

BAB I

PENDAHULUAN

1.1 Latar Belakang

Basis data temporal adalah suatu bentuk pengayaan dari basis data relasional yang memperhitungkan aspek waktu. Aspek waktu penting karena saat ini banyak aplikasi basis data yang membutuhkan penanganan data yang berhubungan dengan waktu, seperti aplikasi yang menyimpan data historis dan aplikasi yang berhubungan dengan penjadwalan. Akan tetapi, teknologi basis data yang ada saat ini hanya menyediakan dukungan yang sedikit bagi penanganan data temporal [JEN00].

Dalam basis data temporal, terdapat dua dimensi waktu, yaitu *transaction time* dan *valid time*. *Transaction time* merupakan waktu keberlakuan data di basis data, sedangkan *valid time* adalah waktu keberlakuan data di dunia nyata. Relasi pada basis data temporal bisa merupakan relasi *transaction time* (relasi yang hanya mengandung dimensi *transaction time*), relasi *valid time* (relasi yang hanya mengandung dimensi *valid time*), dan relasi bitemporal (relasi yang mengandung kedua dimensi waktu tersebut). Relasi bitemporal dapat menampilkan data historis yang lebih jelas karena mencakup data historis di basis data dan di dunia nyata.

Terdapat beberapa pemodelan relasi bitemporal. Salah satunya adalah model relasi 1NF dengan *tuple stamping* yang menggunakan dua *chronon* (satuan waktu) terpisah yang menyimpan waktu awal dan waktu akhir dari interval keberlakuan data. Model ini mudah diimplementasikan pada basis data konvensional karena berbentuk 1NF. Contoh model relasi bitemporal ini dapat dilihat pada Gambar I-1.

NIP	Nama	Alamat	Gaji	Ts	Te	Vs	Ve
103	Heru	Jl. Delima 5	2500000	3/1/06	3/1/06	1/1/06	Now
103	Heru	Jl. Delima 6	2500000	4/1/06	2/1/07	1/1/06	Now
103	Heru	Jl. Delima 6	2500000	3/1/07	UC	1/1/06	31/12/06
103	Heru	Jl. Delima 6	3000000	3/1/07	UC	1/1/07	Now

Gambar I-1 Model 1 NF dengan *tuple stamping*

Atribut Ts dan Te merupakan nilai waktu awal dan waktu akhir dimensi *transaction time* dari suatu *tuple*. Sedangkan Vs dan Ve merupakan nilai waktu awal dan waktu akhir dari dimensi *valid time*. Relasi tersebut menggunakan interval waktu yang bersifat *closed ended*, artinya

atribut waktu akhir (T_e dan V_e) berisi nilai *chronon* terakhir saat data berlaku. Gambar I-1 memperlihatkan data historis seorang pegawai bernama Heru. Pada tanggal 3 Januari 2006, data Heru dimasukkan dengan alamat Jl. Delima 5 dan gaji 2.500.000. Gaji berlaku dari tanggal 1 Januari 2006 sampai waktu yang belum ditentukan. Tanggal 4 Januari 2006, data alamat Heru dikoreksi menjadi Jl. Delima 6. Ini mengakhiri keberlakuan data sebelumnya. Pada tanggal 1 Januari 2007, gaji Heru naik menjadi 3.000.000 dan belum berubah sampai sekarang. Data gaji yang baru ini dicatat pada tanggal 3 Januari 2007.

