



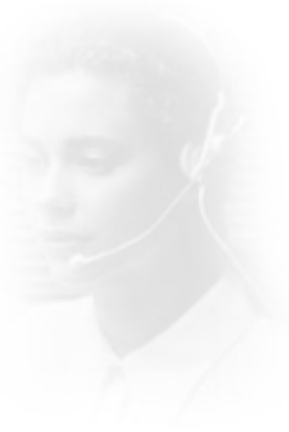
JDBC

Pemrograman Lanjut

Tita Karlita

Pembahasan

- Sistem Database
- Pengantar teknologi JDBC
- Sejarah JDBC
- Desain JDBC
- JDBC Driver
- Arsitektur aplikasi JDBC
- Petunjuk langkah penggunaan JDBC
- Retrieve data dari ResultSet



Sistem Database

- Berbagai macam jenis sistem database :
 - High performance commercial databases – eg. Oracle, DB2, Informix, Microsoft SQL server
 - Open-source – eg. PostgreSQL, MySQL and Interbase
 - Lightweight Java databases – eg. Cloudscape, InstantDB and Pointbase.
 - Desktop databases – eg. Paradox and Access.
- Need to choose and install database first.

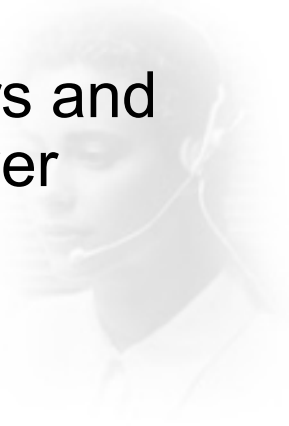


JDBC Kit - Introduction

- JDBC Driver diperlukan bila kode Java ingin berkomunikasi dengan database.
- Diperlukan untuk mengalokasikan dan me-load (locate and load) driver class kedalam Java program untuk database tertentu.
- Java dan JDBC bersifat platform independent dan vendor independent.
- Menyediakan pure Java API untuk mengakses SQL melalui driver manager untuk mengizinkan third-party drivers connect ke spesifik database.
- JDBC API (for application developers and programmers) and SPI (Service Provider Interface containing rules for database vendors and database tool providers to develop drivers).

JDBC History

- In 1995, attempt to extend Java to talk to any database, using only pure Java.
- Too many databases, too many protocols!
- Database vendors in favour of Sun to provide a standard network protocol, but only theirs.
- Result:
 - Sun provides the API for programming and a driver manager while vendors provide their own drivers to plug in.
 - Vendors have to use SPI rules for writing drivers and to register third-party drivers with the Sun's driver manager.

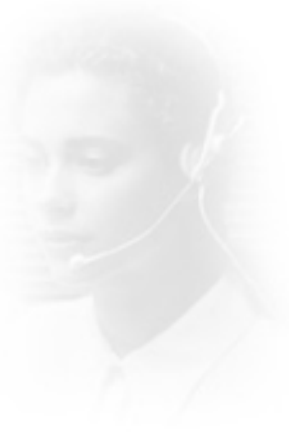


- JDBC adalah Application Programming Interface (API) yang menyediakan fungsi-fungsi dasar untuk akses data.
- JDBC API terdiri atas sejumlah class dan interface yang dapat digunakan untuk menulis aplikasi database dengan menggunakan Java.
- Class dan Interface JDBC terdapat pada package `java.sql`
- Contoh standard API JDBC:
 - Membuat koneksi ke database
 - Mengakses data dengan query
 - Membuat stored (parameterized) query
 - Mendapatkan struktur data dari result query (tabel) :
 - Menyatakan jumlah kolom
 - Mendapatkan nama kolom, dll
 - dll



JDBC Driver

- Masing-masing database server memiliki arsitektur dan sistem yang berbeda → cara komunikasi berbeda.
- Sehingga tiap database server memiliki driver sendiri.
- Untuk aplikasi Java, driver database disebut dengan JDBC Driver.
- **JDBC Driver** adalah software library yang diperlukan agar program JDBC dapat berkomunikasi dengan database tertentu



JDBC Driver

- Masing-masing driver memiliki implementasi dari spesifikasi JDBC secara berbeda.
- Perbedaan:
 - Kecepatan
 - Kestabilan
 - Fasilitas

