SQL Factoring

Ajustes de SQL sem acesso aos Fontes

Por : José Laurindo Chiappa

Breve Biografia do Palestrante

- Atua como DBA Oracle desde 2.000, tendo além disso mais de dez anos como Desenvolvedor em PL/SQL, Oracle Forms & Reports e criação de SQLs ad-hoc

- Moderador do grupo oracle-br

- Pode ser contactado via https://br.groups.yahoo.com/neo/groups/oracle_br/info , https://br.linkedin.com/in/jlchiappa , jlchiappa@gmail.com / jlchiappa@yahoo.com.br

Importância do código SQL

A única maneira de acessarmos os dados contidos num RDBMS é através de programação SQL, que será processado pelo RDBMS e os dados retornados .

Dado o amplo leque de possibilidades de representação das necessidades em comandos SQL, é de vital importância que o código enviado esteja Correto funcionalmente e além disso ser o mais otimizado possível.

Como Corrigir SQLs Impróprios

Sendo SQL uma linguagem, a presença de código impróprio implica em BUG (o qual pode ser físico, ie, codificação inválida, OU lógico, ie, uma codificação sintaticamente perfeita mas inapropriada à lógica desejada) .

Fica patente portanto que a única maneira de eliminar BUGs em código é re-escrever o código em questão – existem as mais diversas ferramentas de depuração que auxiliam na busca pelo código impróprio, mas uma vez definido/encontrado a re-escrita é Inevitável.

Re-escrita de SQLs de Terceiros

Via de regra não há nenhum problema na re-escrita quando o código SQL a ser corrigido está embutido numa aplicação própria, pois nesse caso os fontes estão acessíveis e disponíveis.

Porém, num cenário em que isso não exista e os fontes não estejam acessíveis o que é possível ser feito ? Responderemos nesta Apresentação ...

!!! AVISO !!!

Antes de iniciarmos, deve ficar Claro a todos que :

⇒ A responsabilidade por correção de bugs (e portanto sobre o código-fonte) necessariamente se entende como sendo do Fornecedor da Aplicação

⇒ As técnicas de alteração de código SQL sem acesso aos fontes da Aplicação que demonstraremos DEVEM ser vistas como um recurso Emergercial, pontual, a ser aplicado em poucos casos específicos : DE FORMA ALGUMA é um procedimento Recomendado a ser utilizado rotineiramente

⇒ A eficácia / eficiência e Custo/Esforço necessariamente vão ser variáveis (pois estamos contrariando Frontalmente a prática mais recomendável de re-escrever o código impróprio, diretamente) , E não é incomum ocorrerem efeitos colaterais, já que estamos manipulando o código de maneiras não necessariamente previstas pelo Fornecedor

⇒ Será VITAL a análise de Aplicabilidade antes de se aplicar qualquer uma das técnicas, não se garante a Viabilidade de qualquer uma delas em qualquer situação a priori

Em Resumo, podemos afirmar que as técnicas que demonstraremos aqui são um WORK-AROUND, um “quebra-galho” a ser usado apenas no momento, com Extrema parcimônia e Cuidado, quando for Identificado código SQL impróprio que por qualquer motivo o responsável não pode / não vai corrigir (via RE-ESCRITA) no momento...

Podemos realizar uma comparação não técnica dizendo que estaremos aplicando um Curativo temporário num paciente grave : até pode ser uma ação benéfica mas DE FORMA ALGUMA essa é a Solução....

Tipos de Alterações

São basicamente os tipos abaixo de alterações em código SQL que discutiremos nesta Apresentação :

1) Introdução de Hints/Diretivas que influenciam no Plano de Execução, para que a metodologia de processamento (e eventualmente a performance) do SQL seja diferenciada

2) Introdução de cláusulas adicionais que não alteram as colunas a retornar (principalmente condições de WHERE/filtros adicionais)

3) Substituição completa de texto, substituindo um dado texto por um outro previamente indicado

OBS :

utilizaremos nesta Apresentação os comandos/recursos do RDBMS Oracle – porém, nenhum deles é único, praticamente todos os grandes fabricantes de RDBMSs possuem itens similares, não sendo de modo algum portanto inviável a adaptação das técnicas para outros RDBMSs.

SQL Original

SQL com Alterações

VIEWS – uma ferramenta versátil

Antes de demonstrarmos técnicas específicas para cada tipo de Alteração sem acesso ao fonte citadas, falaremos sobre as VIEWS, que são uma ferramenta genérica, servindo para diversas.

