SUNGARD 金仕达

Oracle SQL 规范

组织标准软件过程文档

文档标识

文档名称	Oracle SQL 规范 V1.0
状况	☐ 草案 ☐ 评审过的 ☐ 更新过的 ☑ 定为基线的
模板版本号	<>

文档修订历史

版本	日期	描述	文档所有者
V0.1	2007/6/19	起草	林强、王凌洁
V1.0	2007/6/29	评审	杨疆湖、林强、汤成

此版本文档的正式核准

姓名	签字	日期

分发控制

副本	接受人	机构

目 录

1. SQL 书写规范

1) sql 语句的所有表名、字段名全部小写，系统保留字、内置函数名、sql 保留字大写。

2) 连接符 or、in、and、以及＝、<=、>=等前后加上一个空格。

3) 对较为复杂的 sql 语句加上注释，说明算法、功能。

2. PLSQL 注释规范

1) 应对不易理解的分支条件表达式加注释；

2) 对重要的计算应说明其功能；

3) 过长的函数实现，应将其语句按实现的功能分段加以概括性说明；

4) 每条 SQL 语句均应有注释说明（表名、字段名）。

5) 常量及变量注释时，应注释被保存值的含义(必须)，合法取值的范围(可选)

3. 使用 ORACLE 共享 SQL 语句机制

✧ 要求 SQL 书写规范： sql 语句的所有表名、字段名全部小写，系统保留字、内置函数名、sql 保留字大写。

因共享的语句必须满足三个条件:

A. 字符级的比较:
当前被执行的语句和共享池中的语句必须完全相同.
例如: SELECT * FROM EMP;
SELECT * from EMP;
Select * From Emp;

B. 两个语句所指的对象必须完全相同:

C. 两个 SQL 语句中必须使用相同的名字的绑定变量(bind variables)
例如： select pin , name from people where pin = :blk1.pin;
select pin , name from people where pin = :blk1.pin;

4. 选择最有效率的表名顺序

✧ ORACLE 的解析器按照从右到左的顺序处理 FROM 子句中的表名,因此 FROM 子句中写在最后的表(基础表 driving table)将被最先处理例如:

表 TAB1 16,384 条记录

表 TAB2 10 条记录

A. **选择 TAB2 作为基础表 (最好的方法)**

select count(*) from tab1,tab2 执行时间 0.96 秒

B. **选择 TAB2 作为基础表 (不佳的方法)**

select count(*) from tab2,tab1 执行时间 26.09 秒

✧ 如果有 3 个以上的表连接查询, 那就需要选择交叉表(intersection table)作为基础表, 交叉表是指那个被其他表所引用的表.

例如:

EMP 表描述了 LOCATION 表和 CATEGORY 表的交集.

A. SELECT * FROM LOCATION L , CATEGORY C, EMP E
WHERE E.EMP_NO BETWEEN 1000 AND 2000
AND E.CAT_NO = C.CAT_NO AND E.LOCN = L.LOCN

B. 将比下列 SQL 更有效率
SELECT * FROM EMP E , LOCATION L , CATEGORY C
WHERE E.CAT_NO = C.CAT_NO AND E.LOCN = L.LOCN
AND E.EMP_NO BETWEEN 1000 AND 2000

5.要注意 WHERE 子句中的连接顺序

✧ ORACLE 采用自下而上的顺序解析 WHERE 子句,根据这个原理,表之间的连接必须写在其他 WHERE 条件之前, 那些可以过滤掉最大数量记录的条件必须写在 WHERE 子句的末尾。

A. **(高效,执行时间 10.6 秒)**

SELECT … FROM EMP E WHERE 25 < (SELECT COUNT(*) FROM EMP WHERE MGR=E.EMPNO)
AND SAL > 50000
AND JOB = 'MANAGER';

B. **例如: (低效,执行时间 156.3 秒)**

SELECT … FROM EMP E WHERE SAL > 50000
AND JOB = 'MANAGER' AND 25 < (SELECT COUNT(*) FROM EMP WHERE MGR=E.EMPNO);

6.SELECT 子句中避免使用 *

✧ 当你想在 SELECT 子句中列出所有的 COLUMN 时,使用动态 SQL 列引用 '*' 是一个方便的方法.不幸的是,这是一个非常低效的方法. 实际上,ORACLE 在解析的过程中, 会将'*' 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费

更多的时间.

7.尽量减少访问数据库的次数

✧ 有三种方法可以检索出雇员号等于 0342 或 0291 的职员.

