macromedia
COLDFUSION 5
The Fastest Way to Build and Deploy Powerful Web Applications

# *Slicing and Dicing Data in CF and SQL: Part 1*

Charlie Arehart
Founder/CTO Systemanage
carehart@systemanage.com

SysteManage: *our practice makes you perfect* SM



# Agenda

- **Slicing and Dicing Data in Many Ways**
- **Handling Distinct Column Values**
- **Manipulating Data with SQL**
- **Summarizing Data with SQL (Counts, Averages, etc.)**
- **Grouping Data with SQL**
- **Where to Learn More**
- **Q&A**

# Slicing and Dicing Data in Many Ways

- **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

- **As well as:**
  - Summarizing data
    - Show how many employees we have
    - Show how many employees make more than $40k
    - Show how many employees have not been terminated
    - Show the average, max, and min salary for all employees
    - Show the total salary for all employees
    - Show how many distinct salary levels there are

# Slicing and Dicing Data in Many Ways (cont.)

- **As well as:**
  - Grouping Data
    - Show those counts, averages, or totals by department
    - Show those departments whose count/avg/total meets some criteria



# 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

# ColdFusion vs SQL Functions

- **You may know that CF offers hundreds of functions, for string, numeric, date, list and other manipulation**
  - These are used in a format such as Left(), DateFormat()
  - Used within CF expressions, can be used to build SQL
  - Evaluated ***before*** SQL is passed to the DBMS
- **SQL also offers several functions, as we will learn**
  - Also used in ***same format***, such as Left()
  - Indeed, many ***share the same name***!
  - Evaluated by DBMS while processing the SQL
    - Effects how the query results appear or are processed
- **Could indeed use both CF and SQL functions in a given SQL statement**
  - Again, need to take care in deciding which to use
  - In this seminar, focus is on SQL functions



# Handling Distinct Column Values

- **Typical Problems:**
  - Show each distinct lastname for employees
  - Create a phone directory with each lastname listed only once
- **Can try to do it manually, looping through all rows and placing unique values in an array**
  - Tedious, Slow, Unnecessary!
- **Both SQL and ColdFusion have simple solutions to produce list of unique values**
  - Use SQL approach to obtain just unique values
  - Use CF approach to create report breaks on each unique value

# Handling Distinct Column Values: DISTINCT Keyword

- **Problem: Show each distinct lastname for employees**
- **Solution: *DISTINCT* keyword used before column name**
- **Example: (assuming we had a Lastname column)**

```
SELECT Distinct LastName
FROM Employees
ORDER BY Lastname
```

- **Possible Query Result Set Values:**

```
Abbot
Brown
Coleman
```

- **Note: when used with multiple columns, DISTINCT must be specified first. Applies to all columns**
  - Can't do `SELECT Degree, DISTINCT Salary`
  - Can do `SELECT DISTINCT Salary, Degree`
    - Creates distinct instances of the combined values from each



# Handling Distinct Column Values: CFOUTPUT GROUP

- **Could have solved that same problem in CF**
  - Either manually (don't do it!)
  - Or by way of CFOUTPUT's *GROUP* attribute
    - Provide name of column by which data was sorted
    - Will show **only the unique values** of that column

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT LastName FROM Employees
   ORDER BY LastName
</CFQUERY>
<CFOUTPUT QUERY="GetEmployees" GROUP="LastName">
   #LastName#<br>
</CFOUTPUT>
```

- **Would produce equivalent result to that on previous slide**
  - Note that it has **nothing to do with GROUP** in SQL (later)
  - It works. But for this problem, DISTINCT is better
  - Power of CFOUTPUT GROUP, though, is in showing both the distinct values **and all the other rows for each value**

# Handling Distinct Column Values: CFOUTPUT GROUP (cont.)

- **Problem: Create a phone directory with each lastname listed only once**
- **Solution: CFOUTPUT GROUP, with embedded CFOUTPUT to process each row per unique value**
- **Example:**

Once per LastName

Once for each row having that LastName

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT LastName, Minit, FirstName, Phone
   FROM Employees
   ORDER BY LastName
</CFQUERY>
<CFOUTPUT QUERY="GetEmployees" GROUP="LastName">
   <u>#LastName#</u><br>
   <CFOUTPUT>
      #FirstName# #Minit# - #Phone#<br>
   </CFOUTPUT>
</CFOUTPUT>
```

