

Building Database Application with JDBC

資料庫處理

Java Programming



Content

- ◆ 資料庫基本觀念
- ◆ 使用JDBC API 開發資料庫應用程式
- ◆ Transaction 交易模式
- ◆ RowSet 資料集合

Content

- ◆ 資料庫基本觀念
- ◆ 使用JDBC API 開發資料庫應用程式
- ◆ Transaction 交易模式
- ◆ RowSet 資料集合

資料庫基本觀念

◆ 資料庫Database

- 一種永續性存放資料的媒體

◆ DataBase Management System (DBMS)

- 管理資料庫的電腦軟體系統
- 一般具有儲存、擷取、安全保障、備份等基礎功能

◆ 資料庫的型態

- 關聯式資料庫(Relational Database)
- 物件導向型資料庫(Object-Oriented Database)
- 鍵值對資料庫(key-value Database)
- XML 資料庫

常用資料庫管理系統

◆ 開放原始碼資料庫系統

- Apache Derby - Apache開發，純Java資料庫管理系統
- LevelDB - Google研發，鍵 / 值對資料庫
- MySQL - 網路系統上廣泛應用
 - LAMP：MySQL結合Linux系統、PHP語言、Apache網路伺服器
- Xindice - Apache開發，簡單的XML資料庫

◆ 商業資料庫系統

- Oracle - 最受歡迎的商業資料庫
- MS SQL-Server - Microsoft
- Informix
- Sybase

關聯式資料庫

◆ Relational Database.

- 最廣泛被使用的資料庫型態
- 能夠維護及管理具有相依性資料間的完整性
- 由許多資料表(Table) 所組成



(Relational Database Management System)

LID	year	season	title
001	2001	Spring	Soccer League (Spring '01)
002	2001	Summer	Summer Soccer Fest 2001
003	2001	Fall	Fall Soccer League 2001
004	2004	Summer	The Summer of Soccer Love

PID	name	address	city	province	postal_code
047	Steve Sterling	12 Grove Park Road	Manchester	Manchester	M4 6NF
048	Alice Hornblower	62 Woodside Lane	Reading	Berks	RG31 9TT
049	Wally Winkle	17 Chippenham Road	London	London	SW19 4FT

table_name	ID_number
League	005
Player	050

LID	PID	division
001	047	Amateur
001	048	Amateur
002	048	Semi-Pro
002	049	Professional
003	048	Professional

關聯式資料庫

◆ 資料表 Table.

- 欄與列所組成的邏輯性的資料結構。
- 每一列代表一筆資料紀錄(Record), 每一筆紀錄(Record) 都有著相同數量與型態的欄位. 每一個欄位資料的型態及邏輯意義相同

Columns		LID	year	season	title
Rows		001	2001	Spring	Soccer League (Spring '01)
		002	2001	Summer	Summer Soccer Fest 2001
		003	2001	Fall	Fall Soccer League 2001
		004	2004	Summer	The Summer of Soccer Love

PID	name	address	city	province	postal_code
047	Steve Sterling	12 Grove Park Road	Manchester	Manchester	M4 6NF
048	Alice Hornblower	62 Woodside Lane	Reading	Berks	RG31 9TT
049	Wally Winkle	17 Chippenham Road	London	London	SW19 4FT

結構化查詢語言 SQL

◆ Structured Query Language

- 一種與關聯式資料庫系統 RDBMS 進行新增(Insert)、查詢(Query)、修改(Update)、刪除>Delete)等交易(Transaction) 的標準語言.
- 在標準SQL語法的規範下，不同的RDBMS(廠商可能會擁有自己非標準的SQL 語法產生).

基本的 SQL 語法

◆ 新增資料表

➤ 依照所下達的條件建立資料表

➤ Basic Syntax

- CREATE TABLE [**TableName**] (
 ColumnName DataType(size) [ColumnConstraint]
 [,....]
 [TableConstraint]
);

基本的 SQL 語法

◆ 新增資料表

```
CREATE TABLE EMPLOYEE (  
    ID INTEGER NOT NULL,  
    FIRSTNAME VARCHAR(40) NOT NULL,  
    LASTNAME VARCHAR(40) NOT NULL,  
    BIRTHDATE DATE,  
    SALARY REAL,  
    PRIMARY KEY (ID)  
);
```

基本的 SQL 語法

◆ 新增

➤ 將資料新增到資料表中.

➤ Basic Syntax

- INSERT INTO [TableName]([Field]*) VALUES([Field]*);

➤ Example

- INSERT INTO League (LID, yearno, season, title)
VALUES (3, 2003, 'Fall', 'Fall Soccer League (2003)');

基本的 SQL 語法

◆ 查詢

➤ 依照所下達的條件將資料取出.

➤ 可能由許多的資料表中將資料取出.

➤ Basic Syntax

- SELECT [Field*] FROM [TableName] [WHERE CONDITION*];

➤ Example

- SELECT * FROM League;
- SELECT pid,address,city FROM Player WHERE Name = 'Steve Sterling';

基本的 SQL 語法

◆ 修改

➤ 依照所下達的條件修改資料表中的資料.

➤ Basic Syntax

- UPDATE [TableName] SET [FIELD=VALUE]* [WHERE CONDITION*];

➤ Example

- UPDATE League SET Season='Autumn' WHERE lid=3;

基本的 SQL 語法

◆ 刪除

➤ 依照所下達的條件刪除資料表中的資料.

➤ Basic Syntax

- DELETE [TableName] [WHERE CONDITION*];

➤ Example

- DELETE Player WHERE pid=49;

常用SQL資料型態

種類	資料型別	說明
文字	char	固定儲存長度的字串。例如在 char(5)之下，abc 與 abcde 會以"abc "與"abcde"儲存
	varchar	依資料實際長度儲存。例如在 varchar(5)之下，abc與 abcde 會以"abc"與"abcde"儲存
	text	依資料實際長度儲存。text可儲存超過8000個位元的字串
	nchar	固定儲存長度的字串。但可儲存各國不同語系的Unicode字元
	nvarchar	依資料實際長度儲存。但可儲存各國不同語系的Unicode字元
	ntext	依資料實際長度儲存。但可儲存各國不同語系的Unicode字元
日期時間	datetime	儲存 1753/1/1到9999/12/31 之間合法的日期時間
	smalldatetime	儲存 1900/1/1到2079/6/6之間合法的日期時間
數值	int	儲存 -2,147,483,648到2,147,483,647之間的整數
	smallint	儲存-32768到32767之間的整數
	decimal	可儲存包含小數的數值
布林值	bit	True 或 False

Java型態與SQL型態對應

Java 型態	SQL 型態
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	FLOAT
double	DOUBLE
byte[]	BINARY、VARBINARY、LONGBINARY
java.lang.String	CHAR、VARCHAR、LONGVARCHAR
java.math.BigDecimal	NUMERIC、DECIMAL
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Java型態與SQL型態對應

