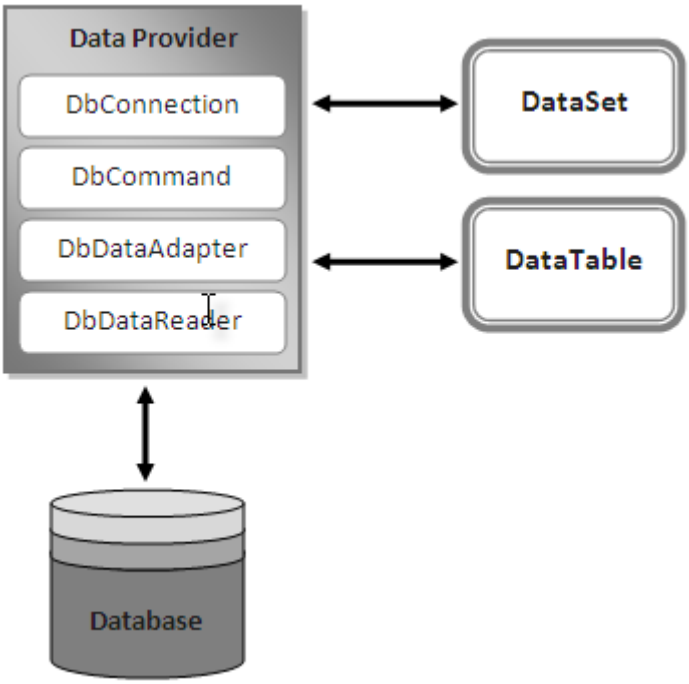
BAB. I
KONEKSI DAN APLIKASI TERKONEKSI

Pada modul ini, dibahas tentang koneksi kedalam database. Dari koneksi, dapat dilakukan pembacaan data menggunakan command dan datareader. Keduanya merupakan obyek dalam aplikasi terkoneksi

Materi:
SQLConnection
SQLCommand
SQLDataReader

Arsitektur ADO .NET dibagi menjadi 3 kelompok besar, yaitu Data Provider, DataSet dan DataTable (Thearon Willis, 2006, *Beginning Visual Basic 2005 Database*). Data Provider merupakan penyedia layanan agar DataSet dan DataTable dapat terhubung dengan Database. Sedangkan DataSet itu sendiri merupakan sebuah repository data pada lokal yang mampu merepresentasikan database di sisi server.



Gambar 1. Arsitektur ADO.NET

Seperti yang dapat dilihat pada Gambar 1, Data Provider itu sendiri dibagi menjadi 4 obyek yaitu Connection, Command, DataAdapter dan DataReader. Data Provider juga dibagi kedalam beberapa tipe berdasarkan jenis database seperti ODBC Data Provider, SQL Server Data Provider, Oracle Data Provider dan OLEDB Data Provider.

Untuk masing-masing jenis dari Data Provider, penamaan obyek-obyek dalam Data Provider mengikuti jenisnya. Contoh untuk

penggunaan database SQLServer, maka obyek yang dipakai adalah SqlConnection, SqlCommand, SqlDataAdapter dan SqlDataReader. Semua obyek ini ada didalam sebuah namespace yaitu System.Data.SqlClient.

1. SqlConnection

SqlConnection adalah fasilitas dalam komputer yang dapat digunakan agar aplikasi client dapat berkomunikasi dengan server, dalam sebuah pc atau tidak. Koneksi dibutuhkan untuk mengirimkan request (dalam bentuk command) dan menerima respon (dalam bentuk resultset).

1.1. Membuat Obyek Connection

Sebagai acuan teknis dalam pembuatan koneksi, dapat dilihat pada situs *www.connectionstrings.com*.

1. ConnectionString

Sebuah ConnectionString mendefinisikan berbagai parameter yang dibutuhkan oleh SqlConnection untuk membuat koneksi kedalam database SQLServer. Parameter-parameter ini meliputi:

- DATA SOURCE

  Nama lainnya adalah SERVER. Parameter ini berisi nama server atau alamat (IP address) server yang dituju diikuti nama instance dari SQLServer (jika ada).

- INITIAL CATALOG

  Nama lainnya adalah DATABASE. Parameter ini berisi nama database (dalam SQLServer) yang akan digunakan.

- INTEGRATED SECURITY

  Bernilai *TRUE* atau *FALSE*. Dengan mengaktifkan parameter ini (*TRUE*), maka SqlConnection menggunakan otentikasi Windows. Umumnya dipakai oleh aplikasi client-server pada satu pc.

- USER ID dan PASSWORD

  Penggunaan parameter ini merupakan kebalikan dari INTEGRATED SECURITY. Otentikasi yang digunakan adalah

otentikasi dari SQL Server. Umumnya dipakai oleh aplikasi client-server pada pc yang berbeda.

Contoh membuat ConnectionString:

Server : MY-PC\SQLEXPRESS
Database : VISUAL2
Otentikasi : WINDOWS

Maka ConnectionString dapat dilihat pada Kode 1.1:

Kode 1.1 ConnectionString

```
Dim strConn As String = _
"SERVER=MY-PC\SQLEXPRESS;DATABASE= " & _
"VISUAL2;INTEGRATED SECURITY=TRUE"
```

Penerapan ConnectionString pada SqlConnection (beserta penggunaan NameSpace) dapat dilihat pada Kode 1.2:

Kode 1.2 SqlConnection dengan ConnectionString

```
Imports System.Data.SqlClient

Dim conn As SqlConnection
Private Sub Form_Load(…) Handles MyBase.Load
   Dim strConn As String = _
   "SERVER=MY-PC\SQLEXPRESS;DATABASE= " & _
   "VISUAL2;INTEGRATED SECURITY=TRUE"

   conn = New SqlConnection(strConn)
End Sub
```

2. Metode Open() dan Close()

Method Open dalam obyek SQLConnection digunakan untuk membuka koneksi kedalam SQLServer. Sedangkan method Close digunakan untuk menutup koneksi. Pada dasarnya, kedua method ini dilakukan hanya ketika terjadi transaksi kedalam database. Dengan begitu, dapat memperkecil error yang terjadi pada aplikasi karena penggunaan obyek SQLConnection yang berulang-ulang. Perhatikan Kode 1.3:

Kode 1.3 Open() dan Close()

```
...
conn = New SqlConnection(strConn)
conn.Open()
'Transaksi
conn.Close()
...
```

3. Mengatasi kegagalan Koneksi

Koneksi tidak luput dari error sehingga koneksi gagal dilakukan, dan pada akhirnya aplikasi terpaksa harus *debug* dan *exit*. Tujuan mengatasi kegagalan koneksi sebenarnya hanya untuk meng-*handle* agar aplikasi tidak *exit*, selain itu juga untuk memberi tahu error apa yang terjadi dan apa penyebabnya.

Dibutuhkan *try…catch* untuk mengatasi kegagalah koneksi. Namun informasi error yang terjadi dan apa penyebabnya, dibuat dalam bahasa Inggris. Untuk membuat *custom information*, perlu diketahui terlebih dahulu macam-macam Exception yang mungkin terjadi ketika ada kegagalan koneksi.

Simply, kode program koneksi setelah ditambahkan *try…catch* dapat dilihat pada Kode 1.4:

Kode 1.4 Try Catch untuk SqlConnection

```
...
Try
   conn = New SqlConnection(strConn)
   conn.Open()
   'Transaksi
Catch ex As Exception
   MessageBox.Show(ex.Message)
Finally
   If conn.State = ConnectionState.Open Then
      conn.Close()
   End If
End Try
...
```

2. Aplikasi Terkoneksi

Aplikasi terkoneksi pada ADO .NET adalah sebuah skenario tentang cara aplikasi *running* yang selalu terhubung dengan database. Selalu terhubung maksudnya adalah aplikasi harus membuka dan menutup koneksi setiap ada kebutuhan untuk menampilkan dan/atau memanipulasi data dan/atau obyek database. Setiap satuan transaksi akan berpengaruh terhadap database, secara langsung. Memang, efek negatif akan terasa dalam jaringan. Namun, connected environment memberikan keunggulan pada real time application dan kemampuannya dalam mengolah berbagai jenis query, seperti SELECT, DML (Data Manipulation Language), dan DDL (Data Definition Language).

Lingkungan ini tidak membutuhkan banyak requirement, hanya SqlCommand sebagai pengirim dan SqlDataReader sebagai penerima. Model aplikasi ini merupakan model paling tua dan masih dipakai hingga sekarang karena *simplicity* dan kecepatannya dalam mengolah data.

3. SqlCommand

Seperti yang disebutkan sebelumnya, SqlCommand merupakan pengirim. Lebih tepatnya adalah pengirim query (bahasa *native* dalam database), baik query untuk menampilkan data (SELECT) maupun query untuk manipulasi data (INSERT/UPDATE/DELETE dan CREATE/ALTER/DROP).

Agar lebih memudahkan pemahaman tentang pemakaian SQLCommand, lihat Kode 1.5 berikut ini:

Kode 1.5 SqlCommand

```
...
Dim comm As SqlCommand
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   conn.Open()
   Dim strQuery As String = _
   "SELECT * FROM MAHASISWA"
   comm = New SqlCommand(strQuery, conn)
   conn.Close()
   ...
End Sub
```

Setelah command didefinisikan, untuk mengeksekusi query yang ada dalam SqlCommand, dibutuhkan salah satu dari empat metode antara lain ExecuteReader, ExecuteScalar, ExecuteNonQuery, dan ExecuteXmlReader.

3.1. ExecuteReader

Method ini menghasilkan sekumpulan data yang berbentuk DataReader yang mempunyai sifat *read-only* dan *forward-only*. Method ini hanya dipakai jika query yang dipakai adalah SELECT.

3.2. ExecuteScalar

Method ini menghasilkan sebuah nilai (dalam bentuk 1 baris dan 1 kolom – biasanya disebut dengan 1 cell). Method ini umumnya

dipakai ketika query yang dipakai adalah SELECT dengan GROUP FUNCTION didalamnya. Untuk pemakaiannya, perhatikan Kode 1.6 berikut ini:

Kode 1.6 ExecuteScalar

```
...
Dim strQuery As String = _
"SELECT COUNT(*) FROM Mahasiswa"

comm = New SqlCommand(strQuery, conn)

Dim jumlahMahasiswa As Object = _
comm.ExecuteScalar

Messagebox.Show(jumlahMahasiswa)
...
```

3.3. ExecuteNonQuery

Method ini menghasilkan sebuah nilai integer yang mengindikasikan berapa jumlah baris yang dipengaruhi oleh DML (INSERT/UPDATE/DELETE) atau DDL (CREATE/ALTER/DTOP).