A. **方法 1（高效）**

```
SELECT A.EMP_NAME , A.SALARY , A.GRADE,
B.EMP_NAME , B.SALARY , B.GRADE FROM EMP A,EMP B
WHERE A.EMP_NO = 342 AND B.EMP_NO = 291;
```

B. **方法 2（最低效）**

```
SELECT EMP_NAME , SALARY , GRADE FROM EMP
WHERE EMP_NO = 342;
SELECT EMP_NAME , SALARY , GRADE FROM EMP
WHERE EMP_NO = 291;
```

8.使用 DECODE 函数来减少处理时间

✧ 使用 DECODE 函数可以避免重复扫描相同记录或重复连接相同的表.

例如:

```
SELECT COUNT(*), SUM(SAL) FROM EMP WHERE DEPT_NO = 0020 AND ENAME LIKE
'SMITH%';
SELECT COUNT(*), SUM(SAL) FROM EMP WHERE DEPT_NO = 0030 AND ENAME LIKE
'SMITH%';
```

你可以用 DECODE 函数高效地得到相同结果

```
SELECT COUNT(DECODE(DEPT_NO,0020,'X',NULL))
D0020_COUNT,COUNT(DECODE(DEPT_NO,0030,'X',NULL)) D0030_COUNT,
SUM(DECODE(DEPT_NO,0020,SAL,NULL)) D0020_SAL,
SUM(DECODE(DEPT_NO,0030,SAL,NULL)) D0030_SAL FROM EMP WHERE ENAME LIKE
'SMITH%';
```

9.用 ROWID 删除重复记录

✧ 最高效的删除重复记录方法 (因为使用了 ROWID)

```
DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP_NO
= E.EMP_NO);
```

10.考虑用 TRUNCATE 替代 DELETE

当删除表中的记录时,在通常情况下, 回滚段(rollback segments) 用来存放可以被恢复的信息. 如果你没有 COMMIT 事务,ORACLE 会将数据恢复到删除之前的状态(准确地说是 恢复到执行删除命令之前的状况)

而当运用 TRUNCATE 时, 回滚段不再存放任何可被恢复的信息.当命令运行后,数据不能被恢复.因此很少的资源被调用,执行时间也会很短.

11.用 Where 子句替换 HAVING 子句

✧ 避免使用HAVING子句, HAVING 只会在检索出所有记录之后才对结果集进行过滤. 这个处理需要排序,总计等操作. 如果能通过 WHERE 子句限制记录的数目,那就能减少这方面的开销.

A. **高效**

```
SELECT REGION, AVG(LOG_SIZE) FROM LOCATION
WHERE REGION REGION != 'SYDNEY'
AND REGION != 'PERTH'
GROUP BY REGION
```

B. **低效:**

```
SELECT REGION, AVG(LOG_SIZE) FROM LOCATION
GROUP BY REGION
HAVING REGION REGION != 'SYDNEY'
AND REGION != 'PERTH'
```

12.使用正确的方法来减少对表的操作

在含有子查询的 SQL 语句中,要特别注意减少对表的查询. 使用多列的比较，更新等.

✧ **Update 多个 Column 例子:**

A. **高效:**

```
UPDATE EMP SET (EMP_CAT, SAL_RANGE) = (SELECT MAX(CATEGORY) , MAX(SAL_RANGE)
FROM EMP_CATEGORIES) WHERE EMP_DEPT = 0020;
```

B. **低效:**

```
UPDATE EMP SET EMP_CAT = (SELECT MAX(CATEGORY) FROM EMP_CATEGORIES), SAL_RANGE
= (SELECT MAX(SAL_RANGE) FROM EMP_CATEGORIES) WHERE EMP_DEPT = 0020;
```

✧ **比较 多个 Column 例子:**

A. **高效**

SELECT TAB_NAME FROM TABLES WHERE (TAB_NAME,DB_VER) = (SELECT TAB_NAME,DB_VER) FROM TAB_COLUMNS WHERE VERSION = 604)

B. 低效:

SELECT TAB_NAME FROM TABLES WHERE TAB_NAME = (SELECT TAB_NAME FROM TAB_COLUMNS WHERE VERSION = 604) AND DB_VER= (SELECT DB_VER FROM TAB_COLUMNS WHERE VERSION = 604)

13.使用表的别名(Alias)

当在 SQL 语句中连接多个表时, 请使用表的别名并把别名前缀于每个 Column 上.这样一来,就可以减少解析的时间并减少那些由 Column 歧义引起的语法错误.

