Bab 15

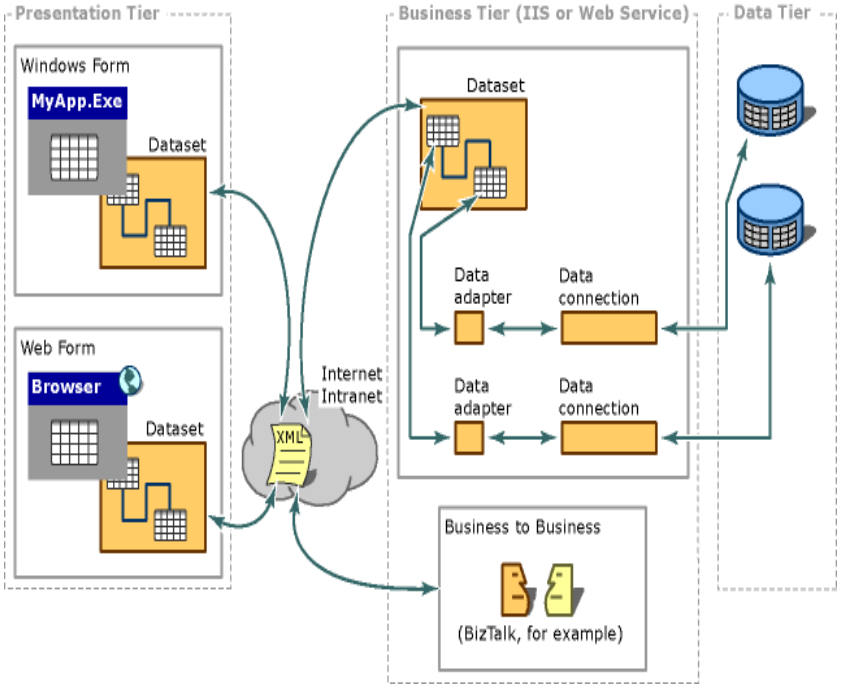
ADO.NET (2)

Pada kerangka kerja ADO.NET terdapat dua teknologi yaitu *connection oriented* dan *connectionless.* Teknologi *connection oriented* telah dibahas pada bab 14, dan teknologi *connectionless* dibahas pada bab 15 ini. Diawali dengan motifasi bahwa sebuah aplikasi banyak diakses oleh pengguna secara serentak (pararel) sehingga operasi terhadap sebuah sumber data harus dijamin keberhasilnnya, maka dengan teknologi *connection oriented* , satu pengguna akan menghabiskan banyak waktunya untuk sebuah koneksi. Dengan demikian harga yang harus dihitung untuk sejumlah pengguna akan semakin besar.

Persoalan pada *connection-oriented* diatas memberikan sebuah ide untuk mengembangkan teknologi *connectionless* dengan cara:

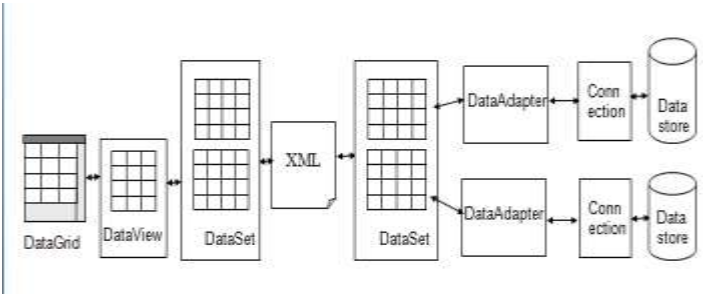
- Menyiapkan ruang memori yang cukup pada memori utama sebagai *cache* data pada sumber data.
- Waktu koneksi diusahakan sesingkat mungkin untuk membaca dan meng-update data
- Data yang ada di memori bersifat *independent* terhadap sumber data.

