Programming with Databases

1DL301 Database Design I

APPLICATION PROGRAMMING INTERFACE (API)

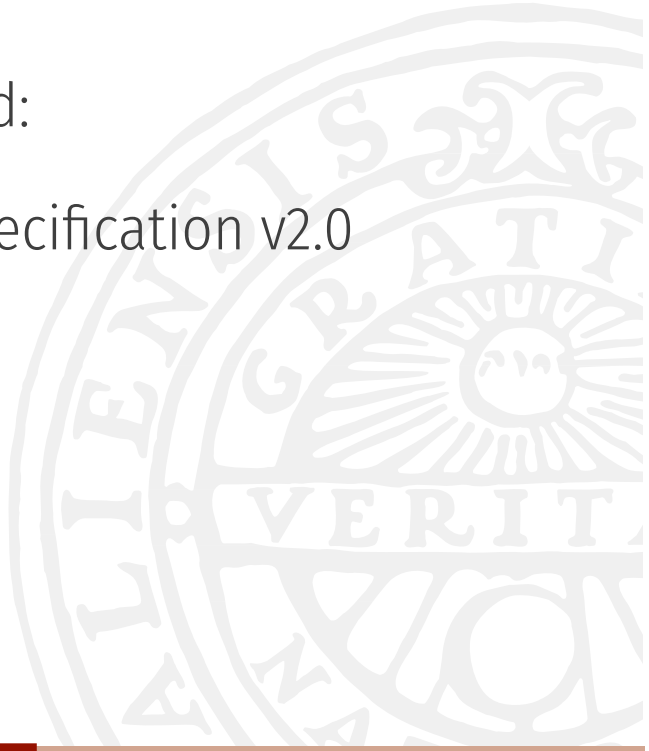
Application programming interface (API) is a set of functions allowing a program to communicate with a server or another software component.

In our case: a set of functions to query and modify data stored in a database.

Database APIs in Python and Java are standardized:

- In Python: PEP 249 – Python Database API Specification v2.0
- In Java: Java Database Connectivity (JDBC)

PEP = Python Enhancement Proposal



HOW TO USE THE DATABASE API IN PYTHON

1. Create a connection to the database:

   ```
   conn = module.connect(parameters)
   ```

   Replace *module* with the name of module you want to use, e. g. sqlite3 or MySQLdb. Check the documentation to find out what parameters you need to specify.

2. Create a cursor:

   ```
   cur = conn.cursor()
   ```

3. Repeat for each SQL statement or query:
   Execute the SQL statement or query:

   ```
   cur.execute(sql, parameters)
   ```

   For queries, fetch and process rows:

   ```
   for row in cur:
       do something with row
   ```

   Alternatively you can use fetchone(), fetchmany() or fetchall() methods to fetch the rows.

4. Close the cursor:

   ```
   cur.close()
   ```

5. Close the connection:

   ```
   conn.close()
   ```

PREPARED STATEMENTS

Prepared statements – SQL with placeholders which will be replaced by the values in a secure way.

Example:

```
INSERT INTO course (id, title) VALUES (%s, %s)
```

The placeholders (`%s`) will be replaced by the values in the second argument of `execute()`:

```
cur.execute('INSERT INTO course (id, title) VALUES (%s, %s)',
    (1, 'Database design 1'))
```

Reminder: Tuples with only one value must use a trailing comma, e. g.: `('Only one value in a tuple',)`

SQLITE AND MYSQLDB

SQLite

```
import sqlite3

conn = sqlite3.connect(filename, isolation_level=None)
```

SQLite3 uses `?` as the placeholder in prepared statements.

MySQLdb

```
import MySQLdb

conn = MySQLdb.connect(host, user, passwd, db)
conn.autocommit(True)
```

MySQLdb uses `%s` as the placeholder in prepared statements.

For SQLite, isolation_level=None switches on the autocommit mode.
In the autocommit mode each SQL is executed in its own transaction which is immediately committed.

MOST IMPORTANT METHODS

Most important methods on a `Connection` object:

- `.close()` – closes the connection
- `.commit()` – commits any pending transaction
- `.rollback()` – rollbacks any pending transaction
- `.cursor()` – creates and returns a new cursor

Most important properties/methods on a `Cursor` object:

- `.rowcount` – number of rows the last execute() produced
- `.execute(sql, params)` – prepares and executes a query or a command
- `.close()` – closes the cursor
- `.fetchone()` – fetches the next row of a query result
- `.fetchall()` – fetches all remaining rows of a query result

EXCEPTIONS

Exceptions are used for error handling:

- Warning
- Error
  - InterfaceError
  - DatabaseError
    - DataError
    - OperationalError
    - IntegrityError
    - InternalError
    - ProgrammingError
    - NotSupportedError

Check the documentation for more details.

EXAMPLE

```
1  import MySQLdb
2
3  conn = MySQLdb.connect("localhost", "root", "", "hotels")
4  conn.autocommit(True)
5  cursor = conn.cursor()
6
7  country = "NO"
8  min_rating = 4
9
10 cursor.execute("""SELECT Name, City, LowRate, HighRate
11 FROM Property
12 WHERE Country=%s AND StarRating >= %s""", (country, min_rating))
13
14 for (name, city, low_rate, high_rate) in cursor:
15     print("Hotel", name, "in", city, \
16         "- prices from", low_rate, "to", high_rate)
17
18 cursor.close()
19 conn.close()
```

JDBC

Java Database Connectivity (JDBC) is a Java API for communication with database systems.

