# SQL四条最基本的语句

　　 熟练掌握SQL是数据库用户的宝贵财富。在本文中，我们将引导你掌握四条最基本的数据操作语句—SQL的核心功能—来依次介绍比较操作符、选择断言以及三值逻辑。当你完成这些学习后，显然你已经开始算是精通SQL了。

　　在我们开始之前，先使用CREATE TABLE语句来创建一个表（如图1所示）。DDL语句对数据库对象如表、列和视进行定义。它们并不对表中的行进行处理，这是因为DDL语句并不处理数据库中实际的数据。这些工作由另一类SQL语句—数据操作语言（DML）语句进行处理。

　　SQL中有四种基本的DML操作：INSERT，SELECT，UPDATE和DELETE。由于这是大多数SQL用户经常用到的，我们有必要在此对它们进行一一说明。在图1中我们给出了一个名为EMPLOYEES的表。其中的每一行对应一个特定的雇员记录。请熟悉这张表，我们在后面的例子中将要用到它。

　　INSERT语句

　　用户可以用INSERT语句将一行记录插入到指定的一个表中。例如，要将雇员John Smith的记录插入到本例的表中，可以使用如下语句：

　　INSERT INTO EMPLOYEES VALUES

　　 ('Smith','John','1980-06-10',

　　 'Los Angles',16,45000);

　　通过这样的INSERT语句，系统将试着将这些值填入到相应的列中。这些列按照我们创建表时定义的顺序排列。在本例中，第一个值“Smith”将填到第一个列LAST\_NAME中；第二个值“John”将填到第二列FIRST\_NAME中……以此类推。

　　我们说过系统会“试着”将值填入，除了执行规则之外它还要进行类型检查。如果类型不符（如将一个字符串填入到类型为数字的列中），系统将拒绝这一次操作并返回一个错误信息。

　　如果SQL拒绝了你所填入的一列值，语句中其他各列的值也不会填入。这是因为SQL提供对事务的支持。一次事务将数据库从一种一致性转移到另一种一致性。如果事务的某一部分失败，则整个事务都会失败，系统将会被恢复（或称之为回退）到此事务之前的状态。

　　 回到原来的INSERT的例子，请注意所有的整形十进制数都不需要用单引号引起来，而字符串和日期类型的值都要用单引号来区别。为了增加可读性而在数字间插入逗号将会引起错误。记住，在SQL中逗号是元素的分隔符。

　　同样要注意输入文字值时要使用单引号。双引号用来封装限界标识符。

　　对于日期类型，我们必须使用SQL标准日期格式（yyyy-mm-dd），但是在系统中可以进行定义，以接受其他的格式。当然，2000年临近，请你最好还是使用四位来表示年份。

　　既然你已经理解了INSERT语句是怎样工作的了，让我们转到EMPLOYEES表中的其他部分：

　　INSERT INTO EMPLOYEES VALUES

　　 ('Bunyan','Paul','1970-07-04',

　　 'Boston',12,70000);

　　INSERT INTO EMPLOYEES VALUES

　　 ('John','Adams','1992-01-21',

　　 'Boston',20,100000);

　　INSERT INTO EMPLOYEES VALUES

　　 ('Smith','Pocahontas','1976-04-06',

　　 'Los Angles',12,100000);

　　INSERT INTO EMPLOYEES VALUES

　　 ('Smith','Bessie','1940-05-02',

　　 'Boston',5,200000);

　　INSERT INTO EMPLOYEES VALUES

　　 ('Jones','Davy','1970-10-10',

　　 'Boston',8,45000);

　　INSERT INTO EMPLOYEES VALUES

　　 ('Jones','Indiana','1992-02-01',

　　 'Chicago',NULL,NULL);

　　在最后一项中，我们不知道Jones先生的工薪级别和年薪，所以我们输入NULL（不要引号）。NULL是SQL中的一种特殊情况，我们以后将进行详细的讨论。现在我们只需认为NULL表示一种未知的值。

　　有时，像我们刚才所讨论的情况，我们可能希望对某一些而不是全部的列进行赋值。除了对要省略的列输入NULL外，还可以采用另外一种INSERT语句，如下：

　　INSERT INTO EMPLOYEES(

　　 FIRST\_NAME, LAST\_NAME,

　　 HIRE\_DATE, BRANCH\_OFFICE)

　　VALUE(

　　 'Indiana','Jones',

　　 '1992-02-01','Indianapolis');

　　这样，我们先在表名之后列出一系列列名。未列出的列中将自动填入缺省值，如果没有设置缺省值则填入NULL。请注意我们改变了列的顺序，而值的顺序要对应新的列的顺序。如果该语句中省略了FIRST\_NAME和LAST\_NAME项（这两项规定不能为空），SQL操作将失败。

