

BAB II

LANDASAN TEORI

2.1 Basis Data Temporal

Basis data temporal adalah basis data yang mendukung aspek temporal di luar waktu yang didefinisikan oleh pengguna. Basis data temporal memungkinkan pengasosiasian fakta dengan waktu [JEN98]. Basis data temporal merupakan pengayaan dari basis data relasional yang memperhitungkan aspek waktu. Pada basis data konvensional tidak terdapat penanganan khusus bagi waktu. Dalam hal ini, waktu diperlakukan sebagai sebuah tipe data yang dapat digunakan dalam atribut. Basis data konvensional juga tidak mengurus aspek historis keberlakuan data di basis data. Jika suatu *record* di-*update*, data lama yang tercatat dalam *record* tersebut hilang dan tidak dapat diakses lagi.

Basis data temporal mendukung penanganan aspek waktu yang kompleks dan dapat menyimpan aspek historis suatu data. Proses *update* atau penghapusan terhadap suatu data tidak menyebabkan data tersebut secara fisik hilang dari basis data, tetapi hanya menandai bahwa waktu keberlakuannya di basis data telah habis. Selain itu, terdapat operator khusus untuk waktu yang tidak ada di basis data konvensional, seperti operator untuk mengetahui dua periode waktu yang *overlap* atau bersinggungan.

2.1.1 Dimensi Waktu

Pada basis data temporal dikenal tiga dimensi waktu [SNO85]:

1. *User-defined time*

User-defined time adalah waktu yang semantiknya ditentukan oleh pengguna. Dimensi waktu ini diperlakukan sama seperti atribut lainnya dan tidak ditangani secara khusus. Sebagai contoh, sebuah tabel yang menyimpan data pegawai dapat mempunyai dua atribut bertipe tanggal untuk menyimpan data tanggal lahir dan tanggal mulai bekerja. Keberadaan atribut ini tidak mempunyai kaitan dengan waktu keberlakuan data.

2. *Transaction time*

Transaction time adalah waktu keberlakuan data di basis data. Secara teknis, ini adalah waktu saat data dimasukkan ke basis data sampai dengan data diubah atau dihapus dari basis data. Nilai *transaction time* suatu data dikelola oleh sistem dan tidak pernah dimasukkan secara langsung oleh pengguna. Nilai keberlakuan data ini dicatat secara otomatis saat terjadi operasi terhadap data, yaitu *insert*, *update*, dan *delete*. Domain

transaction time mencakup masa lalu hingga saat ini. Nilai *transaction time* yang digunakan untuk menandakan bahwa data berlaku untuk saat ini adalah UC (*until changed*).

3. *Valid time*

Valid time adalah waktu keberlakuan data di dunia nyata. Nilai keberlakuan data ini dimasukkan oleh pengguna. Sebagai contoh, suatu data bahwa pegawai bernama Heru mendapat gaji Rp 2.500.000,- dapat ditentukan keberlakuannya dari tanggal 1 Januari sampai dengan akhir tahun. Representasi *valid time* bisa bermacam-macam tergantung pengguna. Domainnya mencakup masa lalu sampai dengan masa depan. Nilai *valid time* yang digunakan untuk menandakan bahwa data berlaku sampai dengan waktu yang belum ditentukan adalah Now.

Dimensi waktu yang khusus terdapat pada basis data temporal adalah dimensi *transaction time* dan *valid time*. *User-defined time* bukan merupakan sesuatu yang baru karena terdapat juga pada basis data konvensional. Dimensi waktu ini bukan dimensi waktu yang ditangani secara khusus pada basis data temporal karena hanya dianggap sebagai nilai dalam suatu atribut, bukan aspek yang berhubungan dengan keberlakuan data.

2.1.2 Format Waktu

Format waktu adalah struktur waktu yang digunakan dalam menyimpan waktu keberlakuan data. Ada tiga jenis format waktu, yaitu:

1. *Time point*

Time point merupakan sebuah titik waktu atau *event* yang diasosiasikan dengan data. Terdapat dua representasi waktu dengan *time point*, yaitu waktu saat sesuatu terjadi dan waktu saat suatu aksi mulai berlaku. Sebagai contoh, pada Gambar II-1, terdapat *time point* bernilai "1989" yang diasosiasikan dengan data "Heru" dan "TK Al Hidayah." Ini dapat diartikan bahwa Heru bersekolah selama satu tahun, yaitu tahun 1989 di TK tersebut, atau dapat pula diartikan bahwa Heru bersekolah mulai tahun 1989 sampai 1990 berdasarkan data selanjutnya. Representasi ini tergantung dari pendefinisian .

Nama	Sekolah	
Heru	TK Al Hidayah	1989
Heru	SDN Karang Pawulang	1991

Gambar II-1 Contoh penggunaan *time point*

Format waktu *time point* mempunyai kelemahan, yaitu:

- Untuk representasi *time point* sebagai waktu saat sesuatu terjadi, *record* yang dibutuhkan akan menjadi banyak karena suatu *record* hanya mempunyai satu *event*

yang menunjukkan keberlakuannya. Ini mengakibatkan suatu data yang berlaku pada lebih dari satu satuan waktu ditulis berkali-kali pada basis data.

- Untuk representasi *time point* sebagai waktu saat sesuatu dimulai, sulit mendapatkan informasi mengenai akhir keberlakuan data. Untuk mengetahuinya, perlu dilihat *time point* pada data selanjutnya yang berhubungan dengan data tersebut. Keterhubungan data ini juga perlu didefinisikan secara jelas. Selain itu, *dummy record* diperlukan jika keberlakuan antara satu *record* dengan *record* lainnya tidak berkelanjutan.

