve > CURRENT_DATE;
IF (i > 0)
THEN RAISE EXCEPTION 'possible primary key violation in the
future';
END IF;
END;
END IF;

IF (new.ve > CURRENT_DATE)
THEN
BEGIN
SELECT INTO i count(*) FROM <table_name>
WHERE <pkey> = new.<pkey> [AND ...]
      AND te is null AND ve is null;
IF (i > 0)
THEN RAISE EXCEPTION 'possible primary key violation in the
future';
END IF;
END;
END IF;

/* cek primary key */
SELECT INTO i count(*) FROM <table_name>
WHERE <pkey> = new.<pkey> [AND ...]
      AND (<nonpkey> <> new.<nonpkey> [OR ...]
      AND te is null
      AND (sys_overlaps(new.vs,dbtoactual(new.ve),vs,dbtoactual(ve)));
IF (i > 0)
THEN RAISE EXCEPTION 'primary key violation';
END IF;

/* menggabungkan tuple value-equivalent yang adjacent atau overlaps
dengan tuple baru */
SELECT INTO min_vs, max_ve min(vs), max(dbtoactual(ve))
FROM <table_name>
WHERE <attr> = new.<attr> [AND ...]
      AND te is null

```

```

        AND (sys_overlaps(new.vs, dbtoactual(new.ve), vs,
        dbtoactual(ve)) OR
            sys_meets(new.vs, dbtoactual(new.ve), vs, dbtoactual(ve)) OR
            sys_meets(vs, dbtoactual(ve), new.vs, dbtoactual(new.ve)));
IF (min_vs is not null)
THEN
BEGIN
    SELECT INTO i count(*) FROM <table_name>
    WHERE <attr> = new.<attr> [AND ...]
        AND te is null
        AND (sys_overlaps(new.vs, dbtoactual(new.ve), vs,
            dbtoactual(ve)) OR
            sys_meets(new.vs, dbtoactual(new.ve), vs,
            dbtoactual(ve)))
        AND ve is null;
    IF (i > 0) THEN max_ve = null; END IF;

    IF (min_vs < new.vs) THEN new.vs = min_vs; END IF;
    IF ((max_ve is null) OR (max_ve > new.ve))
    THEN new.ve = max_ve;
    END IF;

    UPDATE <table_name> SET te = CURRENT_DATE - INTERVAL '1 DAY'
    WHERE <attr> = new.<attr> [AND ...]
    AND te is null AND ts < CURRENT_DATE
    AND (sys_overlaps(new.vs, dbtoactual(new.ve), vs,
        dbtoactual(ve)) OR
        sys_meets(new.vs, dbtoactual(new.ve), vs, dbtoactual(ve)) OR
        sys_meets(vs, dbtoactual(ve), new.vs, dbtoactual(new.ve)));

    DELETE FROM <table_name>
    WHERE <attr> = new.<attr> [AND ...]
    AND te is null AND ts = CURRENT_DATE
    AND (sys_overlaps(new.vs, dbtoactual(new.ve), vs,
        dbtoactual(ve)) OR
        sys_meets(new.vs, dbtoactual(new.ve), vs, dbtoactual(ve)) OR
        sys_meets(vs, dbtoactual(ve), new.vs, dbtoactual(new.ve)));
END;
END IF;

```

```

/* return tuple baru */
    RETURN new;
END;

```

Gambar III-15 Fungsi yang digunakan untuk *trigger before insert*

Fungsi untuk penanganan *constraint foreign key* dapat dilihat pada Gambar III-16. Fungsi ini memeriksa jika pada relasi yang mempunyai atribut *foreign key* terdapat *tuple* yang tidak mempunyai *tuple* acuan pada relasi lain, atau terdapat *tuple* dengan kasus 'now' seperti pada Subbab 3.1.1. Relasi yang mengacu adalah <table_name>, atribut *foreign key* adalah <fkey>, relasi yang diacu adalah <ref_table_name>, dan atribut acuan adalah <ref_column_name>.

