BAB 2

LANDASAN TEORI

2.1 Framework

Framework adalah sebuah kumpulan dari modul-modul pembangun perangkat lunak yang umum dan tak dapat di eksekusi dimana programmer dapat menggunakannya, menambahnya atau mengkonfigurasi untuk solusi atau program/aplikasi yang spesifik. Dengan *framework, developer* tidak harus memulai dari awal setiap mereka memulai membuat aplikasi. *Framework* terdiri dari kumpulan modul atau objek sehingga desain dan kode dari *framework* dapat digunakan kembali.

2.1.1 .NET Framework

Pada bulan Februari 2002, Microsoft secara resmi merilis Visual Studio .NET (VS .NET) versi beta dimana salah satu bahasa pemrogramannya adalah Microsoft Visual C# .NET (C# .NET).

Setahun setelah VS .NET diluncurkan, Microsoft merilis VS.NET 2003 yang memperbaiki performa dan aspek keamanan dari VS.NET versi sebelumnya. Pada VS.NET 2003 digunakan teknologi *.NET Framework 1.1* yang baru.

Framework .NET adalah lingkungan untuk membangun, menyebarkan/*deploying*, dan menjalankan aplikasi dan layanan berbasis web. *.Net Framework* disusun oleh dua komponen utama, yaitu *Common Language Runtime (CLR)* dan *.NET Framework Class Library* termasuk aplikasi *Console, Windows GUI, ASP.NET*, layanan web *XML* dan layanan *Windows.*

Visual Studio.NET dibangun menggunakan fondasi *.NET Framework*. *.NET Framework* menyediakan lingkungan yang cerdas, mudah dikembangkan untuk membangun, menyebarkan dan menjalankan aplikasi dan layanan web XML yang terdistribusi. Dalam istilah yang mudah, *.NET Framework* memisahkan *platform* sistem operasi menjadi dua lapisan, yaitu lapisan pemrograman dan lapisan eksekusi.

Tujuan dari *.NET Framework* adalah :

1. Menyediakan lingkungan pemrograman berorientasi objek, apakah kode objek disimpan dan dijalankan secara lokal, dijalankan secara lokal tetapi disebarkan melalui internet atau dijalankan secara *remote* (dijalankan dari suatu tempat).
2. Menyediakan lingkungan untuk menjalankan suatu kode yang menjamin keamanan saat kode dijalankan.
3. Menyediakan lingkungan untuk menjalankan suatu kode yang dapat mengeliminasi masalah performa dari lingkungan *scripted* dan *interpreted*.
4. Menyediakan lingkungan untuk menjalankan suatu kode yang meminimalkan konflik pada *deployment* dan *versioning* perangkat lunak.
5. Menyatukan model-model pemrograman dengan didukung oleh banyak bahasa dan membuat berbagai tipe aplikasi.

2.1.2 iBatis.Net

iBatis adalah sebuah *framework* yang digunakan untuk membantu para pemrogram untuk membuat sebuah aplikasi yang dinamik, khususnya dalam hal pengaksesan basis data. iBatis berada di *persistence layer framework* yang membuat penulisan kode SQL menjadi lebih mudah dikerjakan dan

lebih mudah diintegrasikan ke dalam aplikasi berbasis *Object Oriented Programming*.

Ibatis merupakan *persistence layer framework*. Persistence layer berada diantara *business logic layer* dari aplikasi dan basis data. Pemisahan antara *bussiness logic* dengan basis data ini sangat diperlukan agar tidak akan mengacaukan kode *business logic*. Keuntungan dari pemisahan ini adalah kode aplikasi menjadi lebih mudah untuk dipelihara.

Dalam sistem *object-oriented*, titik utama dari persistence layer adalah pada penyimpanan dan pengaksesan objek, atau lebih spesifik data yang disimpan dalam objek tersebut. Dalam aplikasi berskala enterprise, *persistence layer* biasanya berinteraksi dengan *relational database system* untuk penyimpanan data.

