Database Design and Implementation
Chapter 11 Conceptual Exercises

Questions	Answers
11.1.	**Client-Side Objects:** 1. **NetworkDriver Instance:** Created when the client code initializes a new **NetworkDriver**. 2. **Registry Object:** The **Registry** object, **reg**, is created to interact with the RMI registry. 3. **RemoteDriver Stub:** This is the client-side proxy for the server's **RemoteDriver** service. 4. **RemoteConnection Stub:** Obtained from the **RemoteDriver** stub, it represents the server's **RemoteConnection** service. 5. **NetworkConnection:** Wraps the **RemoteConnection** stub and implements the **java.sql.Connection** interface, providing the client-side representation of the server connection. **Server-Side Objects:** 1. **RMI Registry:** Holds references to remote objects. It's not explicitly created here, but we assume it is already running on the server and listening on the default RMI port (1099). 2. **RemoteDriver Service:** An instance of the server's implementation of **RemoteDriver**, which would be bound in the server's RMI registry. 3. **RemoteConnection Implementation:** When **rdvr.connect()** is called, the server-side implementation of **RemoteConnection** is created. **Threads:** • **Client-Side Threads:** No new threads are explicitly created by the **NetworkDriver**. It relies on the underlying RMI infrastructure, which will manage threads for remote method invocations as needed. • **Server-Side Threads:** The RMI registry and **RemoteDriver** service would run within their own threads, managed by the RMI runtime. A new thread for each client connection may be spawned by the RMI framework to handle the remote method calls.

11.2.	The suggestion of passing a transaction to each **RemoteStatementImpl** object at creation via its constructor could cause issues:
	
	• **Concurrent Transactions Issue:**
	  • **Initial Analysis**: If a single **RemoteStatementImpl** instance holds a reference to a particular transaction, concurrent execution of **executeQuery** or **executeUpdate** by multiple threads could lead to race conditions, as they would be attempting to manipulate the same transaction instance.
	  • **Hindsight Adjustment**: This approach would also be problematic in scenarios where transaction semantics require that operations

	be isolated from each other. Sharing the same transaction across multiple statements could inadvertently lead to a loss of isolation.
	
	• **Transaction Lifecycle Management:**
	  • **Initial Analysis**: The lifecycle of a transaction is typically managed separately from the creation of statement objects to allow for flexibility in controlling transaction boundaries. Transactions may be committed or rolled back independently of the statements executed within them.
	  • **Hindsight Adjustment**: In this design, if a transaction fails and needs to be rolled back, all statements created with that transaction would be affected. The client code would lack the control to manage transaction boundaries effectively.

11.3.	For the client configuration,
	• **Remote Implementation Classes (Server-Side):**
	  • These are likely **not needed** on the client-side because the client interacts with the server-side objects via network protocols.
	  • It is important to confirm that the client does not instantiate any server-side classes directly but instead uses some form of proxy or stub objects.
	• **Remote Interfaces (Client-Side):**
	  • The client does need the remote interfaces because they define the contract that the client-side stubs must adhere to in order to communicate with the server-side implementations.
	  • These interfaces are **critical** for ensuring that the client knows what methods it can call on the server objects, even though the actual implementation resides on the server.
	
	In configuring the client, these would be kept:
	• All classes in the **simpledb.jdbc.network** package that are used by the client to establish connections and communicate with the server.
	• Remote interfaces which are implemented by the server-side objects.
	
	These can be removed:
	• Server-side implementation classes of the remote interfaces as the client will only use stubs or proxies for communication.
	• Any utility classes that are solely used within the server context.
	
	The **sql** and **remote** folders would need to be scrutinized to identify the specific classes that are only relevant to the server-side implementation and can be removed from the client configuration.