Server-Side Web Development

JSP

Lecture #6 2007

1. JDBC
   - Relational Databases
   - SQL
   - JDBC
2. XML
   - XML
   - XML Schema
   - XPath

Java Database Connectivity (JDBC)

- (not an official acronym)
- Java API for database access
- Aims to virtualize Java database access

Relational Databases

- Database server software referred to as Database Management Systems (DBMS)
- Database schemas describe databases
- Data ordered in tables and columns
- Tables provide groupings of data
- Columns provide field structure

Relational Databases

- Keys are used for ids and references
- Data stored in rows (aka records, tuples)
- Data accessed and manipulated via SQL, JDBC etc
- Databases can store vast amounts of data
- Databases also contains simple functions
- Creation of custom functions (stored procedures) supported by most major database systems

Database Table

id	username	birthdate	email
1	test1	1970-01-01	test1@jsp.com
2	test2	1970-01-02	test2@jsp.com
3	test3	1970-01-03	test3@jsp.com
4	test4	1970-01-04	test4@jsp.com
5	test5	1970-01-05	test5@jsp.com

Transactions

- A way to provide concurrent access to databases
- Allows for several operations to be viewed as a single, atomic operation
- If an error occurs, all changes are undone (rollback)

Structured Query Language (SQL)

- Text-based language used to access databases
- Standardized versions exists (ANSI SQL)
- Most database vendors provide their own dialects
- Select queries are used to ask questions about data
- Insert, update and delete statements are used to manipulate databases
- Create, alter & drop statements are used to create, modify and destroy databases
- Most databases provide functions which can be called using SQL

SQL Queries

```
SELECT *
FROM profile
WHERE profile.id = '123'

SELECT username, birthdate, email
FROM profile
WHERE profile.id = '123'
```

SQL Queries

```
SELECT COUNT(*)
FROM profile
WHERE LEN(profile.email) > 0

SELECT *
FROM user, profile
WHERE user.id = profile.id
AND LEN(profile.email) > 0
ORDER BY profile.username
LIMIT 25                              <- MySQL dialect
```

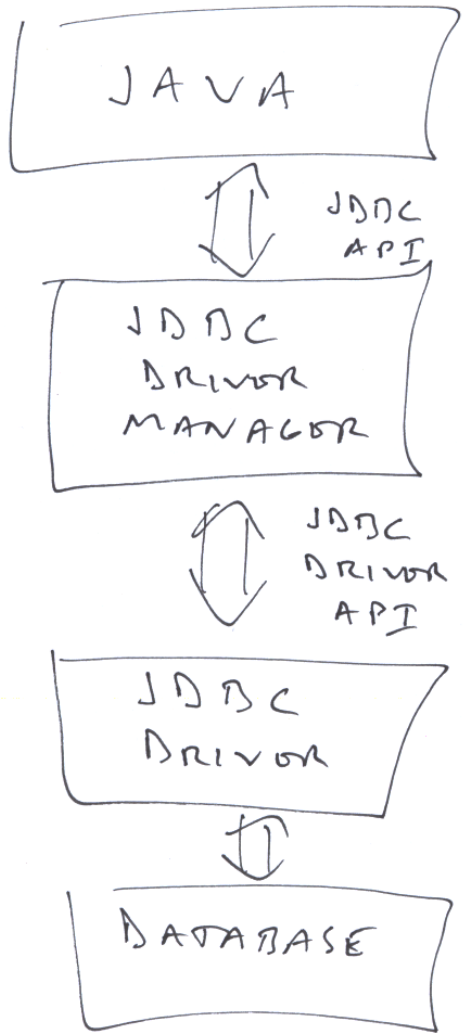
SQL Updates

```
INSERT INTO profile (id, username, birthdate, email)
VALUES (123,'test1','1970-01-01','test1@jsp.com')

UPDATE profile SET email = 'test1@jsp.com'
WHERE id = '123'

DELETE FROM profile
WHERE id = '123'
```

JDBC

- Java API for database access
- Constructed around a JDBC Driver Manager
- Database vendors provide JDBC drivers
- Switch database without changes in Java code

