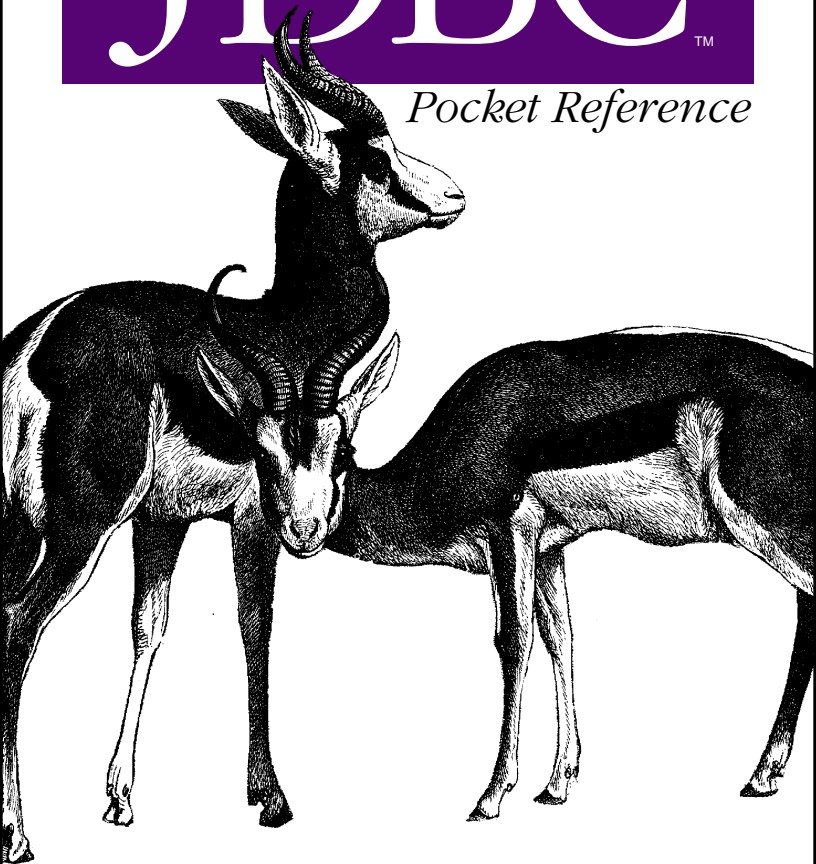


A Quick Guide for Programmers

JDBC™

Pocket Reference



O'REILLY®

Donald Bales

JDBC™
Pocket Reference

Donald Bales

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

User-Defined Data Types

User-defined data types (UDTs) bring the world of object-orientation to relational databases. Using UDTs you can create an object-relational database, or objectbase, in which an object model can be directly implemented in the persistence layer. To store and retrieve UDTs, you can either manipulate them as SQL STRUCTs and ARRAYs (which I will not show here), or you can define Java classes that are mapped to the SQL UDTs, which you then use to materialize the UDTs in your Java program.

Creating a Java Class for a UDT

Java classes that will be used to materialize a UDT need to implement the `SQLData` interface. Most databases that support UDTs also provide a tool to generate Java classes that implement `SQLData`. Implementing the `SQLData` interface consists of coding three methods. The first, `getSQLTypeName()`, needs to return the fully qualified name of the UDT as it exists in the database. The second, `readSQL()`, reads your class attributes from an `SQLInput` stream in the order they exist in the UDT. The third, `writeSQL()`, writes the values of your class attributes onto the `SQLOutput` stream in the order they exit in the UDT. Here's an example of a hand-coded class that implements the `SQLData` interface for an object table PERSON:

```
import java.io.*;
import java.sql.*;

/**
 * A mirror class to hold a copy of SCOTT.PERSON_TYPE
 */
public class Person
    implements SQLData, Serializable {

    private int         person_id;
    private String     last_name;
    private String     first_name;
```

```

private java.sql.Date birth_date;
private String      gender;

public Person() { }

// SQLData interface
public String getSQLTypeName()
    throws SQLException {
    return "SCOTT.PERSON_TYPE";
}
public void readSQL(SQLInput stream, String type)
    throws SQLException {
    person_id = stream.readInt();
    last_name = stream.readString();
    first_name = stream.readString();
    birth_date = stream.readDate();
    gender     = stream.readString();
}
public void writeSQL(SQLOutput stream)
    throws SQLException {
    stream.writeInt(person_id);
    stream.writeString(last_name);
    stream.writeString(first_name);
    stream.writeDate(birth_date);
    stream.writeString(gender);
}
// Accessors
public int getPersonId() {
    return person_id;
}
public String getLastName() {
    return last_name;
}
public String getFirstName() {
    return first_name;
}
public java.sql.Date getBirthDate() {
    return birth_date;
}
public String getGender() {
    return gender;
}
// Mutators
public void setPersonId(int person_id) {
    this.person_id = person_id;
}

