ΠΡΟΣΠΕΛΑΣΗ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗΝ JDBC

Θέματα σε αυτό το Κεφάλαιο

- Σύνδεση με βάσεις δεδομένων: τα επτά βασικά βήματα
- Απλοποίηση της χρήσης της JDBC: μερικές βοηθητικές κλάσεις
- Χρήση προμεταγλωττισμένων (παραμετρικών) ερωτημάτων
- Δημιουργία και εκτέλεση αποθηκευμένων διαδικασιών
- Ενημέρωση δεδομένων μέσω συναλλαγών
- Χρήση του JDO και άλλων αντιστοιχίσεων αντικειμένων σε σχεσιακές βάσεις δεδομένων

Κεφάλαιο 17

Η JDBC παρέχει μια τυπική βιβλιοθήκη για την προσπέλαση σχεσιακών βάσεων δεδομένων. Χρησιμοποιώντας την API της JDBC μπορείτε να προσπελάσετε μια μεγάλη ποικιλία βάσεων δεδομένων SQL χρησιμοποιώντας την ίδια ακριβώς σύνταξη Java. Είναι σημαντικό να σημειώσουμε ότι, αν και η API της JDBC τυποποιεί τη σύνδεση με βάσεις δεδομένων, τη σύνταξη για την αποστολή ερωτημάτων και την εκτέλεση συναλλαγών, καθώς και τις δομές δεδομένων για την αναπαράσταση των αποτελεσμάτων, η JDBC *δεν* προσπαθεί να τυποποιήσει τη σύνταξη SQL. Έτσι μπορείτε να χρησιμοποιήσετε οποιεσδήποτε επεκτάσεις SQL που υποστηρίζει η βάση δεδομένων σας. Ωστόσο, επειδή τα περισσότερα ερωτήματα ακολουθούν την καθιερωμένη σύνταξη SQL, η χρήση της JDBC σας επιτρέπει να αλλάξετε τους υπολογιστές υπηρεσίας των βάσεων δεδομένων, τις θύρες, ή ακόμα και το είδος της βάσης δεδομένων, με ελάχιστες αλλαγές στον κώδικά σας.

DILBERT Ανατύπωση με την άδεια της United Feature Syndicate, Inc.

Επισήμως η JDBC δεν είναι ακρωνύμιο, και έτσι δεν αντιπροσωπεύει κάτι. Ανεπισήμως σημαίνει "Java DataBase Connectivity" (Συνδετικότητα της Java με βάσεις δεδομένων).

Αν και είναι έξω από τους στόχους αυτού του κεφαλαίου το να δώσουμε ένα πληρες εκπαιδευτικό βοήθημα για τον προγραμματισμό βάσεων δεδομένων, στην Ενότητα 17.1 (Γενική χρήση της JDBC) θα καλύψουμε τα βασικά της χρήσης της JDBC, υποθέτοντας ότι είστε ήδη εξοικειωμένοι με τη γλώσσα SQL.

Μετά από τα βασικά σχετικά με την JDBC, στην Ενότητα 17.2 (Βασικά παραδείγματα της JDBC) θα παρουσιάζουμε μερικά παραδείγματα JDBC στα οποία θα προσπελάσουμε μια βάση δεδομένων Microsoft Access.

Για απλοποίηση του κώδικα JDBC στο υπόλοιπο μέρος του βιβλίου, στην Ενότητα 17.3 (Προσπέλαση βάσεων δεδομένων με βοηθητικές κλάσεις JDBC) θα δώσουμε μερικές βοηθητικές κλάσεις για τη δημιουργία συνδέσεων με βάσεις δεδομένων.

Στην Ενότητα 17.4 (Χρήση προκατασκευασμένων εντολών) εξετάζουμε τις προκατάσκευασμένες εντολές (prepared statements), που σας επιτρέπουν να εκτελείτε παρόμοιες εντολές SQL πολλές φορές· αυτό είναι συνήθως πιο αποτελεσματικό από την εκτέλεση ενός νέου ερωτήματος κάθε φορά.

Στην Ενότητα 17.5 (Δημιουργία καλούμενων εντολών) εξετάζουμε τις καλούμενες εντολές (callable statements). Οι καλούμενες εντολές σάς επιτρέπουν να εκτελείτε αποθηκευμένες διαδικασίες ή συναρτήσεις σε βάσεις δεδομένων.

Στην Ενότητα 17.6 (Χρήση συναλλαγών βάσεων δεδομένων) περιγράφουμε τη διαχείριση συναλλαγών για τη διατήρηση της ακεραιότητας σε μια βάση δεδομένων. Όταν εκτελείτε τις τροποποιήσεις σε μια βάση δεδομένων μέσα στα πλαίσια μιας συναλλαγής, εξασφαλίζετε ότι οι τιμές της βάσης δεδομένων θα επιστραφούν στην αρχική τους μορφή αν προκύψει κάποιο πρόβλημα.

Στην Ενότητα 17.7 εξετάζουμε σε συντομία την αντιστοίχιση αντικειμένων με σχεσιακές βάσεις δεδομένων (object-to-relational mapping, ORM). Τα πλαίσια εργασιών ORM παρέχουν μια πλήρως αντικειμενοστρεφή προσέγγιση στη διαχείριση των πληροφοριών μιας βάσης δεδομένων. Με την ORM απλώς καλείτε μεθόδους σε αντικείμενα, αντί να χρησιμοποιείτε άμεσα την SQL και την JDBC.

Για προχωρημένα θέματα JDBC — όπως η προσπέλαση βάσεων δεδομένων με προσαρμοσμένες ετικέτες JSP, η χρήση προελεύσεων δεδομένων με JNDI, και η βελτίωση των επιδόσεων με δεξαμενές συνδέσεων βάσεων δεδομένων — σας παραπέμπουμε στο βιβλίο More Servlets and JavaServer Pages. Περισσότερες πληροφορίες σχετικά με την JDBC μπορείτε να βρείτε στη διεύθυνση http://java.sun.com/products/jdbc/, στην ηλεκτρονική τεκμηρίωση της API για την `java.sql`, ή στο εκπαιδευτικό βοήθημα για την JDBC, στη διεύθυνση http://java.sun.com/docs/books/tutorial/jdbc/.

17.1 Γενική χρήση της JDBC

Σε αυτή την ενότητα θα παρουσιάσουμε τα επτά τυπικά βήματα για την εκτέλεση ερωτημάτων σε βάσεις δεδομένων. Στην Ενότητα 17.2 θα δούμε δύο απλά παραδείγματα (ένα πρόγραμμα γραμμής διαταγών και μια μικροϋπηρεσία), τα οποία παρουσιάζουν αυτά τα βήματα με ένα ερώτημα σε βάση δεδομένων Microsoft Access.

Ακολουθεί μια επισκόπηση αυτών των βημάτων, ενώ στο υπόλοιπο μέρος της ενότητας θα δούμε τις λεπτομέρειες.

1. **Φόρτωση του προγράμματος οδήγησης JDBC.** Για να φορτώσετε ένα πρόγραμμα οδήγησης (driver), καθορίζετε το όνομα κλάσης της βάσης δεδομένων στη μέθοδο `Class.forName`. Με αυτόν τον τρόπο δημιουργείτε αυτόματα ένα πρόγραμμα οδήγησης που καταγράφεται στον διαχειριστή προγραμμάτων οδήγησης της JDBC.
2. **Ορισμός του URL για τη σύνδεση.** Στην JDBC, η διεύθυνση URL της σύνδεσης καθορίζει τον υπολογιστή υπηρεσίας του διακομιστή, τη θύρα, και το όνομα της βάσης δεδομένων με την οποία δημιουργείται η σύνδεση.
3. **Δημιουργία της σύνδεσης.** Με το URL σύνδεσης, το όνομα χρήστη, και τον κωδικό πρόσβασης μπορεί να δημιουργηθεί μια δικτυακή σύνδεση με τη βάση δεδομένων. Από τη στιγμή που έχει δρομολογηθεί η σύνδεση, μπορείτε να εκτελείτε ερωτήματα στη βάση δεδομένων μέχρι να κλείσετε τη σύνδεση αυτή.
4. **Δημιουργία ενός αντικειμένου `Statement`.** Η δημιουργία ενός αντικειμένου `Statement` σάς δίνει τη δυνατότητα να στείλετε ερωτήματα και εντολές στη βάση δεδομένων.
5. **Εκτέλεση ερωτήματος ή ενημέρωσης.** Με δεδομένο το αντικείμενο `Statement`, μπορείτε να στείλετε εντολές SQL στη βάση δεδομένων χρησιμοποιώντας τις μεθόδους `execute`, `executeQuery`, `executeUpdate`, ή `executeBatch`.
6. **Επεξεργασία των αποτελεσμάτων.** Όταν εκτελείται ένα ερώτημα βάσης δεδομένων, επιστρέφεται ένα αντικείμενο `ResultSet`. Το αντικείμενο `ResultSet` αντιπροσωπεύει ένα σύνολο γραμμών και στηλών, το οποίο μπορείτε να επεξεργαστείτε με κλήσεις των μεθόδων `next` και `getXxx`.
7. **Κλείσιμο της σύνδεσης.** Όταν ολοκληρώσετε την εκτέλεση ερωτημάτων και την επεξεργασία των αποτελεσμάτων, θα πρέπει να κλείσετε τη σύνδεση έτσι ώστε να αποδεσμεύσετε τους πόρους που έχουν χρησιμοποιηθεί στη βάση δεδομένων.

Φόρτωση του προγράμματος οδήγησης JDBC

Το πρόγραμμα οδήγησης είναι το κομμάτι λογισμικού που γνωρίζει πώς να επικοινωνήσει με τον πραγματικό διακομιστή της βάσης δεδομένων. Για να φορτώσετε το πρόγραμμα οδήγησης, φορτώνετε απλώς την κατάλληλη κλάση· ένα μπλοκ `static` στην ίδια την κλάση του προγράμματος οδήγησης δημιουργεί αυτόματα ένα στιγμιότυπο του προγράμματος οδήγησης και το καταγράφει (register) στο διαχειριστή προγραμμάτων οδήγησης της JDBC. Για να κάνετε τον κώδικά σας όσο το δυνατόν πιο ευέλικτο, αποφύγετε την "κυριολεκτική" αναφορά του ονόματος της κλάσης. Στην Ενότητα 17.3 (Προσπέλαση βάσεων δεδομένων με βοηθητικές κλάσεις JDBC) παρουσιάζουμε μια βοηθητική κλάση για τη φόρτωση προγραμμάτων οδήγησης από ένα αρχείο `Properties`, οπότε το όνομα κλάσης δεν αναγράφεται αυτούσιο μέσα στο πρόγραμμα.

Από αυτές τις απαιτήσεις ανακύπτουν δύο ενδιαφέροντα ερωτήματα. Πρώτον, πώς φορτώνετε μια κλάση χωρίς να δημιουργήσετε ένα στιγμιότυπό της; Δεύτερον, πώς μπορείτε να κάνετε αναφορές στο όνομα κάποιας κλάσης όταν αυτό δεν είναι γνωστό κατά τη μεταγλώττιση του κώδικα; Η απάντηση και στις δύο ερωτήσεις είναι η χρήση της μεθόδου `Class.forName`. Αυτή η μέθοδος δέχεται ένα αλφαριθμητικό που αντιπροσωπεύει το πλήρες όνομα κλάσης (δηλαδή, αυτό που περιλαμβάνει τα ονόματα των πακέτων) και φορτώνει την αντίστοιχη κλάση. Επειδή αυτή η κλάση μπορεί να μεταβιβάσει μια εξαίρεση `ClassNotFoundException`, θα πρέπει να βρίσκεται στο εσωτερικό ενός μπλοκ `try/catch`, όπως φαίνεται παρακάτω.

```
try {
  Class.forName("connect.microsoft.MicrosoftDriver");
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Class.forName("com.sybase.jdbc.SybDriver");
} catch(ClassNotFoundException cnfe) {
  System.err.println("Error loading driver: " + cnfe);
}
```

Ένα από τα όμορφα χαρακτηριστικά της προσέγγισης JDBC είναι ότι ο διακομιστής της βάσης δεδομένων δεν χρειάζεται καμία απολύτως αλλαγή. Αντίθετα, το πρόγραμμα οδήγησης JDBC (που βρίσκεται στον πελάτη) μεταφράζει τις κλήσεις (οι οποίες έχουν γραφτεί στη γλώσσα προγραμματισμού Java) στην εγγενή μορφή που χρειάζεται ο διακομιστής. Αυτή η προσέγγιση σημαίνει ότι πρέπει να αποκτήσετε ένα πρόγραμμα οδήγησης JDBC ειδικά για τη βάση δεδομένων που χρησιμοποιείτε, καθώς και ότι θα πρέπει να ελέγξετε την τεκμηρίωση του κατασκευαστή για το πλήρες όνομα κλάσης που θα χρησιμοποιήσετε.

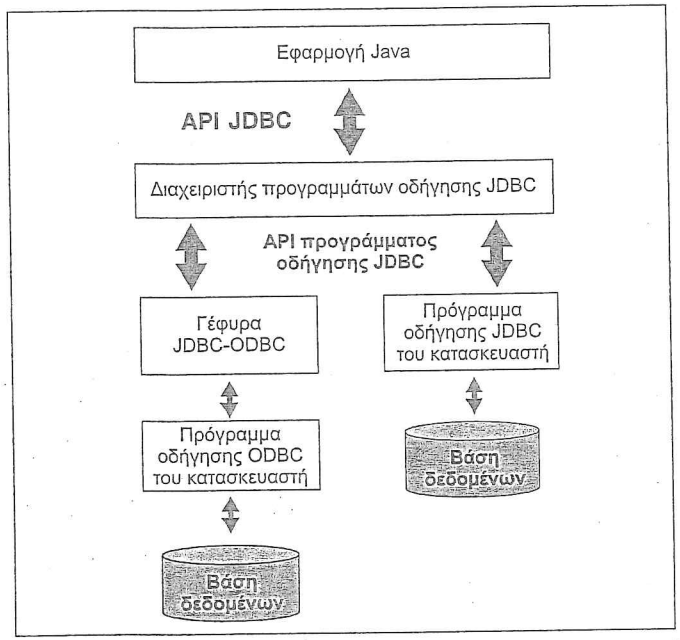
Θεωρητικά, μπορείτε να χρησιμοποιήσετε τη μέθοδο `Class.forName` για οποιαδήποτε κλάση που υπάρχει στη μεταβλητή περιβάλλοντος `CLASSPATH`. Στην πράξη, όμως, οι περισσότεροι κατασκευαστές προγραμμάτων οδήγησης JDBC διανέμουν τα προγράμματα οδήγησής τους σε αρχεία JAR. Έτσι, στη φάση της ανάπτυξης, φροντίστε να συμπεριλάβετε στις ρυθμίσεις της `CLASSPATH` τη διαδρομή για το αρχείο JAR του προγράμματος οδήγησης. Στη φάση της εκδίπλωσης σε κάποιο διακομιστή Ιστού, τοποθετήστε το αρχείο JAR στον κατάλογο WEB-INF/lib της εφαρμογής Ιστού (δείτε το Κεφάλαιο 2, "Εγκατάσταση και διευθέτηση διακομιστή Ιστού"). Θα πρέπει όμως να συμβουλευθείτε το διαχειριστή του διακομιστή Ιστού. Συχνά, όταν πολλές εφαρμογές Ιστού χρησιμοποιούν τα ίδια προγράμματα οδήγησης βάσεων δεδομένων, ο διαχειριστής τοποθετεί το αρχείο JAR σε έναν κοινό κατάλογο που χρησιμοποιείται από το διακομιστή. Για παράδειγμα, στον Apache Tomcat, τα αρχεία JAR που είναι κοινόχρηστα για πολλές εφαρμογές μπορούν να τοποθετηθούν στον κατάλογο *κατάλογος_εγκατάστασης*/common/lib.

Σημείωση

Μπορείτε να τοποθετήσετε το αρχείο του προγράμματος οδήγησης JDBC (αρχείο JAR) στον κατάλογο WEB-INF/lib του διακομιστή εκδίπλωσης της εφαρμογής σας. Ωστόσο, ο διαχειριστής μπορεί να επιλέξει να μετακινήσει το αρχείο JAR σε έναν κοινό κατάλογο βιβλιοθηκών του διακομιστή.

Η Εικόνα 17-1 παρουσιάζει δύο συνηθισμένες υλοποιήσεις προγραμμάτων οδήγησης JDBC. Η πρώτη προσέγγιση είναι μια γέφυρα JDBC-ODBC, ενώ η δεύτερη προσέγγιση είναι μια αμιγής υλοποίηση σε Java. Τα προγράμματα οδήγησης που χρησιμοποιούν την προσέγγιση της γέφυρας JDBC-ODBC είναι γνωστά ως προγράμματα οδήγησης Τύπου I. Επειδή πολλές βάσεις δεδομένων υποστηρίζουν την προσπέλαση μέσω ODBC (Open DataBase Connectivity, Ανοικτή Συνδετικότητα Βάσεων Δεδομένων), το JDK περιλαμβάνει μια γέφυρα JDBC-ODBC για τη σύνδεση με βάσεις δεδομένων. Παρόλα αυτά, καλό είναι να χρησιμοποιείτε το πρόγραμμα οδήγησης του κατασκευαστή σε αμιγή Java, αν είναι διαθέσιμο, επειδή η υλοποίηση JDBC-ODBC των προγραμμάτων οδήγησης είναι πιο αργή από την αμιγή υλοποίηση σε Java. Τα προγράμματα οδήγησης σε καθαρή Java είναι γνωστά ως προγράμματα οδήγησης Τύπου IV. Οι προδιαγραφές της JDBC ορίζουν και άλλους δύο τύπους προγραμμάτων οδήγησης, τον Τύπο II και τον Τύπο III, που είναι λιγότερο συνηθισμένοι. Περισσότερες πληροφορίες για τους τύπους προγραμμάτων οδήγησης μπορείτε να βρείτε στη διεύθυνση http://java.sun.com/products/jdbc/driverdesc.html.