Beberapa alasan mengapa menggunakan *iBATIS* :

1. Kesederhanaan
   iBATIS mudah digunakan bagi pengguna Java dan .Net karena cara bekerjanya menyerupai JDBC dan ADO.NET.
2. Produktivitas
   Tujuan utama dari *framework* yang baik adalah membuat *developer* lebih produktif. *iBATIS* mengurangi sejumlah code di *persistence layer* hingga 62 persen.
3. Kinerja
   Hal yang lebih signifikan dalam melihat kinerja dari *iBATIS* adalah bagaimana kita mengambil data dari basis data, kapan dan seberapa sering. Sebagai contoh, menggunakan *data paging* yang

secara dinamis mengambil *records* dari basis data secara signifikan dapat meningkatkan kinerja dari aplikasi karena kita tidak melakukan operasi yang tidak diperlukan dalam mengambil ribuan *records* dari basis data dalam sekali waktu.

4. Pembagian *concern*

   Kita melihat bagaimana aplikasi dilapisan yang tinggi dan juga bagaimana *persistence layer* diletakkan pada bagian internal. *iBATIS* membantu untuk mendukung *layering* dengan mengatur semua *resources* yang berhubungan dengan *persistence*, seperti koneksi basis data, *prepared statements*, dan *result sets*. *iBATIS* juga menyediakan interface yang tidak bergantung pada basis data dan API yang membantu *resources* lainnya dalam aplikasi. Dengan *iBATIS*, kita bekerja hanya dengan objek yang nyata, bukan dengan *result sets* yang berubah-ubah.

5. Pembagian kinerja

   Dalam *iBATIS*, SQL dipisahkan dari coding aplikasi, pemrogram SQL dapat menulis SQL dengan caranya sendiri tanpa harus menghawatirkan pemotongan (*concatenation*) string.

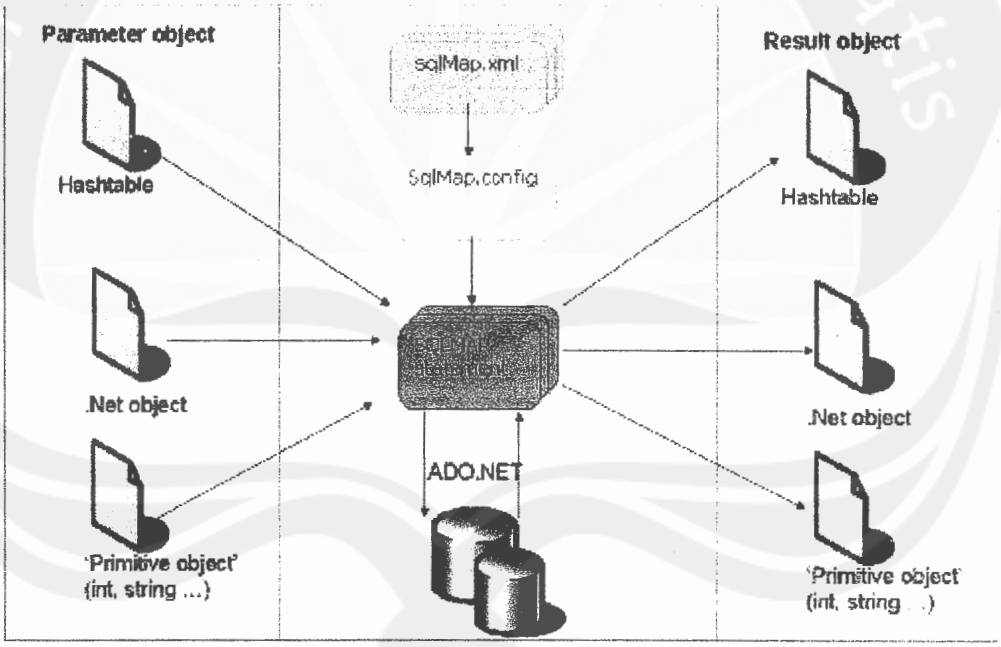
6. *Portable*

   iBATIS merupakan konsep yang sangat *portable*. Dikarenakan kesederhanaan *design*-nya, *iBATIS* dapat diimplementasikan untuk hampir setiap bahasa atau *platform*. Saat ini, *iBATIS* mendukung 3 platform seperti *Java*, *Ruby*, dan C# untuk Microsoft .NET.