Pengembangan teknologi ADO.NET salah satunya ditujukan untuk memenuhi pengembangan aplikasi yang memiliki 3 lapisan seperti yang ditunjukkan pada gambar 15.1 dibawah ini:



Gambar 15.1 Arsitektur Microsoft 3-Tier

Terdapat tiga lapisan pada arsitektur diatas yaitu lapisan data, bisnis dan presentasi. Pada tingkat implementasi, penerapan tiga lapisan diatas, dapat dilakukan dengan memakai teknologi ADO.NET, seperti yang ditunjukkan pada gb 15.2:



Gambar 15.2 Rantai Teknologi ADO.NET

Lapisan presentasi diwakili dengan datagrid yang datanya disediakan oleh sebuah dataview. Representasi dataview tunggal dapat terdiri dari beberapa dataset, dimana masing-masing memiliki dataAdapter dengan sumber data yang berbeda-beda.

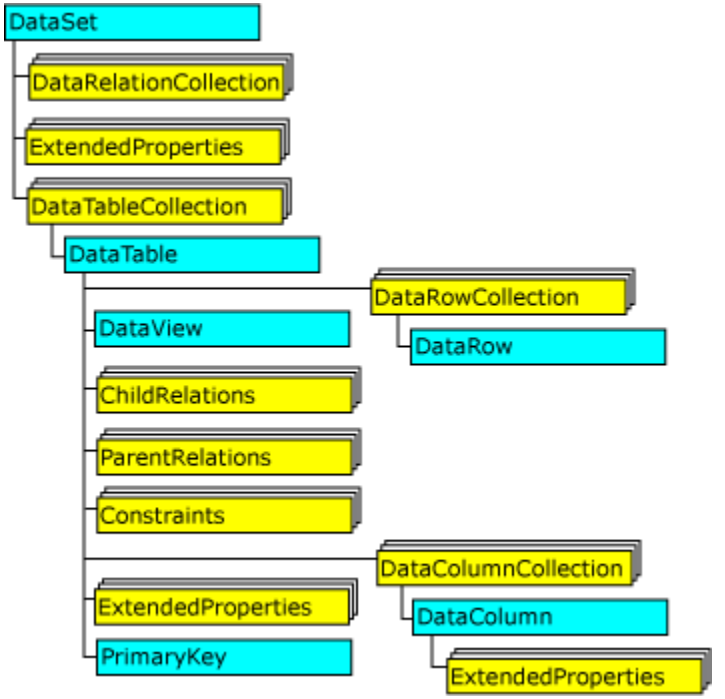
15.1 DataSet

Komponen dari ADO.NET pada teknologi connectionless adalah DataSet; konsep ini menyesuaikan dengan konsep recordset ADO pada vb sebelumnya. Perbedaan paling nyata antara ADO dan Data Set adalah pada koneksi. Data Set selalu *disconnect*, sehingga Data Set dapat dipakai untuk manipulasi data secara internal di memori client. Kemudian, ketika dataset perlu update ke database, maka perlu mempergunakan DataAdapter sebagai satu perantara di antara DataSet dan data penyedia .NET

Suatu obyek DataSet adalah titik utama teknologi untuk mendukung teknologi *connectionless*, penyebaran data secara. Obyek DataSet adalah sebuah representasi data yang ada di *memory-resident* yang menyediakan data untuk pemrograman dengan model database relasional tanpa memandang sumber datanya. Pengelolaan data dengan DataSet berarti sumber data dapat terdiri dari beberapa sumber misalnya data XML, data lokal dsb.

Representasi DataSet dinyatakan sebagai satu set lengkap data, termasuk tabel terkait, kendala, dan hubungan antara tabel.