Στα αρχικά παραδείγματα αυτού του κεφαλαίου χρησιμοποιούμε τη γέφυρα JDBC-ODBC, που περιλαμβάνεται στο JDK 1.4, για σύνδεση με μια βάση δεδομένων Microsoft Access. Στα επόμενα παραδείγματα χρησιμοποιούμε προγράμματα οδήγησης σε αμιγή Java για σύνδεση με βάσεις δεδομένων MySQL και Oracle9i.



Εικόνα 17-1 Δυο συνηθισμένες υλοποιήσεις προγραμμάτων οδήγησης JDBC. Το JDK 1.4 περιλαμβάνει μια γέφυρα JDBC-ODBC· ωστόσο, τα προγράμματα οδήγησης σε αμιγή Java (που παρέχονται από τον κατασκευαστή) δίνουν καλύτερα αποτελέσματα.

Στην Ενότητα 18.1 (Διευθέτηση της Microsoft Access για χρήση με την JDBC) δίνουμε πληροφορίες για το πρόγραμμα οδήγησης της Microsoft Access. Πληροφορίες για το πρόγραμμα οδήγησης της MySQL δίνονται στην Ενότητα 18.2 (Εγκατάσταση και διευθέτηση της MySQL), ενώ πληροφορίες για το πρόγραμμα οδήγησης της Oracle9i δίνουμε στην Ενότητα 18.3 (Εγκατάσταση και διευθέτηση της Oracle9i). Οι περισσότεροι κατασκευαστές άλλων βάσεων δεδομένων παρέχουν δωρεάν προγράμματα οδήγησης JDBC για τις βάσεις δεδομένων τους. Μια ενημερωμένη λίστα με αυτά τα προγράμματα οδήγησης, καθώς και για προγράμματα οδήγησης τρίτων κατασκευαστών, μπορείτε να βρείτε στη διεύθυνση http://industry.java.sun.com/products/jdbc/drivers/.

Ορισμός του URL σύνδεσης

Αφού φορτώσετε το πρόγραμμα οδήγησης JDBC, θα πρέπει να καθορίσετε τη θέση του διακομιστή της βάσης δεδομένων. Οι διευθύνσεις URL που αναφέρονται σε βάσεις δεδομένων χρησιμοποιούν το πρωτόκολλο `jdbc:`, και ενσωματώνουν στο URL τον υπολογιστή υπηρεσίας του διακομιστή, τη θύρα, και το όνομα (ή μία αναφορά) για τη βάση δεδομένων. Η ακριβής μορφή καθορίζεται στην τεκμηρίωση που συνοδεύει το συγκεκριμένο πρόγραμμα οδήγησης, αλλά τα παρακάτω είναι μερικά αντιπροσωπευτικά παραδείγματα.

```
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;
String msAccessURL = "jdbc:odbc:" + dbName;
```

Δρομολόγηση της σύνδεσης

Για να πραγματοποιήσετε την πραγματική δικτυακή σύνδεση, μεταβιβάζετε το URL, το όνομα χρήστη της βάσης δεδομένων, και τον κωδικό χρήστη της βάσης δεδομένων στη μέθοδο `getConnection` της κλάσης `DriverManager`, όπως φαίνεται στο παρακάτω παράδειγμα. Σημειώστε ότι η μέθοδος `getConnection` μεταβιβάζει την εξαίρεση `SQLException`, γι' αυτό θα πρέπει να χρησιμοποιείτε ένα μπλοκ `try/catch`. Στο παράδειγμα που ακολουθεί έχουμε παραλείψει αυτό το μπλοκ επειδή οι μέθοδοι των επόμενων βημάτων μεταβιβάζουν επίσης την ίδια εξαίρεση, οπότε θα χρησιμοποιήσουμε το ίδιο μπλοκ `try/catch` για όλες τις μεθόδους.

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


Η κλάση `Connection` περιλαμβάνει και άλλες χρήσιμες μεθόδους, τις οποίες περιγράφουμε σε συντομία στη συνέχεια. Οι πρώτες τρεις από αυτές εξετάζονται λεπτομερώς στις Ενότητες 17.4-17.6.

- **`prepareStatement`.** Δημιουργεί προμεταγλωττισμένα ερωτήματα για υποβολή στη βάση δεδομένων. Για λεπτομέρειες δείτε την Ενότητα 17.4 (Χρήση προκατασκευασμένων εντολών).
- **`prepareCall`.** Προσπελάζει αποθηκευμένες διαδικασίες της βάσης δεδομένων. Για λεπτομέρειες δείτε την Ενότητα 17.5 (Δημιουργία καλούμενων εντολών).
- **`rollback/commit`.** Ρυθμίζει τη διαχείριση των συναλλαγών. Για λεπτομέρειες δείτε την Ενότητα 17.6 (Χρήση συναλλαγών βάσεων δεδομένων).

- **`close`.** Τερματίζει την ανοικτή σύνδεση.
- **`isClosed`.** Προσδιορίζει αν η σύνδεση έληξε ή τερματίστηκε ρητά.

Μια προαιρετική ενέργεια για τη δρομολόγηση της σύνδεσης είναι η αναζήτηση πληροφοριών σχετικά με τη βάση δεδομένων, με τη μέθοδο `getMetaData`. Αυτή η μέθοδος επιστρέφει ένα αντικείμενο `DatabaseMetaData`, το οποίο διαθέτει μεθόδους με τις οποίες μπορείτε να ανακαλύψετε το όνομα και την έκδοση της ίδιας της βάσης δεδομένων (`getDatabaseProductName`, `getDatabaseProductVersion`) ή του προγράμματος οδήγησης JDBC (`getDriverName`, `getDriverVersion`). Ας δούμε ένα παράδειγμα.

```
DatabaseMetaData dbMetaData = connection.getMetaData();
String productName =
  dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =
  dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
```

Δημιουργία ενός αντικειμένου Statement

Το αντικείμενο `Statement` χρησιμοποιείται για την αποστολή ερωτημάτων και εντολών στη βάση δεδομένων. Δημιουργείται από το αντικείμενο `Connection` με τη μέθοδο `createStatement`, με τον εξής τρόπο:

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

Τα περισσότερα προγράμματα οδήγησης βάσεων δεδομένων, όχι όμως όλα, επιτρέπουν να είναι ταυτόχρονα ανοικτά πολλά αντικείμενα `Statement` στην ίδια σύνδεση.

Εκτέλεση ερωτήματος ή ενημέρωσης

Αφού αποκτήσετε το αντικείμενο `Statement`, μπορείτε να το χρησιμοποιήσετε για την αποστολή ερωτημάτων SQL μέσω της μεθόδου `executeQuery`, η οποία επιστρέφει ένα αντικείμενο τύπου `ResultSet`. Ας δούμε ένα παράδειγμα.

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

Ο κατάλογος που ακολουθεί συνοψίζει τις πιο συχνά χρησιμοποιούμενες μεθόδους της κλάσης `Statement`.

- **`executeQuery`**. Εκτελεί ένα ερώτημα SQL και επιστρέφει τα δεδομένα σε ένα αντικείμενο `ResultSet`. Το αντικείμενο `ResultSet` μπορεί να είναι άδειο, ποτέ όμως δεν μπορεί να είναι `null`.
- **`executeUpdate`**. Χρησιμοποιείται για τις εντολές `UPDATE`, `INSERT`, και `DELETE`. Επιστρέφει τον αριθμό των γραμμών που επηρεάζονται, ο οποίος μπορεί να είναι μηδέν. Παρέχει επίσης υποστήριξη για εντολές της Γλώσσας Ορισμού Δεδομένων (Data Definition Language, DDL), όπως για παράδειγμα `CREATE TABLE`, `DROP TABLE`, και `ALTER TABLE`.
- **`executeBatch`**. Εκτελεί μια ομάδα εντολών σαν να ήταν μία οντότητα, και επιστρέφει έναν πίνακα με τις ενημερωμένες καταμετρήσεις για κάθε εντολή. Για να προσθέσετε μια εντολή στην ομάδα εντολών χρησιμοποιείτε τη μέθοδο `addBatch`. Σημειώστε ότι δεν είναι υποχρεωτικό να υλοποιούν οι κατασκευαστές αυτή τη μέθοδο στο πρόγραμμα οδήγησής τους προκειμένου να έχουν συμβατότητα με τη JDBC.
- **`setQueryTimeOut`**. Καθορίζει το χρόνο κατά τον οποίο θα περιμένει για αποτελέσματα το πρόγραμμα οδήγησης, πριν μεταβιβάσει μια εξαίρεση τύπου `SQLException`.
- **`getMaxRows/setMaxRows`**. Καθορίζει τον αριθμό των γραμμών που μπορεί να περιέχει ένα αντικείμενο `ResultSet`. Οι επιπλέον γραμμές απορρίπτονται σιωπηρά. Η προεπιλεγμένη τιμή είναι μηδέν, που σημαίνει ότι δεν υπάρχει όριο ως προς τον αριθμό των γραμμών.

Εκτός από τη χρήση των μεθόδων που περιγράφονται εδώ για την αποστολή εντολών, μπορείτε να χρησιμοποιήσετε το αντικείμενο `Statement` για να δημιουργήσετε παραμετρικά ερωτήματα (parameterized queries) μέσω των οποίων θα παρέχονται τιμές σε ένα προμεταγλωττισμένο ερώτημα σταθερής μορφής. Για λεπτομέρειες δείτε την Ενότητα 17.4 (Χρήση προκατασκευασμένων εντολών).

Επεξεργασία των αποτελεσμάτων

Ο απλούστερος τρόπος χειρισμού των αποτελεσμάτων είναι η χρήση της μεθόδου `next` της κλάσης `ResultSet`, έτσι ώστε να κινηθείτε μέσα στον πίνακα αποτελεσμάτων κατά μία γραμμή κάθε φορά. Μέσα σε κάθε γραμμή, η κλάση `ResultSet` παρέχει διάφορες μεθόδους `getXxx` οι οποίες δέχονται ως όρισμα το όνομα της στήλης ή τον αριθμοδείκτη της στήλης και επιστρέφουν το αποτέλεσμα σε μια ποικιλία διαφορετικών τύπων Java. Για παράδειγμα, χρησιμοποιείτε τη μέθοδο `getInt` αν η τιμή είναι ακέραιος, τη μέθοδο `getString` αν η τιμή είναι `String`, κ.ο.κ για τους περισσότερους τύπους δεδομένων. Αν θέλετε απλώς να εμφανίσετε τα αποτελέσματα, μπορείτε να χρησιμοποιήσετε τη μέθοδο `getString` για τους περισσότερους τύπους στηλών. Αν χρησιμοποιήσετε όμως την εκδοχή της μεθόδου `getXxx` που δέχεται ως όρισμα τον αριθμοδείκτη της στήλης (και όχι το όνομα της στήλης), θα πρέπει να προσέξετε ότι οι αριθμοδείκτες των στηλών αρχίζουν από το 1 (ακολουθώντας τη σύμβαση της SQL), και όχι από το 0 όπως συμβαίνει στους πίνακες, τα διανύσματα, και τις περισσότερες δομές δεδομένων της γλώσσας προγραμματισμού Java.

Προειδοποίηση

Η πρώτη στήλη σε μια γραμμή του αντικειμένου `ResultSet` έχει αριθμοδείκτη 1, και όχι 0.

Ακολουθεί ένα παράδειγμα που τυπώνει τις τιμές των δύο πρώτων στηλών, μαζί με ένα όνομα και επώνυμο, για όλες τις γραμμές ενός αντικειμένου `ResultSet`.

```
while(resultSet.next()) {
  System.out.println(resultSet.getString(1) + " " +
                     resultSet.getString(2) + " " +
                     resultSet.getString("firstname") + "  "
                     resultSet.getString("lastname"));
}
```

Σας συνιστούμε, όταν προσπελάζετε τις στήλες ενός αντικειμένου `ResultSet`, να χρησιμοποιείτε το όνομα της στήλης αντί για τον αριθμοδείκτη της στήλης. Με αυτόν τον τρόπο, αν αλλάξει η δομή των στηλών, ο κώδικας που αλληλεπιδρά με το αντικείμενο `ResultSet` θα είναι λιγότερο πιθανό να αστοχήσει.

Η προσέγγιση του βιβλίου

Χρησιμοποιήστε το όνομα της στήλης, αντί του αριθμοδείκτη της στήλης, όταν προσπελάζετε τα δεδομένα σε ένα αντικείμενο `ResultSet`.

Στην JDBC 1.0 μπορείτε να κινηθείτε μόνο προς τα εμπρός μέσα στο αντικείμενο ResultSet· ωστόσο, στην JDBC 2.0 μπορείτε να κινηθείτε τόσο προς τα εμπρός (next) όσο και προς τα πίσω (previous) στο αντικείμενο ResultSet, καθώς επίσης και να μεταφερθείτε σε μια συγκεκριμένη γραμμή (relative, absolute). Στο βιβλίο More Servlets and JavaServer Pages παρουσιάζουμε αρκετές προσαρμοσμένες ετικέτες οι οποίες επιδεικνύουν τις μεθόδους της JDBC 2.0 που είναι διαθέσιμες σε ένα αντικείμενο ResultSet.

Να θυμάστε ότι ούτε η JDBC 1.0, ούτε η JDBC 2.0 παρέχουν κάποιον άμεσο μηχανισμό για τον προσδιορισμό της έκδοσης JDBC του προγράμματος οδήγησης. Στην JDBC 3.0 αυτό το πρόβλημα επιλύεται με την προσθήκη των μεθόδων getJDBCMajorVersion και getJDBCMinorVersion στην κλάση DatabaseMetaData. Αν η έκδοση JDBC δεν είναι σαφής από την τεκμηρίωση του κατασκευαστή, μπορείτε να γράψετε ένα σύντομο πρόγραμμα για να πάρετε ένα αντικείμενο ResultSet και να δοκιμάσετε τη μέθοδο previous σε αυτό. Επειδή η μέθοδος resultSet.previous υπάρχει μόνο στη JDBC 2.0 ή νεότερη, ένα πρόγραμμα οδήγησης JDBC 1.0 θα μεταβιβάσει σε αυτό το σημείο μια εξαίρεση. Δείτε την Ενότητα 18.4 (Έλεγχος της βάσης δεδομένων σας μέσω μιας σύνδεσης JDBC) για ένα παράδειγμα προγράμματος το οποίο εκτελεί ένα "χαλαρό" έλεγχο για να προσδιορίσει την έκδοση JDBC του προγράμματος οδήγησης της βάσης δεδομένων σας.

Ο κατάλογος που ακολουθεί συνοψίζει μερικές χρήσιμες μεθόδους της κλάσης ResultSet.

- **next/previous**. Μετακινεί, αντίστοιχα, το δρομέα (cursor) στην επόμενη γραμμή (σε οποιαδήποτε έκδοση JDBC) ή στην προηγούμενη γραμμή (JDBC έκδοση 2.0 ή νεότερη) του αντικειμένου ResultSet.
- **relative/absolute**. Η μέθοδος relative μετακινεί το δρομέα κατά ένα σχετικό αριθμό γραμμών, είτε θετικά (προς τα επάνω) είτε αρνητικά (προς τα κάτω). Η μέθοδος absolute τοποθετεί το δρομέα στο συγκεκριμένο αριθμό γραμμής. Αν η απόλυτη τιμή είναι αρνητική, ο δρομέας τοποθετείται σε σχέση με το τέλος του αντικειμένου ResultSet (JDBC 2.0).
- **getXxx**. Επιστρέφει την τιμή από τη στήλη που καθορίζεται από το όνομα της στήλης ή τον αριθμοδείκτη της στήλης, σε τύπο δεδομένων *Xxx* της Java (δείτε σχετικά την κλάση java.sql.Types). Μπορεί να επιστρέψει 0 ή null αν η τιμή είναι SQL NULL.
- **wasNull**. Ελέγχει αν η τελευταία ανάγνωση *Xxx* ήταν SQL NULL. Αυτός ο έλεγχος είναι σημαντικός αν ο τύπος της στήλης είναι κάποιος από τους βασικούς τύπους δεδομένων (int, float, κ.λπ.) και η τιμή στη βάση δεδομένων είναι 0. Η τιμή μηδέν δεν μπορεί να διακριθεί από την τιμή NULL στη βάση δεδομένων, αφού και αυτή επιστρέφεται ως 0. Αν ο τύπος στήλης είναι κάποιο αντικείμενο (String, Date, κ.λπ.) μπορείτε απλώς να συγκρίνετε την επιστρεφόμενη τιμή με την τιμή null.
- **findColumn**. Επιστρέφει τον αριθμοδείκτη του αντικειμένου ResultSet που αντιστοιχεί στο καθορισμένο όνομα στήλης.
- **getRow**. Επιστρέφει τον τρέχοντα αριθμό γραμμής, με την πρώτη γραμμή να αρχίζει από 1 (JDBC 2.0).

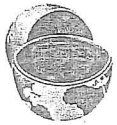
- **getMetaData**. Επιστρέφει ένα αντικείμενο ResultSetMetaData που περιγράφει το αντικείμενο ResultSet. Το αντικείμενο ResultSetMetaData παρέχει τον αριθμό και τα ονόματα των στηλών.

Η μέθοδος getMetaData είναι εξαιρετικά χρήσιμη. Έχοντας στη διάθεσή σας μόνο ένα αντικείμενο ResultSet, θα πρέπει να γνωρίζετε το όνομα, τον αριθμό, και τον τύπο των στηλών, προκειμένου να μπορέσετε να επεξεργαστείτε κατάλληλα τον πίνακα. Για τα περισσότερα ερωτήματα σταθερής μορφής, κάτι τέτοιο αποτελεί εύλογη προσδοκία. Για ειδικά ερωτήματα, όμως, είναι χρήσιμη η δυνατότητα της ανακάλυψης, με δυναμικό τρόπο, πληροφοριών υψηλού επιπέδου σχετικά με το αποτέλεσμα. Αυτός είναι ο ρόλος της κλάσης ResultSetMetaData: σας επιτρέπει να προσδιορίσετε τον αριθμό, τα ονόματα, και τους τύπους των στηλών στο αντικείμενο ResultSet.

Παρακάτω περιγράφονται οι μέθοδοι της κλάσης ResultSetMetaData.