JDBC


JAVA
JDBC API
JDBC DRIVER MANAGER
JDBC DRIVER API
JDBC DRIVER
DATABASE

JDBC Data Types

JDBC type	Java type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	BigDecimal

JDBC Data Types

JDBC type	Java type
DATE	java.sql.Date
TIME, TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
DISTINCT	mapping underlying type
STRUCT	Struct
REF	Ref
JAVA_OBJECT	underlying Java class

Using JDBC

1. Load driver
2. Create database URL
3. Connect to database
4. Create a statement
5. Execute a query / update
6. Process results
7. Close connection

JDBC Drivers

- Data buffering
- Connection pooling
- Links to stored procedures
- Driver instantiated via dynamic class loading (required, need only be done once)

JDBC Driver Example

```
public static void instantiateDriver (String driver)
  throws ClassNotFoundException
{
  try
  {
    Class.forName(driver).newInstance();
  }
  catch (InstantiationException e)
  {
    throw new ClassNotFoundException(e.getMessage());
  }
  catch (IllegalAccessException e)
  {
    throw new ClassNotFoundException(e.getMessage());
  }
}
```

JDBC Driver Example

```
static
{
  // Apache Derby
  instantiateDriver("org.apache.derby.jdbc.EmbeddedDriver");

  // MySQL
  instantiateDriver("com.mysql.jdbc.Driver");

  // PostgreSQL
  instantiateDriver("org.postgresql.Driver");
}
```

JDBC Database URLs

jdbc:*dbms*:[//*host*:*port*/]*database*

- Provides JDBC database connection information
- Contains host, port, database (optionally username/password)

JDBC Database URL Examples

DBMS	URL
Apache Derby	jdbc:derby:*database*
MySQL	jdbc:mysql://*host*:3306/*database*
PostgreSQL	jdbc:postgresql://*host*:5432/*database*

JDBC Database Connections

- Represents a connection to a database
- May result in an underlying TCP/IP connection
- Connection may be compressed / encrypted
- Costly to create, reuse is recommended

JDBC Statement Types

- Statement - simple SQL statement
- PreparedStatement - precompiled SQL statement
- CallableStatement - database stored procedure

JDBC Statements

- SQL is constructed and parsed for each request
- Quick and easy
- Should be replace with prepared statements (when optimizing)

JDBC PreparedStatements

- Used to precompile a JDBC statement
- Parameters are added to the statement
- More efficient when queries are repeated

JDBC CallableStatements

- Used to call stored procedures in a database
- Parameters are added to call
- Stored procedure executes in the DBMS
- More efficient than prepared statements
- Requires a stored procedure
- Ties the application to a particular DBMS

JDBC ResultSets

- Interfaces to statement results
- Can be thought of as table views
- Doesn't actually contain all data returned from queries (requests data as needed)
- Can be used to both read and write data
- Exact behavior of ResultSets depends on statement type

JDBC Select

- Retrieves data from the database

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_READ_ONLY);
String sql = "SELECT * FROM profile";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.next();
String id = resultSet.getString("id");
String username = resultSet.getString("username");
String birthdate = resultSet.getString("birthdate");
String email = resultSet.getString("email");
```

JDBC SQL Insert

- Updates data in the database
- Data must be manually filtered / escaped

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql =
  "INSERT INTO profile (id, username, birthdate, email)" +
  "VALUES (123,'test1','1970-01-01','test1@jsp.com')";
ResultSet resultSet = statement.execute(sql);
```

JDBC ResultSet Insert

- Inserts data into the database
- Data is filtered / escaped by JDBC

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT * FROM profile";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.moveToInsertRow();
resultSet.updateString("id","123");
resultSet.updateString("username","test1");
resultSet.updateString("birthdate","1970-01-01");
resultSet.updateString("email","test1@jsp.com");
resultSet.insertRow();
```

JDBC ResultSet Update

- Updates existing data in the database
- Data is filtered / escaped by JDBC

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT * FROM profile WHERE id='123'";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.next();
resultSet.updateString("username","test1");
resultSet.updateString("birthdate","1970-01-01");
resultSet.updateString("email","test1@jsp.com");
resultSet.updateRow();
```