14.用表连接替换 EXISTS

✧ 通常来说 , 采用表连接的方式比 EXISTS 更有效率

A. 高效

SELECT ENAME FROM DEPT D, EMP E WHERE E.DEPT_NO = D.DEPT_NO AND DEPT_CAT = 'A' ;

B. 低效

SELECT ENAME FROM EMP E WHERE EXISTS (SELECT * FROM DEPT WHERE DEPT_NO = E.DEPT_NO AND DEPT_CAT = 'A');

15.用 EXISTS 替换 DISTINCT

✧ 当提交一个包含一对多表信息(比如部门表和雇员表)的查询时,避免在 SELECT 子句中使用 DISTINCT. 一般可以考虑用 EXIST 替换 例如:

A. 高效:

SELECT DEPT_NO,DEPT_NAME FROM DEPT D WHERE EXISTS (SELECT * FROM EMP E WHERE E.DEPT_NO = D.DEPT_NO);

B. 低效:

SELECT DISTINCT DEPT_NO,DEPT_NAME FROM DEPT D,EMP E WHERE D.DEPT_NO = E.DEPT_NO

16. ORACLE 使用索引规则

当 SQL 语句的执行路径可以使用分布在多个表上的多个索引时, ORACLE 会同时使用多

个索引并在运行时对它们的记录进行合并，检索出仅对全部索引有效的记录. 在 ORACLE 选择执行路径时,唯一性索引的等级高于非唯一性索引. 然而这个规则只有 当 WHERE 子句中索引列和常量比较才有效.如果索引列和其他表的索引列相比较. 这种子句在优化器中的等级是非常低的. 如果不同表中两个想同等级的索引被引用，FROM 子句中最后的表的索引将有最高的优先级. 如果相同表中两个想同等级的索引将被引用，WHERE 子句中最先被引用的索引将有最高的优先级.

举例:

DEPTNO 上有一个非唯一性索引,EMP_CAT 也有一个非唯一性索引.

SELECT ENAME, FROM EMP WHERE DEPT_NO = 20 AND EMP_CAT = 'A';

这里,DEPTNO 索引将被最先检索,然后同 EMP_CAT 索引检索出的记录进行合并. 执行路径如下:

TABLE ACCESS BY ROWID ON EMP

AND-EQUAL

INDEX RANGE SCAN ON DEPT_IDX

INDEX RANGE SCAN ON CAT_IDX

16.1 等式比较和范围比较

✧ 当 WHERE 子句中有索引列，ORACLE 不能合并它们,ORACLE 将用范围比较.

举例:

DEPTNO 上有一个非唯一性索引,EMP_CAT 也有一个非唯一性索引.

SELECT ENAME FROM EMP WHERE DEPTNO > 20

AND EMP_CAT = 'A';

这里只有 EMP_CAT 索引被用到,然后所有的记录将逐条与 DEPTNO 条件进行比较. 执行路径如下:

TABLE ACCESS BY ROWID ON EMP

INDEX RANGE SCAN ON CAT_IDX

16.2 避免在索引列上使用 NOT

✧ 通常， 我们要避免在索引列上使用 NOT，NOT 会产生在和在索引列上使用函数相同的影响. 当 ORACLE"遇到"NOT,他就会停止使用索引转而执行全表扫描.

举例:

A. **高效:(这里,使用了索引)**

SELECT … FROM DEPT WHERE DEPT_CODE > 0;

B. **低效:(这里,不使用索引)**

```
SELECT … FROM DEPT WHERE DEPT_CODE NOT = 0;
```

16.3 用>=替代>

如果 DEPTNO 上有一个索引,
两者的区别在于, 前者 DBMS 将直接跳到第一个 DEPT 等于 4 的记录而后者将首先定位到 DEPTNO=3 的记录并且向前扫描到第一个 DEPT 大于 3 的记录.

A. **高效:**

```
SELECT * FROM EMP WHERE DEPTNO >=4
```

B. **低效:**

```
SELECT * FROM EMP WHERE DEPTNO >3
```

16.4 用 UNION 替换 OR (适用于索引列)

通常情况下, 用 UNION 替换 WHERE 子句中的 OR 将会起到较好的效果. 对索引列使用 OR 将造成全表扫描. 注意, 以上规则只针对多个索引列有效. 如果有 column 没有被索引, 查询效率可能会因为你没有选择 OR 而降低。
在下面的例子中, LOC_ID 和 REGION 上都建有索引.

