# Multiple Choice Questions

**Question 1**

Which of the following statements is NOT CORRECT?

1. Embedded database APIs can use early binding.
2. Embedded database APIs can use late binding.
3. Call-level database APIs can use early binding.
4. Call-level database APIs can use late binding.

**Question 2**

Which of the following statements is NOT CORRECT?

1. One drawback of ODBC is that the architecture is mostly native to Microsoft-based platforms.
2. One drawback of ODBC is that application code needs to be modified every time a different driver needs to be used.
3. One drawback of ODBC is it is not using an object-oriented paradigm.
4. One drawback of ODBC is that performance can be worse compared to proprietary DBMS APIs.

**Question 3**

Which of the following statements is CORRECT?

1. JDBC drivers come in different types, which come with different tradeoffs in terms of portability and performance.
2. JDBC was originally developed to be used in the C++ programming language.
3. JDBC can only be used on Linux- and Unix-based systems.
4. Just like ODBC, JDBC does no expose programmer-friendly object classes to work with.

**Question 4**

Which JDBC driver type is implemented completely in Java and communicates directly with the vendor’s DBMS through a network socket connection?

1. Type 1.
2. Type 2.
3. Type 3.
4. Type 4.

**Question 5**

Which of the following statements is NOT CORRECT?

1. SQLJ uses a pe-compiler to translate embedded SQL statement before invoking the Java compiler.
2. SQL syntax can be checked before runtime using SQLJ.
3. JDBC uses SQLJ as an underlying technology.
4. Many IDEs do not have SQLJ support.

**Question 6**

Which database access technique does the following C# statement illustrate?

public void Example() {

DataClassesContext dc = new DataClassesContext();

var q=

from a in dc.GetTable<Order>()

where a.CustomerName.StartsWith(“Seppe”)

select a;

dataGrid.DataSource = q;

}

1. JDBC.
2. ODBC.
3. Language-integrated queries.
4. None of the above.

**Question 7**

Which of the following statements is CORRECT?

1. A centralized DBMS uses a mainframe that only handles the execution of applications and the querying of the results, but not the interface to the user.
2. A fat client makes sure that there is a tight coupling between the way the application works and interface with the user.
3. A fat server variant presents the results to the user, runs the applications and executes the queries on the server.
4. None of the above.

**Question 8**

Which of the following statements is NOT CORRECT?

1. If the marketing department runs its applications on separate server which interacts with the DMBS, the company uses a three-tier architecture.
2. Browsing the internet can be seen as an n-tier activity where the internet browser handles the presentation logic and the web server the application logic.
3. The advantage of working with a separate web and application server is it adds an extra layer of security, since the application server can only be called be the web server.
4. Working with a traditional client-server set-up leads to a lot of maintenance issues since all the PCs must always be up-to-date.

**Question 9**

Which statement is NOT CORRECT?

1. ADO.net is an example of proprietary API since it is native to Microsoft platforms.
2. An advantage of an API is that the users do not have to be concerned with the underlying data structures of the DBMS.
3. The main of goal of an API is to provide an interface to programmers, application developers and end-users to interact with the data from the DBMS.
4. An embedded API like SQLJ uses a separate pre-compiler to parse the SQL code and translate it to host language.

**Question 10**

Consider the following two statements:

1. Early binding can only be used with embedded APIs, just as late binding can only be used with call-level APIs.
2. A drawback of dynamic SQL is that syntax errors can only be spotted when the program is wrongly executed.

Which of the following statements is CORRECT?

1. Statement 1 and 2 are correct.
2. Only statement 1 is correct.
3. Only statement 2 is correct.
4. Both statement are incorrect.

**Question 11**

Which step is NOT CORRECT if you want to retrieve the vendor name?

1. The user must first set up a connection with the driver using registerDriver method from the DriverManager class
2. The user must open a connection with the getConnection method to get a Connection object.
3. The getMetaData method of the connection object can be used to retrieve metadata such as the vendor name.
4. The Connection object should be used to create a Statement object which can then be used to retrieve the vendor name.

**Question 12**

What statement about JDBC is CORRECT?

1. The cursor mechanism was created to solve the impedance mismatch problem between the record-oriented SQL language and the set-oriented Java language.
2. A cursor mechanism will iterate over the records in the ResultSet just like the cursor in Word keeps track of the position in the text.
3. Stored procedures should be executed with the PrepareStatement method.
4. A ResultSet object is a collection of records retrieved by a SELECT query from the exceuteQuery object from a Statement method.

**Question 13**

Which of the following statements is NOT CORRECT?

1. One of the advantages of JDBC is that the Connection object allows you to manually coordinate transactions.
2. A downside of dynamic SQL is that interactive querying via a graphical user interface can lead to wrongly inserted tuples or even deleted tables.
3. In static SQL the colon operator (“:”) is used to perform additional checks on the input or output variables.
4. Syntactic and semantic issues can only be solved in dynamic SQL by using prepared statements.

**Question 14**

Suppose you have created the following PreparedStatement:

String selectQuery = "select prodname from product where available\_quantity > ? and available\_quantity < ?";

Statement preparedSelectStatement = conn.prepareStatement(selectQuery);

If you want to retrieve the products with more than 50 items available and less then 100. What JDBC code is correct?



ResultSet resultSet1 = preparedSelectStatement.executeQuery();

preparedSelectStatement.setInt(1,50);

preparedSelectStatement.setInt(2,100);



preparedSelectStatement.setInt(1,50);

preparedSelectStatement.setInt(1,100);

ResultSet resultSet1 = preparedSelectStatement.executeQuery();



preparedSelectStatement.setInt(1,50);

preparedSelectStatement.setInt(2,100);

ResultSet resultSet1 = preparedSelectStatement.executeQuery();

1. None of the above.

**Question 15**

Consider the following JDBC code fragment. Which of the statements below is NOT CORRECT?

