COP 4610L: Operating Systems Lab

Distributed Applications in the Enterprise

Lecture Set 4

Dr. R. Lent

Introduction

- Database
  - Collection of data
- DBMS
  - Database management system
  - Storing and organizing data
- SQL
  - Relational database
  - Structured Query Language
- JDBC
  - Java Database Connectivity
  - JDBC driver

Relational-Database Model

- Relational database
  - A DB = one or more tables
  - Table = a number of records
  - Record = a row of a table
  - Field = a column of a table
  - Primary key = Unique data
- SQL statement
  - Query
  - Record sets

Example RDB model

Number	Name	Department	Salary	Location
23603	Jones	413	1100	New Jersey
24568	Kerwin	413	2000	New Jersey
34589	Larson	642	1800	Los Angeles
35761	Myers	611	1400	Orlando
47132	Neumann	413	9000	New Jersey
78321	Stephens	611	8500	Orlando

Row/Record

Primary key

Column/Field

Relational-database structure of an **`Employee`** table.

Relational-Database Model (cont'd)

Department	Location
413	New Jersey
611	Orlando
642	Los Angeles

Result set formed by selecting **Department** and **Location** data from the **Employee** table.

Example: The **books** Database

Sample **books** database

- Four tables
  - **Authors**, **publishers**, **authorISBN** and **titles**
- Relationships among the tables

The **books** Database (cont'd)

Field	Description
authorID	Author's ID number in the database. In the **books** database, this integer field is defined as an *autoincremented field*. For each new record inserted in this table, the database automatically increments the **authorID** value to ensure that each record has a unique **authorID**. This field represents the table's primary key.
firstName	Author's first name (a string).
lastName	Author's last name (a string).

Fig. 8.3 authors table from **books**.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.4 Data from the **authors** table of **books**.

The **books** Database (cont'd)

Field	Description
publisherID	The publisher's ID number in the database. This autoincremented integer is the table's primary-key field.
publisherName	The name of the publisher (a string).

Fig. 8.5 **publishers** table from **books**.

publisherID	**publisherName**
1	Prentice Hall
2	Prentice Hall PTG

Fig. 8.6 Data from the **publishers** table of **books**.

The **books** Database (cont'd)

Field	Description
authorID	The author's ID number, which allows the database to associate each book with a specific author. The integer ID number in this field must also appear in the **authors** table.
isbn	The ISBN number for a book (a string).

Fig. 8.7 authorISBN table from **books**.

The **books** Database (cont'd)

authorID	isbn	authorID	isbn
1	0130895725	2	0139163050
1	0132261197	2	013028419x
1	0130895717	2	0130161438
1	0135289106	2	0130856118
1	0139163050	2	0130125075
1	013028419x	2	0138993947
1	0130161438	2	0130852473
1	0130856118	2	0130829277
1	0130125075	2	0134569555
1	0138993947	2	0130829293
1	0130852473	2	0130284173
1	0130829277	2	0130284181
1	0134569555	2	0130895601
1	0130829293	3	013028419x
1	0130284173	3	0130161438
1	0130284181	3	0130856118
1	0130895601	3	0134569555
2	0130895725	3	0130829293
2	0132261197	3	0130284173
2	0130895717	3	0130284181
2	0135289106	4	0130895601

Fig. 8.8 Data from the **authorISBN** table of **books**.

The **books** Database (cont'd)

Field	Description
isbn	ISBN number of the book (a string).
title	Title of the book (a string).
editionNumber	Edition number of the book (an integer).
copyright	Copyright year of the book (a string).
publisherID	Publisher's ID number (an integer). This value must correspond to an ID number in the **publishers** table.
imageFile	Name of the file containing the book's cover image (a string).
price	Suggested retail price of the book (a real number). [*Note:* The prices shown in this book are for example purposes only.]

Fig. 8.9 **titles** table from **books**.

