Advanced SQL For ColdFusion

Sean Woods
sewoods@adobe.com

Lansdowne Resort, Leesburg VA August 12- 15, 2009

www.cfunited.com

Who Am I?

- Some dude that works at Adobe (for like 10 years!).
- Been working with Oracle for about 12 years, CF for about 6.
- Totally sucks at PowerPoint.
- Other than that, semi-normal geek, married, 2 kids, 2 dogs, 2 cats, blah blah blah.

Why Are You Here?

- To get some ideas for coding your apps so they get the best performance out of your DB.

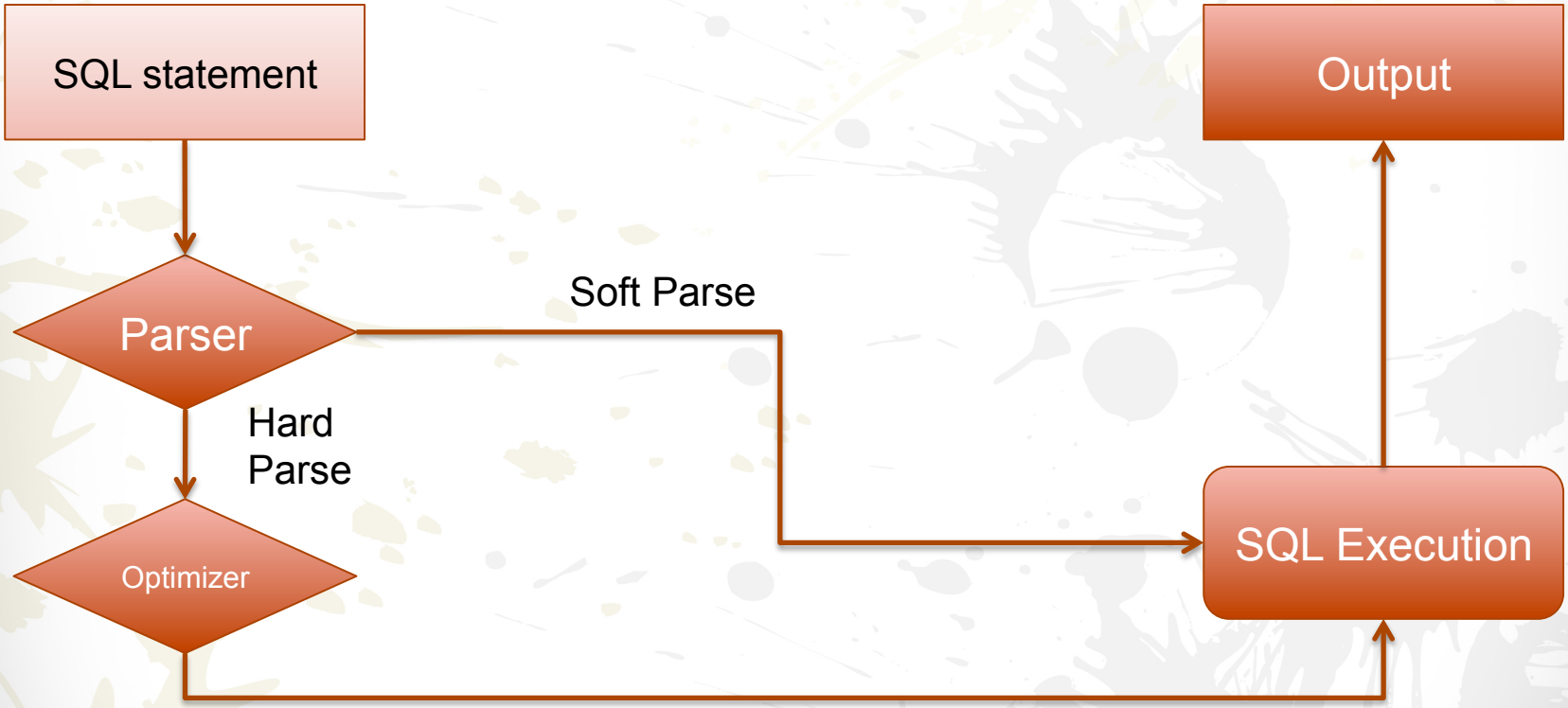
Become One With Your Database

- Understand what your database is capable of.
- Really really know SQL.