◆ 日期時間在JDBC中，並不是使用java.util.Date

➤ 年、月、日、時、分、秒、毫秒

◆ 在JDBC中要表示日期，是使用java.sql.Date

➤ 年、月、日

◆ 要表示時間的話則是使用java.sql.Time

➤ 時、分、秒

◆ 使用java.sql.Timestamp

➤ 時、分、秒、**微秒**

常用資料限制條件

資料限制條件	說明
NULL/NOT NULL	欄位可以為空值/欄位不可為空值
UNIQUE	資料表中的某欄位或某些欄位的值，在資料表中必需為唯一值 不可重複 ，可以有空值
PRIMARY KEY (主鍵值)	資料表中某欄位或某些欄位為資料表中 唯一 值，用來確認資料表中的每一筆資料， 不可為空值
FOREIGN KEY (外部鍵)	資料表中某欄位或某些欄位需指向另外一個表格主鍵的欄位。用來確定資料的參考完整性(referential integrity)
CHECK	額外的檢查條件，必需為true

基本的 SQL 語法

◆ 刪除資料表

➤ 刪除已建立資料表

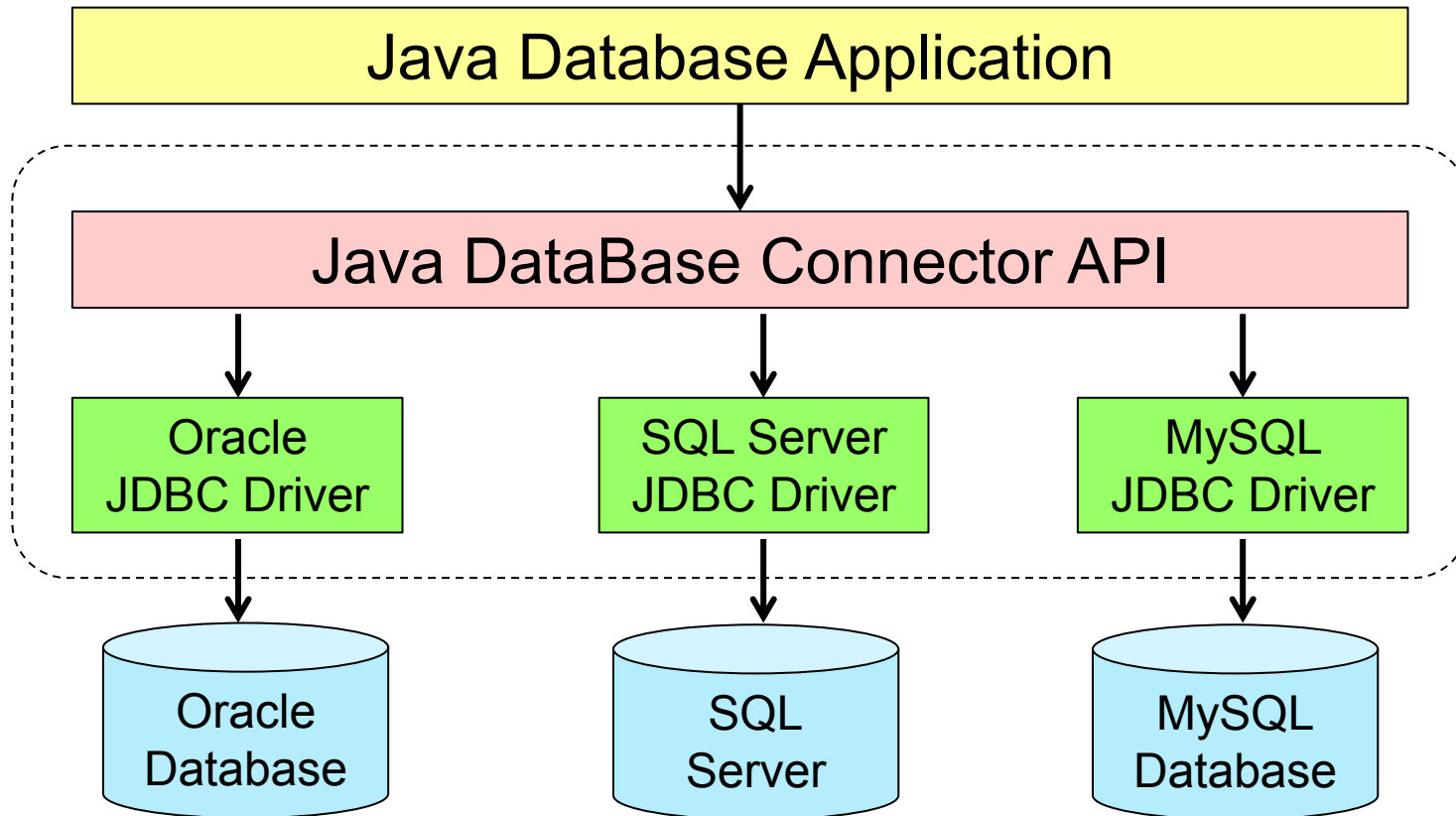
➤ Basic Syntax

- DROP TABLE [TableName] ;

➤ Example

- DROP TABLE Employee;

JDBC 簡介



Write Once **Match to All**
Database Management System

JDBC 驅動程式

- ◆ 廠商在實作JDBC驅動程式時，依方式可將驅動程式分作四種類型
 - Type 1 : JDBC-ODBC Bridge Driver
 - 利用Bridge的方式將JDBC的呼叫方式轉換為ODBC的呼叫方式
 - Type 2 : Native API Driver
 - 驅動程式將Java應用程式中JDBC呼叫轉為原生程式碼的呼叫(ex : C、C++)來與資料庫作溝通
 - Type 3 : JDBC-Middleware Driver
 - 驅動程式以特定中介伺服器(Network Server)的網路協定，來完成資料庫存取動作
 - Type 4 : Native Protocol Driver
 - 直接以資料庫的通訊協定與資料庫作溝通，而不透過橋接或中介伺服器來存取資料庫