3.4. ExecuteXmlReader

Method ini menghasilkan versi XML dari DataReader

4. SqlDataReader

DataReader merupakan sebuah obyek yang digunakan untuk membaca data dari database secara cepat. Seperti namanya, sifat dari DataReader adalah *read-only* dan *forward-only* sehingga data yang ada pada obyek DataReader tidak dapat dimanipulasi (hanya untuk keperluan pembacaan data saja), dan sekali dibaca maka tidak dapat kembali lagi untuk membaca data sebelumnya.

SQLDataReader berasosiasi langsung dengan SQLCommand ketika melakukan perintah ExecuteReader. Sehingga pembuatan SqlDataReader sedikit berbeda dengan obyek lainnya. Untuk lebih jelasnya, lihat Kode 1.7.

Kode 1.7 SqlDataReader

```
...
Dim reader As SqlDataReader
Private Sub Form_Load(…)Handles MyBase.Load
   ...
   comm = New SqlCommand(strQuery, conn)
   reader = comm.ExecuteReader
   ...
End Sub
```

Pembacaan data dalam SQLDataReader adalah per baris data dan menggunakan proses looping (perulangan) mengingat sifatnya yang berupa *forward-only*. Karena tidak diketahui sampai kapan looping ini selesai, maka perulangan yang dipakai adalah WHILE (Kode 1.8).

Kode 1.8 While untuk baca SqlDataReader

```
While reader.Read
   MessageBox.Show(reader(0))
End While
```

Didalam WHILE, dituliskan kode program untuk menampung, menampilkan secara langsung, dan bahkan untuk memproses data. Jika dilihat, dalam messagebox ada `reader(0)`. Kode ini digunakan untuk mengambil value dari kolom pertama dalam tabel. 0 merupakan *column ordinal* yang dimulai dari 0. Ada banyak cara untuk mengambil value, seperti pada Kode 1.9:

Kode 1.9 Cara lain membaca data dalam SqlDataReader

```
While reader.Read
   MessageBox.Show(reader("NIM"))
   MessageBox.Show(reader.GetString(0))
End While
```

Pada baris kedua Kode 1.8, pengambilan data menggunakan nama kolom. Sedangkan baris ketiga Kode 1.8 menggunakan metode spesifik untuk mengambil string.

BAB. II
DML PADA APLIKASI TERKONEKSI

Aplikasi client tidak hanya digunakan untuk menampilkan data, tetapi juga untuk mengolah (memanipulasi) data. Bentuk manipulasi ini antara lain INSERT, UPDATE, dan DELETE yang dalam bahasa query disebut sebagai DML. Pembahasan pada bab ini difokuskan pada bagaimana mengolah DML pada aplikasi client

Materi:
Insert
Update
Delete
Query Dinamis

Data Manipulation Language merupakan sebuah bahasa yang digunakan untuk melakukan INSERT, UPDATE dan DELETE data dalam tabel.

Sebagai contoh, perhatikan tabel Mahasiswa berikut ini:

Nama Kolom	Tipe Data
NIM	Char(11)
Nama	Varchar(50)
Tgl_Lahir	DateTime
Semester	Numeric(18,0)

Untuk masing-masing DML dapat dilihat pada sub bab berikut ini.

1. Insert

Secara sederhana dalam proses INSERT perlu diperhatikan jumlah data dengan jumlah kolom yang akan dientrikan (Kode 2.1). Oleh karena itulah, akan lebih baik jika menggunakan contoh INSERT yang kedua.

Kode 2.1 Query Insert

```
INSERT INTO Mahasiswa
VALUES ('05410104003','Adi','3-Apr-1986',6)
-- atau
INSERT INTO Mahasiswa
(NIM, Nama, Tgl_Lahir, Semester)
VALUES ('05410104003','Adi','3-Apr-1986',6)
```

Untuk implementasi INSERT pada ADO.NET, perhatikan Kode 2.2 berikut ini:

Kode 2.2 Implementasi Insert dalam SqlCommand

```
Dim comm As SqlCommand
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   conn.Open()
   Dim strQuery As String = _
   "INSERT INTO Mahasiswa " & _
   "(NIM, Nama, Tgl_lahir, Semester) " & _
   "VALUES " & _
   "('05410104003','Adi','3-Apr-1986',6)"
   comm = New SqlCommand(strQuery, conn)
   comm.ExecuteNonQuery()
   conn.Close()
   ...
End Sub
```

Dari contoh tersebut, dapat disimpulkan bahwa sebenarnya tidak ada perubahan pada pola kode jika dibandingkan dengan penggunaan SELECT pada SqlCommand. Hanya dengan mengubah value dari strQuery dari SELECT menjadi INSERT INTO, maka SqlCommand dapat melakukan INSERT kedalam table.

Setelah query ditampung, untuk menjalankan query tersebut diperlukan metode ExecuteNonQuery yang telah dijelaskan sebelumnya pada Bab I. Hal ini berlaku juga untuk UPDATE dan DELETE.

2. Update

Dalam proses UPDATE, hal yang perlu diperhatikan adalah bahwa tidak semua data diubah. Pembatasan untuk beberapa data dapat dilakukan dengan menggunakan kata kunci WHERE. Perhatikan Kode 2.3 berikut:

Kode 2.3 Query Update

```
UPDATE Mahasiswa
SET Nama = 'Budi',
    Semester = 7
WHERE NIM = '05410104003'
```

Untuk implementasi UPDATE pada ADO.NET, perhatikan Kode 2.4 berikut ini:

Kode 2.4 Implementasi Update dalam SqlCommand

```
Dim comm As SqlCommand
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   conn.Open()
   Dim strQuery As String = _
   "UPDATE Mahasiswa " & _
   "SET Nama = 'Budi', " & _
   "    Semester = 7 " & _
   "WHERE NIM = '05410104003'"
   comm = New SqlCommand(strQuery, conn)
   comm.ExecuteNonQuery()
   conn.Close()
   ...
End Sub
```

3. Delete

Sama seperti UPDATE, tidak semua data dihapus. Oleh karena itu, penggunaan WHERE juga diperlukan disini. Sehingga native query menjadi seperti pada Kode 2.5.

Kode 2.5 Query Delete

```
DELETE Mahasiswa
WHERE NIM = '05410104003'
```

Untuk implementasi DELETE pada ADO.NET, perhatikan Kode 2.6 berikut ini:

Kode 2.6 Implementasi Delete dalam SqlCommand

```
Dim comm As SqlCommand
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   conn.Open()
   Dim strQuery As String = _
   "DELETE Mahasiswa " & _
   "WHERE NIM = '05410104003'"
   comm = New SqlCommand(strQuery, conn)
   comm.ExecuteNonQuery()
   conn.Close()
   ...
End Sub
```

4. Query Dinamis

Dalam aplikasi client-server, semua data yang dientrikan kedalam database harus melalui sebuah antar muka. Entri data dapat dilakukan melalui textbox, combobox, radio button, dan bahkan listview, serta control-control input lainnya. Dari beberapa contoh diatas, value dari strQuery adalah tetap atau dengan kata lain query yang dipakai adalah query statis. Padahal sebenarnya, data yang dientrikan bisa berubah-ubah sesuai dengan entrian pada antar muka. Sehingga, query yang dikirimkan ke database server juga berbeda-beda sesuai dengan entrian pada antar muka tersebut atau dengan kata lain query yang digunakan adalah query dinamis.

Untuk membentuk query dinamis, hanya diperlukan kata kunci untuk menggabungkan dua buah string yaitu dengan menggunakan *ampersand* (&). Perhatikan Kode 2.7 berikut ini:

Kode 2.7 Query Statis

```
Dim strQuery As String = _
"INSERT INTO Mahasiswa " & _
"(NIM, Nama, Tgl_lahir, Semester) " & _
"VALUES " & _
"('05410104003','Adi','3-Apr-1986',6)"
```

Dari query statis tersebut, query dinamis menjadi seperti pada Kode 2.8:

Kode 2.8 Query dinamis

```
Dim strQuery As String = _
"INSERT INTO Mahasiswa " & _
"(NIM, Nama, Tgl_lahir, Semester) " & _
"VALUES " & _
"('" & txtNim.Text & "', " & _
"'" & txtNama.Text & "', " & _
"'" & dtLahir.Value & "', " & _
"" & nupSemester.Value & ")"
```

4.1. Command Parameters

Penggunaan *ampersand* dapat mengakibatkan kebingungan ketika diperlukan pengubahan query ketika jumlah kolom mencapai puluhan. Untuk mengatasi hal ini, ada cara yang jauh lebih baik yaitu menggunakan *command parameters*, yaitu sebuah *placeholder* dalam SqlCommand untuk menyimpan nilai-nilai yang akan diproses oleh query. Dimulai dengan karakter @ dan diikuti oleh nama parameter tanpa ada spasi (Kode 2.9).

Kode 2.9 Command Parameters

```
Dim strQuery As String = _
"INSERT INTO Mahasiswa " & _
"(NIM, Nama, Tgl_lahir, Semester) " & _
"VALUES " & _
"(@vNIM, @vNama, @vTgl, @vSmt)
comm = New SqlCommand(strQuery, conn)

'Parameter @vNIM
comm.Parameters.Add("@vNIM", _
SqlDbType.VarChar, 11)
comm.Parameters("@vNIM").Value = _
txtNIM.Text

'Parameter @vNama
comm.Parameters.Add("@vNama", _
SqlDbType.VarChar, 11)
comm.Parameters("@vNama").Value = _
txtNama.Text

'Parameter @vTgl
comm.Parameters.Add("@vTgl", _
SqlDbType.DateTime, 11)
comm.Parameters("@vTgl").Value = _
dtLahir.Value

'Parameter @vSmt
comm.Parameters.Add("@vSmt", _
SqlDbType.Int, 11)
comm.Parameters("@vSmt").Value = _
nupSemester.Value

'ExecuteNonQuery
comm.ExecuteNonQuery()
```

BAB. III
STORED PROCEDURE
DALAM APLIKASI TERKONEKSI

Salah satu bentuk kekuatan dari aplikasi database adalah adanya proses bisnis disisi server, yang di-handle langsung oleh database melalui bahasa pemrograman yang disebut sebagai T-SQL (Transact SQL). T-SQL terdiri dari Stored Procedure, User-Defined Function, dan Trigger. Namun, yang hanya bisa diakses oleh client adalah Stored Procedure. Bab ini difokuskan pada pembuatan dan pemanggilan Stored Procedure dalam ADO.NET

Materi:
Stored Procedure
SqlCommand untuk Stored Procedure

1. Stored Procedure

Stored procedure merupakan program yang berjalan disisi server yang mampu mengelompokkan query-query dalam sebuah package agar dapat digunakan kembali.