The **books** Database (cont'd)

isbn	title	edition-Number	copy-right	publish-erID	image-File	price
0130895725	C How to Program	3	2001	1	chtp3.jpg	69.95
0132261197	C How to Program	2	1994	1	chtp2.jpg	49.95
0130895717	C++ How to Program	3	2001	1	cpphtp3.jpg	69.95
0135289106	C++ How to Program	2	1998	1	cpphtp2.jpg	49.95
0139163050	The Complete C++ Training Course	3	2001	2	cppctc3.jpg	109.95
013028419x	e-Business and e-Commerce How to Program	1	2001	1	ebechtp1.jpg	69.95
0130161438	Internet and World Wide Web How to Program	1	2000	1	iw3htp1.jpg	69.95
0130856118	The Complete Internet and World Wide Web Programming Training Course	1	2000	2	iw3ctc1.jpg	109.95
0130125075	Java How to Program (Java 2)	3	2000	1	jhtp3.jpg	69.95
0138993947	Java How to Program (Java 1.1)	2	1998	1	jhtp2.jpg	49.95
0130852473	The Complete Java 2 Training Course	3	2000	2	javactc3.jpg	109.95
0130829277	The Complete Java Training Course (Java 1.1)	2	1998	2	javactc2.jpg	99.95
0134569555	Visual Basic 6 How to Program	1	1999	1	vbhtp1.jpg	69.95
0130829293	The Complete Visual Basic 6 Training Course	1	1999	2	vbctc1.jpg	109.95
0130284173	XML How to Program	1	2001	1	xmlhtp1.jpg	69.95

Relational Database Overview: The **books** Database (cont'd)

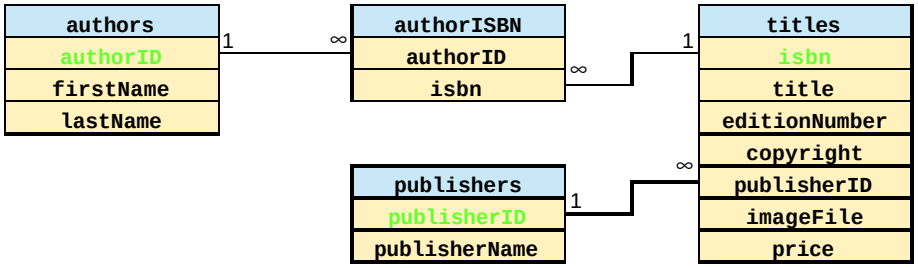

Table relationships in **books**.

Structured Query Language (SQL)

- SQL=Structured Query Language
- Provides the API that allows data to be manipulated (entered, edited, and selected) from most relational DBs
- MySQL an open-source DBMS
  - Daemon: mysqld (add .exe on windows)
  - Command line client: mysql
- A single DB engine can handle more than one DB

SQL Commands

- ALTER
- CREATE
- DELETE
- DESCRIBE
- DROP
- EXPLAIN
- FLUSH
- GRANT
- INSERT
- LOAD
- REPLACE
- SELECT
- SET
- SHOW
- UPDATE
- USE

Installation of MySQL

- Download and install MySQL
- Start server
  - Automatically or manually: mysqld --console
- Tighten security
  - mysql –u root mysql
  - UPDATE user SET password=PASSWORD(abc123') WHERE User='root';
  - DELETE FROM user WHERE Host=‘%’;
  - DELETE FROM user WHERE User=‘’;
  - DELETE FROM db WHERE Host=‘%’;
  - FLUSH PRIVILEGES;

Create a New User

- Add a user
  - INSERT INTO user (host, user, password) VALUES ('localhost', 'rlent', PASSWORD(‘abc123'));
  - flush privileges;
- Grant access:
  - GRANT ALL PRIVILEGES ON books.* to rlent@localhost;
  - Or define what privileges: select,insert,update,delete,create,drop
  - no need to “flush privileges”

Creation and Deletion of a DB

- Creation:
  - mysql –u root –p
  - CREATE DATABASE test;
  - Use test;
- Deletion
  - mysql –u root –p
  - DROP DATABASE test;
  - Conditional deletion: DROP DATABASE IF EXISTS test;

Creation and Deletion of Tables

- CREATE TABLE name (fielname1 type modifiers, …);
- To all tables: SHOW TABLES;
- Types:
  - Numbers: INT or INTEGER, FLOAT
  - Strings: VARCHAR (up to 255), TEXT (64 Kb), LONGTEXT (4 Gb)
  - Dates:
    - DATE: YYYY-MM-DD
    - TIME: HH:MM:SS
    - DATETIME: YYYY:MM:DD HH:MM:SS
    - YEAR: YYYY
    - TIMESTAMP (updated every time the row is modified)
- Modifiers:
  - AUTO_INCREMENT (automatically assign next number)
  - DEFAULT value
  - NOT NULL
  - PRIMARY KEY
  - UNSIGNED