2. *Time interval*

Format waktu *time interval* merupakan himpunan *chronon-chronon* terurut atau durasi waktu keberlakuan data. Ada dua representasi *time interval*, yaitu:

- *Closed ended*, dilambangkan dengan [*<waktu awal>*,*<waktu akhir>*]. Waktu akhir adalah waktu saat data terakhir kali berlaku.
- *Open ended*, dilambangkan dengan [*<waktu awal>*,*<waktu akhir>*). Waktu akhir bernilai satu *chronon* setelah data berlaku.

Sebagai contoh, jika sebuah buku dipinjam dari tanggal 1 September 2007 dan dikembalikan tanggal 6 September 2007, maka representasi keberlakuan data untuk peminjaman buku tersebut adalah [1-9-2007, 5-9-2007] dengan *closed ended* dan [1-9-2007, 6-9-2007) dengan *open ended*. *Time interval* hanya mencakup *chronon* yang berurutan. Jadi, jika suatu data kembali berlaku pada *chronon* yang terpisah, data tersebut menjadi data baru. Pada Gambar II-2, buku berkode C124 dipinjam oleh anggota perpustakaan bernomor A1 pada dua interval yang terpisah. Data ini menjadi dua *record* berbeda.

ID Buku	ID Peminjam	
C124	A1	[1-9-2007, 6-9-2007)
C124	B2	[6-9-2007, 10-9-2007)
C124	A1	[15-9-2007,20-9-2007)

Gambar II-2 Contoh penggunaan *time interval* dengan *open ended*

3. *Temporal elements*

Format waktu *temporal elements* adalah himpunan *chronon-chronon* dari gabungan interval yang terpisah. Dengan format waktu ini, data yang sama tidak pernah berulang, hanya nilai waktunya yang berubah. Sebagai contoh, jika data peminjaman buku C124 oleh A1 pada Gambar II-2 direpresentasikan dengan *temporal elements*, data tersebut hanya akan termuat dalam satu *record* seperti dalam Gambar II-3. Seperti format *time interval*, format waktu ini juga mempunyai representasi *closed ended* dan *open ended*.

ID Buku	ID Peminjam	
C124	A1	[1-9-2007, 6-9-2007) U [15-9-2007,20-9-2007)
C124	B2	[6-9-2007, 10-9-2007)

Gambar II-3 Contoh penggunaan *temporal elements*

2.1.3 Granularitas Waktu

Granularitas waktu adalah tingkat ketelitian waktu yang digunakan dalam keberlakuan data. Ada dua pendekatan dalam pengimplementasian granularitas waktu, yaitu:

1. *Single database wide granularity*, artinya pada basis data hanya terdapat satu satuan waktu yang dipakai, misalnya tahun, hari, atau semester. Pendekatan ini mudah diimplementasikan, tetapi dalam penggunaannya diperlukan banyak kompromi karena terkadang antara satu kumpulan data dan kumpulan data yang lain mempunyai satuan waktu umum yang berbeda.
2. *Multiple granularity*, artinya pada basis data terdapat berbagai satuan waktu yang dapat digunakan. Ini dapat memberikan gambaran yang lebih jelas mengenai keberlakuan data. Akan tetapi, konversi sangat diperlukan jika ingin menggabungkan atau membandingkan data yang mempunyai granularitas berbeda.

Penentuan granularitas waktu pada suatu basis data temporal adalah hal yang penting. Jika granularitas terlalu besar, kemungkinan adanya data yang tidak tercatat lebih besar karena bisa saja data berubah pada satu *chronon*. Jika granularitas terlalu kecil, data yang tidak terlalu penting dapat tercatat juga sehingga jumlah data menjadi lebih banyak dari yang dibutuhkan.

2.1.4 Taksonomi Relasi

Berdasarkan dimensi waktu pada basis data temporal, yaitu *transaction time* dan *valid time*, ada beberapa jenis relasi:

1. Relasi *snapshot*
Relasi ini adalah relasi yang hanya memuat data pada satu waktu tertentu dan tidak mengelola aspek keberlakuan data. Relasi *snapshot* sama seperti relasi pada basis data konvensional. Waktu yang dapat digunakan hanya *user-defined time*. Jika suatu data diubah atau dihapus, data tersebut benar-benar diubah atau dihapus secara fisik pada basis data.
2. Relasi *transaction time*
Relasi *transaction time* mengelola dimensi *transaction time* atau sejarah keberlakuan data. Data yang dimuat dalam relasi ini tidak pernah hilang. Jika suatu data diubah, data lama tidak hilang, tetapi nilai waktu keberlakuannya saja yang diakhiri. Data baru kemudian dimasukkan secara fisik ke basis data dengan nilai *transaction time* yang

berlaku sampai saat tersebut. Begitu pula jika data dihapus, data tersebut tidak benar-benar dihilangkan, tetapi nilai keberlakuannya diakhiri.

3. Relasi *valid time*

Relasi *valid time* mengelola dimensi *valid time*. Data yang dimuat dalam relasi ini adalah data yang mengandung nilai waktu keberlakuan di dunia nyata. Pada relasi *valid time*, dimensi *transaction time* tidak dikelola sehingga data yang ada merupakan data yang valid untuk saat ini saja.

4. Relasi bitemporal

Relasi bitemporal adalah relasi yang mengelola dimensi *valid time* dan *transaction time*. Data yang dimuat adalah data yang mengandung nilai waktu keberlakuan data di dunia nyata dan juga di basis data. Karena mengandung dimensi *transaction time*, pengubahan atau penghapusan data tidak akan menghilangkan data secara fisik. Ini dapat memberikan gambaran yang lebih jelas mengenai aspek waktu dari data, tetapi sebagai konsekuensinya jumlah data menjadi besar.

