BAB 2
LANDASAN TEORI

Bab ini berisi mengenai dasar teori yang berhubungan dengan pengembangan sistem informasi pembayaran vakasi Fakultas Teknologi Industri Universitas Islam Indonesia (SIPeV FTI UII). Berdasarkan permasalahan yang ada maka dasar teori ini dapat dikelompokkan menjadi beberapa sub-bab, yaitu sistem informasi penggajian yang menjadi dasar dari SIPeV, *database* sebagai media penyimpanan dari SIPeV FTI UII, pemrograman *client/server* yang menjelaskan tentang dasar dari komunikasi data SIPeV FTI UII, dan *stored procedure* sebagai pengurang beban *resource* yang terpakai saat aplikasi dijalankan.

2.1 Vakasi

Menurut kamus besar bahasa indonesia (KBBI) vakasi adalah uang imbalan bagi penguji atau pemeriksa kertas, dapat diambil di bagian keuangan. Berdasarkan pengertian tersebut vakasi adalah uang yang didapat pegawai setelah pegawai melakukan tugas atau kewajibannya. Dalam hal ini kewajiban yang dimaksud adalah sebagai penguji atau pemeriksa kertas ujian, dalam kasus SIPeV vakasi akan disamakan dengan gaji dan upah yang didapat oleh pegawai di Fakultas Teknologi Industri (FTI).

Peraturan Pemerintah No. 8 tahun 1981 tentang perlindungan upah memberikan definisi upah sebagai berikut: "…..suatu penerimaan sebagai imbalan dari pengusaha kepada tenaga kerja untuk suatu pekerjaan atau jasa yang telah atau akan dilakukan, dinyatakan atau dinilai dalam bentuk uang yang ditetapkan menurut persetujuan atau perturan perundang-undangan dn dibayarkan atas dasar suatu perjanjian kerja antara pengusaha (pemberi kerja) dan pekerja termasuk tunjangan baik untuk pekerja sendiri maupun keluarganya (Rucky, 2006)."

Peraturan Menteri No. 3 tahun 1996 tentang Pemutusan Hubungan Kerja memberikan definisi yang lebih detail tentang upah karena ditujukan untuk keperluan perhitungan pasongan. Dalam peraturan menteri 3/96 ini yang dimaksud dengan upah mencakup: "upah pokok, segala tunjangan berkala dan teratur, harga pembelian dari catu yang diberikan kepada pekerja, penggantian untuk perumahan yang diberikan cuma-cuma, dan penggantian untuk pengobatan dan perawatan kesehatan (Rucky, 2006)."

Dalam praktek di perusahaan dan dalam buku-buku manajemen sumber daya manusia istilah upah dan gaji digunakan dalam konteks yang berbeda. Pertama, istilah upah digunakan untuk menggambarkan pembayaran jasa kerja untuk satuan waktu pendek, misalnya per hari atau malahan per jam. Gaji/vakasi menggambarkan pembayaran jasa kerja untuk satuan waktu lebih panjang biasanya sebulan.

Kedua, untuk menggambarkan kaitan pekerja penerima upah dengan proses produksi pada industri manufaktur. Upah atau dalam bahasa inggris disebut *wage* dibayarkan kepada pekerja yang terlibat langsung dalam proses produksi, baik terlibat langsung *(direct labor)* misalnya operator, maupun tidak langsung *(indirect labor)* misalnya bagian pemeliharaan mesin. Biaya untuk upah tersebut diikutkan langsung dalam perhitungan biaya produksi barang tersebut *(cost of goods sold)*. Sebaliknya, pekerja yang menerima gaji/vakasi termasuk dalam kategori *fixed costs* (biaya tetap) atau *overhead costs*. Contohnya adalah para supervisor atau manajer, staf administrasi. Dalam industri proses misalnya industri pupuk, kimia, semen yang sukar untuk menghitung komponen biaya personel per unit produksi, tidak terdapat pengelompokan penerima gaji atau upah.

Dalam usaha untuk menghilangkan ketidakadilan dalam pengupahan/penggajian ini Indonesia baru saja dengan bangga meratifikasi Konvensi ILO nomor 100 yang menetapkan bahwa tidak ada diskriminasi dalam pengupahan untuk pekerja yang mempunyai nilai yang sama. Yang dimaksudkan

oleh Konvensi ini adalah bahwa pekerja-pekerja yang mempunyai nilai atau bobot *(value)* yang sama harus mendapat imbalan yang sama. Sebaliknya, dari Konvensi ini dapat disimpulkan bahwa yang harus menjadi dasar untuk menetapkan apakah dua atau beberapa jabatan dinyatakan mempunyai nilai/bobot yang sama bukan hanya tingkat pendidikan formal yang dimiliki oleh orang yang melakukannya atau masa kerjanya, tetapi juga harus dinilai faktor-faktor lain yang relevan. Sebenarnya bila dikatakan imbalan yang sama persis adalah tidak tepat, tetapi yang dimaksudkan adalah dimasukkan ke dalam *job class/ job grade* (penggolongan jabatan) yang sama sehingga dengan demikian akan masuk juga ke dalam patokan upah yang sama.

