BAB II

LANDASAN TEORI

2.1. Database dan Sistem Database

2.1.1 Database

Database (basis data) terdiri dari 2 (dua) kata, yaitu: *Basis* dan *Data*. *Basis* dapat diartikan sebagai markas atau gudang, tempat bersarang atau berkumpul. Sedangkan *data* adalah representasi fakta dunia nyata yang mewakili suatu obyek seperti manusia, barang, hewan, peristiwa, konsep, keadaan dan sebagainya, yang direkam dalam bentuk angka, huruf, simbol, teks, gambar, bunyi, atau kombinasinya (*Fathansyah, Basis Data, Informatika, Bandung 1999*).

Databases merupakan kumpulan dari data-data yang saling berhubungan yang terorganisasi sedemikian rupa sehingga mudah digunakan kembali. Database merupakan salah satu bagian yang terpenting dalam sistem informasi, karena merupakan dasar dalam menyediakan informasi bagi para pemakai. Disamping berisi/menyimpan data, setiap database juga mengandung definisi struktur (baik untuk database maupun objek-objeknya) seperti file/tabel, indeks dan lain-lain secara detail. Database dapat memberikan keuntungan pada data yang diolah, antara lain :

a. Kecepatan dan Kemudahan (*Speed*)

   Pemanfaatan database memungkinkan kita untuk dapat menyimpan data, melakukan perubahan atau manipulasi terhadap data, menampilkan kembali

data tersebut dengan lebih cepat dan lebih mudah daripada kita menyimpan data secara manual.

b. Efisiensi Ruang Penyimpanan (*Space*)

Dengan database, efisiensi atau optimalisasi penggunaan ruang penyimpanan dapat dilakukan, hal ini dikarenakan database mampu melakukan penekanan jumlah *redundansi* data, baik dengan menerapkan sejumlah pengkodean atau dengan membuat relasi (dalam bentuk file) antar kelompok data yang saling berhubungan.

c. Keakuratan (*Accuracy*)

Pemanfaatan pengkodean atau pembentukan relasi bersama dengan penerapan aturan atau batasan (*Constraint*) tipe data, domain data, keunikan data, dan sebagainya, yang secara ketat dapat diterapkan dalam sebuah database. Cara ini sangat berguna untuk menekan ketidakakuratan pemasukan atau penyimpanan data.

d. Ketersediaan (*Avaibility*)

Pertumbuhan data (baik dari sisi jumlah maupun jenisnya) sejalan dengan waktu akan semakin membutuhkan ruang penyimpanan yang besar. Akan tetapi tidak semua data itu akan selalu digunakan atau dibutuhkan. Dengan penggunaan database, data yang sudah jarang atau tidak pernah digunakan, dapat diatur dengan dilepaskan dari database yang sedang aktif. Di sisi lain, karena kepentingan pemakaian data, sebuah database dapat memiliki data yang tersebar di banyak lokasi.

e. Kelengkapan (*Completeness*)

Dalam sebuah database, disamping data kita juga harus menyimpan struktur (baik yang mendefinisikan objek-objek dalam basis data maupun definisi detail dari setiap objek, seperti struktur file atau tabel atau indeks). Dan juga dapat mengakomodasi kebutuhan yang terus berkembang terhadap data, selain dapat menambah *recoard* namun juga dimungkinkan menambahkan *field-field* baru pada suatu tabel data.

f. Keamanan (*Security*)

Dengan memanfaatkan keamanan pada database kita dapat menentukan siapa (pemakai) yang boleh menggunakan basis data beserta objek-objek didalamnya dan menentukan jenis-jenis operasi apa saja yang boleh dilakukan.

g. Kebersamaan Pemakaian (*Sharability*)

Pemanfaatan database dengan menggunakan aplikasi atau sistem yang *multi user* akan memenuhi kebutuhan banyak pemakai, namun tetap dengan menjaga atau menghindari munculnya inkonsistensi data kondisi *deadlock.*

2.1.2 Sistem Database

Penerapan database dalam sebuah sistem informasi disebut dengan sistem database (*database system*). Sistem database merupakan suatu sistem yang terdiri atas kumpulan file (tabel) yang saling berhubungan (dalam sebuah database disebut sistem komputer) dan sekumpulan program (DBMS) yang memunginkan beberapa

pemakaian atau program lain untuk mengakses dan memanipulasi file-file (tabel-tabel) tersebut.

Sebuah sistem database dapat memiliki beberapa database. Setiap database dapat berisi/memiliki sejumlah objek database seperti file/tabel, indeks dan lain-lain (*Fathansyah, Basis data, Informatika, Bandung 1999*).

Ada beberapa sistem yang dapat di gunakan dalam suatu aplikasi database, di mana masing-masing sistem memiliki kekurangan dan kelebihan. Sistem-sistem itu antara lain: sistem *single user*, sistem *multiuser* klasik dan sistem *client/server*.(*D. D. Prasetio. Membangun Aplikasi Web pada Sistem Database Terdistribusi, PT ElexMedia Komputino, Jakarta*)

a) Sistem *Single User*

Sistem *single user* merupakan sistem yang paling sederhana didalam pengaksesan suatu database. Dimana dalam sistem ini, program aplikasi dan file-file data diletakan pada komputer yang sama. Jadi dengan sistem ini hanya akan terdapat satu user yang akan mengakses data dalam waktu yang sama, hal ini disebabkan komputer tersebut tidak terhubung dengan komputer lain (*stand alone*).

b) Sistem *multiuser*

Sistem *multiuser* merupakan tahap pengembangan dari sistem *single user.* Pada sistem ini file-file data (*data source*) disimpan pada komputer lain yang biasanya disebut dengan komputer *server*. Hal ini bertujuan agar database dapat diakses secara bersama dengan *workstation* lainnya.

c) Sistem *Client/Server*

Client merupakan komputer yang berisi aplikasi, dan memiliki tanggung jawab untuk melakukan interaksi dengan user.