```

DECLARE i int;
BEGIN
    SELECT INTO i count(*) FROM <table_name> r
    WHERE te is null AND NOT EXISTS (
        SELECT * FROM <ref_table_name> s
        WHERE r.<fkey> = s.<ref_column_name>
            AND s.te is null
            AND sys_contains(s.vs,dbtoactual(s.ve),r.vs,
                dbtoactual(r.ve))
            AND (r.ve is not null OR
                (r.ve is null AND (s.ve is null OR
                    s.ve='99991231'))
        );
    IF (i <> 0)
    THEN RAISE EXCEPTION 'foreign key violation';
    END IF;
END;

```

Gambar III-16 Fungsi untuk penanganan *constraint foreign key*

Untuk pemeliharaan *constraint* dan fungsi ini, akan dibuat tabel sistem *sys_table*, *sys_pkey* dan *sys_fkey* yang menyimpan data mengenai *primary key*, *foreign key*, dan fungsi yang menanganinya. Tabel sistem *sys_table* memiliki atribut sebagai berikut:

1. *table_name*: memuat nama relasi
2. *column_name*: memuat nama semua atribut
3. *function_name*: memuat nama fungsi yang didefinisikan dalam *trigger before insert* pada relasi yaitu fungsi pada Gambar III-15

Tabel sistem `sys_pkey` memiliki atribut sebagai berikut:

1. `pkey_name`: memuat nama *constraint primary key*
2. `table_name`: memuat nama relasi
3. `column_name`: memuat nama semua atribut *primary key* dalam relasi

Jika *primary key* bersifat komposit, nilai yang dimasukkan dalam tabel adalah nilai semua atribut yang merupakan *primary key* dan dipisahkan dengan sebuah *delimiter*. Misalkan *primary key* adalah {`id_anggota,id_buku`}, nilai atribut `column_name` pada tabel `sys_pkey` yang dimasukkan adalah '`id_anggota,id_buku`'.

Tabel sistem `sys_fkey` yang memiliki atribut sebagai berikut:

1. `fkey_name`: memuat nama *constraint foreign key*
2. `table_name`: memuat nama relasi
3. `column_name`: memuat nama atribut *foreign key* dalam relasi
4. `ref_table_name`: memuat nama relasi yang diacu
5. `ref_column_name`: memuat nama atribut yang nilainya diacu oleh `column_name`
6. `function_name`: memuat nama fungsi yang melakukan penanganan *foreign key*, yaitu fungsi pada Gambar III-16.

Nilai-nilai pada tabel sistem `sys_table`, `sys_pkey` dan `sys_fkey` dimasukkan saat operasi *create table*. Nilai pada `sys_pkey` dan `sys_fkey` diketahui dari pernyataan *primary key* dan *foreign key* pada *column constraint* atau *table constraint*. Contoh *query create table* yang mengandung pernyataan *primary key* dan *foreign key* dapat dilihat pada Gambar III-17.

```
CREATE TABLE pegawai (
    nip    char(5)    PRIMARY KEY,
    nama   varchar(30) NOT NULL,
    gaji   integer    DEFAULT 0,
    id_dept integer,
    id_jabatan integer
    CONSTRAINT fkey1 FOREIGN KEY (id_dept) REFERENCES
    departemen(id)
    CONSTRAINT fkey2 FOREIGN KEY (id_jabatan) REFERENCES
    jabatan(id)
) AS VALID AND TRANSACTION
```

Gambar III-17 Contoh *query create table* dengan *primary key* dan *foreign key*

Nilai yang dimasukkan pada tabel sistem berdasarkan *query* pada Gambar III-17 adalah seperti pada Gambar III-18. Jika *primary key* atau *foreign key* dinyatakan pada *column*

constraint, nama *constraint* akan ditentukan secara otomatis. Nama fungsi untuk *trigger before insert* fungsi untuk penanganan tiap *foreign key* ditentukan secara otomatis.

table_name	column_name	function_name
pegawai	nip,nama,gaji,id_dept	func_pegawai()

pkey_name	table_name	column_name
pkey_pegawai	pegawai	nip

fkey_name	table_name	column_name	ref_table_name	ref_column_name	function_name
fkey1	pegawai	id_dept	departemen	id	func_fkey1()
fkey2	pegawai	id_jabatan	jabatan	id	func_fkey2()

Gambar III-18 Contoh nilai yang dimasukkan pada tabel sistem

Penerapan cara penanganan *primary key* dan *foreign key* yang spesifik untuk setiap operasi dapat dilihat pada Subbab 3.4.

3.4 Konversi Query

Proses konversi *query* dimulai dengan melihat *grammar query* temporal yang merupakan aturan semantik *query* yang akan diterima. Lewat *grammar*, dapat dilihat bahwa suatu *query* termasuk dalam operasi tertentu, seperti *create table*, *insert*, *delete*, *update*, dan *select*. Nilai-nilai penting pada *query* seperti nama tabel, nama kolom, dan kondisi juga dapat diidentifikasi dari *grammar*. Setelah semua nilai yang penting didapatkan, akan disusun *query* non-temporal yang bersesuaian. Perlu dilakukan pengecekan terhadap integritas entitas dan *referential integrity* berdasarkan konsep *primary key* dan *foreign key* temporal seperti pada subbab 3.3. Jika operasi tersebut tidak akan mengakibatkan pelanggaran terhadap konsep *primary key* dan *foreign key* temporal, operasi dilakukan terhadap basis data.

3.4.1 Create Table

Sintaks *query* non-temporal yang bersesuaian dengan *query* temporal untuk *create table* pada Gambar II-8 dapat dilihat pada Gambar III-19. Model representasi relasi bitemporal pada basis data relasional dibuat dengan menambahkan atribut vs, ve, ts, dan te untuk menyimpan nilai dimensi waktu.