Masalah utama dalam menjamin tercapainya keadilan melalui nilai jabatan menentukan cara yang dianggap tepat untuk menetapkan nilai/bobot tersebut, sehingga pekerjaan/jabatan X yang kemudian atas dasar kriteria tertentu diangap mempunyai nilai/bobot yang sama dengan jabatan Y akan mendapat imbalan yang sama. Pemberian nilai/bobot kepada sebuah jabatan/pekerja dilakukan melalui sebuah proses atau kegiatan yang biasa disebut evaluasi jabatan.

Dengan kata lain, evaluasi jabatan adalah sebuah proses yang dilakukan dalam ruang lingkup manajemen sumber daya manusia yang bertujuan menentukan nilai relatif dalam arti berat-ringannya suatu pekerjaan dibandingkan dengan pekerjaan-pekerjaan yang lain dalam sebuah organisasi.

2.2 Sistem Informasi Penggajian

Sumber daya keuangan merupakan bagian yang penting di dalam organisasi. Semua elemen organisasi mempunyai kepentingan terhadap sumber daya keuangan tersebut (Mulyanto, 2009). Dalam hal ini adalah pihak fakultas, jurusan, dan tiga divisi operasional meliputi divisi administrasi umum, divisi administrasi perkuliahan dan akademik, divisi administrasi keuangan sangat membutuhkan informasi mengenai keuangan yang ada di dalam Fakultas Teknologi Industri. Oleh karena itu, sumber daya keuangan harus dikelola dengan

baik agar dapat menghasilkan informasi yang berkualitas dan mendukung manajerial dalam mengambil keputusan.

Sistem informasi keuangan merupakan suatu sistem yang kompleks dan terdapat beberapa sub sistem, salah satu dari sistem yang berhubungan dengan sistem informasi keuangan adalah sistem informasi pembayaran gaji atau dalam hal ini adalah SIPeV. SIPeV bertujuan menangani data pembayaran dan memberikan informasi kepada personal unit struktural yang berada di FTI mengenai vakasi atau gaji yang didapat setelah melakukan tugas dan kewajiban dari fakultas.

2.3 Dasar dalam Membangun SIPeV

Dalam membangun aplikasi SIPeV ini terdapat beberapa teori pengembangan software yang digunakan yaitu media penyimpanan data atau database, media dan model integrasi yang mengacu pada pemrograman *client/server* serta *stored procedure* yang digunakan untuk mengurangi beban *resource* aplikasi. Selain itu terdapat 3 persyaratan sistem informasi pembayaran yand dikemukakan oleh (Howland, 1996):

1. Persyaratan Keamanan.

   Keamanan sistem harus jauh lebih ketat dikarenakan sistem pembayaran dapat dikatakan suatu hal yang rahasisa, oleh karena itu sistem yang harus dibuat dapat meminimalisir dari celah yang dapat akses oleh pemakai *(user)* yang tidak mempunyai kepentingan terhadap sistem. Hal itu dilakukan agar data-data dari sistem dapat terjaga keamanaannya.

2. Persyaratan Privasi.

   Sistem pembayaran harus benar-benar berorientasi untuk memberikan pembayaran yang aman, dan tidak mengungkapkan informasi asing dalam situasi apapun. Seperti halnya Informasi tambahan harus diberikan (seperti pada permintaan vakasi, ketika nama dan alamat akan diperlukan), untuk hal tersebut sangat diperlukan suatu privasi yang dapat mengatur akses informasi

kepada pihak yang berhak mengetahui informasi dan pihak yang tidak boleh mengetahui informasi tersebut. Fitur yang terpenting lebih mengarah pada sistem pembayaran itu sendiri dimana pengguna (*user*) menggunakan informasi tambahan untuk memenuhi kebutuhan bukan untuk melemahkan sistem.

3. Persyaratan Pelaksanaan.

   Dalam pelaksanaanya sebuah sistem pembayaran harus memenuhi aturan-aturan sebagai berikut:

   - Sistem harus sederhana

     Sistem secara keseluruhan harus sederhana, sistem yang sederhana dapat dengan mudah dipahami oleh pemakai, memberikan keyakinan yang lebih besar dalam keamanan. Selain itu, sebuah sistem sederhana akan lebih mudah dalam pengembangan sistem itu sendiri, dan mendorong lebih banyak implementasi.