Statement Processing

- Parsing
  - Syntax / Semantic validation.
  - Soft parse or hard?
- Optimization
  - Generation of an optimal plan for execution of the submitted SQL.
- Statement Execution
  - Actual execution of the query and initial fetch whereby the client receives data.

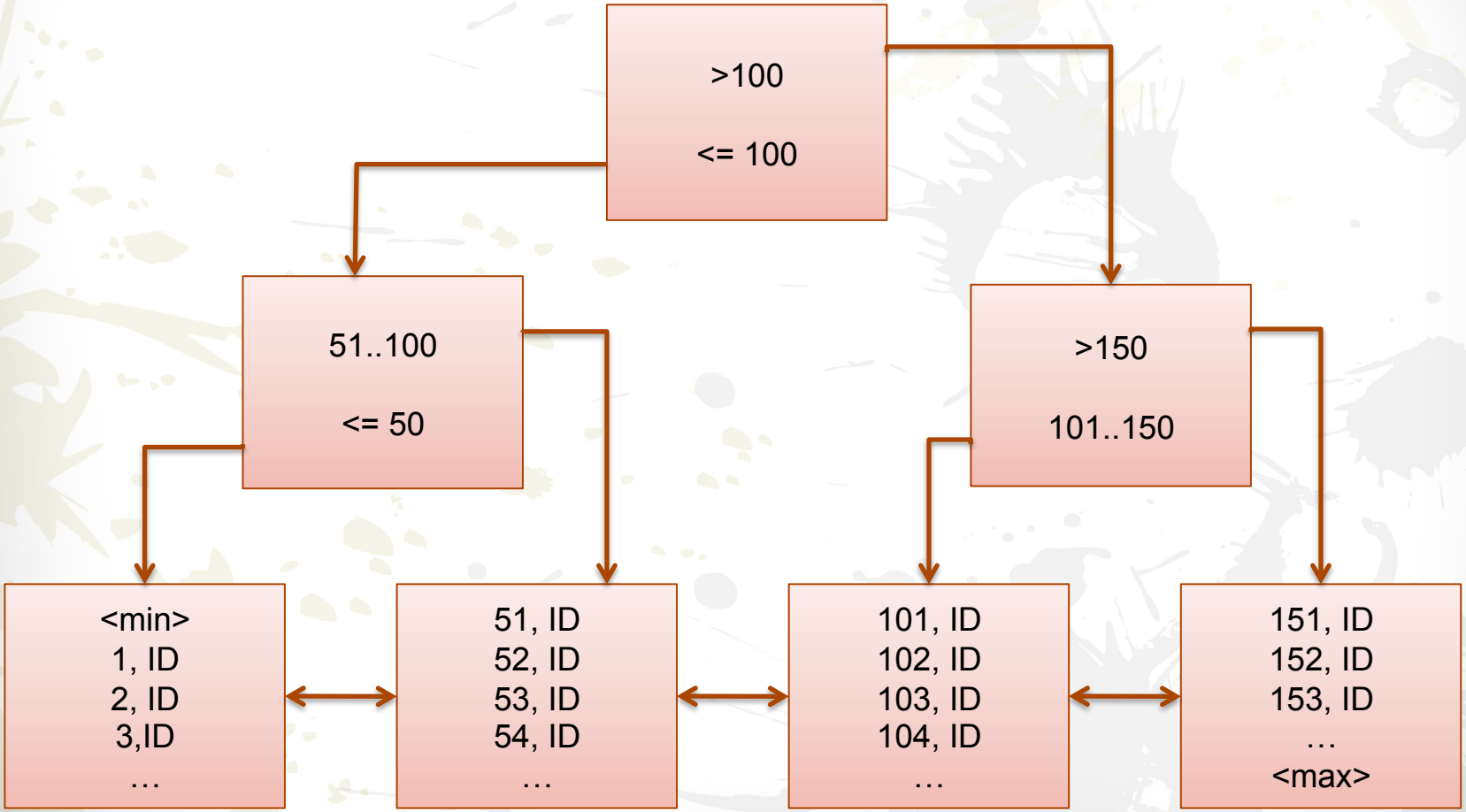
Statement–processing steps



Access Paths

- Full Scan
  - Entire table is read sequentially.
  - Very efficient for reading a lot of data.
- Index Scan
  - Generally either a “unique scan” or “range scan”.