JDBC 演進

➤ JDBC 1.0

- 提供對資料庫基本的存取功能
- 需要自行撰寫連接資料庫及關閉連線的程式碼
- 選擇和啟動JDBC驅動程式也需要手動控制

➤ JDBC 2.0

- 引入data source 概念,由data source 取得資料庫連線
- 資料庫連接池 Connection Pool

➤ JDBC 3.0

- 使用JNDI 來獲得data source , 將業務邏輯與資料庫層分離
- 資料庫連接池成為應用伺服器或Servlet 容器標準功能

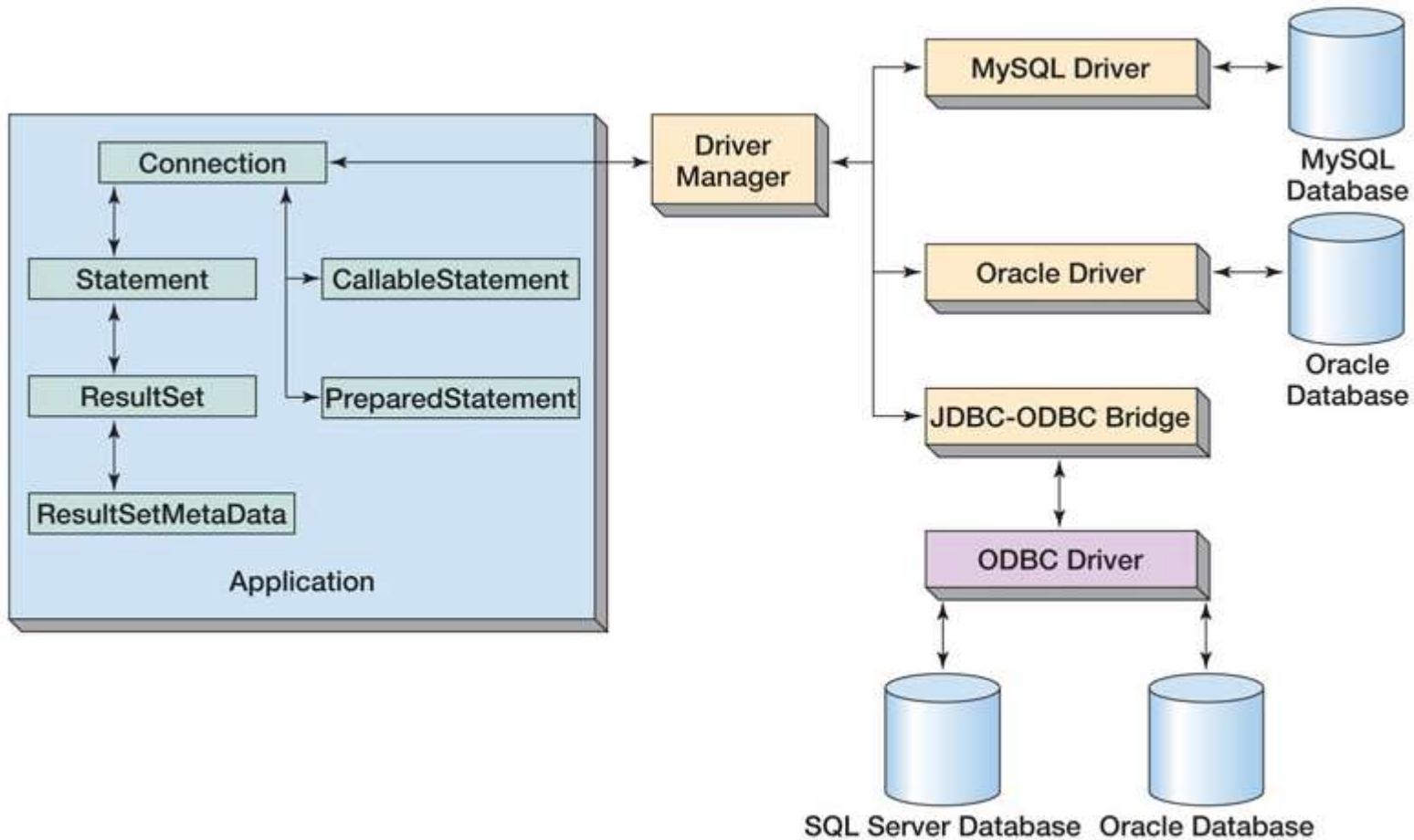
➤ JDBC 4.0

- 不必再使用Class.forName註冊JDBC驅動程式。DriverManager透過在classpath中搜尋並載入JDBC驅動程式
- 支援XML資料類型

Content

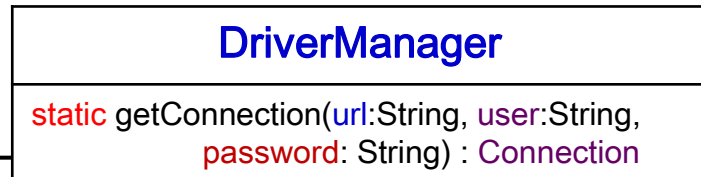
- ◆ 資料庫基本觀念
- ◆ 使用JDBC API 開發資料庫應用程式
 - JDBC使用步驟
 - ResultSet
- ◆ Transaction 交易模式
- ◆ RowSet 資料集合

JDBC 元件

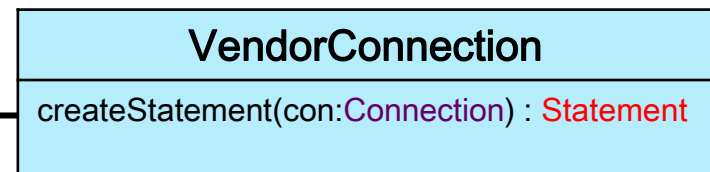
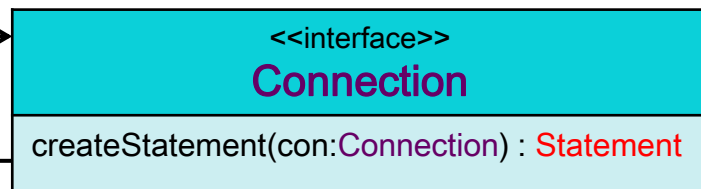


JDBC使用步驟

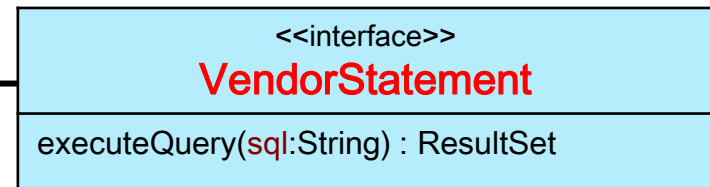
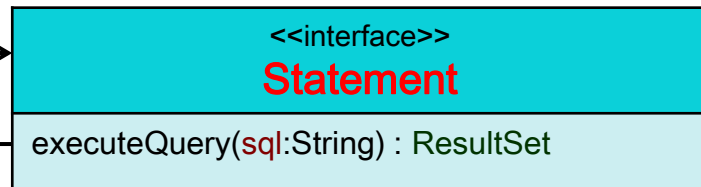
1 載入及註冊JDBC驅動程式



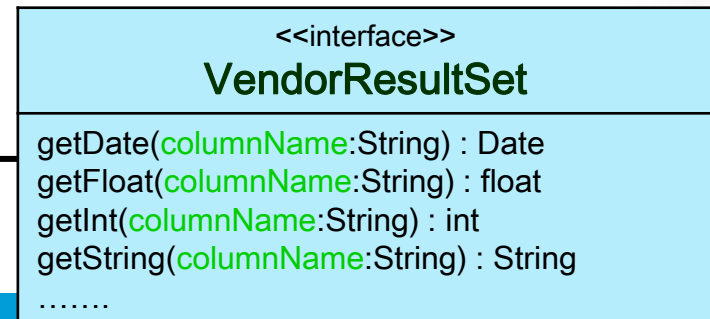
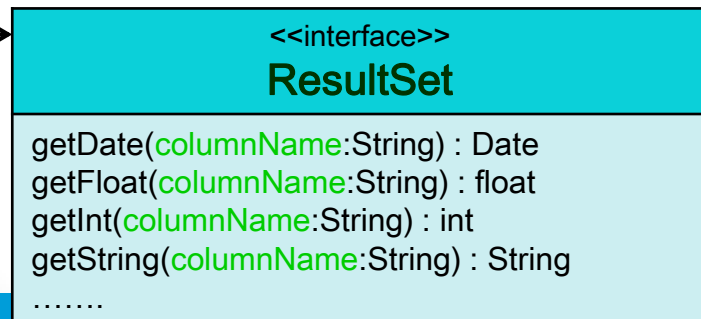
2 建立一個資料庫連結物件



