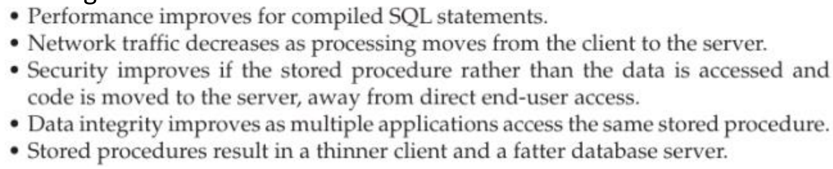
KEY CONSIDERATIONS IN THREE-TIER APPLICATIONS

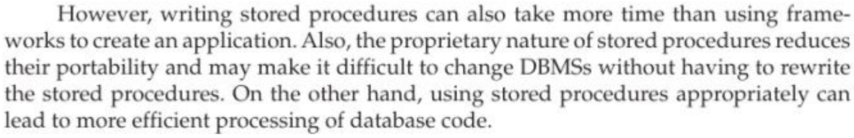
Stored Procedures (procedures or in other words sql functions)

- are modules of code that implement application logic and are included on the database server.
- Advantages:



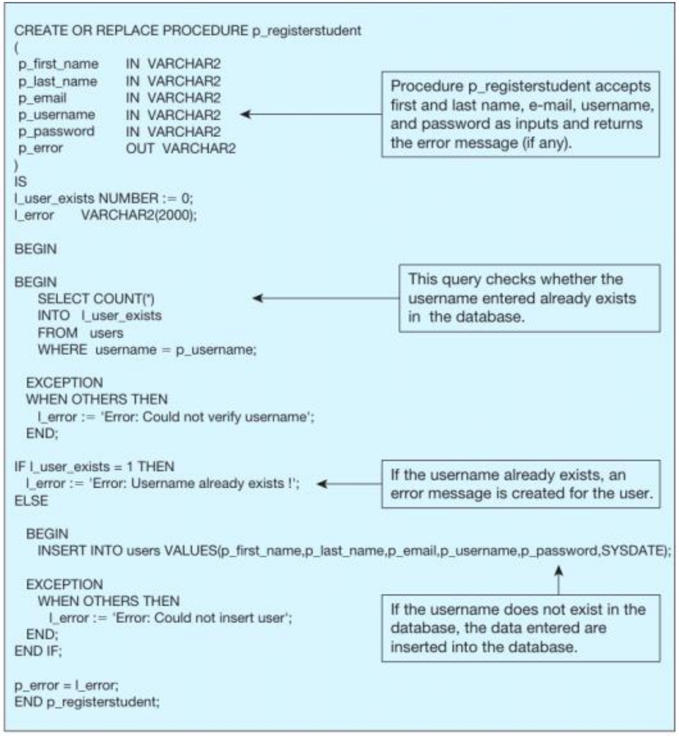
- Performance improves for compiled SQL statements.
- Network traffic decreases as processing moves from the client to the server.
- Security improves if the stored procedure rather than the data is accessed and code is moved to the server, away from direct end-user access.
- Data integrity improves as multiple applications access the same stored procedure.
- Stored procedures result in a thinner client and a fatter database server.

- Pros and cons



However, writing stored procedures can also take more time than using frameworks to create an application. Also, the proprietary nature of stored procedures reduces their portability and may make it difficult to change DBMSs without having to rewrite the stored procedures. On the other hand, using stored procedures appropriately can lead to more efficient processing of database code.

- Showing how to create a procedure in Oracle sql



```
CREATE OR REPLACE PROCEDURE p_registerstudent
(
 p_first_name     IN  VARCHAR2
 p_last_name      IN  VARCHAR2
 p_email          IN  VARCHAR2
 p_username       IN  VARCHAR2
 p_password       IN  VARCHAR2
 p_error          OUT  VARCHAR2
)
IS
l_user_exists NUMBER := 0;
l_error      VARCHAR2(2000);

BEGIN

BEGIN
    SELECT COUNT(*)
    INTO  l_user_exists
    FROM  users
    WHERE  username = p_username;

  EXCEPTION
  WHEN OTHERS THEN
    l_error := 'Error: Could not verify username';
  END;

IF l_user_exists = 1 THEN
  l_error := 'Error: Username already exists !';
ELSE

  BEGIN
    INSERT INTO users VALUES(p_first_name,p_last_name,p_email,p_username,p_password,SYSDATE);

  EXCEPTION
    WHEN OTHERS THEN
      l_error := 'Error: Could not insert user';
  END;
END IF;

p_error = l_error;
END p_registerstudent;
```

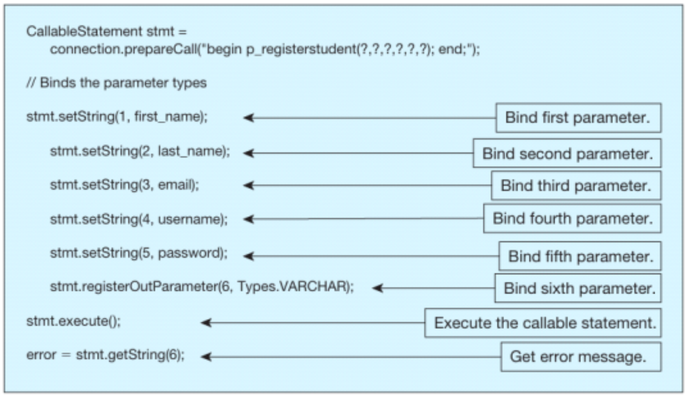
Procedure p_registerstudent accepts first and last name, e-mail, username, and password as inputs and returns the error message (if any).