Conventional Index Structure



Make Sure Your Indexes Are Useable

- The classic example: a search condition on an indexed date field in Oracle.

```
<cfquery name="getOrders" datasource="foobar">
        select a.order_id
         from orders a
         where order_date = to_date('2009-08-01','yyyy-mm-dd')
</cfquery>
```

- But, Oracle uses the "date" datatype to store date **and** time, resolved to the second. To test the uninitiated, the default date format omits the time from display. So, the developer attempts to suppress the time from the order_date field as such:

```
<cfquery name="getOrders" datasource="foobar">
        select a.order_id
         from orders a
         where trunc(order_date) = to_date('2009-08-01','yyyy-mm-dd')
</cfquery>
```

- And the ability for the optimizer to use the index on the order_date field is torpedoed by the trunc() function.

Make Sure Your Indexes Are Useable

- The solution in this case is to refer to the order_date column in full using a range condition.

```
<cfquery name="getOrders" datasource="foobar">
        select a.order_id
        from orders a
        where order_date >= to_date('2009-08-01','yyyy-mm-dd') and
                order_date < to_date('2009-08-02','yyyy-mm-dd')
</cfquery>
```

- The range condition leaves the index on the order_date column useable by the optimizer.

<CFQUERYPARAM>

- Make your queries
  - Faster
  - Safer

<CFQUERYPARAM>

Executes in approx. 60 sec

```
<cfloop from="1" to="5000" index="i">
    <cfquery name="q"
        datasource="#dsn#">
    select * from test
    where f1 = #i#
    </cfquery>
</cfloop>
```

Executes in approx. 10 sec

```
<cfloop from="1" to="5000" index="i">
    <cfquery name="q"
        datasource="#dsn#">
    select * from test
    where f1 =
        <cfqueryparam value="#i#" />
    </cfquery>
</cfloop>
```

But why is it faster?

Parse Once, Execute Many

- Most DBMS's employ some concept of SQL statement caching. For example, in Oracle, SQL is stored in a shared memory area called the Library Cache.
- Each time SQL is executed, the optimizer either re-uses an existing (cached) statement from the Library Cache (a.k.a. a "soft parse") or loads it into the Library Cache where it is initially optimized (a.k.a. a "hard parse").
- You want to Soft Parse you queries whenever possible. Using CFQUERYPARAM helps your DB use cached SQL statements. This is vital to building applications that scale.

In the previous example, the CFML which did not use <CFQUERYPARAM> ran about 6 times slower than the CFML that did. Nearly all of this extra time was used by the DBMS to perform repeated parsing.

SQL Injection – a scenario…

- A CFQUERY block that does not use a parameterized query.

```
<cfquery name="getEmps" datasource="foobar">
        select * from employees
         where emp_id = #url.emp_id#
</cfquery>
```

- The value of url.emp_id *should* be a number. But what happens if it isn't?

```
getEmps.cfm?emp_id=312%3delete%20from%20employees%20
```