- **Possible Results:**

**Abbot**
John A – x3456
John R – x3476
**Brown**
Alice C – x3421
**Coleman**
Bob H – x3499



# Handling Distinct Column Values: CFOUTPUT GROUP (cont.)

- **Can nest CFOUTPUT Groups**
  - Once for each ORDER BY column listed
- **Example:**

Once for each row having same LastName and FirstName

No QUERY attribute

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT LastName, FirstName, Minit, Phone
   FROM Employees
   ORDER BY LastName, FirstName
</CFQUERY>
<CFOUTPUT QUERY="GetEmployees" GROUP="LastName">
   <u>#LastName#</u><br>
   <CFOUTPUT GROUP="FirstName">
     #FirstName#<br><ul>
     <CFOUTPUT>
       <li>#Minit# - #Phone#
     </CFOUTPUT>
     </ul>
   </CFOUTPUT>
</CFOUTPUT>
```

- **Possible Results:**

**Abbot**
John
- A - x3456
- R - x3476

**Brown**
Alice
- C - x3421

**Coleman**
Bob
- H – x3499

# Manipulating Data with SQL

- **Typical Problems:**
  - Show the first 30 characters of a description column
  - Find rows where the year in a date column is a particular year
- **Tempting to try with CF functions**
  - May be wasteful, or impossible
- **SQL functions may be more efficient, and could even have more features**
  - In any case, remember admonition:
    - Don't do in CF that which you can do in SQL
  - Beware: while some SQL functions are shared by all DBMS's, each supports its own or variations



# Manipulating Data with SQL: Text Functions

- **Problem: Show the first 30 characters of a description column**
  - Can certainly use CF's Left() function to substring the result passed back from SQL
    - But this means sending all data from DB to CF, only to then be stripped down to 30 chars. Wasteful!
- **Solution: Use SQL** Left() **function**
- **Example:**

```
SELECT Left(Description,30) FROM Products
```

- **Note: There are many other similar text manipulation functions, depending on DBMS**
  - Length(), Lower(), Upper(), Ltrim(), Soundex(), etc.
  - Investigate DBMS documentation to learn more

# Manipulating Data with SQL: Date Functions

- **Problem: Find rows where the year in a date column is a particular year**
  - Assuming date column contains month, day, and year, how to just search on year?
  - Could find records between 01/01/xx and 12/31/xx
- **Solution: Use SQL** DatePart() **function**
- **Example:**

```
SELECT * FROM Employees
WHERE DatePart("yyyy",HireDate) = 2001
```

- **Note: each DBMS will have its own date handling functions and function arguments**
  - This example is from Access. Could also use Year(HireDate)
- **There are many other similar date manipulation functions, depending on DBMS**
  - Also will find numeric functions, system functions, and more



# Summarizing Data with SQL

- **Typical Problems:**
  - Show how many employees we have
  - Show how many employees make more than $40k
  - Count how many employees have not been terminated
  - Show the average, max, and min salary for all employees
  - Show the total salary for all employees
  - Show how many distinct salary levels there are
- **Again, tempting to try with CF processing**
  - May be complicated, wasteful
  - SQL functions may be more efficient, more powerful
  - SQL functions for summarixing data are known as "aggregate functions": Count, Min, Max, Avg, Sum
    - Others include StdDev (standard deviation), Var (variance)

# Summarizing Data with SQL: Count(*) Function

- **Problem: Show how many employees we have**
  - Yes, we can find all records and look at recordcount
    - But if all we want it the count, this is **wasteful**!!!
- **Solution: Use SQL** Count(*) **function**
- **Example:**

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT Count(*) as RecCount
   FROM Employees
</CFQUERY>
<CFOUTPUT>
   Total Employees: #GetEmployees.RecCount#<br>
</CFOUTPUT>
```

- **Possible Query Result Set Values:**

```
Total Employees: 54
```

- **Notes:**
  - We must use a column alias in order to refer to that count within ColdFusion
  - Returns only a single-record resultset (and does it FAST!)
  - Not to be confused with SELECT * (which is SLOW!)



