DBMaker

JSONCOLS Type Columns

Author: DBMaster Support Team
Production Team

Publish Date: 5 16, 2017

凌羣電腦
THE SYSCOM GROUP

目錄

1. JSON 介紹

JSON 指的是 JavaScript 物件標記法（JavaScript Object Notation），是一種羽量級的基於純文字的資料交換格式。它基於 ECMAScript 規範的一個子集，採用完全獨立於程式設計語言的文本格式來存儲和表示資料。

1.1 優點

- 羽量級的資料交換格式
- 簡潔和清晰的層次結構
- 易於人閱讀和編寫
- 同時也易於機器解析和生成

1.2 為什麼 DBMaker 要支持 JSON

JSON 是近來快速竄起的開放標準格式，方便於不同的系統上分享資料。越來越多 NoSQL 資料庫紛紛支援 JSON,都可直接用 JSON 格式來儲存資料。而像 MongoDB 也因支援 JSON 而吸引了更多人使用。所以從 DBMaker5.4 開始新的資料類型 JSONCOLS。

下面章節我們將重點講述 DBMaker 中對 JSONCOLS 類型的支援及動態欄位的使用。

2. JSONCOLS 類型和動態欄位的介紹

DBMaker支援動態欄位。動態欄位不在表定義中出現，他是從JSON字串延伸的KEY，只能在表將一個欄位聲明為JSONCOLS欄位時使用。

動態欄位DYNAMIC COLUMN是不需要在TABLE中定義就可使用，它只存儲有效的資料在JSONCOLS中，對NULL資料不進行存儲。

使用DYNAMIC COLUMN之前必須先建立JSONCOLS欄位，用於存儲動態欄位DYNAMIC COLUMN，在DBMaker中，JSONCOLS類型表示為JSONCOLS欄位。

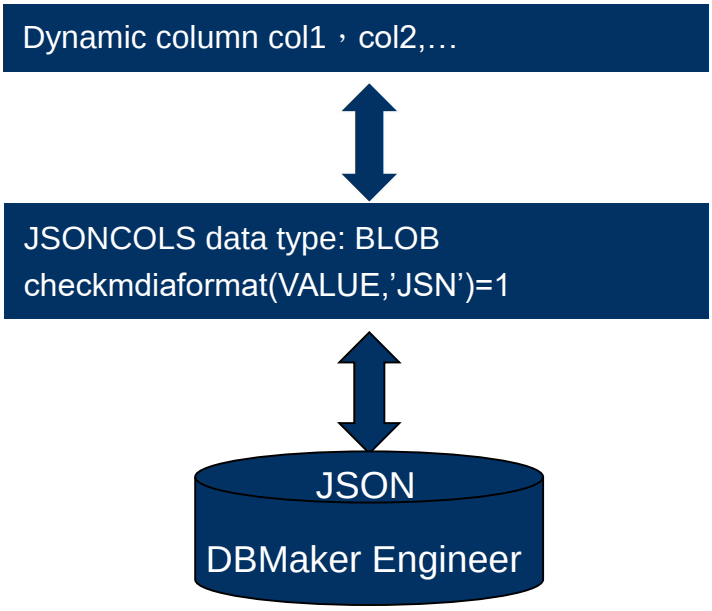

圖 1-1：JSONCOLS 和動態欄位

JSONCOLS類型用於存儲表中的所有動態欄位，可以結合動態欄位使用。那麼使用者在何時可以考慮使用 JSONCOLS類型和動態欄位(Dynamic Columns)呢？
例如：

- 目前DBMaker一個Table只能建立2000個欄位，如果用戶的表中有很多欄位且大部分欄位的值為NULL，且分別操作很麻煩時就需要考慮使用JSONCOLS類型。

- 如果使用者要設計的表格可能含有不固定欄位， 或者是由於AP方面的需求而需要新增欄位时， 除了 alter table 增加必要的固定欄位以外， 也可以考慮使用 dynamic column 的功能。
- 當記錄的屬性有數千種或者資料類型需要經常更改。