7. *Open Source* dan apa adanya

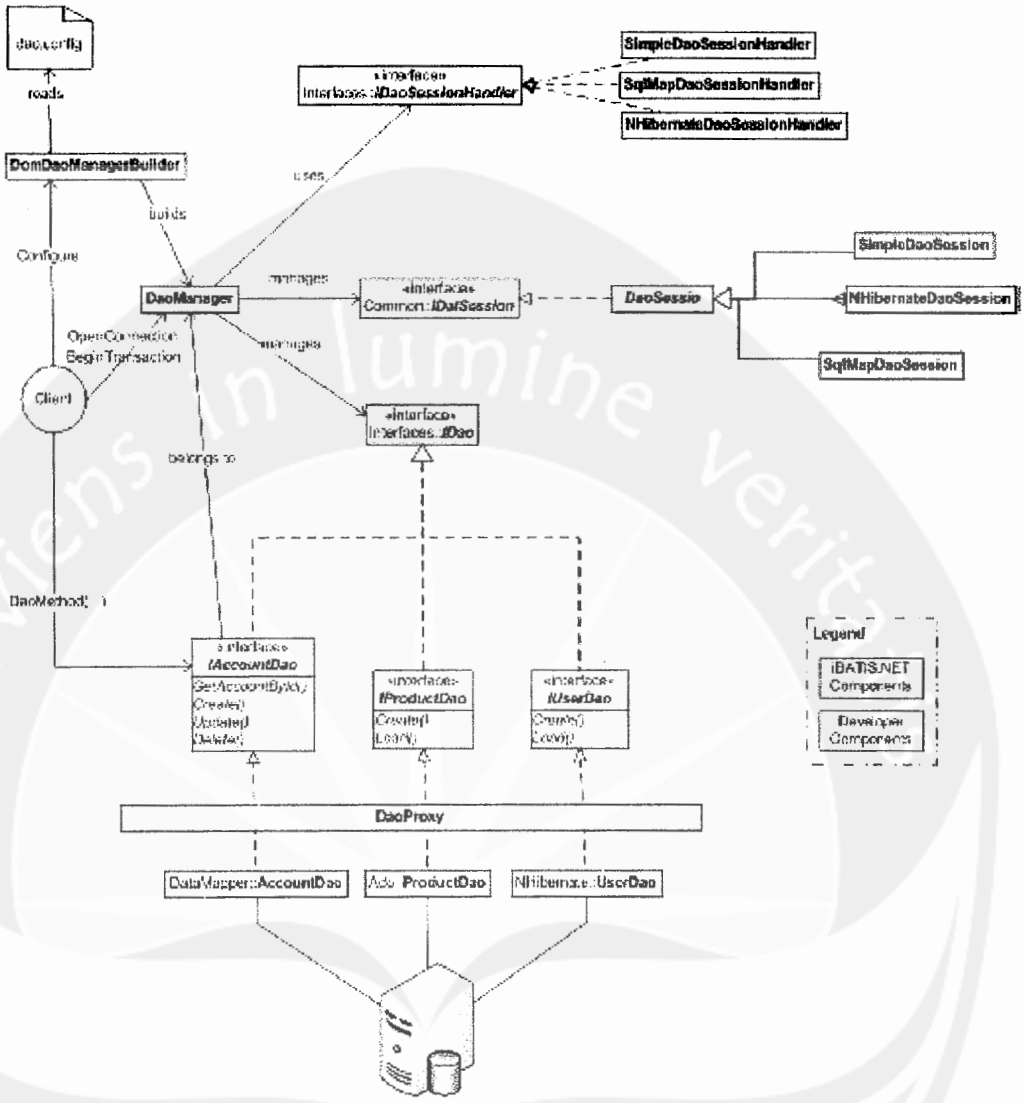
   iBATIS merupakan software yang gratis, dan *open source* yang dapat digunakan oleh siapa saja, tanpa harus membeli.

Disamping itu, *iBATIS* merupakan cara lain untuk menuliskan code JDBC dan ADO.NET. *iBATIS* berjalan hampir sama dengan ADO.NET dan JDBC. *iBATIS* mengatur koneksi ke basis data, menempatkan parameter, mengeksekusi *statement*, mengambil hasil dan menutup semua *resources*. Akan tetapi, sejumlah code yang perlu kita tuliskan dikurangi secara signifikan.



Gambar 2.1 *Ibatis Data Mapper Workflow*

Gambar 2.1 merupakan Workflow dari Ibatis Data Mapper yang digunakan dalam Gambar 2.2 yang merupakan contoh penggunaan Ibatis Data Access.



