Oracle Database 10g Expression Filter Overview

An Oracle White Paper
May 2005
Executive Overview................................................................. 3
Introduction................................................................................. 3
  Application Attributes................................................................. 4
  Match Incoming Data with Conditional Expressions............... 4
  Maintain Complex Table Relationships .................................. 5
Features....................................................................................... 6
  Expressions .............................................................................. 6
  Attribute Sets ........................................................................ 7
  Expression Columns ................................................................. 8
  Insert, Update, And Delete Expressions ................................... 9
  Evaluate Operator ................................................................ 9
  Evaluation Semantics............................................................... 11
  Privileges ............................................................................. 11
Indexing....................................................................................... 12
Expressions With Xpath Predicates ........................................... 13
Expression Filter Management.................................................. 13
Feature summary....................................................................... 13
Conclusion.................................................................................. 14
EXECUTIVE OVERVIEW

Expression Filter is a feature of the Oracle Database 10g that allows application developers to store, index, and evaluate conditional expressions (“expressions”) in one or more columns of a relational table. Expressions are a useful way to describe interest in some expected data. Applications involving information distribution, demand analysis, and task assignment can benefit from Expression Filter.

Expression Filter matches incoming data with expressions stored in a column to identify rows of interest. It can also maintain complex relationships by matching data in one table with expressions in a second table.

Expression Filter saves application developers time and labor:

- It simplifies application SQL queries
- It allows expressions to be inserted, updated, and deleted without changing the application
- It enables reuse of conditional expressions by separating them from the application and storing them in the database

If you are considering Expression Filter you may also want to consider Rules Manager, which uses Expression Filter and provides additional capabilities. Rules Manager is a feature of Oracle Database 10g Release 2 that allows application developers to define rules in Oracle Database to respond to events of any complexity with better scalability and operational characteristics than the alternatives.

This white paper provides an overview of Expression Filter concepts, use cases and features. Please refer to the Oracle white paper Oracle Database 10g: Handling Application Events with Rules Manager and documentation Oracle Database Application Developer’s Guide – Rules Manager and Expression Filter for more information.

INTRODUCTION

Expression Filter provides a datatype, an operator, and an indextype to store, evaluate, and index conditional expressions (expressions) that describe interest in a data item or piece of information. Expressions are stored in a column of a
user table. Expression Filter matches expressions in a column with a data item passed by a SQL statement or with data stored in one or more tables and evaluates each expression to be true or false. Optionally, expressions can be indexed when using the Enterprise Edition of the Oracle Database.

Expression Filter includes the following elements:

- **Expression datatype**: A virtual datatype created through a constraint placed on a `VARCHAR2` column in a user table that stores expressions.
- **EVALUATE operator**: A SQL operator that evaluates expressions for each data item.
- **Administrative utilities**: A set of utilities that validate expressions and suggest optimal index structure.
- **Expression indexing**: Enhances performance of the EVALUATE operator for large expression sets. Expression indexing is available in the Enterprise Edition of the Oracle Database.

**Application Attributes**

Expression Filter is a good fit for applications that have data with the following characteristics:

- A large number of data items exist to be evaluated.
- Each data item has structured data attributes, for example, `VARCHAR`, `NUMBER`, `DATE`, `XMLTYPE`.
- Incoming data is evaluated by a significant number of unique and persistent queries containing conditional expressions.
- The conditional expression (in SQL `WHERE` clause format) describes an interest in incoming data items.
- The conditional expressions compare attributes to values using relational operators (`=`, `!=`, `<`, `>`, and so on).

Potential uses for Expression Filter include matching incoming data with conditional expressions stored in the database to identify rows of interest and maintaining many-to-many relationships between tables. The following examples illustrate these uses.