3. 使用 JSONCOLS 類型

DBMaker 5.4 開始支持新的資料類型 JSONCOLS，JSONCOLS類型是一個JSON運算式，可以將表中的所有動態欄位轉化為結構化的輸出。
JSONCOLS是所有DYNAMIC COLUMNS的集合，JSONCOLS的輸入輸出格式為JSON string：{"key1":value1,"key2":value2,"key3":value3,...}
其中JSON string中的key代表DYNAMIC COLUMN的欄位名，value代表此DYNAMIC COLUMN的資料。

3.1 定義 JSONCOLS 類型

JSONCOLS類型可以通過CREATE TABLE或ALTER TABLE語法的<JSONCOLS_type_name> JSONCOLS關鍵字來定義。

```
CREATE TABLE table-name(JSONCOLS-column-name JSONCOLS, common-column-name data-type,…)
ALTER TABLE table-name ADD [COLUMN] JSONCOLS-column-name JSONCOLS;
```

例[1-1]：建立一個帶有JSONCOLS類型的表。
在CREATE TABLE的時候，可以直接指定JSONCOLS類型，info JSONCOLS：

```
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
```

或者，建立一張普通的表，然後通過 ALTER TABLE 語法向這張表添加 JSONCOLS 類型的列，info JSONCOLS：

```
dmSQL> CREATE TABLE student(name CHAR(30));
dmSQL> ALTER TABLE student ADD COLUMN info JSONCOLS;
```

3.2 定義 JSON 運算式 (Define JSON expression)

您可以通過如下格式來定義JSONCOLS類型的JSON運算式：

```
{col1_name:col1_value,col2_name:col2_value,col3_name:col3_value....}
```

例[1-2]：JSONCOLS類型的範例。

```
{"ID":1001,"NAME":"Wang","PHONE":"029-81232689"}
```

注意：在JSONCOLS類型的JSON運算式中，可以省略含NULL值的動態欄位。

例[1-2]中，

- 學生對象包含的鍵值Key-Value ：

```
ID:1001
NAME: Wang
PHONE: 029-81232689
```

- JSON運算式：

```
{"ID":1001,"NAME":"Wang","PHONE":"029-81232689"}
```

- 資料庫存儲格式：

```
Create table stu (c1 jsoncols);
Insert into stu values('{"ID":"1001","NAME":"Wang","PHONE":"029-81232689"} ');
select c1 from stu; //result: {"NAME":"Wang","ID":"1001","PHONE":"029-81232689"}
```

3.3 使用 JSONCOLS 類型的原則(The guidelines when using JSONCOLS TYPE)

使用JSONCOLS類型時，您需要考慮以下幾點原則：

- 不能更改JSONCOLS類型。如果要更改，則需要刪除和重建JSONCOLS類型。
- 一個表中只允許一個JSONCOLS類型存在。
- 不能在JSONCOLS類型中定義範例或預設值。

3.4 JSONCOLS 類型執行許可權及特點(the permissions and feature of JSONCOLS TYPE)

JSONCOLS類型的安全模組(Security Model)類似於普通欄位，在JSONCOLS欄位上執行DML(SELECT、INSERT、 UPDATE和DELETE)語句時，要求您具備和JSONCOLS欄位相應的許可權。

當對JSONCOLS欄位的資料執行操作時，需要使用單個動態欄位的名稱，或者參照JSONCOLS類型的名稱以及使用JSONCOLS類型的JSON運算式來指定JSONCOLS類型的值。

動態欄位能夠以任何順序顯示在JSONCOLS欄位中。因為DBMaker是以內部方式來存儲JSONCOLS欄位的資料，因此在查詢時，動態欄位的顯示順序可能與其插入順序不同。

