Slicing and Dicing Data in CF and SQL: Part 2

Charlie Arehart
Founder/CTO Systemanage
carehart@systemanage.com
SysteManage: our practice makes you perfect℠

Agenda

- Slicing and Dicing Data in Many Ways
- Cross-Referencing Tables (Joins)
- Handling Nulls
- Handling Long Text
- Where to Learn More
- Q&A
Slicing and Dicing Data in Many Ways

As we learned in Part 1, there’s more to database processing than simply selecting columns for display. May want to massage the data:

- Handling distinct column values
  - Show each distinct lastname for employees
  - Create a phone directory with each lastname listed only once
- Manipulating data before or after selecting it
  - Show the first 30 characters of a description column
  - Find rows where the year in a date column is a particular year

Slicing and Dicing Data in Many Ways (cont.)

May also want to:

- Cross-reference tables
  - Show each employee and their department
  - Show all employees and their department, even if not assigned to one
  - Show each employee and their manager
- Handle Nulls
  - Show employees who have not been terminated (TerminationDate column is null)
  - Count how many employees do not live in NYC
- Handle Long Text Fields
  - Retrieve a column that has thousands of characters
Working with Data in SQL Versus ColdFusion

- SQL provides the means to do each of those tasks
  - And ColdFusion has some means to do some of them
- Many developers create complicated CF programs to do what both CF and SQL can enable with simpler constructs
  - Same problems arise in other web app dev environments
- Experienced developers will admonish:
  - Don’t do things in your program that you can better do in SQL
  - The challenge is deciding which to use
- This seminar is about:
  - making maximum use of both CF and SQL for query processing and data manipulation
  - saving time for you and your system
  - creating more effective applications
  - Only 1 topic, though, is CF-specific. Rest is pure SQL

Understanding Relational Database Design

- Relational Databases are comprised of several tables, each storing data about a particular aspect of the subject being described
- Goals are:
  - store only related data in a single table
  - don’t repeat data (don’t store it in more than one place)
  - ensure integrity of data cross-referenced between tables
- Can be challenging to cross-reference that data
Understanding Foreign Keys

Recall previous examples of GROUPing on Dept column
- Assumed that Employees table had DEPT column holding string values for department name

<table>
<thead>
<tr>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

- Problems with this include:
  - We’re storing the same string multiple times on many records
  - If a mistake is made entering a given value, that record will no longer be found in searches on value (see EmpID 4)

More appropriate solution:
- Have Department table with just a list of each valid Dept and a unique DeptID (that table’s primary key)
- Then in Employees table, simply store that DeptID to indicate an employee’s department

<table>
<thead>
<tr>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

- This DeptID in the Employees table is called a Foreign Key
  - Since it holds a value that comes from the primary key of another table
  - This is the fundamental aspect of a “relational” design
Cross-Referencing Tables (Joins)

- **Typical Problems:**
  - Show each employee and their department
  - Show all employees and their department, even if not assigned to one
  - Show each employee and their manager

- **May be tempting for beginners to loop through resultset of one query (departments) and search for related records (employees for each dept):**
  - Bad! Bad! Bad!
  - Correct solution is to instead JOIN the tables together
  - There are several kinds of joins, each serving different purposes

Understanding Joins

- **To retrieve data from multiple tables, simply list both tables in FROM clause, such as:**

  ```sql
  SELECT Name, Dept
  FROM Employees, Departments
  ```

  - Note that if columns of the same name existed in each table, we’d need to prefix the table name to the column

- **Only problem is that this selects all combinations of the values in the two columns:**
  - In our example table, would create 8 rows in result
    - 4 employees times 2 departments

  - Not really what we likely wanted
    - Called a **cartesian product** or a cross join
Inner Joins

- **Problem:** Show each employee and their department
- **Solution:** Perform *Inner Join* of the two tables
  - indicate columns in each table that share common value. SQL automatically matches them
    - Typically, where one table’s foreign key maps to its corresponding primary key in a related table
- **Example:**
  ```sql
  SELECT Name, Dept
  FROM Employees, Departments
  WHERE Employees.DeptID = Departments.DeptID
  ```
- **Correct Result:**
<table>
<thead>
<tr>
<th>Bob</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cindy</td>
<td>Engineering</td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
</tr>
<tr>
<td>Beth</td>
<td>Engineering</td>
</tr>
</tbody>
</table>
- **Note:** the datatype of the columns being joined must match

Join via WHERE vs JOIN clause

- **ANSI SQL standard (and most databases) supports an alternative means of indicating joins**
  - Rather than indicate joined columns in WHERE clause
    - Use them with JOIN keyword on FROM clause
- **Example:**
  ```sql
  SELECT Name, Dept
  FROM Employees
  INNER JOIN Departments
  ON Employees.DeptID = Departments.DeptID
  ```
- **Notes:**
  - If INNER keyword is not specified, INNER may be assumed
    - Not true in MS Access
  - Can join more than two tables with additional join clauses (of either format)
    - Any limit will be set by DBMS
    - Practical limit is that performance suffers with too many joins in a single SELECT
Outer Joins

- With inner join, if value of join columns don’t match, records will not be retrieved
  - Unexpected problems can occur when foreign key is null
- Assume we had at least one employee with no department indicated (null value for DeptID)
  - With inner join, his record will not be displayed at all
    - he has no DeptID to match on DeptIDs in Departments table
  - Could be a real problem if expecting SELECT to show all employees!