3 建立一個Statement物件



4 執行並取得執行結果



載入及註冊JDBC驅動程式

◆ 連接資料庫前準備工作

- 必需要有廠商實作的JDBC驅動程式
- CLASSPATH中設定指向驅動程式JAR檔案

◆ 載入及註冊適當的JDBC驅動程式(JDBC4.0之前)

- `java.sql.DriverManager`

常用方法	傳回值	說明
<code>registerDriver(Driver driver)</code>	static void	註冊指定JDBC driver 至 DriverManager.
<code>deregisterDriver(Driver driver)</code>	static void	將指定JDBC driver 由 DriverManager中清單移除
<code>getConnection(String url)</code>	Static Connection	以指定URL 字串建立資料庫連線
<code>getConnection(String url, String user, String password)</code>	Static Connection	以指定URL 字串,帳號及密碼,建立資料庫連線

```
Driver driver = (Driver)
```

```
Class.forName(jdbcDriverClass).newInstance();
```

```
DriverManager.registerDriver(driver);
```

JDBC使用步驟

◆ 建立一個資料庫連結物件

➤ JDBC URL 語法

<Protocol>:<Subprotocol>:<DataSourceName>

➤ 建立一個Connection物件

```
String url = "jdbc:derby://localhost:1527/EmployeeDB";
```

```
String user = "pub";
```

```
String password = "tiger";
```

```
Connection con = DriverManager.getConnection(url, user, password);
```

JDBC使用步驟

◆ 建立一個SQL敘述物件

➤ createStatement()

```
Statement stmt = con.createStatement();
```

◆ 建立陳述字串

```
String sql1 = "SELECT * FROM EMPLOYEE";
```

```
String sql2 = "INSERT INTO EMPLOYEE VALUES (123, 'Sean', 'Cheng', '1974-03-21', 75000.00)";
```

```
String sql3 = "UPDATE EMPLOYEE SET SALARY=85000 WHERE ID=123";
```

```
String sql4 = "DELETE EMPLOYEE WHERE ID=123";
```

JDBC使用步驟

◆ 執行並取得執行結果

➤ executeQuery() : ResultSet

- 用於查詢 (Select) 敘述
- 傳回值為 ResultSet 物件

```
Statement stmt = con.createStatement();  
String sql = "SELECT * FROM EMPLOYEE";  
ResultSet rs = stmt.executeQuery(sql);  
while(rs.next()){  
    int id = rs.getInt(1);  
    String first = rs.getString(2);  
    String last = rs.getString(3);  
    .....  
}
```

JDBC使用步驟

◆ executeUpdate() : int

- 用於更新(Insert/Update/Delete) 敘述
- 傳回值為修改成功筆數

```
Statement stmt = con.createStatement();
```

```
String sql = "INSERT INTO EMPLOYEE VALUES (123, 'Sean', 'Cheng', '1974-03-21', 75000.00)";
```

```
int rs = stmt.executeUpdate(sql);
```

➤ execute() : boolean

- 不確定執行查詢還是更新時
- 成功傳回 true, 失敗傳回 false

java.sql.ResultSet

◆ java.sql.ResultSet物件

- 代表查詢的結果
- 具有指向當前資料行的游標
 - 游標一開始在查詢結果的第一行
 - 透過 `next()` 方法將游標移動到下一行查詢結果

rs.next()	→	110	Troy	Hammer	1965-03-31	102109.15
rs.next()	→	123	Michael	Walton	1986-08-25	93400.20
rs.next()	→	201	Thomas	Fitzpatrick	1961-09-22	75123.45
rs.next()	→	101	Abhijit	Gopali	1956-06-01	70000.00
	→	null				

java.sql.ResultSet

java.sql.ResultSet 常用方法	傳回值	說明
first()	boolean	將游標移動到 ResultSet 物件的第一行(筆)
next()	boolean	將游標由ResultSet從當前位置移動到下一行(筆)
last()	boolean	將游標移動到 ResultSet 物件的最後一行(筆)
close()	void	釋放此 ResultSet 物件的資料庫和 JDBC 資源
isFirst()	boolean	游標是否位於此 ResultSet 物件的第一行(筆)
isLast()	boolean	游標是否位於此 ResultSet 物件的最後一行(筆)
isClosed()	boolean	此 ResultSet 物件是否已關閉
getRow()	int	取得當前游標指向的資料行(筆)數
setFetchSize(int rows)	void	設定ResultSet一次取得資料的最多行(筆)數
getInt(int columnIndex)	int	取得ResultSet 當前行中第幾欄位的 int 值
getInt(String columnLabel)	int	取得ResultSet 當前行中指定欄位名稱的 int 值
getString(int columnIndex)	String	取得ResultSet 當前行中第幾欄位的字串值
getString(String columnLabel)	String	取得ResultSet 當前行中指定欄位名稱的字串值
getDouble(int columnIndex)	double	取得ResultSet 當前行中第幾欄位的double值
getDouble(String columnLabel)	double	取得ResultSet 當前行中指定欄位名稱的double值
getDate(int columnIndex)	java.sql.Date	取得ResultSet 當前行中第幾欄位的日期值
getDate(String columnLabel)	java.sql.Date	取得ResultSet 當前行中指定欄位名稱的日期值
getTimestamp(int columnIndex)	java.sql.Timestamp	取得ResultSet 當前行中第幾欄位的時間值
getTimestamp(String columnLabel)	java.sql.Timestamp	取得ResultSet 當前行中指定欄位名稱的時間值

java.sql.DatabaseMetaData

◆ java.sql.DatabaseMetaData 物件

➤ 關於資料庫的資訊

- 不同 DBMS 支持不同的功能，以不同方式實作這些功能，並使用不同的資料型別
- 由驅動程序供應商實作，讓使用者瞭解DBMS使用JDBC技術時的能力
- 由Connection物件中 `getMetaData()` 方法取得

常用方法	傳回值	說明
<code>supportsANSI92EntryLevelSQL()</code>	boolean	此資料庫是否支援 基本 ANSI92 SQL 語法
<code>supportsANSI92FullSQL()</code>	boolean	此資料庫是否支援 完整 ANSI92 SQL 語法
<code>supportsBatchUpdates()</code>	boolean	資料庫是否支持 批次更新

java.sql.SQLException

◆ java.sql.SQLException 物件

- 提供關於資料庫存取錯誤的例外
- 包含下列資訊
 - `message`：描述錯誤的字串,
 - `SQLState`：XOPEN SQLstate 或 SQL:2003
 - `vendorCode`：資料庫廠商提供之異常/錯誤代碼
 - `cause`：產生此SQLException的例外(被包覆)

常用方法	傳回值	說明
<code>getMessage()</code>	String	取得描述錯誤的字串
<code>getSQLState()</code>	String	取得SQLException 物件的SQLState
<code>getErrorCode()</code>	int	取得資料庫廠商提供之異常/錯誤代碼
<code>iterator()</code>	Iterator <Throwable>	SQLExceptions 上進行迭代的迭代器, 以取得被包覆的例外物件