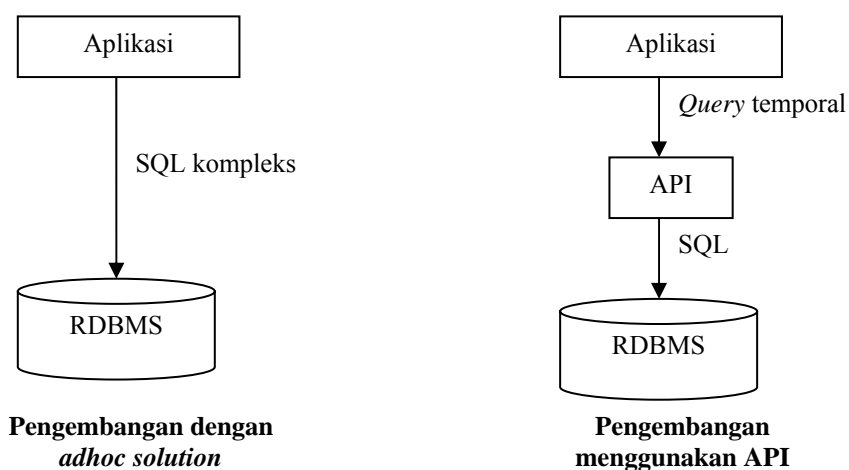
Ada dua pendekatan dalam arsitektur penyimpanan dan metode pengaksesan untuk data temporal, yaitu *integrated approach* dan *layered approach* [JEN00]. Pada *integrated approach*, modul internal pada DBMS konvensional dimodifikasi agar DBMS dapat menangani basis data temporal. Pada *layered approach*, DBMS konvensional tidak dimodifikasi, tetapi terdapat *layer* antara aplikasi dan DBMS yang menyediakan dukungan temporal [JEN00]. Konversi *query* dan model basis data dari temporal ke konvensional dan sebaliknya ditangani pada *layer* ini. *Integrated approach* memungkinkan sistem penyimpanan dan pengaksesan yang lebih efisien, tetapi implementasinya dapat memakan waktu lama. Arsitektur *layered approach* lebih mudah diterapkan karena dapat diimplementasikan terpisah dengan DBMS. Contoh aplikasi yang menggunakan pendekatan ini adalah TimeDB yang dikembangkan oleh Andreas Steiner. TimeDB 2.0 menyediakan dukungan bagi penanganan relasi *valid time* [STE99].

Sebagian besar pendekatan yang dilakukan dalam mengembangkan aplikasi yang berhubungan dengan basis data temporal adalah *ad hoc solution* [DET01]. Dalam hal ini, aplikasi mengakses DBMS konvensional secara langsung dan menggunakan *query* SQL yang kompleks untuk menangani aspek temporal. Fungsi yang dibutuhkan untuk penanganan aspek temporal ditulis pada program atau dibuat dalam basis data sehingga pengembangannya sangat spesifik terhadap kebutuhan dan tidak dapat digunakan untuk aplikasi lainnya. Dengan cara ini, implementasi penanganan aspek temporal dilakukan saat pengembangan aplikasi. Padahal, pengembangan aplikasi dapat lebih efisien jika telah tersedia fungsi dan prosedur yang dapat digunakan untuk operasi basis data temporal.

API (*Application Programming Interface*) adalah kumpulan *routine*, protokol, dan *tools* yang dapat digunakan dalam pembangunan aplikasi [STR06]. API menyediakan fungsi dan prosedur yang dibutuhkan untuk melakukan aksi tertentu dan memungkinkan tersedianya fungsi dan prosedur yang *reusable*. Hal ini dikarenakan karakteristik fungsi dan prosedur pada API bersifat general dan independen terhadap aplikasi, berbeda dengan *ad hoc solution* yang fungsi dan prosedurnya dibuat spesifik terhadap aplikasi sehingga tidak dapat digunakan

lagi. Karena itu, diperlukan adanya API untuk penanganan basis data temporal agar pengembangan aplikasi menjadi lebih efisien.

Dengan menggunakan API, konsep basis data temporal dapat diterapkan dalam membangun aplikasi. Lewat fungsi dan prosedur pada API, aplikasi dapat mengakses basis data dengan *query* temporal yang lebih sederhana dibanding SQL. Salah satu bahasa *query* temporal adalah TSQL2 yang merupakan pengembangan dari SQL92 yang ditujukan bagi operasi pada basis data temporal [SNO95]. Fungsi dan prosedur yang ada dalam API bersifat general sehingga bisa digunakan dalam pengembangan aplikasi yang berbeda-beda. Perbandingan antara pembangunan dengan *ad hoc solution* dan dengan menggunakan API dapat dilihat pada Gambar I-2. RDBMS hanya mendukung *user-defined time* atau dimensi waktu yang dianggap sebagai tipe data biasa. API menjadi sebuah *layer* yang akan melakukan penanganan tertentu terhadap tipe waktu yang diimplementasikan di basis data sehingga konsep *valid time* dan *transaction time* dapat digunakan pada aplikasi.