Gambar 2.2 *Ibatis Data Access Workflow Example*

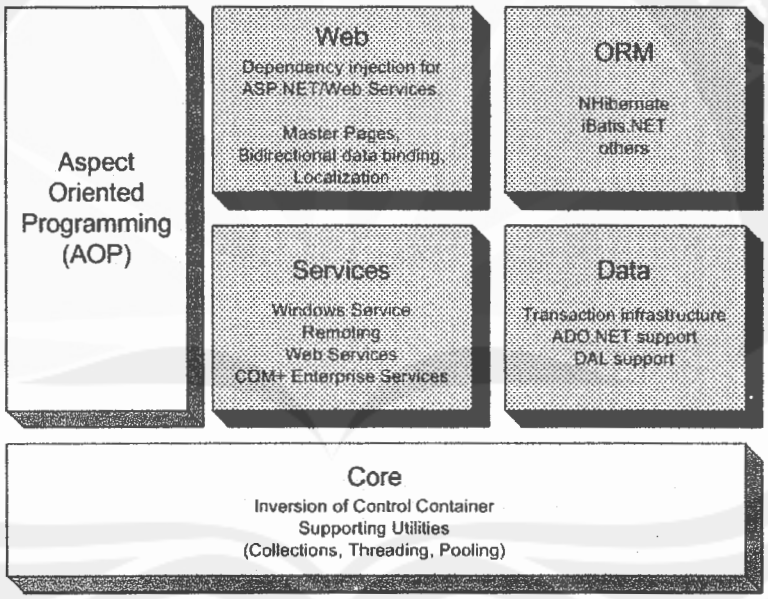
2.1.3 Spring.NET

Spring.NET adalah sebuah *framework* aplikasi yang fokus untuk membangun aplikasi *enterprise* pada .NET. *Spring.NET* ini menyediakan fungsionalitas yang digunakan secara luas seperti *Dependency Injection*, *Aspect Oriented Programming* (AOP), *Data Access Abstraction*, dan *ASP.NET Integration*.

Aplikasi *enterprise* biasanya dibangun dari sejumlah lapisan fisik yang berbeda-beda fungsinya dan di setiap

lapisan tersebut sangat sering dipisahkan oleh lapisan fungsional. Contohnya lapisan *business service* biasanya menggunakan objek-objek yang terdapat pada lapisan *data access*.

Komponen IOC (Inversion Of Control) dari *Spring Framework* mengatur kelas-kelas, objek-objek, dan services yang membangun suatu aplikasi enterprise, dengan menyediakan suatu wadah formal dalam menggabungkan komponen-komponen yang berbeda dalam suatu aplikasi yang siap digunakan.



Gambar 2.3 Arsitekur *Spring Framework*

2.1.4 *Dependency Injection*

Dependency Injection merupakan salah satu teknik pemrograman untuk mengurangi ketergantungan diantara komponen-komponen penyusun suatu program dan meningkatkan kemampuan "*reusability*" dari komponen tersebut. Sebagai contoh jika sebuah komponen menggunakan *sub-component*, penggunaan kembali komponen tersebut dengan *sub-component* yang berbeda akan menjadi sangat sulit. Contoh konkrit

adalah jika sebuah aplikasi menggunakan suatu *basis data* dari *DBMS* A sebagai tempat penyimpanan datanya, lalu di kemudian hari program tersebut ingin diubah dengan menggunakan *DBMS* B maka yang terjadi adalah mengubah (membongkar) kode program.

Ide dasar dari *dependency injection* adalah memindahkan sebagian kode yang meng-instansi sub-*component* ke sebuah *configuration file* (biasanya *XML file*) dan menggunakan sebuah *framework* untuk mewujudkannya.

2.2 *E-Banking*

Electronic Banking, atau *e-banking* bisa diartikan sebagai aktivitas perbankan di internet. Layanan ini memungkinkan nasabah sebuah bank dapat melakukan hampir semua jenis transaksi perbankan melalui sarana internet, khususnya via web. Mirip dengan penggunaan mesin ATM, lewat sarana internet seorang nasabah dapat melakukan aktifitas pengecekan rekening, transfer dana antar rekening, hingga pembayaran tagihan tagihan rutin bulanan (listrik, telepon, dsb.) melalui rekening banknya. Jelas banyak keuntungan yang bisa didapatkan nasabah dengan memanfaatkan layanan ini, terutama bila dilihat dari waktu dan tenaga yang dapat dihemat karena transaksi *e-banking* jelas bebas antrian dan dapat dilakukan dari mana saja sepanjang nasabah dapat terhubung dengan jaringan internet.

