上海复旦金仕达计算机有限公司

文档名称	SQL 编程规范 V1.0
文档 ID	3SPE-GSqlCodingStd-V1.0
读者	开发人员

文档修订历史

版本	修改理由	日期	编者	TOSSP 版本
1.0	新增	2006-10-23	林强	V1.0

ID : 3SPE-GSqlCoding Std-V1.00

SQL 编码规范 V1.0

前言

1 注释

注释风格：注释单独成行、放在语句前面。

- 可采用单行/多行注释。（-- 或 /* */ 方式）；
- 对较为复杂的 SQL 语句加上注释；
- 应对不易理解的分支条件表达式加注释；
- 对重要的计算应说明其功能；
- 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；

2 命名和格式约定

所有表名、字段名遵照数据字典的定义，系统保留字、内置函数名、PL/SQL 保留字大写。

2.1、 变量名小写;

普通变量名使用“v_”开头；

光标变量以“cur_”开头；

输入参数以“i_”开头；

输出参数以“o_”开头；

输入输出参数以“io_”

2.2、 PL/SQL 中的子程序命名，使用大写;

函数以“FN_”开始；

过程以“SP_“开始；

触发器以大写的“TR_B[A]dmltype_”；--b: before a: after dmltype: insert , update , delete

类型声明以大写的“TP_”开始；

包以大写的“PKG_”开始；

表以大写的”T_”开始；

sequence 以大写的”SEQ_”开始；

view 以大写的”V_”开始；

index 以大写的”INX_”开始；

3 缩进与断行风格

3.1、一行有多列，超过 80 个字符时，基于列对齐原则，采用下行缩进 。

3.2、WHERE 子句书写时，每个条件占一行，语句另起一行时，以保留字或者连接符开始，连接符右对齐。例如：语句另起一行时以保留字开始 （ 如 and ） 左对齐可以和 where 对齐。

```
                                where f1 = 1
                                and    f2 = 2
                                or     f3 = 3
```

3.5、 连接符 OR、IN、AND、以及＝、<=、>=等前后加上一个空格。

3.6、 INSERT 语句，尽量书写字段，字段可 5 个或 6 个一组。例如：

```
INSERT INTO T_MemberCapital ( memberID,accountType,available,lastBalance,lastOccupied,
                              todayBalance,todayOccupied,profit,todayIn,todayOut,
                              payment,received,miscFee,frozen,basefund )
      SELECT memberID,accountType,available,todayBalance,todayOccupied,
             0,0,0,0,0,0,0,0,paltFrozen+Frozen,basefund
      FROM   T_HisMemberCapital
      WHERE effectDate=i_lastday;
        AND   F1=1;
        OR    F3=3;
```

4 编写规范

4.1、**共享的SQL语句必须满足三个条件**（共享SQL语句，SQL分析器不用对SQL语句重新分析产生执行计划，系统响应时间大大减少）。

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。

```
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) 。

```
SELECT pin , name FROM people WHERE pin = :blk1.pin;
SELECT pin , name FROM people WHERE pin = :blk1.pin;
```

4.2、 一行不要超过80个字符，在1024*768分辨率下只能显示这么110字符。

4.3、 多表连接时要使用表的别名来引用，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。使用大写的表名作为别名。

4.4、 不要用SELECT ‘*’, 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，这是一个非常低效的方法. 实际上,数据库在解析的过程中, 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。

4.5、 不要依赖任何隐式的数据类型转换。例如，不能为数字变量赋予字符值，而假定 SQL 会进行必要的转换。相反，在为变量赋值或比较值之前，应使用适当的 CONVERT 函数使数据类型相匹配。 隐式转换可能带来的问题：性能和版本升级的问题.

两个隐式转换的例子：