Server merupakan sebuah komputer yang berisi DBMS serta database, dan menjadi pusat seluruh kegiatan untuk menangani permintaan (*request*) yang dilakukan oleh *client.*

Sistem *client/server* merupakan sistem yang paling baik untuk digunakan, sistem ini mampu menghasilkan aplikasi yang paling baik untuk digunakan, sistem ini mampu menghasilkan aplikasi database yang tangguh dalam hal sekuritas, serta mampu mengurangi kepadatan lalu-lintas jaringan. Pada sistem ini terdapat dua aplikasi yang terpisah, yaitu aplikasi *client* dan aplikasi *server*. Pada aplikasi *client* membuat suatu permintaan (*make a request*) dan kemudian *server* merespon permintaan (*server responding*). Dengan adanya dua aplkasi yang terpisah tersebut, maka sebelum aplikasi *client* dapat terhubung ke *server*, maka aplikasi *server* harus dijalankan terlebih dahulu. Proses *query* pada sistem database *client/server* sangat sederhana, dimana aplikasi *client* cukup mengirimkan perintah (*statement*) SQL ke *server*. Perintah yang telah dikirim kemudian akan direspon oleh *server* dengan kembali mengirimkan hasil perintah yang diminta tersebut.

Sistem database *client/server* ini memungkinkan kuantitas data yang ditransmisikan dapat direduksi, sehingga waktu akses yang diperlukan menjadi sangat singkat.

2.2 MySQL

MySQL merupakan salah satu dari sekian banyak *Relational Database Management System* (RDMS), yang didistribusikan secara gratis di bawah lisensi GPL (*GNU Pablic Lisence*). Sehingga setiap orang mudah untuk mendapatkan dan bebas untuk menggunakan MySQL, dengan batasan tidak boleh dijadikan sebuah produk turunan yang bersifat *closed source* atau dikomersilkan. Meski sebelumnya dalam masalah lisensi ini, MySQL pernah mengalami perubahan, namun hingga saat ini lisensi tersebut sudah erat pada MySQL.

MySQL merpakan turunan dari salah satu konsep utama dalam database sejak lama, yaitu SQL (*Structured Query Language*). SQL sendiri adalah sebuah konsep pengoprasian database, terutama untuk pemilihan/seleksi dan pemasukan data, yang memungkinkan pengoprasian data dikerjakan dengan mudah secara *automatis*. Hal ini akan memudahkan pengguna database ketika ingin menggunakan kembali database yang ada.

MySQL merupakan terobosan solusi yang tepat dalam aplikasi database. Sifatnya yang *open source* serta dukungan jutaan komunitas pengguna di Internet, menjadikan MySQL sebagai *software* database yang cukup banyak digunakan. Tak hanya itu, kemampuannya yang bisa digunakan pada berbagai sistem operasi juga menjadikan MySQL sebagai *software* database pilihan. Selain itu juga tersedia *mailing list* dan *homepage* khusus yang memberikan tutorial serta dokumentasi lengkap.

Pada dasarnya, keandalan suatu sistem database (DBMS) dapat diketahui dengan melihat cara kerja *optimazer*-nya dalam melakukan proses perintah-perintah SQL, baik itu yang dibuat oleh *user* maupun oleh program-program aplikasinya. MySQL merupakan *database server* yang dapat dikatakan lebih unggul dibandingkan *database server* lainnya dalam *query* dan (*Riyanto. Koneksi Data Melalui Borland Delphi dengan Database MySQL, PT. Elex Media Komputindo, Jakarta, 2003*).

2.2.1 Sejarah MySQL

MySQL dikembangkan sekitar tahun 1994 oleh sebuah perusahaan pengembang *software* dan konsultan database bernama *MySQL AB* yang bertampat di *Swedia,* pada waktu itu perusahaan itu masih bernama *TeX DataKonsult AB,* dan tujuan awal dikembangkannya MySQL adalah untuk mengembangkan aplikasi berbasis web pada *client.*

Awalnya *Michael Widenius "Monty"* (pengembang satu-satunya di *TeX*) memiliki sebuah aplikasi UNIREG dan rutin ISAM buatannya sendiri dan sedang mencari *interface* SQL yang cocok untuk diimplementasikan ke dalamnya. Mula-mula *Monty* memakai miniSQL (mSQL) dalam eksperimentnya, namun mSQL dirasa kurang sesuai karena terlalu lambat dalam pemrosesan *query*.

Akhirnya *Monty* menghubungi *David Hughes,* pembuat mSQL yang sedang meliris versi kedua dari mSQL. Kemudian Monty mencoa sendiri mesin SQL yang memiliki interface mirip dengan mSQL, tapi dengan kemampuan yang lebih sesuai dan lahirlah MySQL.

Pengambilan nama MySQL, sampai saat ini masih belum jelas asal usulnya. Ada yang berpendapat nama My diambil dari huruf depan dan belakang *Monty*, tetapi versi lain mengatakan nama itu diambil dari nama putri *Monty* yang kebetulan bernama *My*.

2.2.2 Kelebihan MySQL

Menurut Riyanto dalam buku yang berjudul *Koneksi Data Melalui Borland Delphi dalam Database MySQL*, sebagai *software* database dengan konsep database modern, MySQL mempunyai banyak kelebihan, seperti :

1) *Portability*

MySQL dapat digunakan dengan stabil tanpa kendala yang berarti pada berbagai sistem operasi seperti Linux, Windows, FreeBSD, Mac OS X Server, Solaris, Amiga, HP-UX dan masih banyak lagi.