　　让我们来看一看上述INSERT语句的语法图：

　　INSERT INTO table

　　 [(column { ,column})]

　　VALUES

　　 (columnvalue [{,columnvalue}]);

　　和前一篇文章中一样，我们用方括号来表示可选项，大括号表示可以重复任意次数的项（不能在实际的SQL语句中使用这些特殊字符）。VALUE子句和可选的列名列表中必须使用圆括号。

　　SELECT语句

　　SELECT语句可以从一个或多个表中选取特定的行和列。因为查询和检索数据是数据库管理中最重要的功能，所以SELECT语句在SQL中是工作量最大的部分。实际上，仅仅是访问数据库来分析数据并生成报表的人可以对其他SQL语句一窍不通。

　　SELECT语句的结果通常是生成另外一个表。在执行过程中系统根据用户的标准从数据库中选出匹配的行和列，并将结果放到临时的表中。在直接 SQL（direct SQL）中，它将结果显示在终端的显示屏上，或者将结果送到打印机或文件中。也可以结合其他SQL语句来将结果放到一个已知名称的表中。

　　SELECT语句功能强大。虽然表面上看来它只用来完成本文第一部分中提到的关系代数运算“选择”（或称“限制”），但实际上它也可以完成其他两种关系运算—“投影”和“连接”，SELECT语句还可以完成聚合计算并对数据进行排序。

　　SELECT语句最简单的语法如下：

　　SELECT columns FROM tables;

　　当我们以这种形式执行一条SELECT语句时，系统返回由所选择的列以及用户选择的表中所有指定的行组成的一个结果表。这就是实现关系投影运算的一个形式。

　　让我们看一下使用图1中EMPLOYEES表的一些例子（这个表是我们以后所有SELECT语句实例都要使用的。而我们在图2和图3中给出了查询的实际结果。我们将在其他的例子中使用这些结果）。

　　假设你想查看雇员工作部门的列表。那下面就是你所需要编写的SQL查询：

　　SELECT BRANCH\_OFFICE FROM EMPLOYEES;

　　以上SELECT语句的执行将产生如图2中表2所示的结果。

　　由于我们在SELECT语句中只指定了一个列，所以我们的结果表中也只有一个列。注意结果表中具有重复的行，这是因为有多个雇员在同一部门工作（记住SQL从所选的所有行中将值返回）。要消除结果中的重复行，只要在SELECT语句中加上DISTINCT子句：

　　SELECT DISTINCT BRANCH\_OFFICE

　　FROM EMPLOYEES;

　　这次查询的结果如表3所示。

　　现在已经消除了重复的行，但结果并不是按照顺序排列的。如果你希望以字母表顺序将结果列出又该怎么做呢？只要使用ORDER BY子句就可以按照升序或降序来排列结果：

　　SELECT DISTINCT BRANCH\_OFFICE

　　FROM EMPLOYEES

　　ORDER BY BRANCH\_OFFICE ASC;

　　这一查询的结果如表4所示。请注意在ORDER BY之后是如何放置列名BRANCH \_OFFICE的，这就是我们想要对其进行排序的列。为什么即使是结果表中只有一个列时我们也必须指出列名呢？这是因为我们还能够按照表中其他列进行排序，即使它们并不显示出来。列名BRANCH\_ OFFICE之后的关键字ASC表示按照升序排列。如果你希望以降序排列，那么可以用关键字DESC。

　　同样我们应该指出ORDER BY子句只将临时表中的结果进行排序；并不影响原来的表。

　　假设我们希望得到按部门排序并从工资最高的雇员到工资最低的雇员排列的列表。除了工资括号中的内容，我们还希望看到按照聘用时间从最近聘用的雇员开始列出的列表。以下是你将要用到的语句：

　　SELECT BRANCH\_OFFICE,FIRST\_NAME,

　　 LAST\_NAME,SALARY,HIRE\_DATE

　　FROM EMPLOYEES

　　ORDER BY SALARY DESC,

　　 HIRE\_DATE DESC;

　　这里我们进行了多列的选择和排序。排序的优先级由语句中的列名顺序所决定。SQL将先对列出的第一个列进行排序。如果在第一个列中出现了重复的行时，这些行将被按照第二列进行排序，如果在第二列中又出现了重复的行时，这些行又将被按照第三列进行排序……如此类推。这次查询的结果如表5所示。