1.1. Membuat Stored Procedure

Stored procedure mempunyai parameter yang bisa digunakan untuk input atau output. Mereka juga mempunyai return value bernilai integer (dengan nilai default adalah nol), dan mereka bisa mengembalikan nol atau lebih resultset. Mereka bisa dipanggil dari program client atau stored procedure lainnya. Mereka powerful dan menjadi model yang disukai bagi kebanyakan programmer database, khususnya untuk aplikasi multi-tier dan web services, karena (dari salah satu kelebihan) mereka dapat mengurangi lalu lintas jaringan.

Sebagai contoh sederhana dalam stored procedure, perhatikan Kode 3.1 berikut ini:

Kode 3.1 Stored Procedure sederhana

```
CREATE PROCEDURE SP_AmbilMahasiswa
AS
SELECT NIM, Nama
FROM Mahasiswa;
```

Dari kode 3.1, perintah `CREATE PROCEDURE` membuat stored procedures. Kata kunci `AS` memisahkan nama procedure dan parameter (yang pada contoh tersebut tidak ada) dengan body dari stored procedure. Setelah `AS`, pada body stored procedure hanya terdapat sebuah komponen yaitu sebuah query sederhana `SELECT NIM, Nama FROM Mahasiswa`.

Dalam lingkungan SQL Server Management Studio (SSMS), stored procedure disimpan dalam PROGRAMMABILITY untuk kemudian dapat dipanggil melalui Query Editor ataupun langsung dari Object Explorer.

1.2. Stored Procedure dengan Input Parameter

Kode 3.1 merupakan sebuah contoh stored procedure sederhana yang mengembalikan seluruh data mahasiswa. Lebih dari itu, stored

procedure mempunyai kekuatan tersendiri melalui parameter-nya. Berikut sedikit contoh pada Kode 3.2 tentang bagaimana parameter dalam stored procedure.

Kode 3.2 Stored Procedure dengan Input Parameter

```
CREATE PROCEDURE SP_Nilai_per_Mahasiswa
@pNIM varchar(11)
AS
SELECT NIM, Kode_Kuliah, Tugas, UTS, UAS
FROM Mahasiswa
WHERE NIM = @pNIM;
```

Terlihat pada Kode 3.2 ada sebuah parameter dengan nama `@pNIM`. Ya, parameter adalah variabel, yang dalam SQL Server dimulai dengan @ diikuti dengan nama variabel dan tipe datanya. Seperti yang telah disebutkan sebelumnya bahwa parameter dapat berupa input atau output. Dalam Kode 3.2, parameter `@pNIM` (secara default) bertipe input. Penggunaan parameter @pNIM ada pada kondisi `WHERE NIM = @pNIM`. Penulisan ini sama seperti penulisan kode program pada umumnya.

1.3. Stored Procedure dengan Output Parameter

Ouput parameter, seperti namanya, digunakan untuk melempar nilai antar stored procedure dan bahkan dari stored procedure ke client. Perhatikan Kode 3.3 berikut ini:

Kode 3.3 Stored Procedure dengan Output Parameter

```
CREATE PROCEDURE SP_Nilai_per_Mahasiswa
@pNIM varchar(11),
@pCountNilai numeric = 0 output
AS
-- ambil data
SELECT NIM, Kode_Kuliah, Tugas, UTS, UAS
FROM Mahasiswa
WHERE NIM = @pNIM;
-- menghitung jumlah data
SELECT @pCountNilai = COUNT(*)
FROM Mahasiswa
WHERE NIM = @pNIM;
-- mengembalikan nilai jumlah data
RETURN @pCountNilai;
```

Pada Kode 3.3, parameter output didefinisikan dengan adanya kata kunci `output` pada parameter `@pCountNilai`. Selain itu, parameter ini juga memilii default value yaitu nol. Ada penambahan

query untuk menghitung jumlah data dan menampungnya pada parameter output. Lalu mengembalikan nilai yang sama melalui `RETURN @pCountNilai`.

1.4. Mengubah dan Menghapus Stored Procedure

Untuk mengubah stored procedure, gunakan kata kunci ALTER sebagai pengganti CREATE. Perhatikan Kode 3.4 berikut ini:

Kode 3.4 Alter Stored Procedure

```
ALTER PROCEDURE SP_Nilai_per_Mahasiswa
@pNIM varchar(11),
@pCountNilai numeric = 0 output
AS
-- ambil data
SELECT NIM, Kode_Kuliah, Tugas, UTS, UAS
FROM Mahasiswa
WHERE NIM = @pNIM
ORDER BY Kode_Kuliah --penambahan kode
-- menghitung jumlah data
SELECT @pCountNilai = COUNT(*)
FROM Mahasiswa
WHERE NIM = @pNIM
-- mengembalikan nilai jumlah data
RETURN @pCountNilai
```

Pengubahan tidak hanya pada kata kunci ALTER, tetapi juga pada body dari stored procedure seperti pada `ORDER BY Kode_Kuliah`. Sedangkan untuk menghapus stored procedure, hanya mengganti kata kunci CREATE dengan DROP diikuti dengan nama stored procedure. Perhatikan Kode 3.5 berikut ini:

Kode 3.5 Drop Stored Procedure

```
DROP PROCEDURE SP_Nilai_per_Mahasiswa
```

2. Memanggil Stored Procedure dari Client

Stored procedure bisa dipanggil melalui SqlCommand dengan mengubah tipenya. Untuk lebih jelasnya, perhatikan Kode 3.6 berikut ini:

Kode 3.6 SqlCommand untuk Stored Procedure

```
...
Dim comm As SqlCommand
Dim reader As SqlDataReader
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   conn.Open()
   comm = New SqlCommand()
   comm.CommandType = _
   CommandType.StoredProcedure
   comm.CommandText = "SP_AmbilMahasiswa"

   reader = comm.ExecuteReader
   ...
   conn.Close()
   ...
End Sub
```

Pada Kode 3.6, tipe SqlCommand diubah menjadi CommandType.StoredProcedure. Mengikuti tipe, CommandText diisi dengan nama stored procedure. Karena stored procedure SP_AmbilMahasiswa digunakan untuk menampilkan seluruh data mahasiswa, maka untuk menampung hasilnya, digunakan SqlDataReader. Setelah itu, perlakukan sama seperti yang telah dijelaskan pada bab 1.

Bagaimana jika stored procedure mempunyai parameter? Perhatikan kode program 3.7. Ada tiga parameter yang dispesifikasikan pada client, dengan perbedaan Direction untuk masing-masing tipe parameter. Pemberian nilai melalui properti Value dilakukan pada parameter dengan tipe input. Sedangkan nilai dari output dan return value ditampilkan melalui messagebox setelah SqlCommand dijalankan melalui metode ExecuteReader.

Kode 3.7 SqlCommand untuk SP berparameter

```
...
comm = New SqlCommand
comm.CommandType = CommandType.StoredProcedure
comm.CommandText = "SP_Nilai_per_Mahasiswa"

'Atur parameter input
Dim pNIM As SqlParameter = _
comm.Parameters.Add("@pNIM", _
SqlDbType.VarChar, 11)
pNIM.Direction = ParameterDirection.Input
pNIM.Value = "01410100077"

'Atur parameter output
Dim pCountNilai As SqlParameter = _
comm.Parameters.Add("@pCountNilai", _
SqlDbType.Int)
pCountNilai.Direction =
ParameterDirection.Output

'Atur return value
Dim retVal As SqlParameter = _
comm.Parameters.Add("return_value", _
SqlDbType.Int)
retVal.Direction = _
ParameterDirection.ReturnValue

reader = comm.ExecuteReader

MessageBox.Show("output: " & _
pCountNilai.Value)
MessageBox.Show("return: " & retVal.Value)
```

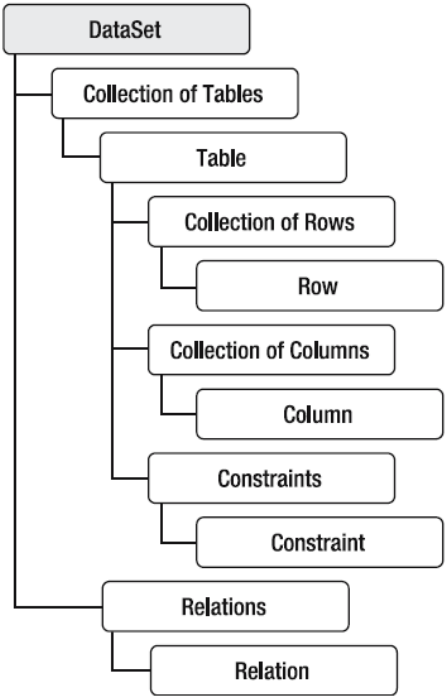
BAB. IV
DATASET

Untuk aplikasi yang dapat mengolah data secara lokal (dalam jumlah besar), dengan kata lain data tidak dibutuhkan secara real-time, penggunaan obyek dalam lingkungan terkoneksi menjadi kurang tepat. Oleh karena itulah dibutuhkan lingkungan tidak terkoneksi. Salah satu bagian terpenting dalam lingkungan tidak terkoneksi adalah DataSet. Sederhananya, DataSet merupakan database lokal. Fokus dalam bab ini adalah memperkenalkan dan bagaimana membangun DataSet.

Materi:
DataSet
DataTable
DataColumn

Pada 2 materi sebelumnya, kita semua belajar tentang koneksi database menggunakan *connected environment* yang bersifat *read-only* dan *forward-only.* Pada modul ini, kita akan belajar tentang cara lain untuk berkomunikasi dengan database, yaitu dengan menggunakan obyek DataSet. DataSet merupakan obyek dari class System.Data.DataSet yang bisa dipakai oleh semua data provider, independen, dan bisa digunakan baik secara terkoneksi atau tidak terkoneksi dengan *data source*.

DataSet mampu menggambarkan sekumpulan data yang tersimpan dalam berbagai DataTable, constraint serta hubungan antar tabel. Selain itu, dia juga mampu menyimpan begitu banyak data dan memprosesnya di secara offline, dengan kata lain, *disconnected* dari database. Tujuan utama adanya DataSet adalah sebagai penyedia layanan database secara virtual yang disimpan dalam *memory cache.*



Gambar 2. Struktur DataSet

Seperti yang bisa kita liat di gambar, DataSet terdiri dari sekumpulan tabel yang biasa disebut sebagai DataTable. DataTable ini dibentuk dari sekumpulan kolom yang biasa disebut sebagai DataColumn dan sekumpulan baris yang biasa disebut sebagai DataRow. Selain terdiri dari sekumpulan DataTable, DataSet juga mampu mengatur Constraint dan hubungan antar table yang biasa disebut sebagai DataRelation.