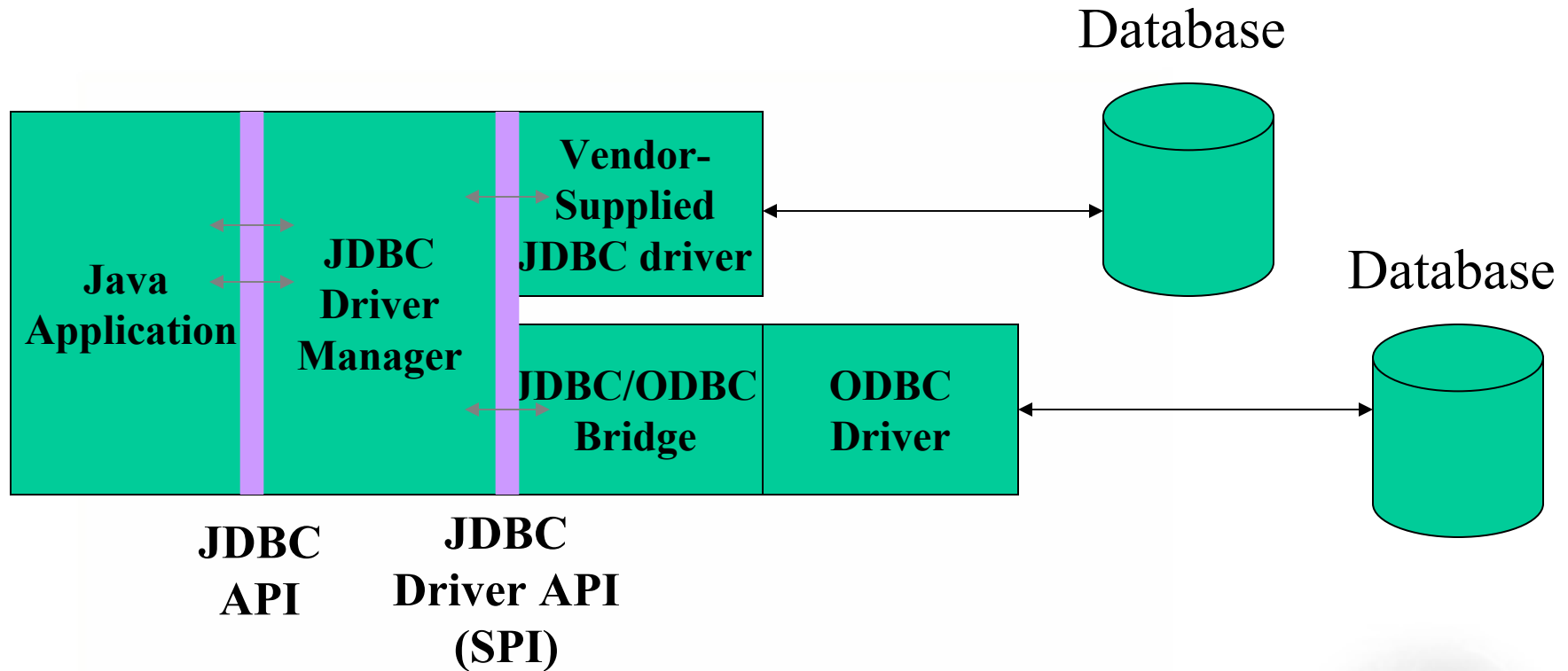


Registrasi JDBC Driver

- Sebelum menggunakan JDBC Driver, dilakukan *registrasi* driver.
 - `Class.forName(String namaDriver).newInstance()`
 - `DriverManager.registerDriver(Driver namaDriver)`
 - Menggunakan properti `jdbc.drivers`



Database Communication



- Two Layers of JDBC API involved in communication.

JDBC Design

“Programs written according to the JDBC API would talk to the JDBC driver manager, which, in turn, would use the drivers that were plugged into it at that moment to talk to the actual database.”

-- Core Java 2 Advanced Features.



Classifications of JDBC Drivers

Type 1: JDBC-ODBC Bridge and ODBC Driver – translates JDBC to Microsoft's ODBC (Open Database Connectivity, written in C) and relies on ODBC to communicate with the database. Need to install ODBC binary code on every client.

Type 2: Native-API Partly Java Drivers – wraps a thin layer of Java around existing native code. Partially written in Java. Converts JDBC calls to calls on the client API for databases. Also need to install system-specific binary code on client machines in addition to a Java library.

Type 3: JDBC-Net pure Java Drivers – Java to talk to a server component which translate DBMS-independent requests to database-specific protocols. Uses a generic network protocol that interfaces with custom middleware. Connects pure Java clients to many different databases. The client library is independent of the actual database – simplifying deployment.

Type 4: Native-Protocol All-Java Drivers – All Java, able to interact with a specific database directly. Translates JDBC requests directly to a database-specific protocol. Useful for intranet access.

A list of JDBC drivers and their corresponding type can be found at:

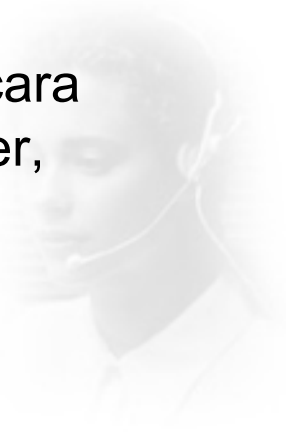
<http://developers.sun.com/product/jdbc/drivers>

Type 1: JDBC-ODBC Bridge and ODBC Driver

- Sebelum JDBC → ODBC sudah ada.
- Bisakah pakai ODBC dari Java untuk koneksi JDBC?
→ bisa yaitu dengan bantuan JDBC API yang berupa JDBC-ODBC bridge
- Kombinasi ini menghasilkan akses JDBC melalui driver ODBC. Kode binari ODBC harus di load pada masing-masing mesin klien yang menggunakan JDBC-ODBC bridge.
- Sun menyediakan sebuah driver JDBC-ODBC bridge, yang cocok untuk keperluan eksperimental dan untuk situasi-situasi dimana tidak ada driver yang sesuai.
- Tipe driver ini juga paling cocok untuk jaringan korporat dimana instalasi klien bukan masalah besar, atau untuk aplikasi server yang ditulis dalam bahasa Java dalam arsitektur 3-tier.

Karakteristik JDBC - ODBC?

- ODBC tidak cocok untuk dipakai langsung dari bahasa pemrograman Java karena ODBC menggunakan bahasa C. Pemanggilan dari Java ke kode native C memiliki beberapa masalah dalam sekuritas, implementasi, robustness, dan portabilitas aplikasi.
- Penerjemahan secara literal dari ODBC C API kedalam Java API bukanlah hal yang baik. Contoh: Java tidak memiliki pointer.
- ODBC sulit untuk dipelajari. ODBC memiliki option-option yang kompleks walaupun untuk query-query sederhana. Sebaliknya JDBC didesain tetap sederhana meski mengerjakan hal-hal kompleks.
- Jika menggunakan ODBC, driver manager harus diinstall secara manual pada tiap mesin klien. Jika menggunakan JDBC driver, kode JDBC secara otomatis diinstall.



Type 2: Native-API Partly Java Drivers

- Tipe driver ini mengkonversi JDBC call kedalam client API untuk Oracle, Sybase, Informix, DB2, dan DBMS yang lain.
- Seperti bridge driver, tipe ini memerlukan kode binari yang spesifik terhadap sistem operasi yang di-load kedalam masing-masing mesin klien.



Type 3: JDBC-Net pure Java Drivers

- Model driver ini menerjemahkan JDBC call kedalam protokol middleware vendor, yang kemudian diterjemahkan ke protokol DBMS oleh server middleware. Middleware menyediakan konektivitas ke berbagai jenis database yang berbeda.



Type 4: Native-Protocol All-Java Drivers

- Model driver ini mengonversi JDBC call langsung kedalam protokol network yang digunakan oleh DBMS, memungkinkan direct call dari mesin klien ke server DBMS dan memberikan solusi praktis untuk akses intranet.



Secara garis besar, teknologi JDBC melakukan:

- Membangun sebuah koneksi ke sumber data (data source).
- Mengirim statement ke sumber data.
- Memproses hasil dari statement tersebut.