- **getColumnCount**. Επιστρέφει τον αριθμό των στηλών στο αντικείμενο ResultSet.
- **getColumnName**. Επιστρέφει το όνομα μιας στήλης στη βάση δεδομένων (η αρίθμηση ξεκινά από το 1).
- **getColumnType**. Επιστρέφει τον τύπο SQL, για σύγκριση με τις καταχωρίσεις της κλάσης java.sql.Types.
- **isReadyOnly**. Δείχνει αν η καταχώριση είναι τιμή μόνο για ανάγνωση.
- **isSearchable**. Δείχνει αν η καταχώριση μπορεί να χρησιμοποιηθεί σε μια εντολή WHERE.
- **isNullable**. Δείχνει αν είναι έγκυρη η αποθήκευση μιας τιμής NULL στη στήλη.

Η κλάση ResultSetMetaData *δεν* περιλαμβάνει πληροφορίες σχετικά με τον αριθμό των γραμμών· αν όμως το πρόγραμμα οδήγησης που χρησιμοποιείτε ακολουθεί τις προδιαγραφές της JDBC 2.0, μπορείτε να καλέσετε τη μέθοδο last στο αντικείμενο ResultSet έτσι ώστε να μετακινήσετε το δρομέα στην τελευταία γραμμή, και μετά να καλέσετε τη μέθοδο getRow για να ανακτήσετε τον αριθμό της τρέχουσας γραμμής. Στην JDBC 1.0, ο μόνος τρόπος για να προσδιορίσετε τον αριθμό των γραμμών είναι η συνεχής κλήση της μεθόδου next στο αντικείμενο ResultSet μέχρι να επιστρέψει τιμή false.



Σημείωση

Οι κλάσεις ResultSet και ResultSetMetaData δεν διαθέτουν άμεσα κάποια μέθοδο που να σας δίνει το πλήθος των γραμμών οι οποίες επιστρέφονται από ένα ερώτημα. Ωστόσο, στην JDBC 2.0 μπορείτε να τοποθετήσετε το δρομέα στην τελευταία γραμμή του αντικειμένου ResultSet, μέσω της μεθόδου last, και μετά να πάρετε τον τρέχοντα αριθμό γραμμής με τη μέθοδο getRow.

Κλείσιμο της σύνδεσης

Για να κλείσετε τη σύνδεση με ρητό τρόπο, θα πρέπει να δώσετε την εντολή

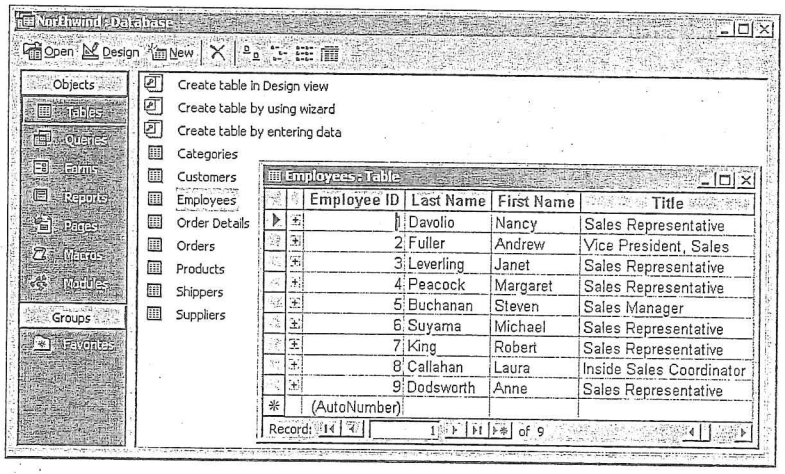
```
connection.close();
```

Το κλείσιμο της σύνδεσης κλείνει επίσης τα αντίστοιχα αντικείμενα `Statement` και `ResultSet`.

Θα πρέπει να αναβάλλετε το κλείσιμο της σύνδεσης στην περίπτωση που αναμένετε ότι μπορεί να εκτελέσετε πρόσθετες λειτουργίες στη βάση δεδομένων, επειδή η επιβάρυνση από το άνοιγμα μιας σύνδεσης είναι συνήθως μεγάλη. Στην πραγματικότητα, η επαναχρησιμοποίηση συνδέσεων που υπάρχουν ήδη αποτελεί τόσο σημαντική βελτιστοποίηση, ώστε η API της JDBC 2.0 ορίζει μια διασύνδεση `ConnectionPoolDataSource` για τη λήψη μιας δεξαμενής συνδέσεων (pooled connections). Οι δεξαμενές συνδέσεων εξετάζονται στο βιβλίο More Servlets and JavaServer Pages.

17.2 Βασικά παραδείγματα JDBC

Σε αυτή την ενότητα θα παρουσιάσουμε δύο απλά παραδείγματα JDBC τα οποία συνδέονται με τη βάση δεδομένων Northwind της Microsoft Access (φαίνεται στην Εικόνα 17-2) και εκτελούν ένα απλό ερώτημα. Η βάση δεδομένων Northwind περιλαμβάνεται στην ενότητα παραδειγμάτων του Microsoft Office. Για πληροφορίες σχετικά με τη διευθέτηση της βάσης δεδομένων Northwind για προσπέλαση από την JDBC, δείτε την Ενότητα 18.1.



Εικόνα 17-2 Η βάση δεδομένων Northwind της Microsoft Access, όπου φαίνονται οι τέσσερις πρώτες στήλες του πίνακα Employees. Για πληροφορίες σχετικά με τη χρήση αυτής της βάσης δεδομένων, δείτε την Ενότητα 18.1.

Η Northwind είναι μια καλή βάση δεδομένων για έλεγχο και πειραματισμό, επειδή είναι ήδη εγκατεστημένη σε πολλά συστήματα αλλά και επειδή η γέφυρα JDBC-ODBC για τη σύνδεση με τη Microsoft Access υπάρχει ήδη στο JDK. Ωστόσο, η Microsoft Access δεν προορίζεται για σοβαρές online βάσεις δεδομένων. Για χρήση σε ένα πραγματικό περιβάλλον παραγωγής, είναι πολύ καλύτερο να επιλέξετε κάποια λύση υψηλότερης απόδοσης — όπως η MySQL (δείτε την Ενότητα 18.2), η Oracle9i (δείτε την Ενότητα 18.3), ο Microsoft SQL Server, η Sybase, ή η DB2.

Το πρώτο παράδειγμα, στη Λίστα 17.1, παρουσιάζει μια αυτόνομη κλάση με το όνομα `NorthwindTest`, η οποία ακολουθεί τα επτά βήματα που περιγράψαμε στην προηγούμενη ενότητα για να εμφανίσει τα αποτελέσματα ενός ερωτήματος στον πίνακα `Employees`.

Τα αποτελέσματα της `NorthwindTest` παρουσιάζονται στη Λίστα 17.2. Επειδή η `NorthwindTest` βρίσκεται στο πακέτο `coreservlets`, τοποθετείται στον υποκατάλογο coreservlets. Πριν από τη μεταγλώττιση του αρχείου, ορίστε τη μεταβλητή `CLASSPATH` έτσι ώστε να περιλαμβάνει τον κατάλογο που *περιέχει* τον κατάλογο coreservlets. Για λεπτομέρειες δείτε την Ενότητα 2.7 (Διαμόρφωση του δικού σας περιβάλλοντος ανάπτυξης). Με αυτή τη διευθέτηση, η μεταγλώττιση του προγράμματος γίνεται με την εκτέλεση της διαταγής `javac NorthwindTest.java` από τον υποκατάλογο coreservlets (ή με την επιλογή της διαταγής "build" ή "compile" του IDE που χρησιμοποιείτε). Για να εκτελέσετε την εφαρμογή `NorthwindTest`, πρέπει να αναφέρετε το πλήρες όνομα του πακέτου: `java coreservlets.NorthwindTest`.

Το δεύτερο παράδειγμα, στη Λίστα 17.3 (`NorthwindServlet`), συνδέεται με τη βάση δεδομένων μέσα από μια μικροϋπηρεσία και παρουσιάζει τα αποτελέσματα του ερωτήματος σε έναν πίνακα HTML. Τόσο η Λίστα 17.1, όσο και η Λίστα 17.3 χρησιμοποιούν το πρόγραμμα οδήγησης γέφυρας JDBC-ODBC (`sun.jdbc.odbc.JdbcOdbcDriver`) που περιλαμβάνεται στο JDK.

Λίστα 17.1 NortwindTest.java

```
package coreservlets;

import java.sql.*;

/** Ένα παράδειγμα JDBC που συνδέεται με τη βάση δεδομένων Northwind
 *  της MicroSoft Access, εκτελεί ένα απλό ερώτημα SQL στον πίνακα
 *  Employees, και τυπώνει τα αποτελέσματα.
 *  <P>
 */

public class NorthwindTest {
  public static void main(String[] args) {
    String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    String url = "jdbc:odbc:Northwind";
    String username = ""; // Δεν χρειάζεται όνομα χρήστη/κωδικός πρόσβασης
    String password = ""; // για πρόσβαση στη MS Access στο ίδιο σύστημα.
    showEmployeeTable(driver, url, username, password);
  }

  /** Ερώτημα στον πίνακα Employees και εκτύπωση των ονομάτων και
   *  των επώνυμων.
   */
```

Λίστα 17.1 NortwindTest.java *(συνέχεια)*

```
  public static void showEmployeeTable(String driver,
                                        String url,
                                        String username,
                                        String password) {
    try {
      // Φόρτωση του προγράμματος οδήγησης της βάσης δεδομένων,
      // αν δεν έχει ήδη φορτωθεί
      Class.forName(driver);
      // Δρομολόγηση δικτυακής σύνδεσης με τη βάση δεδομένων.
      Connection connection =
        DriverManager.getConnection(url, username, password);
      System.out.println("Employees\n" + "==========");
      // Δημιουργία αντικειμένου statement για την εκτέλεση ερωτημάτων.
      Statement statement = connection.createStatement();
      String query =
        "SELECT firstname, lastname FROM employees";
      // Αποστολή ερωτήματος στη βάση δεδομένων και αποθήκευση
      // των αποτελεσμάτων.
      ResultSet resultSet = statement.executeQuery(query);
      // Εκτύπωση αποτελεσμάτων.
      while(resultSet.next()) {
        System.out.print(resultSet.getString("firstname") + " ");
        System.out.println(resultSet.getString("lastname"));
      }
      connection.close();
    } catch(ClassNotFoundException cnfe) {
      System.err.println("Error loading driver: " + cnfe);
    } catch(SQLException sqle) {
      System.err.println("Error with connection: " + sqle);
    }
  }
}
```

Λίστα 17.2 Αποτέλεσμα της NorthwindTest

```
Σήμα γραμμής διαταγών> java coreservlets.NorthwindTest
Employees
==========
Nancy Davolio
Andrew Fuller
Janet Leverling
Margaret Peacock
Steven Buchanan
Michael Suyama
Robert King
Laura Callahan
Anne Dodsworth
```


Για το δεύτερο παράδειγμα, το NorthwindServlet (Λίστα 17.3), οι πληροφορίες για την εκτέλεση του ερωτήματος λαμβάνονται από μια φόρμα HTML, τη NorthwindForm.html, που φαίνεται στη Λίστα 17.4. Εδώ μπορείτε να καταχωρίσετε το ερώτημα στην περιοχή κειμένου της φόρμας, πριν υποβάλλετε τη φόρμα στη μικροϋπηρεσία. Η μικροϋπηρεσία διαβάζει το πρόγραμμα οδήγησης, το URL, το όνομα χρήστη, τον κωδικό πρόσβασης, και το ερώτημα από τις παραμέτρους αίτησης και παράγει τον πίνακα HTML με βάση τα αποτελέσματα του ερωτήματος. Η μικροϋπηρεσία δείχνει επίσης τη χρήση της κλάσης DatabaseMetaData για την αναζήτηση του ονόματος και της έκδοσης της βάσης δεδομένων. Η φόρμα HTML φαίνεται στην Εικόνα 17-3· η Εικόνα 17-4 δείχνει το αποτέλεσμα της υποβολής της φόρμας. Σε αυτό το παράδειγμα, η φόρμα HTML και η μικροϋπηρεσία βρίσκονται στην εφαρμογή Ιστού με όνομα jdbc. Για περισσότερες πληροφορίες σχετικά με τη δημιουργία και τη χρήση εφαρμογών Ιστού δείτε την Ενότητα 2.11.

Λίστα 17.3 NorthwindServlet.java

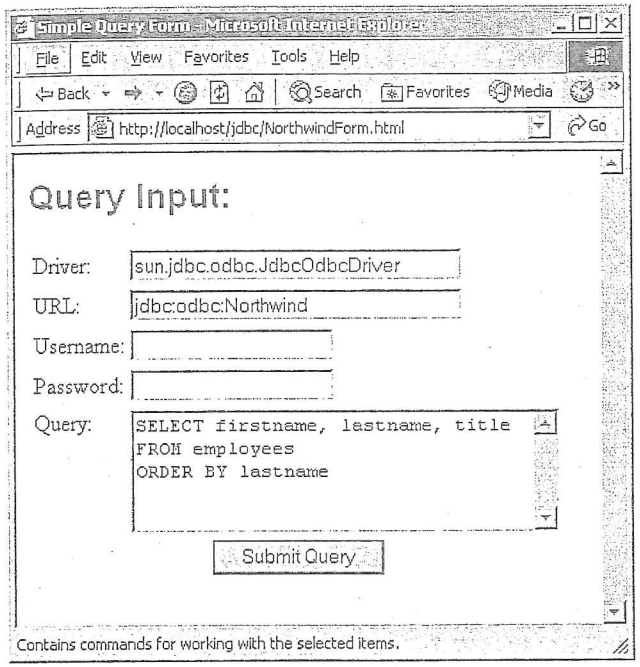
```
package coreservlets;

import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Μια απλή μικροϋπηρεσία που συνδέεται με βάση δεδομένων και
 *  παρουσιάζει τα αποτελέσματα ενός ερωτήματος σε έναν πίνακα HTML.
 *  Το πρόγραμμα οδήγησης, το URL, το όνομα χρήστη, ο κωδικός πρόσβασης,
 *  και το ερώτημα λαμβάνονται από τις παραμέτρους εισόδου της φόρμας.
 */

public class NorthwindServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\"\n";
    String title = "Northwind Results";
    out.print(docType +
              "<HTML>\n" +
              "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
              "<BODY BGCOLOR=\"#FDF5E6\"><CENTER>\n" +
              "<H1>Database Results</H1>\n");
    String driver = request.getParameter("driver");
    String url = request.getParameter("url");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String query = request.getParameter("query");
    showTable(driver, url, username, password, query, out);
    out.println("</CENTER></BODY></HTML>");
  }
```

Λίστα 17.3 NorthwindServlet.java *(συνέχεια)*

```
public void showTable(String driver, String url,
                      String username, String password,
                      String query, PrintWriter out) {
  try {
    // Φόρτωση του προγράμματος οδήγησης της βάσης δεδομένων,
    // αν δεν έχει ήδη φορτωθεί.
    Class.forName(driver);
    // Δρομολόγηση της δικτυακής σύνδεσης με τη βάση δεδομένων.
    Connection connection =
      DriverManager.getConnection(url, username, password);
    // Αναζήτηση πληροφοριών για τη βάση δεδομένων συνολικά.
    DatabaseMetaData dbMetaData = connection.getMetaData();
    out.println("<UL>");
    String productName =
      dbMetaData.getDatabaseProductName();
    String productVersion =
      dbMetaData.getDatabaseProductVersion();
    out.println("  <LI><B>Database:</B> " + productName +
                "  <LI><B>Version:</B> " + productVersion +
                "</UL>");
    Statement statement = connection.createStatement();
    // Αποστολή του ερωτήματος στη βάση δεδομένων
    // και αποθήκευση των αποτελεσμάτων.
    ResultSet resultSet = statement.executeQuery(query);
    // Εκτύπωση των αποτελεσμάτων.
    out.println("<TABLE BORDER=1>");
    ResultSetMetaData resultSetMetaData =
      resultSet.getMetaData();
    int columnCount = resultSetMetaData.getColumnCount();
    out.println("<TR>");
    // Ο αριθμοδείκτης στηλών ξεκινά από το 1 (όπως στην SQL),
    // και όχι από το 0 (όπως στη Java).
    for(int i=1; i <= columnCount; i++) {
      out.print("<TH>" + resultSetMetaData.getColumnName(i));
    }
    out.println();
    // Προσπέλαση κάθε γραμμής στο σύνολο των αποτελεσμάτων.
    while(resultSet.next()) {
      out.println("<TR>");
      // Προσπέλαση της γραμμής και ανάκτηση των δεδομένων από κάθε
      // κελί στήλης σε μορφή String.
      for(int i=1; i <= columnCount; i++) {
        out.print("<TD>" + resultSet.getString(i));
      }
      out.println();
    }
    out.println("</TABLE>");
    connection.close();
  } catch(ClassNotFoundException cnfe) {
    System.err.println("Error loading driver: " + cnfe);
```


Λίστα 17.3 NorthwindServlet.java *(συνέχεια)*

```
    } catch(SQLException sqle) {
      System.err.println("Error connecting: " + sqle);
    } catch(Exception ex) {
      System.err.println("Error with input: " + ex);
    }
  }
}
```



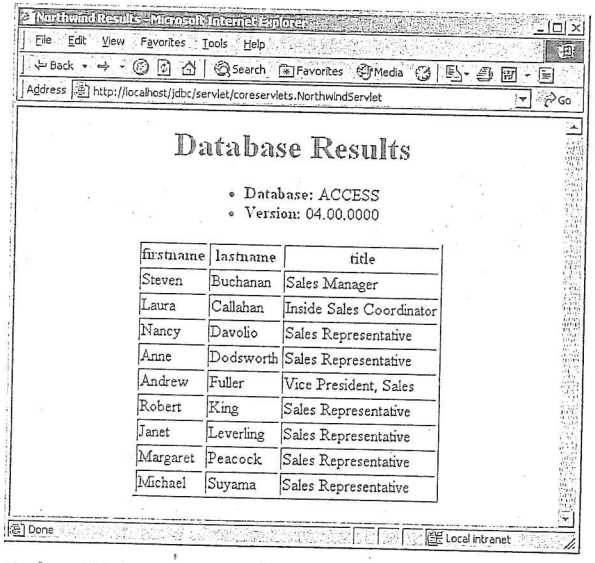
Εικόνα 17-3 NorthwindForm.html: εμπροσθοφυλακή της μικροϋπηρεσίας που εκτελεί ερώτημα στη βάση δεδομένων Northwind.