# Summarizing Data with SQL: Count(*) Function and Filter

- **Problem: Show how many employees make more than $40k**
- **Solution: Use SQL** Count(*) **function and a filter**
  - Simple matter of adding a WHERE clause to indicate the desired criteria
- **Example:**

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT Count(*) as RecCount
   FROM Employees
   WHERE Salary > 40000
</CFQUERY>
<CFOUTPUT>
 Num. employees making +40k: #GetEmployees.RecCount#<br>
</CFOUTPUT>
```

# Summarizing Data with SQL: Count(*col*) Function

- **Problem: Count how many employees have been terminated**
- **Solution: Use SQL** Count(*column*) **function**
  - Instead of counting all records, count all having a value for a given column
  - Assume terminated employees have a value in the TerminationDate column
- **Example:**

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT Count(TerminationDate) as RecCount
   FROM Employees
 </CFQUERY>
<CFOUTPUT>
 Num. Employees terminated: #GetEmployees.RecCount#<br>
</CFOUTPUT>
```

- **Note: doesn't count records having null column value**
  - Will discuss nulls later
  - In this case, the behavior is as expected. May not always be



# Summarizing Data with SQL: AVG/MAX/MIN Functions

- **Problem: Show the average, max, and min salary for all employees**
- **Solution: Use SQL** Avg()**,** Min()**, or** Max() **functions**
  - Besides just counting records having any value for a given column, can also use these functions to summarize
- **Example:**

```
<CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees">
   SELECT Avg(Salary) as AvgSal, Min(Salary) as MinSal,
          Max(Salary) as MaxSal
   FROM Employees
 </CFQUERY>
<CFOUTPUT>
 Avg Sal: #GetEmployees.AvgSal#<br> ...
</CFOUTPUT>
```

- **Notes:**
  - Like Count(*column*) function, these functions ignores columns with null values
    - I.e., is average of records having a value for that column
  - Also, can add a filter in order to compute summaries for records meeting some other criteria

# Summarizing Data with SQL: SUM Function

- **Problem: Show the total salary for all employees**
- **Solution: Use SQL** Sum() **function**
  - Just as other functions compute Avg/Min/Max, can use Sum function to add up all values of column
- **Example:**

```
SELECT Sum(Salary) as SumSal
FROM Employees
```

- **Notes:**
  - Can also perform mathematical computation on the column and sum that:

```
SELECT SUM(Salary * 1.20)
```

  - Or perform computation between two or more columns and sum that, as in:

```
SELECT SUM(Salary*RaisePct)
```



# Summarizing Data with SQL: Using DISTINCT with Functions

- **Problem: Show how many distinct salary levels there are**
- **Solution: Use DISTINCT keyword with functions**
  - Rather than perform given function against all values of the given column in all records, can performs it against only the unique values that exist
- **Example:**

```
SELECT Count(DISTINCT Salary) as NumDistinctSals
FROM Employees
```

- **Notes:**
  - Note that this will produce just one number: the number of distinct salary values that exist
    - To produce instead a count of employees at each salary level, need to learn about SQL *GROUP BY* clause (coming next)
  - Can also use AVG (average of distinct values rather than of all values). MIN and MAX would return same result either way

# Summarizing Data with SQL: Using DISTINCT with Functions

- **Notes:**
  - Note also, there's an opposing ALL keyword that can be used, instead of DISTINCT; performs aggregation against all values
    - This is the default and doesn't need to be specified
  - MS Access does not support this use of DISTINCT (or ALL) within aggregate functions



# Grouping Data with SQL

- **Typical Problems:**
  - Show those counts, averages, or totals by department
  - Show which departments have count/avg/total meets some criteria
- **SQL provides a GROUP BY clause that can be used to create a list of unique values for a column**
  - Difference from DISTINCT is that it also "rolls up" the rows
    - aggregates some computation over all the records having that unique value