2) *Open Source*

MySQL didistribusikan secara *open source* (gratis), di bawah lisensi GPL, sehingga dapat diperoleh dan digunakan secara cuma-cuma tanpa dipungut biaya sepeser pun.

3) *Multiuser*

MySQL dapat digunakan untuk menangani beberapa *user* dalam waktu yang bersamaan tanpa mengalami masalah atau konflik. Hal ini memungkinkan sebuah *database server* MySQL dapat diakses *client* secara bersamaan dalam waktu yang bersamaan pula.

4) *Performance*

Dalam hal *performance,* MySQL memiliki kecepatan yang cukup menakjubkan dalam menangani *query-query* sederhana, serta mampu memproses lebih banyak SQL persatuan waktu.

5) *Column Types*

MySQL mendukung tipe kolom (tipe data) yang sangat kompleks, seperti *signed/unsigned integer, float, double, char, varchar, text, blob, date, time, datetime, timestamp, year, set,* serta *enum*. Bukan tidak mungkin tipe-tipe data tersebut akan terus dikembangkan oleh pengembang MySQL.

6) *Comma and Function*

MySQL mempunyai operator dan fungsi secara penuh yang mendukung perintah ***SELECT*** dan ***WHERE*** dalam *query.*

7) *Security*

Sekuritas sistem merupakan hal yang sangat diperhatikan pada MySQL. Terbukti dengan adanya beberapa lapisan sekuritas seperti *subnet mask,* nama *host,* dan izin ases *user* dengan sistem perizinan yang mendetail serta *password* terenkripsi.

8) *Scalability and Limits*

Dalam hal batas kemampuan, MySQL terbukti mampu menangani database dalam skala cukup besar, dengan jumlah *record* lebih dari 50 juta dan 60 ribu

tabel serta 5 miliar baris. Selain itu, batas indeks yang dapat ditampung mencapai 32 indeks pada setiap tabelnya.

9) *Connectivity*

MySQL mampu melakukan koneksi dengan *client* menggunakan *protocol* TCP/IP, *Unix soket* (Unix) atau *Named Pipes* (NT).

10) *Localstation*

Localstation merupakan salah satu kemampuan MySQL dalam mendeteksi pesan kesalahan (*error code*) pada *client* dengan menggunakan lebih dari dua puluh bahasa. Dengan demikian, pengguna cukup menyesuaikan bahasa yang digunakan.

11) *Interface*

Sama halnya dengan software-software database lainnya MySQL memiliki *interface* (antarmuka) terhadap berbagai aplikasi dan bahasa pemrograman dengan menggunakan fungsi API (*Aplication Prgramming Interface*).

12) *Client and Tools*

MySQL dilengkapi dengan berbagai tool yang dapat digunakan untuk administrasi database, dan pada setiap tool yang ada disediakan petunjuk *online*. Hal ini akan semakin memudahkan pengguna dalam menggunakan MySQL.

13) *Structure Tabel*

Struktur tabel MySQL cukup baik serta cukup fleksibel, misalnya ketika menangani **ALTER TABLE**, dibandingkan dengan database lainnya semacam PostgreSQL, ataupun Oracle.

Dengan keunggulan MySQL yang mampu berjalan pada berbagai sistem operasi semakin menjanjikan masa depan yang cerah, khususnya pada web database dimana posisi *web server* saat ini masih dipegang kuat oleh Linux dengan berbagai distronya.

Seiring dengan perkembangannya, MySQL mulai banyak dilirik pengguna database non-web. Selain tangguh dalam *flatform web*, MySQL juga telah memenuhi syarat sebagai database desktop. Terbukti banyaknya pengguna database seperti Yahoo! Finance, perusahaan sekuritas Aizawa bahkan Badan Antariksa Amerika Serikat (NASA) menggunakan MySQL untuk mengelola data keuangan dan transaksi bisnisnya.

2.2.3 Structured Query Languange (SQL)

Database merupakan kumpulan data yang setiap perintahnya menggunakan bahasa SQL. SQL merupakan bahasa ANSI (*America National Standart Institute*) yang digunakan untuk melakukan *query* data pada database. Hampir semua *software* database yang ada mengimplementasikan bahasa ini sebagai komponen utamanya.

SQL adalah bahasa database yang dipergunakan dalam menyelesaikan masalah-masalah dalam database. Perintah SQL memungkinkan pengguna database

untuk melakukan hal-hal seperti memasukan atau menambah *record* baru, mengeksekusi *query* database, mengambil data (*retrieve* data), mengubah *record*, serta menghapus *record*. Dengan kelebihannya dalam mengelolah data, saat ini SQL menjadi standart untuk bekerja dengan database *client/server*.

Fungsi-fungsi yang dimiliki MySQL :

1. Data *Definition*, unutk mendefinisikan struktur tabel, tipe data dari *field* sebuah tabel dan hubungan *field* dari tabel lain.
2. Data *Integrity*, untuk menjaga integritas data dengan berbagai konstrain.
3. Data *Retrieval* merupakan aplikasi *client* yang menggunakan SQL untuk meminta data dari *server*.
4. Data *processing*, untuk memberi hak akses *client* dengan SQL untuk menambah, menghapus, mengubah, data dari *server*.
5. *Security*, untuk memproteksi data dengan mendefinisikan hak akses dalam pengaksesan database.
6. *Concurent access* untuk mengatur pengaksesan secara simultan, sehingga satu pengaksesan tidak mengganggu pengaksesan lain.

