CSCD 327: Relational Database Systems

Database application development

Instructor: Dr. Dan Li

Overview

- JDBC
- Stored procedures
- Functions/Triggers

SQL as an Independent Query Language

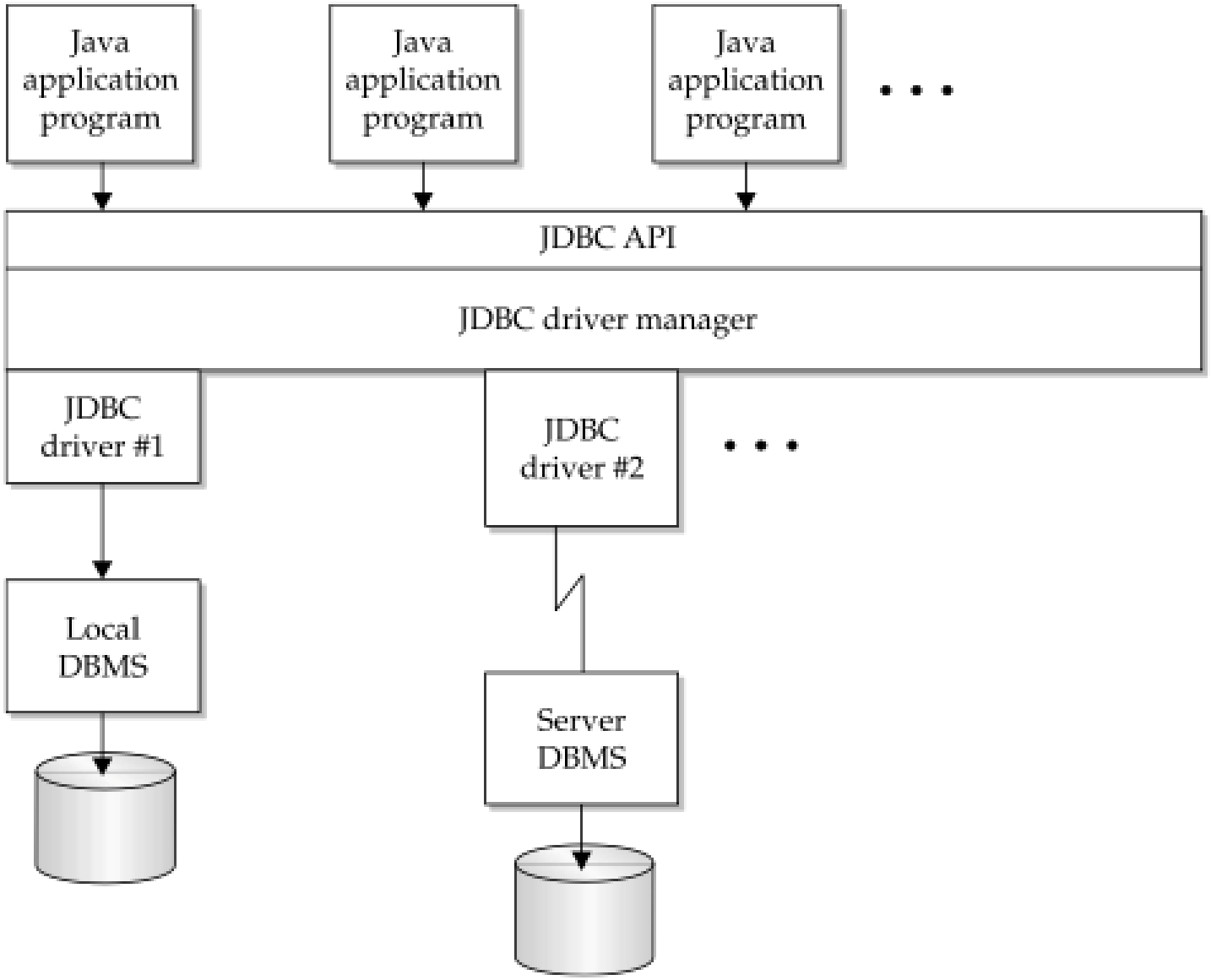
- SQL is a independent query language; as such, it has limitations.
- via programming languages :
  - Complex computational processing of the data.
  - Specialized user interfaces.
  - Access to more than one database at a time.