　　将一个很长的表中的所有列名写出来是一件相当麻烦的事，所以SQL允许在选择表中所有的列时使用\*号：

　　SELECT \* FROM EMPLOYEES;

　　这次查询返回整个EMPLOYEES表，如表1所示。

　　 下面我们对开始时给出的SELECT语句的语法进行一下更新（竖直线表示一个可选项，允许在其中选择一项。）：

　　SELECT [DISTINCT]

　　 (column [{, columns}])| \*

　　FROM table [ {, table}]

　　[ORDER BY column [ASC] | DESC

　　 [ {, column [ASC] | DESC }]];

　　定义选择标准

　　在我们目前所介绍的SELECT语句中，我们对结果表中的列作出了选择但返回的是表中所有的行。让我们看一下如何对SELECT语句进行限制使得它只返回希望得到的行：

　　SELECT columns FROM tables [WHERE predicates];

　　WHERE子句对条件进行了设置，只有满足条件的行才被包括到结果表中。这些条件由断言（predicate）进行指定（断言指出了关于某件事情的一种可能的事实）。如果该断言对于某个给定的行成立，该行将被包括到结果表中，否则该行被忽略。在SQL语句中断言通常通过比较来表示。例如，假如你需要查询所有姓为Jones的职员，则可以使用以下SELECT语句：

　　SELECT \* FROM EMPLOYEES

　　WHERE LAST\_NAME = 'Jones';

　　LAST\_NAME = 'Jones'部分就是断言。在执行该语句时，SQL将每一行的LAST\_NAME列与“Jones”进行比较。如果某一职员的姓为“Jones”，即断言成立，该职员的信息将被包括到结果表中（见表6）。

　　使用最多的六种比较

　　我们上例中的断言包括一种基于“等值”的比较（LAST\_NAME = 'Jones'），但是SQL断言还可以包含其他几种类型的比较。其中最常用的为：

　　等于 =

　　不等于 <>

　　小于 <

　　大于 >

　　小于或等于 <=

　　大于或等于 >=

　　下面给出了不是基于等值比较的一个例子：

　　SELECT \* FROM EMPLOYEES

　　WHERE SALARY > 50000;

　　这一查询将返回年薪高于$50,000.00的职员（参见表7）。

　　逻辑连接符

　　有时我们需要定义一条不止一种断言的SELECT语句。举例来说，如果你仅仅想查看Davy Jones的信息的话，表6中的结果将是不正确的。为了进一步定义一个WHERE子句，用户可以使用逻辑连接符AND，OR和NOT。为了只得到职员 Davy Jones的记录，用户可以输入如下语句：

　　SELECT \* FROM EMPLOYEES

　　WHERE LAST\_NAME = 'Jones' AND FIRST\_NAME = 'Davy';

　　在本例中，我们通过逻辑连接符AND将两个断言连接起来。只有两个断言都满足时整个表达式才会满足。如果用户需要定义一个SELECT语句来使得当其中任何一项成立就满足条件时，可以使用OR连接符：

　　SELECT \* FROM EMPLOYEES

　　WHERE LAST\_NAME = 'Jones' OR LAST\_NAME = 'Smith';

　　有时定义一个断言的最好方法是通过相反的描述来说明。如果你想要查看除了Boston办事处的职员以外的其他所有职员的信息时，你可以进行如下的查询：

　　SELECT \* FROM EMPLOYEES

　　WHERE NOT(BRANCH\_OFFICE = 'Boston');

　　关键字NOT后面跟着用圆括号括起来的比较表达式。其结果是对结果取否定。如果某一职员所在部门的办事处在Boston，括号内的表达式返回true，但是NOT操作符将该值取反，所以该行将不被选中。

　　断言可以与其他的断言嵌套使用。为了保证它们以正确的顺序进行求值，可以用括号将它们括起来：

　　SELECT \* FROM EMPLOYEES

　　WHERE (LAST\_NAME = 'Jones'

　　AND FIRST\_NAME = 'Indiana')

　　OR (LAST\_NAME = 'Smith'

　　AND FIRST\_NAME = 'Bessie');

　　SQL沿用数学上标准的表达式求值的约定—圆括号内的表达式将最先进行求值，其他表达式将从左到右进行求值。

　　以上对逻辑连接符进行了说明，在对下面的内容进行说明之前，我们再一次对SELECT语句的语法进行更新：

　　SELECT [DISTINCT]

　　 (column [{, column } ] )| \*

　　FROM table [ { , table} ]

　　[ORDER BY column [ASC] | [DESC

　　[{ , column [ASC] | [DESC } ] ]

　　WHERE predicate [ { logical-connector predicate } ];