   - Sistem harus didasarkan pengujian.

     Sistem harus melewati tahapan pengujian untuk mendapatkan tanggapan dari pengguna tentang keadaan sistem, sehingga sistem dapat dipakai oleh pengguna dan bukan memaksakan pengguna memakai sistem yang baru.

   - Sistem harus ditujukan pada pelayanan pengguna *(user)* bukan untuk memaksa pelanggan mengikuti aturan dari sistem sehingga mengetahui kelebihan dari sistem tersebut berdasarkan pada penggunaannya.

   - Biaya untuk mendapatkan aplikasi dan pembiayaan perawatan harus rendah.

2.3.1 *Database*

Database atau Basisdata merupakan komponen utama dalam membangun sebuah sistem yang menyangkut pendokumentasian data ke dalam sebuah *database* (Nugroho, 2005). *Database* adalah sebuah cara mendokumentasikan berbagai macam data yang kemudian diolah dengan sebuah sistem untuk selanjutnya disimpan dalam sebuah media penyimpanan. Dengan demikian data-data tersebut dapat diakses dengan mudah dan cepat. Dalam *database*, data yang ada tidak hanya diletakkan dan disimpan begitu saja dalam sebuah media penyimpanan, akan tetapi dikelola dengan sebuah sistem pengaturan *database* yang lebih dikenal dengan *Database Management System* (DBMS). Dengan begitu suatu data dengan jumlah besar dan kompleks dapat tersusun sangat baik sehingga memungkinkan pengaksesan data dengan mudah dan cepat oleh *user*.

2.3.1.1 *Database Management System* (DBMS)

Remakrishnan dan Geherke (2003) menyatakan *database management system* (DBMS) adalah perangkat lunak yang didesain untuk membantu memelihara dan memanfaatkan kumpulan data yang besar (Simarmata & Prayudi, 2006). Kebutuhan akan sistem termasuk pula penggunaannya yang berkembang pesat. Alternatif penggunaan DBMS adalah menyimpan data dalam file dan menulis kode aplikasi tertentu untuk mengaturnya. Penggunaan DBMS memiliki beberapa keuntungan dan kerugian.

1. Keuntungan menggunakan DBMS

   DBMS memungkinkan perusahaan maupun pengguna individu untuk :

   a) Mengurangi pengulangan data

      Apabila dibandingkan dengan *file-file* komputer yang disimpan terpisah disetiap aplikasi komputer, DBMS mengurangi jumlah total *file* dengan menghapus data yang terduplikasi diberbagai *file*. Data terduplikasi selebihnya dapat ditempatkan dalam satu *file*.

b) Mencapai independensi data

Spesifikasi data disimpan dalam skema pada tiap program aplikasi. Perubahan dapat dibuat pada struktur data tanpa mempengaruhi program yang mengakses data.

c) Mengintegrasikan data beberapa file

Saat file dibentuk sehingga menyediakan kaitan logis, maka organisasi fisik bukan merupakan kendala. Organisasi logis, pandangan pengguna, dan program aplikasi tidak harus tercermin pada media penyimpanan fisik.

d) Mengambil data dan informasi dengan cepat

Hubungan-hubungan logis, bahasa mannipulasi data, serta bahasa *query* memungkinkan pengguna mengambil data dalam hitungan detik atau menit.

e) Meningkatkan keamanan

DBMS mainframe maupun komputer mikro dapat menyertakan beberapa lapis keamanan seperti kata sandi *(password)*, direktori pemakai, dan bahasa sandi *(encryption)* sehingga data yang dikelola akan lebih aman.

2. Kerugian DBMS

Keputusan menggunakan DBMS mengikat perusahaan atau pengguna untuk:

a) Memperoleh perangkat lunak yang mahal.

DBMS *mainframe* masih sangat mahal. Walaupun harga DBMS berbasis komputer mikro lebih murah, tetapi tetap merupakan pengeluaran besar bagi suatu organisasi kecil.

b) Memperoleh konfiguradi perangkat keras yang besar

DBMS sering memerlukan kapasitas penyimpanan dan memori lebih besar daripada program aplikasi lain.

c) Mempekerjakan dan mempertahankan staf DBA.

DBMS memerlukan pengetahuan khusus agar dapat memanfaatkan kemampuannya secara penuh. Pengetahuan khusus ini disediakan paling baik oleh para pengelola basisdata (DBA).

Berdasarkan keuntungan dan kerugian DBMS tersebut dapat ditarik kesimpulan bahwa DBMS memberikan dasar-dasar menggunakan komputer sebagai suatu sistem informasi bagi spesialis informasi dan pengguna.