**Match Incoming Data with Conditional Expressions**

Expression Filter can match incoming data with conditional expressions stored in the database to identify rows of interest. For example, consider an application that matches buyers and sellers of cars. A table called `CONSUMER` includes a column called `BUYER_PREFERENCES` with an Expression datatype. The `BUYER_PREFERENCES` column stores an expression for each consumer.
describing the kind of car they want to purchase, including make, model, year, mileage, color, options, and price. Data about cars for sale is included with the `EVALUATE` operator in the `SQL WHERE` clause.

The `SQL EVALUATE` operator matches the incoming car data with the expressions to find prospective buyers. The `SQL EVALUATE` operator also enables batch processing of incoming data. Data can be stored in a table called `CARS` and matched with expressions stored in the `CONSUMER` table using a join between the two tables.

The `EVALUATE` operator saves time by matching a set of expressions with incoming data and enabling large expression sets to be indexed for performance. It saves labor by allowing expressions to be inserted, updated, and deleted without changing the application and providing a result set that can be manipulated in the same `SQL` statement, for instance to order or group results. In contrast, a procedural approach stores results in a temporary table that must be queried for further processing, and expressions cannot be indexed.

### Maintain Complex Table Relationships

Expression Filter can maintain many-to-many relationships between tables. Using the previous example:

- A car may be of interest to one or more buyers.
- A buyer may be interested in one or more cars.
- A seller may be interested in one or more buyers.

To answer questions about these relationships, the incoming data about cars is stored in a table called `CARS` with an expression column called `SELLER_PREFERENCES`. The `CONSUMERS` table includes a column called `BUYER_PREFERENCES`. The `SQL EVALUATE` operator can answer questions such as:

- Which cars are of interest to each consumer?
- Which buyers are of interest to each seller?
- What demand exists for each car? This can help determine optimal pricing.
- What unsatisfied demand is there? This can help determine inventory requirements.

This declarative approach saves labor. No action is needed if changes are made to the data or the expressions. Compare this to the traditional approach where a mapping table is created to store the relationship between the two tables. A trigger must be defined to recompute the relationships and to update the
mapping table if the data or expressions change. In this case, new data must be
compared to all expressions and a new expression must be compared to all data.

FEATURES

Expression Filter provides a comprehensive PL/SQL API that you can add to
new and existing applications. The DBMS_EXPRFIL package contains procedures
to manage expression data (see the Key Features section at the end of this
document). These procedures are referenced in the following sections that
describe the four basic steps to create and use an Expression column:

1. Define an attribute set.
2. Define a column of Expression datatype in a user table.
3. Insert expressions in the table.
4. Apply the EVALUATE operator to compare expressions to incoming data
   items.

Expressions

Conditional expressions describe interests in an item of data. Expressions are
stored in a column of a user table and compared, using the EVALUATE operator,
to incoming data items specified in a SQL WHERE clause or to a table of data.
Expressions are evaluated as true or false or return a null value if an expression
does not exist for a row.

An expression describes interest in an item of data using one or more variables,
known as elementary attributes. An expression can also include literals, Oracle
supplied functions, user-defined functions, and table aliases. A valid expression
consists of one or more simple conditions called predicates. The predicates in
the expression are linked by the logical operators AND and OR. Expressions must
adhere to the SQL WHERE clause format. An expression is not required to use all
the defined elementary attributes; however, the incoming data must provide a
value for every elementary attribute. Null is an acceptable value.

For example, the following expression captures the interest of a user in a car (the
data item) with the model, price, and year as attributes and includes the Oracle
supplied function UPPER.

UPPER(Model) = 'TAURUS' and Price < 20000 and Year > 2000

Expressions are stored in a column of a user table with an Expression datatype.
The values stored in a column of this type are constrained to be expressions. A
user table can have one or more expression type columns. A query to display the
contents of an expression column displays the expressions in string format.

You insert, update, and delete expressions using standard SQL. A group of
expressions that are stored in a single column is called an expression set and
shares a common set of elementary attributes. This set of elementary attributes
plus any functions used in the expressions are the metadata for the expression set. This metadata is referred to as the **attribute set**. The attribute set consists of the elementary attribute names and their datatypes and any functions used in the expressions. The expression column uses the attribute set to validate changes and additions to the expression set. An expression stored in the expression column can only use the elementary attributes and functions defined in the corresponding attribute set. Expressions cannot contain subqueries.