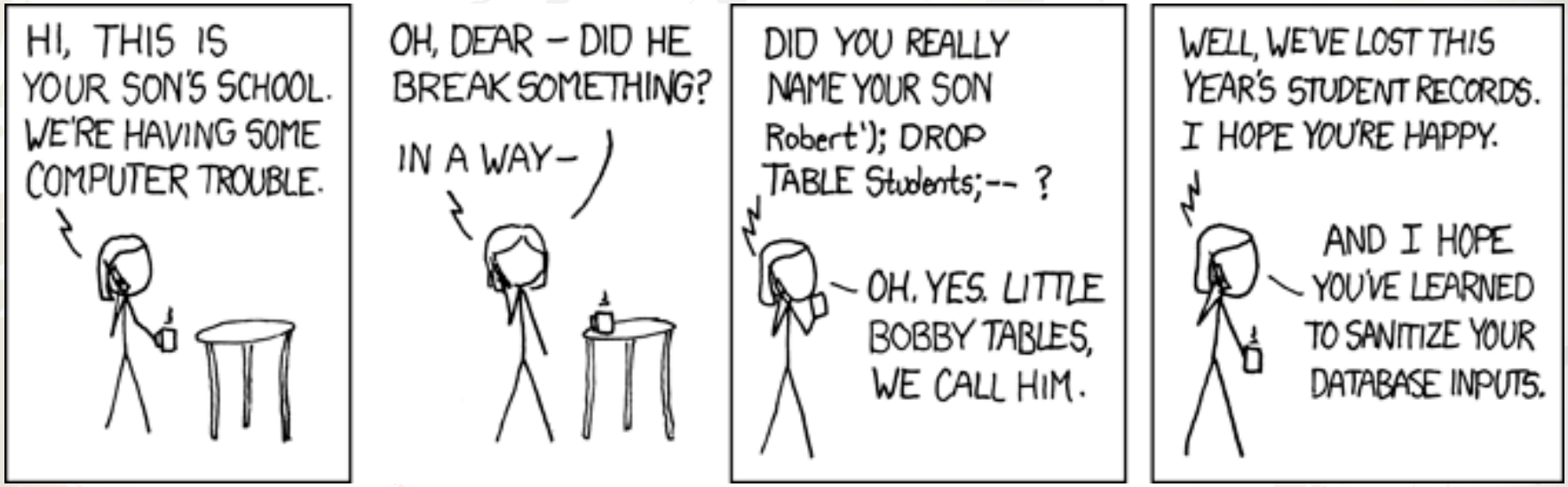
- The SQL that would be passed to the database would look something like

```
Select * from employees
Where emp_id=312;delete from employees;
```

- Since most databases will allow multiple statements inside a single CFQUERY block, the database would execute both of the above statements. First selecting all the employees from the table and then deleting them. Nice!!!

```
<cfquery name="getEmps" datasource="foobar">
        select * from employees
         where emp_id =<cfqueryparam value="#url.emp_id#" cfsqltype="cf_sql_integer" />
</cfquery>
```

<CFQUERYPARAM>



Some Guidelines for <CFQUERYPARAM>

- You need to use it whenever you are passing a dynamic value to your SQL where/set/values clause.
- Two things to be aware of
  - cfsqltype attribute tells the database what type of value is being passed. There is a long list of available types to support the myriad database vendors on the market.
  - You have to use it in the where/set/values clause of the query. That is, passing a dynamic value to your select clause using <CFQUERYPARAM> is not an option.
- Are there exceptions? Of course.

<CFQUERYPARAM> - Exceptions

- Sometimes, literal values in queries provide the optimizer with crucial information needed to generate an optimal plan at runtime.
  - Data that is highly skewed.
- Seconds-per-Query Systems (data warehouses).
  - Here, the queries are few but big.
  - Overhead of parsing is a tiny fraction of overall execution time.
  - Here the goal is to get the best execution plan possible (not to execute as many queries as possible).

<CFTRANSACTION>

- Takes the database from one consistent state to the next.
  - This is one of the hallmarks of an RDBMS and what sets it apart from a file system.
- Instructs the DBMS to treat multiple database operations as a single unit of work.
- Provides database commit and rollback processing (where supported).

```
<CFTRANSACTION action="begin>
        <CFOUTPUT query="qry_with_10000_rows">
                <CFQUERY name="DoInsert" datasource="#variables.dsn#">
                   insert into employees (emp_id, fname)
                   values (
                          <CFQUERYPARAM value="#emp_id#" cfsqltype="cf_sql_integer" />,
                           <CFQUERYPARAM value="#fname#" cfsqltype="cf_sql_varchar" />
                             )
                </CFQUERY>
        </CFOUTPUT>
</CFTRANSACTION>
```

