# • LINGKUNGAN DATABASE

Tujuan utama dari sistem database adalah menyediakan pemakai melalui suatu pandangan abstrak mengenai data, dengan menyembunyikan detail dari bagaimana data disimpan dan dimanipulasikan. Oleh karena itu, titik awal untuk perancangan sebuah database haruslah abstrak dan deskripsi umum dari kebutuhan-kebutuhan informasi suatu organisasi harus digambarkan di dalam database.

Lebih jauh lagi, jika sebuah database merupakan suatu sumber yang bisa digunakan bersama maka setiap pemakai membutuhkan pandangan yang berbeda-beda terhadap data di dalam *database*.

**Tiga Tingkatan Arsitektur *Database* ANSI-SPARC**

Ada 3 tingkat dalam arsitektur *database* yang bertujuan membedakan cara pandang pemakai terhadap *database* dan cara pembuatan *database* secara fisik.

3 tingkatan arsitektur *database* :

1. *Tingkat Eksternal (External Level)*

   Tingkat eksternal merupakan cara pandang pemakai terhadap *database*. Pada tingkat ini menggambarkan bagian *database* yang relevan bagi seorang pemakai tertentu. Tingkat eksternal terdiri dari sejumlah cara pandang yang berbeda dari sebuah database. Masing-masing pemakai merepresentasikan dalam bentuk yang sudah dikenalnya. Cara pandang secara eksternal hanya terbatas pada entitas, atribut dan hubungan antar entitas (*relationship*) yang diperlukan saja.

   Contoh : viw dari mahasiswa, view dari mata kuliah

2. *Tingkat Konseptual (Conseptual Level)*

   Tingkat konseptual merupakan kumpulan cara pandang terhadap *database*. Pada tingkat ini menggambarkan data yang disimpan dalam *database* dan hubungan antara datanya.

   Hal-hal yang digambarkan dalam tingkat konseptual adalah :

   - semua entitas beserta atribut dan hubungannya
   - batasan data
   - informasi semantik tentang data
   - keamanan dan integritas informasi

   Semua cara pandang pada tingkat eksternal berupa data yang dibutuhkan oleh pemakai harus sudah tercakup di dalam tingkat konseptual atau dapat diturunkan dari data yang

ada. Deskripsi data dari entitas pada tingkat ini hanya terdiri dari jenis data dan besarnya atribut tanpa memperhatikan besarnya penyimpanan dalam ukuran byte.

Contoh : entity, relationship, tipe data dan constraint.

3 *Tingkat Internal (Internal Level)*

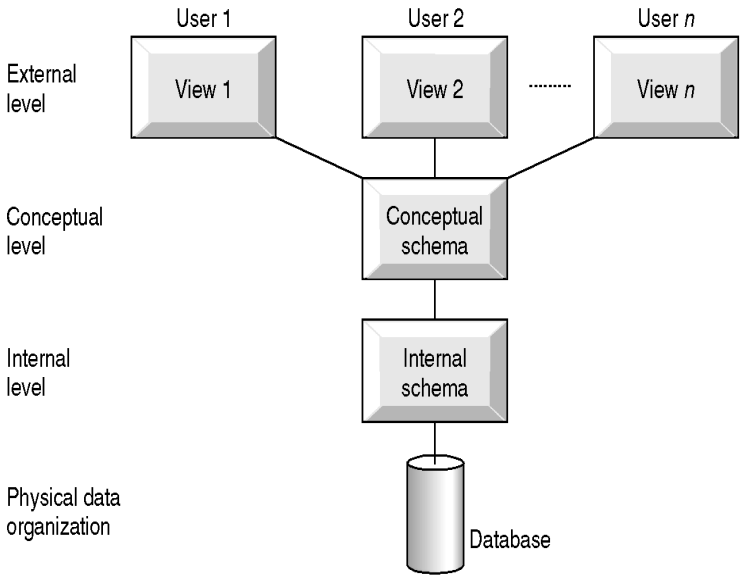
Tingkat internal merupakan perwujudan *database* dalam komputer. Pada tingkat ini menggambarkan bagaimana *database* disimpan secara fisik di dalam peralatan *storage* yang berkaitan erat dengan tempat penyimpanan/*physical storage*.