3.5 在 JSONCOLS 欄位上執行 DML

这一部分主要讲述在JSONCOLS欄位進行INSERT、SELECT、UPDATE操作。

3.5.1 使用 JSONCOLS 類型的名稱插入(INSERT)資料

例[1-3]：使用JSONCOLS類型的名稱將資料插入表student。

```
dmSQL> create table student ( NAME CHAR(30) , INFO JSONCOLS );
dmSQL> Insert into student values('Wang','{"ID":"1001","PHONE":"029-81232689"} ');
```

例[1-4]：使用“SELECT *”命令查詢表student。

```
dmSQL> set blobwidth 100;
dmSQL> SELECT * FROM student;
或
dmSQL> SELECT name, info FROM student;
         NAME                    INFO
============           ====== =========== ===========
Wang                   {"ID":"1001","PHONE":"029-81232689"}
```

3.5.2 使用 JSONCOLS 類型的名稱更新(UPDATE)資料

例[1-5]：使用JSONCOLS類型的名稱更新表student的資料。

```
dmSQL> UPDATE student SET info = '{"ID":1002, "PHONE":"029-88899999"}' WHERE name='Wang';
1 rows updated
```

```
dmSQL> SELECT name, info FROM student;
         NAME                     INFO
============           ====== =========== ===========
Wang                   {"ID":1002,"PHONE":"029-88899999"}
```

3.5.3 JSONCOLS 類型欄位元上建立文本索引（TEXT INDEX）

定義了JSONCOLS欄位之後即可將其作為普通欄位使用。但是，由於JSONCOLS欄位是從LONG VARBINARY延伸的，且在DBMaker中使用者不能在大物件上建立索引，因此也不能在已定義的JSONCOLS欄位元欄位上建立索引，但是可以建立全文索引。

全文索引可以快速訪問表中包含一個或多個單詞或短語欄位的資料行。全文索引包含了從相應欄位搜索得到的所有文本的描述，資料進行編碼和重組，使讀取效率比直接從表中要高。一旦使用者為表建立了全文索引，其操作就是透明的。DBMS使用索引來提高文本的查詢性能。

不允許在JSONCOLS欄位上建立索引：

```
dmSQL> CREATE INDEX idx1 ON student(info);
ERROR (6532): [DBMaster] cannot create index on large object column : INFO
```

但是可以在JSONCOLS欄位上建立全文索引：

例[1-6]：在JSONCOLS類型欄位info上建立文本索引。

```
dmSQL> CREATE TEXT INDEX idx_stu ON student(INFO);
```

3.5.4 JSONCOLS 欄位上建立視圖

視圖對資料庫的查詢是非常有效的。例如，對於一個相當複雜的查詢指令，您可以把它定義成視圖。如此一來，您就可以重複使用這個視圖，而不用每次都重新輸入這個複雜的查詢指令。由於視圖可以用它的定義來限制使用者存取的表字段和資料，所以，定義視圖可以加強資料庫的安全性。

例[1-7]：在JSONCOLS類型欄位info上建立一個視圖。

```
dmSQL> CREATE VIEW v1 AS SELECT info FROM student;
```

4. 使用動態欄位(Dynamic Columns)

動態欄位不在表定義中出現，它是從JSON字串延伸的KEY，只能在表將一個欄位聲明為JSONCOLS欄位時使用。

動態欄位用於存儲半結構化的資料、記錄的屬性有數千種或資料類型經常更改的資料，可以結合JSONCOLS類型使用。

動態欄位的安全模組（Security Model）類似於普通欄位，其特徵如下所示：

- 可以在動態欄位元上建立索引
- 動態欄位僅支援更改資料類型
- 動態欄位支援如下資料類型： SMALLINT、 INT、FLOAT、 DOUBLE、DATE、TIME、TIMESTAMP、CHAR、VARCHAR、 NCHAR、NVARCHAR。
- 動態欄位的預設資料類型是varchar(256)
- 動態欄位必須為可空的值
- 動態欄位不能有預設值
- 動態欄位不能有欄位範例
- 動態欄位不能用於存儲命令
- 動態欄位不能用於存儲過程
- 動態欄位不能用於觸發器
- 動態欄位建立後無需定義即可直接使用