The illustration below shows the relationship between the CAR4SALE attribute set and the expression set stored in the INTEREST column with some examples of valid expressions.

<table>
<thead>
<tr>
<th>Clid</th>
<th>Zipcode</th>
<th>Phone</th>
<th>Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32611</td>
<td>917 768 4633</td>
<td>Model = ‘Taurus’ and Price &lt; 15000 and Mileage &lt; 25000</td>
</tr>
<tr>
<td>2</td>
<td>03060</td>
<td>603 983 3463</td>
<td>Model = ‘Mustang’ and Year &gt; 1999 and Price &lt; 20000</td>
</tr>
<tr>
<td>3</td>
<td>03060</td>
<td>603 484 7013</td>
<td>HorsePower(Model, Year) &gt; 200 and Power &lt; 20000</td>
</tr>
</tbody>
</table>

**Attribute Sets**

A special form of an Oracle object type is used to create an attribute set. The attribute set defines the elementary attributes for an expression set. It implicitly allows all Oracle supplied SQL functions to be valid references in the expression set. If the expression set refers to a user-defined function, it must be explicitly added to the attribute set. An elementary attribute in an attribute set can refer to data stored in another database table using table alias constructs. One or more or all elementary attributes in an attribute set can be table aliases. If an elementary attribute is a table alias, the value assigned to the elementary attribute is a ROWID from the corresponding table.

You can create an attribute set using one of two approaches:
Use an existing object type to create an attribute set with the same name as the object type. This approach is most appropriate to use when the attribute set does not contain any table alias elementary attributes. Use the CREATE_ATTRIBUTE_SET procedure to do this.

```sql
CREATE OR REPLACE TYPE Car4Sale AS OBJECT
(Model VARCHAR2(20),
 Year NUMBER,
 Price NUMBER,
 Mileage NUMBER);
/
BEGIN
 DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set => 'Car4Sale',
 from_type => 'YES');
END;
/
```

The other approach is to add individual elementary attributes to an existing attribute set. Expression Filter automatically creates an object type to encapsulate the elementary attributes and gives it the same name as the attribute set. This approach is most appropriate to use when the attribute set contains one or more elementary attributes defined as table aliases. Use the ADD_ELEMENTARY_ATTRIBUTE procedure to do this.

```sql
BEGIN
 DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set => 'Car4Sale');
 DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
 attr_set => 'Car4Sale',
 attr_name => 'Mileage',
 attr_type => 'NUMBER');
END;
/
```

If the expressions refer to user-defined functions, you must add the functions to the corresponding attribute set using the ADD_FUNCTIONS procedure.

```sql
CREATE or REPLACE FUNCTION CrashTestRating(Model VARCHAR2, Year VARCHAR2)
 ...
 END CrashTestRating; /
BEGIN
 BEGIN
 DBMS_EXPFIL.ADD_FUNCTIONS (attr_set => 'Car4Sale',
 funcs_name => 'CrashTestRating');
 END; /
```

**Expression Columns**

The **Expression datatype** is a virtual datatype. Assigning an attribute set to a VARCHAR2 column in a user table creates a column of Expression datatype. The attribute set determines which elementary attributes and user-defined functions can be used in the expression set. An attribute set can be used to create multiple
columns of Expression datatype in the same table and in other (multiple) tables in the same schema. Note that an attribute set in one schema cannot be associated with a column in another schema. To create a column of Expression datatype, assign an attribute set to a new or existing VARCHAR2 column using the ASSIGN_ATTRIBUTE_SET procedure.

```
BEGIN
    DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET (
        attr_set => 'Car4Sale',
        expr_tab => 'Consumer',
        expr_col => 'Interest');
END;
/
```

To remove an attribute set from a column, use the UNASSIGN_ATTRIBUTE_SET procedure. To drop an attribute set not being used for any expression set, use the DROP_ATTRIBUTE_SET procedure. To copy an attribute set across schemas, use the COPY_ATTRIBUTE_SET procedure.