Creating Database books in MySQL

- Create database books
  - use script books.sql provided:

    mysql –u rlent –p < books.sql
- Take a look at books.sql to see how a DB can be created

Basic SELECT Query

- Simplest format of a SELECT query
  - **SELECT** * **FROM** tableName
    - **SELECT** * **FROM** **authors**
- Select specific fields from a table
  - **SELECT** **authorID**, **lastName** **FROM** **authors**

authorID	lastName		
1	Deitel		
2	Deitel		
3	Nieto		
4	Santry		

Fig. 8.13 **authorID** and **lastName** from the **authors** table.

WHERE Clause

- Specify the selection criteria
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **WHERE** criteria
    - **SELECT** title, editionNumber, copyright
      FROM titles
      WHERE copyright > 1999
- **WHERE** clause condition operators
  - <, >, <=, >=, =, <>
  - **LIKE**
    - wildcard characters **%** and **_**

WHERE Clause (cont'd)

title	editionNumber	copyright
C How to Program	3	2001
C++ How to Program	3	2001
The Complete C++ Training Course	3	2001
e-Business and e-Commerce How to Program	1	2001
Internet and World Wide Web How to Program	1	2000
The Complete Internet and World Wide Web Programming Training Course	1	2000
Java How to Program (Java 2)	3	2000
The Complete Java 2 Training Course	3	2000
XML How to Program	1	2001
Perl How to Program	1	2001
Advanced Java 2 Platform How to Program	1	2002

Fig. 8.14 Titles with copyrights after 1999 from table **titles**.

WHERE Clause (cont'd)

- **SELECT** authorID, firstName, lastName
  FROM authors
  WHERE lastName **LIKE** 'D%'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel

Fig. 8.15 Authors whose last name starts with **D** from the **authors** table.

WHERE Clause (cont'd)

- **SELECT** authorID, firstName, lastName
  FROM authors
  WHERE lastName **LIKE** '_i%'

authorID	firstName	lastName
3	Tem	Nieto

Fig. 8.16 The only author from the **authors** table whose last name contains **i** as the second letter.

ORDER BY Clause

- Optional **ORDER BY** clause
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **ORDER BY** field **ASC**
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **ORDER BY** field **DESC**
- **ORDER BY** multiple fields
  - **ORDER BY** field1 sortingOrder, field2 sortingOrder, …
- Combine the **WHERE** and **ORDER BY** clauses

ORDER BY Clause (cont'd)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **ASC**

authorID	firstName	lastName
2	Paul	Deitel
1	Harvey	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.17 Authors from table **authors** in ascending order by **lastName**.

ORDER BY Clause (cont'd)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **DESC**

authorID	firstName	lastName
4	Sean	Santry
3	Tem	Nieto
2	Paul	Deitel
1	Harvey	Deitel

Fig. 8.18 Authors from table **authors** in descending order by **lastName**.

ORDER BY Clause (cont'd)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName, firstName

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.19 Authors from table **authors** in ascending order by **lastName** and by **firstName**.

ORDER BY Clause (cont'd)

- **SELECT** isbn, title, editionNumber, copyright, price
  FROM titles **WHERE** title **LIKE** '%How to Program'
  ORDER BY title **ASC**

isbn	title	edition-Number	copy-right	price
0130895601	Advanced Java 2 Platform How to Program	1	2002	69.95
0132261197	C How to Program	2	1994	49.95
0130895725	C How to Program	3	2001	69.95
0135289106	C++ How to Program	2	1998	49.95
0130895717	C++ How to Program	3	2001	69.95
0130161438	Internet and World Wide Web How to Program	1	2000	69.95
0130284181	Perl How to Program	1	2001	69.95
0134569555	Visual Basic 6 How to Program	1	1999	69.95
0130284173	XML How to Program	1	2001	69.95
013028419x	e-Business and e-Commerce How to Program	1	2001	69.95

Fig. 8.20 Books from table **titles** whose title ends with **How to Program** in ascending order by **title**.

