What Are You Expecting? Oracle 10g's Expression Filter Uncovered

An Introductory Overview

Melanie Caffrey
Harris Corporation
NYOUG - December, 2005

Lecture Objectives

- What Exactly Is Expression Filter?
- Attribute Sets and Expression Sets
- Storing Expressions and the Evaluate Operator
- Expression Indexing *
- Validation Utility and the Exceptions Table
- Uses and Benefits
- Expression Filter Goodies and Gotchas

* You must be using the Enterprise Edition of the Oracle Database to take advantage of expression indexing capabilities.

What Exactly is Expression Filter?

- A facility that allows you to write and store WHERE clauses, essentially.
- You store these WHERE clauses in a text column of a table.
- The database can watch the table and alert you when rows matching any of the WHERE clause expressions become available.
- Expression processing functionality (procedures and functions) are owned by the EXFSYS schema.

How Does It Work?

- To describe data you're watching for, you create the WHERE clause expressions ahead of time.
- Oracle Expression Filter processing can initiate whenever a new row is stored in the database and continues on to find WHERE clauses that match the new data in that row.

Attribute Sets

- First, decide which elements or types of values are important to you.
- Each of these elements is an *attribute*, and the entire collection of attributes used by an application using Expression Filter is an *attribute set*.
- You'll need to
  1. Name each attribute
  2. Specify a datatype for each attribute
  3. Name the entire attribute set

Creating an Attribute Set

First decide which tables you'll use towards expression filter capability:

```
SQL> CREATE TABLE student (
  2  student_id NUMBER,
  3  name VARCHAR2(10),
  4  class_year VARCHAR2(10),
  5  registration_date DATE,
  6  prereqs_needed VARCHAR2(1),
  7  estimated_graduation_date DATE);
```

```
SQL> CREATE TABLE enrollment (
  2   student_id NUMBER,
  3   enroll_date DATE,
  4   section_id NUMBER,
  5   max_enrollees VARCHAR2(1));
```

```
SQL> CREATE TABLE section (
  2  section_id NUMBER,
  3  course_no VARCHAR2(6),
  4  section_no NUMBER,
  5  location VARCHAR2(40));
```

```
SQL> CREATE TABLE course (
  2  course_no VARCHAR2(6),
  3  course_description VARCHAR2(40),
  4  required_prereq VARCHAR2(6));
```

Creating an Attribute Set (cont.)

Next create an object type for your attribute set:

```
SQL> CREATE OR REPLACE TYPE reg_priorities AS OBJECT (
2  student_id NUMBER,
3  class_year VARCHAR2(10),
4  prereqs_needed VARCHAR2(1),
5  estimated_graduation_date DATE,
6  enroll_date DATE,
7  section_id NUMBER,
8  course_no VARCHAR2(6),
9  required_prereq VARCHAR2(6),
10 max_enrollees VARCHAR2(1));
```

Then create an attribute set, of the same name, using the DBMS_EXPFIL package:

```
SQL> BEGIN
  2     DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set  => 'reg_priorities',
  3                                      from_type =>'yes');
  3  END;
  4  /
```

The Expression Repository

You can either create a table for your expressions or add a text column (to contain expressions) to one of your existing tables.

```
SQL> CREATE TABLE reg_interests (
  2  interest_id NUMBER,
  3  interest    VARCHAR2(200));
```

In this example, "interest" is the column that will store expressions. Therefore, it must be linked to an attribute set.

```
SQL> BEGIN
  2     DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET(
  3        attr_set => 'reg_priorities',
  4        expr_tab => 'reg_interests',
  5        expr_col => 'interest');
  6  END;
  7 /
```