2.3.1.2 *Struktur Query Language* (SQL)

SQL adalah sebuah bahasa permintaan *database* yang terstruktur. Bahasa SQL dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam *database* maupun merelasikan antar *database*. Bahasa SQL ditulis langsung dalam sebuah program *database* sehingga seorang pengguna dapat melihat langsung permintaan yang diinginkan, sekaligus melihat hasilnya. MySQL sendiri adalah sebuah program *database server* yang memerlukan sebuah bahasa permmintaan dalam melayani permintaan *user*. Jadi, SQL-lah yang menjadi bahasa permintaan yang digunakan dalam MySQL SQL *(Structure Query Language)* dibagi menjadi dua bentuk *query*, yaitu:

1. DDL *(Data Definition Language)*

   DDL *(Data Definition Language)* adalah sebuah metode *query* SQL yang berguna untuk mendefinisikan data pada sebuah *database*, adapun *query* yang dimiliki adalah sebagai berikut:

   a) `CREATE`: digunakan untuk melakukan pembuatan tabel dan *database*.

   b) `DROP`: digunakan untuk melakukan penghapusan tabel maupun *database*.

   c) `ALTER`: digunakan untuk melakukan pengubahan struktur tabel yang telah dibuat, baik menambah *field (add)*, mengganti nama *field (change)* ataupun menamakannya kembali *(rename)*, serta menghapus *(drop)*.

2. DML *(Data Manipulation Language)*

   DML *(Data Manipulation Language)* adalah sebuah *query* yang dapat digunakan apabila DDL telah terjadi, sehingga fungsi dari *query* ini adalah untuk melakukan pemanipulasian *database* yang telah ada atau telah dibuat sebelumnya. Adapun *query* yang termasuk di dalamnya adalah sebagai berikut:

   a. `Select`

      Adalah perintah untuk memilih/menyeleksi menampilkan data yang diambiil dari suatu *table*.

   b. `Insert`

      Digunakan untuk malukan peng-*input*-an /pemasukan data pada tabel *database*.

   c. `Update`

      Digunakan untuk melakukan pengubahan atau peremajaan terhadap data yang ada pada tabel.

   d. `Delete`

      Digunakan untuk melakukan penghapusan data pada tabel. Penghapusan ini dapat dilakukan secara sekaligus (seluruh isi tabel) maupun hanya beberapa *recordset*.

Kedua bentuk *query* SQL tersebut dapat digunakan pada semua model basis data yang mendukungnya, baik yang berbasis Unix maupun Windows. Di dalam Windows misalnya, *user* dapat mengimplementasikan pada program penghasil *database* seperti Microsoft Acces atau pada semua *database* bawaan program visual seperti *database desktop* yang bernama Paradox.

2.3.1.3 Arsitektur *Database System*

Secara garis bersar, *database* sistem terdiri dari tiga buah komponen utama, masing-masing sebagai berikut.

1. Aplikasi

   Komponen ini bertanggung jawab dalam memformat dan mempresenting data pada pengguna atau alat keluaran dan mengendalikan masukan pengguna dari *keyboard* atau alat input lainnya. Menyajikan data dalam bentuk tampilan atau *interface* yang terbagi menjadi empat *layer* yaitu *user interface*, *data interface*, *transaction interface*, dan *external access interface*.

2. *Database Engine* sebagai *bussiness logic*

   Komponen ini berguna untuk menangani logika pemprosesan data (data processing logic), logika aturan bisnis (business rule logic), dan logika manajemen data (*data management logic*). Proses *data logic* merupakan aktifitas untuk memvalidasi data dan mengindentifikasi proses eror pada data. *Business rule* tidak mempunyai kode pada DSMS, tetapi mempunyai kode pada komponen pemprosesan. *Data management logic* mengindentifikasikan data yang diperlukan/penting untuk memprosesan transaksi atau *query*.

3. *Database* fisik

   Komponen ini bertanggung jawab pada penyimpanan data dan perbaikan data dari alat penyimpan yang bekerja dengan aplikasi.

Dari ketiga komponen diatas dapat digambarkan sebuah tahapan atau layer, seperti pada Gambar 2-1 (Darmayuda, 2007).