Typical JDBC Use

- Two-tier architecture:

Client Machine



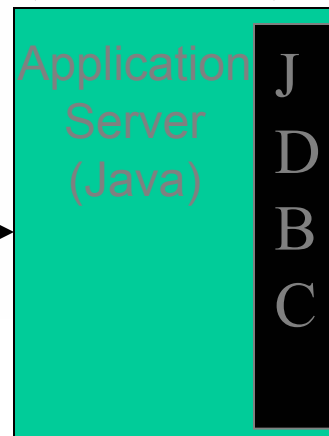
Database

- Three-tier architecture:

Client Machine



Server Machine
(middle-tier)

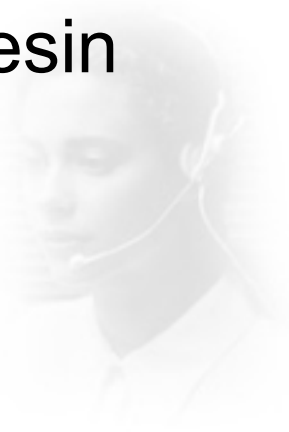


Database

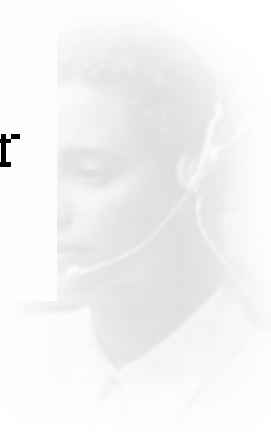
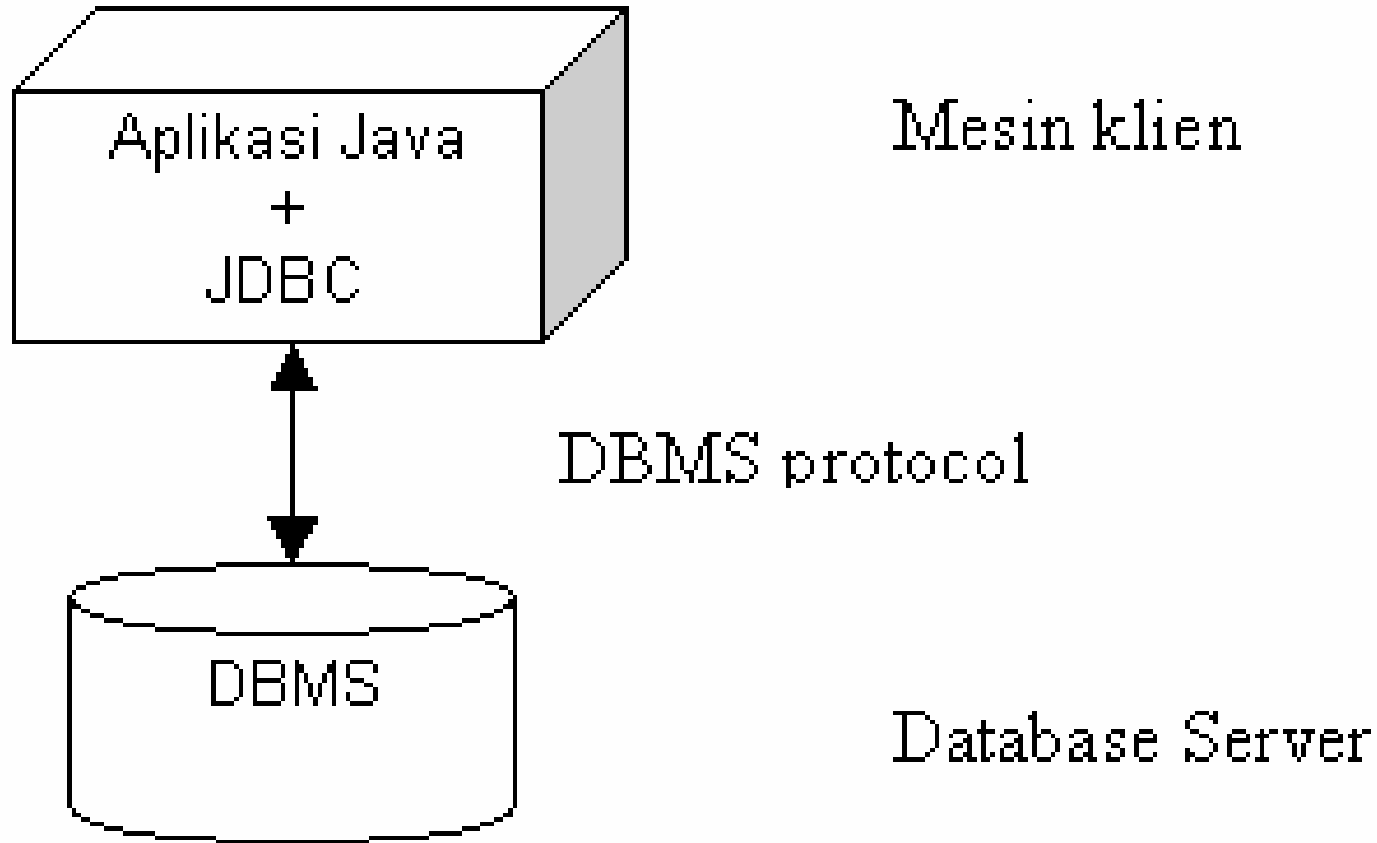


Model 2-tier

- Sebuah applet atau aplikasi java berbicara langsung ke database.
- Sebuah perintah atau statement dari user dikirim ke database dan hasil dari statement dikirim balik ke user.
- Database dapat terletak pada mesin yang sama atau berbeda dengan klien.
- Jika letak database berbeda dengan mesin klien maka disebut dengan client/server. Mesin user → client dan mesin dimana database berada → server.