EXFSYS Validation Trigger

Assigning an attribute set to the column you'll use to store expressions creates a trigger in the EXFSYS schema on the REG_INTERESTS table. This trigger ensures that any values inserted into the "interest" column are valid expressions involving the attributes in the REG_PRIORITIES attribute set.

```
TRIGGER EXF$VALIDATE_1
   BEFORE INSERT OR UPDATE OF "INTEREST" ON "SCOTT"."REG_INTERESTS"
     FOR EACH ROW
  declare
    caller VARCHAR2(32);
    isvalid  NUMBER := 1;
    code   VARCHAR2(1) := 'C';
  begin
    select user into caller from dual;
    if (caller != 'SCOTT') then
      if (UPDATING) then code:='U'; end if;
      if (code = 'U' or :new."INTEREST" is not null) then
exfsys.exf$check_privilege(code,'SCOTT','REG_INTERESTS','INTEREST',caller);
 end if;
   end if;
    if (:new."INTEREST" is not null) then
    isvalid := exfsys.exf$expisvalid (54,'SCOTT','REG_PRIORITIES', 'SELECT
/*+ EXPR_CORR_CHECK USE_WEAK_NAME_RESL */  1 FROM
    TABLE(CAST(null AS EXF$NTT_53085)) exf$_eqast
     WHERE  '||:new."INTEREST");
    end if;
  exception when others then raise;
 end;
```

Creating an Expression Set

Once you've inserted a few WHERE clauses (without the keyword WHERE) consisting of predicates linked together with AND, OR and NOT into the column linked to an attribute set, this set of values, taken together, becomes the *expression set*.

```
SQL>  INSERT INTO reg_interests VALUES (
2     1, 'class_year = ''SENIOR'' AND prereqs_needed = ''Y''');

SQL>  INSERT INTO reg_interests VALUES (
2     2, 'class_year = ''SENIOR'' AND estimated_graduation_date < SYSDATE +
180');

SQL>  INSERT INTO reg_interests VALUES (
2     3, 'required_prereq IS NOT NULL');

SQL> INSERT INTO reg_interests VALUES (
2    4, 'enroll_date < SYSDATE + 180 AND estimated_graduation_date < SYSDATE
+ 180');

SQL>  INSERT INTO reg_interests VALUES (
2     5, 'class_year <> ''SENIOR'' AND estimated_graduation_date < SYSDATE +
180');

SQL>  INSERT INTO reg_interests VALUES (6, 'max_enrollees = ''Y''');

SQL>  INSERT INTO reg_interests VALUES (
2     7, '(enroll_date < SYSDATE + 180 AND prereqs_needed = ''Y'') OR
     (estimated_graduation_date < SYSDATE + 180 AND prereqs_needed = ''Y'')');
```

Expression Evaluation

```
SQL>  SELECT a.student_id, a.name, e.interest_id
•       FROM student a, enrollment b, section c, course d,
•            reg_interests e
4      WHERE a.student_id = b.student_id
5        AND b.section_id = c.section_id
6        AND c.course_no  = d.course_no
7        AND EVALUATE(
8            e.interest,
9            reg_priorities(a.student_id, a.class_year,
10           a.prereqs_needed, a.estimated_graduation_date,
11           b.enroll_date, b.section_id, c.course_no,
12           d.required_prereq, b.max_enrollees).getVarchar()
13   ) = 1
14   ORDER BY a.student_id, e.interest_id;
```

Expression

Set of attribute values

The SQL EVALUATE operator takes two arguments:

- An expression
- A set of attribute values

Table Aliases

When using table joins and the SQL EVALUATE operator, you may want to consider using the EXF$TABLE_ALIAS type in your attribute sets.

```
BEGIN
   DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE('reg_priorities', 'enrollment'
                                        EXF$TABLE_ALIAS ('scott.enrollment'));
END;
/
```

This allows you to store expressions of the form

```
ENROLLMENT.ENROLL_DATE > SYSDATE - 5
```

How Does the SQL EVALUATE Operator Work?

```
EVALUATE(expression, attribute set) = 1;
```

```
EVALUATE(
        e.interest,
        reg_priorities(a.student_id, a.class_year,
       a.prereqs_needed, a.estimated_graduation_date,
       b.enroll_date, b.section_id, c.course_no,
       d.required_prereq, b.max_enrollees).getVarchar()
   ) = 1
```

- The first argument is merely the text column containing the expression(s).
- To generate the second argument, you can use the constructor function, reg_priorities, to create an instance of an object type corresponding to the reg_priorities attribute set.
- To return a formatted string of attribute name-value pairs, invoke the getVarchar method.
- EVALUATE returns a 1 when a given set of attribute values results in an expression being true. Otherwise, EVALUATE returns zero (0).

What Takes Place When Expressions Are Evaluated?

