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
Relational-database structure of an **Employee** table.
Result set formed by selecting `Department` and `Location` data from the `Employee` table.

<table>
<thead>
<tr>
<th>Department</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>413</td>
<td>New Jersey</td>
</tr>
<tr>
<td>611</td>
<td>Orlando</td>
</tr>
<tr>
<td>642</td>
<td>Los Angeles</td>
</tr>
</tbody>
</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.

<table>
<thead>
<tr>
<th><strong>authorID</strong></th>
<th><strong>firstName</strong></th>
<th><strong>lastName</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
</tbody>
</table>

Fig. 8.4 Data from the *authors* table of books.
### The books Database (cont’d)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publisherID</td>
<td>The publisher’s ID number in the database. This autoincremented integer is the table’s primary-key field.</td>
</tr>
<tr>
<td>publisherName</td>
<td>The name of the publisher (a string).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>publisherID</th>
<th>publisherName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prentice Hall</td>
</tr>
<tr>
<td>2</td>
<td>Prentice Hall PTG</td>
</tr>
</tbody>
</table>

**Fig. 8.6** Data from the publishers table of books.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorID</td>
<td>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.</td>
</tr>
<tr>
<td>isbn</td>
<td>The ISBN number for a book (a string).</td>
</tr>
</tbody>
</table>

*Fig. 8.7* authorISBN table from books.
Fig. 8.8 Data from the authorISBN table of books.
The *books* Database (cont’d)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isbn</td>
<td>ISBN number of the book (a string).</td>
</tr>
<tr>
<td>title</td>
<td>Title of the book (a string).</td>
</tr>
<tr>
<td>copyright</td>
<td>Copyright year of the book (a string).</td>
</tr>
<tr>
<td>publisherID</td>
<td>Publisher’s ID number (an integer). This value must correspond to an ID number in the <em>publishers</em> table.</td>
</tr>
<tr>
<td>imageFile</td>
<td>Name of the file containing the book’s cover image (a string).</td>
</tr>
<tr>
<td>price</td>
<td>Suggested retail price of the book (a real number). [Note: The prices shown in this book are for example purposes only.]</td>
</tr>
</tbody>
</table>

Fig. 8.9 *titles* table from *books*. 
<table>
<thead>
<tr>
<th>isbn</th>
<th>title</th>
<th>edition</th>
<th>copyright</th>
<th>publisherID</th>
<th>image - File</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130895725</td>
<td>C How to Program</td>
<td>3</td>
<td>2001</td>
<td>1</td>
<td>chtp3.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0132261197</td>
<td>C How to Program</td>
<td>2</td>
<td>1994</td>
<td>1</td>
<td>chtp2.jpg</td>
<td>49.95</td>
</tr>
<tr>
<td>0130895717</td>
<td>C++ How to Program</td>
<td>3</td>
<td>2001</td>
<td>1</td>
<td>cpphtp3.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0135289106</td>
<td>C++ How to Program</td>
<td>2</td>
<td>1998</td>
<td>1</td>
<td>cpphtp2.jpg</td>
<td>49.95</td>
</tr>
<tr>
<td>0139163050</td>
<td>The Complete C++ Training Course</td>
<td>3</td>
<td>2001</td>
<td>2</td>
<td>cppetc3.jpg</td>
<td>109.95</td>
</tr>
<tr>
<td>013028419X</td>
<td>e-Business and e-Commerce How to Program</td>
<td>1</td>
<td>2001</td>
<td>1</td>
<td>ebechtp1.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0130161438</td>
<td>Internet and World Wide Web How to Program</td>
<td>1</td>
<td>2000</td>
<td>1</td>
<td>iw3htp1.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0130856118</td>
<td>The Complete Internet and World Wide Web How to Program</td>
<td>1</td>
<td>2000</td>
<td>2</td>
<td>iw3ctc1.jpg</td>
<td>109.95</td>
</tr>
<tr>
<td>0130125075</td>
<td>Java How to Program (Java 2)</td>
<td>3</td>
<td>2000</td>
<td>1</td>
<td>jhtp3.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0138993947</td>
<td>Java How to Program (Java 1.1)</td>
<td>2</td>
<td>1998</td>
<td>1</td>
<td>jhtp2.jpg</td>
<td>49.95</td>
</tr>
<tr>
<td>0130852473</td>
<td>The Complete Java 2 Training Course</td>
<td>3</td>
<td>2000</td>
<td>2</td>
<td>javactc3.jpg</td>
<td>109.95</td>
</tr>
<tr>
<td>0130829277</td>
<td>The Complete Java Training Course (Java 1.1)</td>
<td>2</td>
<td>1998</td>
<td>2</td>
<td>javactc2.jpg</td>
<td>99.95</td>
</tr>
<tr>
<td>0134569555</td>
<td>Visual Basic 6 How to Program</td>
<td>1</td>
<td>1999</td>
<td>1</td>
<td>vbhtp1.jpg</td>
<td>69.95</td>
</tr>
<tr>
<td>0130829293</td>
<td>The Complete Visual Basic 6 Training Course</td>
<td>1</td>
<td>1999</td>
<td>2</td>
<td>vbctc1.jpg</td>
<td>109.95</td>
</tr>
<tr>
<td>0130284173</td>
<td>XML How to Program</td>
<td>1</td>
<td>2001</td>
<td>1</td>
<td>xmlhtp1.jpg</td>
<td>69.95</td>
</tr>
</tbody>
</table>
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 is 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 (filename1 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
    - 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`