Λίστα 17.4 NorthwindForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Simple Query Form</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
```

Λίστα 17.4 NorthwindForm.html *(συνέχεια)*

```
      TYPE="text/css">
</HEAD>
<BODY>
<H2>Query Input:</H2>
<FORM ACTION="/jdbc/servlet/coreservlets.NorthwindServlet"
      METHOD="POST">
<TABLE>
  <TR><TD>Driver:
      <TD><INPUT TYPE="TEXT" NAME="driver"
                 VALUE="sun.jdbc.odbc.JdbcOdbcDriver" SIZE="35">
  <TR><TD>URL:
      <TD><INPUT TYPE="TEXT" NAME="url"
                 VALUE="jdbc:odbc:Northwind" SIZE="35">
  <TR><TD>Username:
      <TD><INPUT TYPE="TEXT" NAME="username">
  <TR><TD>Password:
      <TD><INPUT TYPE="PASSWORD" NAME="password">
  <TR><TD VALIGN="TOP">Query:
      <TD><TEXTAREA ROWS="5" COLS="35" NAME="query"></TEXTAREA>
  <TR><TD COLSPAN="2" ALIGN="CENTER"><INPUT TYPE="SUBMIT">
</TABLE>
</FORM>
</BODY></HTML>
```



Εικόνα 17-4 Αποτέλεσμα του ερωτήματος στη βάση δεδομένων Northwind.

Στο προηγούμενο παράδειγμα, ο πίνακας HTML κατασκευάστηκε από τα αποτελέσματα του ερωτήματος μέσα σε μια μικροϋπηρεσία. Στο βιβλίο More Servlets and JavaServer Pages παρουσιάζουμε διάφορες προσαρμοσμένες ετικέτες για την παραγωγή του πίνακα HTML από τα αποτελέσματα του ερωτήματος μέσα στην ίδια τη σελίδα JSP. Επιπρόσθετα, αν το μοντέλο ανάπτυξης που ακολουθείτε προτιμά τις σελίδες JSP, η Τυπική Βιβλιοθήκη Ετικετών JSP (JSP Standard Tag Library, JSTL) παρέχει την ενέργεια `sql:query` για την εκτέλεση ερωτημάτων σε μια βάση δεδομένων και την αποθήκευση των αποτελεσμάτων του ερωτήματος σε μια μεταβλητή προσδιορισμένης εμβέλειας, με στόχο την επεξεργασία στη σελίδα JSP. Η JSTL εξετάζεται στο βιβλίο More Servlets and JavaServer Pages.

17.3 Προσπέλαση βάσεων δεδομένων με βοηθητικές κλάσεις JDBC

Σε αυτή την ενότητα θα παρουσιάσουμε μερικές βοηθητικές κλάσεις τις οποίες χρησιμοποιούμε σε ολόκληρο το κεφάλαιο έτσι ώστε να απλοποιήσουμε τον κώδικα. Αυτές οι κλάσεις παρέχουν βασικές λειτουργίες για τη φόρτωση προγραμμάτων οδήγησης και τη δημιουργία συνδέσεων με βάσεις δεδομένων.

Για παράδειγμα, η κλάση `DriverUtilities` (Λίστα 17.5) απλοποιεί τη δόμηση του URL με το οποίο γίνεται η σύνδεση με μια βάση δεδομένων. Για να δομήσετε ένα URL για μια βάση δεδομένων MySQL, το οποίο έχει τη μορφή

```
String url = "jdbc:mysql://host:3306/dbname";
```

θα πρέπει πρώτα να φορτώσετε τα δεδομένα του κατασκευαστή με κλήση της μεθόδου `loadDriver`. Κατόπιν καλείτε τη μέθοδο `makeURL` για να δομήσετε το URL, όπως στο ακόλουθο παράδειγμα

```
DriverUtilities.loadDrivers();
String url =
  DriverUtilities.makeURL(host, dbname, DriverUtilities.MYSQL);
```

όπου ο υπολογιστής υπηρεσίας (host), το όνομα της βάσης δεδομένων, και ο κατασκευαστής καθορίζονται δυναμικά, ως ορίσματα. Με αυτόν τον τρόπο το URL της βάσης δεδομένων δεν χρειάζεται να γραφτεί "κυριολεκτικά" στα παραδείγματα αυτού του κεφαλαίου. Το πιο σημαντικό, όμως, είναι ότι μπορείτε απλώς να προσθέσετε πληροφορίες σχετικά με τη βάση δεδομένων σας στη μέθοδο `loadDriver` της κλάσης `DriverUtilities` (καθώς και μια σταθερά που να αναφέρεται στο πρόγραμμα οδήγησής σας, αν το επιθυμείτε). Αφού το κάνετε αυτό, τα παραδείγματα αυτού του κεφαλαίου θα πρέπει να λειτουργούν στο περιβάλλον σας.

Για να δούμε ένα ακόμα παράδειγμα, η κλάση `ConnectionInfoBean` (Λίστα 17.9) παρέχει μια βοηθητική μέθοδο, την `getConnection`, για τη λήψη ενός αντικειμένου `Connection` με μια βάση δεδομένων. Έτσι, για να πάρετε ένα αντικείμενο σύνδεσης με τη βάση δεδομένων, αντικαθιστάτε τις παρακάτω γραμμές κώδικα

```
Connection connection = null;
try {
  Class.forName(driver);
  connection = DriverManager.getConnection(url, username,
                                           password);
} catch(ClassNotFoundException cnfe) {
  System.err.println("Error loading driver: " + cnfe);
} catch(SQLException sqle) {
  System.err.println("Error connecting: " + sqle);
}
```

με

```
Connection connection =
  ConnectionInfoBean.getConnection(driver, url,
                                   username, password);
```

Αν προκύψει κάποια εξαίρεση `SQLException` κατά την πραγματοποίηση της σύνδεσης, επιστρέφεται τιμή `null`.

Σε αυτή την ενότητα ορίζουμε τέσσερις βοηθητικές κλάσεις.

1. **`DriverUtilities`**

   Αυτή η κλάση, που φαίνεται στη Λίστα 17.5, φορτώνει ρητά κωδικοποιημένες πληροφορίες προγραμμάτων οδήγησης για διάφορους κατασκευαστές βάσεων δεδομένων. Κατόπιν παρέχει μεθόδους για τη λήψη της κλάσης προγράμματος οδήγησης για κάποιον κατασκευαστή (`getDriver`) και δημιουργεί ένα URL (`makeURL`) με βάση τον υπολογιστή υπηρεσίας, το όνομα της βάσης δεδομένων, και τον κατασκευαστή. Παρέχουμε πληροφορίες προγράμματος οδήγησης για βάσεις δεδομένων Microsoft Access, MySQL, και Oracle9i — αλλά μπορείτε εύκολα να ενημερώσετε την κλάση σύμφωνα με το δικό σας περιβάλλον.

2. **`DriverUtilities2`**

   Αυτή η κλάση, που φαίνεται στη Λίστα 17.6, επεκτείνει την κλάση `DriverUtilities` (Λίστα 17.5) και υποσκελίζει τη μέθοδο `loadDrivers` έτσι ώστε να λαμβάνει τις πληροφορίες προγραμμάτων οδήγησης από ένα αρχείο XML. Ένα αντιπροσωπευτικό αρχείο XML είναι το drivers.xml, που φαίνεται στη Λίστα 17.7.

3. **`DriverInfoBean`**

   Η κλάση `DriverInfoBean`, που φαίνεται στη Λίστα 17.8, ενθυλακώνει πληροφορίες για το πρόγραμμα οδήγησης ενός συγκεκριμένου κατασκευαστή (οι οποίες χρησιμοποιούνται από την κλάση `DriverUtilities`, Λίστα 17.5). Ο κόκκος περιέχει μια λέξη-κλειδί (το όνομα του κατασκευαστή), μια σύντομη περιγραφή του προγράμματος οδήγησης, το όνομα της κλάσης του προγράμματος οδήγησης, και το URL για τη σύνδεση με μια βάση δεδομένων.

4. **`ConnectionInfoBean`**

   Αυτή η κλάση, που φαίνεται στη Λίστα 17.9, ενθυλακώνει πληροφορίες για τη σύνδεση με μια *συγκεκριμένη* βάση δεδομένων. Ο κόκκος ενθυλακώνει ένα όνομα για τη σύνδεση, μια σύντομη περιγραφή της σύνδεσης, την κλάση του προγράμματος οδήγησης, το URL σύνδεσης με τη βάση δεδομένων, το όνομα χρήστη, και τον κωδικό πρόσβασης. Επιπλέον, ο κόκκος παρέχει τη μέθοδο `getConnection` για την άμεση λήψη ενός αντικειμένου `Connection` με μια βάση δεδομένων.

Λίστα 17.5 DriverUtilities.java

```
package coreservlets;

import java.io.*;
import java.sql.*;
import java.util.*;
import coreservlets.beans.*;

/** Απλές βοηθητικές κλάσεις για τη δημιουργία συνδέσεων JDBC με
 *  βάσεις δεδομένων διαφόρων κατασκευαστών. Τα προγράμματα οδήγησης
 *  που φαίνονται εδώ είναι κωδικοποιημένα μέσα στο πρόγραμμα και αφορούν
 *  ειδικά την τοπική μας διευθέτηση. Μπορείτε είτε να τροποποιήσετε
 *  τη μέθοδο loadDrivers και να μεταγλωττίσετε πάλι το πρόγραμμα, ή
 *  να χρησιμοποιήσετε την εντολή <CODE>DriverUtilities2</CODE>
 *  για να πάρετε τις πληροφορίες του προγράμματος οδήγησης για κάθε
 *  κατασκευαστή από ένα αρχείο XML.
 */

public class DriverUtilities {
  public static final String MSACCESS = "MSACCESS";
  public static final String MYSQL = "MYSQL";
  public static final String ORACLE = "ORACLE";

  // Προσθήκη εδώ μιας σταθεράς για αναφορά στη βάση δεδομένων σας...

  protected static Map driverMap = new HashMap();

  /** Φόρτωση των πληροφοριών του κατασκευαστή για το πρόγραμμα οδήγησης.
   *  Εδώ έχουμε περάσει ως "κυριολεκτικό" μέσα στον κώδικα τις
   *  πληροφορίες προγράμματος οδήγησης οι οποίες αφορούν συγκεκριμένα
   *  την τοπική μας διευθέτηση. Τροποποιήστε τις τιμές σύμφωνα με τη
   *  δική σας διευθέτηση. Εναλλακτικά, μπορείτε να χρησιμοποιήσετε
   *  την εντολή <CODE>DriverUtilities2</CODE> για να φορτώσετε τις
   *  πληροφορίες του προγράμματος οδήγησης από ένα αρχείο XML.
   *  <P>
   *  Κάθε κατασκευαστής αντιπροσωπεύεται από μια ετικέτα
   *  <CODE>DriverInfoBean</CODE> που ορίζει το όνομα του κατασκευαστή
   *  (λέξη-κλειδί), μια περιγραφή, την κλάση προγράμματος οδήγησης, και
   *  το URL. Ο κόκκος αποθηκεύεται σε ένα χάρτη προγραμμάτων οδήγησης.
   *  Το όνομα του κατασκευαστή χρησιμοποιείται ως λέξη-κλειδί για την
   *  ανάκτηση των πληροφοριών.
```

Λίστα 17.5 DriverUtilities.java *(συνέχεια)*

```
   *  <P>
   *  Η μεταβλητή url θα πρέπει να περιέχει τα δεσμευτικά θέσης
   *  [$host] και [$dbName] που θα αντικατασταθούν από τα <I>host</I>
   *  και <I>database name</I> στην ετικέτα <CODE>makeURL</CODE>.
   */

  public static void loadDrivers() {
    String vendor, description, driverClass, url;
    DriverInfoBean info = null;

    // MSAccess
    vendor = MSACCESS;
    description = "MS Access 4.0";
    driverClass = "sun.jdbc.odbc.JdbcOdbcDriver";
    url = "jdbc:odbc:[$dbName]";
    info = new DriverInfoBean(vendor, description,
                              driverClass, url);
    addDriverInfoBean(info);

    // MySQL
    vendor = MYSQL;
    description = "MySQL Connector/J 3.0";
    driverClass = "com.mysql.jdbc.Driver";
    url = "jdbc:mysql://[$host]:3306/[$dbName]";
    info = new DriverInfoBean(vendor, description,
                              driverClass, url);
    addDriverInfoBean(info);

    // Oracle
    vendor = ORACLE;
    description = "Oracle9i Database";
    driverClass = "oracle.jdbc.driver.OracleDriver";
    url = "jdbc:oracle:thin:@[$host]:1521:[$dbName]";
    info = new DriverInfoBean(vendor, description,
                              driverClass, url);
    addDriverInfoBean(info);

    // Εδώ προσθέστε πληροφορίες για τη βάση δεδομένων σας...
  }

  /** Προσθήκη πληροφοριών (<CODE>DriverInfoBean</CODE>) στο χάρτη
   *  των διαθέσιμων προγραμμάτων οδήγησης για το νέο κατασκευαστή.
   */

  public static void addDriverInfoBean(DriverInfoBean info) {
    driverMap.put(info.getVendor().toUpperCase(), info);
  }

  /** Προσδιορίζει αν ο κατασκευαστής αντιπροσωπεύεται από τις
   *  πληροφορίες προγράμματος οδήγησης που έχουν φορτωθεί.
   */
```


Λίστα 17.5 DriverUtilities.java *(συνέχεια)*

```
  public static boolean isValidVendor(String vendor) {
    DriverInfoBean info =
      (DriverInfoBean)driverMap.get(vendor.toUpperCase());
    return(info != null);
  }

  /** Δομεί μια διεύθυνση URL στη μορφή που χρειάζονται τα προγράμματα
   *  οδήγησης βάσεων δεδομένων. Στην κατασκευή του τελικού URL, οι
   *  λέξεις-κλειδιά [$host] και [$dbName] στο URL (που έχουν ληφθεί από
   *  την παράσταση <CODE>DriverInfoBean</CODE> για τον κατασκευαστή)
   *  αντικαθίστανται κατάλληλα με τα ορίσματα host και dbName
   *  της μεθόδου.
   */

  public static String makeURL(String host, String dbName,
                               String vendor) {
    DriverInfoBean info =
      (DriverInfoBean)driverMap.get(vendor.toUpperCase());
    if (info == null) {
      return(null);
    }
    StringBuffer url = new StringBuffer(info.getURL());
    DriverUtilities.replace(url, "[$host]", host);
    DriverUtilities.replace(url, "[$dbName]", dbName);
    return(url.toString());
  }

  /** Λήψη του πλήρους ονόματος του προγράμματος οδήγησης. */

  public static String getDriver(String vendor) {
    DriverInfoBean info =
      (DriverInfoBean)driverMap.get(vendor.toUpperCase());
    if (info == null) {
      return(null);
    } else {
      return(info.getDriverClass());
    }
  }

  /** Εκτέλεση αλφαριθμητικής αντικατάστασης, όπου η πρώτη ταύτιση
   *  ("match") αντικαθίσταται από τη νέα τιμή ("value").
   */

  private static void replace(StringBuffer buffer,
                              String match, String value) {
    int index = buffer.toString().indexOf(match);
    if (index > 0) {
      buffer.replace(index, index + match.length(), value);
    }
  }
}
```

Στην κλάση DriverUtilities, οι πληροφορίες για το πρόγραμμα οδήγησης του κάθε κατασκευαστή (Microsoft Access, MySQL, και Oracle9i) κωδικοποιούνται ρητά στο πρόγραμμα. Αν χρησιμοποιείτε κάποια διαφορετική βάση δεδομένων, θα πρέπει να τροποποιήσετε την κλάση DriverUtilities έτσι ώστε να συμπεριλάβετε τις πληροφορίες του δικού σας προγράμματος οδήγησης και μετά να μεταγλωττίσετε ξανά τον κώδικα. Επειδή αυτή η προσέγγιση δεν είναι τόσο εύχρηστη, συμπεριλάβαμε ένα δεύτερο πρόγραμμα, το DriverUtilities2 στη Λίστα 17.6, το οποίο διαβάζει τις πληροφορίες προγράμματος οδήγησης από ένα αρχείο XML. Έτσι, για να προσθέσετε ένα νέο κατασκευαστή βάσεων δεδομένων στο πρόγραμμά σας, διορθώνετε απλώς το αρχείο XML. Στη Λίστα 17.7 φαίνεται ένα παράδειγμα αρχείου XML, το drivers.xml.

Όταν χρησιμοποιείτε την κλάση DriverUtilities2 σε μια εφαρμογή γραμμής διαταγών, τοποθετείτε το αρχείο περιγραφής προγραμμάτων οδήγησης (εδώ το αρχείο drivers.xml) στον κατάλογο εργασίας από τον οποίο ξεκινάτε την εφαρμογή. Στη συνέχεια καλείτε τη μέθοδο loadDrivers με το πλήρες όνομα αρχείου (συμπεριλαμβανομένης της διαδρομής).

Για μια εφαρμογή Ιστού, σας συνιστούμε να τοποθετήσετε το αρχείο drivers.xml στον κατάλογο WEB-INF. Μια καλή ιδέα είναι να καθορίσετε το όνομα αρχείου ως παράμετρο απόδοσης αρχικής τιμής του πλαισίου εφαρμογής στο αρχείο web.xml (για λεπτομέρειες, δείτε το Κεφάλαιο σχετικά με το αρχείο web.xml στο βιβλίο More Servlets and JavaServer Pages). Θυμηθείτε επίσης ότι από το πλαίσιο μικροϋπηρεσίας μπορείτε να χρησιμοποιήσετε τη μέθοδο getRealPath για να προσδιορίσετε τη φυσική διαδρομή προς ένα αρχείο σε σχέση με τον κατάλογο της εφαρμογής Ιστού, όπως φαίνεται στο ακόλουθο απόσπασμα κώδικα.