**Insert, Update, And Delete Expressions**

Use standard SQL to insert, update, and delete expressions. When an expression is inserted or updated, it is checked for correct syntax and constrained to use the elementary attributes and functions specified in the corresponding attribute set. An error message is returned if the expression is not correct. Expression data can be loaded into a column of Expression datatype using SQL*Loader.

```
INSERT INTO Consumer VALUES (1, 32611, '917 768 4633',
    'Model=''Taurus'' and Price < 15000 and Mileage < 25000');
```

**Evaluate Operator**

Use the SQL EVALUATE operator in the WHERE clause of a SQL statement to compare stored expressions to incoming data items. The EVALUATE operator returns 1 for an expression that matches the data item and 0 for an expression that does not match. For any null values stored in the expression column, the EVALUATE operator returns NULL.

The EVALUATE operator has two arguments: the name of the column storing the expressions and the data item to which the expressions are compared. In the data item argument, values must be provided for all elementary attributes in the attribute set associated with the expression column. Null is an acceptable value. The data item can be specified either as a VARCHAR containing string-formatted name-value pairs, or as an AnyData instance containing one or more binary typed attributes.
**Data Item Formatted as a String**

If the values of all the elementary attributes in the attribute set can be represented as readable values, such as those stored in VARCHAR, DATE, and NUMBER datatypes and constructors formatted as a string, then the data item can be formatted as a string.

```sql
SELECT * FROM Consumer WHERE
EVALUATE
(Consumer.Interest, 'Model=>' ''Mustang'', Year=>2000, Price=>18000, Mileage=>22000) = 1;
```

If a data item does not require a constructor for any of its elementary attribute values, then a list of values provided for the data item can be formatted as a string (name-value pairs) using two getVarchar methods (a STATIC method and a MEMBER method) in the object type associated with the attribute set. If you use one of these methods you don’t need to specify the elementary attribute names in the WHERE clause. The STATIC method formats the data item without creating the object instance. The MEMBER method can be used if the object instance is already available. The STATIC and MEMBER methods are implicitly created for the object type.

The following examples do not specify the elementary attribute names for model, year, mileage and price. The STATIC and MEMBER methods are used instead to format the values into name/value pairs.

```sql
SELECT * FROM Consumer WHERE
EVALUATE (Consumer.Interest, Car4Sale.getVarchar('Mustang', -- STATIC
   getVarchar API --
   2000,18000,22000)
) = 1;

SELECT * FROM Consumer WHERE
EVALUATE (Consumer.Interest,
   Car4Sale('Mustang',2000,18000,22000).getVarchar( -- MEMBER getVarchar() API --
)) = 1;
```

**Data Item Formatted as an AnyData Instance**

Any data item can be formatted using an AnyData instance. AnyData is an Oracle supplied type that can hold instances of any Oracle datatype, both Oracle supplied and user-defined. An instance of the object type reflecting the corresponding attribute set is converted into an AnyData instance using the AnyData convertObject API. A data item formatted as an AnyData instance is converted back into the original object before the expressions are evaluated. To avoid the cost of object type conversions, string-formatted data items are recommended whenever possible.

```sql
SELECT * FROM Consumer WHERE
EVALUATE (Consumer.Interest, AnyData.convertObject(
   Car4Sale('Mustang',2000,18000,22000))
)
Evaluation Semantics

When an expression is inserted or updated, Expression Filter validates the syntax and ensures that the expression refers to valid elementary attributes and functions associated with the attribute set. The \texttt{EVALUATE} operator evaluates expressions using the privileges of the owner of the table that stores the expressions. For instance, if an expression includes a reference to a user-defined function, during its evaluation, the function is executed with the privileges of the owner of the table. References to schema objects with no schema extensions are resolved in the table owner's schema.

An expression that refers to a user-defined function may become invalid if the function is modified or dropped. An invalid expression causes the SQL statement evaluating the expression to fail. To recover from this error, replace the missing or modified function with the original function.

The Expression Validation utility is used to verify an expression set. It identifies expressions that have become invalid since they were inserted, perhaps due to a change made to a user-defined function or a table. This utility collects references to the invalid expressions in an exception table.

The following example collects references to invalid expressions found in the \texttt{CONSUMER} table. The \texttt{BUILD EXCEPTIONS TABLE} procedure creates the exception table, \texttt{INTEREST EXCEPTIONS}, in the current schema. The \texttt{VALIDATE EXPRESSIONS} procedure validates the expressions and stores the invalid expressions in the \texttt{INTEREST EXCEPTIONS} table.