A. **高效:**

```
SELECT LOC_ID , LOC_DESC , REGION FROM LOCATION
WHERE LOC_ID = 10
UNION
SELECT LOC_ID , LOC_DESC , REGION FROM LOCATION
WHERE REGION = “MELBOURNE”
```

B. **低效:**

```
SELECT LOC_ID , LOC_DESC , REGION FROM LOCATION
WHERE LOC_ID = 10 OR REGION = “MELBOURNE”
```

如果你坚持要用 OR, 那就需要返回记录最少的索引列写在最前面.
注意: WHERE KEY1 = 10 (返回最少记录) OR KEY2 = 20 (返回最多记录) ORACLE 内部将以上转换为 WHERE KEY1 = 10 AND
((NOT KEY1 = 10) AND KEY2 = 20)

16.5 避免在索引列上使用 IS NULL 和 IS NOT NULL

避免在索引中使用任何可以为空的列，ORACLE 将无法使用该索引 . 对于单列索引，如果列包含空值，索引中将不存在此记录. 对于复合索引，如果每个列都为空，索引中同样不存在此记录. 如果至少有一个列不为空，则记录存在于索引中.
举例:
如果唯一性索引建立在表的 A 列和 B 列上, 并且表中存在一条记录的 A,B 值为(123,null) , ORACLE 将不接受下一条具有相同 A,B 值 (123,null) 的记录(插入). 然而如果
所有的索引列都为空,ORACLE 将认为整个键值为空而空不等于空. 因此你可以插入 1000 条具有相同键值的记录,当然它们都是空!
因为空值不存在于索引列中,所以 WHERE 子句中对索引列进行空值比较将使 ORACLE 停用该索

引.
举例:
低效: (索引失效)
SELECT … FROM DEPARTMENT WHERE DEPT_CODE IS NOT NULL;
高效: (索引有效)
SELECT … FROM DEPARTMENT WHERE DEPT_CODE >=0;

16.6 总是使用复合索引的第一个列

✧ 如果索引是建立在多个列上, 只有在它的第一个列(leading column)被 where 子句引用时,优化器才会选择使用该索引.

这也是一条简单而重要的规则. 见以下实例.

SQL> create table multiindexusage (inda number , indb number , descr varchar2(10));

Table created.

✧

SQL> create index multindex on multiindexusage(inda,indb);

Index created.

✧

SQL> select * from multiindexusage where inda = 1;

16.7 用 UNION-ALL 替换 UNION (如果有可能的话)

当 SQL 语句需要 UNION 两个查询结果集合时,这两个结果集合会以 UNION-ALL 的方式被合并, 然后在输出最终结果前进行排序. 如果用 UNION ALL 替代 UNION, 这样排序就不是必要了. 效率就会因此得到提高.
举例:

A. **低效:**

```
SELECT ACCT_NUM, BALANCE_AMT FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION
SELECT ACCT_NUM, BALANCE_AMT FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
```

B. **高效:**

```
SELECT ACCT_NUM, BALANCE_AMT FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION ALL
SELECT ACCT_NUM, BALANCE_AMT FROM DEBIT_TRANSACTIONS
```

```
WHERE TRAN_DATE = '31-DEC-95'
```

16.8 避免改变索引列的类型

当比较不同数据类型的数据时, ORACLE 自动对列进行简单的类型转换.
假设 EMPNO 是一个数值类型的索引列.
SELECT … FROM EMP WHERE EMPNO = '123'
实际上,经过 ORACLE 类型转换, 语句转化为:
SELECT … FROM EMP WHERE EMPNO = TO_NUMBER('123')
幸运的是,类型转换没有发生在索引列上,索引的用途没有被改变.
现在,假设 EMP_TYPE 是一个字符类型的索引列.
SELECT … FROM EMP WHERE EMP_TYPE = 123
这个语句被 ORACLE 转换为:

```
SELECT … FROM EMP
WHERE TO_NUMBER(EMP_TYPE)=123
```

16.9 需要当心的 WHERE 子句

某些 SELECT 语句中的 WHERE 子句不使用索引. 这里有一些例子.
在下面的例子里, '!=' 将不使用索引. 记住, 索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中.
不使用索引:

```
SELECT ACCOUNT_NAME FROM TRANSACTION
WHERE AMOUNT !=0;
```

使用索引:

```
SELECT ACCOUNT_NAME FROM TRANSACTION WHERE AMOUNT >0;
```