JDBC Transactions

- connection.setAutoCommit(false) *(default: true)*
- connection.commit()
- connection.rollback()

JDBC Transaction Example

```
Connection connection = null;
try
{
  connection = getConnection();
  connection.setAutoCommit(false);
  runSomeQuery();
  runAnotherQuery();
  connection.commit();
}
catch (SQLException e)
{
  connection.rollback();
}
finally
{
  try
  {
    if (connection != null)
      connection.close();
  }
  catch (SQLException e)
  {
  }
}
```

JDBC MetaData

- Provides information about databases and drivers
- Commonly used to list tables, columns etc
- Comparable to introspection / Java reflection

Extensible Markup Language (XML)

- Designed to structure and describe data
- A family of related technologies
- Widely used in a range of technologies today
- Often used for data validation and migration

XML

- Tags are user specified (not predefined)
- XML documents are extensible
  - applications are still able to parse extended documents
  - unrecognized extensions are ignored
- XML is platform, software and hardware independent
- XML is used to structure, store and send information
- XML is bot human and machine-readable

XML Syntax

- XML document = hierarchy of elements
- Document version and encoding stated in prolog (optional)
- A single root element
- All elements (except root) has a parent
- Elements must be closed and properly nested
- Each element may have a set of nested children
- Each element can have a set of attributes
- Attributes are name-value pairs
- Attribute names are unique for the element
- Attribute values are escaped and quoted
- Element names are case sensitive

XML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rootElement>
  <childElement1 attribute1="value1" attribute2="value2">
    <!-- Element names are not necessarily unique -->
    <subChild>Value1</subChild>
    <subChild>Value2</subChild>
  </childElement1>
  <childElement2>
    <subChild3>Value3</subChild3>
  </childElement2>
  <!-- childElement3 is an empty element -->
  <childElement3/>
  <!- childElement4 is an empty element with attribute -->
  <childElement4 attribute4="value4"/>
</rootElement>
```

XML namespaces

- Defines a collection of elements (a "vocabulary")
- Usually has an associated syntax (e.g., a Schema)
- Implies a semantic (user "knows what is means")
- Namespaces resolves naming conflicts
- Namespaces gives elements given fully qualified names
- Namespaces are declared as attributes and givens prefixes
- A namespaces is simply a unique name (URI)
- All child elements with the same prefix are associated with the same namespace
- A document can assign a default namespace (no prefix)

XML Namespace Example

```
<addresslist
     xmlns:us="http://us.addresses"
     xmlns:swe="http://swe.addresses">

  <us:address>
    <us:street>E-street</us:street>
    <us:city>L.A</us:city>
    <us:state>California</us:state>
    <us:zip>12345</us:zip>
  </us:address>

  <swe:address>
    <swe:street>Kungsgatan 50</swe:street>
    <swe:city>Stockholm</swe:city>
    <swe:postcode>12345</swe:postcode>
  </swe:address>

</addresslist>
```

XML Interpretation

- Well-formed: conforms to XML syntax specifications
  e.g., tags properly nested, tags closed, attributes quoted
- Valid: conforms to a namespace-defined syntax
  i.e., document matches a syntax definition (schema)

XML Schema

- Describes the structure of a XML document
- Is used to validate XML documents
- Is itself a XML document
- Typically difficult to read
- Defines elements, attributes and data types

Terminology

- A XML Schema *validates* a XML document
- A XML document *matches* a XML Schema
- A XML document is an *instance of* a XML Schema

XML Schema Documents

- Root tag called schema
- Matches the XML Schema document namespace http://www.w3.org/2000/08/XMLSchema
- Usually declares its own namespace
- All defined elements etc belongs to this namespace
- Instance elements with this namespace are validated against the schema

XML Schema Namespace

```
<?xml version="1.0"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://my.language.ns"
     xmlns="http://my.language.ns"
     elementFormDefault="qualified">

...