1. DataSet

Ada beberapa cara untuk membuat DataSet:

1. Membuat dari RDBMS dengan menggunakan SqlDataAdapter (Bab 6).
2. Membaca dari file XSD dan XML (Bab 7).
3. Membangunnya dari DataTables, DataRelations dan Constraints lalu mempopulasikan datanya.
4. Membuat dari DataReader.

Yang dipelajari untuk bab ini adalah membuat DataSet dari DataTables, DataRelations dan Constraints. Cara ini dilakukan dengan menggunakan kode program. Perhatikan Kode 4.1 berikut ini:

Kode 4.1 Pembuatan DataSet

```
Dim dset As DataSet
Private Sub Form_Load(…) Handles MyBase.Load
   dset = New DataSet("Akademik")
End Sub
```

Semudah itu membuat DataSet. Untuk selanjutnya, membuat DataTable

2. DataTable

Seperti yang sudah diketahui bahwa DataSet dibangun dari DataTables, DataRelations dan Constraints. Pada ADO.NET, DataTables merepresentasikan tabel-tabel pada DataSet. Semua data yang ada pada DataTable hanya bisa digunakan ketika program *running*, namun bisa juga dikaitkan langsung dengan *data source* misal SQL Server dengan menggunakan SQLDataAdapter. Untuk membuat DataTable, perhatikan Kode 4.2 berikut ini:

Kode 4.2 Pembuatan DataTable

```
...
Dim dtMhs As DataTable
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   dtMhs = New DataTable("Mahasiswa")
End Sub
```

Untuk menambahkan DataTable kedalam DataSet, perhatikan Kode 4.3 berikut ini:

Kode 4.3 Menambahkan DataTable kedalam DataSet

```
dset.Tables.Add(dtMhs)
```

3. DataColumn

Dalam DBMS, sebuah tabel dibangun dari sekumpulan kolom. DataTable menggunakan konsep yang sama, namun menggunakan DataColumn. Perhatikan Kode 4.4 berikut ini:

Kode 4.4 Pembuatan DataColumn

```
...
Dim dcNIM As DataColumn
...
Private Sub Form_Load(…) Handles MyBase.Load
   ...
   dcNIM = New DataColumn("NIM")
End Sub
```

Setiap kolom mempunyai *column constraint* atau batasan seperti tipe data, panjang kolom, dan lain-lain. Constraint dibahas pada sub bab selanjutnya. Sedangkan untuk menambahkan DataColumn kedalam DataTable, perhatikan Kode 4.5 berikut ini:

Kode 4.5 Menambahkan DataColumn kedalam DataTable

```
dtMhs.Columns.Add(dcNIM)
```

4. Constraint

Ada 2 macam berdasarkan levelnya, yaitu *column constraints* dan *table constraints*.

4.1. Column Constraint

Seperti yang telah disebutkan sebelumnya, *column constraint* dapat berupa tipe data, panjang kolom, *uniqueness*, *read-only*, *allow null*

values, dan lain-lain. Untuk penggunaan *column constraint*, perhatikan Kode 4.6 program berikut ini:

Kode 4.6 Macam Column Constraint

```
'kolom boleh null
dcNIM.AllowDBNull = False

'mengatur panjang kolom
dcNIM.MaxLength = 11

'nilai kolom tidak dapat diubah
dcNIM.ReadOnly = True

'nilai kolom harus unique
dcNIM.Unique = True
```

4.2. Table Constraint

Ada beberapa table constraint, antara lain:

- Primary Key

Kode 4.7 Pembuatan Primary Key

```
Dim pk() As DataColumn = _
New DataColumn() {dcNIM}

dtMhs.PrimaryKey = pk
```

Bisa menggunakan Kode 4.7 atau dapat menggunakan Kode 4.8 berikut ini:

Kode 4.8 Penambahan Constraint sebagai Primary Key

```
dtMhs.Constraints.Add _
("Mhs_PK", dcNIM, True)
```

Pemberian PK dapat dilakukan juga melalui *column constraints* yaitu melalui property Unique dengan value TRUE.

- Foreign Key

Kode 4.9 Pembuatan Foreign Key

```
dtMhs.Constraints.Add _
("Mhs_FK_Kota", dcKotaKota, dcMhsKota)
```

5. Relation

Terkadang, tabel-tabel dalam database saling berhubungan untuk memastikan apakah data-data yang dientrikan pada tabel-tabel

transaksi sesuai dengan data-data pada tabel master. Pada dasarnya, relation adalah membuat foreign key constraints (seperti yang telah disebutkan pada Table Constraints). Jadi dalam membuat relasi antar tabel, bisa menggunakan foreign key constraint (Kode 4.9), dapat juga menggunakan relation (Kode 4.10).

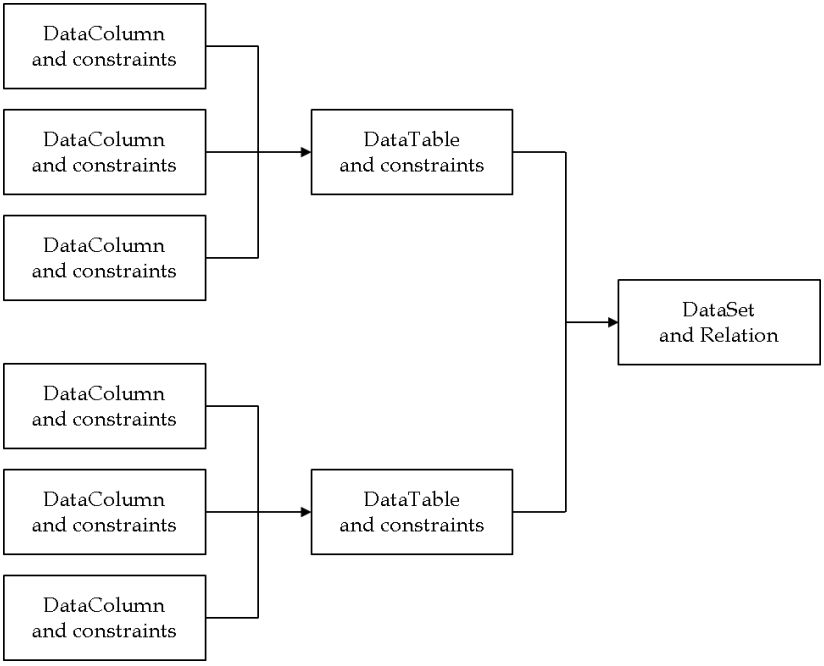
Kode 4.10 Pembuatan Relation

```
ds.Relations.Add _
("Mhs_Kota", dcKotaKota, dcMhsKota)
```

6. Menggabungkan itu semua

Dalam konsep database, seperti yang telah digambarkan di awal bab ini, database terdiri dari kumpulan tabel. Sedangkan tabel dibangun dari kolom. Dalam dataset, hal yang sama pun terjadi.

Namun dalam pengimplementasiannya, proses itu dibalik. Untuk lebih jelasnya, perhatikan gambar berikut ini:



Gambar 3. Pengimplementasian Pembuatan DataSet

Jadi, setelah semua DataColumn beserta constraintnya selesai didefinisikan, semua DataColumn tersebut di-add kedalam DataTable. Setelah itu, dilakukan pengaturan constraint pada seluruh DataTable. Setelah itu, semua DataTable tersebut di-add kedalam DataSet. Setelah itu, dilakukan pengaturan terhadap relation.

BAB. V
MANIPULASI, PENGURUTAN DAN PENCARIAN DATA DALAM DATASET

Fungsi utama dari database adalah sebagai tempat penyimpanan data. Data yang telah disimpan pun harus dapat digunakan kembali setelah dilakukan pencarian. Fokus pada bab ini adalah bagaimana memanipulasi data dan mencari data dalam DataSet. Selain itu, bab ini juga memberikan langkah-langkah dalam menampilkan kembali data dari DataSet kedalam DataGridView.

Materi:
DataRow
Pencarian Data
DataView dan DataGridView

1. Manipulasi Data

Data dalam database disimpan dalam bentuk baris (row). Hal ini berlaku juga pada DataSet, bahwa data disimpan dalam bentuk DataRow. Pada gambar di awal bab 4, disebutkan bahwa DataTable dibentuk dari DataColumn dan DataRow. DataColumn sebagai penyedia struktur DataTable, sedangkan DataRow sebagai penyedia layanan data pada DataTable.

1.1. DataRow

Penggunaan DataRow cukup sederhana. Sebagai contoh struktur DataTable sama seperti struktur tabel di bab 2, maka proses Insert, Update dan Delete menjadi seperti berikut ini:

- Insert

Kode 5.1 Insert menggunakan DataRow

```
Dim row As DataRow = _
dset.Tables("Mahasiswa").NewRow
row("NIM") = "01410100103"
row("Nama") = "Tito Rachmanto"
row("Tgl_Lahir") = "11-July-1982"
row("Semester") = 6
dset.Tables("Mahasiswa").Rows.Add(row)
```

Dari Kode 5.1 dapat dilihat bahwa DataRow dibuat langsung dari struktur kolom DataTable dengan memanggil metode `NewRow`. Pengisian row (baris ke-2 s.d ke-5) selain dapat menggunakan nama kolom, dapat juga menggunakan *column ordinal*. Setelah semua kolom diisi, dilakukan proses penambahan baris melalui baris ke-6.

- Update

Kode 5.2 Update menggunakan DataRow

```
Dim row As DataRow = _
dset.Tables("Mahasiswa").Rows(0)
row.BeginEdit()
row("NIM") = "01410100103"
row("Nama") = "Tito Rachmanto"
row("Tgl_Lahir") = "11-July-1982"
row("Semester") = 6
row.EndEdit()
```

Seperti yang telah dijelaskan sebelumnya pada bab 2, bahwa dalam update dan delete, hanya dilakukan untuk beberapa data saja. Sehingga, dalam query native diperlukan kata kunci WHERE. Dalam DataSet, mulanya dilakukan pencarian terlebih dahulu sebelum dilakukan BeginEdit(). Proses pencarian ini akan diberikan pada sub bab berikutnya.

Pada Kode 5.2, diasumsikan bahwa data yang dihapus adalah data pada baris kesatu. Secara berturut-turut BeginEdit() digunakan untuk memulai editing dan EndEdit() digunakan untuk melakukan *commit* terhadap perubahan.