Bahasa SQL memiliki struktur yang mudah dipahami karena menggunakan perintah-perintah dalam bahasa Inggris. Perintah SQL dapat diketikan dengan huruf besar maupun kecil, tanda titik koma digunakan sebagai tanda akhir dari suatu perintah SQL yang diketikan pada *prompt MySQL* (*Riyanto, Koneksi Data Melalui*

Borland Delphi dengan Database MySQL, PT. Elex Media Komputindo, Jakarta 2003).

2.2.3.1 Struktur Dasar

Sebuah ekspresi SQL dasar sebenarnya hanya terdiri atas 3 klausa, yaitu ***select***, ***from, where*** (*Fathansyah, Basis Data, Informatika, Bandug, 1999*).

a. Klausa ***select*** digunakan untuk menetepkan daftar atribut (*field*) yang diinginkan sebagai hasil *query*.

b. Klausa ***from*** digunakan untuk menetapkan tabel atau gabungan tabel yang akan ditelusuri selama *query* data dilakukan.

c. Klausa ***where,*** yang sifatnya opsional, digunakan sebagai predikat (kriteria) yang harus dipenuhi dalam memperoleh hasil *query*.

Sintak (cara penulisan) dari ekspresi SQL dasar dengan 3 klausa tersebut adalah :

select A1[,A2,…,An]

from t1 [,t2,…,tm]

[**where** P]

Dimana :

a. A1, A2, …, An merupakan daftar atribut

b. t1,t2,…, tm merupakan daftar tabel.

c. P merupakan predikat *query*.

d. [] merupakan tanda opsional (boleh digunakan, boleh tidak digunakan).

2.2.3.2 Fungsi Agregasi

Disamping menampilkan nilai-nilai atribut yang ada dalam tabel, sering pula ada kebutuhan untuk menampilkan data-data agregasi, seperti banyaknya *record*, total nilai suatu atribut, rata-rata nilai atribut, nilai atribut terbesar ataupun nilai atribut terkecil. Data agregasi tersebut dapat diperoleh dengan menggunakan fungsi-fungsi sebagai berikut:

a. ***Count*** untuk mendapatkan nilai banyaknya record hasil *query*

b. ***Sum*** untuk mendapatkan nilai total suatu atribut numerik hasil *query*

c. ***Avg*** untuk mendapatkan nilai rata-rata suatu atribut numerik hasil *query*

d. ***Max*** untuk mendapatkan nilai terbesar suatu atribut numerik hasil *query*

e. ***Min*** untuk mendapatkan nilai terkecil suatu atribut numerik hasil *query*

2.2.3.3 Manipulasi Data

SQL juga dilengkapi dengan sejumlah perintah untuk melakukan manipulasi data seperti penambahan *record* baru. Pengubahan nilai atribut dan penghapusan *record* di dalam suatu tabel.

a) Penambahan *record*

Untuk menambahkan *record* baru pada suatu tabel, contoh perintahnya adalah sebagai berikut:

Insert into mahasiswa (nim, nama, usia, alamat)

values(‘10104486’,’Affiar Akhyar’,’24’,’Jl. Pikitdro No. 34’);

atau

insert into mahasiswa **values**(‘10104486’,’Affiar Akhyar’,’24’,’Jl. Pikitdro No. 34’);

Nama-nama atribut sesudah nama tabel dapat diabaikan jika kita ingin mengisi nilai ke semua atribut dengan urutan atribut sesuai dengan pendefinisian struktur yang telah dilakukan pada saat pembuatan tabel.

b) Pengubahan *Record*

Untuk mengubah nilai atribut pada suatu *record* dari sebuah tabel menggunakan perintah **update.** Misalnya untuk mengubah nama mahasiswa dengan nim ‘10104485’ menjadi ‘M. Guntur’

update mahasiswa

set nama = ‘M. Guntur’

where nim = ‘10104485’;

Penggunaan klausa **where** sangat penting sekali dalam mengubah atribut tertentu. Hal ini untuk menghindari perubahan seluruh atribut yang ada dalam suatu tabel.

c) Penghapusan *record*

Seperti halnya dalam pengubahan *record*, dalam menghapus *record* caranya hampir sama, tetapi perintah yang digunakan adalah **delete.** Misalnya untuk menghapus *record* dalam table mahasiswa dengan kriteria nim=’10104487’

delete from mahasiswa

where nim = ‘10104487’;

Untuk menghilangkan semua *record* dari dalam tabel, klausa **where** dapat diabaikan (tidak ditulis).

2.2.3.4 Data Definition Language (DDL)

Data definition language (DDL) merupakan perintah yang berkaitan dengan pendefinisian objek-objek database. Objek-objek yang ada dapat berupa tabel, indeks, *view*, dan sebagainya. Perintah-perintah yang termasuk dalam *Data Definition Language* contohnya: ***create, alter***, dan ***drop***.

Sebagai contoh untuk membuat tabel baru, perintah yang digunakan adalah sebagai berikut:

Create table mahasiswa(

Nim **char**(7),

Nama_mhs **varchar**(30),

Alamat_mhs **varchar**(50),

Tgl_lahir **date**

);

2.4 JAVA

Java dikenal sebagai bahasa pemrograman tingkat tinggi yang berorientasi objek, atau lazim disebut dengan istilah *Object Oriented Programming* (OOP). Sejarah perkembangan Java dimulai pada tahun 1990 ketika *James Gosling*, seorang

ahli pemrograman yang bekerja di *Sun Microsystem*, ditugaskan untuuk menciptakan program untuk mengendalikan perangkat elektronik rumah tangga. Dimana program tersebut juga harus bersifat *multiplatform*, tidak bergantung pada satu *vendor*. Mula – mula bahasa pemrograman ini dinamakan **OAK** oleh *Jamees Gosling* yang mendapatkan inspirasi tersebut dari sebuah pohon OAK yang berada diseberang kantornya. Namun nama OAK sendiri merupakan nama bahasa pemrograman yang telah ada sebelumnya, maka kemudian Sun mengganti namanya menjadi Java. Nama Java sendiri terinspirasi saat Gosling dan rekan-rekan sedang menikmati secangkir kopi di sebuah kedai kopi yang kemudian tidak sengaja salah satu dari mereka menyebutkan kata Java yang mengandung arti kopi. Akhirnya mereka sepakat untuk memberikan nama bahasa pemrograman tersebut dengan Java.