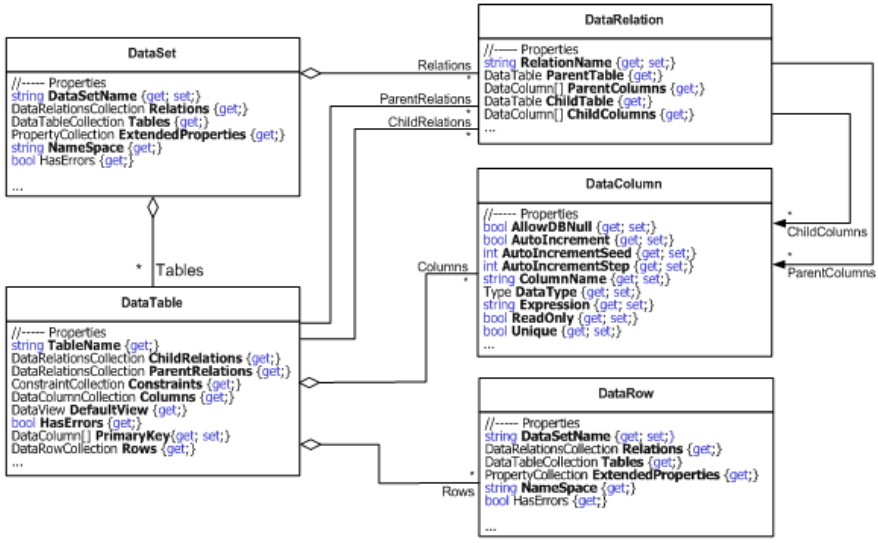
Model obyek dataset dapat dilihat pada gb 15.3 yang menunjukkan kelas dataset memiliki properti koleksi tabel, koleksi relasi.



Gambar 15.3 Model DataSet

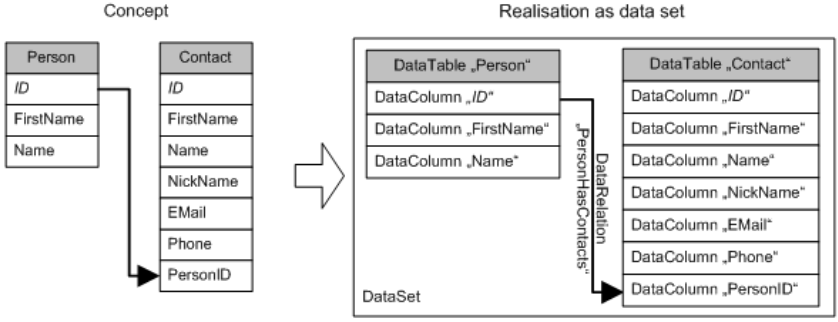
DataSet berada di memori sebagai hasil pemetaan terhadap sebuah database yang ditentukan beserta relasinya. Hasil pemetaan ini, sebuah tabel dianggap sebagai obyek, mengacu pada model pemrograman berbasis obyek. Isi dari dataset paling tidak terdiri dari kolekasi DataTable dan koleksi DataRelations. Sebuah DataTable berisi koleksi DataColumn dan koleksi DataRows, sedangkan sebuah DataRelations berisi definis dari asosiasi dua tabel yang terdiri dari definis parentTable dan childTable.

Gambaran kelas diagram dari DataSet secara umum dapat dilihat pada gambar 15.4,



Gambar 15.4 Kelas Diagram DataSet

Sebagai contoh pemetaan sebuah konsep database ke bentuk dataset, ditunjukkan pada gambar 15.5, dibawah ini:



Gambar 15.5 Pemetaan konsep ke dataset

Dua tabel yaitu person dan contact dengan relasi antara person dan contact lewat field parent adalah ID dan field child adalah PersonID. Pada pemetaan gb 15.5, semua tabel dan field dipetakan dalam DataTable, sedangkan bentuk relasinya dipetakan dalam DataRelations.

Langkah-langkah yang dilakukan untuk pemetaan diatas adalah

1. Pendefinisian skema
2. Pendefinisian data
3. Akses data

Secara pemrograman, langkah tsb adalah:

1. Pembuatan DataSet dan DataTable "Person"

```
DataSet ds = new DataSet("PersonContacts");
DataTable personTable = new DataTable("Person");
```

2. Definiskan kolom 'ID' dan atur propertinya

```
DataColumn col = new DataColumn();
col.DataType = typeof(System.Int64);
col.ColumnName = "ID";
col.ReadOnly = true;
col.Unique = true;
col.AutoIncrement = true;
col.AutoIncrementSeed = -1;
col.AutoIncrementStep = -1;
```