Untuk dapat menggunakan layanan ini, seorang nasabah akan dibekali dengan *login* dan kode akses ke situs *web* dimana terdapat fasilitas *e-banking* milik bank bersangkutan. Selanjutnya, nasabah dapat melakukan login

dan melakukan aktifitas perbankan melalui situs *web* bank bersangkutan.

2.3 Object Oriented Analysis and Design

Object Oriented Analysis (*OOA*) adalah teknik model *driven* yang mengintegrasikan data dan proses kedalam konstruksi yang disebut objek. Model *OOA* berupa gambar yang mengilustrasikan objek sistem dari berbagai perspektif seperti struktur dan *behavior*.

Object Oriented Design (*OOD*) digunakan untuk memperjelas definisi kebutuhan objek yang diidentifikasi sebelumnya selama analisis, dan untuk mendefinisikan desain objek tertentu.

Pada pembangunan perangkat lunak berkonsep Pemrograman Berorientasi Objek, dapat dijelaskan dalam dua model data, yaitu:

1. *Unified Modeling Language* (*UML*), digunakan untuk menggambarkan sistem secara logika tanpa memperhatikan lingkungan fisik di mana user berinteraksi dengan sistem. Adapun simbol-simbol yang sering digunakan dalam penulisan *Unified Modeling Language* (*UML*), yaitu (Fowler, 2005):
   a. *Use Case*, merupakan deskripsi dari interaksi tipikal antara para pengguna sistem dengan sistem itu sendiri, dengan memberi sebuah narasi tentang bagaimana sistem tersebut digunakan, disimbolkan dengan:

Gambar 2.4 Notasi Use Case

b. *Actor,* merupakan sebuah peran yang dimainkan seorang pengguna dalam kaitannya dengan sistem. Actor sering juga disebut user dari sistem, disimbolkan dengan:

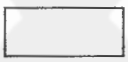
Gambar 2.5 Notasi Actor

c. *Association,* merupakan sebuah garis solid antara dua class yang menunjuk dari class sumber ke class target, disimbolkan dengan:

Gambar 2.6 Notasi Association

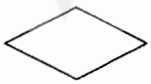
2. *Entity Relationship Diagram*, digunakan untuk mengidentifikasikan hubungan antar entitas dalam sistem. Adapun simbol-simbol dalam *Entity Relationship Diagram* (*ERD*), yaitu (Rob, 2004):

a. Entitas, dapat berupa elemen lingkungan, disimbolkan dengan:



Gambar 2.7 Simbol Entitas

b. *Relationship*, penghubung antara dua entitas, disimbolkan dengan:



Gambar 2.8 Simbol Relationship

c. Konektivitas, cacah hubungan yang ada antara dua entitas. Konektivitas terdiri dari 4 macam, yaitu: satu ke satu (1:1), satu ke banyak (1:M), banyak ke satu (M:1), dan banyak ke banyak (M:M).
d. Kardinalitas, banyaknya data (*tuple*) yang dimiliki oleh sebuah entitas. Kardinalitas ditulis dengan cara (A,B), nilai A sebagai batas nilai terkecil dan nilai B sebagai batas nilai terbesar.

2.4 Tools dan Teknologi yang Digunakan

2.4.1 Teknologi .NET

Microsoft Visual Studio .NET adalah sebuah *platform* untuk membangun, menjalankan dan meningkatkan generasi lanjut dari aplikasi terdistribusi. *Microsoft Visual Studio .NET* memperluas *client*, *server* dan *service-service* yang terdiri atas:

1. Sebuah model pemrograman yang memungkinkan *developer* membangun aplikasi dan layanan web *XML*.
2. Sekumpulan *XML Web services* seperti *Microsoft .NET My Services*, yang membantu *developer* menghasilkan aplikasi yang sederhana dan terpadu.
3. Serangkaian *server* termasuk *Microsoft Windows Server 2003, Microsoft SQL Server* dan *Microsoft BizTalk Server* yang terintegrasi, untuk menjalankan, mengoperasikan dan mengelola aplikasi dan layanan berbasis web.
4. *Tool-tool* pengembang yang menyediakan IDE (*Integrated Development Environment*) untuk

memaksimalkan produktivitas pengembangan menggunakan *.NET Framework*.