Uma view nada mais é do que uma query SQL armazenada no RDBMS sob um nome, e automaticamente aonde ela for referenciada em outros SQLs o texto indicado na criação da view será inserido na referência. Por exemplo, se a texto for :

SELECT colunasdesejadas FROM **X**;

E existir uma view **X** criada em conjunto com a query :

SELECT outrascolunas FROM Y;

Automaticamente o SQL que será processado pelo SQL engine será :

SELECT * FROM (**SELECT outrascolunas FROM Y**);

Ou seja, **X** foi Substituído pelo texto da query associada à view de nome **X** : podemos então, grosso modo, pensar na view como um Sinônimo / atalho para uma dada query SQL...

Exemplo de utilização de View para promover alteração num código SQL

Digamos que temos o seguinte SQL enviado pela aplicação ao database :

SELECT EMPLOYEE_ID , FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEES;

E necessitamos sem acesso aos fontes transformá-lo em :

SELECT EMPLOYEE_ID , FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEES WHERE SALARY > 15000;

Poderia ser feito assim :

```
SQL> RENAME EMPLOYEES TO EMP_RESTRICT;

SQL> CREATE VIEW EMPLOYEES as SELECT EMPLOYEE_ID , FIRST_NAME,
LAST_NAME, SALARY FROM EMP_RESTRICT WHERE SALARY > 15000;
```

⇒ Veja que o texto, sem alteração, vai executar outro comando (o AUTOTRACE é apenas demonstrativo, E as informações irrelevantes para o exemplo foram eliminadas do output) .....

```
SQL> set autotrace on;
SQL> SELECT EMPLOYEE_ID , FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEES;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                      SALARY
----------- -------------------- ------------------------- ----------
        100 Steven               King                           24000
        101 Neena                Kochhar                        17000
        102 Lex                  De Haan                        17000

Execution Plan
....
-----------------------------------------------------------------
| Id  | Operation          | Name         | Rows  | Bytes |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT   |              |     5 |   160 |
|*  1 |  TABLE ACCESS FULL| EMP_RESTRICT |     5 |   160 |
-----------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SALARY">15000)

....
SQL>
```

Restrições ao uso de Views

Mesmo o exemplo simples anterior já Evidencia algumas óbvias restrições ao uso de VIEWS como um mecanismo de alteração/refactoring de SQLs, tais como :

a. A view é um novo objeto sendo criado no modelo da Aplicação, e para que a substituição de uso ocorra normalmente é exigido que o nome da view seja o mesmo referenciado no texto SQL : alterações no modelo desse tipo por via de regra nem sempre são aceitáveis por parte do Fornecedor

b. A view necessariamente IMPLICA num novo sub-nível lógico no SQL, transformando o SQL num SQL com sub-queries : nem sempre isso é isento de efeitos colaterais, já que as sub-queries causadas pelo envolvimento de view podem ou não ser materializadas, ie, podem ou não ser resolvidas isoladamente

1) Introdução de HINTs/ Diretivas

Praticamente todos os RDBMS permitem que além dos comandos uma instrução SQL contenha também diretivas de comportamento para o interpretador SQL, que podem ser no formato de instruções diretas (HINTs), E além disso permitem a criação de elementos auxiliares à otimização de SQLs, como estatísticas e volumes. No RDBMS Oracle temos as seguintes possibilidades :

a) SQL Outlines : são um conjunto de HINTs que o DBA/Analista pode associar a um SQL específico ou a um grupo de SQL – recurso depreciado na versão 11g

b) SQL Profiles : além de HINTs esse objeto pode conter estatísticas de uso adicionais (principalmente volumes e distribuição) E modificações de sessão/ambiente. Os SQL Profiles criados podem ser “inseridos” nos SQLs em execução pelo próprio RDBMS, uma vez permissionado e ativado o recurso, manualmente ou através de Advisors próprios

c) SQL Plan Management : com este recurso o DBA/Analista irá capturar planos e tempos de resposta aceitáveis para os SQLs desejados (o chamado BASELINE) – após isso, se por aumento de volume ou qualquer motivo um plano inferior for gerado, não será usado, sendo removido na fase de escolha do CBO ou ‘ convertido’ ao plano original automaticamente através de hints ou de reparse.