2.3.1.4 Database Engine dan Model Pengaksesan Data

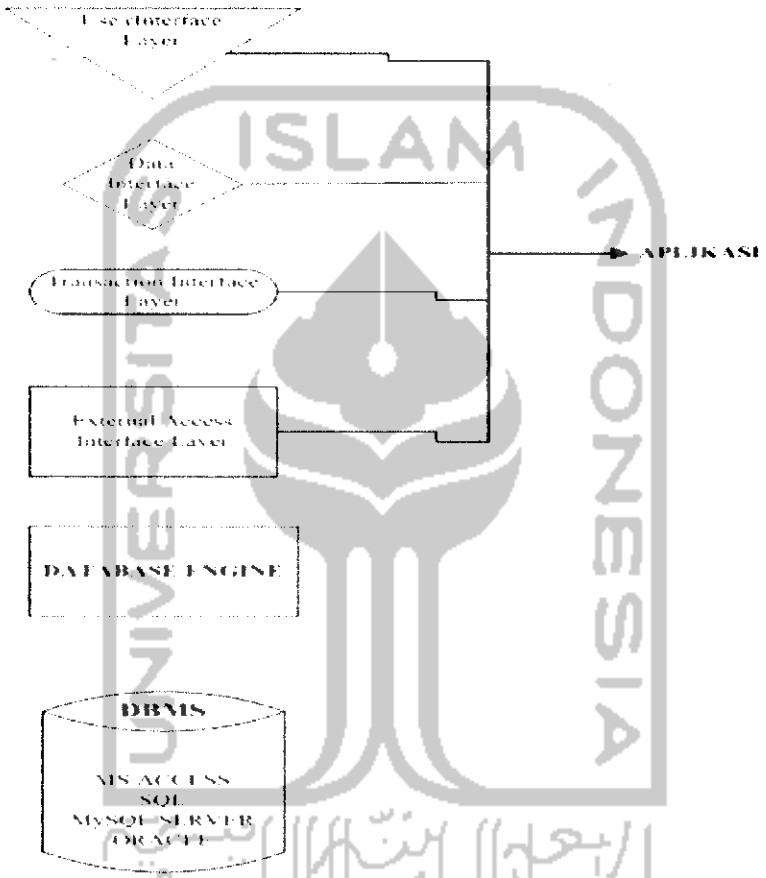
Bagian penghubung antara aplikasi dengan *database* adalah *database engine*. Bagian ini yang bekerja untuk mengatur lalu lintas data dari dan ke pemakai aplikasi. Pada skenario microsoft, terdapat dua jenis *database engine* yang dapat digunakan oleh berbagai jenis aplikasi. *Database engine* tersebut, adalah:

1. *Jet Database Engine*

   Jet Database Engine ditujukan bagi aplikasi yang ingin mengakses sekelas *database* desktop (digunakan untuk *database* menengah ke bawah, misalnya *database* Microsoft *Access*).

2. *SQL Server Engine*

   SQL Server Engine ditujukan bagi aplikasi yang ingin mengakses *database* bersekala besar, dengan jumlah pemakai mencapai ribuan. *SQL Server Engine* ini dirancang dengan arsitektur *client/server* dan ditujukan untuk *database* berukuran menengah ke atas (*Enterprice*).



Gambar 2-1 Arsitektur *Database System*

Untuk memanfaatkan layanan-layanan yang disediakan oleh *database engine*, Dapat menggunakan sejumlah model *object* pengaksesan data yang telah dipersiapkan oleh *Microsoft*. Model-model *object* tersebut meliputi:

1. DAO *(data access object)*

   Data access object, adalah model pengaksesan object paling tua, di mana *engine* yang dapat diaksesnya hanya *jet enggine.*

2. RDC *(remote data control)* dan RDO *(remote data object).*
   Remote data *object*, merupakan perbaikan dari DAO dan ODBC (*Open Database Connectivity*), sebagai contoh untuk *database* SQL *Server* dan oracle.

3. *ActiveX* data *object*, merupakan model pengaksesan object yang menggunakan hirarki atau *overhead* lebih kecil, lebih cepat *database desktop*, sampai *database server*. Selain ODBC, ADO dapat mengakses sumber data lain, yaitu OLE DB (*object linking*).