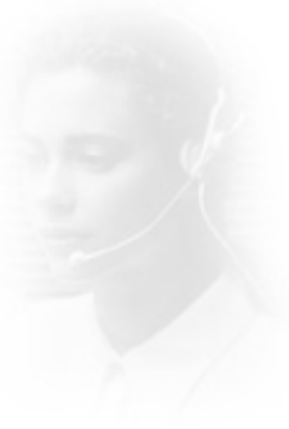


Arsitektur 2-tier

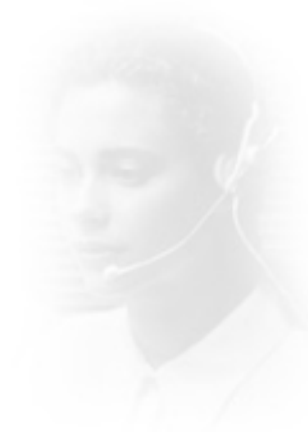
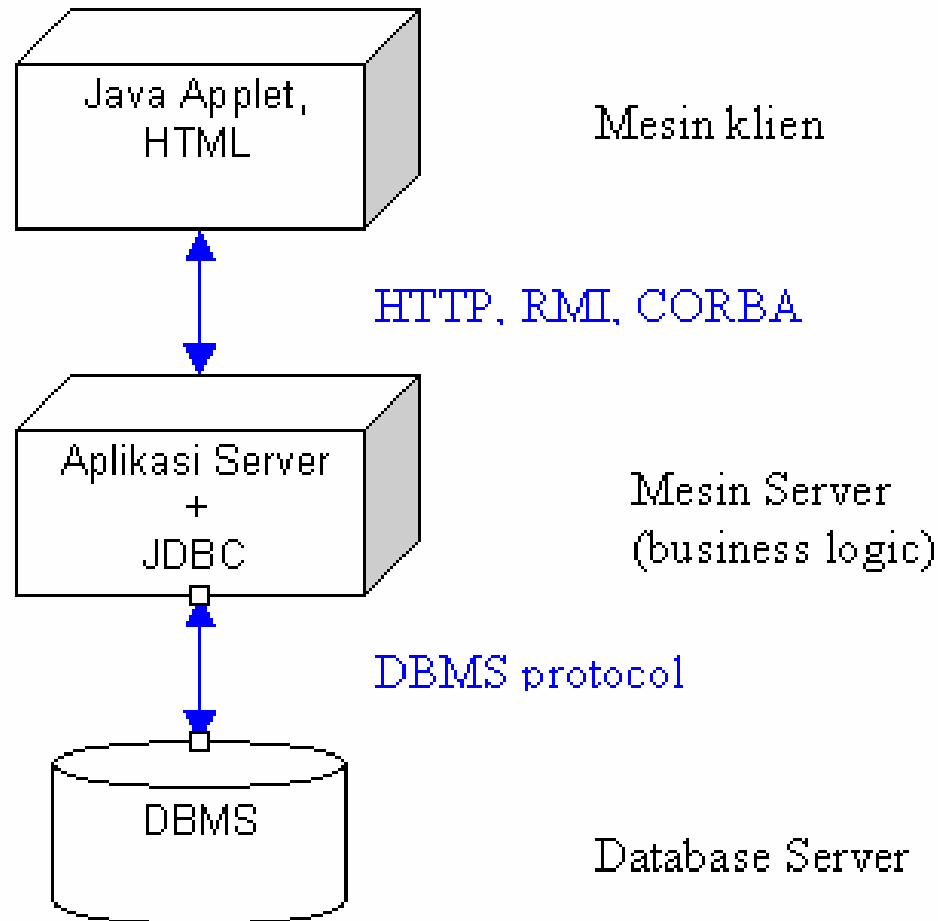


Model 3-tier

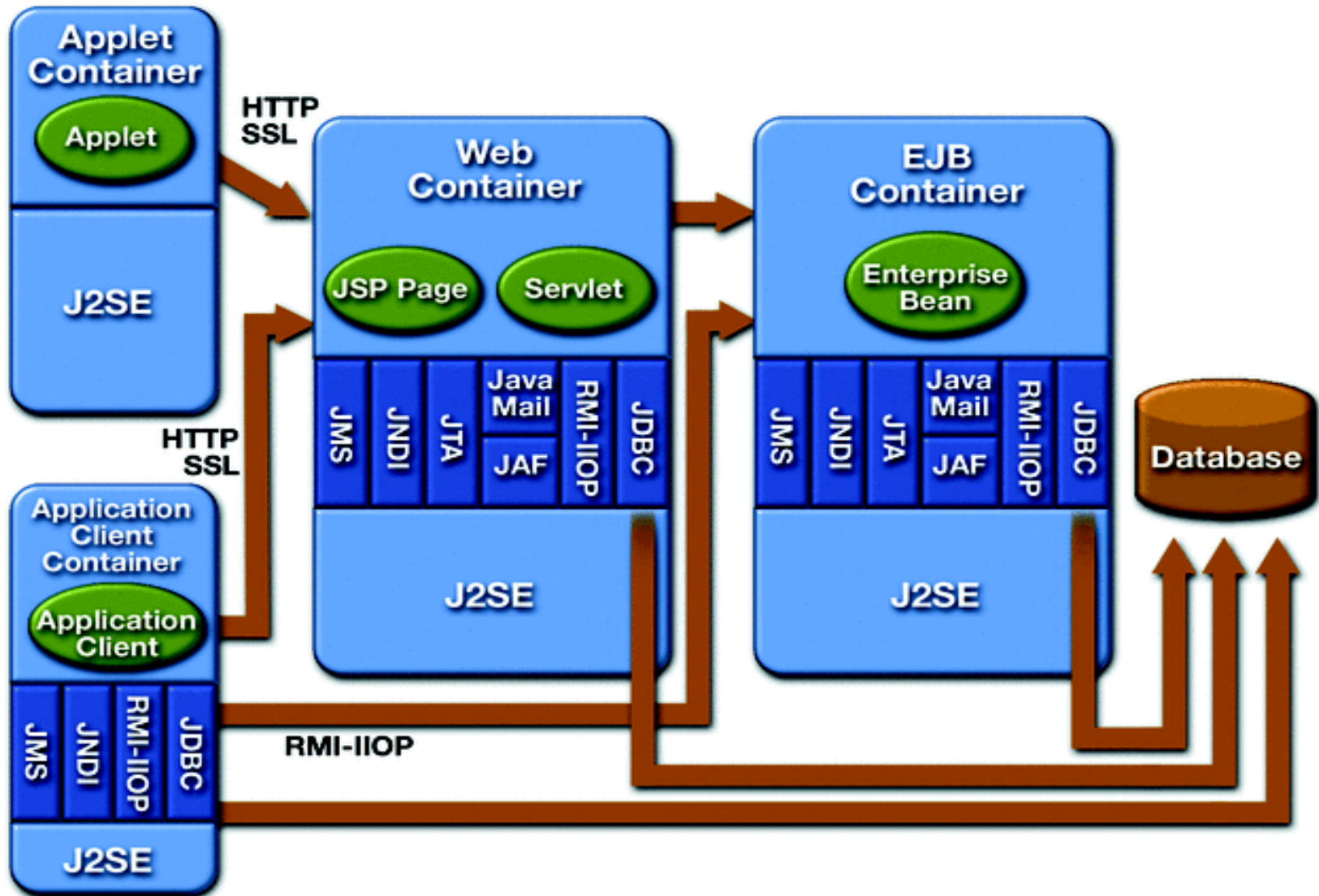
- User mengirimkan perintah ke sebuah middle tier.
- Selanjutnya middle tier mengirimkan perintah tersebut ke database.
- Database memproses perintah tersebut dan mengirim balik hasilnya ke middle tier.
- Kemudian middle tier mengirimkannya ke user.
- Keuntungan: mempermudah aplikasi untuk dideploy dan meningkatkan performansi.



Arsitektur 3-tier



J2EE Architecture – how JDBC fits in



JDBC Data Types

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Langkah Penggunaan JDBC

1. Load driver JDBC
2. Definisikan URL database
3. Membuat dan melakukan koneksi
4. Membuat obyek statement
5. Mengeksekusi query
6. Memproses result
7. Menutup koneksi



1. JDBC : Load Driver

- Driver :
 - library yang digunakan untuk berkomunikasi dengan database server
 - Dengan menggunakan driver, program Java yang menggunakan API JDBC dapat berinteraksi dan dapat dimengerti oleh database server.
- Untuk database yang berbeda dibutuhkan driver yang berbeda.



```
try {  
    Class.forName("sun.jdbc.odbc.JdbcDriver");  
}  
catch (ClassNotFoundException ex) {  
    System.err.println("Driver Error");  
    ex.printStackTrace();  
    System.exit(1);  
}
```

- Contoh di atas jika yang kita gunakan adalah JDBC-ODBC driver.
- Dokumentasi driver anda akan memberikan nama class yang digunakan.