<CFTRANSACTION>

```
<cftimer type="inline">
	<cfloop from="1" to="20000" index="i">
		<cfquery name="load" result="loadRes" datasource="localDerby">
			insert into emp
			(emp_id, fname, lname)
			values
				(
				<cfqueryparam value="#i#" />,
				<cfqueryparam value="Foo#i#" />,
				<cfqueryparam value="Bar#i#" />
				)
		</cfquery>
	</cfloop>
</cftimer>
```

cftimer: 24560ms (about 25 seconds)

- Is this a good response time?
- Are there any potential problems with the program?

<CFTRANSACTION>

- We have a scalability problem.
  - Each row is committed to the database as it is inserted. This is causing lots of unnecessary overhead.
- We have a functional problem.
  - What happens if the DB crashes in the middle of the program? How do we know which rows were inserted?

<CFTRANSACTION>

```
<cftimer type="inline”>
        <ctransaction action=“begin”>
                <cfloop from="1" to="20000" index="i">
                        <cfquery name="load" result="loadRes" datasource="localDerby">
                                insert into emp
                                (emp_id, fname, lname)
                                values
                                        (
                                        <cfqueryparam value="#i#" />,
                                        <cfqueryparam value="Foo#i#" />,
                                        <cfqueryparam value="Bar#i#" />
                                        )
                        </cfquery>
                </cfloop>
          </cftransaction>
</cftimer>
```

cftimer: 8649ms (about 9 seconds)

Query of Query

- Query of Query (QofQ)
  - Allows you to query information that has already been retrieved into memory, using standard SQL constructs.
  - Heterogeneous data retrieval with
    - Integrate data from disparate databases.
    - Use any tag that returns a query object (CFPOP, CFHTTP, CFDIRECTORY, etc…) – awesome!
  - Limitations of
    - Limited to a subset of SQL. Essentially, what is supported universally.
    - ANSI-92 join syntax not supported.

```
<cfquery name="mainQuery" datasource=“prod">
        select a.emp_id,
        a.fname,
        a.lname,
        a.salary
        from employees a
</cfquery>

<cfquery name="legacyQuery" datasource="legacy">
        select b.emp_id,
        b.fname,
        b.lname,
        b.salary
        from employees_legacy b
</cfquery>

<cfquery name="merge" dbtype="query”>
        select emp_id,
        fname,
        lname,
        salary
        from mainQuery
                union
        select emp_id,
        fname,
        lname,
        salary
        from  legacyQuery
</cfquery>
```

CFQuery Blockfactor

```
<CFQUERY name="q" datasource="#variables.dsn#" blockfactor="100">
        select a.emp_id,
               a.emp_name
        from emp
</CFQUERY>
```

- Most drivers can/will fetch and buffer rows in batches.
- You can, with some drivers, control the batch size by using the "blockfactor" attribute in CFQuery. Default is 1, max value is 100 *
- Properly tuning the batch size can have a profound impact on performance, especially when network i/o dominates response time.
- At a high level, the optimal batch size is a function of
  - The size of the rows you are returning.
  - The size of your network packets.

** The documented max value is 100. However, some drivers (such as Oracle's JDBC driver) will happily let you specify values higher than 100.*

```
<cftimer type="debug">
        <cfquery name="getemps" result="resEmps" datasource="skunkworks" blockfactor="1">
                select *
                from employees
                Where department_id = 100
        </cfquery>
</cftimer>

<cfdump var="#resEmps#" label="Query with Blockfactor = 1">
```

Query with Blockfactor = 1 - struct	
CACHED	false
COLUMNLIST	COMMISSION_PCT,DEPARTMENT_ID,EMAIL,EMPLOYEE_ID,FIRST_NAME,HIRE_DATE,JOB_ID,LAST_NAME,MANAGER_ID,PHONE_NUMBER,SALARY
EXECUTIONTIME	308668
RECORDCOUNT	4096
SQL	select * from employees where employee_id = 100

