

User name:

Book: SQL in a Nutshell, 2nd Edition

No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Chapter 5. Database Programming

While SQL plays an important role in standardizing the communication with different RDBMSs, a missing piece still remains for those who want to write database software applications. The missing piece is the database programming Application Programming Interface (API) used to transport SQL statements to and process their results from an RDBMS. While all the database platforms discussed in this book provide their own proprietary interfaces for database application developers, this chapter focuses on two commonly used APIs that provide a consistent interface across database platforms. Specifically, this chapter introduces you to:

ADO.NET

ADO.NET is Microsoft's high-level database programming API on the .NET platform. The ADO.NET API is a collection of .NET interfaces that are accessible from any of the .NET languages. The primary benefits of ADO.NET are ease of use, portability within the .NET platform, XML integration, and access to data sources other than relational databases. The ADO.NET examples covered in this book are written in C#; however, ADO.NET is also available from Visual Basic and the other .NET languages.

JDBC

JDBC, or *Java Database Connectivity*, was developed primarily by Sun Microsystems to be the primary database programming API for the Java language. JDBC is the most popular database programming API for the Java language and offers operating-system portability, reasonable performance for most applications, and a well-documented interface. In addition, drivers for most database systems are typically free. This chapter covers JDBC Version 3.0. For additional information, please browse <http://java.sun.com/jdbc>.

As a quick desk reference, this chapter won't provide all the information needed to develop a large enterprise database application. However, we give you enough to get started by covering components that are common to all database applications, both large and small.

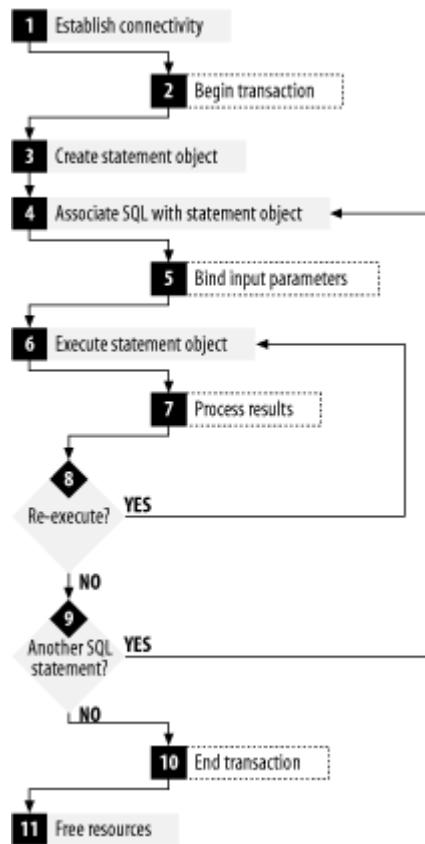
5.1. Database Programming Overview

Developing successful database applications, large and small, involves many steps. Careful thought must be given to application architecture, and especially to the following issues:

- How to map application data, which is typically object-oriented, to a relational database
- How to handle errors gracefully
- How to maximize performance and scalability

A typical database application will require many different SQL statements. The management of so many statements is simplified by the fact that all SQL statements will follow roughly the same pattern of execution within an application. [Figure 5-1](#) is a state diagram showing how SQL statements are prepared, executed, and then processed by a database application when interacting with a relational database system. The state diagram has been broken down into eleven steps, four of which are optional (and are indented in the diagram).

Figure 5-1. Statement execution state diagram



Following are detailed descriptions of each step shown in [Figure 5-1](#):

1. *Establish connectivity*: Establishing connectivity is the first step in every successful database application. It is in this step that the client, or database application, makes a physical connection to the database that will be used to transmit the SQL statements to the database and the results back to the client. The actual physical connection could be over a LAN, WAN, or even a simple logical connection back to the client in cases where the database application and server are running on the same machine. For more information on how to establish connectivity, please see the section [Section 5.2](#) later in this chapter.
2. *Begin transaction (optional)*: A transaction may be begun so that the database changes may be rolled back on failure or committed on success. For more information on transaction control from database programming APIs, please see [Section 5.4](#) later in this chapter.
3. *Create statement object*: Most modern database programming APIs are object-oriented and therefore use an object to represent a SQL statement. There will typically be one statement object per SQL statement executed within the application. The statement object holds the state information required to execute the SQL statement, such as the SQL statement itself and input parameters when they exist.
4. *Associate SQL with statement object*: After the statement object has been created, it needs to have a SQL statement assigned to it. Once this is done, the statement object may be executed.
5. *Bind input parameters (optional)*: While not part of the ANSI SQL standard, the ability to bind parameters to "placeholders" within a SQL statement is supported by all the database platforms covered in this book. If the SQL statement has placeholders for input parameters, the statement object will need to have a program variable associated with each input parameter. If the SQL statement contains no input parameters, then this step may be skipped. Input parameters are useful for optimizing performance when the same SQL statement is re-executed many times, because the server-side parsing of the statement only needs to be done once prior to the first execution. Another reason to use input parameters is to embed binary data, such as *BLOB* data, into SQL statements such as *INSERT* and *UPDATE* statements.
6. *Execute statement object*: After the statement object has been successfully created and initialized with a SQL statement, the statement object can be executed. This step executes the SQL statement on the database server.

- :
7. *Process Results (optional)*: After the database server returns a result set, the application may process the results. This step is optional, since it is typically not required for statements that insert or update data in the database.
 8. *Re-execution*: If the same statement needs to be re-executed to retry on a failure or to execute again with different values for the input parameters, the application can return to Step 6. If the application has no need to re-execute the same statement, it moves on to Step 9.
 9. *Execute another SQL statement*: If the application needs to execute a different SQL statement and can reuse the statement object, then the application can return to Step 4; if not, it can move to Step 10.
 10. *End transaction (optional)*: Assuming a transaction was begun in step 2, the transaction is now either committed or rolled back. If the transaction is rolled back, then all changes to the database made by the statement object will be removed from the database.
 11. *Free resources*: After successfully executing the statement and processing any results, the client- and server-side resources need to be freed for use by other applications.

The remaining sections in this chapter provide examples of how to use the ADO.NET and JDBC database APIs to build applications that follow the steps outlined in [Figure 5-1](#).