Limiting Selection

- **`Select ……. Limit`** <number of results>
- **`Select ……. Limit`** <starting from>, <number of results>

Merging Data from Multiple Tables: Joining

- Join the tables
  - Merge data from multiple tables into a single view
  - **SELECT** fieldName1, fieldName2, …

    FROM table1, table2

    WHERE table1.fieldName = table2.fieldName
  - **SELECT** firstName, lastName, isbn

    FROM authors, authorISBN

    WHERE authors.authorID = authorISBN.authorID

    ORDER BY lastName, firstName

Merging Data from Multiple Tables: Joining (cont'd)

firstName	lastName	isbn	firstName	lastName	isbn
Harvey	Deitel	0130895601	Harvey	Deitel	0130284173
Harvey	Deitel	0130284181	Harvey	Deitel	0130829293
Harvey	Deitel	0134569555	Paul	Deitel	0130852473
Harvey	Deitel	0130829277	Paul	Deitel	0138993947
Harvey	Deitel	0130852473	Paul	Deitel	0130125075
Harvey	Deitel	0138993947	Paul	Deitel	0130856118
Harvey	Deitel	0130125075	Paul	Deitel	0130161438
Harvey	Deitel	0130856118	Paul	Deitel	013028419x
Harvey	Deitel	0130161438	Paul	Deitel	0139163050
Harvey	Deitel	013028419x	Paul	Deitel	0135289106
Harvey	Deitel	0139163050	Paul	Deitel	0130895717
Harvey	Deitel	0135289106	Paul	Deitel	0132261197
Harvey	Deitel	0130895717	Paul	Deitel	0130895725
Harvey	Deitel	0132261197	Tem	Nieto	0130284181
Harvey	Deitel	0130895725	Tem	Nieto	0130284173
Paul	Deitel	0130895601	Tem	Nieto	0130829293
Paul	Deitel	0130284181	Tem	Nieto	0134569555
Paul	Deitel	0130284173	Tem	Nieto	0130856118
Paul	Deitel	0130829293	Tem	Nieto	0130161438
Paul	Deitel	0134569555	Tem	Nieto	013028419x
Paul	Deitel	0130829277	Sean	Santry	0130895601

Fig. 8.21 Authors and the ISBN numbers for the books they have written in ascending order by **lastName** and **firstName**.

INSERT INTO Statement

- Insert a new record into a table
  - **INSERT INTO** tableName (fieldName1, … , fieldNameN)

    VALUES (value1, … , valueN)
    - **INSERT INTO** authors (firstName, lastName)

      VALUES (‘Sue’, ‘Smith’)

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Fig. 8.22 Table **Authors** after an **INSERT INTO** operation to add a record.

UPDATE Statement

- Modify data in a table
  - **UPDATE** tableName

    SET fieldName1 = value1, … , fieldNameN = valueN

    WHERE criteria
  - **UPDATE** authors

    SET lastName = ‘Jones’

    WHERE lastName = ‘Smith’ **AND** firstName = ‘Sue’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Jones

Fig. 8.23 Table **authors** after an **UPDATE** operation to change a record.

DELETE FROM Statement

- Remove data from a table
  - **DELETE FROM** tableName **WHERE** criteria
    - **DELETE FROM** authors

      WHERE lastName = 'Jones' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.24 Table **authors** after a **DELETE** operation to remove a record.

Importing and Exporting Data

- Export:

```
SELECT * INTO OUTFILE “c:/data.txt” FROM test;
```

- Import:

```
LOAD DATA INFILE “c:/data.txt” INTO TABLE test;
```

Optional arguments: … **FIELDS TERMINATED BY ‘,’**

(field1, field2);

Altering the Structure of a Table

- Add a new column:

```
ALTER TABLE name ADD (newFieldName type);
```

- Remove a column:

```
ALTER TABLE name DROP fieldName;
```

Informative Commands

- **`Show databases`**
- **`Show tables`**
- **`Explain table_name or describe table_name`**
- **`Show columns from table_name`**
- **`Show status`**
- **`Show table status`**
- **`Show variables`**

Transactions and MySQL

- Create tables with:

  CREATE TABLE tablename (…..) TYPE=innodb;

- Use BEGIN, COMMIT, or ROLLBACK to handle a group of SQL statements as a single transaction

How to Enable innodb (Linux)

- Edit /etc/mysql/my.cnf

```
# Read the manual if you want to enable InnoDB!
# skip-innodb
innodb_data_home_dir =
innodb_data_file_path = /var/lib/mysql/ibdata/ibdata1:100M:autoextend
```