2.3.2 Pemrograman *Client/Server*

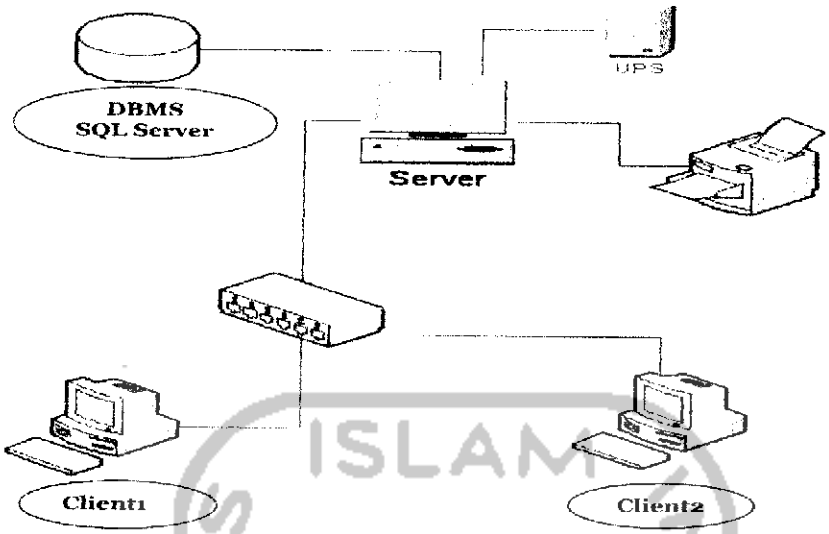
Pemrograman *client/server* adalah sebuah dasar dari pengembangan perangkat lunak dari integrasi data dalam lingkungan jaringan komputer. Program yang terdiri dari beberapa bagian yaitu *user interface, busines logic,* dan data *access* yang semuanya dibuat dalam satu program, pola ini berkembang pada aplikasi *Personal Computer* (PC), pada perkembanganya sistem membutuhkan akses data bersama (*shared resource*) melalui teknologi jaringan, sumber daya seperti *disk* dan file dapat diakses bersama, model komputasi ini dikenal dengan nama *disk server* atau *file server* (Darmayuda, 2007). Model komputasi ini sangat penting karena kebutuhan aplikasi akan kecepatan data dan pengaksesan data. Data lebih cepat dipertukarkan melalui teknologi jaringan daripada melalui tangan ke tangan. Untuk mendukung model komputasi ini dibutuhkan bebarapa *database* yang mempunyai kelebihan *share data,* yaitu *database* yang mempunyai fasilitas dapat mengirim data yang diminta aplikasi melalui teknologi jaringan. Misalnya *database* yang mendukung model komputasi ini antara lain Mysql dan *SqlServer.* Mekanisme alur dari teknologi *client/server* adalah adanya keterkaitan fungsional antara *client/server.* Menurut Darmayuda (2007) *Database sever* menerima request (permintaan) berupa instruksi SQL *(Structure Query Language),* dan sebagai hasil *database server* akan menjawabnya, berhasil atau tidaknya perintah tersebut. Dalam perintah tersebut akan menuntut lokasi pemrosesan, sedangkan lokasi pemrosesan terbagi menjadi dua, yaitu:

1. Komputer Pertama mengolah *input output* ke pemakai (*user interface*). Disebut juga sebagai Presentasi. Di komputer tersebut *program logic* berjalan dan berhubungan dengan *user interface* dan *data interface (*dalam hal ini berbentuk SQL-*Request)*

2. Komputer Kedua menerima SQL-*Request*, memprosesnya dengan mengakses *database*, dan memberikan hasilnya kembali.Sebagai contoh, Komputer Pertama disebut sebagai *client (front end)*, yang terdiri atas aplikasi yang ditulis dalam bahasa pemrograman, misalnya dalam tugas akhir ini digunakan bahasa pemrograman Borland Delphi 2007. sedangkan Komputer Kedua disebut sebagai *server (Back End)*", dalam hal ini adalah *Database Management System* (DBMS).

2.3.2.1 Arsitektur Sistem Komputer

Dalam arsitektur pertukaran dan akses data yang terjadi dalam pemrograman *client server*. Dalam hal ini terdapat 3 buah komponen utama yang tidak bisa dipisahkan yaitu *client, server,* dan DBMS atau *database.* Data yang terimpan di database, akan ditempatkan dalam komputer *server* di jaringan tersebut. Dari *client* untuk mendapatkan data-data yang diperlukan akan menghubungi komputer *server* dimana data tersebut disimpan, selanjutnya komputer *server* akan memproses pengambilan data dari database dan selanjutnya mendistribusikan ke komputer *client* sehingga komputer *client* mendapatkan data-data tersebut, seperti pada Gambar 2-2.



Gambar 2-2 Arsitektur Komputer

Untuk membuat *database* sistem bekerja sesuai dengan volume atau skala transaksi dan pemakainnya, terlebih dahulu menentukan arsitektur yang akan diimplementasikan pada sistem tersebut. Arsitektur dalam *database* sistem terbagi dalam tiga *tier*. Istilah *tier* untuk menjelaskan pembagian sebuah aplikasi yang melalui client dan server. Pembagian proses kerja adalah bagian utama dari konsep *client/server* saat ini. Saat ini pembagian kerja pada *client* dan *server* telah diatur secara lebih spesifik. Masing-masing *tier* dalam dalam arsitektur *database* adalah sebagai berikut:

