BAB II

LANDASAN TEORI

Dalam landasan teori ini akan dibahas mengenai: (1) *Microsoft® ActiveX® Data Objects* (ADO) dan *Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security* (ADOX), serta (2) *Open Database Connectivity* (ODBC). Dalam pembahasan ADO dijelaskan pula tentang tiga perbedaan model obyek ADO yang fungsinya ditemukan juga pada DAO. Ketiga model tersebut adalah ADO (ADODB), *Microsoft ADO Extensions for DDL and Security* (ADOX), dan *Microsoft Jet and Replication Objects (JRO)*.

2.1. *Microsoft® ActiveX® Data Objects* dan *Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security*

Microsoft® ActiveX® Data Objects (ADO) memungkinkan aplikasi klien untuk dapat mengakses dan memanipulasi data dari *database server* melalui penyedia OLE DB. Keunggulan ADO adalah kemudahan dalam penggunaan, kecepatan, penggunaan memori yang rendah, dan membutuhkan kapasitas yang sedikit. ADO merupakan pendukung utama dalam pengembangan aplikasi klien/*server* dan berbasis Web. Karena ADO digunakan dalam pengaksesan data dari berbagai sumber, dibutuhkan pemahaman konsep dasar sistem manajemen relasional *database*, konsep dasar *Online Analytical Processing* (OLAP), dan dasar internet dan protokol internet, dalam mengimplementasikannya. ADO adalah salah satu bagian dari strategi *Microsoft Universal Data Access* (UDA), dan digunakan

bersama dengan teknologi OLE DB. OLE DB berbasis pada *Microsoft Component Object Model* (COM). Karenanya, pemahaman terhadap COM juga akan sangat membantu dalam memahami beberapa konsep ADO lebih dalam.

Universal Data Access (UDA) adalah strategi *Microsoft* untuk menyediakan akses terhadap informasi dari suatu perusahaan. Ide dasar dari UDA adalah untuk dapat mengakses tiap data dari tempat data tersebut berada secara efisien, lebih daripada proses untuk memindahkan data untuk disentralisasikan ke *data store*. UDA adalah suatu *data store,* alat bantu, dan bahasa yang *independen*. UDA menawarkan suatu komponen tingkat tinggi, antarmuka yang mudah digunakan, dan komponen tingkat rendah, antarmuka dengan performansi tinggi secara praktis untuk tiap *data store* yang ada. UDA dapat digunakan secara fleksibel untuk mengintegrasikan *data store* yang berbeda dan digunakan pada alat bantu, aplikasi, dan *platform* untuk menciptakan solusi yang dibutuhkan.

Microsoft® Data Access Components (MDAC) SDK berisi komponen-komponen yang dibutuhkan oleh *Universal Data Access* (UDA) untuk dapat digunakan. Teknologi ini menyediakan kerangka dasar untuk tujuan umum dalam mengakses data pada sisitem operasi *Microsoft® Windows*. Ada tiga teknologi utama dalam MDAC. *ActiveX Data Objects* (ADO) adalah model obyek tingkat tinggi yang mengekspos data yang disediakan oleh OLE DB (Siebold, 2001). OLE DB merupakan komponen tingkat rendah antarmuka terhadap berbagai data store dengan performansi tinggi. ADO dan OLE DB dapat digunakan untuk data relasional (tabular) dan nonrelasional (hirarki). Dan yang terakhir adalah *Open Database Connectivity* (ODBC) yang merupakan komponen tingkat rendah,

antarmuka dengan performansi tinggi yang didesain secara khusus untuk *data store* relasional.

ADO menyediakan suatu layer abstraksi antara klien atau aplikasi middle-tier dan antarmuka tingkat rendah OLE DB. ADO menggunakan sekumpulan kecil objek otomasi yang menungkinkan untuk berhubungan dengan OLE DB secara sederhana dan efisien. Antarmuka ini menjadikan ADO pilihan yang tepat bagi pengembang, yang ingin mengakses data tanpa harus mempelajari COM dan OLE DB secara detail, dengan menggunakan bahasa tingkat yang lebih tinggi, seperti *Visual Basic* dan bahkan *VB Script*.