3. Tambahkan kolom ke tabel dan atur primarykey-nya

```
personTable.Columns.Add(col);
personTable.PrimaryKey = new DataColumn[ ] { col };
```

4. Buat kolom yang lain, kemudian tambahkan ke table

```
col = new DataColumn();
col.DataType = typeof(string);
col.ColumnName = "FirstName";
personTable.Columns.Add(col);

col = new DataColumn();
```

```
col.DataType = typeof(string);
col.ColumnName = "Name";
personTable.Columns.Add(col);
```

5. Tambahan tabel ke dataset

```
ds.Tables.Add(personTable);
```

6. Ulangi langkah pembuatan tabel person dengan pembuatan tabel contact seperti dibawah ini:

```
DataTable contactTable = new DataTable("Contact");
...
ds.Tables.Add(contactTable);
```

Setelah dua tabel terbentuk, langkah berikutnya adalah membuat relasinya, secara pemrograman langkah tsb ditunjukkan dibawah ini:

```
DataColumn parentCol = ds.Tables["Person"].Columns["ID"];
DataColumn childCol =
             ds.Tables["Contact"].Columns["PersonID"];
DataRelation rel = new DataRelation("PersonHasContacts",
             parentCol, childCol);
ds.Relations.Add(rel);
```

Setelah dataset terbentuk beserta relasinya, maka berikutnya dataset telah siap untuk dilakukan operasi data, antara lain operasi penambahan data dan update data ke sumber data.
Operasi penambahan data dilakukan dengan cara:

1. Buat baris baru dan berikan nilai pada masing-masing kolom:

```
DataRow personRow = personTable.NewRow();
personRow[1] = "Wolfgang";
personRow["Name"] = "Beer";
```

2. Tambahkan baris ke table

```
personTable.Rows.Add(row);
```

Ulangi langkah diatas pada tabel Contact, yaitu

```
DataRow contactRow = contactTable.NewRow ();
contactRow[0] = "Wolfgang";
...
contactRow["PersonID"] = (long)personRow["ID"]; // defines
relation
contactTable.Rows.Add (row);
```

Simpan penambahan data diatas ke sumber data secara permanen, hal ini dapat dilakukan dengan perintah: `ds.AcceptChanges();`

Cara untuk mengakses sebuah dataset, yaitu dapat dilakukan dengan membaca data tsb satu persatu secara iterasi, contohnya adalah pada tabel 'personTable' dibawah ini:

```
foreach (DataRow person in personTable.Rows) {
 Console.WriteLine("Contacts of {0}:", person["Name"]);
```

Cara lain, untuk akses data yang memiliki relasi, seperti relasi PersonHasContacts, pada contoh sebelumnya, ditunjukkan pada program dibawah ini:

```
foreach (DataRow contact in
       person.GetChildRows("PersonHasContacts"))
 {
       Console.WriteLine("{0}, {1}: {2}",
       contact[0], contact["Name"],
       contact["Phone"] );
}
```

Perubahan-perubahan yang terjadi pada dataset, harus di jaga agar perubahan itu juga dilakukan di sumber data. Teknik untuk melakukannya yakni:

```
       ...
       if (ds.HasErrors) {
              ds.RejectChanges();
       } else {
              ds.AcceptChanges();
       }
}
```

Data-data yang terdapat pada dataset dapat diketahui nilainya, apakah telah berubah atau tidak, metode ini memakai konsep DataRow Version sebagai nilai enumerasi

```
public enum DataRowVersion {
       Current, Original, Proposed, Default
}
```

- Current: Nilai saat ini
- Original: Nilai asli
- Proposed: Nilai yang dip roses saat ini
- Default: standard, sebagai dasar dari DataRowState