```
ServletContext context = getServletContext();
String path = context.getRealPath("/WEB-INF/drivers.xml");
```

Το JDK 1.4 περιλαμβάνει όλες τις απαραίτητες κλάσεις για τη συντακτική ανάλυση του εγγράφου XML (του drivers.xml). Αν χρησιμοποιείτε το JDK 1.3 ή κάποια προγενέστερη έκδοση, θα χρειαστεί να κατεβάσετε και να εγκαταστήσετε ένα πρόγραμμα συντακτικής ανάλυσης (parser) SAX και DOM. Το Xerces-J της Apache είναι ένα εξαιρετικό πρόγραμμα συντακτικής ανάλυσης, και είναι διαθέσιμο από τη διεύθυνση http://xml.apache.org/xerces2-j/. Οι περισσότεροι διακομιστές εφαρμογών Ιστού συνοδεύονται ήδη από προγράμματα συντακτικής ανάλυσης αρχείων XML, έτσι πιθανότατα δεν θα χρειαστεί να κατεβάσετε το Xerces-J. Ελέγξτε την τεκμηρίωση του κατασκευαστή για να διαπιστώσετε πού βρίσκονται τα αρχεία συντακτικής ανάλυσης και να τα συμπεριλάβετε στη μεταβλητή CLASSPATH, έτσι ώστε να μπορεί να μεταγλωττιστεί η εφαρμογή σας. Για παράδειγμα, ο Tomcat 4.x περιλαμβάνει τα αρχεία JAR συντακτικής ανάλυσης (xercesImpl.jar και xmlParserAPI.jar) στον κατάλογο *κατάλογος_εγκατάστασης*/common/endorsed.

Ας σημειωθεί ότι, αν χρησιμοποιείτε μικροϋπηρεσίες 2.4 (JSP 2.0) σε ένα διακομιστή που είναι πλήρως συμβατός με τη J2EE-1.4, είναι σίγουρο ότι έχετε το JDK 1.4 ή μεταγενέστερο.

Λίστα 17.6 DriverUtilities2.java

```
package coreservlets;

import java.io.*;
import java.util.*;
```


Λίστα 17.6 DriverUtilities2.java *(συνέχεια)*

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import coreservlets.beans.*;
/** Επεκτείνει το αντικείμενο <CODE>DriverUtilities</CODE> με στόχο την
 *  υποστήριξη της φόρτωσης πληροφοριών για προγράμματα οδήγησης διαφόρων
 *  κατασκευαστών βάσεων δεδομένων από ένα αρχείο XML (το προεπιλεγμένο
 *  αρχείο είναι το drivers.xml). Για την ανάγνωση του αρχείου XML
 *  χρησιμοποιούνται η DOM και η JAXP.  Η μορφή του αρχείου XML είναι:
 *  <P>
 *  <PRE>
 *    &lt;drivers&gt;
 *      &lt;driver&gt;
 *        &lt;vendor&gt;ORACLE&lt;/vendor&gt;
 *        &lt;description&gt;Oracle&lt;/description&gt;
 *        &lt;driver-class&gt;
 *          oracle.jdbc.driver.OracleDriver
 *        &lt;/driver-class&gt;
 *        &lt;url&gt;
 *          jdbc:oracle:thin:@[$host]:1521:[$dbName]
 *        &lt;/url&gt;
 *      &lt;/driver&gt;
 *      ...
 *    &lt;drivers&gt;
 *  </PRE>
 *  <P>
 *  Το στοιχείο url θα πρέπει να περιέχει τα δεσμευτικά θέσης
 *  [$host] και [$dbName] που θα αντικατασταθούν από τον υπολογιστή
 *  υπηρεσίας και το όνομα της βάσης δεδομένων στη μέθοδο makeURL.
 */

public class DriverUtilities2 extends DriverUtilities {
  public static final String DEFAULT_FILE = "drivers.xml";

  /** Φόρτωση πληροφοριών προγράμματος οδήγησης από το προεπιλεγμένο
   *  αρχείο XML, το αρχείο drivers.xml.
   */

  public static void loadDrivers() {
    DriverUtilities2.loadDrivers(DEFAULT_FILE);
  }

  /** Φόρτωση πληροφοριών προγράμματος οδήγησης από το καθορισμένο αρχείο
   *  XML. Κάθε κατασκευαστής αντιπροσωπεύεται από ένα αντικείμενο
   *  <CODE>DriverInfoBean</CODE> και αποθηκεύεται στο χάρτη, με το όνομα
   *  του κατασκευαστή ως κλειδί. Χρησιμοποιήστε αυτή τη μέθοδο αν πρέπει
   *  να φορτώσετε κάποιο άλλο αρχείο προγράμματος οδήγησης, αντί του
   *  προεπιλεγμένου drivers.xml.
   */
```

Λίστα 17.6 DriverUtilities2.java *(συνέχεια)*

```
  public static void loadDrivers(String filename) {
    File file = new File(filename);
    try {
      InputStream in = new FileInputStream(file);
      DocumentBuilderFactory builderFactory =
        DocumentBuilderFactory.newInstance();
      DocumentBuilder builder =
        builderFactory.newDocumentBuilder();
      Document document = builder.parse(in);
      document.getDocumentElement().normalize();
      Element rootElement = document.getDocumentElement();
      NodeList driverElements =
        rootElement.getElementsByTagName("driver");
      // Δόμηση του DriverInfoBean για κάθε κατασκευαστή
      for(int i=0; i<driverElements.getLength(); i++) {
        Node node = driverElements.item(i);
        DriverInfoBean info =
          DriverUtilities2.createDriverInfoBean(node);
        if (info != null) {
          addDriverInfoBean(info);
        }
      }
    } catch(FileNotFoundException fnfe) {
      System.err.println("Can't find " + filename);
    } catch(IOException ioe) {
      System.err.println("Problem reading file: " + ioe);
    } catch(ParserConfigurationException pce) {
      System.err.println("Can't create DocumentBuilder");
    } catch(SAXException se) {
      System.err.println("Problem parsing document: " + se);
    }
  }

  /** Δόμηση ενός αντικειμένου DriverInfoBean από έναν κόμβο
   *  XML DOM που αναπαριστάνει το πρόγραμμα οδήγησης του
   *  κατασκευαστή με τη μορφή:
   *  <P>
   *  <PRE>
   *    &lt;driver&gt;
   *      &lt;vendor&gt;ORACLE&lt;/vendor&gt;
   *      &lt;description&gt;Oracle&lt;/description&gt;
   *      &lt;driver-class&gt;
   *         oracle.jdbc.driver.OracleDriver
   *      &lt;/driver-class&gt;
   *      &lt;url&gt;
   *        jdbc:oracle:thin:@[$host]:1521:[$dbName]
   *      &lt;/url&gt;
   *    &lt;/driver&gt;
   *  </PRE>
   */
```


Λίστα 17.6 DriverUtilities2.java *(συνέχεια)*

```
  public static DriverInfoBean createDriverInfoBean(Node node) {
    Map map = new HashMap();
    NodeList children = node.getChildNodes();
    for(int i=0; i<children.getLength(); i++) {
      Node child = children.item(i);
      String nodeName = child.getNodeName();
      if (child instanceof Element) {
        Node textNode = child.getChildNodes().item(0);
        if (textNode != null) {
          map.put(nodeName, textNode.getNodeValue());
        }
      }
    }
    return(new DriverInfoBean((String)map.get("vendor"),
                              (String)map.get("description"),
                              (String)map.get("driver-class"),
                              (String)map.get("url")));
  }
}
```

Λίστα 17.7 drivers.xml

```
<?xml version="1.0"?>
<!--
Χρησιμοποιείται από την κλάση DriverUtilities2. Εδώ διαμορφώνετε τις
πληροφορίες σχετικά με το διακομιστή της βάσης δεδομένων σας σε XML.
Για να προσθέσετε ένα πρόγραμμα οδήγησης, συμπεριλαμβάνετε μια λέξη-κλειδί
για τον κατασκευαστή, μια περιγραφή, την κλάση του προγράμματος οδήγησης,
και το URL. Για γενική χρήση, ο υπολογιστής υπηρεσίας και το όνομα της
βάσης δεδομένων δεν θα πρέπει να συμπεριλαμβάνονται στο URL. Απαιτείται
ειδική σημειογραφία για τον υπολογιστή υπηρεσίας και το όνομα της
βάσης δεδομένων. Χρησιμοποιήστε το [$host] ως δεσμευτικό θέση για το
διακομιστή του υπολογιστή υπηρεσίας, και το [$dbName] ως δεσμευτικό θέση
για το όνομα της βάσης δεδομένων. Καθορίστε τον πραγματικό υπολογιστή
υπηρεσίας και το όνομα της βάσης δεδομένων όταν καλείτε τη μέθοδο
makeUrl (DriverUtilities). Έτσι τα κατάλληλα αλφαριθμητικά θα
αντικαταστήσουν τα δεσμευτικά θέσης [$host] και [$dbName] πριν επιστραφεί
η διεύθυνση URL.
-->
<drivers>
  <driver>
    <vendor>MSACCESS</vendor>
    <description>MS Access</description>
    <driver-class>sun.jdbc.odbc.JdbcOdbcDriver</driver-class>
    <url>jdbc:odbc:[$dbName]</url>
  </driver>
```

Λίστα 17.7 drivers.xml *(συνέχεια)*

```
  <driver>
    <vendor>MYSQL</vendor>
    <description>MySQL Connector/J 3.0</description>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <url>jdbc:mysql://[$host]:3306/[$dbName]</url>
  </driver>
  <driver>
    <vendor>ORACLE</vendor>
    <description>Oracle</description>
    <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
    <url>jdbc:oracle:thin:@[$host]:1521:[$dbName]</url>
  </driver>
</drivers>
```

Λίστα 17.8 DriverInfoBean.java

```
package coreservlets.beans;

/** Πληροφορίες προγράμματος οδήγησης για έναν κατασκευαστή. Ορίζει
 *  τη λέξη-κλειδί για τον κατασκευαστή, την περιγραφή, την κλάση του
 *  προγράμματος οδήγησης, και τη δομή της διεύθυνσης URL για τη σύνδεση
 *  με μια βάση δεδομένων.
 */

public class DriverInfoBean {
  private String vendor;
  private String description;
  private String driverClass;
  private String url;

  public DriverInfoBean(String vendor,
                        String description,
                        String driverClass,
                        String url) {
    this.vendor = vendor;
    this.description = description;
    this.driverClass = driverClass;
    this.url = url;
  }
  public String getVendor() {
    return(vendor);
  }

  public String getDescription() {
    return(description);
  }
```


Λίστα 17.8 DriverInfoBean.java *(συνέχεια)*

```
  public String getDriverClass() {
    return(driverClass);
  }

  public String getURL() {
    return(url);
  }
}
```

Λίστα 17.9 ConnectionInfoBean.java

```
package coreservlets.beans;

import java.sql.*;

/** Αποθηκεύει πληροφορίες για τη δημιουργία μιας σύνδεσης JDBC
 *  σε μια βάση δεδομένων. Οι πληροφορίες περιλαμβάνουν τα εξής:
 *  <UL>
 *    <LI>όνομα σύνδεσης
 *    <LI>περιγραφή της σύνδεσης
 *    <LI>όνομα κλάσης του προγράμματος οδήγησης
 *    <LI>URL για σύνδεση με τον υπολογιστή υπηρεσίας
 *    <LI>όνομα χρήστη
 *    <LI>κωδικός πρόσβασης
 *  </UL>
*/

public class ConnectionInfoBean {
  private String connectionName;
  private String description;
  private String driver;
  private String url;
  private String username;
  private String password;

  public ConnectionInfoBean() { }

  public ConnectionInfoBean(String connectionName,
                            String description,
                            String driver,
                            String url,
                            String username,
                            String password) {
    setConnectionName(connectionName);
    setDescription(description);
    setDriver(driver);
    setURL(url);
    setUsername(username);
    setPassword(password);
```

Λίστα 17.9 ConnectionInfoBean.java *(συνέχεια)*

```
  }

  public void setConnectionName(String connectionName) {
    this.connectionName = connectionName;
  }

  public String getConnectionName() {
    return(connectionName);
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public String getDescription() {
    return(description);
  }

  public void setDriver(String driver) {
    this.driver = driver;
  }

  public String getDriver() {
    return(driver);
  }

  public void setURL(String url) {
    this.url = url;
  }

  public String getURL() {
    return(url);
  }

  public void setUsername(String username) {
    this.username = username;
  }

  public String getUsername() {
    return(username);
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public String getPassword() {
    return(password);
  }
```


Λίστα 17.9 ConnectionInfoBean.java *(συνέχεια)*

```
  public Connection getConnection() {
    return(getConnection(driver, url, username, password));
  }

  /** Δημιουργεί μια σύνδεση JDBC ή επιστρέφει τιμή null αν
   *  παρουσιαστεί κάποιο πρόβλημα.
   */

  public static Connection getConnection(String driver,
                                         String url,
                                         String username,
                                         String password) {
    try {
      Class.forName(driver);
      Connection connection =
        DriverManager.getConnection(url, username,
                                    password);
      return(connection);
    } catch(ClassNotFoundException cnfe) {
      System.err.println("Error loading driver: " + cnfe);
      return(null);
    } catch(SQLException sqle) {
      System.err.println("Error connecting: " + sqle);
      return(null);
    }
  }
}
```

17.4 Χρήση προκατασκευασμένων εντολών

Αν πρόκειται να εκτελέσετε παρόμοιες εντολές SQL πολλές φορές, η χρήση των παραμετρικών (parameterized) ή "προκατασκευασμένων" εντολών μπορεί να αποδειχθεί πιο αποτελεσματική από την εκτέλεση ενός νέου ερωτήματος κάθε φορά. Η ιδέα είναι να χρησιμοποιηθεί μία παραμετρική εντολή σε μια τυποποιημένη μορφή που αποστέλλεται στη βάση δεδομένων για μεταγλώττιση, πριν να χρησιμοποιηθεί στην πραγματικότητα. Χρησιμοποιείτε ένα αγγλικό ερωτηματικό για να υποδείξετε τις θέσεις όπου στην εντολή θα χρησιμοποιηθεί κάποια τιμή. Κάθε φορά που χρησιμοποιείτε την προκατασκευασμένη εντολή, απλώς αντικαθιστάτε τις σημειωμένες παραμέτρους χρησιμοποιώντας μια κλήση `setXxx` που αντιστοιχεί στην καταχώριση την οποία θέλετε να ορίσετε (με αριθμοδείκτες που ξεκινούν από το 1), καθώς και τον τύπο της παραμέτρου (για παράδειγμα, `setInt` ή `setString`). Κατόπιν χρησιμοποιείτε τη μέθοδο `executeQuery` (αν θέλετε να σας επιστραφεί ένα αντικείμενο `ResultSet`) ή τη μέθοδο `execute`/`executeUpdate` για να τροποποιήσετε τον πίνακα δεδομένων, όπως θα κάνατε και με τις κανονικές εντολές.

Για παράδειγμα, στην Ενότητα 18.5 δημιουργούμε τον πίνακα `music` στον οποίο συνοψίζονται οι τιμές και η διαθεσιμότητα των δίσκων από κονσέρτα συναυλιών για διάφορους συνθέτες

κλασικής μουσικής. Υποθέστε ότι, για μια επικείμενη πώληση, θέλετε να αλλάξετε τις τιμές όλων των δίσκων στον πίνακα `music`. Θα μπορούσατε να κάνετε κάτι ανάλογο με το παρακάτω.

```
Connection connection =
  DriverManager.getConnection(url, username, password);
String template =
  "UPDATE music SET price = ? WHERE id = ?";
PreparedStatement statement =
  connection.prepareStatement(template);
float[] newPrices = getNewPrices();
int[] recordingIDs = getIDs();
for(int i=0; i<recordingIDs.length; i++) {
  statement.setFloat(1, newPrices[i]);  // Τιμή
  statement.setInt(2, recordingIDs[i]); // Αναγνωριστικό
  statement.execute();
}
```

Τα πλεονεκτήματα απόδοσης των προκατασκευασμένων εντολών μπορεί να ποικίλλουν σε πολύ μεγάλο βαθμό, ανάλογα με το πόσο καλά υποστηρίζει ο διακομιστής τα προμεταγλωττισμένα ερωτήματα και πόσο αποτελεσματικά γίνεται ο χειρισμός των νέων ερωτημάτων από το πρόγραμμα οδήγησης. Για παράδειγμα, η Λίστα 17.10 παρουσιάζει μια κλάση που στέλνει 100 διαφορετικά ερωτήματα σε μια βάση δεδομένων, χρησιμοποιώντας προκατασκευασμένες εντολές, και μετά επαναλαμβάνει τα ίδια 100 ερωτήματα χρησιμοποιώντας κανονικές εντολές. Από τη μία πλευρά, με ένα PC και μια γρήγορη σύνδεση LAN (100 Mbps) σε μια βάση δεδομένων Oracle9i, οι προκατασκευασμένες εντολές χρειάστηκαν κατά μέσο όρο περίπου το 62% του χρόνου που χρειάστηκαν τα νέα ερωτήματα· 0,61 δευτερόλεπτα για τα 100 ερωτήματα ως προκατασκευασμένες εντολές, και 0,99 δευτερόλεπτα για τα 100 ερωτήματα ως κανονικές εντολές (ο μέσος όρος υπολογίζεται από 5 δοκιμές). Από την άλλη πλευρά, με μια βάση δεδομένων MySQL (Connector/J3.0) οι χρόνοι των προκατασκευασμένων εντολών ήταν σχεδόν ίδιοι με εκείνους των κανονικών εντολών· με μια γρήγορη σύνδεση LAN παρατηρήθηκε μείωση στο χρόνο των ερωτημάτων κατά μόνο περίπου 8%. Για να εκτιμήσετε τη διαφορά στην απόδοση για τη δική σας διευθέτηση, κατεβάστε το αρχείο DriverUtilities.java από τη διεύθυνση http://www.coreservlets.com/, προσθέστε πληροφορίες σχετικά με τα δικά σας προγράμματα οδήγησης, και εκτελέστε το πρόγραμμα `PreparedStatements`. Για να δημιουργήσετε τον πίνακα `music`, δείτε την Ενότητα 18.5.