```

CREATE TABLE table_name (
  column_name type_name [<addition>]
  [, ...]
  , vs date
  , ve date
  , ts date
  , te date
);

```

Gambar III-19 Query konversi untuk operasi *create table*

Pada *query* hasil konversi, *constraint* seperti *primary key* dan *foreign key* tidak disertakan dalam *query create table*. Untuk penanganan *primary key* dan *foreign key*, dibuat *trigger* dan fungsi seperti pada Subbab 3.3.3. Informasi *primary key* dan *foreign key* untuk penanganan integritas data disimpan dalam tabel sistem *sys_table*, *sys_pkey*, dan *sys_fkey*. *Query* untuk menciptakan fungsi dan *trigger before insert* untuk penanganan integritas entitas dan *referential integrity* sebelum *tuple* dimasukkan dapat dilihat pada Gambar III-20. Fungsi untuk pengecekan *referential integrity* setelah operasi dilakukan dapat dilihat pada Gambar III-21. Sedangkan *query* untuk memasukkan data mengenai tabel, *primary key* dan *foreign key* dapat dilihat pada Gambar III-22 dan Gambar III-23.

```

CREATE OR REPLACE FUNCTION func_table_name()
RETURNS "trigger" AS
$BODY$<function1_definition>$BODY$
LANGUAGE 'plpgsql' VOLATILE;

CREATE TRIGGER trig_before_insert_table_name
BEFORE INSERT ON table_name
FOR EACH ROW
EXECUTE PROCEDURE func_table_name;

```

Gambar III-20 Query untuk menciptakan fungsi dan *trigger before insert*

```

CREATE OR REPLACE FUNCTION func_constraint_name()
RETURNS void AS
$BODY$<function2_definition>$BODY$
LANGUAGE 'plpgsql' VOLATILE;

```

Gambar III-21 Query untuk menciptakan fungsi penanganan *foreign key*

<*function1_definition*> pada Gambar III-20 adalah fungsi yang didefinisikan pada Gambar III-15. Sedangkan <*function2_definition*> pada Gambar III-21 adalah fungsi yang didefinisikan pada Gambar III-16. *Query* untuk menciptakan fungsi penanganan *constraint foreign key* dilakukan untuk setiap *constraint foreign key* pada *query create table*.

```

INSERT INTO sys_table(table_name,column_name,function_name)
VALUES('table_name','column_name [,column_name]','function_name');

INSERT INTO sys_pkey(pkey_name,table_name,column_name)
VALUES('constraint_name','table_name','pkey_column_name
[,pkey_column_name]');

```

Gambar III-22 Query untuk memasukkan data tabel dan *primary key*

```

INSERT INTO sys_fkey(fkey_name,table_name,column_name,ref_table_name,
ref_column_name,function_name)
VALUES('constraint_name','table_name','column_name','ref_table_name',
'ref_column_name','func_constraint_name');

```

Gambar III-23 Query untuk memasukkan data *foreign key*

Query untuk memasukkan data tabel dan *primary key* hanya dilakukan sekali, sedangkan *query* untuk memasukkan *foreign key* dilakukan untuk setiap *constraint foreign key* yang dinyatakan dalam *query create table*. Sebelum *query* ini dilakukan, perlu dilakukan pengecekan agar nama kolom atau tabel yang dinyatakan pada *table constraint* benar-benar terdapat pada basis data. Jika nama kolom atau tabel pada *table constraint* tidak valid, relasi baru yang didefinisikan pada *query* tidak akan dibuat.

3.4.2 *Insert*

Insert pada model representasi relasi bitemporal dilakukan dengan memasukkan nilai tambahan dalam atribut *vs*, *ve*, *ts*, dan *te*. Nilai *vs* dan *ve* didapatkan dari klausa *VALID*, sedangkan nilai *ts* dan *te* ditangani secara otomatis berdasarkan waktu pada sistem. Jika klausa *VALID* tidak disertakan, nilai *valid time* bagi *query* tersebut adalah *PERIOD* '[current_date,now]'. Sintaks *query* non-temporal yang bersesuaian dengan *query* temporal untuk *insert* pada Gambar II-10 dapat dilihat pada Gambar III-24. Fungsi *querytodb(string)* digunakan agar nilai *vs* dan *ve* dalam *query* dimasukkan ke basis data sesuai dengan nilai konversinya. Misalnya jika nilai pada *query* adalah *PERIOD* '[beginning,now]', nilai *vs* dan *ve* yang dimasukkan ke basis data adalah '01-01-0001' dan *null*.