2.2 Model Data untuk Relasi Bitemporal

Model data temporal adalah model data yang digunakan untuk menyimpan dimensi waktu dari fakta yang terdapat pada relasi. Lebih dari 24 *extended relational model* diajukan untuk menambahkan dimensi waktu ke dalam model relasional. Kebanyakan dari model tersebut hanya mendukung dimensi *valid time* [WID03]. Ada tiga model data untuk relasi bitemporal, yaitu BCDM (*Bitemporal Conceptual Data Model*), model 1NF dengan *tuple stamping*, dan model N1NF dengan *attribute stamping*.

2.2.1 BCDM (*Bitemporal Conceptual Data Model*)

BCDM diajukan oleh Christian S. Jensen [JEN00]. Ide awal dari model ini adalah mempertahankan kesederhanaan model relasional dan memasukkan aspek temporal ke fakta yang tersimpan di basis data. Pada BCDM, tiap *tuple* diberi *timestamp* yang merupakan himpunan atribut (tt,vt). tt adalah nilai *transaction time* dan vt adalah nilai *valid time*. *Tuple* yang terdapat pada BCDM tidak berulang karena tidak ada *tuple* yang *value equivalent* (mempunyai nilai data non-temporal yang sama).

Sebagai contoh, buku berkode B123 dipinjam oleh anggota perpustakaan bernomor A1 pada tanggal 1 Oktober 2006 dan dikembalikan tanggal 3 Oktober 2006. Data dimasukkan tanggal 2 Oktober 2006. Pada saat buku tersebut dikembalikan, anggota bernomor A2 meminjam buku tersebut untuk dua hari dan benar-benar dikembalikan tanggal 5 Oktober 2006. Kedua

data tersebut masih berlaku hingga saat ini. Model untuk data ini dapat dilihat pada Gambar II-4.

ID Buku	ID Peminjam	
B123	A1	{(2/10/06,1/10/06), (2/10/06,2/10/06),... (UC,1/10/06), (UC,2/10/06)}
B123	A2	{(3/10/06,3/10/06), (3/10/06,4/10/06),... (UC,3/10/06), (UC,4/10/06)}

Gambar II-4 Contoh I pemodelan dimensi waktu dengan BCDM

Jika pada tanggal 1 Oktober 2007 dicatat bahwa buku tersebut kembali dipinjam oleh anggota bernomor A1 pada hari itu sampai waktu yang belum ditentukan, *timestamp* fakta tersebut berubah seperti pada Gambar II-5.

ID Buku	ID Peminjam	
B123	A1	{(2/10/06,1/10/06), (2/10/06,2/10/06),... (31/9/07,1/10/06), (31/9/07,2/10/06), (1/10/07,1/10/06), (1/10/07,2/10/06), (1/10/07,1/10/07)... (1/10/07,Now)... (UC,1/10/06), (UC,2/10/06), (UC,1/10/07)... (UC, Now)}
B123	A2	{(3/10/06,3/10/06), (3/10/06,4/10/06),... (UC,3/10/06), (UC,4/10/06)}

Gambar II-5 Contoh II pemodelan dimensi waktu dengan BCDM

BCDM memiliki keuntungan, yaitu aspek waktu dari sebuah fakta hanya tersimpan dalam satu *tuple*. Akan tetapi, representasi internal sulit dan tidak mudah dipahami pengguna. Selain itu, dengan model ini nilai *timestamp* harus diperbaharui untuk setiap *chronon*.

2.2.2 Model 1NF dengan *Tuple Stamping*

Model 1NF dengan *tuple stamping* telah mengikuti kaidah relasi, yaitu setiap nilai dalam atribut bersifat atomik. Pada model ini, format waktu yang digunakan adalah *time interval* yang direpresentasikan dengan dua titik waktu terpisah yang menunjukkan waktu awal dan waktu akhir keberlakuan data. Atribut *ts* dan *te* merupakan waktu awal dan akhir *transaction time*. Sedangkan *vs* dan *ve* merupakan waktu awal dan akhir *valid time*. Model ini lebih mudah diimplementasikan di basis data relasional dan lebih mudah dipahami oleh pengguna. Contoh pemodelan 1NF dari pencatatan yang sama dengan Gambar II-5 dapat dilihat pada Gambar II-6. Tabel ini menggunakan periode waktu yang *closed ended*.

ID Buku	ID Peminjam	ts	te	vs	ve
B123	A1	2/10/06	UC	1/10/06	2/10/06
B123	A2	3/10/06	UC	3/10/06	4/10/06
B123	A1	1/10/07	UC	1/10/07	Now

Gambar II-6 Contoh model 1NF

Kelebihan dari model 1NF adalah dimensi waktu mudah dipelihara. Berbeda dengan BCDM, pada model 1NF proses perbaharuan nilai waktu lebih mudah dilakukan karena hanya nilai T_e yang perlu diubah dari UC ke nilai tertentu. Sedangkan kelemahannya adalah *tuple* yang dibutuhkan lebih banyak daripada pada BCDM karena pada model ini *tuple* bisa *value equivalent*.

2.2.3 Model N1NF dengan *Attribute Stamping*

Model N1NF (*Non First Normal Form*) mengasosiasikan waktu di level atribut, sehingga waktunya tidak atomik. Ide dari model ini adalah bahwa saat fakta baru dicatat di basis data, nilai atribut-atribut dari fakta tersebut bisa saja telah tercatat di basis data, sehingga tidak perlu dimasukkan kembali. Dengan model ini, cukup nilai waktu yang diasosiasikan dengan atribut saja yang diperbaharui. Setiap atribut diasosiasikan dengan nilai waktu $[T_s, T_e] \times [V_s, V_e]$. Model N1NF untuk pencatatan yang sama dengan Gambar II-5 dan Gambar II-6 adalah seperti pada Gambar II-7.