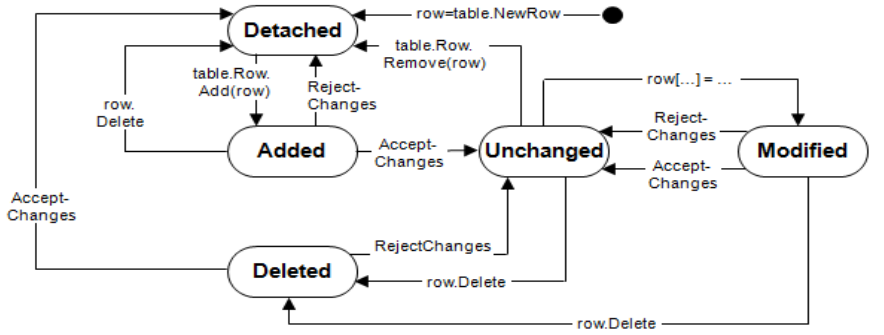
DataRowState	Default
Added, Modified, Unchanged	Current
Deleted	Original
Detached	Proposed

Contoh pemakaian DataRow Version sebagai berikut ini

```
bool hasOriginal =
              personRow.HasVersion(DataRowVersion.Original);
if (hasOriginal) {
string originalName = personRow["Name",
                     DataRowVersion.Original];
}
```

Selain memiliki versi data, dataRow juga memiliki state sebagai penunjuk tentang status data saat ini, Nilai status ini berupa property yang melekat pada kelas dataRow: `public DataRowState RowState {get;}` Nilai RowState adalah enumerasi dari beberapa status antara lain:

```
public enum DataRowState {
       Added, Deleted, Detached, Modified, Unchanged
}
```



Gambar 15.6 State DateSet

15.2 Penentuan DataSet vs DataReader

Konsep dataReader yang telah dijelaskan pada bab 14 dengan konsep dataset memiliki beberapa pandangan sebagai panduan pilihan berdasarkan pertimbangkan jenis fungsionalitas. Gunakan DataSet untuk melakukan hal berikut:

- Aplikasi hanya memerlukan Cache data local ataupun hanya perlu membaca hasil query
- Memakai Remote data diantara lapisan atau dari sebuah layanan web XML.
- Berinteraksi dengan data dinamis seperti mengikat kontrol Windows Forms atau menggabungkan data dari berbagai sumber.
- Melakukan pengolahan data tanpa memerlukan sambungan yang intens, sehungga sambungan dapat dibebaskan dan dapat digunakan oleh klien lain.

Jika tidak memerlukan fungsionalitas yang disediakan oleh DataSet, pilihan dataReader dapat meningkatkan kinerja aplikasi dengan pilihan *read-only* atau *forward only*.. Meskipun DataAdapter menggunakan dataReader untuk mengisi data dari suatu DataSet.

15.3 DataView

Sebuah DataView memungkinkan Anda untuk membuat tampilan yang berbeda dari data yang tersimpan dalam sebuah

DataTable , sebuah kemampuan yang sangat sering digunakan dalam aplikasi yang mengikat data. Pada penggunaan DataView, data dapat diekspos dalam tabel dengan urutan berbeda, dapat menyaring data lewat status baris data atau berdasarkan ekspresi filter.

Sebuah DataView menyediakan tampilan dinamis dari data dalam DataTable yang mendasari: konten, urutan dan keanggotaan yang mencerminkan perubahan yang terjadi. Perilaku ini berbeda dari metode *select* dari DataTable, yang mengembalikan sebuah array DataRow dari tabel berdasarkan / filter dan atau jenis urutan tertentu: isi ini mencerminkan perubahan yang terjadi di table, tetapi keanggotaan dan urutan datanya tetap statis. Kemampuan dinamis dari DataView membuatnya ideal untuk aplikasi data-yang mengikat (*data-binding*).