當對動態欄位的資料執行操作時，需要使用單個動態欄位的名稱，或者參照JSONCOLS類型的名稱以及使用JSONCOLS類型的JSON運算式來指定JSONCOLS類型的值。動態欄位能夠以任何順序顯示在JSONCOLS欄位中。

4.1 使用動態欄位的欄位名執行更新或刪除操作

4.1.1 使用動態欄位的欄位名執行更新操作

例[1-8]：使用動態欄位的欄位名name修改score=100。

```
dmSQL> UPDATE student SET score=100 WHERE name='Li';
1 rows updated

dmSQL> SELECT name, info FROM student;
         NAME                    INFO
============         ====== =========== ===========
Li                     {"PHONE":"029-86622126","SCORE":100,"ID":"1003"}
Lee                    {"ID":"1005","PHONE":"029-86699821"}
```

4.1.2 使用動態欄位的欄位名執行刪除操作

例[1-9]：使用動態欄位的欄位名刪除表student中ID='1002'的資料。

```
dmSQL> DELETE FROM student  WHERE ID ='1002';
1 rows deleted

dmSQL> SELECT name, info FROM student;
         NAME                    INFO
============         ====== =========== ===========
Li                     {"ID":"1003","PHONE":"029-86622126"}
Lee                    {"PHONE":"029-86699821","ID":"1005"}

2 rows selected
```

4.2 ALTER 命令

動態欄位的預設資料類型是varchar(256)，在預設資料類型不適用時，您可以使用ALTER TABLE ADD DYNAMIC COLUMN命令來將其更改為其他資料類型。此外，當該動態欄位被插入到表時，也可以使用該命令聲明動態欄位的資料類型。如果您想要將已聲明的資料類型更改為另一種資料類型，則需使用ALTER TABLE MODIFY DYNAMIC COLUMN命令；如果您想刪除動態欄位的描述資訊，則可以使用ALTER TABLE DROP DYNAMIC COLUMN命令。另外，如果在插入資料時未執行ALTER TABLE ADD DYNAMIC COLUMN命令，但是卻使用該命令聲明了動態欄位的資料類型，之後又無法將之前插入的資料轉換為已聲明的資料類型時，在查詢時該資料會表示為NULL，此時也不會報錯。

4.2.1 ALTER TABLE ADD DYNAMIC COLUMN 命令

該命令用來更改動態欄位的資料類型。此外，當該動態欄位被插入到表時，也可以使用該命令聲明動態欄位的資料類型。

例[1-10]：向表中添加動態欄位成绩score，类型为DOUBLE。

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN score DOUBLE;
```

例[1-11]：向表中添加動態欄位入学时间c_date，类型为DATE。

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN c_date DATE;
```

例[1-12]：向表中添加動態欄位s_age，类型为SMALLINT。

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN s_age SMALLINT;
```

例[1-13]：向表中添加動態欄学生家庭地址s_adr，类型为VARCHAR。

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN s_adr VARCHAR;
```

4.2.2 ALTER TABLE MODIFY DYNAMIC COLUMN 命令

該命令用來將將已聲明的資料類型更改為另一種資料類型。

例[1-14]：修改動態欄位score的資料類型。

```
dmSQL> ALTER TABLE student MODIFY DYNAMIC COLUMN score TYPE TO INT;
```

4.2.3 ALTER TABLE DROP DYNAMIC COLUMN 命令

該命令用來刪除動態欄位的描述資訊。

例[1-15]：刪除動態欄位score 的描述資訊。

```
dmSQL> ALTER TABLE student DROP DYNAMIC COLUMN score;
```