Nama Driver database:

- JDBC-ODBC :
`sun.jdbc.odbc.JdbcOdbcDriver`
- Oracle :
`oracle.jdbc.driver.OracleDriver`
- Sybase :
`com.sybase.jdbc.SybDriver`
- MySQL :
`com.mysql.jdbc.Driver`
- PostgreSQL :
`org.postgresql.Driver`
- Microsoft SQLServer 2000 :
`com.microsoft.jdbc.sqlserver.SQLServerDriver`



2. JDBC : Definisikan koneksi URL

- Menspesifikasikan lokasi database server
- Gunakan dokumentasi driver
- Untuk penggunaan JDBC di applet:
 - database server harus berada pada node yang sama dengan letak applet.
 - Menggunakan proxy server yang me “reroute” request database ke actual server.

- Contoh:

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
    ":" + port + ":" + dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host +
    ":" + port + ":" +
    "?SERVICENAME=" + dbName;
```

Nama URL database:

- JDBC-ODBC :

jdbc:odbc:nama_database

- Oracle :

jdbc:oracle:thin:@nama_host:1521:namaDB

- MySQL :

jdbc:mysql://nama_host:3306/namaDB

- PostgreSQL :

jdbc:postgresql://nama_host:5432/namaDB

- Microsoft SQLServer 2000 :

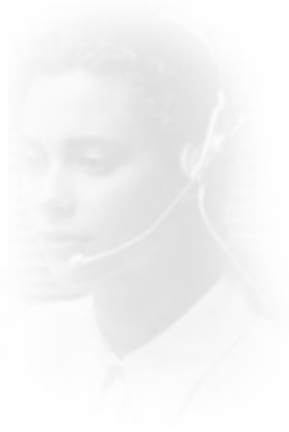
jdbc:microsoft:sqlserver://nama_host:1433;DatabaseName=namaDB



3. JDBC : Membuat Koneksi

- Cara : memanggil method `getConnection` dari class `DriverManager` dengan melewati URL (hasil langkah dua) sebagai argumen.
- `getConnection` akan melempar `SQLException`
- Contoh:

```
String username = "jay_debese";  
String password = "secret";  
Connection connection =  
    DriverManager.getConnection (oracleURL,  
                                username,  
                                password);
```



4. JDBC : Membuat Obyek Statement

- Object Statement digunakan untuk mengirim query dan perintah ke database.
- Object Statement dibuat dengan cara bekerjasama dengan class Connection.
- Cara: memanggil method `createStatement()` dari obyek Connection.
- Contoh:

```
Statement statement = connection.createStatement();
```



5. JDBC : Mengeksekusi Query

- Memanfaatkan object `Statement` untuk memproses query.
- Cara: memanggil method `executeQuery()` dari object `Statement`. → memberikan return value bertipe `ResultSet`
- Returns: `ResultSet`.

```
String sql="select col1, col2, col3 from sometable";  
ResultSet rs=statement.executeQuery(sql) ;
```

Note : Untuk memodifikasi database, gunakan `statement.executeUpdate(sql)`; yang mendukung string sql UPDATE, INSERT INTO, DELETE



ResultSet

- ResultSet memberikan bermacam2 method getXxx dengan parameter index kolom atau nama kolom dan mengembalikan data.
- Method lain object ResultSet:
 - findColumn()
mendapatkan index (integer value) berdasarkan nama kolom.
Kolom pertama mempunyai index 1 bukan 0.
 - getMetaData()
retrieve informasi mengenai ResultSet, returns object ResultSetMetaData.
 - wasNull()
Mengetahui apakah getXxx() menghasilkan SQL null.



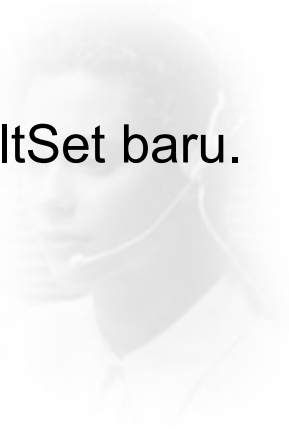
6. JDBC : Memproses result

- Dengan menggunakan method `next()` pada object `ResultSet` → mendapatkan results per satu baris.

- Contoh:

```
String nrp;  
String nama;  
while (rs.next()) {  
    nrp=rs.getString(1);  
    nama=rs.getString(2);  
    System.out.println("NRP : " +nrp);  
    System.out.println("NAMA : " +nama);  
    System.out.println("-----");  
}
```

- Kolom pertama mempunyai index 1 bukan 0.
- Object `ResultSet` otomatis akan ditutup bila ada object `ResultSet` baru.



7. JDBC : Menutup Koneksi

- Harus didefinisikan secara eksplisit.

```
connection.close ();
```

- Karena membuka koneksi adalah mahal, maka penundaan langkah terakhir ini hanya jika masih ada operasi database yang dilakukan.



Contoh

```
import java.sql.*;

public class TestDB {
    public static void main(String[] args) {

        // Use driver from Connect SW.
        String driver = "connect.microsoft.MicrosoftDriver";
        try {
            Class.forName(driver);
            String url = "jdbc:ff-microsoft://" + // FastForward
                "dbtest.apl.jhu.edu:1433/" + // Host:port
                "pubs"; // Database name
            String user = "sa", password="";

            Connection connection =
                DriverManager.getConnection(url, user, password);
            Statement statement = connection.createStatement();
            String query =
                "SELECT col1, col2, col3 FROM testDB";

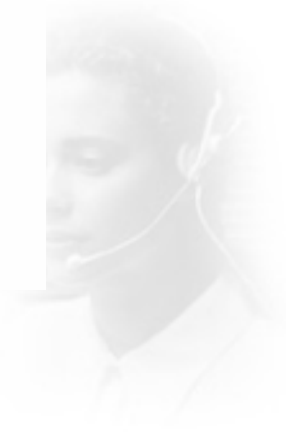
            // Execute query and save results.
            ResultSet results = statement.executeQuery(query);
        }
    }
}
```



```
// Print column names.
String divider = "-----+-----+-----";
System.out.println("Col1 | Col2 | Col3\n" + divider);