Sedangkan *Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security* (ADOX) adalah salah satu ekstensi obyek ADO dan suatu model pemrograman. ADOX dapat membuat dan memanipulasi obyek skema, seperti pengamanan. Karena ADOX merupakan suatu pendekatan obyek dasar untuk memanipulasi skema, penulisan kode akan dilakukan pada berbagai macam data sumber tanpa memperhatikan perbedaan sintak aslinya. ADOX merupakan suatu kamus yang intinya tentang obyek ADO. ADO dapat membuat obyek tambahan, mengubah, dan menghapus obyek skema, seperti tabel dan prosedur. ADO juga mengamankan obyek dari pemakai dan mengijinkan dan mencabut izin pada suatu obyek.

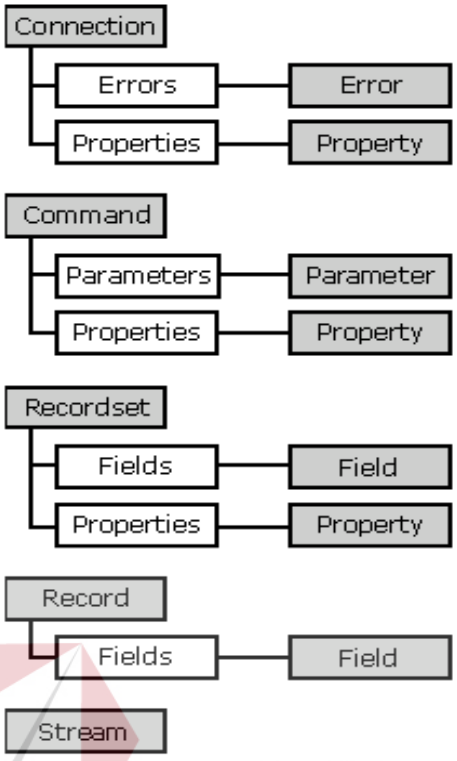
Ada tiga perbedaan model obyek ADO yang fungsinya ditemukan bersama-sama pada DAO. Ketiga model itu adalah ADO (ADODB), *Microsoft ADO Extensions for DDL and Security* (ADOX), dan *Microsoft Jet and Replication Objects* (JRO). Fungsi DAO adalah membagi antara tiga model tersebut karena

banyak aplikasi yang hanya membutuhkan satu sub dari fungsinya. Dengan memecah fungsi-fungsi itu, aplikasi tidak perlu mengeluarkan biaya tambahan untuk menambah informasi sampai memori yang tidak perlu. Pada bagian-bagian berikut akan diberikan gambaran ikhtisar tentang ketiga model obyek tersebut.

***a) ActiveX Data Objects* (ADO)**

ADO memungkinkan klien untuk mengakses suatu aplikasi dan memanipulasi data melalui *provider* OLE DB. ADO berisi obyek-obyek yang menghubungkan suatu sumber data dan dapat membaca, menambah, memperbaharui, ataupun menghapus data.