　　NULL和三值逻辑

　　在SQL中NULL是一个复杂的话题，关于NULL的详细描述更适合于在SQL的高级教程而不是现在的入门教程中进行介绍。但由于NULL需要进行特殊处理，并且你也很可能会遇到它，所以我们还是简略地进行一下说明。

　　首先，在断言中进行NULL判断时需要特殊的语法。例如，如果用户需要显示所有年薪未知的职员的全部信息，用户可以使用如下SELECT语句：

　　SELECT \* FROM EMPLOYEES

　　WHERE SALARY IS NULL;

　　相反，如果用户需要所有已知年薪数据的职员的信息，你可以使用以下语句：

　　SELECT \* FROM EMPLOYEES

　　WHERE SALARY IS NOT NULL;

　　请注意我们在列名之后使用了关键字IS NULL或IS NOT NULL，而不是标准的比较形式：COLUMN = NULL、COLUMN <> NULL或是逻辑操作符NOT（NULL）。

　　这种形式相当简单。但当你不明确地测试NULL（而它们确实存在）时，事情会变得很混乱。

　　例如，回过头来看我们图1中的EM-PLOYEES表，可以看到Indiana Jones的工薪等级或年薪值都是未知的。这两个列都包含NULL。可以想象运行如下的查询：

　　SELECT \* FROM EMPLOYEES

　　WHERE GRADE <= SALARY;

　　此时，Indiana Jones应该出现在结果表中。因为NULL都是相等的，所以可以想象它们是能够通过GRADE小于等于SALARY的检查的。这其实是一个毫无疑义的查询，但是并没有关系。SQL允许进行这样的比较，只要两个列都是数字类型的。然而，Indiana Jones并没有出现在查询的结果中，为什么？

　　正如我们早先提到过的，NULL表示未知的值（而不是象某些人所想象的那样表示一个为NULL的值）。对于SQL来说意味着这个值是未知的，而只要这个值为未知，就不能将其与其他值比较（即使其他值也是NULL）。所以SQL允许除了在true 和false之外还有第三种类型的真值，称之为“非确定”（unknown）值。

　　如果比较的两边都是NULL，整个断言就被认为是非确定的。将一个非确定断言取反或使用AND或OR与其他断言进行合并之后，其结果仍是非确定的。由于结果表中只包括断言值为“真”的行，所以NULL不可能满足该检查。从而需要使用特殊的操作符IS NULL和IS NOT NULL。

　　UPDATE语句

　　UPDATE语句允许用户在已知的表中对现有的行进行修改。

　　例如，我们刚刚发现Indiana Jones的等级为16，工资为$40,000.00，我们可以通过下面的SQL语句对数据库进行更新（并清除那些烦人的NULL）。

　　UPDATE EMPLOYEES

　　SET GRADE = 16, SALARY = 40000

　　WHERE FIRST\_NAME = 'Indiana'

　　 AND LAST\_NAME = 'Jones';

　　上面的例子说明了一个单行更新，但是UPDATE语句可以对多行进行操作。满足WHERE条件的所有行都将被更新。如果，你想让Boston办事处中的所有职员搬到New York，你可以使用如下语句：

　　UPDATE EMPLOYEES

　　SET BRANCH\_OFFICE = 'New York'

　　WHERE BRANCH\_OFFICE = 'Boston';

　　如果忽略WHERE子句,表中所有行中的部门值都将被更新为'New York'。

　　UPDATE语句的语法流图如下面所示：

　　UPDATE table

　　SET column = value [{, column = value}]

　　[ WHERE predicate [ { logical-connector predicate}]];

　　DELETE语句

　　DELETE语句用来删除已知表中的行。如同UPDATE语句中一样，所有满足WHERE子句中条件的行都将被删除。由于SQL中没有UNDO语句或是 “你确认删除吗？”之类的警告，在执行这条语句时千万要小心。如果决定取消Los Angeles办事处并解雇办事处的所有职员，这一卑鄙的工作可以由以下这条语句来实现：

　　DELETE FROM EMPLOYEES

　　WHERE BRANCH\_OFFICE = 'Los Angeles';

　　如同UPDATE语句中一样，省略WHERE子句将使得操作施加到表中所有的行。

　　DELETE语句的语法流图如下面所示：

　　DELETE FROM table

　　[WHERE predicate [ { logical-connector predicate} ] ];

# 数据库基本----SQL语句大全

## 一、基础

1、说明：创建数据库

Create DATABASE database-name

2、说明：删除数据库

drop database dbname

3、说明：备份sql server

--- 创建 备份数据的 device

USE master

EXEC sp\_addumpdevice disk, testBack, c:mssql7backupMyNwind\_1.dat

--- 开始 备份

BACKUP DATABASE pubs TO testBack

4、说明：创建新表

create table tabname(col1 type1 [not null] [primary key],col2 type2 [not null],..)