Προσέξτε όμως: η προκατασκευασμένη εντολή δεν εκτελείται πάντοτε πιο γρήγορα από μια κανονική εντολή SQL. Η βελτίωση της απόδοσης μπορεί να εξαρτάται από τη συγκεκριμένη εντολή SQL που εκτελείτε. Για μια πιο λεπτομερή ανάλυση της απόδοσης των προκατασκευασμένων εντολών στην Oracle, δείτε την ιστοσελίδα http://www.oreilly.com/catalog/jorajdbc/chapter/ch19.html.

Από την άλλη πλευρά, η απόδοση δεν είναι το μόνο πλεονέκτημα των προκατασκευασμένων εντολών. Ένα άλλο πλεονέκτημα είναι η ασφάλεια. Σας συνιστούμε να χρησιμοποιείτε πάντοτε μια προκατασκευασμένη εντολή ή μια αποθηκευμένη διαδικασία (δείτε την Ενότητα 17.5) για την ενημέρωση των τιμών μιας βάσης δεδομένων, όταν δέχεστε είσοδο από τους χρήστες μέσω κάποιας φόρμας HTML. Αυτή η προσέγγιση είναι σαφώς ανώτερη από την προσέγγιση της δόμησης μιας εντολής SQL μέσω της συνένωσης αλφαριθμητικών από τις τιμές εισόδου του χρήστη. Διαφορετικά, κάποιος έξυπνος επιτιθέμενος θα μπορούσε να υποβάλει τιμές φόρμας που να μοιάζουν με εντολές SQL, και μετά την εκτέλεσή τους ο επιτιθέμενος θα μπορούσε να αποκτήσει μη εξουσιοδοτημένη πρόσβαση ή να τροποποιήσει τη βάση δεδομένων. Αυτός ο κίνδυνος ασφαλείας συχνά αναφέρεται ως "Επίθεση Έγχυσης SQL" (SQL Injection Attack). Εκτός από την αποτροπή του κινδύνου από μια τέτοια επίθεση, η προκατασκευασμένη εντολή θα μπορεί να χειριστεί πιο σωστά τα εισαγωγικά που είναι ενσωματωμένα σε αλφαριθμητικά, καθώς και να χειριστεί δεδομένα που δεν είναι χαρακτήρες (για παράδειγμα, την αποστολή ενός σειριοποιημένου αντικειμένου στη βάση δεδομένων).

Η προσέγγιση του βιβλίου

Για να αποφύγετε την Επίθεση Έγχυσης SQL όταν δέχεστε δεδομένα από φόρμες HTML, χρησιμοποιήστε μια προκατασκευασμένη εντολή ή μια αποθηκευμένη διαδικασία για την ενημέρωση της βάσης δεδομένων.

Λίστα 17.10 PreparedStatements.java

```
package coreservlets;

import java.sql.*;
import coreservlets.beans.*;

/** Ένα παράδειγμα ελέγχου των χρονικών διαφορών που προκύπτουν από
 *  επαναλαμβανόμενα κανονικά ερωτήματα σε σύγκριση με τις κλήσεις
 *  προκατασκευασμένων εντολών. Αυτά τα αποτελέσματα θα διαφέρουν
 *  σημαντικά μεταξύ των διακομιστών και των προγραμμάτων οδήγησης
 *  βάσεων δεδομένων. Με τη δική μας διευθέτηση και τα δικά μας
 *  προγράμματα οδήγησης, οι προκατασκευασμένες εντολές στην Oracle9i
 *  χρειάστηκαν μόνο το 62% του χρόνου που χρειάστηκαν τα κανονικά
 *  ερωτήματα, ενώ στη MySQL οι προκατασκευασμένες εντολές χρειάστηκαν
 *  τον ίδιο περίπου χρόνο με τα κανονικά ερωτήματα, με βελτίωση μόνο 8%.

public class PreparedStatements {
  public static void main(String[] args) {
    if (args.length < 5) {
      printUsage();
      return;
    }
    String vendor = args[4];
    // Χρήση της DriverUtilities2.loadDrivers() για φόρτωση
    // των προγραμμάτων οδήγησης από ένα αρχείο XML.
    DriverUtilities.loadDrivers();
    if (!DriverUtilities.isValidVendor(vendor)) {
      printUsage();
      return;
    }
```

Λίστα 17.10 PreparedStatements.java *(συνέχεια)*

```
    String driver = DriverUtilities.getDriver(vendor);
    String host = args[0];
    String dbName = args[1];
    String url =
      DriverUtilities.makeURL(host, dbName, vendor);
    String username = args[2];
    String password = args[3];
    // Χρήση της "print" μόνο για επιβεβαίωση ότι λειτουργεί σωστά,
    // και όχι όταν λαμβάνονται χρονικά αποτελέσματα.
    boolean print = false;
    if ((args.length > 5) && (args[5].equals("print"))) {
      print = true;
    }
    Connection connection =
      ConnectionInfoBean.getConnection(driver, url,
                                       username, password);
    if (connection != null) {
      doPreparedStatements(connection, print);
      doRawQueries(connection, print);
    }
    try {
      connection.close();
    } catch(SQLException sqle) {
      System.err.println("Problem closing connection: " + sqle);
    }
  }

  private static void doPreparedStatements(Connection conn,
                                           boolean print) {
    try {
      String queryFormat =
        "SELECT id FROM music WHERE price < ?";
      PreparedStatement statement =
        conn.prepareStatement(queryFormat);
      long startTime = System.currentTimeMillis();
      for(int i=0; i<100; i++) {
        statement.setFloat(1, i/4);
        ResultSet results = statement.executeQuery();
        if (print) {
          showResults(results);
        }
      }
      long stopTime = System.currentTimeMillis();
      double elapsedTime = (stopTime - startTime)/1000.0;
      System.out.println("Executing prepared statement " +
                         "100 times took " +
                         elapsedTime + " seconds.");
```


Λίστα 17.10 PreparedStatements.java *(συνέχεια)*

```
    } catch(SQLException sqle) {
      System.err.println("Error executing statement: " + sqle);
    }
  }

  public static void doRawQueries(Connection conn,
                                  boolean print) {
    try {
      String queryFormat =
        "SELECT id FROM music WHERE price < ";
      Statement statement = conn.createStatement();
      long startTime = System.currentTimeMillis();
      for(int i=0; i<100; i++) {
        ResultSet results =
          statement.executeQuery(queryFormat + i/4);
        if (print) {
          showResults(results);
        }
      }
      long stopTime = System.currentTimeMillis();
      double elapsedTime = (stopTime - startTime)/1000.0;
      System.out.println("Executing raw query " +
                         "100 times took " +
                         elapsedTime + " seconds.");
    } catch(SQLException sqle) {
      System.err.println("Error executing query: " + sqle);
    }
  }

  private static void showResults(ResultSet results)
      throws SQLException {
    while(results.next()) {
      System.out.print(results.getString(1) + " ");
    }
    System.out.println();
  }

  private static void printUsage() {
    System.out.println("Usage: PreparedStatements host " +
                       "dbName username password " +
                       "vendor [print].");
  }
}
```

Το προηγούμενο παράδειγμα δείχνει πώς μπορείτε να δημιουργήσετε μια προκατασκευασμένη εντολή και να ορίσετε παραμέτρους για την εντολή αυτή, για ένα πρόγραμμα γραμμής διαταγών. Στην περίπτωση των εφαρμογών Ιστού, μπορείτε να υποβάλλετε στη βάση δεδομένων τις προκατασκευασμένες εντολές από μια σελίδα JSP. Αν κάνετε κάτι τέτοιο, η Τυπική Βιβλιοθήκη

Ετικετών JSP (JSTL, δείτε το βιβλίο More Servlets and JavaServer Pages) παρέχει την ενέργεια `sql:query` για τον ορισμό μιας προκατασκευασμένης εντολής που θα υποβληθεί στη βάση δεδομένων, καθώς και την ενέργεια `sql:param` για να καθορίσετε τις τιμές των παραμέτρων για την προκατασκευασμένη εντολή.

17.5 Δημιουργία καλούμενων εντολών

Με το αντικείμενο `CallableStatement` μπορείτε να εκτελέσετε μια αποθηκευμένη διαδικασία (stored procedure) ή συνάρτηση (function) σε μια βάση δεδομένων. Για παράδειγμα, σε μια βάση δεδομένων Oracle, μπορείτε να γράψετε μια διαδικασία ή συνάρτηση σε γλώσσα PL/SQL και να την αποθηκεύσετε στη βάση δεδομένων μαζί με τους πίνακες. Κατόπιν μπορείτε να δημιουργήσετε μια σύνδεση με τη βάση δεδομένων και να εκτελέσετε την αποθηκευμένη διαδικασία ή συνάρτηση μέσω ενός αντικειμένου `CallableStatement`.

Η αποθηκευμένη διαδικασία έχει πολλά πλεονεκτήματα. Για παράδειγμα, τα συντακτικά λάθη συλλαμβάνονται κατά το χρόνο μεταγλώττισης, και όχι κατά το χρόνο εκτέλεσης· η διαδικασία της βάσης δεδομένων μπορεί να εκτελείται ταχύτερα από ένα κανονικό ερώτημα SQL· και τέλος, ο προγραμματιστής θα πρέπει να γνωρίζει μόνο τις παραμέτρους εισόδου και εξόδου, και όχι τη δομή των πινάκων. Επιπλέον, η κωδικοποίηση της αποθηκευμένης διαδικασίας μπορεί να είναι ευκολότερη στη γλώσσα της βάσης δεδομένων απ' ό,τι στη γλώσσα προγραμματισμού Java, επειδή έτσι είναι δυνατή η πρόσβαση σε εγγενείς δυνατότητες της βάσης δεδομένων — όπως ακολουθίες (sequences), διεγέρτες (triggers), πολλαπλοί δρομείς, κ.λπ.

Ένα μειονέκτημα των αποθηκευμένων διαδικασιών είναι ότι μπορεί να χρειαστεί να μάθετε μια νέα γλώσσα που αφορά ειδικά τη βάση δεδομένων (σημειώστε όμως ότι η Oracle8i Database και οι επόμενες εκδόσεις υποστηρίζουν αποθηκευμένες διαδικασίες που έχουν γραφτεί στη γλώσσα προγραμματισμού Java). Ένα δεύτερο μειονέκτημα είναι ότι η επιχειρησιακή λογική της αποθηκευμένης διαδικασίας εκτελείται στο διακομιστή της βάσης δεδομένων, αντί στο μηχάνημα του πελάτη ή το διακομιστή Ιστού. Η τάση στη βιομηχανία είναι η κατά το δυνατόν απομάκρυνση της επιχειρησιακής λογικής από τη βάση δεδομένων και η τοποθέτησή της σε στοιχεία JavaBeans (ή, σε μεγάλα συστήματα, σε στοιχεία Enterprise JavaBeans) που εκτελούνται στο διακομιστή Ιστού. Το κύριο κίνητρο για αυτή την προσέγγιση σε μια αρχιτεκτονική Ιστού είναι ότι η προσπέλαση της βάσης δεδομένων και η δικτυακή είσοδος/έξοδος αποτελούν συχνά τους κύριους υπαίτιους για τη μείωση στην απόδοση.

Η κλήση μιας αποθηκευμένης διαδικασίας σε μια βάση δεδομένων περιλαμβάνει τα έξι βασικά βήματα που φαίνονται στη συνέχεια, τα οποία αναλύονται λεπτομερώς στις υποενότητες που ακολουθούν.

1. **Ορισμός της κλήσης στη διαδικασία της βάσης δεδομένων.** Όπως και με τις προκατασκευασμένες εντολές, χρησιμοποιείτε μια ειδική σύνταξη για να ορίσετε την κλήση μιας αποθηκευμένης διαδικασίας. Ο ορισμός της διαδικασίας χρησιμοποιεί σύνταξη διαφυγής (escape syntax), όπου κατάλληλα αγγλικά ερωτηματικά ορίζουν τις παραμέτρους εισόδου και εξόδου.

2. **Προετοιμασία ενός αντικειμένου `CallableStatement` για τη διαδικασία.** Για να πάρετε ένα αντικείμενο `CallableStatement` από ένα αντικείμενο `Connection`, καλείτε τη μέθοδο `prepareCall`.
3. **Δήλωση των τύπων των παραμέτρων εξόδου.** Πριν εκτελέσετε τη διαδικασία, θα πρέπει να δηλώσετε τον τύπο δεδομένων για κάθε παράμετρο εξόδου.
4. **Παροχή τιμών για τις παραμέτρους εισόδου.** Πριν εκτελέσετε τη διαδικασία, θα πρέπει να δώσετε τις τιμές για τις παραμέτρους εισόδου.
5. **Εκτέλεση της αποθηκευμένης διαδικασίας.** Για να εκτελέσετε την αποθηκευμένη διαδικασία της βάσης δεδομένων, καλείτε τη μέθοδο `execute` της κλάσης `CallableStatement`.
6. **Προσπέλαση των επιστρεφόμενων παραμέτρων εξόδου.** Καλείτε την αντίστοιχη μέθοδο `getXxx`, σύμφωνα με τον τύπο δεδομένων της εξόδου.

Ορισμός της κλήσης για τη διαδικασία της βάσης δεδομένων

Η δημιουργία ενός αντικειμένου `CallableStatement` είναι κάπως παρόμοια με τη δημιουργία ενός αντικειμένου `PreparedStatement` (δείτε την Ενότητα 17.4, "Χρήση προκατασκευασμένων εντολών"), αφού και εδώ χρησιμοποιείται μια ειδική σύνταξη SQL όπου κατάλληλα αγγλικά ερωτηματικά (?) αντικαθίστανται με μια τιμή πριν από την εκτέλεση της εντολής. Ο ορισμός μιας διαδικασίας μπορεί να πάρει γενικά τέσσερις μορφές.

- **Διαδικασία χωρίς παραμέτρους.**
  `{ call όνομα_διαδικασίας }`
- **Διαδικασία με παραμέτρους εισόδου.**
  `{ call όνομα_διαδικασίας(?, ?, ...) }`
- **Διαδικασία με μία παράμετρο εξόδου.**
  `{ ? call όνομα_διαδικασίας }`
- **Διαδικασία με παραμέτρους εισόδου και εξόδου.**
  `{ ? = call όνομα_διαδικασίας(?, ?, ...) }`

Και στις τέσσερις αυτές μορφές, το *όνομα_διαδικασίας* είναι το όνομα της διαδικασίας στη βάση δεδομένων. Θα πρέπει επίσης να θυμάστε ότι η διαδικασία μπορεί να επιστρέψει περισσότερες από μία παραμέτρους εξόδου, και ότι οι αριθμοδείκτες των τιμών των παραμέτρων ξεκινούν με τις παραμέτρους εξόδου. Έτσι, στο τελευταίο παράδειγμα διαδικασίας, η πρώτη παράμετρος *εισόδου* έχει τιμή αριθμοδείκτη 2 (και όχι 1).

Σημείωση

Αν η διαδικασία επιστρέφει παραμέτρους, τότε στον αριθμοδείκτη των παραμέτρων εισόδου πρέπει να συνυπολογίσετε τον αριθμό των παραμέτρων εξόδου.

Προετοιμασία ενός αντικειμένου CallableStatement για τη διαδικασία

Για να πάρετε ένα αντικείμενο `CallableStatement` από το αντικείμενο `Connection`, χρησιμοποιείτε τη μέθοδο `prepareCall` με τον εξής τρόπο:

```
String procedure = "{ ? = call procedure_name( ?, ? ) }";
CallableStatement statement =
  connection.prepareCall(procedure);
```

Δήλωση των τύπων των παραμέτρων εξόδου

Πρέπει να δηλώσετε τον τύπο δεδομένων JDBC για κάθε παράμετρο εξόδου με τη μέθοδο `registerOutParameter`, με τον εξής τρόπο:

```
statement.registerOutParameter(n, τύπος);
```

όπου το *n* αντιστοιχεί στις ταξινομημένες παραμέτρους εξόδου (χρησιμοποιώντας αριθμοδείκτες με βάση το 1) και ο *τύπος* αντιστοιχεί σε μια σταθερά που ορίζεται στην κλάση `java.sql.Types` (`Types.float`, `Types.DATE`, κ.λπ.).

Παροχή τιμών στις παραμέτρους εισόδου

Πριν να εκτελέσετε την αποθηκευμένη διαδικασία, αντικαθιστάτε τις σημειωμένες παραμέτρους εισόδου καλώντας τη μέθοδο `setXxx` που αντιστοιχεί στην καταχώριση την οποία θέλετε να ορίσετε και τον τύπο δεδομένων της παραμέτρου αυτής (για παράδειγμα, `setInt`, `setString`). Έτσι οι εντολές

```
statement.setString(2, "name");
statement.setFloat(3, 26.0F);
```

ορίζουν την πρώτη παράμετρο εισόδου (θεωρώντας ότι υπάρχει μία παράμετρος εξόδου) σε μια τιμή `String` και τη δεύτερη παράμετρο εισόδου σε μια τιμή `float`. Να θυμάστε ότι, αν η διαδικασία έχει παραμέτρους εξόδου, ο αριθμοδείκτης των παραμέτρων εισόδου ξεκινά μετά από τις παραμέτρους εξόδου.

Εκτέλεση της αποθηκευμένης διαδικασίας

Για να εκτελέσετε την αποθηκευμένη διαδικασία, καλείτε απλώς τη μέθοδο `execute` στο αντικείμενο `CallableStatement`. Για παράδειγμα

```
statement.execute();
```


Προσπέλαση των παραμέτρων εξόδου

Αν η διαδικασία επιστρέφει παραμέτρους εξόδου τότε, μετά την κλήση της μεθόδου `execute`, μπορείτε να προσπελάσετε κάθε παράμετρο εξόδου καλώντας τη μέθοδο `getXxx`, όπου το `Xxx` αντιστοιχεί στον τύπο δεδομένων που επιστρέφει η παράμετρος εξόδου (`getDouble`, `getDate`, κ.λπ.) Για παράδειγμα, η εντολή

```
int value = statement.getInt(1);
```

επιστρέφει την πρώτη παράμετρο εξόδου σε μορφή `int`.