- The EVALUATE function issues a recursive SELECT against DUAL (to determine whether an expression is true for each set of attribute values.)
- This SELECT is issued each time EVALUATE is called:

(Number of Expressions x Number of Results)

(7 * 9), in our example, = 63 recursive SELECT statements

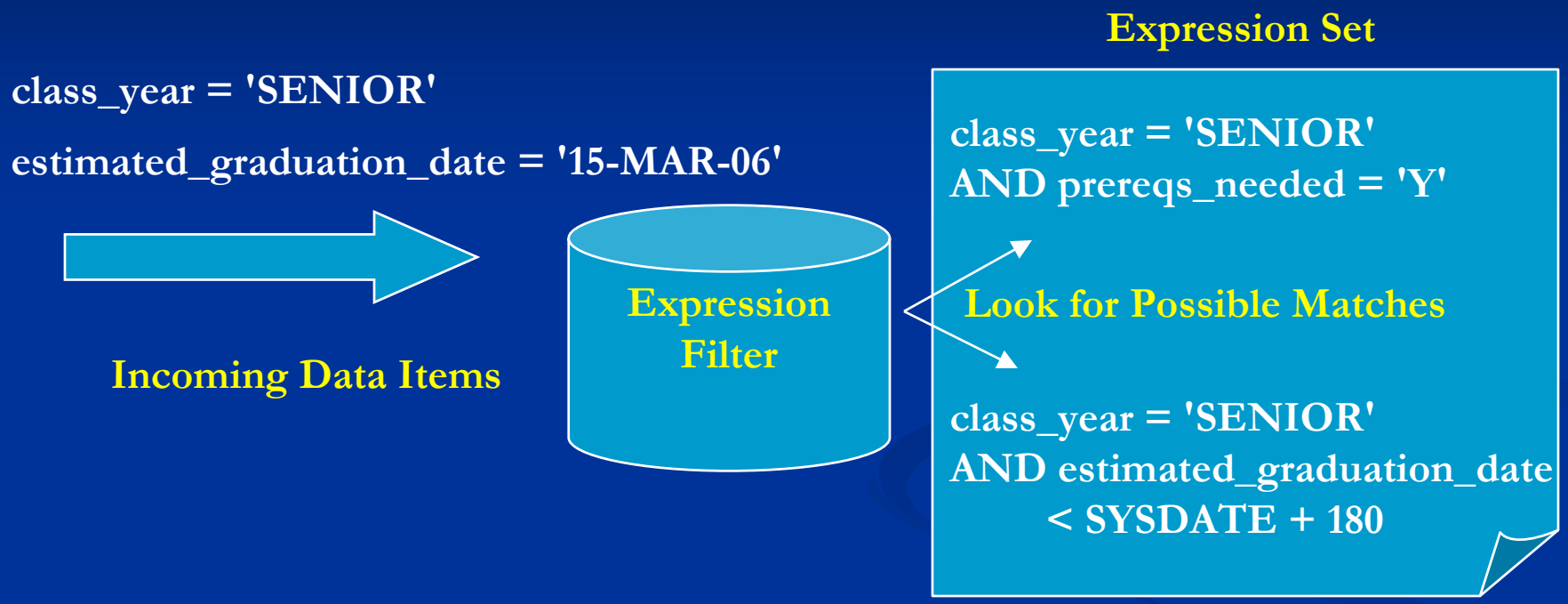
```
SELECT /*+ USE_WEAK_NAME_RESL */ 1 FROM
     (SELECT   :1 STUDENT_ID, :2 CLASS_YEAR, :3 PREREQS_NEEDED,
               :4 ESTIMATED_GRADUATION_DATE, :5 ENROLL_DATE,
               :6 SECTION_ID, :7 COURSE_NO, :8 REQUIRED_PREREQ,
               :9 MAX_ENROLLEES from dual) exf$dumalias
  WHERE class_year = 'SENIOR'
    AND estimated_graduation_date < SYSDATE + 180

SELECT /*+ USE_WEAK_NAME_RESL */ 1 FROM
     (SELECT   :1 STUDENT_ID, :2 CLASS_YEAR, :3 PREREQS_NEEDED,
               :4 ESTIMATED_GRADUATION_DATE, :5 ENROLL_DATE, :6 SECTION_ID,
               :7 COURSE_NO, :8 REQUIRED_PREREQ, :9 MAX_ENROLLEES
        from dual) exf$dumalias
  WHERE class_year = 'SENIOR' AND prereqs_needed = 'Y'
```

Indexing Expressions

- If you have more than a small number of expressions in a set, you should index them, using an expression set index.
- In so doing, you greatly reduce your database's need to use recursive SQL when evaluating your expressions.
- The nice thing about expression indexes (among other things) is that, starting from an attribute value set, the goal is to optimize the path to find expressions that are true for that set.
- This is better than starting from an expression, then searching for data that makes that expression true.