```

INSERT INTO table_name [ ( <column_list> ,vs,ve,ts,te) ]
VALUES ( <value_list> , querytodb(v_start), querytodb(v_end),
CURRENT_DATE, null);
[ SELECT function_name; [ ... ] ]

```

Gambar III-24 Query konversi untuk operasi *insert*

Integritas data pada operasi *insert* temporal telah ditangani oleh fungsi yang dieksekusi pada *trigger before insert* seperti pada Gambar III-15. Operasi `SELECT function_name` seperti pada Gambar III-24 dilakukan jika pada relasi terdapat atribut *foreign key*. Selain itu, Pada operasi *insert*, terdapat beberapa kasus terkait dengan validitas nilai data. Sebelum *query* diterima, perlu ada pengecekan yang berhubungan dengan hal ini. Penanganan validitas nilai data dilakukan dengan memastikan bahwa *query* tidak melakukan *assign* terhadap nilai *vs*, *ve*, *ts*, dan *te* secara langsung. Selain itu, nilai *v_end* harus lebih besar atau sama dengan *v_start*.

3.4.3 Delete

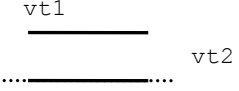
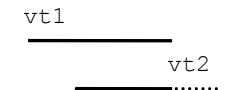
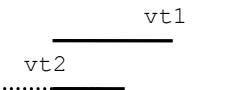
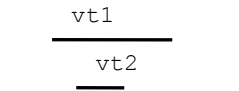
3.4.3.1 Kasus Delete

Operasi *delete* pada relasi bitemporal dilakukan terhadap *tuple* yang memenuhi kondisi dan berlangsung dalam periode *valid time* yang disebutkan dalam *query*. Jika *delete* dilakukan terhadap *tuple* yang baru berlaku kurang dari satu *chronon*, *tuple* tersebut dihapus dari basis data. Sedangkan jika *tuple* sudah berlaku satu *chronon* atau lebih, *tuple* tersebut tidak dihapus, tetapi keberlakuannya di basis data dihentikan dengan dilakukannya *update* terhadap nilai *te* menjadi *current date*-1 satuan *chronon*. Jika nilai *valid time* tidak beririsan sama sekali, *tuple* tidak akan terpengaruh oleh operasi *delete*. Jika *delete* hanya menghapus keberlakuan *tuple* pada periode tertentu, selain dilakukan *update* terhadap nilai *te* dalam *tuple*, *tuple* baru juga dimasukkan. *Tuple* ini bernilai sama dengan *tuple* yang lama, tetapi mempunyai nilai *valid time* yang baru.

Ada beberapa kemungkinan *query* non-temporal yang bersesuaian terkait dengan nilai *valid time*. Perbandingan nilai *valid time* dari kemungkinan-kemungkinan tersebut dapat dilihat pada Gambar III-25. Misalkan nilai *valid time* pada *tuple* adalah *vt1* dan nilai *valid time* pada *query* adalah *vt2*, maka kemungkinan yang terjadi:

- (1) *vt2* sama dengan *vt1* atau *vt2* mengandung *vt1*. Keberlakuan *valid time* pada *tuple* dihilangkan semuanya, sehingga hanya akan dilakukan *update* terhadap nilai *te* *tuple* yang di-*delete*. Tidak ada *tuple* baru yang di-*insert*.
- (2) *vt2* beririsan dengan bagian akhir dari *vt1*. Ini mengubah nilai *valid time* akhir dari *tuple* tersebut. Setelah *update* nilai *te* dilakukan, dimasukkan *tuple* baru dengan nilai $vs \leftarrow vt1_start$ dan $ve \leftarrow vt2_start - 1$ satuan *chronon*.
- (3) *vt2* beririsan dengan bagian awal dari *vt1*. Ini mengubah nilai *valid time* awal dari *tuple* tersebut. Setelah *update* nilai *te* dilakukan, dimasukkan *tuple* baru dengan nilai $vs \leftarrow vt2_end + 1$ satuan *chronon* dan $ve \leftarrow vt1_end$.

(4) vt1 mengandung vt2. Artinya, interval vt2 berada di dalam interval vt1. Setelah *update* nilai te dimasukkan, dimasukkan dua buah *tuple* baru. *Tuple* pertama mempunyai nilai $vs \leftarrow vt1_start$ dan $ve \leftarrow vt2_start-1$ satuan *chronon*. *Tuple* kedua mempunyai nilai $vs \leftarrow vt2_end+1$ satuan *chronon* dan $ve \leftarrow vt1_end$.

		<u>kondisi valid time</u>	<u>nilai valid time baru</u>
(1)		$vt1_start \geq vt2_start$ $vt1_end \leq vt2_end$	tidak ada
(2)		$vt1_start < vt2_start$ $vt1_end \leq vt2_end$	[vt1_start, vt2_start-1]
(3)		$vt1_start \geq vt2_start$ $vt1_end > vt2_end$	[vt2_end+1, vt1_end]
(4)		$vt1_start < vt2_start$ $vt1_end > vt2_end$	[vt1_start, vt2_start-1] [vt2_end+1, vt1_end]

Gambar III-25 Kemungkinan nilai *valid time* pada operasi *delete*

Berdasarkan kondisi *valid time* dan nilai *valid time* baru pada Gambar III-25, kondisi yang akan ditangani dapat disederhanakan sebagai berikut:

1. Jika $vt2_start$ berada di dalam rentang waktu vt1 dan $vt2_start$ tidak sama dengan $vt1_start$, akan dihasilkan *tuple* baru dengan nilai *valid time* [vt1_start, vt2_start-1].
2. Jika $vt2_end$ berada di dalam rentang waktu vt1 dan $vt2_end$ tidak sama dengan $vt1_end$, akan dihasilkan *tuple* baru dengan nilai *valid time* [vt2_end+1, vt1_end].

3.4.3.2 Query Non-Temporal

Berdasarkan *query* temporal pada Gambar II-13, *query* non-temporal yang bersesuaian dapat dilihat pada Gambar III-26. Nilai $\langle converted_condition \rangle$ didapatkan dari klausa kondisi yang telah dikonversi seperti pada Gambar III-6. Pada operasi *delete*, pengecekan *primary key* tidak perlu dilakukan karena seharusnya sudah ditangani saat operasi *insert*. Operasi *delete* tidak akan menambahkan *tuple* baru dengan nilai *valid time* yang beririsan dengan *tuple* lainnya yang masih berlaku di basis data. Akan tetapi, perlu ada pengecekan terkait

referential integrity jika *tuple* yang dihapus merupakan relasi yang diacu oleh *foreign key* pada relasi lain.

<pre>(1) CREATE TABLE temptable1 AS SELECT * FROM table_name WHERE te is null AND dbtoactual(vs) < querytoactual(v_start) AND dbtoactual(ve) >= querytoactual(v_start) [AND <converted_condition>]; UPDATE temptable1 SET ts = CURRENT_DATE, ve = querytoactual(v_start) - INTERVAL '1 DAY';</pre>