String deleteQuery = “delete from books where num\_pages <= 30”;

Statement deleteStatement = conn.createStatement();

int deletedRows = deleteStatement.executeUpdate(deleteQuery);

System.out.println(deletedRows + “ books were deleted”);

1. The return value of the executeUpdate(deleteQuery) method is an integer.
2. This code fragment will not work, because it lacks a ResultSet object.
3. This code fragment would not work properly if the query was a ‘SELECT’ query.
4. This code fragment illustrates late binding.

# **Open Questions**

**Question 1**

Explain the differences between a centralized system and tiered system architecture.

**Answer:**

In a **centralized DBMS architecture**, all responsibilities of the DBMSs are handled by one, centralized entity, meaning that the DBMS logic, the data itself, as well as the application logic and presentation logic (also called the user interface), are all handled by the same system.

A **tiered architecture** aims to decouple this centralized setup by combining the computing capabilities of powerful central computers with the flexibility of PCs. The latter act as active clients, requesting services from the former, the passive server.

It is possible to decouple application logic from the DBMS and place this in a separate layer (i.e., on an application server). This setup is referred to as a **three-tier architecture**. This setup allows different applications on the application server (e.g., a marketing application, a logistics application, an accounting application, etc.) to access the same database. If the applications on the application server, and/or the functionalities on the other tiers, are in turn spread over multiple tiers, we speak of an n-tier architecture.

Note that what we talk about an “application server” or “database server”, these can in fact consist of multiple physical, distributed machines. This becomes especially relevant when talking about a cloud-based DBMS architecture, where the DBMS and database will be hosted by a third-party provider. The DBMS functionality and data itself can then be distributed across multiple computers.

**Question 2**

What is meant with a “fat” client versus a “thin” one? Are web browsers fat or thin clients?

**Answer:**

Both are variants of the two-tier architecture (or also: a client-server architecture). In the **“fat” client variant**, the presentation logic and application logic are handled by the client (i.e., the PC). This is common in cases where it makes sense to couple an application’s workflow (e.g. opening of windows, screens, and forms) with its look-and-feel (i.e., its front-end, the way how it shows itself to the user). The DBMS, however, now fully runs on the database server. In a second variant of this setup, only the presentation logic is handled by the client. Applications and database commands are both executed on the server. This form is common when application logic and database logic are very tightly coupled or similar. This variant is denoted as a **“fat” server**, or also: thin client architecture.

A common, modern example of an n-tier architecture consists of a web browser acting as the client (i.e., handling some of the presentation logic, drawing the user interface and handling user inputs), which sends and receives web requests to a web server. To prepare the result page to be sent back to the client, the web server can then initiate multiple queries to the database server. It is hence an example of an n-tier setup, where the web browser can be regarded as a thin client with some “fat” presentation components, as it is the web browser which will render the retrieved contents, draw forms, and so on. This being said, many thin client setups these days come with just a bare-bones operating system and a web browser installed on the PC, with all the advanced application and database logic being handled by servers on the web or a company’s Intranet. In fact, Google’s Chromebook laptops are a well-known example of consumer-geared thin machines.

**Question 3**

Explain the differences between “static” and “dynamic” SQL and how this relates to early and late binding.

**Answer:**

SQL binding refers to the translation of SQL code to a lower-level representation that can be executed by the DBMS, after performing tasks such as validation of table and field names, checking whether the user or client has sufficient access rights, and generating a query plan to access the physical data in the most performant way possible. Early versus late binding then refers to the actual moment when this binding step is performed.

The distinction between early and late binding coincides with the distinction between embedded and call-level APIs. That is, early binding typically occurs in case a pre-compiler is used, which will then perform this binding step, and is therefore mostly paired with an embedded API. Late binding, on the other hand, performs the binding of SQL-statements at runtime (i.e., during the actual execution of the application). The benefit of this approach is the additional flexibility it offers: SQL statements can be generated at runtime, so this is also referred to as “dynamic SQL”, rather than the “static SQL” as employed by early binding.

**Question 4**

DBMSs such as Microsoft Access and SQLite are oftentimes described as embedded databases. Does this mean they are accessed using embedded APIs? Explain why / why not.

**Answer:**

An **embedded API** embeds SQL statements in the host programming language, or host language for short, meaning that the SQL statement(s) will end up being an integral part of the source code of a program. Before the program is compiled, a “SQL pre-compiler” parses the SQL-related instructions and replaces these with source code instructions native to the host programming language used, invoking a separate code library. The converted source code is then sent to the actual compiler to construct a runnable program.

A **call-level APIs** passes SQL instructions to the DBMS by means of direct calls to a series of procedures, functions or methods as provided by the API to perform the necessary actions, such as setting up a database connection, sending queries and iterating over the query result).

Some call-level APIs might be easily confused with embedded APIs, especially for some DBMS systems that denote themselves as being an **embedded DBMS**. An embedded DBMS is not the same as an embedded API, and an embedded DBMS will oftentimes utilize a call-level API! One example is SQLite. In contrast to many other DBMSs, SQLite does not operate in a client-server environment. Instead, the DBMS is completely contained (hence: embedded) in a single library (written in C), which in turn is embedded into the application itself, so that the DBMS becomes an integral part of the actual application. Invoking the SQLite API from a host-language, however, is performed in a call-level manner, not using an embedded API. No pre-compiler is present.