```

```

public void setLastName(String last_name) {
    this.last_name = last_name;
}
public void setFirstName(String first_name) {
    this.first_name = first_name;
}
public void setBirthDate(
    java.sql.Date birth_date) {
    this.birth_date = birth_date;
}
public void setGender(String gender) {
    this.gender = gender;
}
}

```

Updating a Type Map

Once you have a UDT and Java class to represent it, you must update the `Connection` or `PreparedStatement`'s type map to let the driver know which class to materialize when you retrieve a UDT from a database, and what kind of UDT to materialize when you store the Java class in a database. Updating a type map consists of retrieving the current type map and adding the new mapping as in the following example (assuming a `Connection`, `conn`, already exists):

```

try {
    java.util.Map map = conn.getTypeMap();
    map.put(
        "SCOTT.PERSON_TYPE",
        Class.forName("Person"));
}
catch (ClassNotFoundException e) {
    ...
}
catch (SQLException e) {
    ...
}

```

Inserting a UDT

To insert a UDT, you start out by creating a new instance of the class that represents it, setting attributes as necessary.

You then use `PreparedStatement`'s `setObject()`, passing the column index and new class instance. Since the type map has been updated, the JDBC driver knows which new instance of a database UDT to create. Here's an example that inserts a `PERSON_TYPE` instance into object table `PERSON_OBJ_TAB` (assuming a `Connection`, `conn`, already exists):

```
PreparedStatement pstmt = null;
try {
    // Create a new instance.
    Person person = new Person();
    person.setPersonId(1);
    person.setLastName("DOE");
    person.setFirstName("JOHN");
    person.setBirthDate(
        java.sql.Date.valueOf("1980-02-28"));
    person.setGender("M");
    // Insert it into an object table.
    pstmt = conn.prepareStatement(
        "insert into PERSON_OBJ_TAB values ( ? )");
    pstmt.setObject(1, person);
    int rows = pstmt.executeUpdate();
    pstmt.close();
    pstmt = null;
    conn.commit();
}
catch (SQLException e) {
    ...
}
finally {
    if (stmt != null)
        try { stmt.close(); } catch (Exception i) { }
}
```

Selecting a UDT

To select a UDT, you start by formulating an SQL `SELECT` statement that returns a UDT as an object. Execute the `SELECT` statement, and then use the `Statement`, `PreparedStatement`, or `CallableStatement`'s `getObject()` method, casting the object it returns to the appropriate Java class type. The driver uses its updated type map to determine which Java class to materialize for the selected UDT.

Here's an example that selects a UDT PERSON_TYPE from object table PERSON_OBJ_TAB (assuming a Connection, conn, already exists):

```
ResultSet rset = null;
Statement stmt = null;
try {
    stmt = conn.createStatement();
    // value() is Oracle's materialization function
    rset = stmt.executeQuery(
        "select value(p) from PERSON_OBJ_TAB p");
    while (rset.next()) {
        // Cast the object
        person = (Person)rset.getObject(1);
        ...
    }
    rset.close();
    rset = null;
    stmt.close();
    stmt = null;
}
catch (SQLException e) {
    ...
}
finally {
    if (rset != null)
        try { rset.close(); } catch (Exception i) { }
    if (stmt != null)
        try { stmt.close(); } catch (Exception i) { }
}
```