Παράδειγμα

Στη Λίστα 17.11, η κλάση `CallableStatement` δείχνει την εκτέλεση μιας αποθηκευμένης διαδικασίας (από τεχνικής πλευράς είναι συνάρτηση, αφού επιστρέφει κάποια τιμή) που αφορά τον πίνακα `music` (δείτε την Ενότητα 18.5 για πληροφορίες σχετικά με τη διευθέτηση του πίνακα `music`) σε μια βάση δεδομένων Oracle. Για να δημιουργήσετε την αποθηκευμένη διαδικασία `discount` στη βάση δεδομένων, καλέστε την κλάση `CallableStatement` και καθορίστε `create` στη γραμμή εντολών. Η ενέργεια αυτή θα καλέσει τη μέθοδο `createStoredFunction`, η οποία υποβάλλει τη διαδικασία (ένα μακροσκελές αλφαριθμητικό) στη βάση δεδομένων ως ενημέρωση SQL. Εναλλακτικά, αν έχετε την Oracle SQL Plus, μπορείτε να φορτώσετε άμεσα τη διαδικασία από το αρχείο discounts.sql, που φαίνεται στη Λίστα 17.12. Για πληροφορίες σχετικά με την εκτέλεση σεναρίων SQL στο SQL Plus δείτε την Ενότητα 18.5.

Η αποθηκευμένη διαδικασία `discount` τροποποιεί την καταχώριση `price` στον πίνακα `music`. Συγκεκριμένα, η διαδικασία δέχεται δύο παραμέτρους εισόδου, την `composer_in` (ο συνθέτης που θα επιλεγεί στον πίνακα `music`) και `discount_in` (το ποσοστό έκπτωσης επί της τιμής). Αν η `discount_in` βρίσκεται έξω από τα όρια 0,05 μέχρι 0,5, επιστρέφεται τιμή -1· διαφορετικά, η αποθηκευμένη διαδικασία επιστρέφει τον αριθμό των γραμμών του πίνακα που τροποποιήθηκαν.

Λίστα 17.11 CallableStatements.java

```
package coreservlets;

import java.sql.*;
import coreservlets.beans.*;

/** Ένα παράδειγμα που εκτελεί την αποθηκευμένη διαδικασία "discount"
 *  της Oracle. Συγκεκριμένα, η τιμή όλων των συνθέσεων του Mozart
 *  στον πίνακα "music" ελαττώνεται κατά 10%.

public class CallableStatements {
  public static void main(String[] args) {
    if (args.length < 5) {
```

Λίστα 17.11 CallableStatements.java *(συνέχεια)*

```
      printUsage();
      return;
    }
    String vendor = args[4];
    // Αλλαγή στη DriverUtilities2.loadDrivers() για υποχρεωτική φόρτωση
    // των προγραμμάτων οδήγησης από το προεπιλεγμένο αρχείο XML.
    DriverUtilities.loadDrivers();
    if (!DriverUtilities.isValidVendor(vendor)) {
      printUsage();
      return;
    }
    String driver = DriverUtilities.getDriver(vendor);
    String host = args[0];
    String dbName = args[1];
    String url =
      DriverUtilities.makeURL(host, dbName, vendor);
    String username = args[2];
    String password = args[3];

    Connection connection =
      ConnectionInfoBean.getConnection(driver, url,
                                       username, password);
    if (connection == null) {
      return;
    }

    try {
      if ((args.length > 5) && (args[5].equals("create"))) {
        createStoredFunction(connection);
      }
      doCallableStatement(connection, "Mozart", 0.10F);
    } catch(SQLException sqle) {
      System.err.println("Problem with callable: " + sqle);
    } finally {
      try {
        connection.close();
      } catch(SQLException sqle) {
        System.err.println("Error closing connection: " + sqle);
      }
    }
  }

  private static void doCallableStatement(Connection connection,
                                          String composer,
                                          float discount)
      throws SQLException {
    CallableStatement statement = null;
    try {
      connection.prepareCall("{ ? = call discount( ?, ? ) }");
      statement.setString(2, composer);
```


Λίστα 17.11 CallableStatements.java *(συνέχεια)*

```
      statement.setFloat(3, discount);
      statement.registerOutParameter(1, Types.INTEGER);
      statement.execute();
      int rows = statement.getInt(1);
      System.out.println("Rows updated: " + rows);
    } catch(SQLException sqle) {
      System.err.println("Problem with callable: " + sqle);
    } finally {
      if (statement != null) {
        statement.close();
      }
    }
  }

  /** Δημιουργία της αποθηκευμένης διαδικασίας "discount" σε Oracle PL/SQL.
   *  Η διαδικασία (από τεχνικής πλευράς, είναι μια συνάρτηση PL/SQL αφού
   *  επιστρέφεται κάποια τιμή), μειώνει την τιμή για το συγκεκριμένο
   *  συνθέτη στον πίνακα "music".
   */

  private static void createStoredFunction(
                       Connection connection)
      throws SQLException {
    String sql = "CREATE OR REPLACE FUNCTION discount " +
                 "  (composer_in IN VARCHAR2, " +
                 "   discount_in IN NUMBER) " +
                 "RETURN NUMBER " +
                 "IS " +
                 "  min_discount CONSTANT NUMBER:= 0.05; " +
                 "  max_discount CONSTANT NUMBER:= 0.50; " +
                 "BEGIN " +
                 "  IF discount_in BETWEEN min_discount " +
                 "                 AND max_discount THEN " +
                 "    UPDATE music " +
                 "    SET price = price * (1.0 - discount_in) "+
                 "    WHERE composer = composer_in; " +
                 "    RETURN(SQL%ROWCOUNT); " +
                 "  ELSE " +
                 "    RETURN(-1); " +
                 "  END IF; " +
                 "END discount;";
    Statement statement = null;
    try {
      statement = connection.createStatement();
      statement.executeUpdate(sql);
    } catch(SQLException sqle) {
      System.err.println("Problem creating function: " + sqle);
    } finally {
```

Λίστα 17.11 CallableStatements.java *(συνέχεια)*

```
      if (statement != null) {
        statement.close();
      }
    }
  }

  private static void printUsage() {
    System.out.println("Usage: CallableStatement host " +
                       "dbName username password " +
                       "vendor [create].");
  }
}
```

Λίστα 17.12 discount.sql (συνάρτηση PL/SQL για την Oracle)

```
/* Επιβάλλει έκπτωση στην τιμή για όλα τα έργα του καθορισμένου
 * συνθέτη composer_in. Η τιμή των έργων μειώνεται κατά το ποσοστό
 * που καθορίζεται από την παράμετρο discount_in.
 *
 * Επιστρέφει τον αριθμό των γραμμών που τροποποιήθηκαν, ή επιστρέφει -1
 * αν η τιμή της έκπτωσης δεν είναι έγκυρη.
 */

CREATE OR REPLACE FUNCTION discount
  (composer_in IN VARCHAR2, discount_in IN NUMBER)
RETURN NUMBER
IS
  min_discount CONSTANT NUMBER:= 0.05;
  max_discount CONSTANT NUMBER:= 0.50;
BEGIN
  IF discount_in BETWEEN min_discount AND max_discount THEN
    UPDATE music
    SET price = price * (1.0 - discount_in)
    WHERE composer = composer_in;
    RETURN(SQL%ROWCOUNT);
  ELSE
    RETURN(-1);
  END IF;
END discount;
```

17.6 Χρήση συναλλαγών βάσεων δεδομένων

Όταν ενημερώνετε μία βάση δεδομένων, εξ ορισμού οι αλλαγές γράφονται μόνιμα στη βάση δεδομένων (αυτό στην ορολογία των βάσεων δεδομένων ονομάζεται *commit*). Όμως αυτή η προεπιλεγμένη συμπεριφορά μπορεί να απενεργοποιηθεί προγραμματιστικά. Αν απενεργοποιηθεί η αυτόματη μόνιμη εγγραφή (autocommit) και συμβεί κάποιο πρόβλημα κατά την ενημέρωση, τότε

μπορεί να γίνει επαναφορά στις αρχικές τιμές (roll back) για κάθε αλλαγή που έχει γίνει στη βάση δεδομένων. Αν οι ενημερώσεις εκτελεστούν με επιτυχία, τότε οι αλλαγές μπορούν να καταγραφούν μόνιμα στη βάση δεδομένων. Αυτή η προσέγγιση είναι γνωστή ως *διαχείριση συναλλαγών* (transaction manager).

Η διαχείριση συναλλαγών βοηθά στη διασφάλιση της ακεραιότητας της βάσης δεδομένων σας. Για παράδειγμα, υποθέστε ότι μεταφέρετε χρήματα από ένα λογαριασμό ταμιευτηρίου σε ένα λογαριασμό όψεως. Αν κάνετε πρώτα την ανάληψη από το λογαριασμό ταμιευτηρίου και μετά κάνετε την κατάθεση στο λογαριασμό όψεως, τι θα συμβεί αν υπάρξει σφάλμα αφού γίνει η ανάληψη αλλά πριν γίνει η κατάθεση; Οι λογαριασμοί του πελάτη θα έχουν λιγότερα χρήματα, και οι ελεγκτές της τράπεζας θα επιβάλουν πολύ αυστηρά πρόστιμα. Από την άλλη πλευρά, τι θα συμβεί αν γίνει πρώτα η κατάθεση στο λογαριασμό όψεως και μετά η ανάληψη από το λογαριασμό ταμιευτηρίου και συμβεί κάποιο σφάλμα μετά την κατάθεση αλλά πριν από την ανάληψη; Οι λογαριασμοί του πελάτη θα έχουν περισσότερα χρήματα και το διοικητικό συμβούλιο της τράπεζας θα πρέπει να απολύσει όλο το προσωπικό της μηχανογράφησης. Η ουσία είναι ότι, ανεξάρτητα από τη σειρά των λειτουργιών, οι λογαριασμοί θα βρίσκονται σε μια ασυνεπή κατάσταση αν η μία λειτουργία πραγματοποιηθεί με επιτυχία και η άλλη όχι. Θα πρέπει να διασφαλίσετε ότι είτε θα πραγματοποιηθούν και οι δύο λειτουργίες, ή καμία από τις δύο. Αυτό ακριβώς κάνει η διαχείριση συναλλαγών.

Η προεπιλογή για μια σύνδεση βάσης δεδομένων είναι η αυτόματη εγγραφή (autocommit)· δηλαδή, κάθε εντολή που εκτελείται, αυτομάτως εγγράφεται μόνιμα στη βάση δεδομένων. Έτσι, αν θέλετε να έχετε διαχείριση συναλλαγών, θα πρέπει πρώτα να απενεργοποιήσετε την αυτόματη εγγραφή για τη σύνδεση, καλώντας τη μέθοδο `setAutocommit(false)`.

Για να πραγματοποιήσετε τη διαχείριση συναλλαγών, συνήθως χρησιμοποιείτε ένα μπλοκ `try/catch/finally`. Καταρχήν θα πρέπει να καταγράψετε την κατάσταση της αυτόματης εγγραφής. Κατόπιν, σε ένα μπλοκ `try`, θα πρέπει να καλέσετε την `autoCommit(false)` και να εκτελέσετε κάποια ερωτήματα ή ενημερώσεις. Αν συμβεί κάποια αστοχία, καλείτε τη μέθοδο `rollback` στο μπλοκ `catch`· αν οι συναλλαγές ήταν επιτυχείς, καλείτε τη μέθοδο `commit` στο τέλος του μπλοκ `try`. Σε κάθε περίπτωση, στο μπλοκ `finally`, επαναφέρετε την κατάσταση της αυτόματης εγγραφής.

Ας δούμε ένα πρότυπο για αυτή την προσέγγιση ως προς τη διαχείριση συναλλαγών.

```
Connection connection =
  DriverManager.getConnection(url, username, password);
boolean autoCommit = connection.getAutoCommit();
Statement statement;
try {
  connection.setAutoCommit(false);
  statement = connection.createStatement();
  statement.execute(...);
  statement.execute(...);
  ...
  connection.commit();
} catch(SQLException sqle) {
  connection.rollback();
} finally {
```

```
    statement.close();
    connection.setAutoCommit(autoCommit);
  }
```

Εδώ η εντολή για τη λήψη μιας σύνδεσης από το αντικείμενο DriverManager βρίσκεται έξω από το μπλοκ try/catch. Έτσι δεν καλείται η μέθοδος rollback, εκτός και αν δημιουργηθεί με επιτυχία η σύνδεση. Ωστόσο η μέθοδος getConnection μπορεί να μεταβιβάσει μια εξαίρεση SQLException, και αυτή είτε θα πρέπει να μεταβιβαστεί από την εσωκλειόμενη μέθοδο, ή θα πρέπει να συλληφθεί από ένα ξεχωριστό μπλοκ try/catch.

Στη Λίστα 17.3 προσθέτουμε μερικά νέα έργα στον πίνακα music μέσα σε ένα μπλοκ συναλλαγής (για πληροφορίες σχετικά με τη δημιουργία του πίνακα music δείτε την Ενότητα 18.5). Για να γενικεύσουμε την εργασία, δημιουργούμε ένα αντικείμενο TransactionBean, στη Λίστα 17.14, στο οποίο καθορίζουμε τη σύνδεση με τη βάση δεδομένων και υποβάλλουμε ένα μπλοκ εντολών SQL με τη μορφή πίνακα αλφαριθμητικών. Ο κόκκος διασχίζει τον πίνακα των εντολών SQL, τις εκτελεί μία προς μία, και αν προκύψει εξαίρεση SQLException εκτελεί επαναφορά (rollback) και επαναμεταβιβάζει την εξαίρεση.

Λίστα 17.13 Transactions.java

```
package coreservlets;

import java.sql.*;
import coreservlets.beans.*;

/** Ένα παράδειγμα που δείχνει την υποβολή ενός μπλοκ εντολών
 *  SQL στα πλαίσια μίας συναλλαγής. Συγκεκριμένα, εισάγονται
 *  στον πίνακα music τέσσερις νέες εγγραφές. Αυτό γίνεται σε
 *  ένα μπλοκ συναλλαγής, οπότε αν προκύψει κάποιο πρόβλημα
 *  γίνεται επαναφορά (rollback) και οι αλλαγές δεν εγγράφονται
 *  στη βάση δεδομένων.
 *  <P>

public class Transactions {
  public static void main(String[] args) {
    if (args.length < 5) {
      printUsage();
      return;
    }
    String vendor = args[4];
    // Αλλαγή της DriverUtilities2.loadDrivers() για φόρτωση προγραμμάτων
    // οδήγησης από ένα αρχείο XML, αντί για φόρτωση προγραμμάτων οδήγησης
    // που έχουν εγγραφεί ως "κυριολεκτικά" στην DriverUtilities.
    DriverUtilities.loadDrivers();
    if (!DriverUtilities.isValidVendor(vendor)) {
      printUsage();
      return;
    }
    String driver = DriverUtilities.getDriver(vendor);
```


Λίστα 17.13 Transactions.java *(συνέχεια)*

```
    String host = args[0];
    String dbName = args[1];
    String url =
      DriverUtilities.makeURL(host, dbName, vendor);
    String username = args[2];
    String password = args[3];
    doTransactions(driver, url, username, password);
  }

  private static void doTransactions(String driver,
                                     String url,
                                     String username,
                                     String password) {
    String[] transaction =
    { "INSERT INTO music VALUES " +
      " ( 9, 'Chopin',       'No. 2 in F minor',  100, 17.99)",
      "INSERT INTO music VALUES " +
      " (10, 'Tchaikovsky',   'No. 1 in Bb minor', 100, 24.99)",
      "INSERT INTO music VALUES " +
      " (11, 'Ravel',        'No. 2 in D major',  100, 14.99)",
      "INSERT INTO music VALUES " +
      " (12, 'Schumann',     'No. 1 in A minor',  100, 14.99)"};
    TransactionBean bean = new TransactionBean();
    try {
      bean.setConnection(driver, url, username, password);
      bean.execute(transaction);
    } catch (SQLException sqle) {
      System.err.println("Transaction failure: " + sqle);
    } finally {
      bean.close();
    }
  }

  private static void printUsage() {
    System.out.println("Usage: Transactions host " +
                       "dbName username password " +
                       "vendor.");
  }
}
```

Λίστα 17.14 TransactionBean.java

```
package coreservlets.beans;

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

Λίστα 17.14 TransactionBean.java *(συνέχεια)*

```
/** Κόκκος για την εκτέλεση συναλλαγών JDBC. Μετά τον καθορισμό της
 *  σύνδεσης, υποβάλλεται ένα μπλοκ εντολών SQL στα πλαίσια μιας
 *  συναλλαγής με κλήση της execute. Αν προκύψει εξαίρεση SQLException,
 *  γίνεται επαναφορά (roll back) τυχόν προηγούμενων εντολών.
 */
public class TransactionBean {
  private Connection connection;

  public void setConnection(Connection connection) {
    this.connection = connection;
  }

  public void setConnection(String driver, String url,
                            String username, String password) {
    setConnection(ConnectionInfoBean.getConnection(
                    driver, url, username, password));
  }

  public Connection getConnection() {
    return(connection);
  }

  public void execute(List list) throws SQLException {
    execute((String[])list.toArray(new String[list.size()]));
  }

  public void execute(String transaction)
      throws SQLException {
    execute(new String[] { transaction });
  }

  /** Εκτέλεση ενός μπλοκ εντολών SQL στα πλαίσια μίας συναλλαγής.
   *  Αν παρουσιαστεί εξαίρεση SQLException, γίνεται επαναφορά
   *  και μεταβιβάζεται η εξαίρεση.
   */

  public void execute(String[] transaction)
      throws SQLException {
    if (connection == null) {
      throw new SQLException("No connection available.");
    }
    boolean autoCommit = connection.getAutoCommit();
    try {
      connection.setAutoCommit(false);
      Statement statement = connection.createStatement();
      for(int i=0; i<transaction.length; i++) {
        statement.execute(transaction[i]);
      }
      statement.close();
    } catch(SQLException sqle) {
```


Λίστα 17.14 TransactionBean.java *(συνέχεια)*