4.2.4 使用隱式資料轉換插入(INSERT)或更新(UPDATE)資料

如果没有设置動態欄位的描述类型，默认为VARCHAR(256),在这种情况下，
當您使用參數將資料插入動態欄位或更新動態欄位的資料時，DBMaker僅支援插入VARCHAR類型的資料，此時，您可以使用隱式資料轉換功能，插入其他類型的資料。DBMaker支援開啟隱式資料轉換功能。

- 開啟隱式資料轉換功能：SET itcmd ON;
- 關閉隱式資料轉換功能：SET itcmd OFF;(默認關閉)

例[1-16]：通過開啟隱式轉換功能向JSONCOLS類型的欄位插入數值1006

```
dmSQL> insert into student(name,id,phone) values(?,?,?);

dmSQL/Val> 'Li','1003','029-86622126';
1 rows inserted

dmSQL/Val> 'Lee',1006,'029-86699821';
ERROR (9629): value list syntax error
dmSQL/Val> end;

dmSQL> SET itcmd ON;

dmSQL> insert into student(name,id,phone) values(?,?,?);

dmSQL/Val> 'Lee', 1006,'029-86699821';
1 rows inserted
```

```
dmSQL> SELECT name, info FROM student;

         NAME                    INFO
=============         ====== =========== ===========
Wang                    {"ID":1002,"PHONE":"029-88899999"}
Li                      {"ID":"1003","PHONE":"029-86622126"}
Lee                     {"ID":"1006","PHONE":"029-86699821"}

3 rows selected
```

例[1-17]：通過開啟隱式轉換功能更新JSONCOLS類型的欄位的值

```
dmSQL> SET itcmd OFF;


dmSQL> update student set id=1005 where name='Lee';
ERROR (6150): [DBMaster] the insert/update value type is incompatible with column data
type or compare/operand value is incompatible with column data type in expression/predicate
```

```
dmSQL> SET itcmd ON;

dmSQL> update student set id=1005 where name='Lee';
1 rows updated
```

```
dmSQL> SELECT name, info FROM student where name='Lee';

         NAME                    INFO
============         ====== =========== ===========
Lee                    {"PHONE":"029-86699821","ID":"1005"}
1 rows selected
```

4.3 動態欄位上建立索引 (Create index)

使用索引可以對資料行作快速的隨機訪問。通過在表上建立索引來加快資料的查詢。例如：當用戶執行一個“SELECT NAME FROM student WHERE ID = ‘1001’;”查詢語句時，如果在欄位ID上建有索引，那麼讀取資料的速度就會快很多。

例[1-18]：在動態欄位ID上建立索引。

```
dmSQL>CREATE INDEX idx1 ON student(ID);
```

5. 動態欄位在應用程式中的使用 (Using DYNAMIC COLUMN in application)

JSONCOLS類型和動態欄位可以像普通欄位一樣在AP中使用，另外對於那些資料類型經常更改的資料，AP頻繁需要新增欄位等情況有更好的靈活性。 或是使用者也可以將原本使用的 JSON Object 轉為JSON String 後，新增到 JSONCOLS 欄位。

下面我們簡單列舉兩個在AP（JAVA&C#）中應用的例子，分別列舉了通過動態欄位插入值，使用參數將資料插入動態欄位，使用JSON String插入值，或是使用者也可以將原本在AP中使用的JSON Object 轉為JSON String 後，直接新增 or 插入JSON String 資料到 JSONCOLS 欄位，然后再使用SQL語法來做查詢或是更新。

