Introduction to SQL, OleDB interface to Access from VB.NET

SQL

• Structured Query Language, abbreviated SQL
  – Usually pronounced “sequel” but also “ess-cue-ell”
  – The common language of client/server database management systems.
  – Standardized – you can use a common set of SQL statements with all SQL-compliant systems.
  – Defined by E.F. Codd at IBM research in 1970.
  – Based on relational algebra and predicate logic
SQL Data Retrieval

• Given an existing database, the SELECT statement is the basic statement for data retrieval.
  – Both simple and complex, and it may be combined with other functions for greater flexibility.

```sql
SELECT data_element1 [, (data_element2 | function(..)) ] Or *
FROM table_1, [, table_2, ...]
WHERE condition_1 [, {not, or, and} condition_2] ]
GROUP BY data_1, ...
HAVING aggregate function(…)...
ORDER BY data1, ...
```

SELECT statement

• Some sample aggregate functions:
  – COUNT(*) SUM(item)
  – AVG(item) MAX(item)
  – MIN(item)

• Conditional Operators
  – = Equal
  – < Less than
  – > Greater than
  – <>,!= Not equal to
  – <= Less than or equal to
  – >= Greater than or equal to
SELECT Examples

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>PROD_ID</th>
<th>COST</th>
<th>SALESPERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>P999</td>
<td>20</td>
<td>Jones</td>
</tr>
<tr>
<td>101</td>
<td>P999</td>
<td>30</td>
<td>Jones</td>
</tr>
<tr>
<td>101</td>
<td>X310</td>
<td>500</td>
<td>Parker</td>
</tr>
<tr>
<td>102</td>
<td>Z225</td>
<td>15</td>
<td>Smith</td>
</tr>
</tbody>
</table>

• Select every row, column from the table:
  – SELECT * FROM Orders;
  – SELECT Orders.cust_id, Orders.prod_id, Orders.cost, Orders.salesperson
    FROM Orders;
• Returns a set of all rows that match the query

SELECT

• If a table has spaces or certain punctuation in it, then Access needs to have the items enclosed in square brackets []. The previous query is identical to the following:
  – SELECT [orders].[cust_id], orders.prod_id, orders.cost, orders.[salesperson]
    FROM Orders;
SELECT Query in Access

- Can flip back and forth between SQL View, Run, and Design Mode

```sql
SELECT Students.LastName, Students.FirstName, Students.DOB, Students.Address
FROM Students;
```

- More SELECT Statements

  - Note that we can have duplicates as a result of the selection. If we want to remove duplicates, we can use the DISTINCT clause:

    ```sql
    SELECT DISTINCT Orders.cust_id
    FROM Orders;
    ```

  - We can combine a selection and a projection by using the WHERE clause:

    ```sql
    SELECT Orders.cust_id
    FROM Orders
    WHERE Salesperson = "Jones";
    ```

  - This could be used if we wanted to get all the customers that Jones has sold to, in this case, CUST_ID=101 and CUST_ID=100. By default, Access is not case-sensitive, so “jones” would also result in the same table.
More SELECT

- We can further refine the query by adding AND, OR, or NOT conditions. If we want orders from Jones or from Smith then the query becomes:

  ```
  SELECT Orders.cust_id
  FROM Orders
  WHERE Salesperson = "Jones" or Salesperson = "Smith";
  ```

- Another refinement is to use the BETWEEN operator. If we want only those orders between 10 and 100 then we could define this as:

  ```
  SELECT Orders.cust_id, Orders.cost
  FROM Orders
  WHERE Orders.cost >10 and Orders.cost <100;
  ```

- Or use the between operator:

  ```
  SELECT Orders.cust_id, Orders.cost
  FROM Orders
  WHERE Orders.cost BETWEEN 10 and 100;
  ```

- Finally, we might want to sort the data on some field. We can use the ORDER BY clause:

  ```
  SELECT Orders.cust_id, Orders.cost
  FROM Orders
  WHERE Orders.cost >10 and Orders.cost <100
  ORDER BY Orders.cost;
  ```

- This sorts the data in ascending order of cost. An example is shown in the table:

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>15</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>101</td>
<td>30</td>
</tr>
</tbody>
</table>

- If we wanted to sort them in descending order, use the DESC keyword:

  ```
  SELECT Orders.cust_id, Orders.cost
  FROM Orders
  WHERE Orders.cost >10 and Orders.cost <100
  ORDER BY Orders.cost DESC;
  ```
Joining Data from Multiple Tables

• If our data is in multiple tables we can join them together in one query.
  – Use a JOIN operator (Access default w/Design view)
  – Add tables to the FROM, WHERE section (what we will use here)

• Say we have the following table in addition to Orders:

<table>
<thead>
<tr>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>

• Say we have the following table in addition to Orders:

<table>
<thead>
<tr>
<th>ORDERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>

SELECT Orders.cust_id, Customer.Cust_Name
FROM Orders, Customer
WHERE Orders.cost >10 and Orders.cost <100;

Result:

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>Cust_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Thomas Jefferson</td>
</tr>
<tr>
<td>101</td>
<td>Thomas Jefferson</td>
</tr>
<tr>
<td>102</td>
<td>Thomas Jefferson</td>
</tr>
<tr>
<td>100</td>
<td>Bill Clinton</td>
</tr>
<tr>
<td>101</td>
<td>Bill Clinton</td>
</tr>
<tr>
<td>102</td>
<td>Bill Clinton</td>
</tr>
<tr>
<td>100</td>
<td>George Bush</td>
</tr>
<tr>
<td>101</td>
<td>George Bush</td>
</tr>
<tr>
<td>102</td>
<td>George Bush</td>
</tr>
</tbody>
</table>

PRODUCT of two tables!

