Database-driven decision support

- Effective decision support systems require flexibility in interacting with the user
- Decision makers may not always know exactly what questions they want to (need to) ask
  - It is thus important to support "what if?" questions
- Pre-determining a list of potential queries can be very limiting

Parameterized queries

- Using parameters within a query allows the user to decide exactly what data to return
  - The database programmer sets up the structure (template) for the SQL statement, but leaves certain values unspecified.
- Greatly improves the usability of the system
  - Frequently used in web-based DSS
  - Also used in desktop applications (NET → MS Access)
  - …and also can be used within the actual DBMS
Calling parameterized queries externally

- Within MS Access, parameterized queries can be used by building userforms and collecting the parameter values as variables from the user.
- Such queries can also be called from other applications using a DB provider (or driver):
  - PHP / ASP.NET / VB.NET / Java, etc.

Ex: Dim cmd as New OleDbCommand("SELECT * FROM Customer WHERE Cust_LName = ?", cn)

DBMS-specific formats

- **Access / MySQLi** (parameter order matters)
  "SELECT * FROM table1 WHERE col1 = ? OR col2 LIKE ?"

- **SQL Server** (supports named parameters)
  "SELECT * FROM table1 WHERE col1 = @someCol1 OR col2 LIKE @theColour"

- **Oracle**
  "SELECT * FROM table1 WHERE col1 = :someCol1 OR col2 LIKE :theColour"

Advantages of parameterized queries

- Query can be compiled once at beginning of process
  - Parameter values can then be set as needed
- Data types can be enforced without focus on syntax
  
  SELECT COUNT(Lunch_ID) FROM Lunch
  WHERE Emp_ID = ? AND Lunch_Date = ?;
- Security
  - Can help to protect against **SQL Injection**
SQL Injection

• Consists of inserting or "injecting" SQL code via data that is sent to a DBMS, when (non-parameterized) dynamic SQL is used:

  Ex: strSql = "SELECT * FROM employee WHERE emp_lname=" & lastname & ";

    \[ lastname = "Perkins"; DROP TABLE employee;--\]

    strSql = "SELECT * FROM employee WHERE emp_lname=Perkins; DROP TABLE employee;--";

SQL Injection (cont.)

• Because a parameterized query is expecting only the values for the missing parameters, it will protect against many SQL injection attacks

  strSql = "SELECT * FROM employee WHERE emp_lname = ?;"

  OleDbCommand cmd = new OleDbCommand(strSql);

  cmd.Parameters.Add(new OleDbParameter("Custname", OleDbType.VarChar));

  cmdParameters["Custname"].Value = "Perkins";

SQL Injection (cont.)

OleDbCommand cmd = new OleDbCommand(strSql);

cmd.Parameters.Add(new OleDbParameter("Custname", OleDbType.VarChar));

cmdParameters["Custname"].Value = "Perkins";