1. *Arsitektur Single-tier*

   Arsitektur *Single-tier* merupakan arsitektur yang paling sederhana dimana seluruh komponen *database* sistem secara lohis dikelolmpokan ke dalam sebuah *tier*, contoh: sistem *stand-alone desktop database.*

2. *Arsitektur* Two-tier

   Arsitektur *Two-tier* merupakan arsitektur dimana *database engine* dan *database* terletak pada tier yang berbeda dengan aplikasi. Arsitektur ini disebut sistem *client/server database* sistem seperti MySQL dan SQL *Server.*

3. *Arsitektur n-tier*

   Arsitektur n-tier merupakan arsitektur dimana memisahkan seluruh komponen ke dalam tier-tier yang berbeda untuk menanggulangi masalah skabilitas sistem yang semakin membesar. Pada arsitektur teknologi yang biasanya digunakan adalah *Microsoft Transaction Service (MTS)* dan *Distributed Compoonent Object Model (COM)*, serta *COM +*.

2.3.3 *Stored Procedure*

Stored prosedure adalah sekumpulan perintah-perintah umum SQL berbentuk *script* yang berhubungan dengan *database* yang disimpan di *server*.

Stored prosedure akan meningkatkan *performance database*, keamanan *database*, dan memungkinkan membuat program untuk *database*. *Stored procedure* dapat dikategorikan menjadi beberapa jenis (Darmayuda, 2007), yaitu :

1. *Stored Procedure System*

   Stored Procedure yang disimpan di *database Master*.

2. *Stored Procedure Lokal*

   Stored procedure lokal biasanya disimpan dalam *database user* yang memang digunakan untuk kepentingan *database* yang bersangkutan.

3. *Stored Procedure Temporer*

   Mirip dengan stred procedure lokal tetapi akan ada sampai koneksi yang membuatnya berhenti/ SQL *Server* dimatikan. Terdapat 3 jenis *stored procedure temporer*. Yaitu lokal (privat) yang diawali dengan #, global yang diawali dengan ##, dan yang langsung dibuat di dalam tempdb.

4. *Stored Procedure Extended*

   Stored procedure ini memakai program eksternal dan dikompilasi sebagai *Dynamic Link Library* (DLL) 32 *bit* utnuk menambahkan kemampuan *stored procedure*.

Function *OBJECT PROPERTY* dipakai untuk menentukan apakah sebuah stored procedure adalah *extended*. Jika hasilnya adalah 1 maka *extended*, jika hasilanya 0 bukan.

5. *Stored procedure remote*

   Stored procedure ini berada pada instalasi sel lain. Jenis dipakai untuk kompilasi dengan versi sebelumnya.

Dalam MySQL versi 5.0 mampu mendukung *stored procedure, triggers, views, information schema. Stored procedure* mempunyai banyak keunggulan yaitu :

1. Membagi beban *resource* yang terpakai saat aplikasi di jalankan. Apabila semua query dijalankan pada aplikasi/*client* maka *resource* yang terpakai pada client tersebut akan besar, untuk mengatasi hal tersebut perintah *query* tersebut dibuat pada *procedure* ataupun *view* yang dieksekusi pada sisi server.

2. Mempermudah dalam menampilkan data, terkadang struktur *database* yang dibuat mempersulit pekerjaan dalam proses *query* dikarenakan kebutuhan akan proses spesial *reporting* maka pembuatan `view` dapat dijadikan alasan untuk mempermudah reporting.

3. Mempermudah *maintain* aplikasi, apabila ada proses *query* yang sama dan berulang maka digunakan *store procedure* akan membuat lebih sederhana dalam proses pembuatan aplikasi.

2.3.3.1 *Trigger*

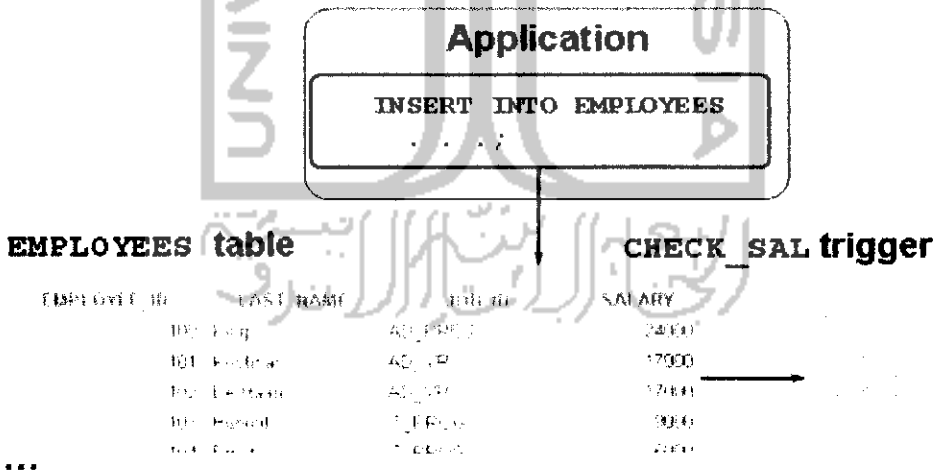
Trigger adalah blok PL/SQL atau prosedur yang berhubungan dengan table, *view*, skema atau database yang dijalankan secara implicit pada saat terjadi sebuah event (Badriyah, 2007). *trigger* dapat dibedakan menjadi 2 tipe, masing-masing adalah :