<table>
<thead>
<tr>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Outer Joins

- Problem: Show all employees and their department, even if not assigned to one
- Solution: Perform Outer Join of the two tables
- Example:
  ```sql
  SELECT Name, Dept
  FROM Employees LEFT OUTER JOIN Departments
  ON Employees.DeptID = Departments.DeptID
  ```

- Possible Query Result Set Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Sales</td>
</tr>
<tr>
<td>Cindy</td>
<td>Engineering</td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
</tr>
<tr>
<td>Beth</td>
<td>Engineering</td>
</tr>
<tr>
<td>Bill</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- This example indicated LEFT OUTER JOIN: there are 2 other types
  - LEFT join means retrieve all rows from table on left of JOIN even if they don’t have match for join column in right table
  - Creates null values in join columns that did not match
Outer Joins (cont.)

- **WHERE clause syntax for LEFT join:**
  ```sql
  WHERE ON Employees.DeptID = Departments.DeptID
  ```
  - Syntax not supported in MS Access

- **Two other kinds of Outer joins:**
  - **RIGHT OUTER JOIN** retrieves all rows from table on right
    - In current example, that would be useful if we had a row in Departments not pointed to by an employee
    - A RIGHT join would then show a row in the resultset for Accounting (with name being null)
      - Even though no employees had that DeptID
    - **WHERE clause syntax for LEFT join (where supported):**
      ```sql
      WHERE ON Employees.DeptID = Departments.DeptID
      ```

- **Second kind of Outer join**
  - A **FULL OUTER JOIN** (or **FULL JOIN**) retrieves rows from both tables even if join values don’t match
    - In current example, would show both:
      - a row for Bill with no department and
      - A row with no employee name for Accounting
    - Not supported in MS Access
    - No equivalent WHERE clause syntax at all
Self-Joins

- Is possible to join a table to itself
- Assume Employees table has column for ManagerID, to indicate each employee's manager
  - Values for that ManagerID column simply point to the EmpID for their manager
  - How to show who works for who?

<table>
<thead>
<tr>
<th>Employees</th>
<th>EmpID</th>
<th>Name</th>
<th>HireDate</th>
<th>DeptID</th>
<th>ManagerID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Bob</td>
<td>06-04-98</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Cindy</td>
<td>12-01-00</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>John</td>
<td>01-01-01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Beth</td>
<td>05-30-99</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Bill</td>
<td>10-10-97</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Problem: Show each employee and their manager
- Solution: Use self-join (just join table to itself using alias)
  - There is no SELF keyword
  - Example:

```sql
SELECT Employees.Name, Employees.Dept, Mgr.Name
FROM Employees INNER JOIN Employees as Mgr
ON Employees.ManagerID = Mgr.EmpID
```

- Possible Query Result Set Values:

- Note: Why isn’t Bill listed?
  - This was an INNER join. He has null ManagerID
    - We can see from others that he’s the boss and has no boss
    - To show him in table, would need OUTER join
Handling Nulls

- **About Nulls**
  - Columns that have no value are considered NULL
    - Null is not the same as a space or 0 or empty string ("""). It's no value at all
  - A column can be defined to not allow nulls
  - Can select which columns are or aren't null with IS NULL or IS NOT NULL in WHERE clause
  - When a column with a null value is selected and referred to the ColdFusion variable for the column, it will appear as an empty string

- **Typical Problems:**
  - Show employees who have not been terminated
  - Count how many employees do not live in NYC

Handling Nulls: Searching for Nulls

- **Problem:** Show employees who have not been terminated
  - Assume TerminationDate is null if not yet terminated

- **Solution:** Use IS NULL in WHERE clause

- **Example:**
  ```sql
  SELECT *
  FROM Employees
  WHERE TerminationDate IS NULL
  ```
Handling Nulls: Negated Searching And Impact of Nulls

- **Problem:** Count how many employees do not live in NYC
  - Be careful selecting records that don’t have some given value
  - Tempting to use:
    ```sql
    SELECT count(*)
    FROM Employees
    WHERE City <> 'New York'
    ```
  - Problem is it doesn’t find records that don’t have a value for city
    - Consider 200 records: 10 in New York, 5 are null
    - Is answer 185 or 190? Depends on if you think nulls count
      - City <> ‘New York’ ignores records with null values (null is neither equal to nor not equal to “new york”)

- **Solution:** May want to add “OR column IS NULL”

- **Example:**
  ```sql
  SELECT Count(*)
  FROM Employees
  WHERE CITY <> 'New York'
  OR CITY IS NULL
  ```

Handling Long Text

- **See Long Text Retrieval Settings for a given ODBC datasource in CF Administrator**
  - Hidden under “CF Settings” button
  - Can enable retrieval of very long text fields
  - Enabling the option will hamper query performance

- **May want to consider creating multiple datasources for same database**
  - one for when retrieving such columns
  - one for when not doing so

- **Place long text fields last in list of columns being SELECTed**
Some Other Tidbits for You to Investigate

- Nesting multiple joins
- TOP, TOP n PERCENT options on SELECT
- UNIONs
- Nested Subquery
- EXISTS predicate
- Using NULL in INSERT, UPDATE

Where to Learn More

- **Version 5 CF manuals:**
  - Installing and Configuring ColdFusion Server
  - Developing ColdFusion Applications
  - CFML Reference
- **Books by Ben Forta:**
  - Teach Yourself SQL in 10 Minutes
  - Certified ColdFusion Developer Study Guide
  - ColdFusion Web Application Construction Kit
  - Advanced ColdFusion Development
- **Many other CF and SQL books available, including**
  - Practical SQL Handbook (new edition available)
  - SQL For Smarties (any Joe Celko book)