The JDBC classes are in packages `java.sql` and `javax.sql`. The most relevant classes:

- `java.sql.DriverManager`
- `java.sql.Connection`
- `java.sql.Statement`
- `java.sql.PreparedStatement`
- `java.sql.ResultSet`

JDBC DRIVERS

JDBC API calls are “translated” into requests to the database server by a vendor-specific JDBC driver.

MySQL JDBC Driver (Connector), including the documentation and the source files, can be downloaded from:

http://dev.mysql.com/downloads/connector/j/

The JDBC driver must be included in the classpath:

```
javac MyClass.java
java -cp /path/to/mysql-connector-java-8.0.18.jar:. MyClass
```

CONNECTING TO THE DATABASE

```
1 Connection conn = DriverManager.getConnection(url, username, password);
```

The url must be in form `jdbc:`*vendor*`:`*vendor-specific data*

For MySQL:

`jdbc:mysql://`*hostname*[`:`*port*]`/`*dbname*

PREPARED STATEMENTS IN JAVA

Prepared statements in Java use `?` as placeholders.

```
PreparedStatement pstmt = conn.prepareStatement(
  "SELECT id, name FROM employee WHERE department_id = ?");
```

Each parameter has an index (first placeholder has index 1, second 2, etc.) and its datatype determines which of PreparedStatement's methods must be used to set its value:

```
.setInt(index, value)      .setLong(index, value)
.setDouble(index, value)   .setDate(index, value)
.setString(index, value)   ...
```

Example (setting department_id to 4):

```
pstmt.setInt(1, 4);
```

PREPARED STATEMENTS IN JAVA, CONT'D

Methods for executing SQL statements:

- `.executeQuery()` – executes a SELECT query and returns a ResultSet object
- `.executeUpdate()` – executes an INSERT, UPDATE or DELETE and returns number of affected rows
- `.execute()` – can be used for other SQL commands

Example:

```
ResultSet rs = pstmt.executeQuery();
```

Note: Prepared statements can be reused with new parameter values.

RESULT SETS

Rows in a result set are access one by one in a loop.

The ResultSet object maintains a “cursor” which points to the current row.

ResultSet’s `next()` method moves the cursor to the next row if it exists and returns true, otherwise returns false.

Initially (after executing the query) the cursor is position before the first row. Use `next()` to move to the first row.

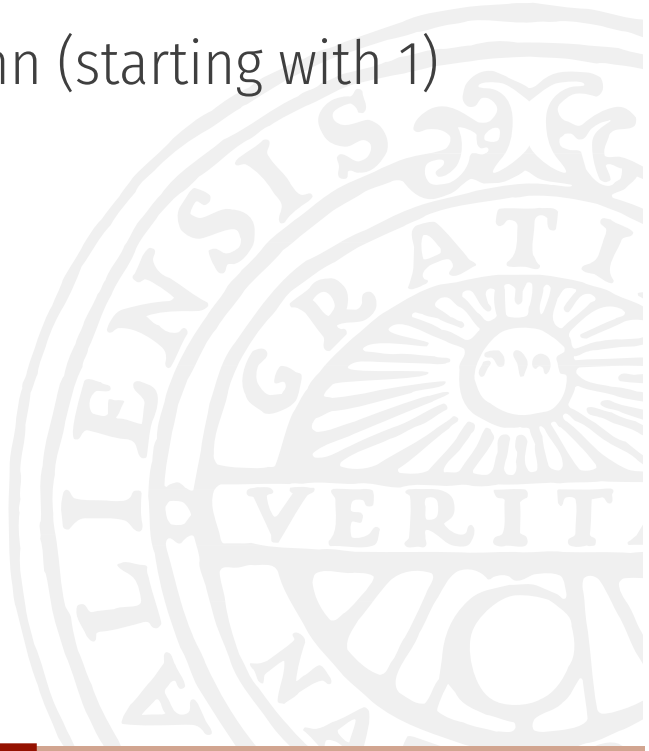
RESULT SETS, CONT'D

To get the value for an attribute one of ResultSet's methods must be used based on the attribute's datatype:

```
.getInt(col)      .getLong(col)
.getDouble(col)   .getDate(col)
.getString(col)   ...
```

The `col` parameter is either the index of the column (starting with 1) or its name/alias.

```
while (rs.next()) {
    System.out.printf(
        "%4d | %32s\n",
        rs.getInt(1),
        rs.getString(2),
    );
}
```



EXCEPTIONS

`Class.forName` may throw a `ClassNotFoundException`.

JDBC methods may throw an `SQLException`.

EXAMPLE IN JAVA

```java
import java.sql.*;

public class Example
{
    static final String URL = "jdbc:mysql://localhost/hotels?useSSL=false";
    static final String USERNAME = "root";
    static final String PASSWORD = "";

    public static void main(String[] args) throws ClassNotFoundException, SQLException
    {
        Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement stmt = conn.prepareStatement("SELECT Name, City, LowRate, HighRate"
          + " FROM Property WHERE Country=? AND StarRating >= ?");
        stmt.setString(1, "NO");
        stmt.setInt(2, 4);
        ResultSet rs = stmt.executeQuery();
        while (rs.next())
        {
            System.out.printf("Hotel %s in %s - prices from %.2f to %.2f\n",
                rs.getString(1), rs.getString(2), rs.getDouble(3), rs.getDouble(4));
        }
        stmt.close();
        conn.close();
    }
}
```

ACKNOWLEDGEMENT

The slide is derived from
Jan Kudlicka's DB1-Fall19 class.