Expression Filter Works its Way from the Data to the Expression Set



Creating Expression Indexes

- In its simplest form, the statement to create an index is as follows:

```
CREATE INDEX RegInterestIdx ON Reg_Interests(Interest)
  INDEXTYPE IS EXFSYS.EXPFILTER;
```

Creating Expression Indexes from Statistics

If you already have a representative set of expressions (which we do), you can automate the tuning process by collecting statistics on the expression set first, then creating the index from the statistics.

```
BEGIN
   DBMS_EXPFIL.GET_EXPRSET_STATS(expr_tab => 'Reg_Interests',
                                 expr_col => 'Interest');
END;
/

CREATE INDEX RegInterestIdx ON Reg_Interests(Interest)
                               INDEXTYPE IS EXFSYS.EXPFILTER
PARAMETERS ('STOREATTRS TOP 4 INDEXATTRS TOP 2');
```

What Does That Parameters Clause Do?

For the answer to that, we'll need to back up a bit ……

- **Expressions in an expression set tend toward particular commonalities in their predicates.**
- **Expression Filter indexes work to group predicates, based on these commonalities, to reduce processing costs.**

- **For Example:**

```
class_year = 'SENIOR' and
class_year = 'JUNIOR'
```

Both share a common left-hand side operand.

- **The truth or falseness of one predicate can be determined based on the outcome of the other.**

Which Predicates are Indexable?

- Any predicate with a constant or literal on the right hand side that uses one of the following predicate operators:
  - =
  - !=
  - >
  - <
  - >=
  - >=
  - BETWEEN
  - IS NULL
  - IS NOT NULL
  - LIKE
  - NVL

And Which Are Not?

- These predicates are stored in their original form and are not indexed. They are evaluated last during expression evaluation:
  - Predicates using a variable in the right-hand side operand (as opposed to using constants and literals.)
  - IN Lists
  - Predicates using the LIKE operator with a leading wild-card character.
  - Duplicate predicates in an expression with the same left-hand operand.
  - Predicates using combinations of NOT and BETWEEN.

So Back to That Parameters Clause

- This statement:

```
PARAMETERS ('STOREATTRS TOP 4 INDEXATTRS TOP 2');
```

Simply instructs the index creation statement to store the four most selective predicates and, of those four, index the top two.

What Happens When You Create an Expression Filter Index?

- Several objects are created in the schema of the owner of the table housing the Expression column.
  - A predicate table: EXF$PTAB_*n*
  - One or more indexes on this predicate table: EXF$PTAB_*n*_IDX_*m*
  - A package known as the Access Function package: EXF$AFUN_*n*

The EXP$PTAB_*n* Table

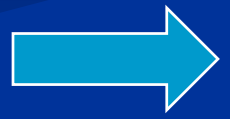
```
SQL> desc exf$ptab_53206
 Name                                      Null?    Type
----------------------------------------- -------- -------
 EXF$EXPROWID                                       ROWID
 EXF$SPARSEPRED                             VARCHAR2(4000)
 EXF$PTATTR_1_OP                                    NUMBER
 EXF$PTATTR_1_CT                              VARCHAR2(10)
 EXF$PTATTR_2_OP                                    NUMBER
 EXF$PTATTR_2_CT                               VARCHAR2(1)
 EXF$PTATTR_3_OP                                    NUMBER
 EXF$PTATTR_3_CT                               VARCHAR2(1)
 EXF$PTATTR_4_OP                                    NUMBER
 EXF$PTATTR_4_CT                               VARCHAR2(6)
```

What Are Its Contents Based on Our Index?

```
SQL> select exf$ptattr_1_op, exf$ptattr_1_ct, exf$ptattr_2_op,
  2         exf$ptattr_2_ct
  3    from exf$ptab_53206;

EXF$PTATTR_1_OP EXF$PTATTR EXF$PTATTR_2_OP E
--------------- ---------- --------------- -
              1 SENIOR                    1 Y
                                          1 Y
                                          1 Y
              1 SENIOR
```