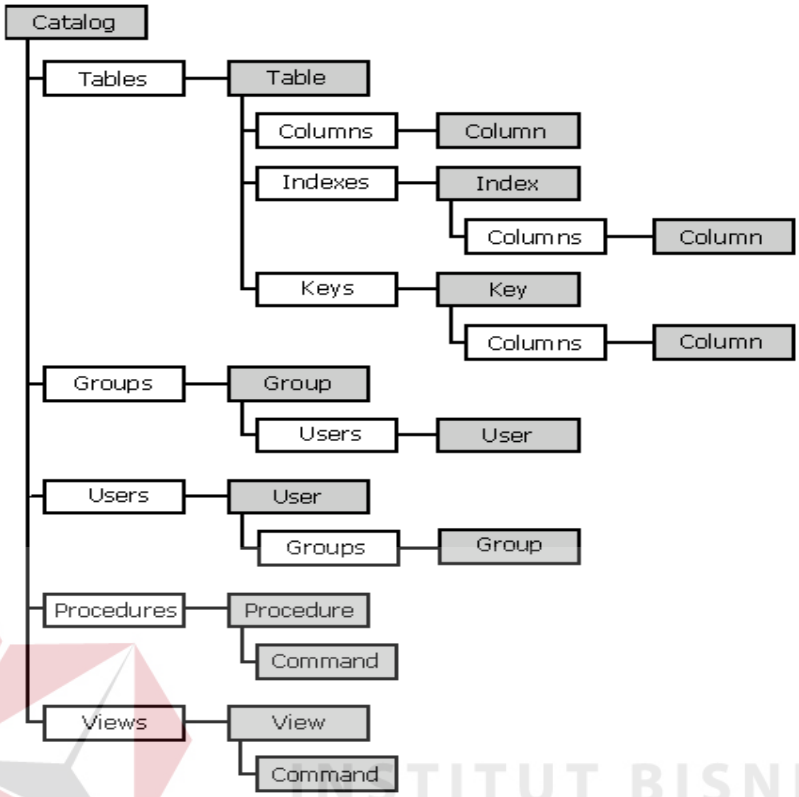
Gambar 2.1, menjelaskan bahwa hubungan (*connection*) pemakai sumber data khusus terhadap suatu obyek. Bedanya ADO dengan DAO terdapat pada obyek *workspace* bagi pemakai dan obyek *database* menjelaskan tentang sumber data. Perintah (*command*) obyek ADO sama dengan *querydef* pada obyek DAO dimana kedua obyek dapat digunakan pada saat menjalankan pernyataan SQL pada suatu sumber data. Demikian juga, *Recordset* dalam kedua obyek ADO dan DAO dapat digunakan untuk melihat isi tabel ataupun hasil dari pernyataan SQL.



Gambar 2.1. Diagram Obyek ADO

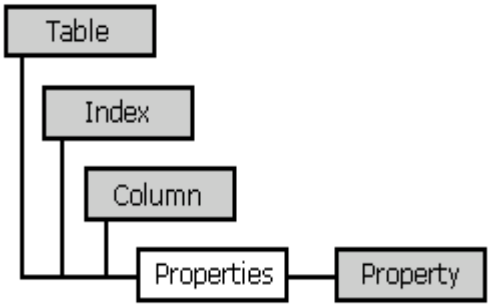
b) ***ActiveX Data Objects Extensions for Data Definition Language and Security* (ADOX)**

Obyek-obyek dalam model ADOX menyatakan obyek *database* seperti tabel, tampilan, dan indeks (Siebold, 2001). Dengan ADOX, suatu pengelola dapat mengkontrol skema *database* dan dan mengijinkan dan mencabut izin pemakai dan grup pada suatu obyek.



Gambar 2.2. Diagram Obyek ADOX

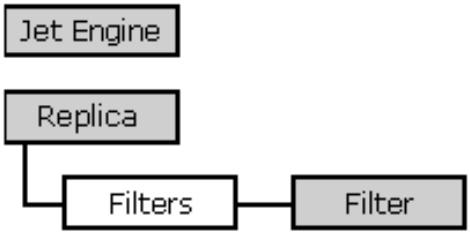
Gambar 2.2 menjelaskan bahwa obyek *katalog* merupakan suatu wadah kumpulan definisi data (*table, procedur,* dan *view*) dan kumpulan pengaman (*user* dan *grup*). Yang berbeda dari DAO adalah obyek *database* berisi kumpulan definsi data dan obyek *workspace* berisi kumpulan pengaman. Setiap obyek *katalog* merupakan penghubung yang hanya dengan satu penghubung saja sedangkan suatu *workspace* DAO dapat berisi banyak *database*. Obyek *Tabel*, *Index*, dan C*olumn* dalam ADO hampir sama dengan obyek *tabeldef, indekdef,* dan *field* pada DAO. Setiap obyek tersebut juga mempunyai kumpulan *properties* standar ADO yang dapat dilihat pada gambar 2.3 berikut ini.



Gambar 2.3. Diagam Obyek Standar ADO.