- Create dir:

```
Mkdir /var/lib/mysql/ibdata/ibdata1
Chown mysql:mysql /var/lib/mysql/ibdata/ibdata1
```

- **Restart mysqld:**

```
/etc/init.d/mysql restart
```

- Check:

```
Show variables like ‘have%’;
```

JDBC (`java.sql`)

- Idea:
  - Establish a *connection* with a database
  - Send SQL commands
  - Receive error codes, and/or sets of records
- JDBC provides a standard library for accessing RDBs
  - Standardizes:
    - Way to establish a connection to DB
    - How to initiate queries
    - Methods to create stored queries
    - The structure of the data in a query result
  - Does not standardize:
    - SQL syntax

JDBC Components

JDBC consists of two parts:

1. JDBC API, which is a java-based API
2. JDBC Driver Manager, which communicates with specific drivers that perform the real connection with the DB

Get the MySQL JDBC driver

Note: Under Linux, install mysql connector driver and then comment out line *skip-networking* in /etc/mysql/my.cfg

Seven Steps in Using JDBC

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

JDBC Details

1. Load the driver

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Class.forName(“com.mysql.jdbc.Driver");
} catch { ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " cnfe);
}
```

2. Define the Connection URL

```
String host = “eola.cs.ucf.edu";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
":" + port + ":" + dbName;
String mysqlURL = "jdbc:mysql://" +host+ "/“ +dbName;
```

JDBC Details (cont'd)

3. Establish the Connection

```
String username = "rlent";
String password = "abc123";
Connection connection =
  DriverManager.getConnection(mysqlURL,username,
  password);
```

- **We may look up information about the database**

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

JDBC Details (cont'd)

4. Create a Statement

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

5. Execute a Query

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

Notes:

- **`UPDATE, INSERT, DELETE`** must use **`executeUpdate`**,
- **Method `setQueryTimeout`** can define the maximum time to wait for results

JDBC Details (cont'd)

6. Process the Result

```
while(resultSet.next()) {
    System.out.println(resultSet.getString(1) + " " +
    resultSet.getString(2) + " " +
    resultSet.getString(3));
}
```

Note:columns start with index 1 and not 0

7. Close the Connection

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

See example: simpleTest.java

Statement

- An **`statement`** object allows SQL statements to be sent to a database.
- There 3 types of statement objects:
  1. **`Statement`**

     For executing a simple SQL statement
  2. **`PreparedStatement`**

     For executing a precompiled SQL statement passing in parameters
  3. **`CallableStatement`**

     For executing a database stored procedure

`Statement` Methods

`executeQuery`

- Executes the SQL query and returns the data in a table **`ResultSet`**
- The resulting table may be empty but never null

```
ResultSet rs =
statement.executeQuery("SELECT name FROM table");
```

`executeUpdate`

- Used to execute for INSERT, UPDATE, or DELETE SQL statements
- The return is the number of rows that were affected in the database

```
int rows =
statement.executeUpdate("DELETE FROM authors WHERE
  name='Fred'";
```

`Statement` Methods (cont'd)

`execute`

- Method for executing stored procedures and prepared statements
- The statement execution may (or may not) return a **`ResultSet`** (**`statement.getResultSet`**). If the return value is true, two or more result sets were produced

`getMaxRows/setMaxRows`

- Determines the maximum number of rows a ResultSet may contain
- By defaults the number of rows is unlimited (a value of 0)

`getQueryTimeout/setQueryTimeout`

- Specifies the time a driver will wait for a statement complete before throwing a **`SQLException`**

Prepared Statements

- Prepared statements are precompiled queries that are more efficient than regular statements
- Useful if you need to execute similar SQL statements many times
- A statement in standard form that is sent to the database for compilation before actually being used
- Each time you use it, you simply replace some of the marked parameters using the set… methods (e.g. setString)
- The corresponding execute methods need no parameters
  - execute()
  - executeQuery()
  - executeUpdate()

Prepared Statement Methods

- **setInt, setString, etc.**
  - Sets the correspoding parameter (?) in the SQL statement to the given value
- **clearParameters**
  - Clears all set parameter values in the statement

Prepared Statement Example

```
Connection connection =
   DriverManager.getConnection(url, user, password);
PreparedStatement statement =
   connection.prepareStatement("UPDATE authors "+
   "SET authorName = ? " + "WHERE authorID = ?");

for(int i=0; i<10; i++) {
   statement.setString(1, newAuthorName[i]);
   statement.setInt(2, newAuthorID[i]);
   statement.executeUpdate();
}
```

Transactions and JDBC

- By default, after each SQL statement is executed the changes are automatically committed to the database
- By turning off auto-commit, two or more statements can be grouped into a transaction