- The numbers in the OP columns correspond to the operator (in our case, the "=" operator), and its frequency within each data item corresponding to a particular expression.
- The numbers in the CT columns correspond to the right-hand side operand value assigned to the predicate being indexed.
- To find out which left-hand side operands were indexed, you can read the source code for the FILTER_PROC procedure in the EXF$AFUN_53206 (EXF$AFUN_*n*) package,

OR 

USER_EXPFIL_EXPRSET_STATS

You could also issue the following query:

```
SQL> SELECT attribute_exp, pct_occurrence
 2     FROM user_expfil_exprset_stats
 3    WHERE expr_table = 'REG_INTERESTS'
 4      AND expr_column = 'INTEREST'
 5   ORDER BY pct_occurrence, attribute_exp DESC

ATTRIBUTE_EXP                            PCT_OCCURRENCE
---------------------------------------- --------------
STUDENT_ID                                            0
SECTION_ID                                            0
ESTIMATED_GRADUATION_DATE                             0
ENROLL_DATE                                           0
COURSE_NO                                             0
REQUIRED_PREREQ                                    12.5
MAX_ENROLLEES                                      12.5
PREREQS_NEEDED                                     37.5
CLASS_YEAR                                         37.5
```

Expression Validation Utility

- Used to verify an expression set.
- Identifies expressions that have become invalid since insertion.
- Collects references to the invalid expressions in an expression table.
- However, an exception table must be provided. Otherwise, the utility fails upon first encounter with an invalid expression.

BUILD_EXCEPTIONS Table

- You can ensure that the Validation Utility does not fail to collect references to invalid expressions with the following bit of code:

```
BEGIN
   DBMS_EXPFIL.BUILD_EXCEPTIONS_TABLE
      (exception_tab => 'RegInterestExceptions');

   DBMS_EXPFIL.VALIDATE_EXPRESSIONS
      (expr_tab      => 'RegInterests',
       expr_col      => 'Interest',
       exception_tab => 'RegInterestExceptions');
END;
/
```

Bulk Loading Expressions

- For SQL*Loader operations, expressions are treated as strings loaded into a VARCHAR2 column of a database table.

```
LOAD DATA
INFILE *
INTO TABLE reg_interests
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(Interest_ID, Interest)
BEGINDATA
1, "class_year = 'SENIOR' AND prereqs_needed = 'Y'"
2, "class_year = 'SENIOR' AND estimated_graduation_date <
   SYSDATE + 180"
3, "required_prereq IS NOT NULL"
```

Looking to Perform a Direct Load?

❶
```
DROP INDEX RegInterestIdx;
```

❷
```
BEGIN
 DBMS_EXPFIL.UNASSIGN_ATTRIBUTE_SET
   (expr_tab => 'RegInterests',
    expr_col => 'Interest');
END;
```

❸ Then you can perform the bulk load operation and direct load is possible.

❹ Reassign the attribute set to the expression column using a value of "TRUE" for the "FORCE" parameter of the ASSIGN_ATTRIBUTE_SET procedure.

❺ Validate the newly-added expressions with the VALIDATE_EXPRESSIONS procedure.

❻ Recreate the indexes on the expression columns.

Expression Filter Uses and Benefits

- Performance
- Code reusability and flexibility
- Code accuracy

- The optimizer can use Oracle Expression Filter's bitmap indexes and expression set index tables to optimize evaluation.

Expression Filter Uses and Benefits

- Performance
- Code reusability and flexibility
- Code accuracy

- Expressions in text columns are much easier to change than trigger-based logic or code embedded within stored procedures. Users can even write their own expressions.