ID Buku	ID Peminjam
$[2/10/06, UC] \times [1/10/06, 2/10/06]$ $[1/10/07, UC] \times [1/10/07, \infty]$ $[3/10/06, UC] \times [3/10/06, 4/10/06]$	$[2/10/06, UC] \times [1/10/06, 2/10/06]$ $[1/10/07, UC] \times [1/10/07, \infty]$ A1
	$[3/10/06, UC] \times [3/10/06, 4/10/06]$ A2

Gambar II-7 Contoh model N1NF

Implementasi model ini menyangkut pemisahan atribut-atribut relasi menjadi *tuple* yang berbeda, sehingga terjadi anomali vertikal. Penanganan data pada model ini membutuhkan manipulasi yang kompleks dan sulit diterapkan pada basis data relasional.

2.3 Integritas Data

Integritas data adalah jaminan bahwa data tersebut konsisten di basis data. Integritas data terdiri dari *entity integrity* dan *referential integrity*. *Entity integrity* menjamin bahwa setiap *tuple* unik dan tidak ada *missing value* untuk setiap nilai unik tersebut. *Referential integrity* menjamin bahwa suatu data yang di-refer oleh data lainnya benar-benar eksis dan nilainya tidak hilang [LEV00]. Dengan adanya integritas data, penanganan dan manipulasi data dapat berjalan aman karena konsistensi tetap terjaga. Pada pengimplementasian integritas data di basis data, *entity integrity* ditangani oleh *primary key*, sedangkan *referential integrity* ditangani oleh *foreign key*.

Primary key dan *foreign key* yang ditangani pada relasi bitemporal berbeda dengan pada relasi biasa karena berhubungan dengan *valid time* dan *transaction time*. Pada *primary key* biasa,

nilai atribut kunci pada setiap *tuple* tidak boleh sama. Ini tidak berlaku jika pada relasi tersebut terdapat dimensi waktu. Pada waktu yang berbeda bisa saja terdapat nilai atribut kunci yang sama. *Primary key* pada relasi bitemporal menjamin bahwa pada setiap *snapshot*, nilai atribut kunci berbeda untuk setiap *chronon* dalam *valid time*. *Foreign key* pada relasi biasa hanya memastikan bahwa nilai yang diacu oleh sebuah relasi benar-benar eksis di relasi lain. Sedangkan *foreign key* pada relasi bitemporal memastikan bahwa nilai yang diacu benar-benar eksis pada *valid time* dan *transaction time* yang sama.

Operasi *update* dan *delete* pada relasi yang diacu oleh nilai *foreign key* dapat menjadi masalah jika nilai pada relasi tersebut sedang diacu oleh relasi lain. Ada tiga aksi yang dapat dilakukan, yaitu *restrict*, *cascade*, dan *set null*. Pada aksi *restrict*, operasi *delete* atau *update* terhadap nilai yang diacu tersebut tidak dapat dilakukan. Pada aksi *cascade*, operasi *delete* atau *update* dapat dilakukan. Operasi tersebut kemudian akan berdampak pada *tuple* yang mengacu nilai tersebut. Jika *update* terhadap nilai yang diacu dilakukan, nilai pada relasi yang mengacu juga ikut berubah. Jika *delete* dilakukan, *tuple* yang mengacu nilai yang dihilangkan akan di-*delete* juga. Pada aksi *set null*, jika operasi *delete* dan *update* mempengaruhi nilai yang mengacunya, nilai pada *foreign key* tersebut diset dengan nilai *null*.

2.4 Query untuk Operasi Relasi Bitemporal

Ada beberapa bahasa *query* temporal yang telah diajukan pada berbagai literatur, seperti TSQL2, ChronoBase, SQL+T, TMQL, TOSQL, dan TQuel. Salah satu bahasa yang mendapatkan dukungan terbesar dalam riset mengenai basis data temporal adalah TSQL2. TSQL2 telah menjadi standar dalam *temporal database community* [SNO96]. TSQL2 merupakan pengembangan dari SQL92 yang ditujukan bagi operasi pada basis data temporal [SNO95]. Bahasa ini dirancang oleh TSQL2 Design Language Committee yang terdiri dari sembilan belas pakar basis data temporal, termasuk Richard T. Snodgrass dan Christian S. Jensen. Spesifikasi untuk TSQL2 telah diterbitkan pada tahun 1994. Saat ini, beberapa perubahan telah diajukan untuk menyempurnakan spesifikasi tersebut. Bahasa *query* yang digunakan dalam penulisan *query* temporal pada tugas akhir ini adalah TSQL2.