```
connection.setAutoCommit(false)
```

- After successfully executing a group of statements, call commit to permanently record the changes.
- Call rollback if an error occurs

`connection` methods for Transactions

- **`getAutoCommit/setAutoCommit`**
  - A connection is set to auto-commit by default
  - Gets or sets the auto-commit mode
- **`commit`**
  - Force all changes since the last call to commit to become permanent
  - Any database locks currently held by this Connection object are released
- **`rollback`**
  - Drops all changes since the previous call to commit
  - Releases any database locks held by this Connection object

A Transaction Example

```
Connection connection =
DriverManager.getConnection(url, username, passwd);
connection.setAutoCommit(false);
try {
    statement.executeUpdate(...);
    statement.executeUpdate(...);
    connection.commit();
} catch (Exception e) {
try {
   connection.rollback();
} catch (SQLException e) {
   // report problem
}
} finally {
try {
   connection.close();
} catch (SQLException e) { }
}
```

Batch Processing

Series of updates can be performed in a *batch update*

- Add SQL statement to a batch
- Execute the batch later

- **`Statement`** provides the methods to support batch processing
- **`PreparedStatement`** and **`CallableStatement`** inherits the methods

Methods for Batch Processing

Method	Description
`public void addBatch(String sql)`	Method of interface **`Statement`** that receives a **`String`** argument specifying an SQL statement to add to the **`Statement`**'s batch for future execution. This method should not be used with **`PreparedStatement`**s and **`CallableStatement`**s.
`public void clearBatch()`	Method of interface **`Statement`** that clears the statement's batch.
`public int[] executeBatch()`	Method of interface **`Statement`** that executes the statement's batch. The method returns an array of **`int`** values indicating the status of each SQL statement in the batch. The order of the values in the array corresponds to the order in which the SQL statements are added to the batch.

Return Values of **executeBatch**

Return value	Description
a value greater than or equal to **0**	Indicates successful execution of the SQL statements in the batch update. The value specifies the actual number of rows updated in the database.
-2	Indicates successful execution of the SQL statements in the batch update and that the affected number of rows is unknown.
-3	Indicates an SQL statement that failed to execute properly during a batch update. When the batch update is allowed to complete its processing, the array returned by **getUpdateCounts** contains the value **-3** for any SQL statement that failed. When the batch update is not allowed to continue after an exception, the array returned by **getUpdateCounts** contains elements for only the SQL statements that executed successfully before the exception occurred. When a failure occurs, **executeUpdate** throws a **BatchUpdateException**. In such cases, the program can catch the exception and invoke **BatchUpdateException** method **getUpdateCounts** to obtain the array of update counts. Some databases allow a batch update to continue executing when an exception occurs, while others do not.

Processing Multiple **`ResultSet`**s or Update Counts

- Execute the SQL statements
- Identify the result type
  - **`ResultSet`**s
  - Update counts
- Obtain result
  - **`getResultSet`**
  - **`getUpdateCount`**

Processing Multiple **`ResultSet`**s or Update Counts (cont'd)

Method	Description
`public boolean execute()`	Programs use this method to execute SQL statements that can return multiple **`ResultSet`**s or update counts. This method returns a **`boolean`** indicating whether the first result is a **`ResultSet`** (**`true`**) or an update count (**`false`**). Based on the value returned, the program can call **`getResultSet`** or **`getUpdateCount`** to obtain the first result.
`public boolean getMoreResults()`	After obtaining the first result returned from method **`execute`**, a program invokes this method to move to the next result. This method returns a **`boolean`** indicating whether the next result is a **`ResultSet`** (**`true`**) or an update count (**`false`**). Based on the value returned, the program can call **`getResultSet`** or **`getUpdateCount`** to obtain the next result.
`public ResultSet getResultSet()`	Obtains a **`ResultSet`** from the results returned by method **`execute`**. This method returns **`null`** if the result is not a **`ResultSet`** or if there are no more results.
`public int getUpdateCount()`	Obtains an update count from the results returned by method **`execute`**. This method returns **`-1`** if the result is not an update count or if there are no more results.