Expression Filter Uses and Benefits

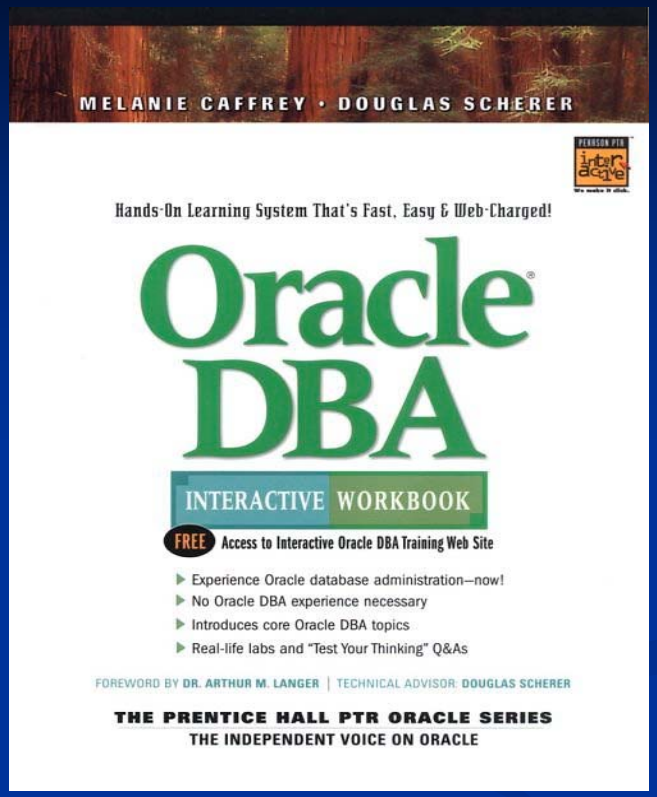
- Performance
- Code reusability and flexibility
- Code accuracy

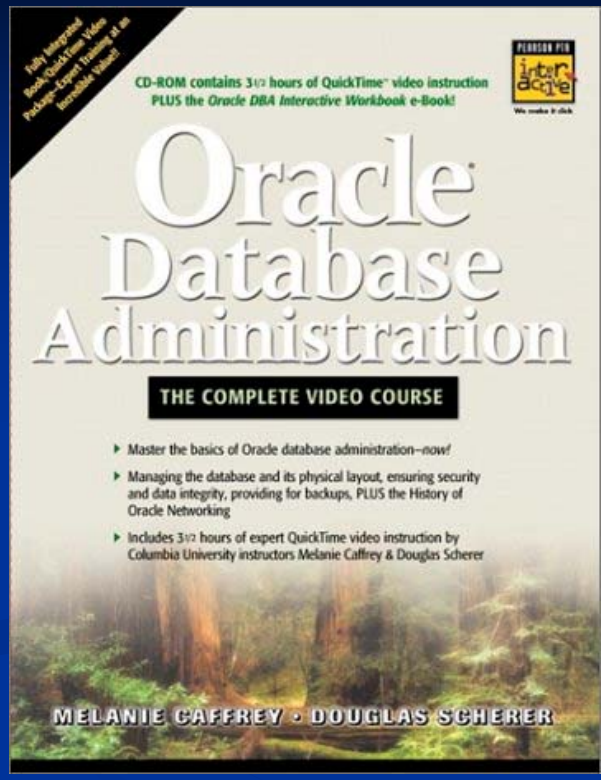
- Since the expressions are already in the database engine, they have been parsed and (hopefully) tested. Oracle Expression Filter can leverage that capability.

Expression Filter Goodies and Gotchas

- Goodie: Greatly reduces the need for storing such items of data in intermediate, associative tables.
- Goodie: Enables batch processing of incoming data.
- Gotcha: Expressions cannot contain subqueries.
- Gotcha: If the expressions refer to user-defined functions, these functions must be explicitly added to an attribute set. (Such functions cannot be derived from object types.)
- Goodie: Expression Indexing greatly reduces redundant, recursive SQL processing.
- Gotcha: Expression Indexing is only available within the Enterprise Edition of Oracle.
- Gotcha: See the laundry list on Slide #21 for operators that Expression Indexing is not yet equipped to handle.
- Goodie: NULL values are acceptable when passing values for all attributes to the SQL EVALUATE operator.
- Gotcha: A join condition column cannot be NULL, otherwise you receive incorrect Expression results.
- Goodie: Expression Filter creates objects to enhance and improve its usability.
- Gotcha: Some of these objects get stored in the EXFSYS schema. And some get stored in the schema that owns the table containing Expression(s). Make sure you know which items get created in which location.

Thank You Very Much!





Melanie Caffrey
Harris Corporation
mcaffrey@harris.com
mlc51@columbia.edu