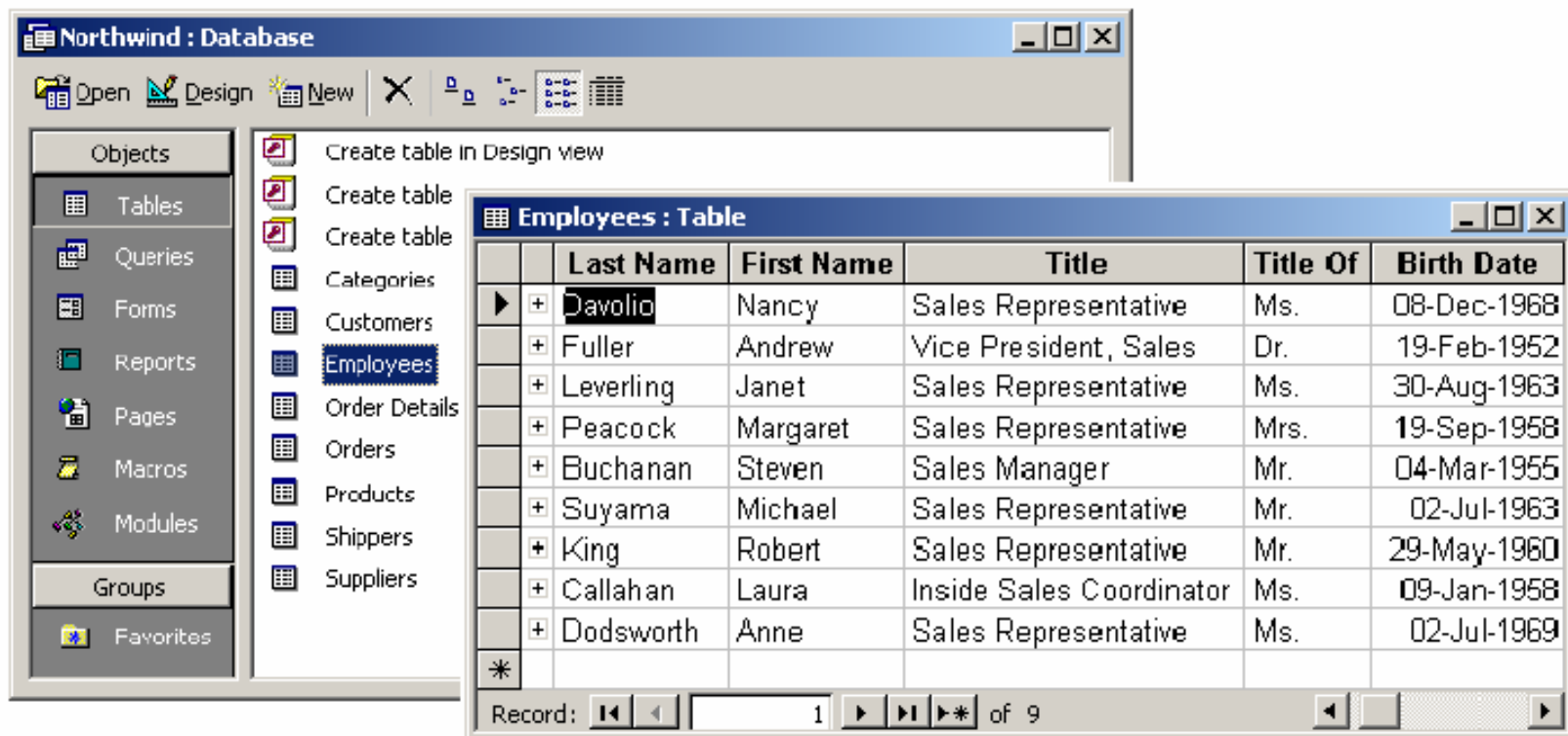
// Print results
while(results.next()) {
    System.out.println
        (pad(results.getString(1), 4) + " | " +
         pad(results.getString(2), 4) + " | " +
         results.getString(3) + "\n" + divider);
}
connection.close();
} catch(ClassNotFoundException cnfe) {
    System.out.println("No such class: " + driver);
} catch(SQLException se) {
    System.out.println("SQLException: " + se);
}
}
...

```



Menggunakan Microsoft Access via ODBC(1)

Northwind sample database



The screenshot displays the Microsoft Access interface for the Northwind database. The 'Employees' table is selected in the 'Objects' pane and is shown in a data view window. The table contains the following data:

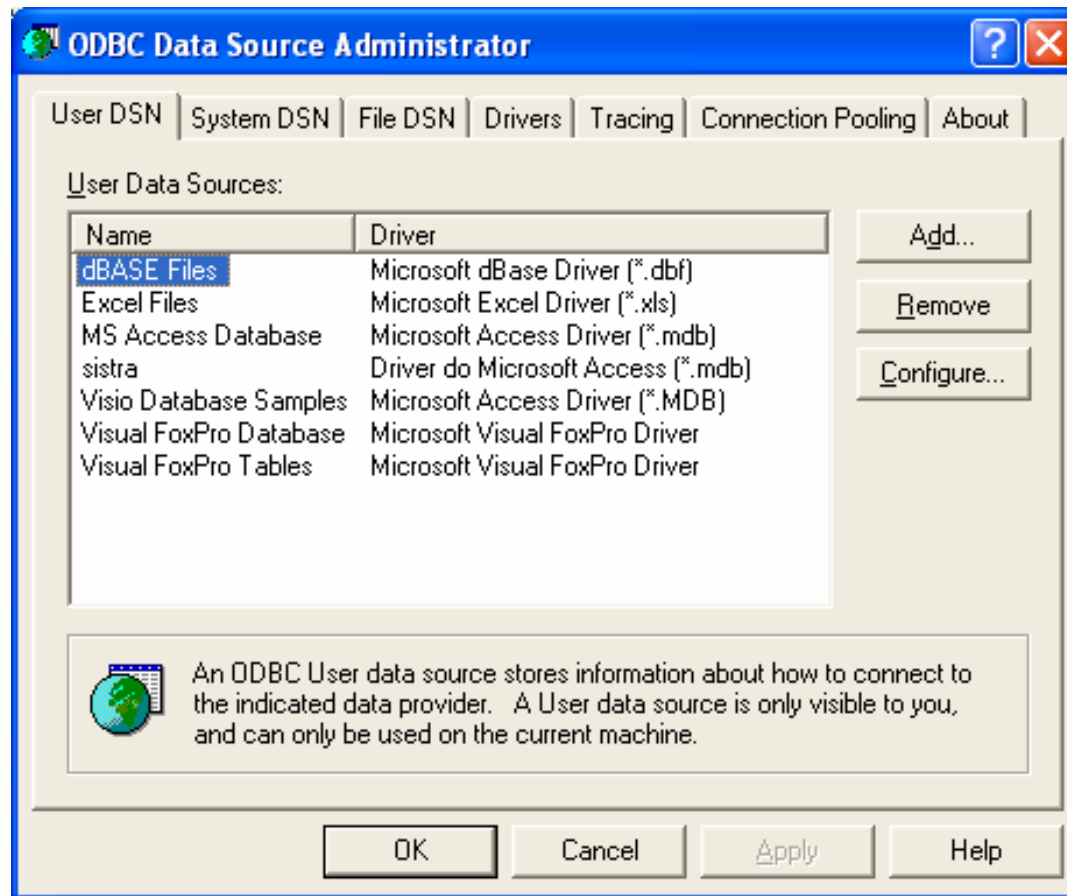
	Last Name	First Name	Title	Title Of	Birth Date
▶ +	Davolio	Nancy	Sales Representative	Ms.	08-Dec-1968
+	Fuller	Andrew	Vice President, Sales	Dr.	19-Feb-1952
+	Leverling	Janet	Sales Representative	Ms.	30-Aug-1963
+	Peacock	Margaret	Sales Representative	Mrs.	19-Sep-1958
+	Buchanan	Steven	Sales Manager	Mr.	04-Mar-1955
+	Suyama	Michael	Sales Representative	Mr.	02-Jul-1963
+	King	Robert	Sales Representative	Mr.	29-May-1960