JDBC資源釋放

◆ JDBC資源釋放

➤ 只關閉Connection物件

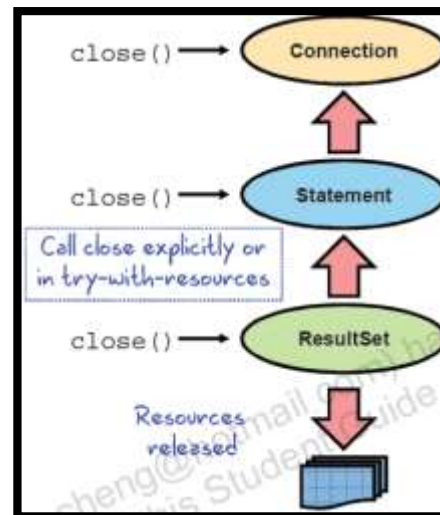
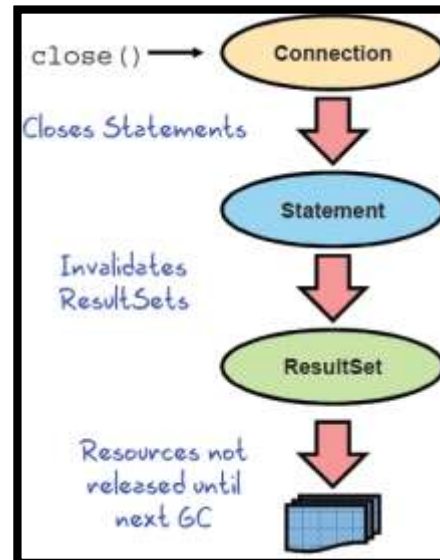
- 自動關閉Statement
- 自動作廢ResultSet
- ResultSet資源需等垃圾收集才釋放

➤ 自行關閉所有資源

- ResultSet -> Statement -> Connection
- ResultSet資源於關閉ResultSet時釋放

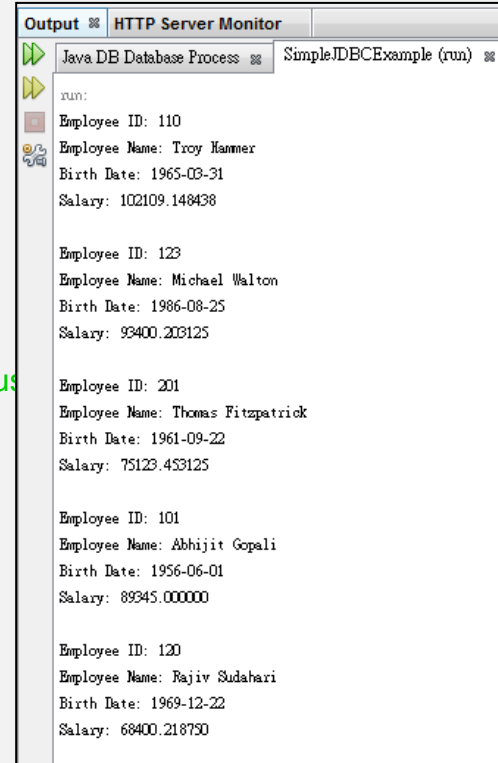
➤ 使用try with resources

- JDBC 4.1 資源均實作AutoCloseable
- 多個可關閉資源用 ';' 區隔
- 依資源建立相反順序關閉



簡易 JDBC 範例

```
01 import java.sql.*;
02 import java.util.Date;
03 public class SimpleJDBCTest {
04     public static void main(String[] args) {
05         String url = "jdbc:derby://localhost:1527/EmployeeDB";
06         String username = "pub";
07         String password = "tiger";
08         String query = "SELECT * FROM Employee";
09         try (Connection con = DriverManager.getConnection(url, us
10             Statement stmt = con.createStatement();
11             ResultSet rs = stmt.executeQuery(query)){
12             while (rs.next()) {
13                 int emplID = rs.getInt("ID");
14                 String first = rs.getString("FirstName");
15                 String last = rs.getString("LastName");
16                 Date birthDate = rs.getDate("BirthDate");
17                 float salary = rs.getFloat("Salary");
18                 System.out.printf("Employee ID: %d\nEmployee Name: %s %s\n", emplID, first, last);
19                 System.out.printf("Birth Date: %tF\nSalary: %f\n\n", birthDate, salary);
20             } // end of while
21         } catch (SQLException ex) { ..... }
22     }
23 }
```



Output HTTP Server Monitor

Java DB Database Process SimpleJDBCTest (run)

run:

Employee ID: 110
Employee Name: Troy Hammer
Birth Date: 1965-03-31
Salary: 102109.148438

Employee ID: 123
Employee Name: Michael Walton
Birth Date: 1986-08-25
Salary: 93400.203125

Employee ID: 201
Employee Name: Thomas Fitzpatrick
Birth Date: 1961-09-22
Salary: 75123.453125

Employee ID: 101
Employee Name: Abhijit Gopali
Birth Date: 1956-06-01
Salary: 89345.000000

Employee ID: 120
Employee Name: Rajiv Sudahari
Birth Date: 1969-12-22
Salary: 68400.218750

簡易 JDBC 範例

```
09      try (Connection con = DriverManager.getConnection(url, username, password);
10          Statement stmt = con.createStatement();
11          ResultSet rs = stmt.executeQuery(query)){
12          .....
21      } catch (SQLException ex) {
22          while (ex != null) {
23              System.out.println("SQLState: " + ex.getSQLState());
24              System.out.println("Error Code:" + ex.getErrorCode());
25              System.out.println("Message: " + ex.getMessage());
26              Throwable t = ex.getCause();
27              while (t != null) {
28                  System.out.println("Cause:" + t);
29                  t = t.getCause();
30              }
31              ex = ex.getNextException();
32          }
33      }
34  }
35 }
```

java.sql.ResultSetMetaData

◆ java.sql.ResultSetMetaData物件

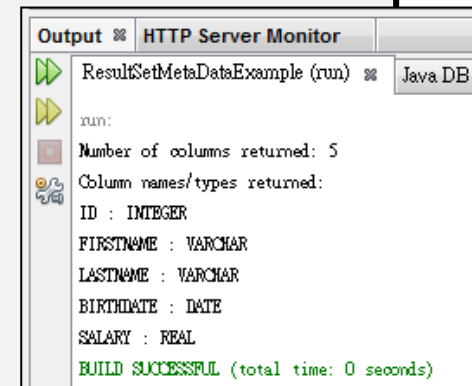
➤ 關於ResultSet中資料欄位的型別和屬性

- 由ResultSet物件中getMetaData()方法取得
- 常用方法

常用方法	傳回值	說明
getColumnCount()	int	取得ResultSet物件中的欄位數
columnName(int column)	String	取得指定第幾欄位的名稱
getColumnType(int column)	int	取得獲取指定欄位的SQL 型別
getColumnTypeName(int column)	String	取得指定第幾欄位的資料庫使用的型別名稱