Debugging Information

ColdFusion Server Enterprise	8,0,1,195765
Template	/dev/dev/emp.cfm
Time Stamp	07-Jul-09 02:03 PM
Locale	English (US)
User Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.4; en-US; rv:1.9.0.11) Gecko/2009060214 Firefox/3.0.11
Remote IP	0:0:0:0:0:0:0:1%0
Host Name	localhost

Execution Time

Total Time	Avg Time	Count	Template
308734 ms	308734 ms	1	/Applications/ColdFusion8/wwwroot/dev/dev/emp.cfm
1 ms	1 ms	1	/Applications/Application.cfm
45 ms			*STARTUP, PARSING, COMPILING, LOADING, & SHUTDOWN*
308780 ms			*TOTAL EXECUTION TIME*

red = over 250 ms average execution time

```
<cftimer type="debug">
        <cfquery name="getemps" result="resEmps" datasource="skunkworks" blockfactor="1000">
                select *
                from employees
                where department = 100
        </cfquery>
</cftimer>

<cfdump var="#resEmps#" label="Query with Blockfactor = 1000">
```

Query with Blockfactor = 1000 - struct	
CACHED	false
COLUMNLIST	COMMISSION_PCT,DEPARTMENT_ID,EMAIL,EMPLOYEE_ID,FIRST_NAME,HIRE_DATE,JOB_ID,LAST_NAME,MANAGER_ID,PHONE_NUMBER,SALARY
EXECUTIONTIME	2426
RECORDCOUNT	4096
SQL	select * from employees where employee_id = 100

Debugging Information

ColdFusion Server Enterprise	8,0,1,195765
Template	/dev/dev/emp.cfm
Time Stamp	07-Jul-09 02:11 PM
Locale	English (US)
User Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.4; en-US; rv:1.9.0.11) Gecko/2009060214 Firefox/3.0.11
Remote IP	0:0:0:0:0:0:0:1%0
Host Name	localhost

Execution Time

Total Time	Avg Time	Count	Template
2463 ms	2463 ms	1	/Applications/ColdFusion8/wwwroot/dev/dev/emp.cfm
1 ms	1 ms	1	/Applications/Application.cfm
42 ms			*STARTUP, PARSING, COMPILING, LOADING, & SHUTDOWN*
2506 ms			*TOTAL EXECUTION TIME*

red = over 250 ms average execution time

Do Only What is Required

- An application whose purpose is to either update an existing row or insert a new one (an *"Upsert"*).
  - Typically done using an existence test to check if a row already exists.
  - Most DBMS's have the ability to do this implicitly.

```
MERGE INTO employees a
USING (
  SELECT emp_id
  FROM employees_archive
  WHERE name = 'john doe') b
ON (a.emp_id = =b.emp_id)
WHEN MATCHED THEN
  UPDATE SET a.dept_id = 1234
WHEN NOT MATCHED THEN
  INSERT (a.name, b.dept_id)
  VALUES ('john doe', 3456);
```

  - Also Available in DB2 (MERGE), MySQL (REPLACE).

Don’t use “Select *”

- But my table is small / doesn’t have many fields.
  - It will get big, eventually.
  - You will add new fields, even if you don’t think you will.
- But it doesn’t affect performance.
  - Actually, it does.

Table Aliases

- Good programming practice
  - All tables get an alias.
  - All columns in SELECT statement are qualified with the alias, even those that are not ambiguous.

```
select a.emp_id,
       a.fname,
       a.lname,
       a.dept_id,
       b.name as dept_name
from employees     a
                join departments     b
                                on (a.dept_id = b.dept_id)
where b.dept_id = 20
```

- Some DBMS's can parse a query faster when the field list is fully qualified.
- Code is cleaner and easier to read.