Exemplo de Hinting Automático com Outlines

=> os privilégios necessários não são default, são concedidos manualmente :

```
SYSTEM:SQL> GRANT CREATE ANY OUTLINE TO SCOTT;
SYSTEM:SQL> GRANT EXECUTE_CATALOG_ROLE TO SCOTT;

=> o usuário permissionado vai criar e usar o Outline :

SCOTT:SQL> ALTER SESSION SET create_stored_outlines=TRUE;
SCOTT:SQL> CREATE OUTLINE emp_dept FOR CATEGORY scott_outlines
ON SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno =
d.deptno;

=> Confirmando que o Outline está presente :

SCOTT:SQL> SELECT name, category, sql_text FROM user_outlines WHERE
category = 'SCOTT_OUTLINES';

NAME                           CATEGORY
------------------------------ ------------------------------
SQL_TEXT
-------------------------------------------------------------
EMP_DEPT                       SCOTT_OUTLINES
SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno =
d.deptno
```

=> consultando os hints :

```
SCOTT:SQL> COLUMN hint FORMAT A50
SELECT node, stage, join_pos, hint FROM user_outline_hints WHERE name =
'EMP_DEPT';SCOTT:SQL>

      NODE      STAGE   JOIN_POS HINT
---------- ---------- ---------- --------------------------------------------
         1          1          0 USE_MERGE(@"SEL$1" "E"@"SEL$1")
         1          1          0 LEADING(@"SEL$1" "D"@"SEL$1" "E"@"SEL$1")
         1          1          2 FULL(@"SEL$1" "E"@"SEL$1")
         1          1          1 INDEX(@"SEL$1" "D"@"SEL$1" ("DEPT"."DEPTNO"))
         1          1          0 OUTLINE_LEAF(@"SEL$1")
         1          1          0 ALL_ROWS
         1          1          0 DB_VERSION('11.2.0.4')
         1          1          0 OPTIMIZER_FEATURES_ENABLE('11.2.0.4')
         1          1          0 IGNORE_OPTIM_EMBEDDED_HINTS

9 rows selected.
```

=> O otimizador incluiu as HINTs que julgou próprias e incorporou os settings de sessão ativos - posteriormente esse conjunto de hints pode também ser alterado/editado, acessando-se manualmente tabelas internas que compõem as views xxx_OUTLINE_nnn (ie, OL$HINTS e OL$xxx – é um dos poucos casos onde é viável usar tabelas internas) ou com a tool OUTLINE Editor (parte do OEM Console 9i), vide nota metalink/My Oracle Support “Editing Stored Outlines in Oracle9i - an example” (Doc ID 144194.1).

=> Associando com um SQL específico e habilitando o OUTLINE criado :

=> executo o SQL desejado para obter seu HASH value :

```
SCOTT:SQL> SELECT e.empno, e.ename, d.dname, e.job FROM emp e, dept d
WHERE e.deptno = d.deptno AND d.dname = 'SALES';

     EMPNO ENAME      DNAME          JOB
---------- ---------- -------------- ---------
      7521 WARD       SALES          SALESMAN
      7844 TURNER     SALES          SALESMAN
....
      7654 MARTIN     SALES          SALESMAN

SCOTT:SQL> SELECT hash_value, child_number, sql_text FROM v$sql WHERE
sql_text LIKE 'SELECT e.empno, e.ename, d.dname, e.job%';

HASH_VALUE CHILD_NUMBER
---------- ------------
SQL_TEXT
------------------------------------------------------------
3909283366             0
SELECT e.empno, e.ename, d.dname, e.job FROM emp e, dept d WHERE
e.deptno = d.deptno AND d.dname = 'SALES'
```

```
=> Associação a um hash value :

SCOTT:SQL> BEGIN
  DBMS_OUTLN.create_outline(
    hash_value    => 3909283366,
    child_number  => 0,
    category      => 'SCOTT_OUTLINES');
END;
/

=> re-executo o SQL para demonstrar o uso :

SCOTT:SQL> ALTER SESSION SET query_rewrite_enabled=TRUE;
SCOTT:SQL> ALTER SESSION SET use_stored_outlines=SCOTT_OUTLINES;

SCOTT:SQL> SELECT e.empno, e.ename, d.dname, e.job FROM emp e, dept d
WHERE e.deptno = d.deptno AND d.dname = 'SALES';


SCOTT:SQL> SELECT name, category, used FROM user_outlines;

NAME                           CATEGORY                       USED
------------------------------ ------------------------------ ------
SYS_OUTLINE_16041518454087604  SCOTT_OUTLINES                 USED
```

2) Filtros e Cláusulas WHERE Adicionais

Outra alteração comumente necessária a se aplicar em SQLs são cláusulas WHERE adicionais – isso pode servir para uma consição de Segurança adicional, quanto também pode influenciar na performance, principalmente no uso ou não de Índices – apenas colunas indexadas citadas no WHERE (ou nas cláusulas de JOIN) normalmente são analisadas pelo Otimizador.

Há diversas maneiras de se obter essa alteração sem acesso aos fontes / SQL , tais como VPD (Virtual Private Database), FGAC (Fine-Grained Access Control) e mesmo Views (numa implementação similar ao Exemplo anterior) - todos trabalham de maneira similar, adicionando uma cláusula WHERE em adição as informadas : as diferenças entre as maneiras citadas são mais no sentido de haver ou não uma função programática a ser validada, utilizar ou não Contextos, se o Predicado a ser aplicado no WHERE é dinâmico ou não, etc.