```
declare
   n number ;
begin
   n :=n +10 ; --存在隐式转换
   n :=n +10.0 ; --没有类型转换
end ;

declare
   c char(2) ;
begin
   c:=24 ;    --隐式转换
   c:=’23’; --没有隐式转换
end ;
```

4.6、 不要将空的变量值直接与比较运算符（符号）比较。如果变量可能为空，应使用 IS NULL 或 IS NOT NULL 进行比较，或者使用 ISNULL 函数。

4.7、 当删除全表记录时，用TRUNCATE替代DELETE，在通常情况下，回滚段(rollback segments) 用来存放可以被恢复的信息，但占用大量的系统资源。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。需要注意，truncate属于DDL语句，在该语句执行前会将之前的事务进行提交。

DB2不支持truncate语句，但提供二种方法来解决上面的问题:

（1） 在建表时加选项not logged initially ，当清空表时通过alter table [name] activate not logged initially with empty table

（2）首先在操作系统上建一个空文件empty.del，接着通过import命令import from empty.del of del replace into [table_name]来清空表中的数据并重组表空间。

4.8、 WHERE子句中，如果需要使用特定列上的索引，则需避免在索引列上使用计算，函数调用，改变索引列的类型等操作。

```
低效：
    SELECT … FROM DEPT WHERE SAL * 12 > 25000;
高效
    SELECT …FROM DEPT WHERE SAL > 25000/12;
```

4.9、 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。 当遇到”NOT,他就会停止使用索引转而执行全表扫描。

```
低效: (这里,不使用索引)
    SELECT … FROM DEPT WHERE DEPT_CODE NOT = 0;
高效: (这里,使用了索引)
    SELECT … FROM DEPT WHERE DEPT_CODE > 0;
```

需要注意的是,在某些时候, ORACLE优化器会自动将NOT转化成相对应的关系操作符. NOT > to <= NOT >= to < NOT < to >= NOT <= to >

4.10、 避免改变索引列的类型：当比较不同数据类型的数据时，数据库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'
```

经过类型转换，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
```

这个语句被转换为:

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

对索引列改变了，此次查询用的是全表扫描。

4.11、 严格使用ORDER BY ，ORDER BY 子句只在两种严格的条件下使用索引，ORDER BY中所有的列必须包含在相同的索引中并保持在索引中的排列顺序. ORDER BY中所有的列必须定义为非空。

4.12、 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 ( aa number , bb number , descr varchar2(10));
Table created.
SQL> create index multindex on multiindexusage(aa,bb);
Index created.
SQL> set autotrace traceonly
SQL> select * from multiindexusage where aa = 1;  （用到索引）
SQL> select * from multiindexusage where bb = 1;  （没用到索引）
```

4.13、 如系统不需要唯一的记录，用UNION-ALL 替换UNION ，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。

4.14、 用For update 共享更新封锁是对一个表的一行或多行进行封锁，因而也称作行级封锁。表级封锁虽然保证了数据的一致性，但却减弱了操作数据的并行性。如没有加关键字[NOWAIT]常发生死锁。

（1）、执行如下的SQL封锁语句，以显示的方式获得：

LOCK TABLE <表名>[,<表名>]....

IN SHARE UPDATE MODE [NOWAIT]

（2）、用如下的SELECT ...FOR UPDATE语句获得：

SELECT <列名>[,<列名>]...

FROM <表名> WHERE <条件>

FOR UPDATE OF <列名>[,<列名>].....[NOWAIT]

4.15、 不必要的排序操作

当在 SQL STATEMENT 中包含下面的语句中，会引起系统进行排序操作。排序是非常

昂贵的操作，如果能够避免就避免了吧。

ORACLE Clause	内部操作
ORDER BY	SORT ORDER BY
DISTINCT,MINUS,INTERSECT,UNION 或者其他需要唯一值的情况	SORT UNIQUE
MIN,MAX,COUNT,SUM	SORT AGGREGATE
GROUP BY	SORT GROUP BY
Queries involving tables using sort merge joins	SORT JOIN