Gambar I-2 Perbandingan *ad hoc solution* dan pengembangan menggunakan API

1.2 Rumusan Masalah

Permasalahan yang akan dibahas dalam tugas akhir ini adalah sebagai berikut:

1. Pengidentifikasian hal-hal yang perlu ditangani oleh API agar RDBMS yang hanya mengenal *user-defined time* dapat menerapkan konsep *valid time* dan *transaction time*. Hal ini terkait dengan mekanisme konversi model data di level API dengan level DBMS yang mencakup konversi *query* dari *query* temporal ke *query* non-temporal, implementasi *primary key* dan *foreign key* temporal, dan konversi variabel serta konstanta waktu.

2. Pengidentifikasian kelas, fungsi, prosedur, dan keterkaitan antarkelas yang perlu diimplementasikan dalam API untuk menangani aksi yang dilakukan dalam operasi relasi bitemporal.

1.3 Tujuan

Tujuan utama dari pelaksanaan tugas akhir ini adalah menyediakan dukungan bagi penanganan operasi relasi bitemporal yang diimplementasikan dalam bentuk API. Sedangkan tujuan spesifiknya adalah sebagai berikut:

1. Memahami sintaks *query* yang digunakan untuk operasi basis data temporal dan sintaks *query* non-temporal yang bersesuaian.
2. Mendefinisikan cara pengkonversian *query* temporal ke *query* non-temporal yang digunakan pada RDBMS konvensional.
3. Mendefinisikan desain API yang mudah digunakan dan dikembangkan.
4. Mengimplementasikan dan mengujicobakan API untuk suatu aplikasi yang berhubungan dengan relasi bitemporal.

1.4 Batasan Masalah

Batasan dari pengerjaan tugas akhir ini adalah sebagai berikut:

1. API yang dikembangkan spesifik untuk satu DBMS, yaitu PostgreSQL. Bagian yang bersifat spesifik adalah *query* non-temporal, sedangkan desain API bersifat umum.
2. Operasi yang ditangani mencakup operasi dasar yaitu CREATE, INSERT, UPDATE, DELETE, dan SELECT...FROM...WHERE sederhana yang tidak mengandung *nested query*.
3. *Chronon* yang digunakan untuk *valid time* dan *transaction time* adalah hari.
4. Interval *valid time* dan *transaction time* bersifat *closed ended*.

1.5 Metodologi

Metodologi yang akan digunakan dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1. Studi literatur
Studi literatur yang dilakukan mencakup konsep relasi bitemporal, aturan sintaks *query* temporal, teknik pengimplementasian yang pernah dilakukan terkait basis data temporal, khususnya relasi bitemporal, dan dasar-dasar pengembangan API yang baik.

2. Analisis

Dalam tahap ini, dilakukan analisis mengenai *query* non-temporal yang bersesuaian dengan *query* temporal yang digunakan untuk melakukan operasi pada relasi bitemporal. Sintaks *query* temporal dan *query* non-temporal ini kemudian dibandingkan untuk menentukan metode konversi dari *query* temporal ke *query* non-temporal.

3. Pembangunan API

Tahap ini meliputi perancangan dan implementasi API. Pada tahap perancangan, dilakukan identifikasi kelas, fungsi, dan prosedur yang akan diimplementasikan beserta keterkaitan antarkelasnya. Hasil perancangan diimplementasikan menjadi API yang mencakup fungsi dan prosedur untuk melakukan operasi relasi bitemporal.

4. Pengujian

Dalam pengujian ini, API yang telah dikembangkan akan diterapkan dalam membangun suatu aplikasi sederhana yang berhubungan dengan relasi bitemporal.

1.6 Sistematika Pembahasan

Sistematika penulisan tugas akhir ini adalah sebagai berikut:

1. Bab I Pendahuluan, berisi pembahasan mengenai latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan dalam penyusunan tugas akhir ini.
2. Bab II Landasan Teori, berisi dasar teori terkait topik tugas akhir yang akan dijadikan landasan dalam perancangan dan implementasi API.
3. Bab III Analisis Penanganan Relasi Bitemporal, berisi analisis permasalahan yang dilakukan untuk mendapatkan solusi terkait topik tugas akhir.
4. Bab IV Pembangunan API, berisi perancangan API dan implementasinya.
5. Bab V Pengujian, berisi pengujian yang dilakukan terhadap API yang telah dibuat dalam tugas akhir.
6. Bab VI Kesimpulan dan Saran, berisi kesimpulan yang didapatkan dari pengerjaan tugas akhir dan saran terkait pengembangan topik tugas akhir.