# Grouping Data with SQL

- **Assume the employees table has a Dept column**
- **Example:**

```
SELECT Dept FROM Employees
GROUP BY Dept
```

- **Note: this simple example creates a result no different than `SELECT DISTINCT Dept`**
  - You would not typically use this statement, because you're also asking the DB to "roll up" rows having the same value of Dept, but are aggregating nothing
  - Difference comes when combined with the previously presented aggregate functions, which then aggregate the data **BY** the unique "grouped" column values



# Grouping Data with SQL: Using GROUP BY with Count Function

- **Problem: Show count of employees by department**
- **Solution: Use GROUP BY with COUNT(*) function**
- **Example:**

```
SELECT Dept, Count(*) as CountEmp
FROM Employees
GROUP BY Dept
```

- **Possible Query Result Set Values:**

| | |
|---|---|
| | **4** |
| **Sales** | **15** |
| **Engineering** | **33** |
| **Marketing** | **7** |

- **Notes:**
  - In example, first row in resultset represents records with null value for Dept column
  - Order of rows is random. Could add `ORDER BY Dept`
    - If present, must be specified AFTER the GROUP BY

# Grouping Data with SQL: Using GROUP BY with Avg Function

- **Problem: Show average salary by department**
- **Solution: Use GROUP BY with Avg(*column*) function**
  - Aggregate on a column other than that being grouped
- **Example:**

```
SELECT Dept, Avg(Salary) as AvgSalary
FROM Employees
GROUP BY Dept
```

- **Possible Query Result Set Values:**

| | 45687 |
|---|---|
| **Sales** | **83276** |
| **Engineering** | **75500** |
| **Marketing** | **55000** |

- **Notes:**
  - Could use Min/Max/Count(*column*) too



# Grouping Data with SQL: Using GROUP BY with Functions

- **More notes:**
  - Columns to be SELECTed can only be aggregate functions and/or column named in GROUP BY
    - Could not

```
SELECT Lastname, Count(*) FROM Employees
GROUP BY Dept
```

      - Since LastName isn't being GROUPed and isn't an aggregate function itself
      - Often a source of confusion, though it clearly wouldn't make sense to show LastName here

# Grouping Data with SQL: Using GROUP BY with Filter

- **Problem: Show average salary by departments of employees who've completed grade 12**
- **Solution: Use GROUP BY with filter**
  - WHERE clause limits which records are to be GROUPed
- **Example:**

```
SELECT Dept, Avg(Salary) as AvgSalary
FROM Employees
WHERE GradeCompleted >= 12
GROUP BY Dept
```

- **More notes:**
  - WHERE must occur after FROM, before GROUP
    - Order of appearance:
      - FROM, WHERE, GROUP BY, ORDER BY
  - To select records whose aggregated values meet some criteria, use HAVING clause



# Grouping Data with SQL: Using GROUP BY with HAVING

- **Problem: Show departments whose employees have an average salary greater than $40,000**
- **Solution: Use GROUP BY with** HAVING
- **Example:**

```
SELECT Dept, Avg(Salary) as AvgSalary
FROM Employees
GROUP BY Dept
HAVING Avg(Salary) > 40000
```

- **Note:**
  - HAVING must occur after GROUP BY, before ORDER BY
  - Order of appearance:
    - FROM, WHERE, GROUP BY, HAVING, ORDER BY
  - Expression in HAVING can't refer to alias from SELECT clause
    - In example above, couldn't use `HAVING AvgSalary > 40000`

# Subjects of Part 2

- **Slicing and Dicing Data in CF and SQL: Part 2**
  - Cross-Referencing Tables (Inner and Outer Joins)
  - Handling Long Text
  - Handling Nulls



# 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)



# Contact Information

**Contact for follow-up issues**

- **Email:** carehart@systemanage.com
- **Phone:** (301) 604-8399
- **Web:** www.systemanage.com

**Also available for**

- Training (custom or pre-written)
  - CF, DB, Jrun/J2EE, Javascript, wireless, and more
- Consulting (very short-term engagements)
  - best practices, architecture, setup, troubleshooting, etc.
- Developer Group Mentoring, and more

# Q&A

?