- Delete

Kode 5.3 Delete menggunaan DataRow

```
Dim row As DataRow = _
dset.Tables("Mahasiswa").Rows(0)
row.Delete()
```

Sama seperti update, pada Kode 5.3 disebutan bahwa proses diawali dengan melakukan pencarian terlebih dahulu sebelum dilakukan Delete().

2. Pencarian data

Beberapa metode dalam pencarian data membutuhkan primary key (yang telah dibahas pada bab 4) dan beberapa lainnya tidak. Ada beberapa cara yang dapat dipakai untuk melakukan pencarian data.

2.1. Find

Metode ini membutuhkan primary key dan mengembalikan hasil pencarian bertipe DataRow. Jumlah yang dikembalikan adalah satu. Umumnya metode ini dipakai bersamaan dengan manipulasi data dalam DataSet. Perhatikan Kode 5.4 berikut ini:

Kode 5.4 Find

```
Dim row As DataRow = _
dset.Tables("Mahasiswa").Rows. _
Find("01410100077")

If Not IsNothing(row) Then
   'Data ada
Else
   'Data tidak ada
End If
```

Bagaimana jika primary key lebih dari satu? Parameter dalam metode Find perlu diubah menjadi seperti pada Kode 5.5:

Kode 5.5 Find dengan Multiple Primary Key

```
Dim row As DataRow = _
dset.Tables("Mahasiswa").Rows. _
Find(New Object() _
{"01410100077", "Tegar"})
```

2.2. Contains

Metode ini juga membutuhkan primary key, namun hasil yang dikembalikan adalah nilai Boolean. Perhatikan Kode 5.6 berikut ini:

Kode 5.6 Contains

```
Dim boolAda As Boolean = _
dset.Tables("Mahasiswa").Rows. _
Contains("01410100077")

If boolAda Then
   'Data ada
Else
   'Data tidak ada
End If
```

Untuk primary key lebih dari satu, parameter dalam metode Contains menjadi seperti pada Kode 5.7:

Kode 5.7 Contains dengan Multiple Primary Key

```
Dim boolAda As Boolean = _
dset.Tables("Mahasiswa").Rows. _
Contains(New Object() _
{"01410100077", "Tegar"})
```

2.3. Select

Untuk metode Find dan Contains, hanya mengembalikan sebuah value. Bagaimana jika proses pencarian mengembalikan lebih dari satu data? Solusinya adalah dengan menggunakan metode Select. Metode ini tidak membutuhkan primary key, jadi dia dapat melakukan pencarian di kolom apapun dan mengembalikan hasil berupa array of DataRow. Perhatikan Kode 5.8 berikut ini:

Kode 5.8 Select

```
Dim row() As DataRow = _
dset.Tables("Mahasiswa"). _
Select("NIM like '%41010%'")

If row.Length > 0 Then
   'Data ada
Else
   'Data tidak ada
End If
```

Seperti yang dapat dilihat pada contoh tersebut, parameter dalam metode Select dapat berupa string apapun, bahkan dapat digunakan *wildcards* (sama seperti native query).

3. DataView

Pada dasarnya, DataView sama seperti View dalam object Database (selain Table, Index, User, dll). Keuntungan dari penggunaan DataView adalah pencarian dan pengurutan yang langsung ter-*binding* dengan *data bound control* (contoh: DataGridView). Selain itu, pencarian data dapat dilakukan tanpa bantuan PK.

Pembuatan DataView menggunakan Kode 5.9 berikut ini:

Kode 5.9 Pembuatan DataView

```
Dim dview As DataView = _
dset.Tables("Mahasiswa").DefaultView
```

Pencarian data pada DataView bisa menggunakan Find dan FindRows. Sedangkan untuk pengurutan data menggunakan Sort. Untuk filtering menggunakan RowFilter.

3.1. Sort

Pengurutan digunakan sebagai pre-requisite untuk melakukan pencarian data melalui DataView. Pencarian ini menggunakan metode Find dan FindRows, namun tidak dibahas dalam buku ini. Untuk mengurutkan, bisa menggunakan cara yang sama seperti pada native query. Perhatikan contoh berikut ini:

Kode 5.10 Sorting

```
dview.Sort = "NIM ASC"
```

3.2. Filtering

Filtering dalam dataview, seperti yang telah disebutkan sebelumnya, digunakan untuk memfilter data yang tampil dalam data bound control sesuai dengan filternya. Penulisan kondisi sama seperti parameter dalam metode Select. Perhatikan Kode 5.11 berikut ini:

Kode 5.11 Filtering

```
dview.RowFilter = "NIM like '%0077%'"
```

Dengan contoh tersebut, data yang tampil dalam, katakanlah DataGridView, hanyalah data dengan NIM yang nilainya memiliki 0077, baik diawal, tengah maupun diakhir nilai.

4. DataGridView

DataGridView digunakan untuk menampilkan data-data dalam bentuk tabular yang diambil dari DataTable. Sama seperti listview, tetapi datagridview mempunyai kemampuan untuk editing data ketika run-time, langsung didalam control-nya.

Untuk dapat menampilkan data, perhatikan Kode 5.12 berikut ini:

Kode 5.12 Set DataSource dari DataGridView

```
DataGridView1.DataSource = _
dset.Tables("Mahasiswa")
```

Manipulasi data dapat dilakukan langsung dalam DataGridView dan secara otomatis mengubah yang ada dalam data source. Namun bagaimana caranya mengambil data yang ada dalam DataGridView untuk kemudian, misalnya, dilakukan perhitungan diluar DataGridView? Untuk hal ini, perhatikan Kode 5.13 berikut ini:

Kode 5.13 Mengambil Data dalam DataGridView

```
Dim rowView As DataGridViewRow = _
DataGridView1.Rows _
(DataGridView1.CurrentRow.Index)

MessageBox.Show(rowView.Cells("NIM").Value)
```

BAB. VI
DATASET DAN
APLIKASI TIDAK TERKONEKSI

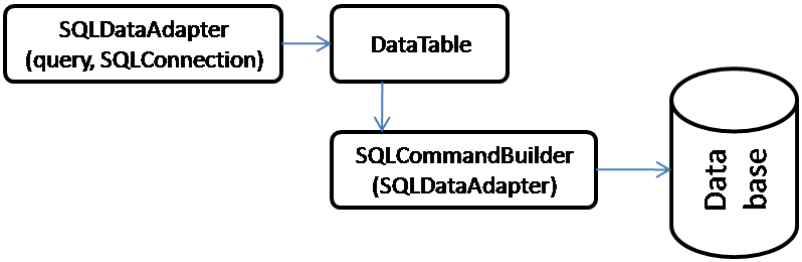
Salah satu cara untuk membuat DataSet adalah dengan mempopulasikan data dari database. Untuk hal ini, diperlukan SqlDataAdapter.

Materi
Aplikasi tidak terkoneksi
SqlDataAdapter

1. Aplikasi tidak terkoneksi

Merupakan sebuah desain aplikasi yang melakukan pengambilan seluruh data dari database dan disimpan pada database lokal. Dengan demikian, transaksi yang terjadi dapat dilakukan secara lokal dan ditransfer ke database dalam bentuk *batch*. Umumnya aplikasi semacam ini digunakan untuk aplikasi-aplikasi mobile.

Karena ada penggunaan database lokal, aplikasi tidak terkoneksi membutuhkan penyimpanan data lokal berupa DataSet. Seperti yang disebutkan pada bab 4, salah satu cara untuk membuat DataSet adalah dengan mempopulasikan data melalui SqlDataAdapter.



Gambar 4. SqlDataAdapter

Pada Gambar 4 dapat dijelaskan bahwa melalui SqlDataAdapter, data dapat dipopulasi untuk kemudian ditampung kedalam DataTable atau DataSet. Dengan menggunakan SqlCommandBuilder, segala bentuk perubahan pada DataTable disinkronkan dengan data yang ada dalam database.

2. SqlDataAdapter

Seperti yang terlihat pada Kode 6.1 bahwa aplikasi tidak terkoneksi masih membutuhkan obyek koneksi (dalam hal ini adalah SqlConnection), sama seperti aplikasi terkoneksi. Perbedaannya adalah untuk aplikasi terkoneksi, setiap terjadi transaksi selalu melakukan conn.Open() dan conn.Close(). Sedangkan untuk aplikasi tidak terkoneksi, setiap terjadi transaksi tidak perlu melakukan conn.Open() dan conn.Close().

Untuk lebih jelasnya, perhatikan Kode 6.1 berikut ini:

Kode 6.1 Pembuatan SqlDataAdapter

```
Imports System.Data.SqlClient
...
Dim conn As SqlConnection
Dim da As SqlDataAdapter
Dim ds As New DataSet

Private Sub Form1_Load(…) Handles MyBase.Load
   Dim strConn As String = _
   "SERVER=MY-PC\SQLEXPRESS;DATABASE= " & _
   "VISUAL2;INTEGRATED SECURITY=TRUE"

   Dim strQuery As String = _
   "SELECT * FROM Mahasiswa"

   da = New SqlDataAdapter(strQuery, conn)
   da.Fill(ds, "Mahasiswa")
End Sub
```

Seperti yang telah disebutkan sebelumnya bahwa kekuatan aplikasi database terletak pada query, maka dalam aplikasi tidak terkoneksi masih dibutuhkan query sebagai bahasa native dalam database. Query ini nantinya menjadi salah satu parameter dalam pembangunan SqlDataAdapter.

Dari contoh, pada pembangunan SqlDataAdapter, dibutuhkan dua parameter yaitu query dan koneksi, sama seperti SqlCommand. Perbedaannya adalah SqlCommand membutuhkan ExecuteReader (dan sejenisnya) sedangkan SqlDataAdapter membutuhkan metode Fill yang sekaligus berfungsi untuk menampung hasil query kedalam DataSet.

3. Constraint

Seperti yang disebutkan pada bab 4, bahwa DataSet juga memiliki constraint yang dibagi menjadi 2 yaitu column constraint dan table constraint. Kedua constraint ini beberapa secara otomatis terambil dari database (seperti struktur kolom, tipe data) melalui metode Fill. Namun beberapa yang lain seperti primary key, harus diatur dalam property MissingSchemaAction (Kode 6.2).

Kode 6.2 MissingSchemaAction

```
da.MissingSchemaAction = _
MissingSchemaAction.AddWithKey
```

Pengaturan ini harus dilakukan sebelum proses Fill dijalankan.