例[1-19]：通過JAVA程式向動態欄位插入值。

```
/*****************************************************************************
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
*****************************************************************************/
import java.sql.*;
import net.sf.json.*;

public class JsoncolsTypeColumns {
  public static void main(String[] args){
      try{
         Class.forName("dbmaker.sql.JdbcOdbcDriver");
         Connection conn =
DriverManager.getConnection("jdbc:dbmaker:dbsample5","sysadm","");
         Statement stmt = conn.createStatement();
/*******************************************************************************
第一種：過動態欄位插入值
*******************************************************************************/
         stmt.executeUpdate("INSERT INTO student (name,desk id,birthday,score) VALUES
('ququ','6','2017-02-12','23')");
         ResultSet rs = stmt.executeQuery("select * from student where name='ququ'");
         while(rs.next()) {
         System.out.println(rs.getString(1).trim() + "," + rs.getString(2));
         }
```

```
        rs.close();
/*****************************************************************************
第二種：使用參數將資料插入動態欄位
*****************************************************************************/
        CallableStatement cstmt = conn.prepareCall("INSERT INTO student
(name,desk id,birthday,score) VALUES ('lulu','9',?,?)");
        cstmt.setString(1,"2016-02-14");
        cstmt.setString(2,"22");
        cstmt.execute();
        rs = stmt.executeQuery("select * from student where name='lulu'");
        while(rs.next()) {
        System.out.println(rs.getString(1).trim() + "," + rs.getString(2));
        }
        rs.close();
/*****************************************************************************
第三種：使用JSON String插入值
*****************************************************************************/
        cstmt = conn.prepareCall("INSERT INTO student (name,info) VALUES ('snow',?)");

cstmt.setString(1,"{\"desk id\":4,\"birthday\":\"1987-03-03\",\"score\":95}");
        cstmt.execute();
        rs = stmt.executeQuery("select name,desk id,birthday,score from student where
name='snow'");
        while(rs.next()) {
         System.out.println(rs.getString(1).trim() + "," + rs.getString(2));
        }
        rs.close();
/*****************************************************************************
第四種：將JSON Object 轉為JSON String 後，直接插入JSON String 資料到 JSONCOLS 欄位
*****************************************************************************/
/*建立JSON對象* /
        cstmt = conn.prepareCall("INSERT INTO student (name,info) VALUES
('tester1',?)");
        JSONArray jsonArray = new JSONArray();
        JSONObject json = new JSONObject();
        json.put("desk id", 12);
        json.put("birthday", "2015-01-15");
        json.put("score", 89);
        jsonArray.add(json);
        json.put("desk id", 22);
        json.put("birthday", "2003-12-15");
        json.put("score", 67);
        jsonArray.add(json);
/*通過jsonArray.toString()將JSON物件轉化為字串，然後插入資料庫*/
        cstmt.setString(1, jsonArray.toString());
        cstmt.execute();
/*查詢新插入的name='tester1'的這筆資料並列印出來*/
        rs = stmt.executeQuery("select * from student where name='tester1'");
        while(rs.next()) {
/*通過JSONArray.fromObject把從資料庫中取出的JSON String方便的轉化為JSON Object，例如本例中，將查詢出來的字串轉
為JSON Object，然後列印出來 */
```

```
         JSONArray array = JSONArray.fromObject(rs.getString(2));
         System.out.println(rs.getString(1).trim() + "," +
array.getJSONObject(1).getString("desk id"));
        }
        rs.close();
        stmt.close();
        conn.close();
      }catch(Exception ex){
         ex.printStackTrace();
      }
     }
}
```