<pre>(2) CREATE TABLE temptable2 AS SELECT * FROM table_name WHERE te is null AND dbtoactual(vs) <= querytoactual(v_end) AND dbtoactual(ve) > querytoactual(v_end) [AND <converted_condition>]; UPDATE temptable2 SET ts = CURRENT_DATE, vs = querytoactual(v_end) + INTERVAL '1 DAY';</pre>
<pre>(3) DELETE FROM table_name WHERE ts = CURRENT_DATE AND te is null AND sys_overlaps(dbtoactual(vs), dbtoactual(ve), querytoactual(v_start), querytoactual(v_end)) [AND <converted_condition>]; UPDATE table_name SET te = CURRENT_DATE - INTERVAL '1 DAY' WHERE ts < CURRENT_DATE AND te is null AND sys_overlaps(dbtoactual(vs), dbtoactual(ve), querytoactual(v_start), querytoactual(v_end)) [AND <converted_condition>];</pre>
<pre>(4) INSERT INTO table_name SELECT * FROM temptable1; INSERT INTO table_name SELECT * FROM temptable2; DROP TABLE temptable1; DROP TABLE temptable2;</pre>
<pre>(5) [SELECT function_name; [...]]</pre>

Gambar III-26 Query konversi untuk operasi delete

Penjelasan untuk *query* pada Gambar III-26 adalah sebagai berikut:

- (1) Menciptakan relasi sementara temptable1 untuk menampung *tuple* yang memenuhi kondisi *delete* temporal dan akan menghasilkan *tuple* baru dengan nilai *ve* yang baru.
- (2) Menciptakan relasi sementara temptable2 untuk menampung *tuple* yang memenuhi kondisi *delete* temporal dan akan menghasilkan *tuple* baru dengan nilai *vs* yang baru.
- (3) Mengakhiri keberlakuan *tuple* di basis data untuk *tuple-tuple* yang memenuhi kondisi *delete* temporal. Jika *tuple* mempunyai keberlakuan *transaction time* kurang dari satu

chronon, *tuple* tersebut akan di-*delete*. Jika keberlakuannya satu *chronon* atau lebih, nilai *te tuple* tersebut di-*update*.

- (4) Memasukkan *tuple-tuple* baru dari *temptable1* dan *temptable2* ke dalam relasi.
- (5) Jika nama relasi *table_name* yang dikenakan operasi *delete* temporal terdapat pada nilai atribut *ref_table_name* di tabel sistem *sys_fkey*, fungsi pada atribut *function_name* dieksekusi untuk pengecekan *referential integrity*. Fungsi ini adalah fungsi yang didefinisikan pada Gambar III-16. Eksekusi dilakukan untuk semua fungsi pengecekan *referential integrity* yang mengacu relasi *table_name*.

Query pada Gambar III-26 dieksekusi dalam satu transaksi sehingga jika terdapat *error* dan *exception*, kondisi relasi kembali ke awal sebelum operasi temporal dilakukan.

3.4.4 Update

Berdasarkan sintaks *query* temporal pada Gambar II-16, *query* konversi untuk operasi *update* dapat dilihat pada Gambar III-27. Klausula kondisi pada *query update* temporal ini dikonversi menjadi *<converted_condition>* seperti pada Gambar III-6. Jika klausula *VALID* tidak disertakan, tidak akan dilakukan *update* terhadap nilai *vs* dan *ve*.

(1)	CREATE TABLE temptable AS SELECT * FROM <i>table_name</i> WHERE <i>te</i> is null [AND <i><converted_condition></i>];
(2)	UPDATE temptable SET <i>ts</i> = CURRENT_DATE, <i>te</i> = null, <i><update_attr_list></i> [, <i>vs</i> = <i>v_start</i> , <i>ve</i> = <i>v_end</i>];
(3)	DELETE FROM <i>table_name</i> WHERE <i>ts</i> = CURRENT_DATE AND <i>te</i> is null [AND <i><converted_condition></i>]; UPDATE <i>table_name</i> SET <i>te</i> = CURRENT_DATE - INTERVAL '1 DAY' WHERE <i>ts</i> < CURRENT_DATE AND <i>te</i> is null [AND <i><converted_condition></i>];
(4)	INSERT INTO <i>table_name</i> SELECT * FROM temptable; DROP TABLE temptable;
(5)	[SELECT <i>function_name</i> ; [...]]

Gambar III-27 Query konversi untuk operasi *update*

Penjelasan untuk *query* pada Gambar III-27 adalah sebagai berikut:

- (1) Menciptakan relasi sementara *temptable* yang akan menampung *tuple* yang memenuhi kondisi *update* temporal.
- (2) Melakukan *update* untuk *tuple* yang memenuhi kondisi dengan nilai baru yang dimasukkan pada *query*.

- (3) Mengakhiri keberlakuan *tuple* yang nilainya akan berubah akibat operasi *update* temporal. Jika keberlakuan *transaction time*-nya kurang dari satu *chronon*, *tuple* di-*delete*. Sedangkan jika keberlakuannya satu *chronon* atau lebih, nilai te pada *tuple* di-*update*.
- (4) Memasukkan *tuple-tuple* yang mempunyai nilai baru dari temptable ke dalam relasi.
- (5) Jika relasi yang dikenakan operasi *update* temporal terdapat pada nilai atribut *table_name* atau *ref_table_name* di tabel sistem *sys_fkey*, fungsi pada atribut *function_name* dieksekusi untuk pengecekan *referential integrity*. Fungsi ini adalah fungsi yang didefinisikan pada Gambar III-16.

Pada operasi *update* temporal juga perlu dilakukan pengecekan integritas data, karena operasi ini memasukkan *tuple* baru ke dalam relasi. Integritas data untuk relasi yang dikenakan operasi *update* ini telah ditangani oleh fungsi yang dieksekusi pada *trigger before insert*. *Query* pada Gambar III-27 dieksekusi dalam satu transaksi sehingga jika terdapat *error* dan *exception*, kondisi relasi kembali ke awal sebelum operasi temporal dilakukan.

3.4.5 Select

Grammar untuk operasi *select* dapat dilihat pada Gambar III-28. Aturan produksi *<select_condition>* pada *grammar* dapat dilihat pada Gambar III-8. Konversi untuk kondisi ini dilakukan dengan aturan produksi *<converted_select_condition>* yang terdapat pada Gambar III-9. Jika pada *query* digunakan *SELECT **, atribut yang akan akan ditampilkan adalah semua atribut kecuali *vs*, *ve*, *ts*, dan *te*.