+	Callahan	Laura	Inside Sales Coordinator	Ms.	09-Jan-1958
+	Dodsworth	Anne	Sales Representative	Ms.	02-Jul-1969
*					

Record: 1 of 9

- Northwind.mdb located in C:\Program Files\Microsoft Office\Office\Samples
- <http://office.microsoft.com/downloads/2000/Nwind2k.aspx>

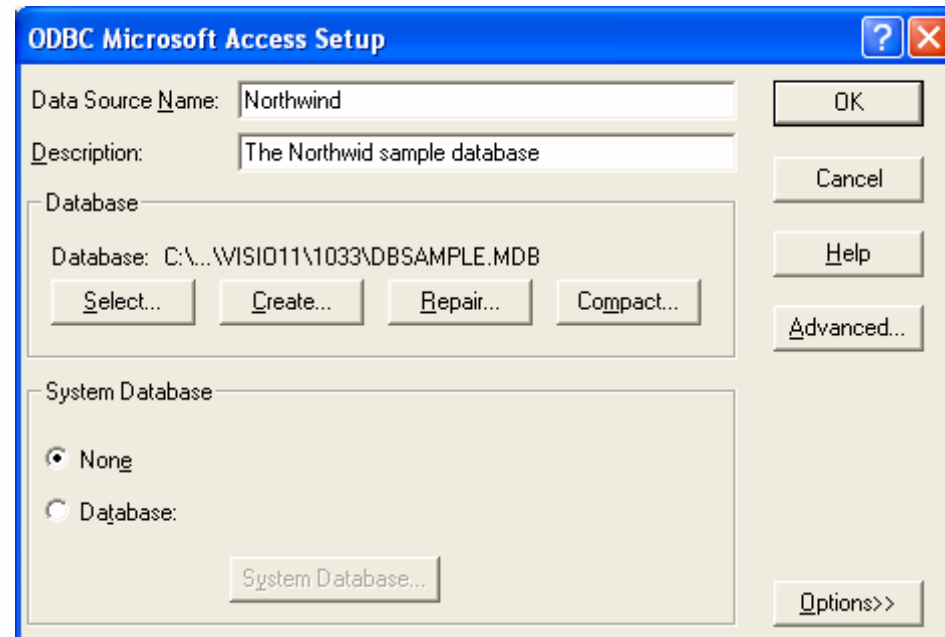
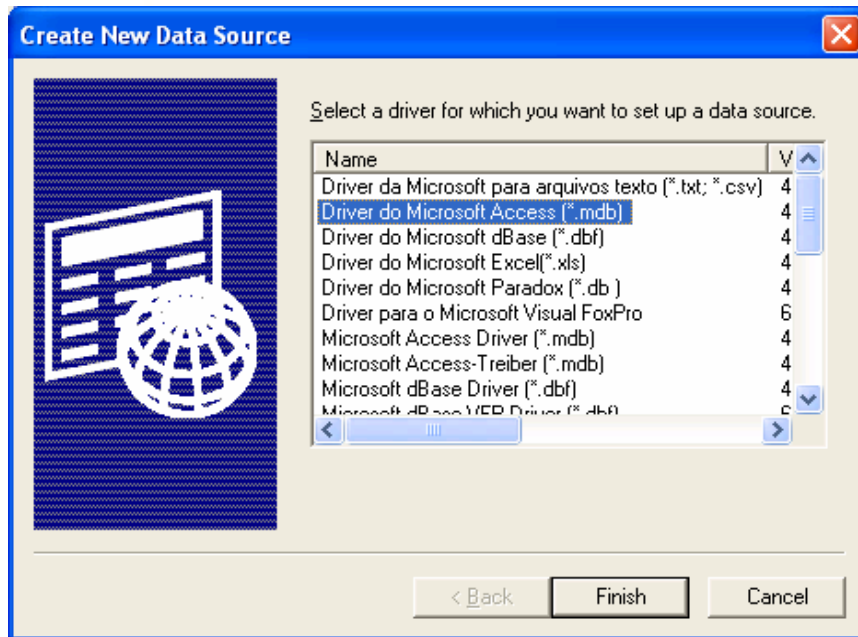
Menggunakan Microsoft Access via ODBC (2)

- Click Start, Settings, Control Panel, Administrative Tools, Data Sources(ODBC), System DSN, dan pilih Add



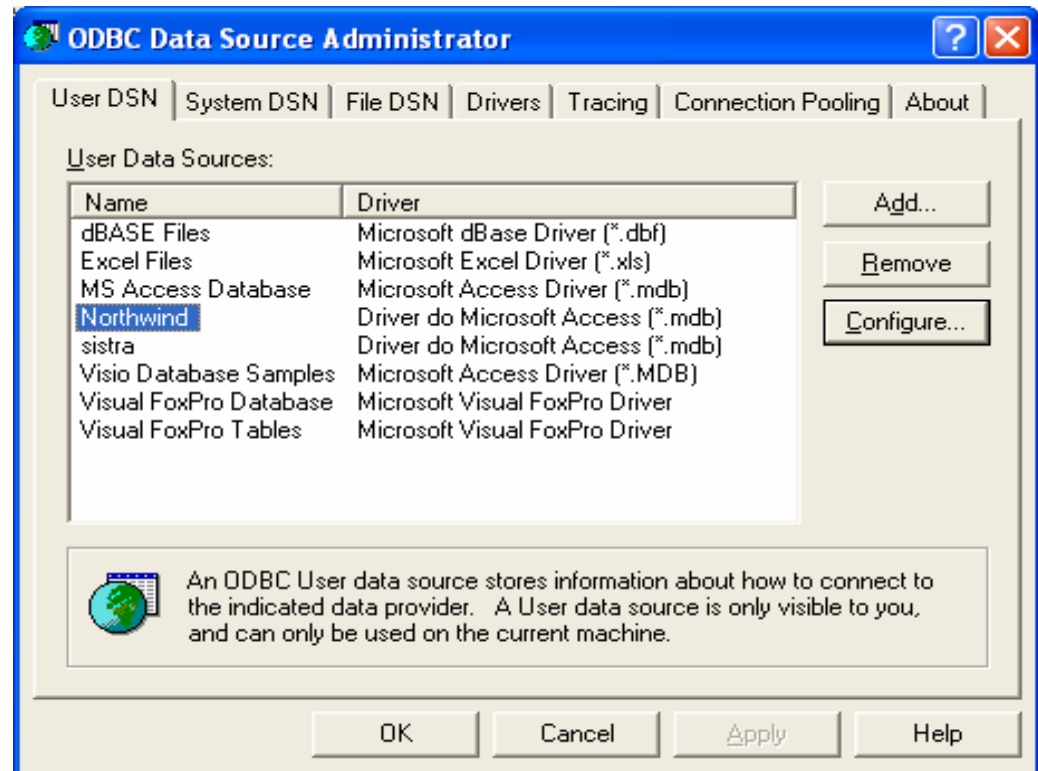
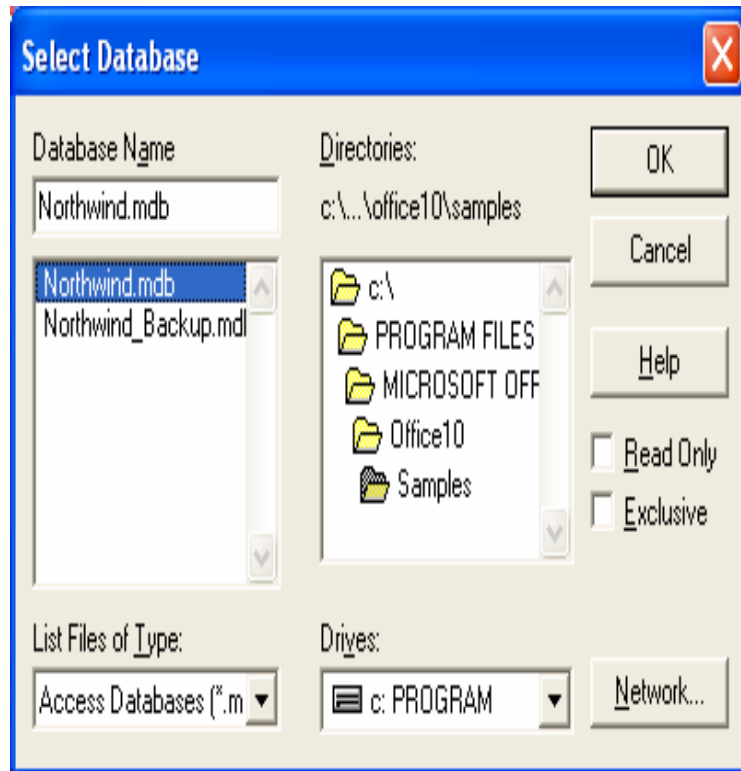
Menggunakan Microsoft Access via ODBC (3)

- Memilih driver Microsoft Access, Finish, ketikkan nama Data Source Name dan tekan Select untuk memilih nama dan lokasi database



Menggunakan Microsoft Access via ODBC (4)

- Navigasi pada directory Samples ms office, pilih Northwind.mdb, tekan OK dan lanjutkan tekan OK pada window II



Menggunakan Microsoft Access via ODBC (5)

- **Gunakan sun.jdbc.JdbcOdbcDriver sebagai nama class dari JDBC driver**

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- **Gunakan "jdbc:odbc:Northwind" sebagai alamat database, dan gunakan empty string pada username dan password**