Tingkat internal memperhatikan hal-hal berikut ini :

- alokasi ruang penyimpanan data dan indeks
- deskripsi record untuk penyimpanan (dengan ukuran penyimpanan untuk data elemen

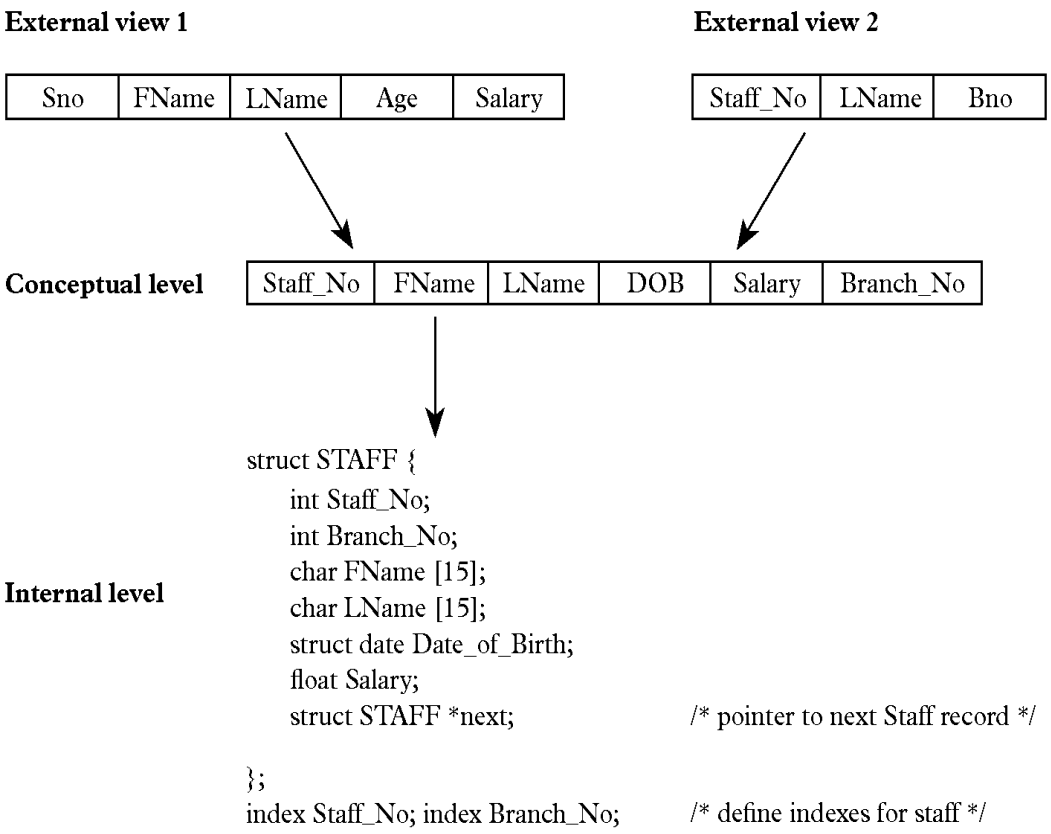
Contoh : organisasi file secaa sequential, raltive atau index sequential

- penempatan record
- pemampatan data dan teknik encryption




Gambar 1. Tingkatan Arsitektur Database

Contoh pengambaran tingkatan arsitektur database :




Gambar 2. Contoh penggambaran tingkatan arsitektur database

## *Data Independence*

Tujuan utama dari 3 tingkat arsitektur adalah memelihara kemandirian data (*data independence*) yang berarti perubahan yang terjadi pada tingkat yang lebih rendah tidak mempengaruhi tingkat yang lebih tinggi.

Ada 2 jenis *data independence*, yaitu

1. *Physical Data Independence*

   bahwa *internal schema* dapat diubah oleh DBA tanpa menggangu *conceptual schema*. Dengan kata lain *physical data independence* menunjukkan kekebalan *conceptual schema* terhadap perubahan *internal schema*.

   Contoh : - Menambah indeks tambahan
   - Merubah penyimpanan data
   - Merubah organisasi file dari sequential ke index sequential

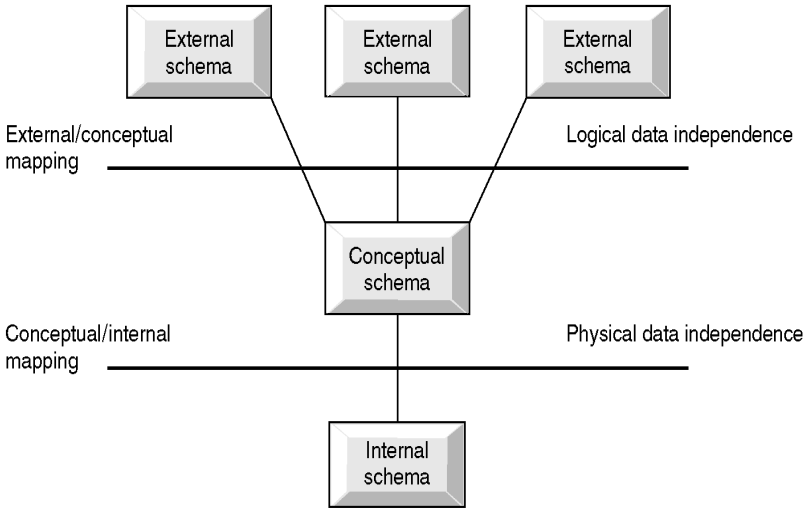
2. *Logical Data Independence*

bahwa *conceptual schema* dapat diubah oleh DBA tanpa menggangu *external schema*. Dengan kata lain *logical data independence* menunjukkan kekebalan *external schema* terhadap perubahan *conceptual schema*.