***c) Jet and Replication Objects* (JRO)**

JRO berisi obyek, propertis, dan metode untuk membuat, mengubah, mengsinkronisasikan suatu *replica* (tiruan). JRO didisain khusus yang digunakan untuk *provider Microsoft Jet*. ADO dan ADOX tidak sama, JRO tidak dapat digunakan oleh sumber data lain selain *database Microsoft Jet* (Haught, 1995). Obyek utama pada model JRO adalah obyek *replica* (tiruan). Obyek *replica* digunakan untuk membuat tiruan-tiruan baru, untuk mendapatkan kembali dan memodifikasi *properties* pada suatu tiruan yang sudah ada, dan mengganti sinkronisasi dengan tiruan yang lain. Yang berbeda dari DAO adalah tugas-tugas pada suatu obyek *database*. JRO juga menyertakan keistimewaan obyek *JetEngine* pada dua mesin khusus *database Microsoft Jet*, yaitu *database* tersusun rapi dan memperbaharui data pada memori cadangan.



Gambar 2.4. Diagram Objek JRO.

2.2. Open Database Connectivity

Open Database Connectivity (ODBC) adalah suatu *access database* yang dapat diterima oleh semua Application Programming Interface (API). Dasar ODBC adalah spesifikasi X/Open untuk Call-Level Interface (CLI) dan ISO/IEC untuk database API dan menggunakan Structured Query Language (SQL) sebagai bahasa access database (Haught, 1995). ODBC didesain agar database dapat dijalankan semaksimal mungkin – dengan begitu, kemampuan suatu aplikasi untuk mengakses berbeda dengan Database Management Systems (DBMS) dengan kode asal yang sama. Aplikasi database berfungsi pada antarmuka ODBC, pengimplementasian modul khusus database disebut dengan drivers. Penggunaan driver akan memisahkan suatu aplikasi dari database khusus yang diambil dengan cara yang sama yaitu driver pencetak akan memisahkan program pengolah kata dari perintah khusus pencetak database. Karena driver diisi pada waktu proses, seorang pemakai hanya bisa menambahkan driver baru untuk mengakses sebuah DBMS baru, jadi tidak perlu menyusun lagi atau menghubungkan lagi sebuah aplikasi.

Menurut sejarah, perusahan-perusahaan banyak yang menggunakan satu DBMS. Seluruh *database* yang diakses salah satunya harus dikerjakan melalui

tujuan awal sistem atau melalui aplikasi yang ditulis untuk sistem kerja semata. Akan tetapi, bagi pemakai komputer dapat mengikuti perkembangan komputer, perangkat keras komputer dan juga perangkat lunak, perusahaan-perusahaan sudah mulai untuk mendapatkan DBMS yang berbeda dari sebelumnya. Alasannya, orang-orang selalu membeli yang murah, yang cepat, yang telah mereka ketahui, yang terbaru dipasaran, yang bekerja dengan baik pada sebuah aplikasi. Alasan lain adalah untuk reorganisasi dan *merger*, sebelumnya hanya beberapa orang dalam departemen yang mengetahui suatu DBMS.

Pokok permasalahannya adalah semakin kompleksnya sebuah komputer. Komputer tersebut menghasilkan sebuah perlengkapan *query*, analisa, dan menunjukkan datanya, selama terbilang murah, *database* sangat mudah digunakan. Maka dari itu, sebuah badan hukum seringkali menguraikan banyak data yang tersebar dalam *desktop, server,* dan komputer mini, disimpan pada berbagai macam *database* yang tidak cocok, dan diakses dengan menggunakan banyak kode yang berbeda, hanya beberapa orang saja yang bisa mendapatkan data lengkap.

Klien/*server* akan memperhitungkan tantangan yang terakhir, yang mana membuat komputer induk mencoba lebih efisien pada penggunaan percobaan. Murahnya komputer pribadi (klien) menempatkan *desktop* dan menyediakan dua buah grafik bagian akhir data dan perlengkapan yang harganya murah, seperti *spreadsheet*, program pemetaan, dan pembuat laporan. Komputer mini dan komputer yang berukuran besar (*server*) pemakai DBMS, dimana mereka dapat menggunakan kemampuan menghitung dan terpusat pada kecepatannya,

menyerasikan pengaksesan data. Bagaimana menghubungkan tujuan awal *software* dengan tujuan akhir *database*?