Integrate SQL into Other Programming Languages

- Embed SQL in the host language
- SQL APIs
  - JDBC
  - ODBC
- Stored Procedures and Triggers
  - Having DBMS take on more responsibility

JDBC: Architecture

- Four architectural components:
  - Application (initiates and terminates connections, submits SQL statements)
  - Driver manager (load JDBC driver)
  - Driver (connects to data source, transmits requests and returns/translates results and error codes)
  - Data source (processes SQL statements)


Java application program
Java application program
Java application program
JDBC API
JDBC driver manager
JDBC driver #1
JDBC driver #2
Local DBMS
Server DBMS

JDBC API

- Java is an object-oriented language, so it's probably no surprise that JDBC organizes its API functions around a collection of database-related objects and the methods that they provide:
- **Driver Manager object** The entry-point to JDBC
- **Connection objects** Represent individual active connections to target databases
- **Statement objects** Represent SQL statements to be executed
- **ResultSet objects** Represent the results of a SQL query
- **MetaData objects** Represent metadata about databases, query results, and statements
- **Exception objects** Represent errors in SQL statement execution

DriverManager Object Methods

Method	Description
`getConnection()`	Creates and returns a database connection object, given a URL for the datasource, and optionally a user name and password, and connection properties
`registerDriver()`	Registers a driver with JDBC driver manager
`setLoginTimeout()`	Sets timeout for connection login
`getLoginTimeout()`	Obtains login timeout value
`setLogWriter()`	Enables tracing of JDBC calls

Connection Object Methods

Method	Description
`close()`	Closes the connection to the datasource
`createStatement()`	Creates a Statement object for the connection
`prepareStatement()`	Prepares a parameterized SQL statement into a `PreparedStatement` for execution
`prepareCall()`	Prepares a parameterized call to a stored procedure or function into a `CallableStatement` for execution
`commit()`	Commits the current transaction on the connection
`rollback()`	Rolls back the current transaction on the connection
`setAutoCommit()`	Sets/resets autocommit mode on the connection
`getWarnings()`	Retrieves SQL warning(s) associated with a connection
`getMetaData`	Returns a `DatabaseMetaData` object with info about database

Statement Object Methods

Method	Description
Basic statement execution	
`executeUpdate()`	Executes a nonquery SQL statement and returns the number of rows affected
`executeQuery()`	Executes a single SQL query and returns a result set
`execute()`	General-purpose execution of one or more SQL statements
Statement batch execution	
`addBatch()`	Stores previously supplied parameter values as part of a batch of values for execution
`executeBatch()`	Executes a sequence of SQL statements; returns an array of integers indicating the number of rows impacted by each one
Query results limitation	
`setMaxRows()`	Limits number of rows retrieved by a query
`getMaxRows()`	Retrieves current maximum row limit setting
`setMaxFieldSize()`	Limits maximum size of any retrieved column
`getMaxFieldSize()`	Retrieves current maximum field size limit
`setQueryTimeout()`	Limits maximum time of query execution
`getQueryTimeout()`	Retrieves current maximum query time limit
Error handling	
`getWarnings()`	Retrieves SQL warning(s) associated with statement execution

ResultSet Object Methods

Method	Description
Cursor motion	
`next()`	Moves cursor to next row of query results
`close()`	Ends query processing; closes the cursor
Basic column-value retrieval	
`getInt()`	Retrieves integer value from specified column
`getShort()`	Retrieves short integer value from specified column
`getLong()`	Retrieves long integer value from specified column
`getFloat()`	Retrieves floating point numeric value from specified column
`getDouble()`	Retrieves double-precision floating point value from specified column
`getString()`	Retrieves character string value from specified column
`getBoolean()`	Retrieves true/false value from specified column
`getDate()`	Retrieves date value from specified column
`getTime()`	Retrieves time value from specified column
`getTimestamp()`	Retrieves timestamp value from specified column
`getByte()`	Retrieves byte value from specified column
`getBytes()`	Retrieves fixed-length or variable-length `BINARY` data from specified column
`getObject()`	Retrieves any type of data from specified column
Large object retrieval	
`getAsciiStream()`	Gets input stream object for processing a character large object (`CLOB`) column
`GetBinaryStream()`	Gets input stream object for processing a binary large object (`BLOB`) column
Other functions	
`getMetaData()`	Returns a `ResultSetMetaData` object with metadata for query
`getWarnings()`	Retrieves SQL warnings associated with the `ResultSet`