Pada awal perkembangannya Java tidak menunjukan pangsa pasar bagi perusahaan SUN, karena Java ditujukan untuk pemrograman peralatan sederhana atau kecil. Selanjutnya Java diarahkan untuk perkembangan Internet.

Setelah melalui beberapa tahapan dan proses, SUN meluncurkan *browser* dari Java yang disebut *Hot Java* yang mampu menjalankan *applet*. Teknologi ini kemudian diadopsi oleh Netscape, sehingga memungkinkan program Java dijalankan pada *browser* Netscape. Beberapa waktu kemudian *Internet Explorer* juga mengadopsi teknologi tersebut, sehingga selain dapat dijalankan pada *browser* Netscape, program Java juga dapat dijalankan pada *browser* **Internet Explorer**.

Pada tahun 1996, SUN secara resmi merilis versi awal Java yang kemudian terus berkembang hingga muncul JDK 1.1 (*Java Development Kit versi 1.1*).

perkembangan terus hingga saat ini hingga muncul versi terbaru yang disebut Java 2. Pada Java 2 ini perubahan yang paling menonjol ialah adanya *swing* yang merupakan teknologi *Graphic User Interface* (GUI) yang mampu menghasilkan aplikasi *desktop* yang benar-benar portable. Selanjutnya pada tahun 1998-1999 dilncurkan kembali teknologi yang berbasis *Enterprise* yang disebut J2EE (*Java Enterprise Edition*). Teknologi J2EE ini diawali dengan *servlet* dan EJB kemudian diikuti JSP. Kelebihan Java cepat terdistribusikan serta memiliki kemampuan *multithreading*, menjadikan Java cepat popular di lingkungan *server*. Selain teknologi berbasis *server, applet*, dan desktop, Java juga mengembangkan teknologi berbasis *Mobile* yaitu J2ME (*Java Mobile Edition*). Teknologi ini memungkinkan untuk membuat aplikasi untuk ponsel seperti, grafis, maupun animasi.

2.4.1 Karakteristik dan Kelebihan Java

Java memiliki beberapa karakteritik yang menjadikan Java memiliki kelebian dibandingkan bahasa pemrograman yang lain. Adapun karakteristik tersebut adalah :

1. Sederhana

   Java adalah suatu bahasa sederhana. Para perancang Java berusaha untuk menciptakan suatu bahasa yang mudah dan cepat untuk dipelajari, sehingga bahasa Java dibuat dengan ukuran relatif kecil. Sintaks Java sama dengan bahasa pemrograman C++, namun dengan memperbaiki beberapa kekurangan dari C++, seperti mengurangi kompleksitas beberapa fitur,

menambahkan fungsi, serta menghilangkan beberapa hal yang menebabkan ketidakstabilan sistem pada C++.

2. Berorientasi Objek

Bahasa pemrograman saat ini mengarah ke bahasa pemrograman berorientasi objek. Rancangan berorientasi objek merupakan suatu teknik yang memusatkan rancangan pada data (objek) dan antarmuka. Fitur pemrograman berorientasi objek pada Java pada dasarnya sama dengan C++. Perbedaannya terletak pada kemampuan penurunan berganda (*multiple inheritance*) pada Java yang lebih baik dari C++.

3. Dapat Didistribusikan dengan Mudah

Java memiliki *library* rutin yang lengkap untuk dirangkai pada protokol TCP/IP, seperti HTTP dan FTP dengan mudah. Kemampuan jaringan Java lebih kuat dan mudah dipakai. Java memudahkan tugas-tugas pemrograman jaringan yang sulit, seperti membuka dan mengakases sebuah soket koneksi. Java bahkan memudahkan pembuatan CGI (*Common Gateway Interface*)

4. Kuat

Program yang dibuat dengan java dapat dipercaya dalam berbagai hal, karena Java banyak menekankan pada pengecekan awal untuk menghindari kemungkinan terjadi masalah, pengecekan pada saat *run-time*, dan mengurangi kemungkinan timbulnya kesalahan (*error*). Perbedaan utama antara Java dan C++ adalah Java memiliki sebuah model *pointer* yang dapat

mengurangi kemungkinan terjadinya penimpaan (*overwriting*) pada memori dan kerusakan data (*data corrupt*).

5. Aman

Java dimaksudkan untuk digunakan pada pemrograman jaringan dan sistem terdistribusi. Dengan demikian, penekanan terutama ditujukan pada masalah keamanan. Java memungkinkan untuk membuat program yang bebas virus dan sistem yang bebas dari kerusakan, karena Java membuat sistem yang mekanisme keamanannya benar-benar kuat. Suatu kelas (*class*) Java yang datang dari sumber yang tidak dapat dipercaya (*untrusted source*) memiliki akses sumber daya yang terbatas.

6. Portabel

Tidak seperti pada C dan C++, spesifikasi Java tidak terdapat aspek yang bergantung pada lingkungan implementasi. Ukuran tipe data primitif telah ditentukan sejak awal. Misal, “int” selalu berarti sebuah integer 32 bit dua komplemen bertanda (*signed two’s complement 32 bit integer*), dan “float” selalu berarti sebuah bilangan *floating point* 32 bit (mengikuti spesifikasi *IEEE* 754). Penentuan ukuran tipe data ini akan membuat keseragaman program pada setiap *platform*, tidak ada tipe data yang tidak kompetibel untuk semua arsitektur peranti keras dan peranti lunak.