Exemplo de VPD :

⇒ Primeiro criamos uma Função a ser aplicada na Policy :

```
HR:SQL>CREATE OR REPLACE FUNCTION FUN_VPD_EMP(p_schema IN VARCHAR2,
p_table  IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
   RETURN 'salary < 10000';
END;
```

⇒ Sem policy :

```
HR:SQL>select count(*) from test_employees;

  COUNT(*)
----------
       107
```

⇒ Adicionando POLICY na tabela de teste em questão :

```
HR:SQL>exec DBMS_RLS.ADD_POLICY(OBJECT_SCHEMA=>
'HR',OBJECT_NAME=>'TEST_EMPLOYEES',POLICY_NAME=>'POL_TEST',
FUNCTION_SCHEMA=>'HR', POLICY_FUNCTION=>'FUN_VPD_EMP');
```

=> Função sendo aplicada para validar linhas no WHERE :

```
HR:SQL>select count(*) from test_employees;

  COUNT(*)
----------
        88
```

⇒ Evidência pelo trace sql*plus :

```
HR:SQL>set autotrace on
HR:SQL>select count(*) from test_employees;

  COUNT(*)
----------
        88

Execution Plan

----------------------------------------------------------
| Id  | Operation          | Name            | Rows  | Bytes |
----------------------------------------------------------
|   0 | SELECT STATEMENT   |                 |     1 |    13 |
|   1 |  SORT AGGREGATE    |                 |     1 |    13 |
|*  2 |   TABLE ACCESS FULL| TEST_EMPLOYEES |    88 |  1144 |
----------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("SALARY"<10000)
```

3) Substituição completa de um comando SQL por outro

Esta é a técnica final que apresentaremos, e é a mais potente, e portanto ao mesmo tempo a mais aplicável em casos de bugfixes pontuais em SQLs sem acesso ao fonte mas também a mais propensa a apresentar efeitos colaterais : basicamente iremos aqui nos aproveitar da capacidade de REESCRITA automática introduzida nas versões mais recentes do RDBMS Oracle com a finalidade de se aproveitar de objetos capazes de evitar acesso á tabelas - um desses objetos por exemplo são as VIEWS MATERIALIZADAS. O objetivo original da Oracle era dotar o otimizador da capacidade de re-escrever automaticamente um SQL cujo resultset pudesse ser atendido pela view materializada, TROCANDO no texto do SQL as referências às tabelas por referências à view materializada.

Originalmente, essa capacidade de SQL REWRITE era interna e inacessível aos usuários, mas nas versões mais recentes a Oracle publicou uma API de acesso a ela, na forma da package interna built-in DBMS_ADVANCED_REWRITE.

Exemplo de Utilização

⇒ As permissões necessárias não são default :

```
SYSDBA:SQL> GRANT execute ON sys.dbms_advanced_rewrite TO scott;
SYSDBA:SQL> GRANT create materialized view TO scott;
```

⇒ crio uma tabela A sem dados e uma tabela B com dados :

```
SCOTT:SQL>create table tableA (c1 VARCHAR2(10));
SCOTT:SQL>create table tableB (c1 VARCHAR2(10));
SCOTT:SQL>insert into tableB values('B!!');
SCOTT:SQL>commit;
```

⇒ indico a equivalência de re-escrita :

⇒
```
SCOTT:SQL>exec sys.dbms_advanced_rewrite.declare_rewrite_equivalence
  ( -
 name => 'EQUIV_TEST', -
 source_stmt => 'SELECT C1 FROM TABLEA', -
 destination_stmt => 'SELECT C1 FROM TABLEB', -
 validate => FALSE, -
 rewrite_mode => 'TEXT_MATCH' -
 );> > > > > >
```

⇒ habilito a re-escrita :

```
SCOTT:SQL>ALTER SESSION SET query_rewrite_integrity = trusted;
```

⇒ executo o SQL referenciando a tabela A, que não tem dados : TRANSPARENTEMENTE, sem alterar o fonte, Automaticamente o texto recebido pelo RDBMS passou a acessar B :

```
SCOTT:SQL> SELECT C1 FROM TABLEA;

C1
----------
B!!

SCOTT:SQL>
```

Obviamente :

- A alteração aqui foi Trivial, mas não há limitações nesse sentido, poderia ter sido um bugfix muito mais Profundo

- Esse recurso é relativamente recente, então ainda pode apresentar instabilidades

- Demanda que o SQL seja conhecido, o que na prática Inviabiliza ambientes com SQLs dinamicamente gerados

Perguntas e Dúvidas ????