```

<select_stmt> ::= select (snapshot)? (distinct)? (* |
                    <select_attr_list>) from <select_table_list>
                    (<select_condition>)?
<select_attr_list> ::= <column_name> (as <alias>)? (, <column_name>
                    (as <alias>)?)*
<select_table_list> ::= <table_name> (<table_alias>)? (, <table_name>
                    (<table_alias>)?)*

```

Gambar III-28 Grammar query temporal untuk operasi select

Pada operasi *select*, terdapat beberapa kemungkinan *retrieval* yang dilakukan:

1. *Retrieval* relasi *snapshot*. *Retrieval* ini dideklarasikan dengan *keyword* *SNAPSHOT*. Pada operasi ini, *tuple* yang dikembalikan tidak ditampilkan informasi *valid time*-nya. Jika dilakukan *join*, operasi yang dilakukan tidak memperhitungkan kesesuaian keberlakuan *tuple*, tetapi hanya berdasarkan perbandingan nilai atribut atau dimensi waktu. Atribut

dimensi waktu diperlakukan seperti atribut eksplisit sehingga penanganan operasi lebih sederhana.

2. *Retrieval* relasi *valid time*. *Retrieval* ini dilakukan jika *keyword* *SNAPSHOT* tidak dideklarasikan pada *query*. Pada operasi ini, informasi *valid time tuple* ditampilkan pada hasil seleksi. Jika dilakukan *join*, operasi yang dilakukan adalah operasi *temporal join* yang hanya memasang *tuple* dengan keberlakuan waktu yang beririsan. *Tuple-tuple* yang dikembalikan tidak ada yang bersifat *value-equivalent*. Nilai *valid time* untuk *tuple-tuple* yang *value-equivalent* ditampilkan dalam satu *tuple*, sehingga pada operasi ini dibutuhkan penanganan untuk *coalescing*.

3.4.5.1 Retrieval Relasi Snapshot

Dengan menggunakan *keyword* *SNAPSHOT*, relasi yang ditampilkan adalah relasi *snapshot*. *Query* non-temporal yang bersesuaian dapat dilihat pada Gambar III-29. `<converted_select_condition>` adalah hasil konversi dari `<select_condition>` yang didapatkan seperti pada Gambar III-9. Jika pada *query* tidak terdapat ekspresi perbandingan *transaction time*, *tuple* yang dikembalikan adalah yang berlaku di basis data saat ini, sehingga pada klausa kondisi ditambahkan kondisi `te is null`.