5. Piranti lunak *client*, seperti *Windows XP, Windows CE* dan *Microsoft Office XP* yang membantu pengembang untuk menyebarkan dan mengelola aplikasinya.

2.4.2 Visual C#.NET

C# adalah bahasa pemrograman baru yang diciptakan oleh *Microsoft*, dikembangkan dibawah kepemimpinan Anders Hejlsberg. Anders dikenal sebagai salah satu pencipta *IDE Pascal*, yakni *Delphi*. Latar belakang Anders kemudian disatukan dengan aspek teknologi *C++* yang telah ada, sehingga hadirlah *C#* sebuah bahasa yang canggih.
Definisi *C#* dari *Microsoft* adalah sebagai berikut :

> *"C# is a simple, modern, object oriented and type-safe programming language derived from C and C++. C# is firmly planted in the C and C++ family tree of languages, and will immediately be familiar to C and C++ programmers. C# aim to combine the high productivity of Visual Basic and the raw power of C++"*

Ada dua kata kunci pada definisi diatas yakni *high productivity* dan *raw power C++*. *C#* hadir bukan saja sebagai official language bagi .NET, dia juga hadir sebagai alternatif bagi para pengembang yang berasal dari *C*, *C++* atau bahkan *Java* tetapi hadir dengan kemudahan *Visual Basic*. Sehingga pengembang dapat dengan mudah mengembangkan suatu solusi sistem informasi dengan konsep dan paradigma yang modern dibawah naungan teknologi *.NET*.

2.4.3 SQL Server 2000

Basis data adalah tempat penyimpanan data. Basis data tidak secara langsung menampilkan data ke pengguna, tetapi pengguna harus menjalankan aplikasi yang mengakses data dari basis data dan menampilkan dalam bentuk yang mudah dimengerti. Untuk bekerja dengan basis data, kita harus memakai sebuah bahasa. Bahasa basis data yang paling banyak dipakai adalah SQL (*Structured Query Language*).

Microsoft SQL Server 2000 adalah sistem manajemen basis data yang memakai perintah-perintah *Transact-SQL* untuk mengirim perintah dari komputer klien ke komputer server. *Transact-SQL* adalah bahasa SQL yang dikembangkan oleh Microsoft dengan menambahkan dialek tertentu. Microsoft SQL Server 2000 berisi basis data, mesin basis data, dan aplikasi yang diperlukan untuk mengelola data dan komponen-komponennya.

Keunggulan dari Microsoft SQL Server 2000 antara lain :

- Integrasi Internet.
  Mesin basis data SQL Server 2000 mendukung integrasi XML, juga mempunyai skalabilitas, ketersediaan dan keamanan yang diperlukan untuk beroperasi sebagai komponen penyimpan data. SQL Server 2000 juga mendukung *English Query* dan *Microsoft Search Service* untuk menyertakan *query* yang mudah dioperasikan dan kemampuan pencarian yang ampuh dalam aplikasi web.
- Skalabilitas dan Ketersediaan.
  Mesin basis data yang sama dapat digunakan dalam platform yang berbeda. SQL Server 2000 *Enterprise Edition* mendukung penggabungan server, *view*

berindeks dan mendukung memori besar yang mengijinkannya untuk menyesuaikan diri ke level kinerja yang diperlukan.

- Keunggulan Basis Data Tingkat *Enterprise*.
  Mesin basis data relasional SQL Server 2000 mendukung kebutuhan lingkungan pemrosesan data. Mesin basis data melindungi integritas data pada saat meminimalkan *overhead* dalam pengaturan pengguna yang memodifikasi basis data.
- Kemudahan instalasi, penyebaran dan penggunaan.
  SQL Server 2000 mencakup satu set administratif dan *tool* pengembangan yang meningkatkan proses penerapan, penyebaran, pengaturan dan penggunaan SQL Server pada beberapa lokasi. SQL Server 2000 juga mendukung suatu model pemrograman standar yang terintegrasi dengan Windows DNA, membuat penggunaan basis data dan gudang data SQL server sebagai bagian dari pembangunan sistem yang ampuh dan terskala.
- Penggudangan Data.
  SQL Server 2000 mencakup *tool* untuk mengekstrak dan menganalisa ringkasan data untuk pengolahan analisis *online*.