2.4.1 Create Table

Operasi *create table* adalah operasi yang digunakan untuk mendefinisikan dan membuat relasi. Pada basis data temporal, selain klausa standar `CREATE TABLE` untuk membuat relasi, perlu ditentukan juga jenis relasi dalam klausa `AS temporal_dimension` seperti pada Gambar II-8.

```
CREATE TABLE table_name (
  { column_name data_type [ DEFAULT default_value ]
    [ column_constraint [...] ]
  | table_constraint } [, ...]
) [ AS temporal_dimension ]
```

Gambar II-8 Query temporal untuk create table

Bagian *temporal_dimension* dapat berisi `VALID`, `TRANSACTION`, atau `VALID AND TRANSACTION`. *Statement create table* tanpa deklarasi dimensi waktu akan menghasilkan relasi *snapshot*. Kata `VALID` digunakan untuk membuat relasi *valid time* dan `TRANSACTION` untuk *transaction time*. Untuk membuat relasi bitemporal, perlu ditambahkan `AS VALID AND TRANSACTION`. Contoh penggunaan *query create table* untuk membuat relasi bitemporal dapat dilihat pada Gambar II-9.

```
CREATE TABLE pegawai (
  nip char(5) PRIMARY KEY,
  nama varchar(30) NOT NULL,
  gaji integer DEFAULT 0
) AS VALID AND TRANSACTION
```

Gambar II-9 Contoh query create table pada relasi pegawai

2.4.2 Insert

Operasi *insert* adalah operasi yang digunakan untuk memasukkan *tuple* baru ke dalam basis data. Pada relasi bitemporal, perlu dimasukkan nilai *valid time* yang dituliskan dalam klausa `VALID` seperti pada Gambar II-10.

```
INSERT INTO table_name [ ( column_name [, ...] ) ]
VALUES ( value [, ...] )
[ VALID { PERIOD '[v_start,v_end]' | INSTANT 'time_value' } ]
```

Gambar II-10 Query temporal untuk insert

`VALID PERIOD` digunakan untuk *time interval*, sedangkan `VALID INSTANT` untuk *time point*. Pada relasi dengan format *time interval*, jika nilai *valid time* tidak dideklarasikan, *valid time* akan bernilai *current_time* pada *time point* atau `[current_time, now]` pada *time interval*. Nilai *transaction time* tidak perlu dimasukkan lewat *query* karena telah ditangani secara otomatis oleh sistem. Contoh penggunaan *query insert* pada relasi bitemporal dapat dilihat pada Gambar II-11. *Query* tersebut memasukkan data pegawai bernama Heru dengan gaji 2500000 yang berlaku mulai tanggal 1 Januari 2007 sampai batas waktu yang belum ditentukan.

```
INSERT INTO pegawai (nip, nama, gaji)
VALUES ('10031', 'Heru Haryadhi', 2500000)
VALID PERIOD '[1 Jan 07, now]'
```

Gambar II-11 Contoh query insert pada tabel pegawai

Gambar II-11 yang dieksekusi pada tanggal 5 Oktober 2007 akan menghasilkan *tuple* pada tabel pegawai seperti pada Gambar II-12.

nip	nama	gaji	vs	ve	ts	te
10031	Heru Haryadhi	2500000	1/1/07	now	5/10/07	UC

Gambar II-12 Contoh hasil query insert pada tabel pegawai

2.4.3 Delete

Operasi *delete* digunakan untuk menghapus suatu *tuple*. Pada relasi bitemporal, *delete* hanya dilakukan secara logik karena relasi mengandung dimensi *transaction time*. Operasi ini tidak akan secara fisik menghapus *tuple* di basis data kecuali jika *delete* dilakukan pada waktu kurang dari satu *chronon transaction time* dengan *insert*. Pada relasi bitemporal, *delete* dapat dilakukan untuk periode *valid time* tertentu Artinya, keberlakuan data pada periode waktu tersebut dihilangkan. Query temporal untuk *delete* dapat dilihat pada Gambar II-13.

```
DELETE FROM table_name
[ WHERE condition ]
[ VALID { PERIOD '[v_start, v_end]' | INSTANT 'time_value' } ]
```

Gambar II-13 Query temporal untuk delete

Pada relasi dengan format *time interval*, jika nilai *valid time* tidak dideklarasikan, waktu keberlakuan data yang dihilangkan adalah [present, forever] [JEN00]. Nilai 'present' adalah waktu saat operasi dilakukan. Contoh query *delete* pada relasi bitemporal dapat dilihat pada Gambar II-14. Pada query tersebut, *tuple* dengan nip='10031' dihilangkan keberlakuan datanya untuk tanggal 1 Januari 2007 sampai 31 Januari 2007.

```
DELETE FROM pegawai
WHERE nip = '10031'
VALID PERIOD '[1 Jan 07, 31 Jan 07]'
```

Gambar II-14 Contoh query delete pada tabel pegawai

Jika pada tanggal 8 Oktober 2007 query pada Gambar II-14 dilakukan terhadap relasi pegawai pada Gambar II-12, relasi pada basis data menjadi seperti pada Gambar II-15. *Tuple* yang memenuhi kondisi akan diakhiri keberlakuannya di basis data. *Tuple* baru dimasukkan dengan keberlakuan *valid time* yang baru, yaitu *valid time* yang lama dikurangi *valid time* pada query *delete*.

nip	nama	gaji	vs	ve	ts	te
10031	Heru Haryadhi	2500000	1/1/07	now	5/10/07	7/10/07
10031	Heru Haryadhi	2500000	1/2/07	now	8/10/07	UC

Gambar II-15 Contoh hasil *query delete* pada tabel pegawai

2.4.4 Update

Operasi *update* adalah operasi yang digunakan untuk memperbarahui nilai dalam suatu *tuple*. Pada basis data temporal, pengubahan nilai yang dilakukan dapat melibatkan nilai atribut eksplisit pada suatu *tuple* atau nilai *valid time* dari *tuple* tersebut. *Query* temporal untuk *update* dapat dilihat pada Gambar II-16.

```
UPDATE table_name
{ SET column_name = value [, ...] [ VALID { PERIOD '[v_start,v_end]'
| INSTANT 'time_value' } ]
| SET VALID { PERIOD '[v_start,v_end]' | INSTANT 'time_value' } }
[ WHERE condition ]
```