Contoh : - Menambah dan menghapus suatu tipe record
- Merubah format data

Prinsip *data independence* adalah salah satu hal yang harus diterapkan di dalam pengelolaan sistem basis data dengan alasan-alasan sbb :

1. DBA dapat mengubah isi, lokasi, perwujudan dalam organisasi basis data tanpa mengganggu program-program aplikasi yang sudah ada.
2. Pabrik/agen peralatan/*software* pengolahan data dapat memperkenalkan produk-produk baru tanpa mengganggu program-program aplikasi yang sudah ada.
3. Untuk memindahkan perkembangan program-program aplikasi
4. Memberikan fasilitas pengontrolan terpusat oleh DBA demi keamanan dan integritas data dengan memperhatikan perubahan-perubahan kebutuhan pengguna.




Gambar 3. Data Independence

**Mapping ( Transformasi )**

Proses pendefinisian informasi dari satu level ke level lainnya.

**Konseptual/Internal Mapping**

Pendefinisian hubungan antara view konseptual dengan basis data di level internal.
(Bagaimana record-record / field-filed didalam level konseptual didefinisikan di level internal)

**Eksternal/Konseptual Mapping**

Pendefinisikan hungan antra view konseptual dengan view eksternal

**Bahasa Dalam DBMS**

DBMS (*Database Management systems*) adalah kumpulan program yang mengkoordinasikan semua kegiatan yang berhubungan dengan *database*. Dengan adanya berbagai tingkatan pandangan dalam suatu *database* maka untuk mengakomodasikan masing-masing pengguna dalam piranti lunak manajemen *database* biasanya terdapat bahasa-bahasa tertentu yang disebut ***Data Sub language***.

*Data sub language* adalah subset bahasa yang dipakai untuk operasi manajemen *database*. Dalam penggunaan biasanya dapat ditempelkan (*embedded*) pada bahasa tuan rumah (Cobol, PL/1, dsb). Secara umum maka setiap pengguna *database* memerlukan bahasa yang dipakai sesuai tugas dan fungsinya.

Dalam database secara umum dikenal 2 data sub language :

1. *Data Definition Language (DDL)*
   Bahasa yang digunakan dalam mendefinisikan struktur atau kerangka dari *database*, di dalamnya termasuk *record*, elemen data, kunci elemen, dan relasinya
2. *Data Manipulation Language (DML)*
   Bahasa yang digunakan untuk menjabarkan pemrosesan dari *database*, fasilitas ini diperlukan untuk memasukkan, mengambil, mengubah data. DML dipakai untuk operasi terhadap isi *database*
   Ada 2 jenis DML :
   1. *Procedural DML*
      Digunakan untuk mendefinisikan data yang diolah dan perintah yang akan dilaksanakan.
   2. *Non Procedural*
      Digunakan untuk menjabarkan data yang diinginkan tanpa menyebutkan bagaimana cara pengambilannya.

Secara khusus pengguna menggunakan berbagai bahasa :

Programmer aplikasi menggunakan bahasa-bahasa seperti Cobol, Informix, dll (*host language*) yang ditempelkan dengan bahasa yang dipakai dalam DBMS. Pemakai terminal menggunakan bahasa Query (misal SQL) atau menggunakan program aplikasi (yang dirancang oleh programmer). Sedangkan DBA lebih banyak menggunakan bahasa DDL dan DML yang tersedia dalam DBMS.

DBMS mempunyai tugas untuk menangani semua bentuk akses kepada *database*, secara konsep :

1. Pengguna menyatakan permintaan akses menggunakan DBMS
2. DBMS menangkap dan menginterpretasikan
3. DBMS mencari :
   - eksternal / *conceptual mapping*
   - *conceptual schema*
   - konseptual / *internal mapping*
   - *internal schema*
4. DBMS melaksanakan operasi yang diminta terhadap *database* tersimpan.

   Proses 1 s/d 4 dapat dilakukan secara interactive atau dicompile dulu.