<table>
<thead>
<tr>
<th>authorID</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Santry</td>
</tr>
</tbody>
</table>

*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 `_`
### Fig. 8.14

Titles with copyrights after 1999 from table `titles`.

<table>
<thead>
<tr>
<th>title</th>
<th>editionNumber</th>
<th>copyright</th>
</tr>
</thead>
<tbody>
<tr>
<td>C How to Program</td>
<td>3</td>
<td>2001</td>
</tr>
<tr>
<td>C++ How to Program</td>
<td>3</td>
<td>2001</td>
</tr>
<tr>
<td>The Complete C++ Training Course</td>
<td>3</td>
<td>2001</td>
</tr>
<tr>
<td>e-Business and e-Commerce How to Program</td>
<td>1</td>
<td>2001</td>
</tr>
<tr>
<td>Internet and World Wide Web How to Program</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>The Complete Internet and World Wide Web Programming Training Course</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>Java How to Program (Java 2)</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>The Complete Java 2 Training Course</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>XML How to Program</td>
<td>1</td>
<td>2001</td>
</tr>
<tr>
<td>Perl How to Program</td>
<td>1</td>
<td>2001</td>
</tr>
<tr>
<td>Advanced Java 2 Platform How to Program</td>
<td>1</td>
<td>2002</td>
</tr>
</tbody>
</table>
WHERE Clause (cont’d)

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

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
</tbody>
</table>

*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%'

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
</tbody>
</table>

**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
**SELECT** authorID, firstName, lastName

**FROM** authors

**ORDER BY** lastName **ASC**

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
</tbody>
</table>

*Fig. 8.17* Authors from table *authors* in ascending order by lastName.
**SELECT** authorID, firstName, lastName

**FROM** authors

**ORDER BY** lastName **DESC**

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
</tbody>
</table>

**Fig. 8.18** Authors from table authors in descending order by lastName.
**SELECT** authorID, firstName, lastName

**FROM** authors

**ORDER BY** lastName, firstName

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
</tbody>
</table>

**Fig. 8.19** Authors from table authors in ascending order by lastName and by firstName.
**SELECT** isbn, title, editionNumber, copyright, price