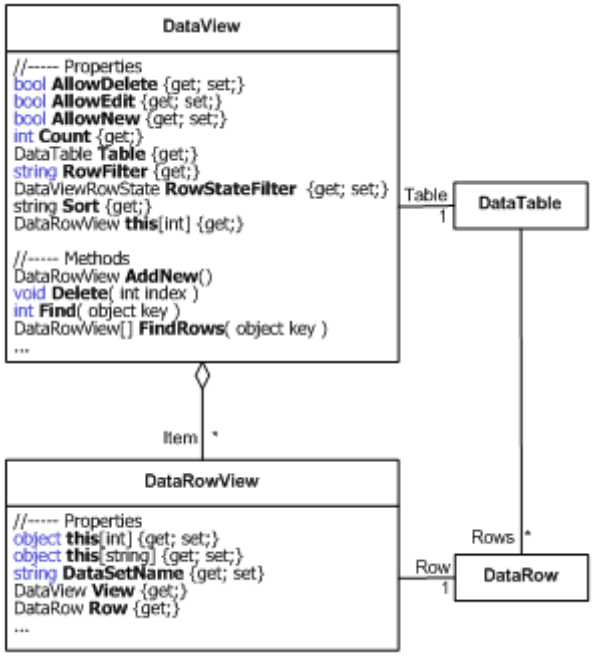
Sebuah DataView menyediakan data dengan tampilan yang dinamis dalam satu set data, seperti tampilan database, yang dapat menerapkan urutan data yang berbeda dan kriteria penyaringan. Tidak seperti tampilan database, Suatu DataView tidak dapat diperlakukan sebagai sebuah tabel dan tidak bisa mendeskripsikan relasi dari tabel Penyajian data dalam dataView juga tidak dapat mengecualikan kolom yang ada dalam tabel sumber data, juga tidak bisa menambahkan kolom, seperti kolom perhitungan, yang tidak ada dalam tabel sumber.

DataView dapat menggunakan DataViewManager untuk mengelola pengaturan tampilan untuk semua tabel dalam sebuah DataSet. DataViewManager menyediakan cara yang mudah untuk mengelola tampilan untuk pengaturan default untuk setiap tabel.

Pengaturan tampilan didasarkan pada beberapa property antara lain:

- RowFilter: Memfilter data berdasarkan filter expression
- RowStateFilter: Memfilter data berdasarkan kondisi row
- Sort: Mengurutkan data berdasarkan suatu columns

Berikut ini kelas diagram dari DataView secara umum



Gambar 15.7 Kelas Diagram DataView

Kemampuan DataView antara lain: dapat melakukan perubahan data, pencarian data secata cepat ketika data telah terurut. Sebuah obyek DataView dapat dijadikan sumber data bagi elemen GUI.

Contoh untuk membuat obyek DataView, mengatur filter dan pengurutan data, seperti yang ditunjukkan dibawah ini:

```
DataView a_kView = new DataView(personTable);
dataView.RowFilter = "FirstName <= 'K'";
dataView.RowStateFilter =
             DataViewRowState.Added |
DataViewRowState.ModifiedCurrent;
dataView.Sort = "Name ASC";
```

Dari program diatas, dataView yang telah dibuat, dapat ditampilkan dalam sebuah elemen GUI seperti dbawah ini:

```
DataGrid grid = new DataGrid();
...
grid.DataSource = dataView;
```

Pencarian data secara cepat dapat dilakukan berdasarkan kolom name:

```
int i = dataView.Find("Beer");
grid.Select(i);
```

15.4 DataAdapter

Sebuah DataAdapter merupakan bagian integral dari ADO.NET dikelola oleh penyedia, yang merupakan sekumpulan objek yang digunakan untuk berkomunikasi antara sumber data dan dataset. DataAdapter juga mengelola objek koneksi, objek data reader, dan objek perintah. DataAdapter digunakan untuk pertukaran data antara sumber data dan dataset. Dalam banyak aplikasi, ini berarti membaca data dari database ke dataset, dan kemudian menulis data yang berubah dari dataset ke database.

Umumnya, adapter dapat dikonfigurasi yang memungkinkan untuk menentukan data apa yang dipindah dari dataset. DataAdapter yang tersedia untuk digunakan dengan jenis jenis database