DatabaseMetaData Methods

Function	Description
`getTables()`	Returns result set of table information of tables in database
`getColumns()`	Returns result set of column names and type info, given table name
`getPrimaryKeys()`	Returns result set of primary key info, given table name
`getProcedures()`	Returns result set of stored procedure info
`getProcedureColumns()`	Returns result set of info about parameters for a specific stored procedure

SQLException Methods

Method	Description
`getMessage()`	Retrieves error message describing the exception
`getSQLState()`	Retrieves `SQLSTATE` value (5-char string, as described in Chapter 17)
`getErrorCode()`	Retrieves driver-specific or DBMS-specific error code
`getNextException()`	Moves to next SQL exception in a series

JDBC Classes and Interfaces

Steps to submit a database query:

- Load the JDBC driver
- Connect to the data source
- Execute SQL statements

JDBC Driver Management

- All drivers are managed by the DriverManager class
- Loading a JDBC driver:
  - In the Java code:
    Class.forName("com.mysql.jdbc.Driver").newInstance();

Connections in JDBC

We interact with a data source through sessions. Each connection identifies a logical session.

- JDBC URL:
  jdbc:<subprotocol>:<otherParameters>

Example:

```
String url= "jdbc:mysql://146.187.134.44:3306/danl_4”;
Connection con;
try{
    con = DriverManager.getConnection(url, username, password);
} catch (SQLException e) {
                e.printStackTrace();}
```

Executing SQL Statements

- Three different ways of executing SQL statements:
  - Statement (static statements)
  - PreparedStatement (dynamic SQL statements)
  - CallableStatment (stored procedures)

Executing SQL Statements (Contd.)

```
query = "select course_id, sec_id from section”+
        “ where year = 2010 and semester  = \‘Fall\’;";

resultSet = statement.executeQuery(query);
```

ResultSets

```
while (resultSet.next()) {
        // It is possible to get the columns via name
        // also possible to get the columns via the column number
        // which starts at 1
        // e.g. resultSet.getSTring(2);
        String cid = resultSet.getString(”course_id");
        String sid= resultSet.getString(”section_id");
        System.out.println(cid+" "+sid+"\n");
                }
```

ResultSets (Contd.)

A ResultSet is a very powerful cursor:

- previous(): moves one row back
- absolute(int num): moves to the row with the specified number
- relative (int num): moves forward or backward
- first() and last()

Matching Java and SQL Data Types

SQL data type	Java data type	
	Simply mappable	Object mappable
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

Examining Database Metadata

DatabaseMetaData object gives information about the database system and the catalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
   “Name:” + md.getDriverName() +
   “version: ” + md.getDriverVersion());
```

Stored Procedures

- What is a stored procedure:
  - A block of program executed through a single defined SQL routine.
  - Executed in the process space of the server
- Advantages:
  - Improve network performance
  - Can encapsulate application logic
  - Reuse of application logic by different users
  - Avoid tuple-at-a-time return of records through cursors

Capabilities of Stored Procedure

- **Conditional execution** An IF...THEN...ELSE structure allows a SQL procedure to test a condition and to carry out different operations depending on the result.
- **Looping** A WHILE or FOR loop or similar structure allows a sequence of SQL operations to be performed repeatedly, until some terminating condition is met.
- **Block structure** A sequence of SQL statements can be grouped into a single block and used in other flow-of-control constructs as if the statement block were a single statement.
- **Named variables** A SQL procedure may store a value that it has calculated, retrieved from the database, or derived in some other way into a program variable, and later retrieve the stored value for use in subsequent calculations.
- **Named procedures** A sequence of SQL statements may be grouped together, given a name, and assigned formal input and output parameters, like a subroutine or function in a conventional programming language. Once defined in this way, the procedure may be called by name, passing it appropriate values for its input parameters.

Calling Stored Procedures

- Once defined by the CREATE PROCEDURE statement, a stored procedure can be used.
- An application program may request execution of the stored procedure, using the appropriate SQL statement.
- Another stored procedure may call it to perform a specific function.
- The stored procedure may also be invoked through an interactive SQL interface.
- The various SQL dialects differ in the specific syntax used to call a stored procedure.

MySQL Stored Procedure Examples

```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
BEGIN
SELECT * FROM products;
END //
DELIMITER ;

CALL GetAllProducts();
```

MySQL Stored Procedure Examples

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
BEGIN
SELECT city, phone
FROM offices
WHERE country = countryName;
END //
DELIMITER ;
```

```
CALL GetOfficeByCountry('USA')
```