ResultSetMetaData 範例

```
01 import java.sql.*;
02 public class ResultSetMetaData {
03     public static void main(String[] args) {
04         String url = "jdbc:derby://localhost:1527/EmployeeDB";
05         String username = "pub";    String password = "tiger";
06         String query = "SELECT * FROM Employee";
07         try (Connection con = DriverManager.getConnection(url, username, password);
08             Statement stmt = con.createStatement();
09             ResultSet rs = stmt.executeQuery(query)){
10             int numCols = rs.getMetaData().getColumnCount();
11             String [] colNames = new String[numCols];
12             String [] colTypes = new String[numCols];
13             for (int i= 0; i < numCols; i++) {
14                 colNames[i] = rs.getMetaData().getColumnName(i+1);
15                 colTypes[i] = rs.getMetaData().getColumnTypeName(i+1);
16             }
17             System.out.println ("Number of columns returned: " + numCols);
18             System.out.println ("Column names/types returned: ");
19             for (int i = 0; i < numCols; i++) System.out.println (colNames[i] + " : " + colTypes[i]);
20         } catch (SQLException ex) { ..... }
21     }
}
```



```
Output HTTP Server Monitor
ResultSetMetaDataExample (run) Java DB
run:
Number of columns returned: 5
Column names/types returned:
ID : INTEGER
FIRSTNAME : VARCHAR
LASTNAME : VARCHAR
BIRTHDATE : DATE
SALARY : REAL
BUILD SUCCESSFUL (total time: 0 seconds)
```

ResultSet 資料筆數

◆ 使用SQL命令

```
String sql = "SELECT COUNT(*) FROM EMPLOYEE";
```

```
ResultSet rs = stmt.executeQuery(sql);
```

```
int count = rs.next();
```

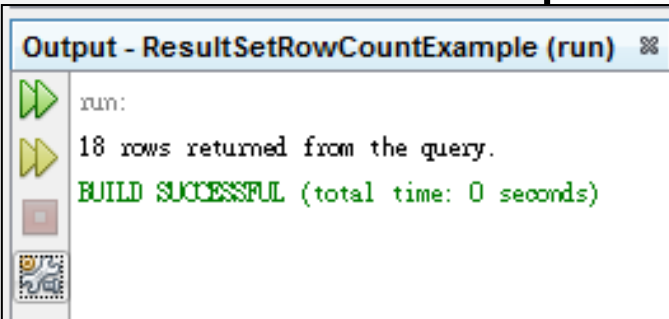
◆ 使用ResultSet 的getRow()方法

- ResultSet 需為 **TYPE_SCROLL_INSENSITIVE** 且 **CONCUR_UPDATABLE**

```
public static int rowCount(ResultSet rs) throws SQLException{
    int rowCount = 0;
    int currRow = rs.getRow(); //取得目前游標位置
    if (!rs.last()) { return -1; } //游標移至最後一行
    rowCount = rs.getRow();
    if (currRow == 0){ rs.beforeFirst(); } //游標移回第一行
    else{ rs.absolute(currRow); } //游標移至currRow位置
    return rowCount;
}
```


ResultSet 資料筆數範例

```
01 import java.sql.*;
02 public class ResultSetRowCount {
03     public static void main(String[] args) {
04         String url = "jdbc:derby://localhost:1527/EmployeeDB";
05         String username = "pub";    String password = "tiger";
06         String query = "SELECT * FROM Employee";
07         try (Connection con = DriverManager.getConnection(url, username, password);
08             Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
09             ResultSet.CONCUR_UPDATABLE);
10             ResultSet rs = stmt.executeQuery(query)){
11             System.out.println (rowCount(rs) + " rows returned from the query.");
12         } catch (SQLException ex) { ..... }
13     }
14     public static int rowCount(ResultSet rs) throws SQLException{
15         int rowCount = 0;
16         int currRow = rs.getRow(); //取得目前游標位置
17         if (!rs.last()) { return -1; } //游標移至最後一行
18         rowCount = rs.getRow();
19         if (currRow == 0){ rs.beforeFirst(); } //游標移回第一行
20         else { rs.absolute(currRow); } //游標移回初始位置
21         return rowCount;
22     }
}
```



Output - ResultSetRowCountExample (run)

```
run:
18 rows returned from the query.
BUILD SUCCESSFUL (total time: 0 seconds)
```

ResultSet

resultSetType

ResultSet.TYPE_FORWARD_ONLY	只能使用 <code>next()</code> 來逐筆取得資料
-----------------------------	----------------------------------

ResultSet.TYPE_SCROLL_INSENSITIVE	可以使用 <code>ResultSet</code> 的 <code>afterLast()</code> 、 <code>previous()</code> 、 <code>absolute()</code> 、 <code>relative()</code> 等方法來移動以取得資料。
ResultSet.TYPE_SCROLL_SENSITIVE	

ResultSet.TYPE_SCROLL_INSENSITIVE與ResultSet.TYPE_SCROLL_SENSITIVE的差別在於能否取得ResultSet改變值後的資料

resultSetConcurrency

ResultSet.CONCUR_READ_ONLY	只能讀，就不可以更新ResultSet的資料。
----------------------------	-------------------------

ResultSet.CONCUR_UPDATABLE	可以更新；就可以使用 <code>updateXXX()</code> 等方法更新ResultSet 的資料。
----------------------------	---

Prepared Statement

◆ 預編程式

- 將SQL敘述事先編譯，statement則是執行前才編譯
- 提高執行效率

◆ 使用步驟

- SQL敘述中未知值用問號表示
- Connection 物件之preparedStatment(String sql)取得
java.sql.PreparedStatement物件
- 執行前用setXXX()來設定

```
String sql = "SELECT * FROM Employee WHERE Salary > ?";
```

```
PreparedStatement pstmt = con.prepareStatement(sql);
```

```
double value = 100_000.00;
```

```
pstmt.setDouble(1, value);
```

```
ResultSet rs = pstmt.executeQuery();
```

Prepared Statement範例

```
01 import java.sql.*;
02 import java.util.Date;
03 public class PreparedStatementTest {
04     public static void main(String[] args) {
05         String url = "jdbc:derby://localhost:1527/EmployeeDB";
06         String username = "pub";
07         String password = "tiger";
08         String input = "";
09         double searchValue;
10         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
11         try (Connection con = DriverManager.getConnection(url, username, password);
12             PreparedStatement pStmt = con.prepareStatement ("SELECT * FROM Employee Where salary
13 > ?")){
14             while (true) {
15                 System.out.print("Enter salary to search for or Q to quit: ");
16                 input = in.readLine();
17                 if (input.equals("q") || input.equals("Q")) {
18                     break;
19                 }
20                 searchValue = Double.valueOf(input);
21                 pStmt.setDouble(1, searchValue);
```

Prepared Statement範例

```
21      ResultSet rs = pstmt.executeQuery();
22      while (rs.next()) {
23          int empID = rs.getInt("ID");
24          String first = rs.getString("FIRSTNAME");
25          String last = rs.getString("LASTNAME");
26          Date birth_date = rs.getDate("BIRTHDATE");
27          float salary = rs.getFloat("SALARY");
28          System.out.printf("Employee ID: %d%n", empID);
29          System.out.printf("Employee Name: %s %s%n", first, last);
30          System.out.printf("Birth Date: %tF%nSalary: %f%n%n", birthDate, salary);
31      } // end of while
32  } // end of while
33  } catch (NumberFormatException n) {
34      System.out.println("Please enter a valid number.");
35  } catch (IOException | SQLException e) {
36      System.out.println("SQLException: " + e);
37  } // end of try-with-resources
38  }
39 }
```