• What do you expect from this query?
Multiple Tables

- Need to link the tables by their common field, the customer ID:

```sql
SELECT Orders.cust_id, Customer.Cust_Name
FROM Orders, Customer
WHERE Orders.cust_id = Customer.Cust_Id and
Orders.cost >10 and Orders.cost <100;
```

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_NAME</th>
<th>MEMBER_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Thomas Jefferson</td>
<td>9/27/99</td>
</tr>
<tr>
<td>101</td>
<td>Bill Clinton</td>
<td>9/26/99</td>
</tr>
<tr>
<td>102</td>
<td>George Bush</td>
<td>9/25/99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>PROD_ID</th>
<th>COST</th>
<th>SALESPERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>P999</td>
<td>20</td>
<td>Jones</td>
</tr>
<tr>
<td>101</td>
<td>P999</td>
<td>30</td>
<td>Jones</td>
</tr>
<tr>
<td>101</td>
<td>X310</td>
<td>500</td>
<td>Parker</td>
</tr>
<tr>
<td>102</td>
<td>Z225</td>
<td>15</td>
<td>Smith</td>
</tr>
</tbody>
</table>

Result:

- 100 Thomas Jefferson
- 101 Bill Clinton
- 102 George Bush

**INSERT command**

- Allows you to insert single or multiple rows of data into a table
- `INSERT INTO table [(column-list)] [VALUES (value-list) | sql-query]`
INSERT examples

Given mytable(field1 as currency, field2 as text, field3 as integer):

INSERT INTO mytable (field1, field2, field3)
VALUES (12.10, “bah”, 20);

Adds a new row to the table mytable

If you don’t specify every field then fields left out get the default:

INSERT INTO mytable (field1, field2) VALUES(24.2, “zot”);

Adds only for field1 and field2.

INSERT Examples

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>PROD_ID</th>
<th>COST</th>
<th>SALESPERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>PH999</td>
<td>20</td>
<td>Jones</td>
</tr>
<tr>
<td>101</td>
<td>PH999</td>
<td>30</td>
<td>Jones</td>
</tr>
<tr>
<td>102</td>
<td>X310</td>
<td>500</td>
<td>Parker</td>
</tr>
<tr>
<td>102</td>
<td>Z225</td>
<td>15</td>
<td>Smith</td>
</tr>
</tbody>
</table>

INSERT INTO ORDERS (CUST_ID, PROD_ID, COST, SALESPERSON)
VALUES (103, ‘Y338’, 55, ‘Smith’);

INSERT INTO ORDERS (PROD_ID, COST, SALESPERSON)
VALUES (’Y638’, 155, ‘Smith’);

Second might be useful if the CUST_ID is an autonumber field
DELETE

- Delete will remove a row from the table.
- DELETE FROM table_name [WHERE search-condition]

Examples:
DELETE FROM mytable1;
Removes all rows!
DELETE FROM mytable1 WHERE field1 > 100;
Removes only rows with field1>100

UPDATE

- Update lets you modify the contents of the data.

UPDATE table_name
SET field_name = expression [, field-name=expression ...] [WHERE search-condition]

UPDATE mytable SET field1 = 0.0;
Changes all field1’s to zero for every row!
UPDATE mytable SET field1 = 0.0, field2 = “woof”;
Sets field1 to 0 and field2 to woof for all rows!
If this is a violation, access will prevent it from happening
UPDATE mytable SET field1 = 25.0 WHERE field2=“foo”;
Only updates the field where field2 is “foo”
SQL Queries

• There are a lot more queries, but that should give you an idea of what is possible and how it is done

• Next we’ll go over an example that uses SQL on an Access Database from VB.NET
  – Uses OleDB which is different from the book
  – Database access technology changes rapidly

OleDb in VB.NET

• Add to the top:

  Imports System.Data.OleDb

• Set the connection string:
  – This tells VB.NET where the database is and how to connect to it:

  Public Class Form1
  Dim connectionString As String
  Private Sub Form1_Load(...) Handles MyBase.Load
    connectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Path\To\database.accdb"
  End Sub

For Office 2007
Example Reading from the DB

```vbnet
Dim cn As New OleDbConnection(connectionString)
    cn.Open()
Dim cmd As New OleDbCommand("SELECT * From Students WHERE Lastname >= 'M'", cn)
    cmd.ExecuteNonQuery()
Dim reader As OleDbDataReader = cmd.ExecuteReader()
    While (reader.Read())
        Dim ID As Integer = Convert.ToInt32(reader("ID"))
        Dim Name As String = Convert.ToString(reader("LastName"))
        Dim DOB As Date = Convert.ToDateTime(reader("DOB"))
        Console.WriteLine(ID.ToString() + " " + Name + " " + DOB.ToString())
    End While
    cn.Close()
```

Example Writing to the DB

```vbnet
Dim cn As New OleDbConnection(connectionString)
    cn.Open()
Dim newLastName As String = "Washington"
' ID is auto-update field so its left out of the insert
' Put single quotes around String fields, # around dates
Dim sql As String = "INSERT INTO Students (LastName, FirstName, DOB, Address) " + 
    "VALUES (" + newLastName + ", 'George', #04/19/2005#, '999 C St.')"
Dim cmd As New OleDbCommand(sql, cn)
    cmd.ExecuteNonQuery()
    Console.WriteLine("Executed command " + sql)
    cn.Close()
```