**Fungsi DBMS**

Layanan - layanan yang sebaiknya disediakan oleh *database management system* adalah :

1. Penyimpanan, pengambilan dan perubahan data

   Sebuah DBMS harus menyediakan kemampuan menyimpan, mengambil dan merubah data dalam *database*.
2. Katalog yang dapat diakses pemakai

   menyediakan sebuah katalog yang berisi deskripsi item data yang disimpan dan diakses oleh pemakai.
3. Mendukung Transaksi

   Menyediakan mekanisme yang akan menjamin semua perubahan yang berhubungan dengan transaksi yang sudah ada atau yang akan dibuat.
4. Melayani kontrol *concurrency*

   Sebuah DBMS harus menyediakan mekanisme yang menjamin *database* ter-update secara benar pada saat beberapa pemakai melakukan perubahan terhadap *database* yang sama secara bersamaan.
5. Melayani *recovery*

   Menyediakan mekanisme untuk mengembalikan *database* ke keadaan sebelum terjadinya kerusakan pada *database* tersebut.
6. Melayani autorisasi

   Sebuah DBMS harus menyediakan mekanisme untuk menjamin bahwa hanya pemakai yang berwenang saja yang dapat mengakses *database*.
7. Mendukung komunikasi data

   Sebuah DBMS harus mampu terintegrasi dengan software komunikasi.
8. Melayani integrity

Sebuah DBMS bertujuan untuk menjamin semua data dalam *database* dan setiap terjadi perubahan data harus sesuai dengan aturan yang berlaku.

9. Melayani *data independence*

   Sebuah DBMS harus mencakup fasilitas untuk mendukung kemandirian program dari struktur *database* yang sesungguhnya.

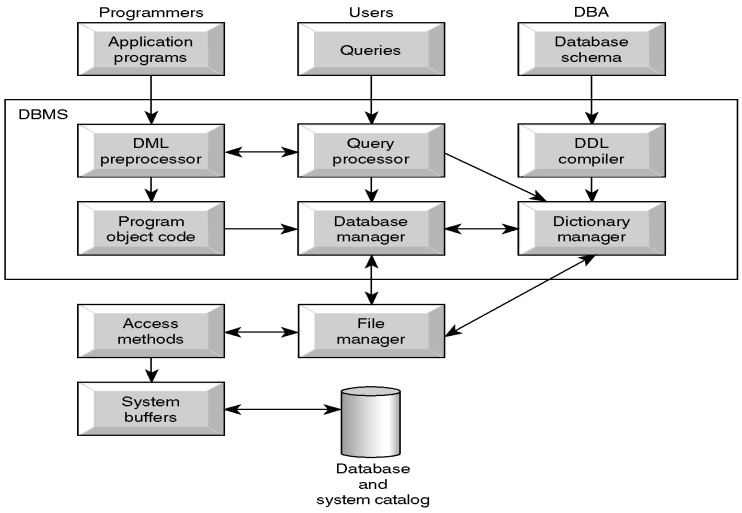
10. Melayani *utility*

    Sebuah DBMS sebaiknya menyediakan kumpulan layanan *utility*.

**Komponen Lingkungan DBMS**

1. Data
   - terintegrasi (integrated)
   - dapat dipakai bersama-sama (shared)
2. Perangkat keras (hardware)
3. Perangkat Lunak (software)
4. Brainware
5. Prosedur

**KOMPONEN DBMS**




Gambar 4. Komponen DBMS

1. *Query Processsor*

   Komponen yang merubah bentuk *query* ke dalam instruksi tingkat rendah ke *database manager*
2. *Database Manager*

   Menerima query-query lalu memeriksa skema eksternal dan konseptual untuk menentukan record-record konseptual yang diperlukan
3. *File Manager*

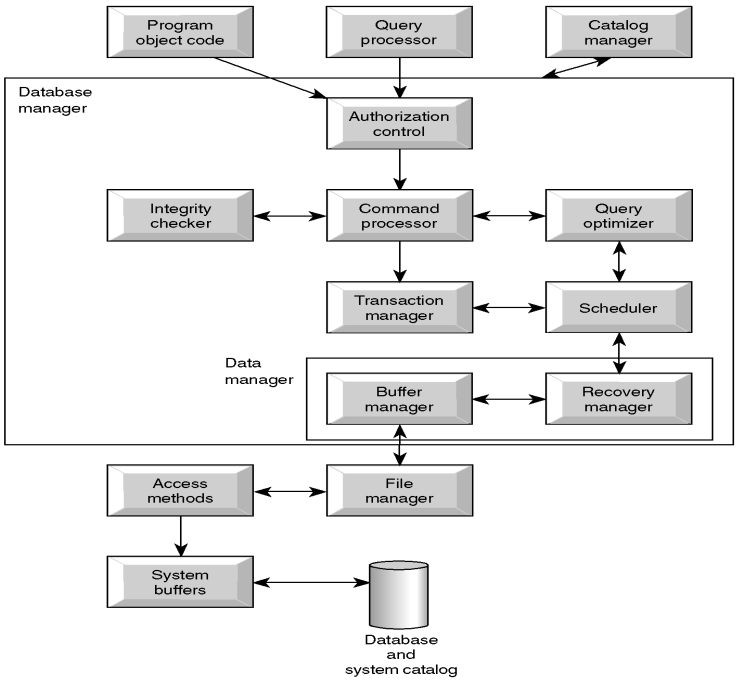
   Memanipulasi penyimpanan file dan mengatur alokasi ruang penyimpanan pada disk dan menentukan record-record konseptual yang diperlukan
4. DML *Preprocessor*