Callable Statement

◆ 可呼叫敘述

- 針對特定資料庫，非標準SQL敘述之預儲程式或函式
 - 實際執行在資料庫
 - 需參考資料庫工具及文件

◆ 使用步驟

- 建立 `java.sql.CallableStatement` 物件
Connection 物件之 `preparedCall(String storeProcedure)`取得
- 註冊in參數
CallableStatement 物件的 `setXXX()` 設定未知值
- 註冊out參數
`registerOutParameter(int parameterIndex, int sqlType)`方法註冊指定欄位的輸出
型態執行CallableStatement 物件的 `execute()` 方法

Callable Statement範例

```
01 import java.sql.*; import java.io.*;
02 public class CallableStatementTest {
03     public static void main(String[] args) {
04         String url = "jdbc:derby://localhost:1527/EmployeeDB";
05         String username = "pub";
06         String password = "tiger";
07         String input = "";
08         double searchValue;
09         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
10         try (Connection con = DriverManager.getConnection(url, username, password) {
11             CallableStatement cStmt = con.prepareCall("{CALL EmplAgeCount (?, ?)}");
12             while (true) {
13                 System.out.print("Enter age to search for or Q to quit: ");
14                 input = in.readLine();
15                 if (input.equals("q") || input.equals("Q")) { break; }
16                 searchValue = Double.valueOf(input);
17                 cStmt.setDouble(1, searchValue);
18                 cStmt.registerOutParameter(2, Types.INTEGER);
19                 boolean result = cStmt.execute();
20                 count = cStmt.getInt(2);
21                 System.out.println("There are " + count + " Employees over the age of " +
22 searchValue);
23             } catch (NumberFormatException n) { System.out.println("Please enter a valid
24 number.");
25             } catch (IOException | SQLException e) { System.out.println("SQLException: " + e) }
        }
    }
}
```

Content

- ◆ 資料庫基本觀念
- ◆ 使用JDBC API 開發資料庫應用程式
- ◆ Transaction 交易模式
- ◆ RowSet 資料集合

Transaction 交易模式

◆ 一系列**不可拆分**，**完整**的資料庫操作

◆ 四大特性

➤ Atomicity(原子性)

- 交易中的所有操作，要麼全部完成，要麼全部不完成
- 執行過程中發生錯誤，會被回復(Rollback)到交易開始前狀態

➤ Consistency(一致性)

- 在交易開始之前和交易結束以後，資料庫的完整性沒有被破壞

➤ Isolation(隔離性)

- 指多個交易並行執行時資料的相互關係
- 分為read uncommitted、read committed、repeatable read和Serializable等四個等級

➤ Durability(持久性)

- 交易完成後，該變更會永久改變資料的狀態







交易模式相關方法

◆ java.sql.Connection 物件提供交易模式相關方法

Connection 方法	傳回值	說明
setAutoCommit(boolean autoCommit)	void	設定此Connection的自動認可模式。 true:每個敘述執行完後自動確認, false: 手動呼叫commit()才確認
setSavepoint(String name)	Savepoint	設定一個交易的回復點，並為其指定名稱
releaseSavepoint(Savepoint savepoint)	void	刪除指定名稱的回復點
rollback()		取消在當前交易中進行的所有更改， 並釋放此 Connection 物件持有的所有資料庫鎖
rollback(Savepoint savepoint)	void	取消在當前交易中進行的所有更改至指定回復點， 並釋放此 Connection 物件持有的所有資料庫鎖
commit()	void	確認自上一次確認/回復後進行的所有變更， 並釋放此 Connection 物件持有的所有資料庫鎖

交易模式範例

```
01 import java.sql.*; import java.util.Date;
02 public class MyTransactionExample {
03     public static void main(String[] args) {
04         String url = "jdbc:derby://localhost:1527/EmployeeDB";
05         String username = "pub";
06         String password = "tiger";
07         try (Connection con = DriverManager.getConnection(url, username, password);
08             Statement stmt = con.createStatement()) {
09             con.setAutoCommit(false);
10             stmt.executeUpdate("INSERT INTO EMPLOYEE VALUES (7, 'Sam', 'Li', '1974-03-21', 75000)");
11             Savepoint sp = con.setSavepoint();
12             stmt.executeUpdate("INSERT INTO EMPLOYEE VALUES (8, 'Sue', 'Hu', '1975-11-26', 50000)");
13             con.rollback(sp);
14             stmt.executeUpdate("INSERT INTO EMPLOYEE VALUES (9, 'Ivy', 'Lin', '1988-07-24', 48000)");
15             con.commit();
16             ResultSet rs = stmt.executeQuery("SELECT * FROM Employee WHERE id < 10");
17             while (rs.next()) {
18                 int empID = rs.getInt("ID");
19                 String first = rs.getString("FirstName");
20                 String last = rs.getString("LastName");
21                 System.out.printf("Employee ID: %d\nEmployee Name: %s %s\n", empID, first, last);
22             }
23             con.commit();
24             rs.close();
25         } catch (SQLException ex) { ..... }
    }
}
```

Output	HTTP Server Monitor
 MyTransactionExample (run)	SQL Command
 run:	
 Employee ID: 7	
 Employee Name: Sam Li	
 Employee ID: 9	
 Employee Name: Ivy Lin	
BUILD SUCCESSFUL (total time: 0 seconds)	