```sql
BEGIN
  DBMS_EXPRFIL.BUILD_EXCEPTIONS_TABLE (exception_tab => 'INTEREST_EXCEPTIONS');
  DBMS_EXPRFIL.VALIDATE_EXPRESSIONS (expr_tab => 'Consumer',
                                       expr_col => 'Interest',
                                       exception_tab => 'INTEREST_EXCEPTIONS');
END; /
```

Privileges

A user requires \texttt{SELECT} privileges on a table storing expressions to evaluate them. The \texttt{EVALUATE} operator evaluates expressions using the privileges of the owner of the table that stores the expressions. The privileges of the user issuing the query are not considered.

Expressions can be inserted, updated, and deleted by the owner of the table. Other users must have \texttt{INSERT} and \texttt{UPDATE} privileges for the table and \texttt{INSERT EXPRESSION} and \texttt{UPDATE EXPRESSION} privileges for a specific
expression column in the table. Use the GRANT_PRIVILEGE and REVOKE_PRIVILEGE procedures to grant and revoke privileges.

In the following example, the owner of the CONSUMER table grants expression privileges, using the GRANT_PRIVILEGE procedure, on the Interest column to a user named Andy.

```sql
BEGIN
    DBMS_EXPRIL.GRANT_PRIVILEGE (expr_tab => 'Consumer',
                                   expr_col => 'Interest',
                                   priv_type => 'INSERT EXPRESSION',
                                   to_user => 'Andy');
END; /
```

**Indexing**

An index can be defined on a column storing expressions to quickly find expressions that evaluate to true for a data item. This is most helpful when a large expression set is evaluated for a data item. The EVALUATE operator determines whether or not to use the index based on its access cost. The indextype, Expression Filter, is used to create and maintain indexes. If a column of Expression datatype is not indexed, the EVALUATE operator builds a dynamic query for each expression stored in the column and executes it using the values passed in as the data item.

An Expression Filter index, defined on a set of expressions, takes advantage of the logical relationships between multiple predicates by grouping them based on the commonality of their left-hand sides and operators. The left-hand sides are arithmetic expressions consisting of one or more elementary attributes and user-defined functions. The left-hand sides appear in the predicates along with an operator and a constant on the right-hand side, as in this example.