7. *Multiplatform*

Java dirancang untuk mendukung aplikasi yang dapat beroperasi di lingkungan jaringan yang berbeda. Untuk mengakomodasi hal tersebut,

kompilator (*compiler*) Java membangkitkan kode *byte*, format yang tidak bergantung pada arsitektur tertentu yang didesain untuk mengirimkan kode ke banyak *platform* peranti keras dan peranti lunak secara efisien. Dengan slogan "*Write Once, Run Anywhere*" (WORA), Java banak dijalankan pada berbagai *platform* seperti Linux, Unix, Wondows, Solaris, maupun Mac.

8. *MultiThread*

Multithreading adalah kemampuan suatu program kompuiter untuk melakukan beberapa pekerjaan sekaligus, misalnya mencetak *life* sambil menjelajah Internet. Multithreading dalam Java sangat mudah dikelola. *Thread* dalam Java juga memiliki kemampuan untuk memanfaatkan kelebihan dari *multiprosesor* jika sistem operasi yang digunakan mendukung *multiprosesor.*

9. *Interpreter*

Interpreter Java dapat mengeksekusi kode *byte* Java secara langsung pada setiap mesin yang terdapat interpreter dan setiap *run-time* Java. Pada sistem Java, tahap *link* dari program adalah sederhana, bertahap, dan ringan. Hal ini membuat siklus pengembangan menjadi sangat cepat.

10. Kinerja yang Tinggi

Java dapat mencapai kinerja yang tinggi dengan cara mengadopsi sebuah skema yang memungkinkan *interpreter* dapat berjalan dengan kecepatan penuh tanpa perlu memeriksa lingkungan *run-time.* Pengumpul sampah otomatis (*automatic garbage collector)* bekerja sebagai *background thread*

prioritas rendah, menjamin ketersediaan memori ketika diperlukan dengan probabilitas tinggi untuk meningkatkan kinerja.

11. Dinamis

    Dalam beberapa hal, Java merupakan bahasa pemrograman yang lebih dinamis dibandingkan dengan C atau C++. Java dirancang untuk beradaptasi dengan lingkungan yang sedang berkembang. Walaupun kompilator Java ketat dalam proses kompilasinya, namun bahasa dan sistem *run-time* Java dinamis dalam tahap *linking*-nya. *Class* di-*link* hanya sebatas yang diperlukan saja. Jika diperlukan modul kode yang baru dapat di-*link* dari beberapa sumber, bahkan dari sumber dalam jaringan internet.

2.5 Replikasi

Replikasi merupakan penduplikasian atau pengkopian database pada beberapa lokasi yang berbeda. Suatu database yang direplikasi berarti database dari sebuah mesin sumber, atau biasanya disebut mesin *master* diduplikasi ke satu atau lebih mesin lain yang disebut mesin *slave*. Replikasi mempunyai manfaat untuk ketahanan, kecepatan, dan sistem administrasi *(http://www.mysql.com/manual)*, selain itu replikasi pada database *client/server* berbasis MySQL tidak memerlukan biaya yang mahal, karena tidak merubah sistem jaringan komputer yang sudah ada.

1. Ketahanan ditingkatkan dengan suatu *master/slave* setup. Ketika terjadi masalah pada mesin *master*, kita dapat beralih pada mesin *slave* sebagai *backup*.

2. Waktu response terhadap permintaan *client* yang lebih baik dapat dicapai dengan pemisahan beban antara mesin master dan mesin *slave.* **SELECT** *query* dapat dikirim ke mesin *slave* untuk mengurangi *query* yang banyak sekali diproses oleh mesin *master* sehingga mesin *master* dan mesin *slave* tetap berkaitan.
3. Manfaat lain yaitu dapat mem-*backup* menggunakan meesin *slave* tanpa mengganggu mesin *master*. Mesin master dapat terus memperbaharui data selagi *backup* dibuat.

Pada dasarnya prinsip kerja replikasi pada MySQL cukup sederhana, dimana meesin *master* me-log setiap *update* database pada sebuah file log biner. Kemudian mesin-mesin *slave* melalui *thread* (proses yang berisi sekumpulan intruksi yang menghubungkan dengan proses lain) replikasinya membuat koneksi ke *master*, dan mengejar *update* (modifikasi) yang terjadi pada *master* berdasarkan log biner tersebut. Untuk menggunakan replikasi dalam sebuah skema *load balancing* maka aplikasi harus memultipleksi *query,* yaitu mengirimkan *query* ke mesin *master* dan mesin *slave* secara bergantian.

Dalam tugas akhir ini akan diimplementasikan replikasi pada sistem database *client/server* yang diterapkan pada jaringan komputer lokal atau LAN (*Local Area Network*) dengan menggunakan MySQL *Server*, dimana *update* (*Write operation*) dapat dilakuakan dimesin *master* dan mesin *slave*. Adapun perangkat lunak yang

digunakan untuk mendukung replikasi dalam sebuah skema *load balancing* untuk menangani aplikasi-aplikasi Java menggunakan *Engine Tomcat.*

2.5.1 Bentuk Replikasi

Perbedaan paling menonjol dalam kasus replikasi adalah pada bentuk yang digunakan. Bentuk-bentuk replikasi yaitu: *full replication* dan *partial replication* (*D. D. Prasetyo, Membangun Aplikasi Web Pada Sistem Database Terdistribusi, PT. Elex Media Komputondo, Jakarta, 2004*).