```
SELECT [DISTINCT] <select_attr_list>
FROM <select_table_list>
[ WHERE <converted_select_condition> ]
```

Gambar III-29 *Query* konversi untuk operasi *select snapshot*

3.4.5.2 Retrieval Relasi Valid Time

Pada tugas akhir ini, *query select* yang diterima untuk *retrieval* relasi *valid time* hanya seleksi terhadap satu dan dua relasi. Proyeksi yang dilakukan pada operasi *select* dapat mengakibatkan kondisi *uncoalesced* pada relasi yang ditampilkan. Pada kondisi *uncoalesced*, terdapat *tuple-tuple* yang *value-equivalent* dengan nilai *valid time* yang berurutan atau beririsan [BOH97].

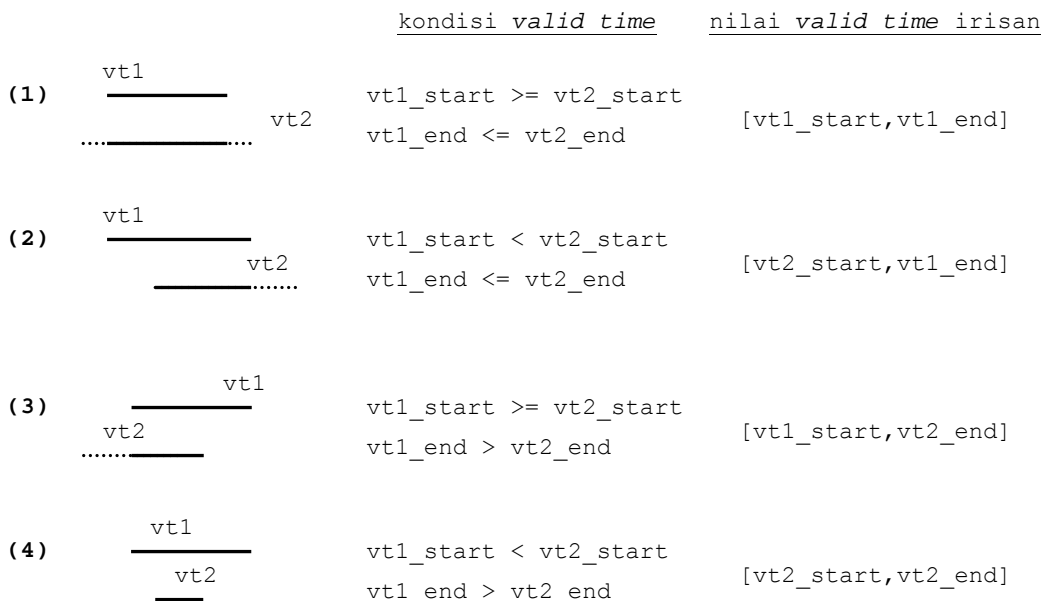
Untuk seleksi terhadap satu relasi, *query select* non-temporal tanpa *coalescing* yang bersesuaian dapat dilihat pada Gambar III-30. Sedangkan *query* non-temporal untuk seleksi terhadap dua relasi dapat dilihat pada Gambar III-32. Pada relasi bitemporal, operasi *join* dikenakan hanya pada *tuple-tuple* yang mempunyai nilai *valid time* dan *transaction time* yang *overlaps*. Interval *valid time* yang ditampilkan adalah irisan *valid time* dua *tuple*. Seperti

klausa kondisi pada *retrieval* relasi *snapshot*, `<converted_select_condition>` adalah hasil konversi dari `<select_condition>` yang didapatkan seperti pada Gambar III-9.