4. Manipulasi data

Pemanipulasian data mengikuti aturan main dalam DataSet, mulai dari Insert, Update, Delete, Sorting, Filtering, Searching, dan lain sebagainya. Namun yang perlu diingat bahwa semua proses tersebut masih secara lokal. Artinya, data dalam database tidak sama dengan kondisi data di lokal.

Agar data dalam database sama dengan data di lokal, diperlukan berbagai native query untuk DML. Hal ini dapat dengan mudah dicapai dengan bantuan SqlCommandBuilder (Kode 6.3).

Kode 6.3 SqlCommandBuilder

```
Dim cb As SqlCommandBuilder
...
cb = New SqlCommandBuilder(da)
```

SqlCommandBuilder secara otomatis membuat native query untuk DML berdasarkan query sederhana yang ada pada SqlDataAdapter. Setelah native query untuk DML dibuat, langkah selanjutnya adalah melakukan proses update (Kode 6.4).

Kode 6.4 Update dari SqlDataAdapter

```
da.Update(ds)
```

Proses update ini membutuhkan DataSet tempat data dari SqlDataAdapter yang dipakai oleh SqlCommandBuilder. Dalam *background process*, urutan proses dalam metode Update adalah (1) Delete, (2) Update, dan (3) Insert.

Bab. VII
XML DATABASE

Awalmula penyimpanan data dalam komputer adalah dalam bentuk berkas (file). Ketidak-terstrukturan data dalam file menjadi motivasi pengembangan RDBMS. Kini, dengan adanya file bertipe XML, programmer dimudahkan untuk melakukan penyimpanan data dalam file.

Materi
Aplikasi tidak terkoneksi
SqlDataAdapter

XML (eXtensible Markup Language) merupakan bahasa markup untuk mendeskripsikan berbagai macam data/informasi dengan lebih akurat dan fleksibel. Berbeda dengan HTML, XML tidak mempunyai format baku karena merupakan *meta-language* sehingga bisa dilakukan kustomisasi desain markup language yang akan digunakan. Satu hal yang menonjol pada XML adalah tersedianya perintah-perintah dasar yang dapat digunakan untuk berbagi informasi dengan mudah antara komputer, aplikasi dan sistem operasi yang berbeda.

Beberapa keunggulan XML:

- Simplicity

  Mudah untuk dibaca oleh programmer dan komputer.

- Extensibility

  Kemampuan untuk menambah styles, linking dan referencing.

- Interoperability

  Dapat digunakan diberbagai macam platform dan tool.

Aturan paling dasar dari pembuatan dokumen XML adalah *well-formed*. Seperti apa *well-formed*, perhatikan Kode 7.1 berikut ini:

Kode 7.1 Well-formed XML

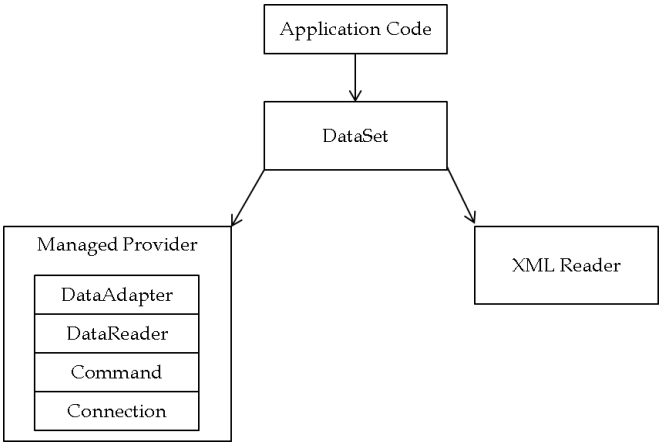
```
<Mahasiswa>
   <NIM>01410100077</NIM>
   <Nama>Tegar Heru Susilo</Nama>
   <Alamat>Krian</Alamat>
   <OrangTua status="Kandung">
      <Ayah>
         <Nama>Nama Ayah</Nama>
         <Alamat>Surabaya</Alamat>
      </Ayah>
      <Ibu>
         <Nama>Nama Ibu</Nama>
         <Alamat>Surabaya</Alamat>
      </Ibu>
   </OrangTua>
   <Telepon HP="085648322xxx" Rumah="-"
   Fax="-" />
</Mahasiswa>
```

Status dalam elemen OrangTua merupakan atribut elemen. Atribut ini tidak boleh sama dalam satu elemen dan mempunyai nilai

atribut. Untuk elemen Telepon, end tag berupa tanda / di akhir start tag.

1. ADO.NET dan Arsitektur XML

ADO.NET tidak hanya mendukung data relasional seperti dalam database, melainkan dapat juga terintegrasi dengan data hirarki seperti XML.



Gambar 5. Arsitektur XML

DataSet dapat dibuat melalui obyek pada Managed Provider jika data diambil langsung dari database. Selain itu, untuk pembuatan DataSet dapat menggunakan XmlReader jika data yang diambil tersedia dalam format XML. Dan dapat pula disimpan kembali dalam bentuk XML menggunaan XmlWriter

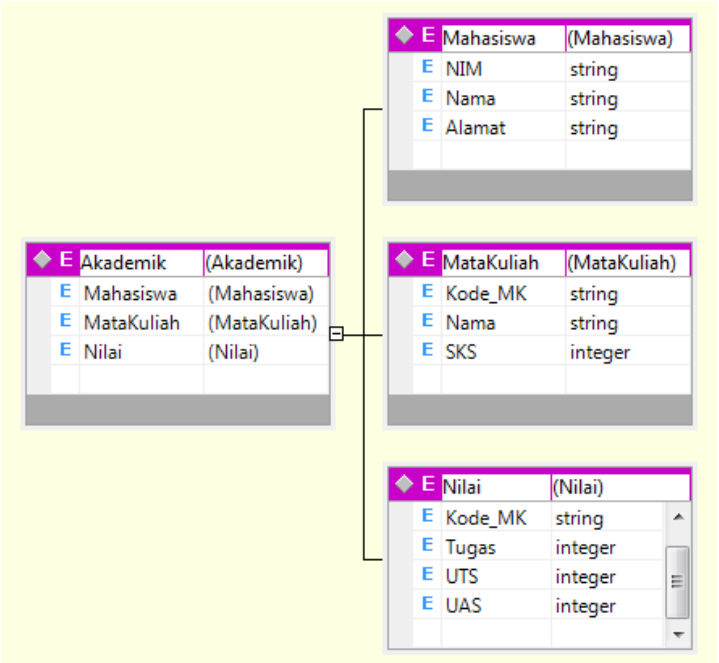
2. XML Schema

Sebuah format XML yang digunakan untuk menyimpan struktur XML layaknya sebuah database. Sehingga pada XML Schema ini, berisi informasi tentang Tabel (nama tabel beserta field dan tipe datanya), Constraint dan Relationship.

Yang akan kami sampaikan disini adalah langkah-langkah dalam membuat XMLSchema. Cara membuat XMLSchema tidak hanya dari design saja, tetapi juga dari kode. Asumsi yang dipakai dalam step-by-step ini adalah bahwa dalam *solution* sudah terdapat sebuah file XMLSchema.

1. Penambahan elemen database. Kalau dalam sistem database, database itu sendiri bertindak sebagai wadah dari tabel. Sehingga kalau diimplementasikan dalam XMLSchema, dibutuhkan sebuah elemen sebagai wadah tersebut.

2. Logikanya, tabel ada di dalamnya database. Untuk menambahkan elemen tabel, tinggal *add element* dari dalam elemen database-nya. Hasil akhir dari pembuatan XmlSchema adalah sebagai berikut:



Gambar 6. Hasil akhir XmlSchema

3. *Primary Key*. Menambahkan elemen yang satu ini sangat mudah. Penambahan dilakukan di area kode XML-nya.

   Yang pertama, tambahkan atribut `xmlns:data` pada elemen `xs:schema`. Ini ibarat `Imports` dalam form. Perhatikan Kode 7.2 berikut ini:

Kode 7.2 xmlns:data

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
id="XSDAkademik"
targetNamespace=
"http://tempuri.org/XSDAkademik.xsd"
elementFormDefault="qualified"
xmlns="http://tempuri.org/XSDAkademik.xsd"
xmlns:mstns=
"http://tempuri.org/XSDAkademik.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:data=
"urn:schemas-microsoft-com:xml-msdata"
>
```

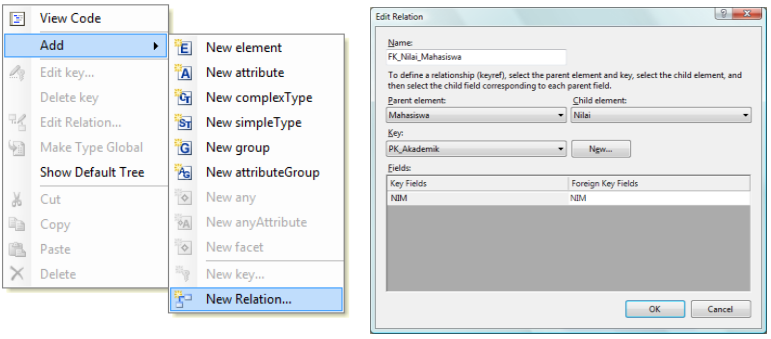
Yang kedua, membuat elemen baru didalam `xs:element` (Kode 7.3). Elemen baru inilah yang disebut dengan *constraint*.

Kode 7.3 xs:element