Gambar II-16 *Query* temporal untuk *update*

Tanpa klausa `VALID time_value`, *update* hanya dilakukan terhadap atribut eksplisit. *Update* terhadap *tuple* yang memenuhi kondisi akan mengakhiri keberlakuan data *tuple* lama dan memasukkan *tuple* baru dengan nilai atribut eksplisit yang baru dan nilai *valid time* yang lama. Jika klausa `VALID time_value` digunakan, *tuple* yang baru dimasukkan dengan nilai *valid time* yang terdapat pada `time_value`. Contoh penggunaan *query update* pada relasi bitemporal dapat dilihat pada Gambar II-17. Pada *query* tersebut, *tuple* yang mengandung nilai nip = 10031 diubah namanya menjadi “Heru Hariyadhi” dan keberlakuan datanya diubah menjadi 1 Februari 2007 sampai 31 Desember 2007.

```
UPDATE pegawai
SET nama = 'Heru Hariyadhi'
VALID PERIOD '[1 Feb 07,31 Dec 07]'
WHERE nip = '10031'
```

Gambar II-17 Contoh *query update* pada tabel pegawai

Jika *query* pada Gambar II-17 dilakukan pada tanggal 10 Oktober 2007 terhadap relasi pada Gambar II-15, data pada relasi akan menjadi seperti pada Gambar II-18. *Tuple* lama akan diakhiri masa keberlakuannya di basis data dan sebuah *tuple* baru dimasukkan.

nip	nama	gaji	Vs	Ve	Ts	Te
10031	Heru Haryadhi	2500000	1/1/07	now	5/10/07	7/10/07
10031	Heru Haryadhi	2500000	1/2/07	now	8/10/07	9/10/07
10031	Heru Hariyadhi	2500000	1/2/07	31/12/07	10/10/07	UC

Gambar II-18 Contoh hasil *query update* pada tabel pegawai

Update pada relasi bitemporal tidak menghapus nilai sebelumnya, kecuali jika waktu *update* kurang dari satu *chronon transaction time*. Misalnya jika *chronon* untuk *transaction time* adalah hari, nilai sebelumnya akan diganti secara fisik dengan yang baru di basis data jika *update* dilakukan di hari yang sama dengan *insert*.

2.4.5 Select

Operasi *select* digunakan untuk me-retrieve *tuple* dalam relasi sesuai dengan kondisi yang diinginkan. Pada relasi bitemporal, *tuple* dapat diseleksi berdasarkan keberlakuan *valid time* atau *transaction time*. *Query* untuk operasi *select* pada relasi bitemporal dapat dilihat pada Gambar II-19.

```
SELECT [SNAPSHOT ] [DISTINCT] column_name [, ...]
[ { VALID | VALID INTERSECT } time_value ]
FROM table_name [, ...]
[ WHERE condition ]
```

Gambar II-19 Query select untuk relasi bitemporal

Query `SELECT column_name [, ...] FROM table_name` tanpa klausa tambahan akan me-retrieve semua *tuple* yang berlaku saat ini di basis data. Ini termasuk informasi mengenai keberlakuan *valid time*. Contoh *query* ini dapat dilihat pada Gambar II-20.

```
SELECT nama,gaji
FROM pegawai
```

Gambar II-20 Contoh query select tanpa klausa tambahan

Misalkan terdapat relasi pegawai di basis data dengan data seperti pada Gambar II-21.

nip	nama	gaji	vs	ve	ts	te
10031	Heru Haryadhi	2500000	1/1/07	now	5/10/07	7/10/07
10031	Heru Haryadhi	2500000	1/2/07	now	8/10/07	9/10/07
10031	Heru Hariyadhi	2500000	1/2/07	31/12/07	10/10/07	UC
10032	Wiyanda Puspita	4000000	1/1/07	31/5/07	1/1/07	UC
10032	Wiyanda Puspita	4500000	1/6/07	now	1/6/07	UC

Gambar II-21 Contoh data pada relasi bitemporal

Query pada Gambar II-20 yang dilakukan pada tanggal 11 Oktober 2007 terhadap relasi pegawai pada Gambar II-21 akan menghasilkan *tuple* pada Gambar II-22.

Nama	gaji	V
Heru Hariyadhi	2500000	[1/2/07,31/12/07]
Wiyanda Puspita	4000000	[1/1/07,31/5/07]
Wiyanda Puspita	4500000	[1/6/07,now]

Gambar II-22 Contoh hasil query select tanpa klausa tambahan

Seleksi berdasarkan keberlakuan data dapat dilakukan dalam klausa `WHERE` dengan menggunakan operator temporal. Seleksi ini dapat melibatkan dimensi *valid time* maupun

transaction time. Nilai *valid time* dari suatu *tuple* pada relasi *table_name* dapat diacu dengan `VALID(table_name)`. Sedangkan *transaction time* diacu dengan menggunakan `TRANSACTION(table_name)`. Contoh seleksi berdasarkan *valid time* dapat dilihat pada Gambar II-23. *Query* ini menyeleksi *tuple* pada relasi pegawai yang mempunyai nilai gaji di atas 3000000 dan mempunyai keberlakuan *valid time* yang beririsan dengan tanggal 1 Mei 2007 sampai 31 Mei 2007. Operator `OVERLAPS` adalah operator khusus pada basis data temporal yang akan dibahas pada subbab 2.5. Karena kondisi seleksi *transaction time* tidak disebutkan, *tuple* yang di-*retrieve* adalah *tuple* yang berlaku saat ini.

```
SELECT nama,gaji
FROM pegawai
WHERE gaji > 3000000 AND
VALID(pegawai) OVERLAPS PERIOD '[1 May 07,31 May 07]'
```

Gambar II-23 Contoh query select dengan seleksi valid time

Query pada Gambar II-23 yang dilakukan pada tanggal 11 Oktober 2007 terhadap relasi pegawai pada Gambar II-21 akan menghasilkan *tuple* seperti pada Gambar II-24.