1. *Full Replication*

Replikasi yang melibatkan seluruh database yang ada pada suatu lokasi disebut dengan *full replication.* Dalam bentuk ini seluruh database yang ada pada satu lokasi diduplikasikan ke beberapa lokasi *database server* lain. Hal ini dapat meningkatkan ketersediaan dengan baik sekali, karena sistem dapat melanjutkan operasi database selama sedikitnya ada satu lokasi yang aktif.

Disamping itu, bentuk *full replication* juga mampu meningkatkan performansi ketika melakukan *retrieve* (pengambilan) data pada *query global*, karena hasil query yang demikian dapat diperoleh secara lokal (ditempat itu juga) dari beberapa lokasi yang ada.

Namun bentuk ini menyebabkan lambatnya operasi modifikasi (*update*) secara drastis, karena melibatkan semua data pada lokasi yang ada.

2. *Partial Replication*

Partial Replication adalah replikasi yang hanya melibatkan beberapa database saja. Meskipun dalam ketersediaan data tidak sebaik *full replication*, namun mampu meningkatkan performansi dengan lebih baik. Hal ini dikarenakan hanya beberapa database saja yang terlibat, sehingga akan mengabaikan database lain yang tidak direplikasi.

2.5.2 Modifikasi Data pada Replikasi

Dalam proses modifikasi data pada replikasi, penarikan atau pengambilan kembali data (retrieve) yang direplikasi dapat dilakukan dengan dua cara, yaitu:

1. *Synchronous*

   Model *synchronous* memungkinkan pemrosesan perintah yang berupa permintaan (*request)* dilakukan pada sistem terdistribusi. Dalam hal ini semua data dimodifikasi selama terjadi proses transaksi, dan *request* berikutnya akan terkunci sampai *request* sebelumnya selesai dilakukan. Dengan demikian, konsistensi akan tetap terjaga dengan baik, namun akibatnya performansi akan berkurang dengan adanya penguncian *request* yang dilakukan.

2. *Asynchoronous*

   Pada model ini, modifikasi pada suatu data akan disebarkan pada lokasi lain dalam beberapa waktu kemudian setelah file sumber dimodifikasi (*update*) meski tidak dalam transaksi yang sama. Model ini memungkinkan terjadinnya pengaksesan data dari lokasi transaksi yang belum termodifikasi oleh *client.*

Akibatnya konsistensi akan kurang terjamin, namun performansi dapat lebih ditingkatkan.

2.5.3 Manfaat Replikasi

beberapa keuntukgan yang didapatkan dengan menggunakan replikasi pada sistem database, yaitu:

1. Meningkatkan Kinerja (*Performance Enhancement*)

   Hal ini dirasakan pada replikasi yang terjadi ketika *single database* sistem berurusan dengan banyak sekali operasi pembacaan *query* dari aplikasi *client* yang itu semua tidak dapat dilayani dengan baik.

2. Meningkatkan Ketersediaan (*enhanced avaibility*)

   Replikasi menawarkan banyak fungsi dalam kaitannya dengan ketersediaan dan keandalan pelayanan pada aplikasi *client.* Tersedianya *backup* database yang sam persis pada beberapa lokasi akan dapat meningkatkan ketersediaan data, sehingga jika salah satu *database server* tidak dapat beroperasi, maka masih ada *backup* database yang dapat digunakan.

3. Konsistensi (*Consistency*)

   Didalam sistem database terdistribusi (*distributed database*) dengan menggunakan replikasi, memungkinkan untuk memberikan hasil yang sama pada beberapa aplikasi *client*. Konsep ini sangat penting sekali ketika database diterapkan pada model *real system*, dimana ketika data yang berada pada

database lokal di-*update*, maka replikasi akan dapat melakukan sinkronisasi data.

4. *Replication Transparency*

   Ketika digunakan pada replikasi, transparansi berfungsi untuk menyembunyikan sistem konfigurasi replikasi dari *end user*, dengan demikian sangat tepat untuk keamanan sistem. Disamping itu juga sangat cocok untuk digunakan pada aplikasi antar muka *client* dengan sistem.

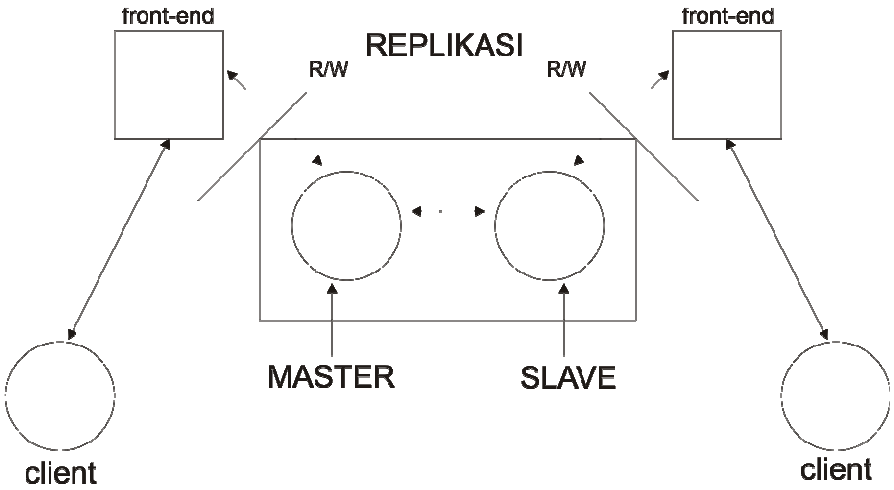
2.5.4 Arsitektur Replikasi

Model dasar arsitektur replikasi database memiliki tiga komponen saling berkaitan (*Bartolini G, 2001, Replication in MySQL, http://www.prato.linux.it/~gbartolini/pdf/mysql.pdf*), yaitu:

1. *Client Application*
2. *Front-end Application*
3. *Replication Manager*

Proses yang terjadi adalah aplikasi *client* mengirimkan permintaan *(request*), baik itu berupa operasi pembacaan (*read-only operation*) atau pun operasi *update* (*write operation*). Permintaan yang dikirimkan tidak akan langsung ditangani oleh *replica manager*, tetapi lebih dulu melalui *middle-tiernode* yaitu aplikasi *front-end*. *Replica manager* merupakan perangkat lunak utama yang digunakan pada replikasi, seperti MySQL server ataupun *database server* lainnya. Aplikasi *front-end*

merupakan aplikasi yang menghubungkan *client* dengan *replica manager*, sebagai contoh adalah aplikasi web dinamis.