MySQL Stored Procedure Examples

```
DELIMITER $$
CREATE PROCEDURE CountOrderByStatus(
IN orderStatus VARCHAR(25),
OUT total INT)
BEGIN
SELECT count(orderNumber)
INTO total
FROM orders
WHERE status = orderStatus;
END$$
DELIMITER ;

CALL CountOrderByStatus('Shipped',@total);
SELECT @total AS total_shipped;
```

MySQL Stored Procedure Examples

```
DELIMITER $$
CREATE PROCEDURE `Capitalize`(INOUT str VARCHAR(1024))
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE myc, pc CHAR(1);
DECLARE outstr VARCHAR(1000) DEFAULT str;
WHILE i <= CHAR_LENGTH(str) DO
SET myc = SUBSTRING(str, i, 1);
SET pc = CASE WHEN i = 1 THEN ' '
ELSE SUBSTRING(str, i - 1, 1)
END;
IF pc IN (' ', '&', '"', '_', '?', ';', ':', '!', ',', '-', '/', '(', '.') THEN
SET outstr = INSERT(outstr, i, 1, UPPER(myc));
END IF;
SET i = i + 1;
END WHILE;
SET str = outstr;
END$$
DELIMITER ;
```

```
SET @str = 'mysql stored procedure tutorial';
CALL Capitalize(@str);
SELECT @str;
```

@str
Mysql Stored Procedure Tutorial

Using CallableStatements to Execute Stored Procedures in Java

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

Using CallableStatements to Execute Stored Procedures in Java: 1. Prepare the callable statement

```
import java.sql.CallableStatement;

...

    //
    // Prepare a call to the stored procedure 'demoSp'
    // with two parameters
    //
    // Notice the use of JDBC-escape syntax ({call ...})
    //

    CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");
```

2. Register the output parameters

```
import java.sql.Types;
```

Register output parameters in two ways:

```
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter(2, Types.INTEGER);

//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
```

3. Set the input parameters

```
//
// Set a parameter by index
//

cStmt.setString(1, "abcdefg");

//
// Alternatively, set a parameter using
// the parameter name
//

cStmt.setString("inputParameter", "abcdefg");

//
// Set the 'in/out' parameter using an index
//

cStmt.setInt(2, 1);

//
// Alternatively, set the 'in/out' parameter
// by name
//

cStmt.setInt("inOutParam", 1);
```

4. Execute the CallableStatement, and retrieve output parameters

Now it's ready to execute the stored procedure.

```
cStmt.execute()
```

Get the output in two ways:

```
int outputValue = cStmt.getInt(2); // index-based

outputValue = cStmt.getInt("inOutParam"); // name-based
```

Functions

- In addition to stored procedures, most SPL dialects support a stored *function* capability.
- The distinction is that a function returns a single thing (such as a data value, an object, or an XML document) each time it is invoked, while a stored procedure can return many things or nothing at all.
  - Support for returned values varies by SPL dialect.
- Functions are commonly used as column expressions in SELECT statements, and thus are invoked once per row in the result set, allowing the function to perform calculations, data conversion, and other processes to produce the returned value for the column.

Function Example

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in number)
                return number
as

/* Declare one local variable to hold the total */
tot_ord number(16,2);

begin
   /* Simple single-row query to get total */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* return the retrieved value as fcn value */
   return tot_ord;
end;
```

Call a Function

```
SELECT COMPANY, NAME
  FROM CUSTOMERS, SALESREPS
 WHERE CUST_REP = EMPL_NUM
   AND GET_TOT_ORDS(CUST_NUM) > 10000.00;
```

Triggers

- A trigger is a special set of stored procedural code whose activation is caused by modifications to the database contents.
- Unlike stored procedures, a trigger is not activated by a CALL or EXECUTE statement. Instead, the trigger is associated with a database table.
- Some DBMS brands allow definition of specific updates that cause a trigger to fire.
- Also, some DBMS brands, notably Oracle, allow triggers to be based on system events such as users connecting to the database or execution of a database shutdown command.

Trigger Example

```
Create or replace trigger upd_tgt
    /* Insert trigger for SALESREPS */
    before insert on salesreps
    for each row
    begin
        if :new.quota is not null
         then
           update offices
               set target = target + new.quota;
         end if;
    end;
```

Summary

- APIs such as JDBC introduce a layer of abstraction between application and DBMS
- Stored procedures execute application logic directly at the server