Sebuah masalah yang serupa dihadapkan pada *Independent Software Vendors* (ISVs). Pengisian *database software vendor* pada komputer mini dan komputer yang berukuran besar biasanya direkayasa untuk diisi pada salah satu versi aplikasi untuk masing-masing DBMS atau mengisi kode khusus DBMS pada masing-masing DBMS yang ingin diakses. Pengisian *vendor* suatu *software* untuk komputer pribadi harus diisi pengakses data secara rutin untuk masing-masing DBMS yang berbeda yang ingin mereka kerjakan. Seringkali banyak maksud yang bersumber pada penulisan yang dipakai dan pemeliharaan data yang sering diakses oleh suatu aplikasi, dan seringkali suatu aplikasi tidak menyediakan kualitas tapi apakah mereka dapat mengakses data yang diberikan DBMS. Kedua perangkat pengembang tersebut membutuhkan cara yang berbeda untuk mengakses data dalam DBMS. Kelompok komputer besar dan komputer mini membutuhkan cara untuk menggabungkan data dari DBMS yang lain pada sebuah aplikasi, sedangkan kelompok komputer pribadi membutuhkan kemampuan lebih sebagai cara untuk menulis sebuah aplikasi yang salah satunya DBMS itu sendiri. Singkatnya, kedua grup tersebut membutuhkan cara untuk mengakses data, mereka membutuhkan penghubung *database* yang lain.

Banyak yang salah paham tentang keberadaan ODBC dalam memperhitungkan kata. Menurut pemakai, ODBC adalah sebuah *icon* pada *Microsoft® Windows® Control Panel*. Bagi *programmer* suatu aplikasi, ini merupakan suatu perpustakaan yang berisi data yang diakses secara rutin.

Sedangkan bagi yang lain, ODBC adalah penjawab untuk semua permasalahan pengaksesan *database*. Yang utama dan yang terpenting, ODBC adalah suatu spesifikasi *database* API. API adalah salah satu *database* yang independen atau sebuah sistem operasi, walaupun masih menggunakan bahasa C yang manual, ODBC API adalah bahasa yang independent (Haught, 1995). ODBC API berdasarkan pada spesifikasi CLI dari X/*Open* dan ISO/IEC. Peralatan kedua spesifikasi ODBC tersebut tiga kali lebih lengkap – spesifikasi ini berdasarkan pada ODBC versi yang terdahulu tetapi peralatannya tidak selengkap ODBC yang sekarang – dan biasanya ditambah keistimewaan yang diperlukan oleh pengembang untuk menyaring aplikasi *database* pokok, seperti kursor penggulung.

Fungsi ODBC API diimplementasikan oleh pengembang pada *driver* khusus DBMS. Aplikasinya akan memanggil fungsi *driver* ini untuk mengakses data dalam suatu DBMS yang independen. Suatu *Driver Manager* mengatur komunikasi antara aplikasi dan *driver*. Meskipun *Microsoft* menyediakan sebuah *Driver Manager* pada komputer untuk menjalankan *Microsoft Windows NT® Server/Windows 2000 Server, Microsoft Windows NT Workstation/Windows 2000 Professional*, dan *Microsoft Windows® 95/98*, beberapa orang mengisi data dari *driver* ODBC, dan memanggil fungsi ODBC dari beberapa aplikasi, setiap orang dapat mengisi data dari aplikasi ODBC dan *driver*. Faktanya, sebagaian besar aplikasi ODBC dan *driver* yang ada pada komputer dijalankan dari *Windows NT Server/Windows 2000 Server, Windows NT Workstation/Windows 2000 Professional*, dan *Windows 95/98* yang diproduksi oleh perusahaan *Microsoft* lain.

Lagi pula, *driver* ODBC dan aplikasinya ada pada *Macintosh*® dan berbagai macam *platform* UNIX.