1. Application *trigger* : diaktifkan pada saat terjadi *event* yang berhubungan dengan sebuah aplikasi

2. Database *trigger* : diaktifkan pada saat terjadi event yang berhubungan dengan data (seperti operasi DML) atau event yang berhubungan dengan sistem (semisal *logon* atau *shutdown*) yang terjadi pada sebuah skema atau *database*.

Trigger dibuat sesuai dengan keperluan, Ada kalanya trigger perlu dibuat, dan kadangkala tidak perlu dibuat. Pembuatan *Trigger* diperlukan untuk membentuk sebuah aksi tertentu terhadap suatu event, selain itu *trigger* juga dibuat untuk memusatkan operasi global. Trigger tidak perlu dibuat, apabila fungsionalitas yang diperlukan suatu ada pada Oracle server dan duplikat atau sama dengan fungsi trigger yang lain.

Prosedur bisa dibuat dalam *database*, kemudian prosedur tersebut dipanggil pada *trigger*. Jika penggunaan *trigger* terlalu berlebihan, maka akan menyebabkan terjadi sifat ketidaktergantungan yang terlalu kompleks sehingga akan mempersulit pemeliharaan dari aplikasi yang besar. Gambar 2-3 menunjukkan ilustrasi dari penggunaan trigger (Badriyah, 2007) :



Gambar 2-3 Ilustrasi dari Penggunaan Trigger

Pada gambar tersebut, database *trigger* `check_sal` memeriksa nilai gaji pada saat suatu aplikasi mencoba untuk memasukkan baris baru ke dalam table `employees`. Nilai yang terletak pada jangkauan di luar kategori pekerjaan akan diabaikan. Sebuah *trigger* mempunyai enam komponen yaitu :

1. *Trigger timing*

   Trigger timing adalah waktu kapan *trigger* diaktifkan. Ada tiga macam *trigger* timing, yaitu :

   a) `BEFORE` : *trigger* dijalankan sebelum DML event pada tabel

   b) `AFTER` : *trigger* dijalankan setelah DML event pada tabel

   c) `INSTEAD OF` : *trigger* dijalankan pada sebuah *view*.

2. *Trigger event*

   Trigger event ada 3 kemungkinan : `insert`, `update` atau `delete`. Pada saat *trigger event* `update`, *user* dapat memasukkan daftar kolom untuk mengidentifikasi kolom mana yang berubah untuk mengaktifkan sebuah *trigger* (contoh : `update of salary ...`). Jika tidak ditentukan, maka perubahannya akan berlaku untuk semua kolom pada semua baris.

3. Nama tabel

4. Tipe *trigger*

5. Klausa `when`

6. *Trigger body*

Secara garis besar *trigger* ada 2 macam tipe dalam pengeksekusiannya, yaitu :

1. Statement : *trigger* dijalankan sekali saja pada saat terjadi sebuah event. Statement *trigger* juga dijalankan sekali, meskipun tidak ada satupun baris yang dipengaruhi oleh event yang terjadi.

2. Row : *trigger* dijalankan pada setiap baris yang dipengaruhi oleh terjadinya sebuah event. Row *trigger* tidak dijalankan jika event dari *trigger* tidak berpengaruh pada satu baris pun. *Trigger* body mendefinisikan tindakan yang

perlu dikerjakan pada saat terjadinya event yang mengakibatkan sebuah *trigger* menjadi aktif.

2.3.3.2 *View*

View merupakan tampilan data yang terdapat pada satu *table* atau beberapa *table* yang saling berhubungan. Fungsi utama membuat `view` adalah agar data pada *table* aslinya tidak dapat diakses secara langsung. Selain itu `view` dapat dijadikan pengurangan beban resource yang terpakai aplikasi karena aplikasi tidak secara langsung memangil semua *record*, tetapi record telah di *buffer* (disimpan sementara) dalam `view`, selanjutnya aplikasi memanggil data yang terdapat dalam *view*.

UNIVERSITAS ISLAM INDONESIA