```
Connection
```

```
con=DriverManager.getConnection("jdbc:odbc:Northwind","","");
```



Simple Northwind Test (1)

```
import java.sql.*;
public class DbTest {
    private Connection con;
    DbTest(){ con=null; }
    public ResultSet dbOpen() {
        String dbname="jdbc:odbc:Northwind";
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection(dbname,"","");
            if (con==null)
                System.err.println("Koneksi ke Database gagal");
            else
                System.err.println("Koneksi ke Database Berhasil");
        }
        catch(ClassNotFoundException ex) {
            System.err.println("Driver Error");
            ex.printStackTrace();
            System.exit(1);
        }
        catch(SQLException ex) {
            System.err.println("Tidak Berhasil Koneksi dengan Sistra");
            System.exit(1);
        }
    }
}
```



Simple Northwind Test (2)

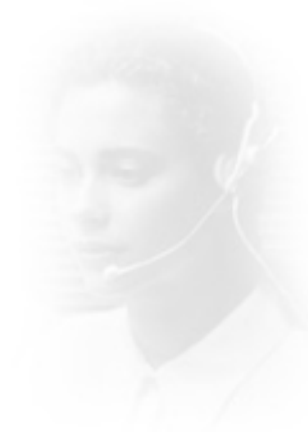
```
ResultSet rs=null;
Statement st;
    try {
        System.out.println("Employees\n" +
                           "=====");
        st=con.createStatement();
        rs=st.executeQuery("SELECT * FROM employees");
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return(rs);
}

public void showEmployee (ResultSet rs) throws SQLException {
    while (rs.next()){
        System.out.print(rs.getString(3)+" ");
        System.out.println(rs.getString(2)+" ");
    }
}
}
```

Simple Northwind Test (3)

```
public void dbClose() {
    try {
        con.close();
    }
    catch(SQLException sqlex){
        System.err.println("Error :Koneksi
        Database tidak Bisa diputus");
    }
}

public static void main(String argv[]) throws Exception {
    DbTest app=new DbTest();
    ResultSet rs;
    rs=app.dbOpen();
    app.showEmployee(rs);
    app.dbClose();
}
}
```



Result Simple Northwind Test

```
C:\j2sdk1.4.1_01\bin\java.exe -classpath
"C:\j2sdk1.4.1_01\jre\lib\rt.jar;D:\DATA" DbTest
Employees
=====
Nancy Davolio
Andrew Fuller
Janet Leverling
Margaret Peacock
Steven Buchanan
Michael Suyama
Robert King
Laura Callahan
Anne Dodsworth
Koneksi ke Database Berhasil
Finished executing
```



References

1. C.S. Horstmann and G. Cornell (2002). Core Java 2. Volume 2 – Advanced Features. Ch 4.
2. J. Wigglesworth (2000). Java Programming – Advanced Topics. Ch 17.
3. Sun Microsystems. JDBC On-line Tutorial.
4. Inixindo, Java Advanced Training