Untuk membantu aplikasi dan pengembang driver, Microsoft memberikan ODBC *Software Development Kit* (SDK) bagi komputer yang menjalankan *Windows NT Server/Windows 2000 Server, Windows NT Workstation/Windows 2000 Professional*, dan *Windows 95/98* yang berisi *Driver Manager, installer DLL, test tools,* dan contoh-contoh aplikasi. *Microsoft* mempunyai tim pada *Visigenic Software* untuk tempat sementara SDK pada *Macintosh* dan berbagai platform UNIX.

Ini sangat penting untuk dimengerti bahwa ODBC didesain untuk meng-*expose* kemampuan *database*, bukan menambah *database*. Jadi, pengisi aplikasi tidak akan menyangka bahwa menggunakan ODBC begitu sulit untuk mengubah suatu *database* sederhana menjadi suatu mesin penghubung *database* yang sangat istimewa. *Driver* tidak berharap pengisi dapat mengimplementasikan fungsi yang tidak ada pada *database* utama. Kecuali jika pengembang yang mengisi *driver* langsung mengakses data file (seperti data pada file Xbase) yang harus ditulis oleh mesin pembantu *database* minimal fungsi SQL. Pengecualian yang lain jika komponen ODBC pada *Microsoft® Data Access Components* (MDAC) SDK menyediakan sebuah kursor penggulung daftar simulasi pada *driver* yang mengimplementasikan suatu level fungsi tertentu.

Penggunaan aplikasi ODBC bertanggung jawab atas setiap fungsi *database* gabungan. Misalnya, ODBC tidak berhubungan dengan mesin lain, maupun

menyebarkan pemroses transaksi. Bagaimanapun, dikarenakan kebebasan DBMS, ini dapat digunakan untuk membuat komponen *database* gabungan.

Dalam membangun ODBC harus mempunyai empat komponen, yaitu:

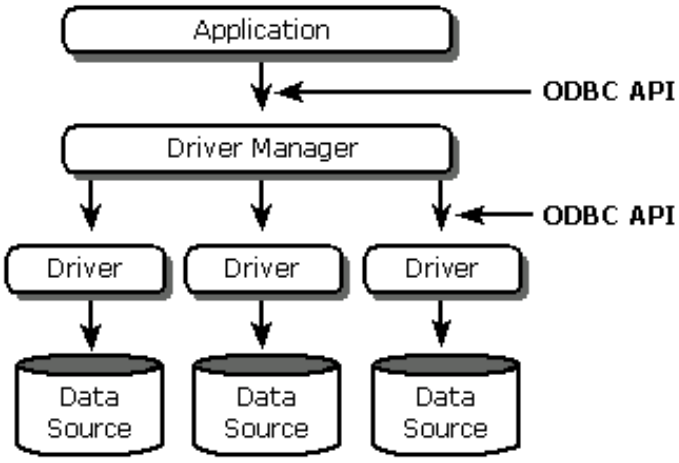
a) ***Application*.** Pemrosesan dan pemanggilan fungsi ODBC dilakukan untuk mengajukan perintah SQL dan mendapatkan hasilnya.

b) ***Driver Manager*.** *Diver* yang diisi dan yang tidak diisi atas nama sebuah aplikasi. Pemrosesan fungsi ODBC dengan memanggil atau hanya melewati fungsi itu ke dalam sebuah *driver.*

c) ***Driver*.** Pemrosesan fungsi ODBC dengan memanggil, mengajukan perintah SQL untuk sebuah sumber data khusus, dan hasilnya dikembalikan ke suatu aplikasi. Jika perlu, *driver* memodifikasi sebuah pengaksesan aplikasi dan juga pengaksesan harus sesuai dengan sintak pembantu yang menghubungkan DBMS.

d) ***Data source*.** Terdiri dari data pemakai yang ingin mengakses dan menghubungkan sistem operasi, DBMS, dan jaringan *platform* (jika ada) digunakan untuk DBMS.

Hubungan antara keempat komponen tersebut dapat ditunjukan oleh gambar berikut ini.



Gambar 2.5. Diagram Penyilangan *Driver* dan Data Sumber.