   Modul yang merubah perintah DML *embedded* ke dalam program aplikasi ke standar fungsi pemanggilan dalam bahasa pemrograman
5. DDL *Compiler*

   Merubah perintah DDL menjadi kumpulan tabel yang berisi metadata.
6. *Dictionary Manager*

   Mengatur akses dan memelihara *data dictionary*. *Data dictionary* diakses oleh komponen DBMS yang lain.

## Komponen Database Manager




Gambar 5. Komponen Software Utama Database Manager

Komponen software utama database manager adalah

1. *Authorization Control*

   Modul yang memeriksa apakah pemakai mempunyai wewenang untuk menyelesaikan operasi

2. *Command Processor*

   Memeriksa apakah pemakai mempunyai wewenang untuk menyelesaikan operasi

3. Integrity Checker

   Untuk semua operasi yang merubah database, integrity checker memeriksa operasi yang diminta memerlukan batasan integritas.

4. Query Optimizer

   Modul ini menentukan strategi yang optimal untuk aksekusi query

5. Transaction Manager

   Modul ini mengerjakan proses-proses yang dibutuhkan operasi yang diterima transaksi

6. Scheduler

   Modul ini bertanggung jawab untuk menjamin operasi secara bersamaan terhadap database sehingga berjalan tanpa ada masalah antara yang satu dengan yang lain.

7. Recovery Manager

   Modul ini menjamin database tetap konsisten walaupun terjadi kerusakan.

8. Buffer Manager

   Modul ini bertanggung jawab terhadap pemindahan data antara main memory dan secondary storage, seperti disk dan tape.

**Utiliti – Utiliti DBMS**

**Loading**

Memasukkan suatu file data ke dalam database

**Backup**

Membuat salinan database yang dapat digunakan untuk menempatkan database pada saat terjadi kegagalan

**File Reorganization**

Mereorganisasikan file databse ke jenis database lain untuk meningkatkan performasi

**Report Generation**

Membuat format laporan-laporan dengan mengontrol spasi, header, footer, total, summary dsb.

**Performance Monitoring**

Memonitor penggunaan database dan menyediakan statistic untuk DBA

**Data Dictionary**

*Data dictionary* adalah tempat penyimpanan informasi yang menggambarkan data dalam database. *Data dictionary* biasa disebut juga dengan *metadata* atau *data mengenai data*. Modul pengontrol autorisasi menggunakan *data dictionary* untuk memeriksa apakah seorang pemakai perlu mempunyai wewenang.

Untuk mengerjakan pemeriksaan tersebut *data dictionary* menyimpan :

- nama-nama pemakai yang mempunyai wewenang untuk menggunakan DBMS
- nama-nama *data item* yang ada dalam *database*
- *data item* yang dapat diakses oleh pemakai dan jenis akses yang diijinkan, misalnya: *insert, update, delete* atau *read*

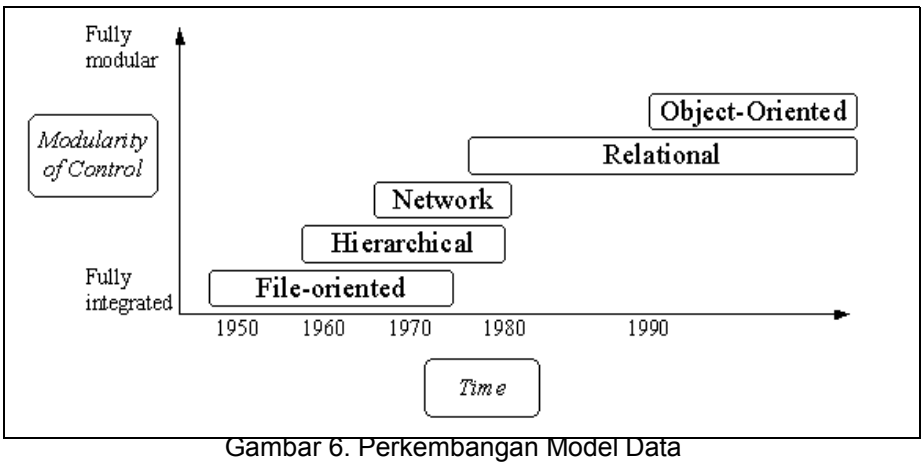
Sedangkan untuk memeriksa integritas data, *data dictionary* menyimpan :

- nama-nama *data item* dalam *database*
- jenis dan ukuran *data item*
- batasan untuk masing-masing *data item*

Sistem *data dictionary* dapat dibedakan atas sistem aktif dan pasif. Sistem aktif selalu konsisten dengan struktur *database* karena secara otomatis dikerjakan oleh sistem. Sebaliknya, sistem pasif tidak konsisten terhadap perubahan *database* yang dilakukan oleh pemakai.