This query checks whether the username entered already exists in the database.

If the username already exists, an error message is created for the user.

If the username does not exist in the database, the data entered are inserted into the database.

- Showing how to call that procedure in Java



```
CallableStatement stmt =
    connection.prepareCall("begin p_registerstudent(?,?,?,?,?,?); end;");

// Binds the parameter types

stmt.setString(1, first_name);

    stmt.setString(2, last_name);

    stmt.setString(3, email);

    stmt.setString(4, username);

    stmt.setString(5, password);

    stmt.registerOutParameter(6, Types.VARCHAR);

stmt.execute();

error = stmt.getString(6);
```

Bind first parameter.

Bind second parameter.

Bind third parameter.

Bind fourth parameter.

Bind fifth parameter.

Bind sixth parameter.

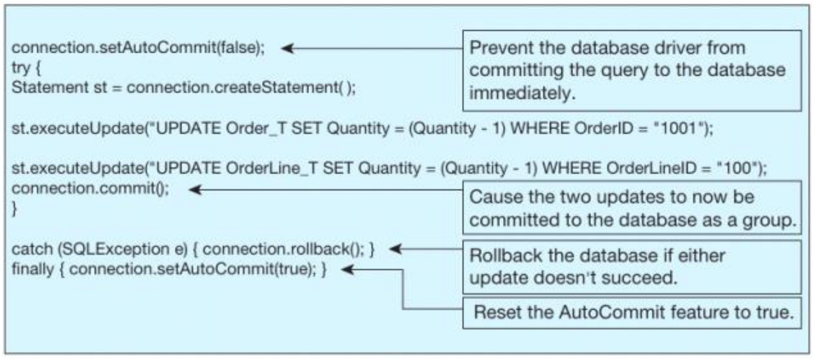
Execute the callable statement.

Get error message.

Transactions: I think this is when you have multiple calls being made to update in some way a record in your table and you need to be able to handle it like if multiple people want to buy the same ring from amazon

- Given that there might be thousands of users simultaneously trying to access and/or update a database through a Web application at any given point time (think Amazon.com or eBay), application developers need to be well versed in the concepts of database transactions and need to use them appropriately when developing applications.

- Sample java code snipper for a sql transaction



```
connection.setAutoCommit(false);
try {
Statement st = connection.createStatement( );

st.executeUpdate("UPDATE Order_T SET Quantity = (Quantity - 1) WHERE OrderID = "1001");

st.executeUpdate("UPDATE OrderLine_T SET Quantity = (Quantity - 1) WHERE OrderLineID = "100");
connection.commit();
}

catch (SQLException e) { connection.rollback(); }
finally { connection.setAutoCommit(true); }
```

Prevent the database driver from committing the query to the database immediately.

Cause the two updates to now be committed to the database as a group.

Rollback the database if either update doesn't succeed.

Reset the AutoCommit feature to true.

Database Connections

In most three-tier applications, while it is very common to have the Web servers and application servers located on the same physical machine, the database server is often located on a different machine. In this scenario, the act of making a database connection and keeping the connection alive can be very resource intensive. Further, most databases allow only a limited number of connections to be open at any given time. This can be challenging for applications that are being accessed via the Internet because it is difficult to predict the number of users. Luckily, most database drivers can relieve application developers of the burden of managing database connections by using the concept of connection pooling. However, application developers should still be careful about how often they make connections to a database and how long they keep a connection open within their application program.

Key Benefits of Three-Tier Applications

- ***Scalability*** Three-tier architectures are more scalable than two-tier architectures. For example, the middle tier can be used to reduce the load on a database server by using a transaction processing (TP) monitor to reduce the number of connections to a server, and additional application servers can be added to distribute application processing. A TP monitor is a program that controls data transfer between clients and servers to provide a consistent environment for online transaction processing.
- ***Technological flexibility*** It is easier to change DBMS engines (although triggers and stored procedures will need to be rewritten) with a **three-tier architecture**. The middle tier can even be moved to a different platform. Simplified presentation services make it easier to implement various desired interfaces, such as Web browsers or kiosks.
- ***Lower long-term costs*** Use of off-the-shelf components or services in the middle tier can reduce costs, as can substitution of modules within an application rather than an entire application.
- ***Better match of systems to business needs*** New modules can be built to support specific business needs rather than building more general, complete applications.
- ***Improved customer service*** Multiple interfaces on different clients can access the same business processes.
- ***Competitive advantage*** The ability to react to business changes quickly by changing small modules of code rather than entire applications, can be used to gain a competitive advantage.
- ***Reduced risk*** Again, the ability to implement small modules of code quickly and combine them with code purchased from vendors limits the risk assumed with a large-scale development project.