Gambar 2.1. Skema sistem Replikasi *Slave/Master*

2.6 Load Balancing

Load balancing merupakan proses penyebaran beban yang seimbang pada sistem distribusi database. *Load balancing* menggunakan replikasi pada MySQL hanya akan menolong jika karakteristik *query* aplikasi didominasi oleh operasi pembacaan (*read query*) *(http://www.mysql.com/manual*).

Selain itu juga digunakan apabila ditemukan adanya penyebab gejala *overload*, yaitu pada aplikasi, desain database, hardware, dan *kernel*. (*D. D. Prasetyo, Membangun Web Pada Sistem Database Terdistribusi, PT. Elex Media Komputindo, Jakarta, 2004*)

1. Aplikasi

   Sebuah halaman web bisa jadi dibuat *on-the-fly* dari sejumlah *query*. Meng-*cache* (menyimpan pada *memory* sementara) hasil *query* akan sangat membantu, jika ini dimungkinkan.

2. Desain Database

   Lalai dalam mengindeks kolom yang menjadi kunci pencarian mengakibatkan *database server* akan melakukan pencarian sekuensial yang semakin mahal untuk ukuran tabel yang semakin besar.

3. Hardware

   Kurangnya RAM (*Random Access Memory*) dapat mengakibatkan *disk swapping* yang ekstensif. Problem pada *hardisk controller* juga dapat memunculkan gejala *overload.* Menggunakan beberapa *hardisk* lebih menguntungkan, karena tabel yang paling tinggi aksesnya dapat dipindahkan pada *hardisk* tersendiri untuk mengurangi efek *seek contention*.

4. *Kernel*

   Beberapa versi kernel dari Linux dan FreeBSD diketahui mengandung *BUG* pada implementasi SMP-nya, sehingga memungkinkan munculnya gejala *overload* ketika MySQL digunakan pada mesin *multiprocessor*.

Umumnya aplikasi-aplikasi yang ada (termasuk mungkin aplikasi yang biasa dibuat) tidak dirancang untuk menggunakan replikasi, sehingga diperlukan

modifikasi kode yang tidak sedikit untuk itu. Lain halnya jika sejak awal memang merancang aplikasi Java yang dibuat untuk mendukung fitur *load balancing* MySQL.

Untuk menghindari modifikasi kode aplikasi yang sudah jadi dan terlanjur kompleks, maka *query* multipleksing di implementasikan di layer antarmuka database dengan mempertahankan *API* yang ada. Namun demikian, desain antarmuka database di Java tidak begitu modular, sehingga tidak dapat membuat *wrapper layer* (lapisan pembungkus) dalam *Java Servlet* di atas (*API MySQL*).

Salah satu solusi untuk menerapkan skema *load balancing* pada replikasi database adalah dengan menggunakan perangkat lunak khusus yang dirancang untuk replikasi database, sebagai contoh *SQL Relay, Jakarta Tomcat* atau SQLB (*SQL load Balancer*)

2.7 Jakarta Tomcat

Untuk dapat menggunakan skema *load balancing*, maka diperlukan *software* khusus untuk *load balancer*. Salah satu software untuk *load balancer* yang mendukung pemprograman Java adalah *Tomcat*. atau biasa disebut *Jakarta Tomcat*.

Tomcat merupakan suatu *software* yang digunakan untuk menyatukan dan me-*load balancing* koneksi database secara terus menerus (*persistent*) pada sistem berbasis Unix, MacOs, Windows dan Linux. *Software* ini memiliki dukungan

terhadap berbagai *database server* seperti MySQL, Oracle, mSQL, PostgreSQL, Sysbase, IBM DB2, Interbase, MS SQL Server, Lago, serta SQLite.

Selain dukungan terhadap berbagai *software* database, Tomcat juga memiliki fitur-fitur yang cukup banyak, antara lain:

1. Memiliki dukungan terhadap berbagai *database software*.
2. Mendukung fasilitas replikasi database.
3. *Server-side*
4. Menyediakan port berbasis *non-Unix*.
5. Dapat digunakan dalam berbagai *platform* (Linux x86/PPC, SCO Open Server, Solaris, FreeBSD, OpenBSD, Windows serta NetBSD).

Cara kerja *Jakarta Tomcat* digambarkan dalam urutan-rutan proses sebagai berikut:

1. *Daemon* koneksi *Jakarta Tomcat* melakukan pencatatan serta mengelola *session* pada database.
2. *Daemon* melakukan identifikasi dengan mendengarkan koneksi *client* melalui *port* yang digunakan. Istilah ini disebut dengan *Listener*, dan port yang digunakan adalah *port* 8009 (*default*).
3. Apabila ada client yang terhubung dengan *Engine Tomcat* dan *daemon* koneksi tersedia, maka koneksi *client* tersebut akan ditangani dan dikelola.
4. Sebaliknya jika tidak ada daemon koneksi yang tersedia, permintaan yang diberikan oleh *client* akan ditunda dan *client* menunggu antrian.

5. Selanjutnya koneksi dari *client* ke database dilakukan melalui pengolaan *session* oleh *daemon*.