Content

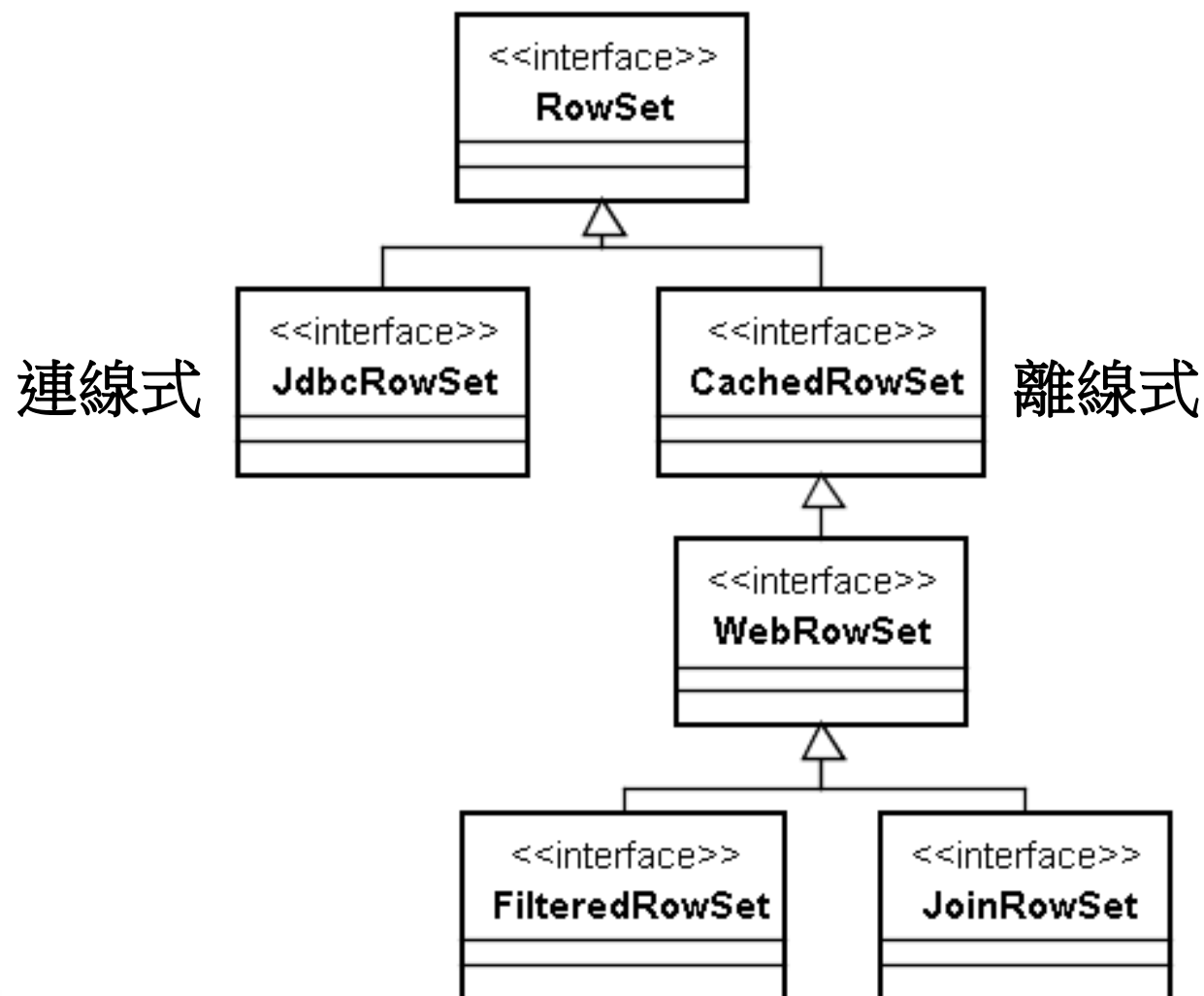
- ◆ 資料庫基本觀念
- ◆ 使用JDBC API 開發資料庫應用程式
- ◆ Transaction 交易模式
- ◆ RowSet 資料集合

RowSet 資料集合

◆ RowSet 資料集合

- 代表資料的列集合
- 減少傳統資料庫操作的程序
 - Connection -> Statement -> ResultSet
- 支援資料庫或其他具有列集合概念的資料來源
 - 試算表、網路XML檔、記憶體離線資料、合併或過濾資料
- 支援JavaBeans元件模型
 - RowSet可直接作為JavaBean使用
- javax.sql 套件及 javax.sql.rowset 套件提供相關類別
 - **javax.sql**.RowSet 介面
 - java.sql.ResultSet的子介面

RowSet 型別繼承架構



RowSet 型別種類

RowSet種類	說明
JdbcRowSet	連線式 資料集, 以JDBC 技術來維持與資料庫的連接 <ul style="list-style-type: none">• 操作JdbcRowSet期間, 會保持與資料庫的連線• 封裝取得、操作ResultSet行為, 簡化JDBC程式的撰寫• 可直接作為JavaBean使用
CachedRowSet	離線式 資料集, 記錄各行資料在快取記憶體後中斷連線 <ul style="list-style-type: none">• 操作時無需持續連接到資料庫(close()關閉)• 可再與資料來源連線進行資料同步(acceptChanges()同步)• 可直接作為JavaBean使用, 可捲動、可更新、可序列化
WebRowSet	離線式並支援 XML讀寫 的資料集
JoinRowSet	支援 資料整合 資料集 <ul style="list-style-type: none">• 將不同 RowSet 的相關資料整合到一個 JoinRowSet 物件中• 類似SQL中JOIN的功能
FilteredRowSet	支援 資料過濾 的資料集 <ul style="list-style-type: none">• 對列集合進行過濾, 類似SQL中WHERE條件式的功能

RowSet 操作流程

◆ 取得 RowSet

- JDK6之前
 - 使用Sun提供之JdbcRowSetImpl的建構子

```
JdbcRowSet jrs = new JdbcRowSetImpl();
```

- JDK7之後(RowSet 1.1) · 使用
 - javax.sql.rowset.RowSetProvider 類別
 - javax.sql.rowset.RowSetFactory 介面

```
RowSetFactory rsFactory = RowSetProvider.newFactory();
```

```
CachedRowSet crs = rsFactory.createCachedRowSet();
```


RowSet 操作流程

◆ RowSet常用方法

Connection 方法	傳回值	說明
setUrl(String url)	void	設定 RowSet 物件使用指定 URL 連接資料來源
setUsername(String name)	void	設定 RowSet 物件使用指定使用者名稱連接
setPassword(String password)	void	設定 RowSet 物件使用指定資料庫密碼連接
setCommand(String cmd)	void	設定 RowSet 物件使用指定的 SQL 敘述查詢
setInt(int paramIndex, int x)	void	以給定 int 值設定命令中的指定位置的參數
getInt(String columnLabel)	int	繼承自ResultSet,取得指定名稱欄位 int 值
setString(int paramIndex, String x)	void	以給定字串值設定命令中的指定位置的參數
getString(String columnLabel)	String	繼承自ResultSet,取得指定名稱欄位字串值
setDate(int paramIndex, Date x)	void	以給定日期值設定命令中的指定位置的參數
getDate(String columnLabel)	Date	繼承自ResultSet,取得指定名稱欄位日期值
execute()	void	執行 RowSet 物件的命令,並將結果填入

RowSet 範例

```
01 import java.sql.*; import java.sql.rowset.*; import java.util.*; import java.text.*;
02 public class SimpleJDBCRowSetExample {
03     public static void main(String[] args) {
04         String url = "jdbc:derby://localhost:1527/EmployeeDB";
05         String username = "pub";
06         String password = "tiger";
07         try (JdbcRowSet jdbcRs = RowSetProvider.newFactory().createJdbcRowSet()) {
08             jdbcRs.setUrl(url);
09             jdbcRs.setUsername(username);
10             jdbcRs.setPassword(password);
11             jdbcRs.setCommand("SELECT * FROM Employee ORDER BY lastname");
12             jdbcRs.execute();
13             while (jdbcRs.next()) {
14                 int empID = jdbcRs.getInt("ID");
15                 String first = jdbcRs.getString("FIRSTNAME");
16                 String last = jdbcRs.getString("LASTNAME");
17                 Date birthDate = jdbcRs.getDate("BIRTHDATE");
18                 float salary = jdbcRs.getFloat("SALARY");
19                 System.out.printf("Employee ID: %d\nEmployee Name: %s %s\n", empID, first, last);
20                 System.out.printf("Birth Date: %tF\nSalary: %f\n\n", birthDate, salary);
21             } // end of while
22         } catch (SQLException ex) { ..... }
23     }
24 }
```

Output - SimpleJDBCRowSetExample (run) ☒

run:

Employee ID:	121
Employee Name:	Kenny Arlington
Birth Date:	1959/10/22
Start Date:	NT\$78,405.22
Employee ID:	200
Employee Name:	Patricia Amant
Birth Date:	1970/10/31
Start Date:	NT\$79,345.00
Employee ID:	190
Employee Name:	Patrice Bergeron
Birth Date:	1970/9/18

Q & A