Selecting Unique Records

- Two ways to do this:
  - Select Distinct – easiest to implement and most common in use.
  - Use a subquery – a little more work to implement but often a preferable solution as it will usually avoid a sort.

```
select distinct a.emp_id,
                a.fname,
                a.lname,
                a.salary
from employees a
        join employee_expenses b
                on (a.emp_id = b.emp_id)
```

```
Select a.emp_id,
                a.fname,
                a.lname,
                a.salary
from employees a
Where a.emp_id in
                (select b.emp_id from employee_expenses b)
```

Embedding Procedural Logic in Your SQL

- SQL is declarative – you ask and you get.
- SQL-92 standard allows for a procedural extension to SQL.
  - Most vendors have implemented this as CASE.
  - Allows for conditional logic to be embedded in your sql – very powerful stuff.

```
SELECT
CASE WHEN dept_id = 10 then 'FINANCE'
          WHEN dept_id = 20 then 'ACCOUNTING'
           WHEN dept_id = 30 then 'SALES'
          ELSE 'UKNOWN DEPARTMENT'
END as department
From employees;
```

Triggers

- A *trigger* instructs the database to execute a particular set of instructions whenever an event (usually an insert, update, delete on a table) occurs.
- Available in almost every Enterprise DBMS but implementation varies from platform to platform.
- Very powerful tool but be aware of the tradeoffs when using them.

```
create trigger trig_upd
        after update on departments
        for each row
insert into departments_audit
          (dept_id,
           name,
           location_id,
           changedwhn,
           changedby)
values
          (new.dept_id,
           new.name,
           new.location_id,
           current_timestamp,
           current_user);
```

Stored Procedures

- Stored subprogram that persists in the database.
- Many uses but at a high level, typically used to encapsulate complex database operations into a single unit of work.
- Vastly different implementations across major DBMSs.

A reaaaaallllly basic mySql Proc…

```
CREATE PROCEDURE employeeCount()
BEGIN
        SELECT count(emp_id) as emp_count
        FROM employees;
END;
.
.
.
CALL employeeCount();
```

Apache Derby Embedded Database

- Full featured RDBMS embedded right into ColdFusion
  - Supports a wide range of features such as
    - Transaction processing
    - Statement caching
    - Robust query optimizer
    - And so much more!!!

- For a full treatment on this topic, check out Charlie Arehart's excellent session.
  - Using Apache Derby, the Open Source Database Embedded in ColdFusion 8
  - Friday 8/14 @ 3PM in Sandtrap

Tools – a few IDEs to consider for DB development

- TOAD (Tool for Oracle Application Development)
  - The good:
    - Super robust set of utilities for development and administration.
    - Easy to master.
    - Contrary to what the name implies, there are versions for other DBMS's.
      - SQL Server, MySQL, DB2, Sybase
  - The bad:
    - Windows only (yeah, I know, that one stings).
    - Somewhat pricey.

IDEs cont…

- SQL Developer (from Oracle)
  - The good:
    - Powerful IDE
    - Can connect to any JDBC compliant DB
    - Runs on WIN and MAC
    - FREE!
  - The bad:
    - Seems to run a bit slower than TOAD
    - Not as feature-rich as TOAD

IDEs cont…

- SQL Explorer
  - Standalone client or plugin for Eclipse 3.2
  - The good:
    - Excellent choice if you're doing light development.
    - Can query and browse any JDBC compliant database.
    - Your ColdFusion development and your DB development are all done in the same IDE.
    - Free.
  - The bad:
    - Lacks the robust features of previous two

Some additional resources

- A few blogs (with good database-specific content)
  - Sean Corfield (www.corfield.org)
  - Ray Camden (www.coldfusionjedi.com)
  - Adrian Moreno (www.iknowkungfoo.com)
    - And of course…
  - Ben Forta (www.forta.com)
- Books (platform agnostic, for the most part)
  - *Apprentice SQL Guide:*
    - Beighley, Lynn, *Head First SQL*
  - *Journeyman SQL Guide:*
    - Celko, Joe *SQL For Smarties*
  - *Zen Master SQL Guide:*
    - Faroult, Stephane, *The Art of SQL*

A couple of final tips

- Analyzing SQL is much like trying to figure out a word problem in math. Identify words in the question that match things you know such as tables and columns and then break things apart.
- Know your schema – if you don't know your data you won't be able to write effective queries.
- Make good friends with your DBA.

Questions/Comments

- Sean Woods
  - sewoods@adobe.com
  - 415.832.4225 (work)
- Thank you!