```
      connection.rollback();
      throw sqle;
    } finally {
      connection.commit();
      connection.setAutoCommit(autoCommit);
    }
  }

  public void close() {
    if (connection != null) {
      try {
        connection.close();
      } catch(SQLException sqle) {
        System.err.println(
          "Failed to close connection: " + sqle);
      } finally {
        connection = null;
      }
    }
  }
}
```

Το προηγούμενο παράδειγμα δείχνει τη χρήση ενός κόκκου για την υποβολή συναλλαγών σε μια βάση δεδομένων. Αυτή η προσέγγιση αποτελεί εξαιρετική λύση στην περίπτωση των μικρο-ϋπηρεσιών· ωστόσο, σε μια σελίδα JSP μπορείτε να χρησιμοποιήσετε την ενέργεια `sql:transaction` που διατίθεται από την Τυπική Βιβλιοθήκη Ετικετών JSP (JSTL). Για λεπτομέρειες σχετικά με την JSTL δείτε το βιβλίο More Servlets and JavaServer Pages.

17.7 Αντιστοίχιση δεδομένων σε αντικείμενα με χρήση πλαισίων εργασίας ORM

Λόγω της ανάγκης για εύκολη μεταφορά δεδομένων ανάμεσα σε μια βάση δεδομένων και ένα αντικείμενο Java, πολλοί κατασκευαστές έχουν αναπτύξει πλαίσια εργασίας (frameworks) για την αντιστοίχιση αντικειμένων σε σχεσιακές βάσεις δεδομένων. Πρόκειται για μια ισχυρή δυνατότητα, αφού ο αντικειμενοστρεφής προγραμματισμός και οι βάσεις δεδομένων πάντοτε είχαν μια αναντιστοιχία που αποτελούσε εμπόδιο: τα αντικείμενα καταλαβαίνουν τις έννοιες της κατάστασης και της συμπεριφοράς και μπορούν να κινηθούν προς άλλα αντικείμενα μέσω σχέσεων, ενώ οι σχεσιακές βάσεις δεδομένων αποθηκεύουν πληροφορίες σε πίνακες που συνήθως συσχετίζονται μεταξύ τους με πρωτεύοντα κλειδιά (primary keys).

Ο Πίνακας 17.1 συνοψίζει μερικά δημοφιλή πλαίσια εργασίας για αντιστοίχιση αντικειμένων σε σχεσιακές βάσεις δεδομένων (object-to-relational mapping, ORM). Υπάρχουν διαθέσιμα και πολυάριθμα άλλα πλαίσια εργασίας ORM. Για μια σύγκριση προϊόντων ORM που βασίζονται

στη Java, δείτε τη διεύθυνση http://c2.com/cgi-bin/wiki?ObjectRelationalToolComparison. Μια άλλη εξαιρετική πηγή για πλαίσια εργασίας ORM και εκπαιδευτικά βοηθήματα υπάρχει στην τοποθεσία http://www.javaskyline.com/database.html. (Θυμηθείτε ότι η αρχειοθήκη πηγαίου κώδικα του βιβλίου, στη διεύθυνση http://www.coreservlets.com, περιλαμβάνει και ενημερωμένους συνδέσμους για όλες τις διευθύνσεις URL που αναφέρονται στο βιβλίο.)

Πίνακας 17.1 Πλαίσια εργασίας για αντιστοίχιση αντικειμένων σε σχεσιακές βάσεις δεδομένων

Πλαίσιο εργασίας	URL
Castor	http://castor.exolab.org/
CocoBase	http://www.cocobase.com/
FrontierSuite	http://www.objectfrontier.com/
Kodo JDO	http://www.solarmetric.com/
ObJectRelationalBridge	http://db.apache.org/ojb/
TopLink	http://otn.oracle.com/products/ias/toplink/

Πολλά, αλλά όχι όλα, από αυτά τα πλαίσια εργασίας υποστηρίζουν την API για τα Java Data Objects (Αντικείμενα Δεδομένων Java, JDO). Η API JDO παρέχει μια πλήρη αντικειμενοστρεφή προσέγγιση για τη διαχείριση αντικειμένων που αντιστοιχίζονται σε διατηρούμενες αποθήκες (βάσεις δεδομένων). Η λεπτομερής περιγραφή των πλαισίων εργασίας ORM και των αντικειμένων JDO είναι έξω από τους σκοπούς αυτού του βιβλίου. Ωστόσο, θα δώσουμε μια σύντομη περίληψη για να δείξουμε την ισχύ που παρέχουν τα αντικείμενα JDO. Για περισσότερες πληροφορίες σχετικά με τα αντικείμενα JDO, επισκεφθείτε τις διευθύνσεις http://java.sun.com/products/jdo/ και http://jdocentral.com/. Μπορείτε επίσης να συμβουλευθείτε το βιβλίο *Core Java Data Objects*, των Sameer Tyagi και συνεργατών.

Στα πλαίσια εργασίας JDO ο προγραμματιστής πρέπει να δώσει "μεταδεδομένα" (metadata) μορφής XML για κάθε κλάση Java που ανιστοιχίζεται στη σχεσιακή βάση δεδομένων. Τα μεταδεδομένα ορίζουν τα διατηρούμενα πεδία κάθε κλάσης και τον εν δυνάμει ρόλο του κάθε πεδίου σε σχέση με τη βάση δεδομένων (π.χ., το πρωτεύον κλειδί). Αφού οριστούν τα μεταδεδομένα και ο πηγαίος κώδικας κάθε κλάσης Java, ο προγραμματιστής πρέπει να εκτελέσει ένα πλαίσιο εργασίας έτσι ώστε να παραγάγει τον αναγκαίο κώδικα JDO για την υποστήριξη της διατήρησης των πεδίων των αντικειμένων στη βάση δεδομένων (τη διατηρούμενη αποθήκη).

Τα πλαίσια εργασίας JDO χρησιμοποιούν δύο προσεγγίσεις για την τροποποίηση μιας κλάσης Java έτσι ώστε να υποστηρίζει τη διατηρούμενη αποθήκη: η πρώτη προσέγγιση είναι η τροποποίηση του πηγαίου κώδικα πριν από τη μεταγλώττιση, και η δεύτερη προσέγγιση είναι η τροποποίηση του ψηφιοκώδικα (bytecode), δηλαδή του αρχείου .class, μετά τη μεταγλώττιση του πηγαίου κώδικα. Η δεύτερη προσέγγιση είναι πιο συνηθισμένη, επειδή απλοποιεί τη συντήρηση του πηγαίου κώδικα — τον παραγόμενο κώδικα βάσης δεδομένων δεν τον βλέπει ποτέ ο προγραμματιστής.

Στην περίπτωση μιας υλοποίησης JDO για μια βάση δεδομένων SQL, το πλαίσιο εργασίας παράγει όλον τον απαραίτητο κώδικα που χρειάζεται ο διαχειριστής διατήρησης αντικειμένων JDO για την εισαγωγή (`INSERT`) νέων γραμμών στη βάση δεδομένων, καθώς επίσης και για την εκτέλεση των λειτουργιών ενημέρωσης (`UPDATE`) και διαγραφής (`DELETE`) με πραγματοποίηση μόνιμων τροποποιήσεων στα δεδομένα. Ο προγραμματιστής δεν χρειάζεται να γράψει κώδικα SQL ή JDBC· το πλαίσιο εργασίας παράγει όλον τον απαραίτητο κώδικα, και ο διαχειριστής διατήρησης αντικειμένων παράγει όλη την απαραίτητη επικοινωνία με τη βάση δεδομένων. Αφού διευθετηθεί το πλαίσιο εργασίας, ο προγραμματιστής πρέπει απλώς να δημιουργήσει αντικείμενα και να κατανοεί τον τρόπο λειτουργίας της API των αντικειμένων JDO.

Η Λίστα 17.5 παρουσιάζει το Music.jdo, ένα παράδειγμα ορισμού μεταδεδομένων για την κλάση `Music` (Λίστα 17.6) σύμφωνα με την υλοποίηση SolarMetric's Kodo JDO. Το αρχείο XML Music.jdo αντιστοιχίζει την κλάση `Music` σε έναν πίνακα της βάσης δεδομένων, χρησιμοποιώντας ένα στοιχείο `extension` όπου η ιδιότητα `key` έχει τιμή `table` και η ιδιότητα `value` έχει τιμή `music`. Δεν είναι αναγκαίο η βάση δεδομένων να διαθέτει ήδη έναν πίνακα με το όνομα `music`· στην πραγματικότητα, το πλαίσιο εργασίας Kodo δημιουργεί στη βάση δεδομένων όλους τους πίνακες που είναι απαραίτητοι για τη διατηρούμενη αποθήκευση πληροφοριών, χρησιμοποιώντας πιθανόν τροποποιημένα ονόματα πινάκων. Η ιδιότητα `name` ορίζει απλώς μία αντιστοίχιση για το πλαίσιο εργασίας.

Το αρχείο .jdo ορίζει επίσης ένα στοιχείο `field` για κάθε πεδίο της κλάσης που θα πρέπει να διατηρηθεί στη βάση δεδομένων. Κάθε στοιχείο `field` ορίζει ένα στοιχείο `extension` που αντιστοιχίζει ένα πεδίο της κλάσης με μια στήλη στον πίνακα της βάσης δεδομένων, όπου η ιδιότητα `value` επισημαίνει το όνομα της στήλης στη βάση δεδομένων. Τα στοιχεία `extension` εξαρτώνται από τον κατασκευαστή, γι' αυτό φροντίστε να συμβουλευθείτε την τεκμηρίωση του κατασκευαστή για τις κατάλληλες τιμές της ιδιότητας `key`.

Η κλάση `PersistentManager` παρέχει πρόσβαση στη διατηρούμενη αποθήκη πληροφοριών (τη βάση δεδομένων). Για παράδειγμα, η Λίστα 17.17 σας δείχνει πώς να εισάγετε στη διατηρούμενη αποθήκη, χρησιμοποιώντας τη μέθοδο `makePersistentAll`, πεδία που σχετίζονται με νέα αντικείμενα. Οι τροποποιήσεις στη διατηρούμενη αποθήκη γίνονται μέσω συναλλαγής, και πρέπει να βρίσκονται μεταξύ κλήσεων των μεθόδων `begin` και `commit` της κλάσης `Transaction`. Έτσι, για να εισαγάγετε στη βάση δεδομένων τα πεδία που σχετίζονται με ένα αντικείμενο `Music`, καλείτε απλώς τις κατάλληλες μεθόδους `setXxx` για το αντικείμενο `Music` και μετά καλείτε τη μέθοδο `makePersistentAll` μέσα από τη συναλλαγή. Ο διαχειριστής διατήρησης αντικειμένων JDO δημιουργεί αυτόματα και εκτελεί τις εντολές SQL για τη μόνιμη εγγραφή των δεδομένων στη βάση δεδομένων. Με ανάλογο τρόπο, η διαγραφή πεδίων που σχετίζονται με ένα αντικείμενο `Music` γίνεται μέσω της μεθόδου `makeDeletePersistent` της κλάσης `PersistentManager`. Για πιο πολύπλοκες εργασίες με τη διατηρούμενη αποθήκη, τα αντικείμενα JDO παρέχουν την κλάση `Query` για την εκτέλεση ερωτημάτων και την επιστροφή αποτελεσμάτων σε μορφή συλλογής αντικειμένων (`Collection`).

Τέλος, η θέση της βάσης δεδομένων, το όνομα χρήστη, ο κωδικός πρόσβασης, και άλλες πληροφορίες που αφορούν το σύστημα διαβάζονται από τις ιδιότητες του συστήματος (στο συγκεκριμένο παράδειγμα καθορίζονται στη Λίστα 17.18).

Λίστα 17.15 Music.jdo

```
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
"-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//EN"
"http://java.sun.com/dtd/jdo_1_0.dtd">
<jdo>
  <package name="coreservlets.jdo">
    <class name="Music" >
      <extension vendor-name="kodo"
                 key="table" value="music"/>
      <extension vendor-name="kodo"
                 key="lock-column" value="none"/>
      <extension vendor-name="kodo"
                 key="class-column" value="none"/>
      <field name="id" primary-key="true">
        <extension vendor-name="kodo"
                   key="data-column" value="id"/>
      </field>
      <field name="composer">
        <extension vendor-name="kodo"
                   key="data-column" value="composer"/>
      </field>
      <field name="concerto">
        <extension vendor-name="kodo"
                   key="data-column" value="concerto"/>
      </field>
      <field name="available">
        <extension vendor-name="kodo"
                   key="data-column" value="available"/>
      </field>
      <field name="price">
        <extension vendor-name="kodo"
                   key="data-column" value="price"/>
      </field>
    </class>
  </package>
</jdo>
```

Λίστα 17.16 Music.java

```
package coreservlets.jdo;

/** Το αντικείμενο music, που αντιστοιχεί σε μια εγγραφή (record) στη
 *  βάση δεδομένων. Το αντικείμενο/εγγραφή music παρέχει πληροφορίες
 *  σχετικά με ένα κονσέρτο που είναι διαθέσιμο για πώληση, και
 *  ορίζει πεδία για το αναγνωριστικό (ID), το συνθέτη, το κονσέρτο,
 *  τα διαθέσιμα κομμάτια, και την τιμή πώλησης.
*/
```


Λίστα 17.16 Music.java *(συνέχεια)*

```
public class Music {
  private int id;
  private String composer;
  private String concerto;
  private int available;
  private float price;

  public Music() { }

  public Music(int id, String composer, String concerto,
               int available, float price) {
    setId(id);
    setComposer(composer);
    setConcerto(concerto);
    setAvailable(available);
    setPrice(price);
  }

  public void setId(int id) {
    this.id = id;
  }

  public int getId() {
    return(id);
  }

  public void setComposer(String composer) {
    this.composer = composer;
  }

  public String getComposer() {
    return(concerto);
  }

  public void setConcerto(String concerto) {
    this.concerto = concerto;
  }

  public String getConcerto() {
    return(composer);
  }

  public void setAvailable(int available) {
    this.available = available;
  }

  public int getAvailable() {
    return(available);
  }
```

Λίστα 17.16 Music.java *(συνέχεια)*

```
  public void setPrice(float price) {
    this.price = price;
  }

  public float getPrice() {
    return(price);
  }
}
```

Λίστα 17.17 PopulateMusicTable.java

```
package coreservlets.jdo;

import java.util.*;
import java.io.*;
import javax.jdo.*;

/** Συμπλήρωση της βάσης δεδομένων με μουσικά κομμάτια, με χρήση των JDO.
*/

public class PopulateMusicTable {
  public static void main(String[] args) {
    // Create seven new music objects to place in the database.
    Music[] objects = {
      new Music(1, "Mozart", "No. 21 in C# minor", 7, 24.99F),
      new Music(2, "Beethoven", "No. 3 in C minor", 28, 10.99F),
      new Music(3, "Beethoven", "No. 5 Eb major", 33, 10.99F),
      new Music(4, "Rachmaninov", "No. 2 in C minor", 9, 18.99F),
      new Music(5, "Mozart", "No. 24 in C minor", 11, 21.99F),
      new Music(6, "Beethoven", "No. 4 in G", 33, 12.99F),
      new Music(7, "Liszt", "No. 1 in Eb major", 48, 10.99F)
    };

    // Φόρτωση αρχείου ιδιοτήτων με πληροφορίες JDO. Το αρχείο ιδιοτήτων
    // περιέχει πληροφορίες του πλαισίου εργασίας ORM που αφορούν ειδικά
    // τον κατασκευαστή της βάσης δεδομένων, καθώς και πληροφορίες για τη
    // σύνδεση με τη βάση δεδομένων.
    Properties properties = new Properties();
    try {
      FileInputStream fis =
        new FileInputStream("jdo.properties");
      properties.load(fis);
    } catch(IOException ioe) {
      System.err.println("Problem loading properties file: " +
                         ioe);
      return;
    }
```


Λίστα 17.17 PopulateMusicTable.java *(συνέχεια)*

```
    // Απόδοση αρχικών τιμών στο διαχειριστή του πλαισίου εργασίας
    // διατηρούμενων αντικειμένων.
    PersistenceManagerFactory pmf =
      JDOHelper.getPersistenceManagerFactory(properties);
    PersistenceManager pm = pmf.getPersistenceManager();

    // Εγγραφή του νέου αντικειμένου Music στη βάση δεδομένων.
    Transaction transaction = pm.currentTransaction();
    transaction.begin();
    pm.makePersistentAll(objects);
    transaction.commit();
    pm.close ();
  }
}
```

Λίστα 17.18 jdo.properties

```
# Πληροφορίες διευθέτησης για το πλαίσιο εργασίας Kodo JDO και
# τη βάση δεδομένων MySQL.
javax.jdo.PersistenceManagerFactoryClass=
     com.solarmetric.kodo.impl.jdbc.JDBCPersistenceManagerFactory
javax.jdo.option.RetainValues=true
javax.jdo.option.RestoreValues=true
javax.jdo.option.Optimistic=true
javax.jdo.option.NontransactionalWrite=false
javax.jdo.option.NontransactionalRead=true
javax.jdo.option.Multithreaded=true
javax.jdo.option.MsWait=5000
javax.jdo.option.MinPool=1
javax.jdo.option.MaxPool=80
javax.jdo.option.IgnoreCache=false
javax.jdo.option.ConnectionUserName: brown
javax.jdo.option.ConnectionURL: jdbc:mysql://localhost/csajsp
javax.jdo.option.ConnectionPassword: larry
javax.jdo.option.ConnectionDriverName: com.mysql.jdbc.Driver
com.solarmetric.kodo.impl.jdbc.WarnOnPersistentTypeFailure=true
com.solarmetric.kodo.impl.jdbc.SequenceFactoryClass=
     com.solarmetric.kodo.impl.jdbc.schema.DBSequenceFactory
com.solarmetric.kodo.impl.jdbc.FlatInheritanceMapping=true
com.solarmetric.kodo.EnableQueryExtensions=false
com.solarmetric.kodo.DefaultFetchThreshold=30
com.solarmetric.kodo.DefaultFetchBatchSize=10
com.solarmetric.kodo.LicenseKey=5A8A-D98C-DB5F-6070-6000
```