Nama	gaji	V
Wiyanda Puspita	4000000	[1/1/07,31/5/07]

Gambar II-24 Contoh hasil query select dengan seleksi valid time

Seleksi terhadap *transaction time* dilakukan untuk melihat data yang berlaku pada masa lalu, disebut juga *obsolete database*. Contoh *query select* untuk melihat *obsolete database* dapat dilihat pada Gambar II-25. *Query* ini melihat data relasi pegawai yang berlaku pada tanggal 8 Oktober 2007.

```
SELECT nama,gaji
FROM pegawai
WHERE TRANSACTION(pegawai) OVERLAPS DATE '8 Oct 07'
```

Gambar II-25 Contoh query select dengan seleksi transaction time

Query pada Gambar II-25 yang dilakukan pada tanggal 11 Oktober 2007 terhadap relasi pegawai pada Gambar II-21 akan menghasilkan *tuple* pada Gambar II-26.

Nama	Gaji	V
Heru Haryadhi	2500000	[1/2/07, now]
Wiyanda Puspita	4500000	[1/6/07, now]

Gambar II-26 Contoh hasil query select dengan seleksi transaction time

Klausa `VALID` atau `VALID INTERSECT` pada operasi *select* digunakan untuk proyeksi nilai *valid time*. Proyeksi nilai ini digunakan pada operasi *insert* untuk memasukkan *tuple* yang dihasilkan dari operasi *select* ke dalam suatu relasi. Klausa `VALID` digunakan jika nilai *valid*

time baru dimasukkan, sedangkan `VALID INTERSECT` digunakan jika nilai *valid time* lama dipertahankan. Contoh penggunaan klausa ini dapat dilihat pada Gambar II-27.

```
INSERT INTO new_pegawai
SELECT nama, gaji
VALID PERIOD '[1 Jan 07, forever]'
FROM pegawai
WHERE VALID(pegawai) OVERLAPS DATE '1 Jan 07'
```

Gambar II-27 Contoh penggunaan klausa *valid* pada operasi *select*

Keyword `SNAPSHOT` digunakan untuk me-*retrieve tuple* tanpa menampilkan informasi *valid time*. Dengan kata lain, relasi yang ditampilkan adalah relasi *snapshot*. Contoh *query select snapshot* dapat dilihat pada Gambar II-28. *Query* ini menampilkan nama pegawai yang pernah menerima gaji di bawah 4200000 tanpa memperhatikan kapan pegawai tersebut menerimanya.

```
SELECT SNAPSHOT nama
FROM pegawai
WHERE gaji < 4200000
```

Gambar II-28 Contoh I *query select snapshot*

Query pada Gambar II-28 yang dilakukan tanggal 11 Oktober 2007 terhadap relasi pegawai pada Gambar II-21 akan me-*retrieve tuple* pada Gambar II-29.

nama
Heru Hariyadhi
Wiyanda Puspita

Gambar II-29 Contoh I hasil *query select snapshot*

Contoh lain penggunaan keyword `SNAPSHOT` dapat dilihat pada Gambar II-30. *Query* ini menampilkan nama pegawai yang pernah mendapat kenaikan gaji, serta gaji sebelum kenaikan dan gaji sesudah kenaikan. *Query* pada Gambar II-30 yang dilakukan terhadap relasi pada Gambar II-21 akan me-*retrieve tuple* seperti pada Gambar II-31.

```
SELECT SNAPSHOT a.nama, a.gaji as gaji_lama, b.gaji as gaji_baru
FROM pegawai a, pegawai b
WHERE a.nip = b.nip AND VALID(a) MEETS VALID(b) AND b.gaji > a.gaji
```

Gambar II-30 Contoh II *query select snapshot*

a.nama	gaji lama	gaji baru
Wiyanda Puspita	4000000	4500000

Gambar II-31 Contoh II hasil *query select snapshot*

2.5 Operator Perbandingan pada TSQL2

Operator perbandingan temporal yang diterima pada *query* tergantung pada bahasa *query* yang digunakan. Terdapat beberapa definisi yang berbeda untuk operator yang sama. Sebagai contoh, definisi “overlaps” berbeda pada operator perbandingan Allen dan TSQL2. Operator untuk perbandingan interval *valid time* pada TSQL2 dapat dilihat pada Tabel II-1 [JEN00]. Definisi operator perbandingan temporal yang digunakan dalam tugas akhir ini adalah definisi pada tabel tersebut.

Tabel II-1 Definisi Operator Perbandingan Interval

Operator	Definisi
A PRECEDES B	END (A) lebih awal daripada BEGIN (B)
A = B	A dan B bernilai sama
A OVERLAPS B	Irisan A dan B tidak kosong
A CONTAINS B	Setiap <i>event</i> di B terdapat pada A
A MEETS B	A PRECEDES B dan tidak ada <i>event</i> antara END (A) dan BEGIN (B)

Operator perbandingan PRECEDES, =, OVERLAPS, CONTAINS, dan MEETS juga dapat digunakan untuk perbandingan periode dengan *event* atau *event* dengan *event* karena *event* dapat dipandang sebagai periode dengan kondisi tertentu, yaitu nilai awal dan akhir yang sama. Untuk perbandingan *event* dengan *event*, operator =, CONTAINS, dan OVERLAPS akan bernilai sama.