```
</xs:element>
   <xs:key name="PK_Akademik"
   data:PrimaryKey="true">
      <xs:selector xpath=
      ".//mstns:Mahasiswa"/>
      <xs:field xpath="mstns:NIM" />
   </xs:key>
</xs:element>
```

Dari Kode 7.3, baris kedua untuk nama *constraint*. Ketiga untuk aktivasi constraint. Keempat dan kelima untuk tabel dan kolom apa.

4. Foreign Key.

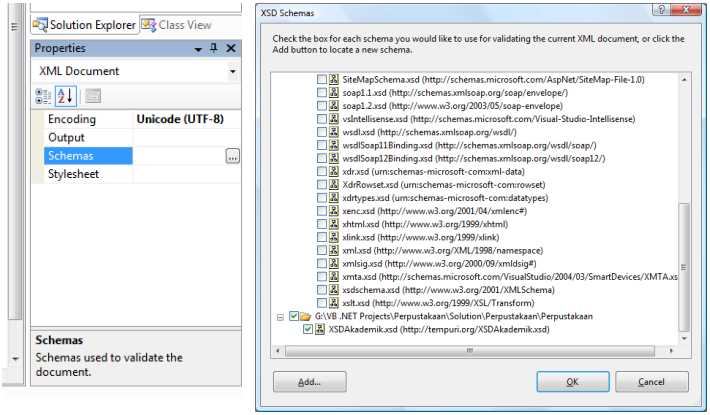


Gambar 7. Membuat Foreign Key

3. XML File

Bila XML Schema digunakan untuk membuat struktur database, berbeda halnya dengan XML File. XML File digunakan untuk menampung data. Dalam step-by-step ini, asumsi yang dipakai adalah sudah terdapat file XMLFile pada solution.

1. Isi properti Schemas. Properti ini seperti foreign key. Jadi setiap kali ada penambahan data (ketika design-time), XMLFile dapat menyajikan struktur database pada layar monitor.



Gambar 8. Properti Schema

2. Hapus semua kode di area-code lalu ganti dengan kode program pada Kode 7.4.

Kode 7.4 Kode pada XMLFile

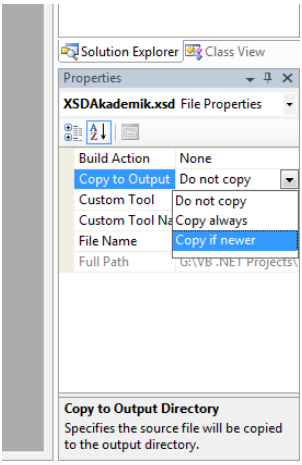
```
<?xml version="1.0" encoding="utf-8" ?>
<Akademik
   xmlns="http://tempuri.org/XSDAkademik.xsd">

</Akademik>
```

3. Manipulasi data ketika design-time dapat dilakukan melalui tampilan datagrid.

4. Pengaplikasian XML pada Form

XML merupakan file atau media penyimpanan permanen bagi Dataset. Yang dibutuhkan tidak hanya XMLFile saja, tetapi juga XMLSchema. Karena XML hanya alat bantu penyimpanan, maka perilaku data mengikuti DataSet. Jadi, yang dibutuhkan dari XML hanya ambil data dan schema, serta untuk penyimpanan.

Ada satu hal yang perlu diubah pada properti dari file fisik XMLSchema dan XMLFile yaitu properti itu adalah Copy to Output (Gambar 9) diganti menjadi Copy if newer.



Gambar 9. Properti Copy to Output

Untuk kode program pengimplementasian XMLSchema dan XMLFile pada form, bisa dilihat pada Kode 7.5.

Kode 7.5 Implementasi XmlSchema dan XmlFile pada Form

```
Dim dsXML As DataSet
...

'untuk ambil data dari XML
dsXML.ReadXmlSchema("XSDAkademik.xsd")
dsXML.ReadXml("XMLAkademik.xml")

'untuk simpan data ke XML
dsXML.WriteXml("XMLAkademik.xml")
```

BAB. VIII
CRYSTAL REPORT

Didalam aplikasi, laporan merupakan rangkaian yang tak terpisah dari hirarki komponen. Ada banyak tools yang bisa digunakan untuk membuat laporan. Model pelaporan yang lagi trend adalah model Executive Information System (EIS) yang hanya menyajikan model grafik kompleks secara real-time tanpa ada fungsi cetak. Pada bab ini khususnya, hanya dibahas mengenai pembuatan laporan cetak menggunakan Crystal Report.

Materi
Aplikasi tidak terkoneksi

Crystal Report untuk Visual Studio .NET merupakan tool standar untuk membuat laporan. Memberikan kemudahan dalam membuat laporan yang interaktif dan presentatif bagi semua platform .NET, baik melalui WEB maupun aplikasi desktop.

1. Membuat Crystal Report

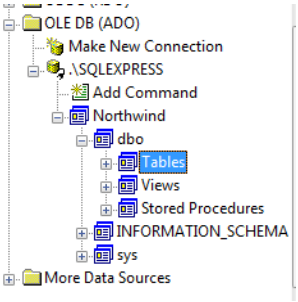
Untuk membuat laporan dengan menggunakan Crystal Report, ikuti langkah-langkah berikut: (tetap dengan asumsi bahwa file Crystal Report sudah tersedia pada solution)

1. Pertama kali CR memberikan Crystal Report Gallery. Disini terdapat pilihan untuk membuat CR memakai report WIZARD, mulai dari BLANK, atau REPORT yang sudah ada. Khusus untuk WIZARD, bisa dipilih salah satu diantara 3 Expert, yaitu (1) Standard, (2) Cross-Tab, dan (3) Mail Label. Sebagai awal, pilih WIZARD-Standard. **OK**.

2. *Standard Report Creation Wizard – Data*. Di sini dibuat koneksi ke database. Ada banyak database yang bisa di-handle oleh CR. Untuk saat ini, lakukan koneksi ke SQLServer dengan mengakses pilihan Create New Connection – OLE DB (ADO).

3. *OLE DB (ADO) – OLE DB Provider*. Pilih Microsoft OLE DB Provider for SQL Server. **NEXT**.

4. *OLE DB (ADO) – Connection Information*. Isi nama server dan database-nya. Pilih juga cara otentikasi-nya. **FINISH**.



Gambar 10. Hasil dari koneksi

5. *Standard Report Creation Wizard – Data*. Setelah koneksi selesai, pindahkan tabel-tabel (ke kotak sebelah kanan) sesuai dengan kebutuhan. **NEXT**.

6. *Standard Report Creation Wizard – Link*. Ini muncul jika hanya jika ada lebih dari satu tabel pada proses kelima. Secara default, terjadi hubungan (link) antar nama yang sama atau antara PK dan FK. Link ini bisa ditambah, diubah, bahkan dihapus. **NEXT**.

7. *Standard Report Creation Wizard – Fields*. data apa saja yang akan ditampilkan dalam CR. Yang perlu dilakukan hanyalah memindahkan kolom-kolom ke kotak sebelah kanan dengan urutan sesuai keinginan. **NEXT**.

8. *Standard Report Creation Wizard – Grouping*. Ini pengelompokan. Sebagai contoh pengelompokan produk berdasarkan kategori akan menampilkan hasil yang berbeda dengan perngelompokan produk berdasarkan supplier. Terlebih jika lebih dari satu pengelompokan. Pengelompokan berdasarkan kategori dan supplier akan memberikan hasil yang berbeda dengan pengelompokan supplier dan kategori. **NEXT**.

   Contoh:

   - **Kategori:**

     Sayur
     - Wortel
     - Kubis

     Buah
     - Apel
     - Pear

   - **Supplier:**

     PT. Abc
     - Apel

     PT. Xyz
     - Wortel
     - Kubis
     - Pear

- **Kategori - Supplier:**

```
Sayur
        PT. Xyz
                Wortel
                Kubis
Buah
        PT. Abc
                Apel
        PT. Xyz
                Pear
```

- **Supplier - Kategori:**