**FROM** titles **WHERE** title LIKE ‘%How to Program’

**ORDER BY** title **ASC**

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

**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)

<table>
<thead>
<tr>
<th>firstName</th>
<th>lastName</th>
<th>isbn</th>
<th>firstName</th>
<th>lastName</th>
<th>isbn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Harvey</td>
<td>Deitel</td>
<td>0130284173</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130284181</td>
<td>Harvey</td>
<td>Deitel</td>
<td>0130829293</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0134569555</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130852473</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130829277</td>
<td>Paul</td>
<td>Deitel</td>
<td>0138993947</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130852473</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130125075</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0138993947</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130856118</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130125075</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130161438</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130856118</td>
<td>Paul</td>
<td>Deitel</td>
<td>013028419x</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130161438</td>
<td>Paul</td>
<td>Deitel</td>
<td>0139163050</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>013028419x</td>
<td>Paul</td>
<td>Deitel</td>
<td>0135289106</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0139163050</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130895717</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0135289106</td>
<td>Paul</td>
<td>Deitel</td>
<td>0132261197</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130895717</td>
<td>Paul</td>
<td>Deitel</td>
<td>0130895725</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0132261197</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130284181</td>
</tr>
<tr>
<td>Harvey</td>
<td>Deitel</td>
<td>0130895725</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130284173</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130829293</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130284181</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>0134569555</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130856118</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>0130161438</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Tem</td>
<td>Nieto</td>
<td>013028419x</td>
</tr>
<tr>
<td>Paul</td>
<td>Deitel</td>
<td>0130895601</td>
<td>Sean</td>
<td>Santry</td>
<td>0130895601</td>
</tr>
</tbody>
</table>

**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' )`

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
<tr>
<td>5</td>
<td>Sue</td>
<td>Smith</td>
</tr>
</tbody>
</table>

**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'`

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
<tr>
<td>5</td>
<td>Sue</td>
<td>Jones</td>
</tr>
</tbody>
</table>

**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’

<table>
<thead>
<tr>
<th>authorID</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Harvey</td>
<td>Deitel</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Deitel</td>
</tr>
<tr>
<td>3</td>
<td>Tem</td>
<td>Nieto</td>
</tr>
<tr>
<td>4</td>
<td>Sean</td>
<td>Santry</td>
</tr>
</tbody>
</table>

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);
  ```
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

```java
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

```java
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;
```
3. Establish the Connection

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

• We may look up information about the database

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

```java
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

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

See example: simpleTest.java
An `Statement` object allows SQL statements to be sent to a database.

There are 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

```java
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

```java
int rows = statement.executeUpdate("DELETE FROM authors WHERE name='Fred'");
```
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 default, 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 corresponding parameter (?) in the SQL statement to the given value

- **clearParameters**
  - Clears all set parameter values in the statement
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.
  ```java
  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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void addBatch(String sql)</code></td>
<td>Method of interface <code>Statement</code> that receives a <code>String</code> argument specifying an SQL statement to add to the <code>Statement</code>’s batch for future execution. This method should not be used with <code>PreparedStatement</code>s and <code>CallableStatement</code>s.</td>
</tr>
<tr>
<td><code>public void clearBatch()</code></td>
<td>Method of interface <code>Statement</code> that clears the statement’s batch.</td>
</tr>
<tr>
<td><code>public int[] executeBatch()</code></td>
<td>Method of interface <code>Statement</code> that executes the statement’s batch. The method returns an array of <code>int</code> 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.</td>
</tr>
</tbody>
</table>
## Return Values of `executeBatch`

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a value greater than or equal to 0</td>
<td>Indicates successful execution of the SQL statements in the batch update. The value specifies the actual number of rows updated in the database.</td>
</tr>
<tr>
<td>-2</td>
<td>Indicates successful execution of the SQL statements in the batch update and that the affected number of rows is unknown.</td>
</tr>
<tr>
<td>-3</td>
<td>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 <code>getUpdateCounts</code> 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 <code>getUpdateCounts</code> contains elements for only the SQL statements that executed successfully before the exception occurred. When a failure occurs, <code>executeUpdate</code> throws a <code>BatchUpdateException</code>. In such cases, the program can catch the exception and invoke <code>BatchUpdateException</code> method <code>getUpdateCounts</code> to obtain the array of update counts. Some databases allow a batch update to continue executing when an exception occurs, while others do not.</td>
</tr>
</tbody>
</table>
Processing Multiple `ResultSet` or Update Counts

- Execute the SQL statements
- Identify the result type
  - `ResultSet`
  - Update counts
- Obtain result
  - `getResultSet`
  - `getUpdateCount`
### Processing Multiple ResultSets or Update Counts (cont’d)

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