**Model Data**

Model data adalah kumpulan konsep yang terintegrasi yang menggambarkan data, hubungan antara data dan batasan-batasan data dala suatu organisasi. Fungsi dari sebuah model data untuk merepresentasikan data sehingga data tersebut mudah dipahami.
Perkembangan model data merupakan representasi dari suatu reaksi terhadap model-model yang mendahuluinya. Sistem hierarkhi suatu basis data merupakan bagian dari perkembangan yang diciptakan untuk mengatasi kekurangan yang ada pada sistem berorientasikan file (file-oriented). Basis data jaringan dikembangkan untuk mengatasi keterbatasan dari desain hierarkhi. Basis data relasional muncul sebagai solusi baru untuk masalah-masalah yang muncul pada desain hierarkhi dan desain network dan seterusnya.



Gambar 6. Perkembangan Model Data

Untuk menggambarkan data pada tingkat eksternal dan konseptual digunakan model data berbasis objek atau model data berbasis *record*.

1. *Model Data Berbasis Objek*

   Model data berbasis objek menggunakan konsep entitas, atribut dan hubungan antar entitas. Beberapa jenis model data berbasis objek yang umum adalah :

   - *entity-relationship*
   - *semantic*
   - *functional*
   - *object-oriented*

2. *Model Data Berbasis Record*

   Pada model data berbasis *record*, *database* terdiri dari sejumlah *record* dalam bentuk yang tetap yang dapat dibedakan dari bentuknya. Ada 3 macam jenis model data berbasis *record* yaitu :

   a. Model data relasional

   Model data ini berdasarkan konsep relasi matematika, data dan relasi yang digambarkan pada sebuah tabel yang mempunyai kolom dan baris dimana kolom-kolom tersebut mempunyai nama yang unik.

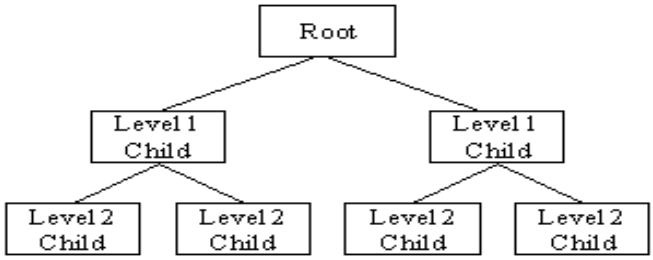
Contoh :

| Name | Address | Course | Grade |
|---|---|---|---|
| Mr. Eric Tachibana | 123 Kensigton | Chemistry 102 | C+ |
| Mr. Eric Tachibana | 123 Kensigton | Chinese 3 | A |
| Mr. Eric Tachibana | 122 Kensigton | Data Structures | B |
| Mr. Eric Tachibana | 123 Kensigton | English 101 | A |
| Ms. Tonya Lippert | 88 West 1st St. | Psychology 101 | A |

b. Model data hierarkhi

Model data ini dikenal sebagai *model struktur pohon*, di mana data direpresentasikan dalam bentuk pohon. Sebuah database hierarkhi terdiri dari kumpulan record-record di mana record yang satu dengan yang lainnya dihubungkan dengan link.

Model hierarki mempunyai dua konsep struktur data yaitu *record* dan *parent-child relationship (PCR)*.



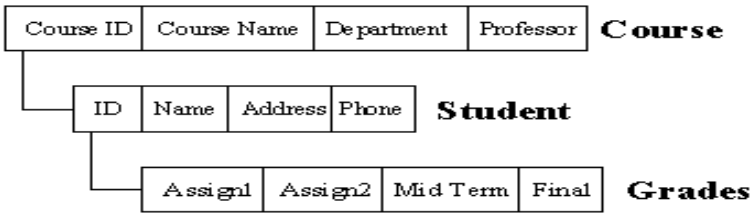

Contoh perangkat lunak dari model data hierarkhi adalah Information Management System (IMS) yang dikembangkan oleh IBM

c. Model data jaringan

Model data jaringan dikenal sebagai **STRUKTUR PLEX**. Pada dasarnya struktur jaringan ini merupakan perluasan dari struktur hirarki. Kalau pada struktur hirarki, setiap child hanya mempunyai satu parent. Sedangkan pada struktur jaringan, setiap child dapat mempunyai lebih dari satu parent.