2.6 Variabel dan Konstanta Waktu

Dalam basis data temporal, dikenal nilai waktu ‘now’, ‘beginning’, dan ‘forever’ untuk *valid time* serta ‘UC’ (*until changed*) untuk *transaction time*. Nilai waktu ini tidak terdapat pada basis data relasional biasa dan harus ditangani secara khusus. ‘Now’ adalah nilai waktu yang di-*bind* dengan waktu saat ini, sehingga nilai aktualnya dapat berubah tergantung waktu di dunia nyata. Nilai ‘now’ umumnya digunakan jika data berlaku sampai saat ini dan akan tetap berlaku sampai waktu yang belum ditentukan.

‘Beginning’ adalah nilai waktu yang merupakan nilai waktu paling awal dalam rentang waktu keberlakuan data atau nilai waktu paling jauh untuk masa lalu. Sedangkan ‘forever’ adalah nilai waktu paling akhir dalam rentang waktu atau nilai waktu paling jauh untuk masa depan. Nilai ini dapat digunakan jika data diasumsikan berlaku selamanya. Berdasarkan nilainya, ‘beginning’ hanya dapat digunakan untuk nilai *valid time* awal. Sedangkan ‘now’ dan

‘forever’ untuk *valid time* akhir. Berbeda dengan ‘now’ yang dapat dipandang sebagai sebuah variabel, ‘beginning’ dan ‘forever’ dipandang sebagai konstanta [JEN00].

Nilai ‘UC’ digunakan pada nilai *transaction time* akhir untuk menyatakan bahwa data masih berlaku di basis data dan belum dihapus secara logik. Nilai ini tidak pernah dimasukkan secara eksplisit oleh pengguna. Data yang baru dimasukkan dan belum diubah akan mempunyai nilai ‘UC’ pada nilai akhir *transaction time*. Secara konsep, ‘UC’ adalah semantik untuk ‘now’ pada *transaction time*. Jika dilakukan operasi *update* atau *delete* terhadap data tersebut, nilai akhir *transaction time* bagi data tersebut akan berubah, sehingga secara konseptual tidak akan ada nilai *transaction time* yang berakhir di masa depan atau lebih besar dari ‘UC’.

2.7 API (*Application Programming Interface*)

API (*Application Programming Interface*) adalah kumpulan *routine*, protokol, dan *tools* yang dapat digunakan dalam pembangunan aplikasi [STR06]. Aplikasi dapat dibangun menggunakan fungsi dan prosedur yang terdapat pada API untuk memudahkan pelaksanaan aksi tertentu. API bersifat *reusable* dan independen terhadap aplikasi.

API yang bagus memiliki beberapa karakteristik [BLO05]:

1. Mudah dipelajari
2. Mudah digunakan
3. Tidak mudah disalahgunakan
4. Kodenya mudah dibaca dan dipelihara
5. Cukup *powerful* untuk memenuhi *requirement*
6. Mudah di-*extend*

Langkah pertama dalam merancang API adalah menentukan *requirement*. Gambaran untuk *requirement* bisa didapatkan dari *use-case* dan sumber lain seperti masukan dari orang ditargetkan sebagai pengguna API yang akan dikembangkan. Pada awalnya, spesifikasi yang ditentukan untuk API sedikit tapi general, sehingga akan lebih mudah untuk dikembangkan. Langkah selanjutnya adalah desain, analisis, dan implementasi. Proses tersebut dan penentuan *requirement* dapat dilakukan secara paralel. Dari spesifikasi API yang dibuat berdasarkan *requirement*, mulai dikembangkan API secara sederhana. Dari proses pengembangan API ini, diidentifikasi kebutuhan tambahan yang dapat dimasukkan dalam spesifikasi dan juga spesifikasi yang ternyata tidak dibutuhkan dalam pengembangan. Ini dapat mencegah pengimplementasian dan pendefinisian spesifikasi yang tidak perlu dalam API, serta

mencegah munculnya hal-hal tak terduga seperti kesalahan atau perubahan requirement yang baru ditemukan setelah pengembangan API berjalan lama.

Prinsip umum desain API yang baik meliputi beberapa hal [BLO05]:

1. API harus dapat melakukan suatu hal dan melakukannya dengan baik. Fungsionalitas dari API harus mudah dijelaskan.
2. API harus dibuat sekecil mungkin, tetapi tidak lebih kecil dari *requirement*. Jika terdapat fungsionalitas yang meragukan, lebih baik tidak diimplementasikan. Ini disebabkan karena dalam pengembangan API, menambah lebih baik daripada menghilangkan.
3. Detail implementasi tidak boleh mempengaruhi API. Detail implementasi berhubungan dengan format dan hal lain yang berhubungan dengan implementasi secara spesifik. Mengimplementasikan detail ini tidak boleh mempengaruhi desain API.
4. Aksesibilitas terhadap kelas dan membernya bersifat minimal. Ini dilakukan dengan menerapkan konsep enkapsulasi. Kelas dan member yang tidak berhubungan langsung dengan pengguna API diusahakan tidak dapat diakses secara langsung dari luar atau bersifat *private*.
5. Penamaan perlu diperhatikan. Nama yang diberikan harus dapat memberikan penjelasan umum mengenai peranan atau fungsionalitasnya. Ini dapat dilakukan dengan menghindari penyingkatan kata secara kriptik, misalnya menggunakan kata “rmv” untuk “remove”.
6. API harus dapat berjalan baik dengan *platform*. Pengembangan desain API tidak boleh terpengaruh oleh *platform*, tetapi pengimplementasiannya harus disesuaikan dengan *platform*, misalnya dengan mematuhi *standard naming convention*, menghindari *return types* yang sudah tidak berlaku, dan dapat memanfaatkan fitur yang sudah ada dalam *platform*.