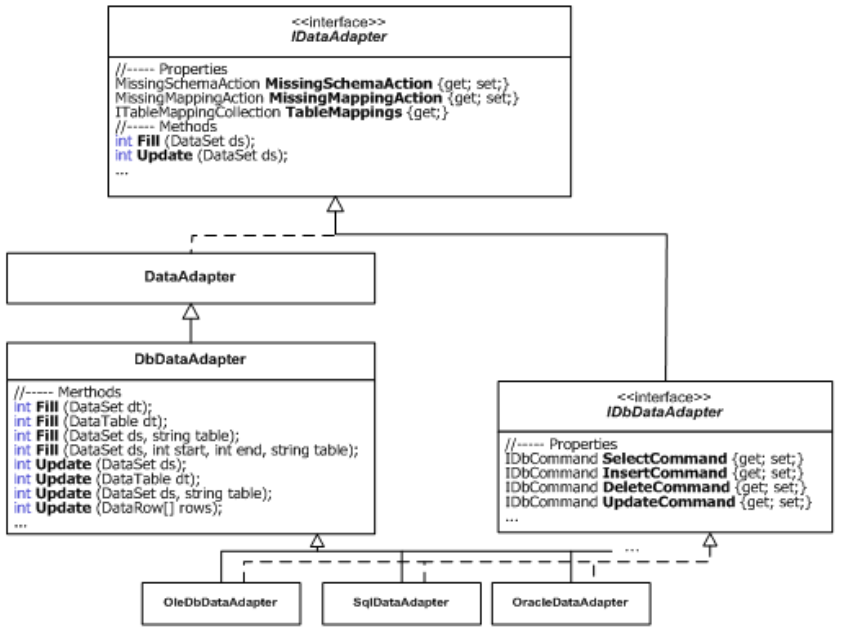
- Obyek OleDbDataAdapter cocok untuk digunakan dengan sumber data dari penyedia OLE DB.
- Obyek SqlDataAdapter, khusus untuk SQL Server. Karena akses data tidak harus melalui lapisan OLE DB, sehingga akses lebih cepat dari kelas OleDbDataAdapter Namun, hanya dapat digunakan dengan SQL Server 7.0 atau yang lebih baru.

- Obyek OdbcDataAdapter untuk mengakses sumber data ODBC.
- Obyek OracleDataAdapter untuk mengakses database Oracle.

Secara umum operasi pada DataAdapter terkait pada dua jenis operasi yakni koneksi ke sumber data untuk melakukan

- Fill: Mempopulasi data ke DataSet
- Update: Menulis perubahan ke sumber data

DataAdapter menggunakan obyek command diantaranya adalah SelectCommand, InsertCommand , DeleteCommand , UpdateCommand . Gambaran umum dari DataAdapter terlihat pada kelas diagram berikut ini:



Gambar 15.8 Kelas Diagram DataAdapter

Cara kerja dari dataAdapter terlebih dahulu sebuah dataAdapter melakukan loading data, secara program ditunjukkan dibawah ini:

1. Buat Obyek DataAdapater dan atur perintah

```
IDbDataAdapter adapter = new OleDbDataAdapter();
OleDbCommand cmd = new OleDbCommand();
```

```
cmd.Connection = new OleDbConnection
("provider=SQLOLEDB; ..." );
cmd.CommandText = "SELECT * FROM Person";
adapter.SelectCommand = cmd;
```

2. Baca data dan masukkan ke dataset

```
adapter.Fill(ds, "Person");
```

3. Terima atau batalkan perubahan

```
if (ds.HasErrors) ds.RejectChanges();
else ds.AcceptChanges();
if (adapter is IDisposable)
((IDisposable)adapter).Dispose();
```

Perubahan data dapat dilakukan dengan perintah update yang ada pada kelas DataAdapter. Terlebih dahulu harus membuat perintah update dengan *commandbuilder*. Contoh berikut ini menunjukkan langkah-langkah untuk melakukan update data

1. Buat obyek dataAdapter

```
OleDbConnection con = new OleDbConnection
             ("provider=SQLOLEDB; …");
adapter = new OleDbDataAdapter("SELECT * FROM Person",
con);
```

2. Buat perintah update lewat commandBuilder

```
OleDbCommandBuilder cmdBuilder = new
             OleDbCommandBuilder(adapter);
```

3. Panggil perintah update dan cek adanya kesalahan

```
try { adapter.Update(ds, tableName);
} catch (DBConcurrencyException) { }
adapter.Dispose();
```