</xsd:schema>
```

XML Schema Basics

- Schemas contain
  - Type definitions
  - Element declarations
  - Attribute declarations
- Simple type
  - Does not have sub-elements
  (no child elements or attributes)
  - Predefined type or derived from predefined type
- Complex type
  - Have sub-element(s) - elements and/or attributes
- Element declarations can reference both simple and complex types
- Attributes can only reference simple types

Predefined Types

- xsd:string
- xsd:QName
- xsd:int
- xsd:long
- xsd:decimal
- xsd:double
- xsd:boolean
- xsd:date
- xsd:time
- xsd:dateTime
- xsd:anyURI

Simple Types

Declarations:

```
<xsd:element name="lastname" type="xsd:string"/>
<xsd:element name="age" type="xsd:int"/>
<xsd:element name="birthdate" type="xsd:date"/>
```

Instances:

```
<lastname>Andersson</lastname>
<age>37</age>
<birthdate>1970-01-01</birthdate>
```

Attributes

Declaration:

```
<xsd:attribute name="nationality" type="xsd:string"/>
```

Instance:

```
<person nationality="Sweden">Kalle Kula</person>
```

Derived Simple Types

- Derived from existing simple types (predefined or derived)
- Typically restricts existing simple types
- Use *restriction* elements with facets to restrict value ranges
- Facets are rules of restriction - Integer: minInclusive, maxInclusive, minExclusive, etc
  - String: pattern, whitespace
  - Any type: enumeration (lists valid values)

Derived Simple Types

Declaration:

```
<xsd:simpleType name="CarType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Audi"/>
    <xsd:enumeration value="BMW"/>
    <xsd:enumeration value="Volvo"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="car" type="CarType"/>
```

Instance:

```
<car>Audi</car>
<car>BMW</car>
<car>Volvo</car>
```

Complex Types

Contains

- Element declarations
- Element references
- Attribute declarations

Model (of how elements occur in the type)

- Sequence - each component in specified order
- Choice - exactly one of the specified components
- All - each component in any order

Complex Types

Declaration:

```
<xsd:complexType name="USAddressType" >
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="state" type="xsd:string" />
    <xsd:element name="zip" type="xsd:decimal" />
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="address" type="USAddressType"/>
```

Complex Types

Instance:

```
<address country="US">
  <name>John</name>
  <street>E-street</street>
  <city>L.A</city>
  <state>California</state>
  <zip>12345</zip>
</address>
```

Derived Complex Types

Declaration:

```
<xsd:complexType name="BaseType">
  <xsd:sequence>
    <xsd:element name="a"/>
    <xsd:element name="b"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ExtendedType">
  <xsd:complexContent>
    <xsd:extension base="BaseType">
       <xsd:sequence>
         <xsd:element name="c" type="xsd:string"/>
         <xsd:element name="d" type="xsd:string"/>
       </xsd:sequence>
     </xsd:entension>
  </xsd:complexContent>
</xsd:complexType>
```

Derived Complex Types

```
Instance:

  <baseTypeInstance>
    <a>...</a>
    <b>...</b>
  </baseTypeInstance>

  <extendedTypeInstance>
    <a>...</a>
    <b>...</b>
    <c>...</c>
    <d>...</d>
  </extendedTypeInstance>
```

Schema Composition

- include - add multiple schemas with the same target namespace to a document
- import - add multiple schemas with differing target namespaces to a document

any

Allows documents to use elements not specified by schema

```
<xs:element name="person" type="PersonType"/>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="firstname"type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:any minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

anyAttribute

Allows documents to use attributes not specified by schema

```
<xs:element name="person" type="PersonType"/>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="firstname"type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="married" type="xs:boolean" use="required"/>
  <xs:anyAttribute/>
</xs:complexType>
```

XML Path Language (XPath)

- Expression language for addressing parts of XML documents
- Used for pattern-based searching of XML documents
- Possible to search on elements, attributes and tag data

Next Time

- Tag libraries
- Web services