```
HORSEPOWER(model,year)>=150
```

The cost of evaluating a predicate in an expression set depends on the group to which it belongs. The index for an expression set can be tuned by identifying the appropriate predicate groups as index parameters.

The predicates that can be indexed with the Expression Filter indexing mechanism include any predicate with a constant on the right-hand side that uses one of the following predicate operators: =, !=, >, <, >=, <=, BETWEEN, IS NULL, IS NOT NULL, LIKE, and NVL. The predicates that cannot be indexed are preserved in their original form and they are evaluated by value substitution in the last stage of expression evaluation.

A query using the EVALUATE operator on an expression column can force the use of the index defined on such a column with an optimizer hint. In other cases, the optimizer determines the cost of the Expression Filter index based scan and compares it with the cost of alternate execution plans.
Expressions With XPath Predicates

The expressions stored in a column of a table may contain XPath predicates defined on XMLType attributes. Using the Oracle supplied XMLType datatype, users can apply XPath predicates to XML documents within a standard SQL WHERE clause of a query. These predicates use operators such as EXTRACT and EXISTSNODE on an instance of XMLType datatype to process an XPath expression for the XML instance.

To process a large set of XPath predicates in an expression set efficiently, the Expression Filter index defined for the expression set can be configured for the XPath predicates (in addition to simple predicates). Such indexes exploit the commonalities in the XPath expressions to efficiently compare them to a data item. These commonalities are based on the positions and the values for the XML elements and attributes commonly appearing in the XPath expressions.

The indexable constructs in an XPath expression are the levels (or positions) of XML elements, the values for text nodes in XML elements, the positions of XML attributes, and the values for XML attributes. For this purpose, an XPath predicate is treated as a combination of positional and value filters on XML elements and attributes appearing in an XML document.

Expression Filter Management

The Expression Filter metadata for attributes, attribute sets, expression sets, indexes, optimizer statistics and privileges can be viewed using the Expression Filter views defined with a xxx_EXPFIL prefix, where xxx can be USER or ALL. These views are read-only to the users and are created and maintained by the Expression Filter APIs.

FEATURE SUMMARY

SQL Operators And Statements

- **EVALUATE** - Matches an expression set with a given data item or table of data items
- **CREATE INDEX** - Creates an Expression Filter index on a column storing expressions
- **ALTER INDEX REBUILD** - Rebuilds an Expression Filter index
- **ALTER INDEX RENAME TO** - Changes the name of an Expression Filter index
- **DROP INDEX** - Drops an Expression Filter index

Management Procedures In The DBMS_EXPFIL Package

- **ADD_ELEMENTARY_ATTRIBUTE** - Adds the specified attribute to the attribute set.
• **ADD_FUNCTIONS** - Adds a function, type, or package to the approved list of functions with an attribute set.

• **ASSIGN_ATTRIBUTE_SET** - Assigns an attribute set to a column storing expressions.

• **BUILD_EXCEPTIONS_TABLE** - Creates an exception table to hold references to invalid expressions.

• **CLEAR_EXPRSET_STATS** - Clears the predicate statistics for an expression set.

• **COPY_ATTRIBUTE_SET** - Makes a copy of the attribute set.

• **CREATE_ATTRIBUTE_SET** - Creates an attribute set.

• **DEFAULT_INDEX_PARAMETERS** - Assigns default index parameters to an attribute set.

• **DEFAULT_XPINDEX_PARAMETERS** - Assigns default XPath index parameters to an attribute set.

• **DEFRAG_INDEX** - Rebuilds the bitmap indexes online to reduce fragmentation.

• **DROP_ATTRIBUTE_SET** - Drops an unused attribute set.

• **GET_EXPRSET_STATS** - Collects predicate statistics for an expression set.

• **GRANT_PRIVILEGE** - Grants an expression DML privilege to a user.

• **INDEX_PARAMETERS** - Assigns index parameters to an expression set.

• **REVOKE_PRIVILEGE** - Revokes an expression DML privilege from a user.

• **UNASSIGN_ATTRIBUTE_SET** - Breaks the association between a column storing expressions and the attribute set.

• **VALIDATE_EXPRESSIONS** - Validates expression metadata and the expressions stored in a column.

• **XPINDEX_PARAMETERS** - Assigns XPath index parameters to an expression set.

**CONCLUSION**

Expression Filter is a feature of the Oracle Database 10g that allows application developers to store, index, and evaluate conditional expressions (expressions) in one or more columns of a relational table. Expressions are a useful way to describe interest in some expected data. Expression Filter provides a datatype, an operator, and an indextype to store, evaluate, and index conditional expressions that describe interest in a data item or piece of information.
Applications involving information distribution, demand analysis, and task assignment can benefit from Expression Filter. Expression Filter saves application developers time and labor by simplifying application SQL, allowing expressions to be inserted, updated, and deleted without changing the application and enabling reuse of conditional expressions by storing them in the database.