```
PT. Abc
        Buah
                Apel
PT XYZ
        Sayur
                Wortel
                Kubis
        Buah
                Pear
```

9. *Standard Report Creation Wizard – Summaries*. Dengan adanya pengelompokan, ada fitur untuk *summary* (sum, count, max, avg, median, dll) dan auto Grand Total. **NEXT**.

10. *Standard Report Creation Wizard – Group Sorting*. Bagian ini bisa diabaikan dengan nilai default None. **NEXT**.

11. *Standard Report Creation Wizard – Chart*. Penambahan grafik memberikan *resume* singkat tentang laporan. Secara default, CR akan mengisi 3 inputan dibawahnya secara otomatis berdasarkan pengelompokan dan *summary* yang telah diisi sebelumnya. **NEXT**.

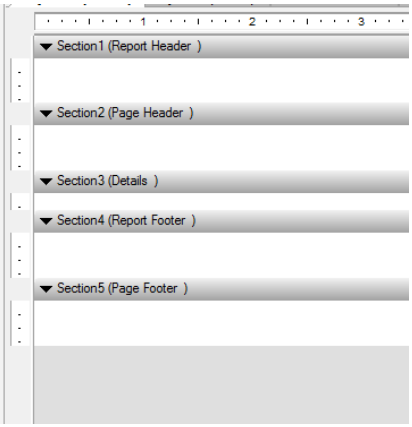
12. *Standard Report Creation Wizard – Record Selection*. Data secara default akan ditampilkan semua. Pembatasan data-data yang ditampilan dapat menggunakan *record selection*. Untuk CR yang lebih dinamis, *record selection* dapat dikolaborasikan dengan *parameter*. Oleh karena itu, kosongi terlebih dulu. **NEXT**.

13. *Standard Report Creation Wizard – Report Style*. Menyediakan beberapa template tampilan. **FINISH**.

14. CR sudah jadi. Yang terlihat sekarang adalah *design*. Untuk preview dapat menggunakan *Main Report Preview* (ada di bagian bawah CR).

2. Bagian-bagian dalam CR

Ada 5 *section* dalam CR. Masing-masing punya fungsi dan perlakuan yang berbeda (Gambar 11).



Gambar 11. Bagian dalam CRe

1. Report Header
   Biasanya digunakan untuk menampilkan KOP laporan. Hanya ditampilkan sekali dalam laporan, yaitu dihalaman pertama laporan. Selain itu secara default, Chart ada disini.

2. Page Header
   Judul laporan, header kolom, dan hal-hal lain yang sejenis bisa ditampilkan disini. Ditampilkan satu kali di setiap halaman.

3. Details
   Untuk menampilkan data. Diulang-ulang sebanyak data yang ditampilkan.

4. Report Footer
   Biasanya untuk menampilkan *grand total*. Ditampilkan satu kali dalam laporan, yaitu di halaman terakhir di bagian paling bawah. Bagian ini ibarat satu paket dengan *Report Header*.

5. Page Footer
   Umumnya halaman dan informasi lain laporan ada disini. Ditampilkan satu kali di setiap halaman. Bagian ini ibarat satu paket dengan *Page Header*.

Masing-masing *section* bisa ditambah. Hanya saja, tidak semua efektif, yang pasti tergantung kebutuhan. Untu setiap pengelompokan, secara default ada *section GroupHeader* dan *GroupFooter*. *GroupHeader* untuk menampilkan nama pengelompokan, sedangkan *GroupFooter* untuk menampilkan *summary* (jika ada).

2.1. Toolbox

Untuk mempercantik laporan, walaupun tidak banyak yang bisa dilakukan, CR menyediakan beberapa *control* di *Toolbox* yang bisa dipakai. Ada 3 *control* disini:

1. Text Object
   Control yang digunakan untuk menampilkan tulisan pada laporan. Biasanya digunakan untuk menampilkan judul laporan dan informasi lain yang bersifat permanen.

2. Line Object
   Control yang digunakan untuk membuat garis vertikal atau horisontal, tetapi tidak bisa digunakan membuat garis diagonal.

3. Box Object
   Control yang digunakan untuk membuat tampilan kotak, persegi panjang atau *ellips* (jika sudutnya *rounded*).

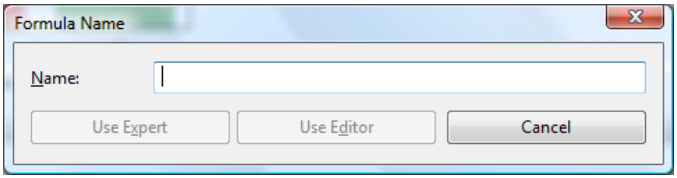
2.2. Formula Field

Ada kalanya data yang disimpan tidak sesuai dengan tampilan yang diinginkan (karena efisiensi di penyimpanan). Misalnya jenis kelamin yang hanya disimpan P dan W, ingin ditampilkan dengan

Pria dan Wanita. Logikanya, kita butuh sebuah rumusan yang ditentukan oleh *state* awalnya. Jika *state* awal bernilai P maka tampilkan Pria, sedangkan jika *state* awal bernilai W maka tampilkan Wanita.

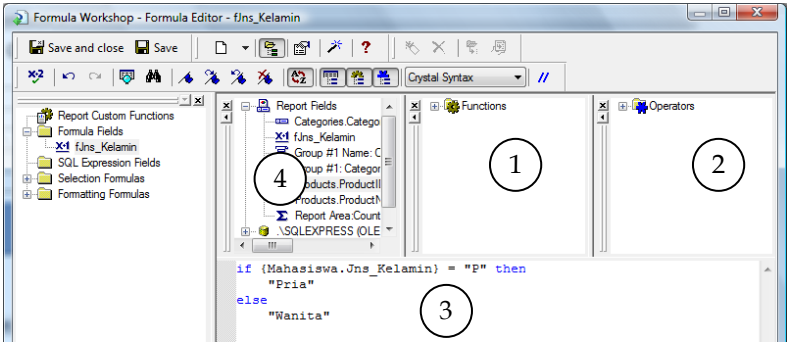
Untuk menampung rumusan ini, dibutuhkan *Formula Field*. Cukup dengan menambahkan *formula* baru di *Formula Field*. *Formulai Field* ini ada di *Field Explorer*.

1. Isikan nama *formula field*-nya. **Use Editor**.



Gambar 12. Formula Name

2. Isikan *formula*-nya pada *Formula Editor*. **Save and Close**.



Gambar 13. Formula Editor

Keterangan Gambar 13:

1. *Functions*: CR menyediakan berbagai function yang bisa kamu gunakan mulai dari konversi tipe data, *string*, *date and time*, dll. Tinggal 2xklik pada function yang ingin digunakan, secara otomatis akan ditulis di poin (3).
2. *Operators*: CR menyediakan beragai operator yang bisa kamu gunakan mulai dari aritmatik, *boolean*, *array*, *variable declaration*, dll.

3. Tempat menuliskan rumusan. Penulisan rumusan hampir sama dengan bahasa pemrograman pada umumnya.
4. Field-field yang bisa kamu pake di rumusan. Mulai dari *database field*, *formula field*, *parameter field*, sampai *running total field*. Tinggal 2xklik pada *field* yang ingin kamu pake, secara otomatis akan di tulis di poin (3).

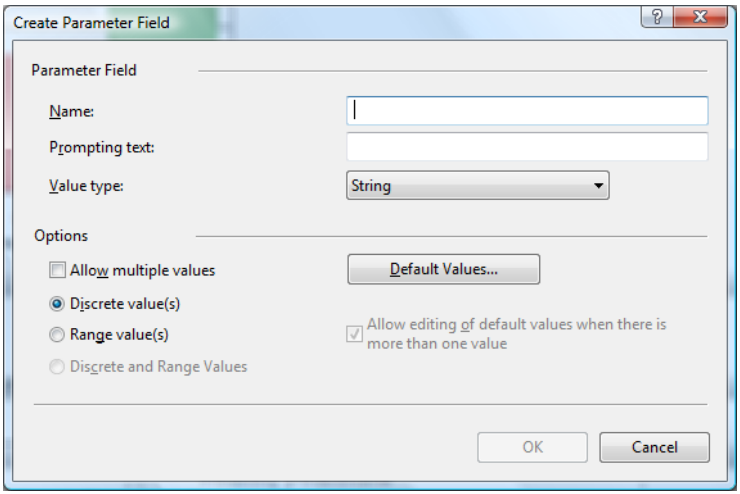
3. *Drag n Drop* nama *formula* yang ada di *Formula Field* kedalam *design* CR.

2.3. Parameter Field

Contoh: *Function Tambah(a as integer, b as integer) as Integer*, a dan b adalah *parameter*. Dari *parameter* tersebut, fungsi tambah melakukan prosesnya dan mengembalikan nilai sesuai *parameter* dan proses.

Parameter pada CR mempunyai fungsi yang sama seperti *parameter* dalam contoh fungsi tersebut diatas. Digunakan untuk apa? Banyak sekali proses yang menggunakan *parameter* dan salah satunya sudah diketahui di awal-awal materi bab ini yaitu sebagai nilai kondisi dari *record selection*. Selain itu, bisa digunakan secara langsung di semua *section* CR-nya.

Untuk membuatnya, cukup mengisi *Create Parameter Field* seperti Gambar 14.



Gambar 14. Create Parameter Field

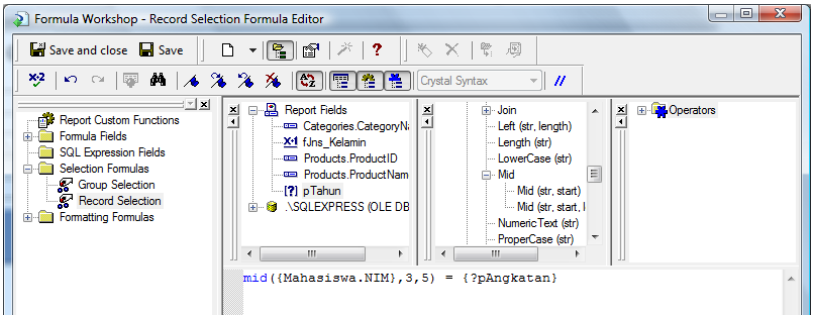
Isikan nama *parameter*, abaikan *prompting text*, pilih *value type* sesuai kebutuhan, dan lakukan *setting options* sesuai kebutuhan. **OK**.

2.4. Filtering Data

Telah disebutkan bahwa CR bisa melakukan *filtering data* secara dinamis menggunakan bantuan *parameter*. Proses *filtering* ini membutuhkan *record selection formula*. Untuk bisa menggunakan semua ini, ikuti langkah-langkah berikut ini:

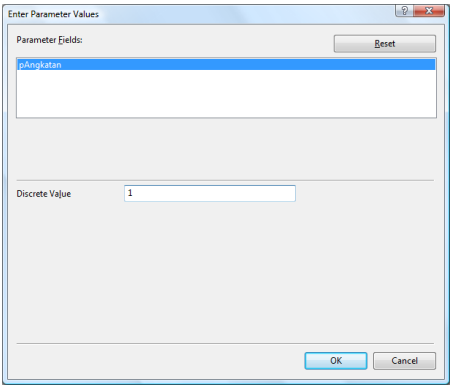
FYI, selain dari record selection formula, filtering juga bisa dilakukan secara langsung dari form dengan menggunakan properti *RecordSelectionFormula* dari objek *ReportDocument*. Pengisian properti ini sama seperti pengisian rumusan dalam *record selection formula editor*.

1. Akses melalui *pop-up menu*. *Report – Selection Formula – Record*.
2. Isikan rumusannya. **Save and Close**.



Gambar 15. Record Selection Formula Editor

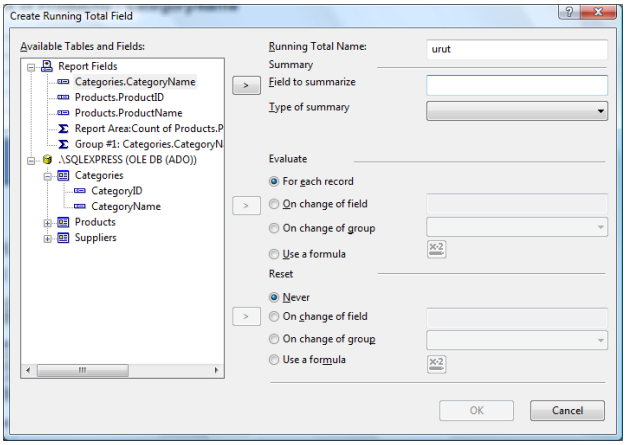
3. Yang perlu diperhatikan, perlakuan *record selection* ini sama seperti *where* dalam *query*. Jadi tidak ada yang namanya *looping*, percabangan, dan logika bahasa pemrograman lain pada umumnya.
4. Ketika di-preview, CR menyuguhi disuguhi pertanyaan tentang nilai dari parameter yang kamu dipakai (Gambar 16).



Gambar 16. Enter Parameter Value

2.5. Running Total Field

Fungsi utamanya adalah sebagai bantuan dalam penomoran. Untuk lebih jelasnya mengenai bagian ini, dapat ditanyakan kepada pengajar.



Gambar 17. Create Running Total Field

3. Implementasi pada aplikasi

Dibutuhkan *CrystalReportViewer* dalam form yang akan dipakai untuk menampilkan CR. Lakukan pengubahan sepertlunya pada beberapa property CrystalReportViewer. Untuk mengisi control ini, perhatikan Kode 8.1

Snippet Code 0-1 Binding CRViewer ke CR

```
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared
Imports System.Data.SqlClient

Public Class Form1
   Dim objRepDoc As New ReportDocument
   Dim objConInfo As New ConnectionInfo
   Dim objTableLogOnInfo As New TableLogOnInfo

   Private Sub Form_Load(…) _
   Handles MyBase.Load
      objRepDoc = New CrystalReport1
      objTableLogOnInfo = _
      objRepDoc.Database.Tables(0).LogOnInfo

      objConInfo.ServerName = "(local)"
      objConInfo.UserID = "myUser"
      objConInfo.Password = "myPassword"
      objConInfo.DatabaseName = "dbLatihan"
      objTableLogOnInfo.ConnectionInfo = _
      objConInfo
      objRepDoc.Database.Tables(0   ). _
      ApplyLogOnInfo(objTableLogOnInfo)

      CrystalReportViewer1.ReportSource = _
      objRepDoc
   End Sub
End Class
```