```
SELECT <select_attr_list> , vs , ve
FROM table_name [alias]
[ WHERE <converted_select_condition> ]
```

Gambar III-30 Query konversi untuk operasi *select* satu relasi

Pada *query* konversi *select* pada Gambar III-32, dilakukan seleksi untuk *tuple-tuple* yang mempunyai keberlakuan data yang beririsan dengan menggunakan fungsi `sys_overlaps()`. Nilai *valid time* yang dikembalikan pada *query select* ini adalah nilai *valid time* hasil irisan dari *valid time* kedua *tuple* yang dibandingkan. Sebagai contoh, jika *valid time tuple*-nya adalah [1 Jan 08, 4 Jan 08] dan [2 Jan 08, 10 Jan 08], nilai *valid time* yang dikembalikan adalah rentang waktu yang beririsan, yaitu [2 Jan 08, 4 Jan 08]. Beberapa kondisi yang mungkin terjadi untuk dua *tuple* yang beririsan ini dapat dilihat pada Gambar III-31.



Gambar III-31 Kondisi *valid time* dua *tuple* yang beririsan pada operasi *join*

Berdasarkan Gambar III-31, untuk mendapatkan nilai *valid time* hasil irisan ini, pengecekan yang dilakukan dapat disederhanakan sebagai berikut:

- Untuk nilai *valid time* awal, dipilih nilai *vs* yang lebih akhir. Jika $vs1 > vs2$, nilai *valid time* awal pada *valid time* irisan adalah $vs1$.
- Untuk nilai *valid time* akhir, dipilih nilai *ve* yang lebih awal. Jika $ve1 < ve2$, nilai *valid time* akhir pada *valid time* irisan adalah $ve1$.

```

SELECT <select_attr_list> ,
      CASE WHEN table_name1.vs > table_name2.vs THEN
      dbtotable(table_name1.vs) ELSE dbtotable(table_name2.vs) END AS
      vs,
      CASE WHEN dbtoactual(table_name1.ve) <
      dbtoactual(table_name2.ve) THEN dbtotable(table_name1.ve) ELSE
      dbtotable(table_name2.ve) END AS ve
FROM table_name1 [ alias1 ] , table_name2 [ alias2 ]
WHERE sys_overlaps(table_name1.vs, dbtoactual(table_name1.ve),
table_name2.vs, dbtoactual(table_name2.ve))
AND sys_overlaps(table_name1.ts, dbtoactual(table_name1.te),
table_name2.ts, dbtoactual(table_name2.te))
[ WHERE <converted_select_condition> ]

```

Gambar III-32 Query konversi untuk operasi *select* dua relasi

3.4.5.3 Coalescing

Coalescing pada relasi *valid time* yang ditampilkan dapat dilakukan dengan beberapa cara, yaitu lewat pendekatan iteratif dan non-iteratif. Pendekatan non-iteratif dilakukan dengan modifikasi terhadap *tuple* dengan cara memeriksa kondisinya satu-persatu, sedangkan pendekatan non-iteratif dilakukan dengan sebuah *query* kompleks yang akan menghasilkan relasi dengan kondisi *coalesced*. *Query* pada Gambar III-33 adalah *query coalescing* dengan pendekatan non-iteratif [BOH97] yang telah dimodifikasi untuk menangani interval *closed ended*.

Relasi *temp* pada Gambar III-33 adalah relasi dari *query* konversi yang dilakukan seperti pada Gambar III-30 dan Gambar III-32. Sedangkan *attr* [,...] adalah semua atribut *non-valid-time* yang ditampilkan.

<pre>(1) SELECT DISTINCT F.attr [,...] , F.vs, L.ve FROM temp AS F, temp AS L WHERE F.vs <= L.ve AND F.attr = L.attr [,...]</pre>
<pre>(2) AND NOT EXISTS (SELECT * FROM Temp AS M WHERE M.attr = F.attr [,...] AND F.vs < M.vs AND M.vs < L.ve AND NOT EXISTS (SELECT * FROM temp AS T1 WHERE T1.attr = F.attr [,...] AND T1.vs < M.vs AND M.vs <= T1.ve+INTERVAL '1 DAY'))</pre>
<pre>(3) AND NOT EXISTS (SELECT * FROM temp AS T2 WHERE T2.attr = F.attr [,...] AND ((T2.vs < F.vs AND F.vs <= T2.ve+INTERVAL '1 DAY') OR (T2.vs <= L.ve+INTERVAL '1 DAY' AND L.ve < T2.ve)))</pre>

Gambar III-33 Query untuk coalescing pada operasi select

Penjelasan *query* pada Gambar III-33 adalah sebagai berikut:

- (1) Menyeleksi *tuple-tuple* yang *value-equivalent*.
- (2) Memastikan bahwa *tuple* hasil *join* yang diterima hanya yang mempunyai interval *valid time extended*, yaitu mempunyai interval yang berkelanjutan (*overlaps* atau *adjacent*). Sebagai contoh, *tuple* hasil *join* dengan interval [1 Jan 07, 28 Feb 07] tidak akan diterima jika interval *tuple* yang *value-equivalent* adalah [1 Jan 07, 10 Jan 07] dan [1 Feb 07, 28 Feb 07].
- (3) Memastikan bahwa *tuple* hasil *join* yang diterima hanya yang mempunyai interval *valid time* terbesar pada setiap *tuple* yang *value-equivalent*. Sebagai contoh, jika terdapat *tuple-tuple* yang *value-equivalent* dengan interval [1 Jan 07, 31 Jan 07] dan [2 Jan 07, 10 Jan 07], *tuple* yang diterima hanya yang mempunyai interval [1 Jan 07, 31 Jan 07].