例[1-20]：通過C#程式向動態欄位插入值。

```
/*****************************************************************************************
 dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
*****************************************************************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.Odbc;
/* JsonConvert.DeserializeObject依賴Newtonsoft.Json*/
using Newtonsoft.Json;
/* 使用JObject Class需要添加引用 Newtonsoft.Json.Linq namespace*/
using Newtonsoft.Json.Linq;

namespace JsoncolsType
{
   class Program
   {
      public static void Main()
         {
          OdbcConnection myCN = null;
          try{
            string myCNString = "dsn=dbsample5;uid=sysadm;pwd=;";
            myCN = new OdbcConnection(myCNString);
            myCN.Open();
/*****************************************************************************************
第一種：通過動態欄位插入
*****************************************************************************************/
            string myCMString = "INSERT INTO student (name,desk id,birthday,score) VALUES
('ququ','6','2017-02-12','23')";
            OdbcCommand myCM = new OdbcCommand(myCMString, myCN);
            myCM.ExecuteNonQuery();
            myCM = new OdbcCommand("select * from student where name='ququ'", myCN);
            OdbcDataReader myDR = myCM.ExecuteReader();
```

```
            while(myDR.Read()){
                  Console.WriteLine(myDR.GetString(0).Trim() + "," + myDR.GetString(1));
            }
/*****************************************************************************
第二種：使用參數將資料插入動態欄位
*****************************************************************************/
            myCMString = "INSERT INTO student (name,desk id,birthday,score) VALUES
('lulu','9',?,?)";
            myCM = new OdbcCommand(myCMString, myCN);
            myCM.Parameters.Add("@Char", OdbcType.Char,10).Value = "2006-05-02";
            myCM.Parameters.Add("@varchar", OdbcType.VarChar,6).Value = "123456";
            myCM.ExecuteNonQuery();
            myCM = new OdbcCommand("select * from student where name='lulu'", myCN);
            myDR = myCM.ExecuteReader();
            while (myDR.Read())
            {
               Console.WriteLine(myDR.GetString(0).Trim() + "," + myDR.GetString(1));
            }
/*****************************************************************************
第三種：使用JSON String插入值
*****************************************************************************/
            myCMString = "INSERT INTO student (name,info) VALUES ('snow',?)";
            myCM = new OdbcCommand(myCMString, myCN);
            myCM.Parameters.Add("@varchar", OdbcType.VarChar, 200).Value =
"{\"desk id\":4,\"birthday\":\"1987-03-03\",\"score\":95}";
            myCM.ExecuteNonQuery();
            myCM = new OdbcCommand("select name,desk id,birthday,score from student where
name='snow'", myCN);
            myDR = myCM.ExecuteReader();
            while (myDR.Read())
            {
               Console.WriteLine(myDR.GetString(0).Trim() + "," + myDR.GetString(1));
            }
/*****************************************************************************
第四種：將JSON Object 轉為JSON String 後，直接插入JSON String 資料到 JSONCOLS 欄位
*****************************************************************************/
/*创建JSON对象*/
            myCMString = "INSERT INTO student (name,info) VALUES ('tester2',?)";
            JArray array = new JArray();
            JObject jsonObject = new JObject();
            jsonObject.Add(new JProperty("desk id", "11"));
            jsonObject.Add(new JProperty("birthday", "1986-11-26"));
            jsonObject.Add(new JProperty("score", "99"));
            array.Add(jsonObject);
            jsonObject = new JObject();
            jsonObject.Add(new JProperty("desk id", "22"));
            jsonObject.Add(new JProperty("birthday", "1934-12-05"));
            jsonObject.Add(new JProperty("score", "35"));
            array.Add(jsonObject);
            myCM = new OdbcCommand(myCMString, myCN);
```

```
/*通過array.ToString()將JSON物件轉化為字串，然後插入資料庫*/
            myCM.Parameters.Add("@varchar", OdbcType.VarChar, 200).Value =
array.ToString();
            myCM.ExecuteNonQuery();
/*查詢新插入的name=’tester2’的這筆資料並列印出來*/
            myCM = new OdbcCommand("select * from student where name='tester2'", myCN);
            myDR = myCM.ExecuteReader();
            while (myDR.Read())
            {
/*通過JsonConvert.DeserializeObject把從資料庫中取出的JSON String轉化為JSON Object，例如本例中，將查詢出來的字串轉為
JSON Object，然後列印出來 */
               JArray array1 = (JArray)JsonConvert.DeserializeObject(myDR.GetString(1));
               Console.WriteLine(myDR.GetString(0).Trim() + "," +
array1[0]["desk id"].ToString());
            }
            myCN.Close();
         }catch (Exception ex){
            Console.WriteLine(ex.Message);
         }
         finally{
            if (myCN != null) myCN.Close();
         }
         Console.WriteLine("press ENTER to exit...");
         Console.Read();
      }
   }
}
```