Struktur jaringan ini merupakan suatu graph, terdiri dari suatu *node (simpul),* yang dihubungkan dengan suatu *edge*.

Ada dua struktur data pada basis data Model Jaringan, yaitu RECORD dan SET. Contoh dari model data jaringan adalah perangkat lunak IDMS (Integrated Database Management System) yang merupakan produk dari perusahaan perangkat lunak CULLINET yang bekerja pada mainframe IBM dengan sistem kerja DOS atau MVS. IDMS menggunakan sistem CODASYL atau DBTG. IDMS yang dikemukakan oleh DBTG terdiri atas 3 bahasa basis data yaitu skema DDL, sub skema DDL dan DML.

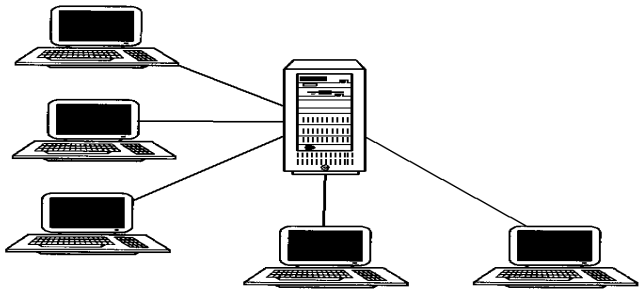



**Arsitektur DBMS Multi User**

***Teleprocessing***

Arsitektur tradisional untuk sistem multi user adalah teleprocessing, dimana satu komputer dengan sebuah CPU dan sejumlah terminal seperti pada gambar di bawah ini.

Semua pemrosesan dikerjakan dalam batasan fisik komputer yang sama. Terminal untuk pemakai berjenis '*dumb*', yang tidak dapat berfungsi sendiri dan masing-masing dihubungkan ke komputer pusat. Terminal-terminal tersebut mengirimkan pesan melalui subsistem pengontrol komunikasi pada sistem operasi ke program aplikasi, yang bergantian menggunakan layanan DBMS.



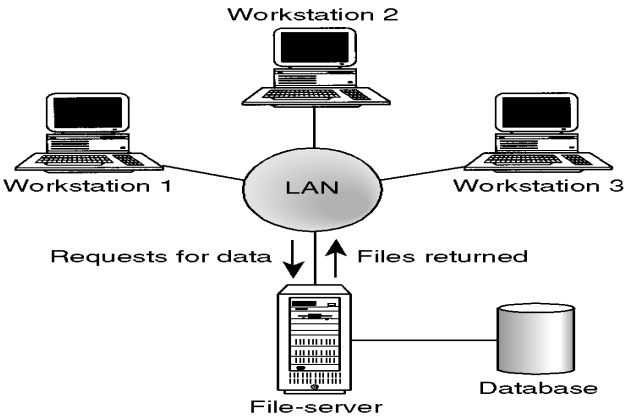
Gambar 7. Arsitektur Teleprocessing

Dengan cara yang sama, pesan dikembalikan ke terminal pemakai. Arsitektur ini menempatkan beban yang besar pada komputer pusat yang tidak hanya menjalankan program aplikasi tetapi juga harus menyelesaikan sejumlah pekerjaan pada terminal seperti format data untuk tampilan di monitor.

• contoh : ATM mesin di bank-bank.

***File-Server***

Proses didistribusikan ke dalam jaringan sejenis LAN (*Local Area Network*). *File server* mengendalikan file yang diperlukan oleh aplikasi dan DBMS. Meskipun aplikasi dan DBMS dijalankan pada masing-masing *workstation* tetapi tetap meminta file dari *file server* jika diperlukan (perhatikan gambar di halaman berikut ini).




Gambar 8. Arsitektur File Server

Dengan cara ini, *file server* berfungsi sebagai sebuah hard disk yang digunakan secara bersamaan.

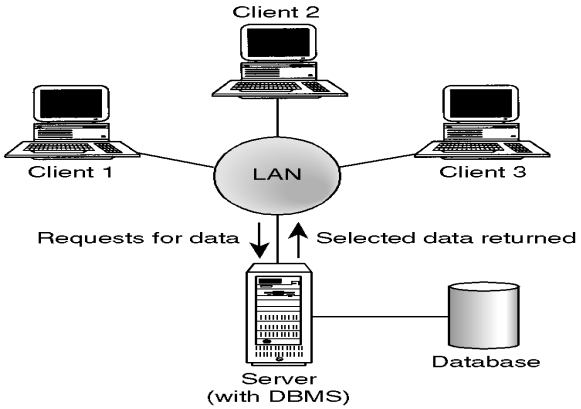
Kerugian arsitektur *file-server* adalah :

- Terdapat lalulintas jaringan yang besar
- Masing-masing *workstation* membutuhkan copy DBMS
- Kontrol terhadap *concurrency*, *recovery* dan *integrity* menjadi lebih kompleks karena sejumlah DBMS mengakses file secara bersamaan

***Client Server***

Untuk mengatasi kelemahan arsitektur-arsitektur di atas maka dikembangkan arsitektur *client-server*. *Client-server* menunjukkan cara komponen *software* berinteraksi dalam bentuk sistem.