Penjelasan gambar 2.5 adalah sebagai berikut. Pertama, penyilangan *driver* dan data sumber harus ada, secara bersama-sama aplikasi mengakses data lebih dari satu data sumber. Kedua, ODBC API digunakan pada dua tempat yaitu antara aplikasi dan *Diver Manager*, dan antara *Diver Manager* dan setiap *driver*. Antarmuka antara *Diver Manager* dan *driver* kadang-kadang diserahkan pada *Service Provider Interface* (SPI). Pada ODBC, *Application Programming Interface* (API) dan S*ervice Provider Interface* (SPI) adalah sama. Berarti, *Driver Manager* dan setiap *driver* mempunyai antarmuka yang sama pada fungsi yang sama.

Kata pengantar pada buku pedoman pemrograman ADO membahas tentang hubungan antara ADO dan meletakkan pemrograman ADO pada *Microsoft Universal Data Access architecture*. OLE DB mendefinisikan sekelompok antarmuka COM agar aplikasi dapat mengakses data yang sama yang disimpan dalam sumber informasi yang berbeda. Pendekatan ini memberikan sebuah sumber

data agar diberikan pada data melalui antarmuka yang banyak membantu fungsi DBMS dengan menyediakan sumber data. Dengan desain, arsitektur performansi OLE DB yang tinggi berdasarkan pada penggunaan yang fleksibel, model dasar pengerjaan komponen. Daripada menentukan nomor pada sebuah tumpukan perantara antara aplikasi dan data, OLE DB hanya membutuhkan beberapa komponen untuk menyempurnakan sebuah tugas khusus.

Misalnya, mengharuskan seorang pemakai menjalankan sebuah *query*. Dengan mempertimbangkan persyaratan berikut:

a) Data yang terletak pada sebuah *database* relasional yang mana sekarang ini ada pada suatu *driver* ODBC tapi *provider* OLE DB tidak asli. Suatu aplikasi yang menggunakan ADO membahas tentang *provider* OLE DB pada ODBC, yang berisi *driver* ODBC yang sesuai. *Driver* mengabaikan perintah SQL pada DBMS, yang tetap menghasilkan data.

b) Data yang terletak pada *Microsoft SQL Server* yang *provider* OLE DB-nya tidak asli. Suatu aplikasi yang menggunakan ADO membahas langsung tentang *provider* OLE DB pada *Microsoft SQL Server*. Tidak perlu perantara.

c) Data yang terletak pada *Microsoft Exchange Server*, tapi *provider* OLE DB tidak dapat meng-*expos*e mesin yang memproses *query* SQL. Aplikasi yang menggunakan ADO membahas tentang *provider* OLE DB pada *Microsoft Exchange* dan memanggil komponen *processor query* OLE DB untuk diselesaikan.

d) Data yang terletak pada sistem file *Microsoft NTFS* dalam bentuk dokumen. Data yang diakses dengan menggunakan sebuah *provider* OLE DB asli di atas

Microsoft Indexing Service, yang mana berisi indek dan memiliki dokumen yang memungkinkan pencarian sistem file yang efisien.

Pada contoh-contoh terdahulu, aplikasi bisa saja meragukan suatu data. Pemakai membutuhkan beberapa komponen yang harus dipenuhi. Pada kasus ini, komponen tambahan digunakan jika perlu, dan hanya membutuhkan komponen yang diperlukan saja. Isi dari persyaratan di atas dapat dipergunakan kembali dan komponen besar untuk menambah performansi yang tinggi ketika OLE DB digunakan.

Ada dua kategori *provider* tingkat tinggi, yaitu kategori pelengkap data dan kategori pelengkap layanan. Sebuah *data provider* memiliki data sendiri dan dapat meng-*expose* bentuk tabular ke dalam suatu apliksi. Sebuah *service provider* harus membantu memproduksi dan memakai data, merupakan ciri-ciri tambahan yang ada pada sebuah aplikasi ADO. Sebuah *service provider* dapat juga mendefinisikan lebih lanjut tentang suatu *service component*, yang bekerja dengan kata penghubung pada *service provider* atau komponen lain. ADO memberikan sesuatu yang sesuai, antarmuka level tinggi sampai berbagai macam *provider* OLEDB.