Sesuai dengan namanya, ada sebuah pemroses *client* yang membutuhkan sumber dan sebuah *server* yang menyediakan sumbernya. Tidak ada kebutuhan *client* dan *server* yang harus diletakkan pada mesin yang sama. Secara ringkas, umumnya *server* diletakkan pada satu sisi dalam LAN dan client pada sisi yang lain.




Gambar 9. Arsitektur Client Server

Dalam konteks database, *client* mengatur *interface* berfungsi sebagai *workstation* tempat menjalankan aplikasi *database*. *Client* menerima permintaan pemakai, memeriksa sintaks dan *generate* kebutuhan database dalam SQL atau bahasa yang lain. Kemudian meneruskan pesan ke *server*, menunggu *response* dan bentuk *response* untuk pemakai akhir. *Server* menerima dan memproses permintaan database kemudian mengembalikan hasil ke *client*.

Proses-proses ini melibatkan pemeriksaan autorisasi, jaminan integritas, pemeliharaan *data dictionary* dan mengerjakan *query* serta proses *update*. Selain itu juga menyediakan kontrol terhadap *concurrency* dan *recovery*.

Adapun beberapa keuntungan jenis arsitektur ini adalah :

- Memungkinkan akses *database* yang besar
- Menaikkan performa
- Jika *client* dan *server* diletakkan pada komputer yang berbeda kemudian CPU yang berbeda dapat memproses aplikasi secara paralel. Hal ini mempermudah merubah mesin server jika hanya memproses database.
- Biaya untuk *hardware* dapat dikurangi
- Hanya *server* yang membutuhkan storage dan kekuatan proses yang cukup untuk menyimpan dan mengatur *database*
- Biaya komunikasi berkurang
- Aplikasi menyelesaikan bagian operasi pada *client* dan mengirimkan hanya bagian yang dibutuhkan untuk akses *database* melewati jaringan, menghasilkan data yang sedikit yang akan dikirim melewati jaringan
- Meningkatkan kekonsistenan
- *Server* dapat menangani pemeriksaan integrity sehingga batasan perlu didefinisikan dan validasi hanya di satu tempat, aplikasi program mengerjakan pemeriksaan sendiri

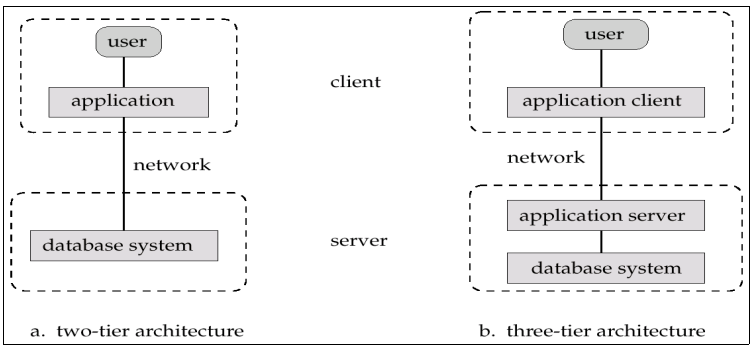
- Map ke arsitektur open-system dengan sangat alami

Berikut ini adalah ringkasan fungsi *client-server*

| ***Client*** | ***Server*** |
|---|---|
| Mengatur *user interface* | Menerima dan memproses *database* yang diminta dari client |
| Menerima dan memeriksa sintaks input dari pemakai | Memeriksa autorisasi |
| Memproses aplikasi | Menjamin tidak terjadi pelanggaran terhadap *integrity constraint* |
| Generate permintaan database dan memindahkannya ke *server* | Melakukan *query*/pemrosesan *update* dan memindahkan response ke client |
| Memberikan *response* balik kepada pemakai | Memelihara *data dictionary* |
| | Menyediakan akses *database* secara bersamaan |
| | Menyediakan kontrol *recovery* |

Ada beberapa jenis model client – server arsitektur yaitu :

- Two Tier Arsitektur
  - E.g. client programs using ODBC/JDBC to communicate with a database
- Three Tier Arsitektur
  - E.g. web-based applications, and applications built using “middleware”




Gambar 10. Arsitektur Aplikasi Two Tier dan Three Tier