根据已有的表创建新表：

A：create table tab\_new like tab\_old (使用旧表创建新表)

B：create table tab\_new as select col1,col2… from tab\_old definition only

5、说明：删除新表

drop table tabname

6、说明：增加一个列

Alter table tabname add column col type

注：列增加后将不能删除。DB2中列加上后数据类型也不能改变，唯一能改变的是增加varchar类型的长度。

7、说明：添加主键： Alter table tabname add primary key(col)

说明：删除主键： Alter table tabname drop primary key(col)

8、说明：创建索引：create [unique] index idxname on tabname(col….)

删除索引：drop index idxname

注：索引是不可更改的，想更改必须删除重新建。

9、说明：创建视图：create view viewname as select statement

删除视图：drop view viewname

10、说明：几个简单的基本的sql语句

选择：select \* from table1 where 范围

插入：insert into table1(field1,field2) values(value1,value2)

删除：delete from table1 where 范围

更新：update table1 set field1=value1 where 范围

查找：select \* from table1 where field1 like ’%value1%’ ---like的语法很精妙，查资料!

排序：select \* from table1 order by field1,field2 [desc]

总数：select count as totalcount from table1

求和：select sum(field1) as sumvalue from table1

平均：select avg(field1) as avgvalue from table1

最大：select max(field1) as maxvalue from table1

最小：select min(field1) as minvalue from table1

11、说明：几个高级查询运算词

A： UNION 运算符

UNION 运算符通过组合其他两个结果表（例如 TABLE1 和 TABLE2）并消去表中任何重复行而派生出一个结果表。当 ALL 随 UNION 一起使用时（即 UNION ALL），不消除重复行。两种情况下，派生表的每一行不是来自 TABLE1 就是来自 TABLE2。

B： EXCEPT 运算符

EXCEPT 运算符通过包括所有在 TABLE1 中但不在 TABLE2 中的行并消除所有重复行而派生出一个结果表。当 ALL 随 EXCEPT 一起使用时 (EXCEPT ALL)，不消除重复行。

C： INTERSECT 运算符

INTERSECT 运算符通过只包括 TABLE1 和 TABLE2 中都有的行并消除所有重复行而派生出一个结果表。当 ALL 随 INTERSECT 一起使用时 (INTERSECT ALL)，不消除重复行。

注：使用运算词的几个查询结果行必须是一致的。

12、说明：使用外连接

A、left outer join：

左外连接（左连接）：结果集几包括连接表的匹配行，也包括左连接表的所有行。

SQL: select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

B：right outer join:

右外连接(右连接)：结果集既包括连接表的匹配连接行，也包括右连接表的所有行。

C：full outer join：

全外连接：不仅包括符号连接表的匹配行，还包括两个连接表中的所有记录。

## 二、提升

1、说明：复制表(只复制结构,源表名：a 新表名：b) (Access可用)

法一：select \* into b from a where 1<>1

法二：select top 0 \* into b from a

2、说明：拷贝表(拷贝数据,源表名：a 目标表名：b) (Access可用)

insert into b(a, b, c) select d,e,f from b;

3、说明：跨数据库之间表的拷贝(具体数据使用绝对路径) (Access可用)

insert into b(a, b, c) select d,e,f from b in ‘具体数据库’ where 条件

例子：..from b in "&Server.MapPath("."&"data.mdb" &" where..

4、说明：子查询(表名1：a 表名2：b)

select a,b,c from a where a IN (select d from b 或者: select a,b,c from a where a IN (1,2,3)

5、说明：显示文章、提交人和最后回复时间

select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b

6、说明：外连接查询(表名1：a 表名2：b)

select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

7、说明：在线视图查询(表名1：a

select \* from (Select a,b,c FROM a) T where t.a > 1;

8、说明：between的用法,between限制查询数据范围时包括了边界值,not between不包括

select \* from table1 where time between time1 and time2

select a,b,c, from table1 where a not between 数值1 and 数值2

9、说明：in 的使用方法

select \* from table1 where a [not] in (‘值1’,’值2’,’值4’,’值6’)

10、说明：两张关联表，删除主表中已经在副表中没有的信息

delete from table1 where not exists ( select \* from table2 where table1.field1=table2.field1

11、说明：四表联查问题：

select \* from a left inner join b on a.a=b.b right inner join c on a.a=c.c inner join d on a.a=d.d where .....

12、说明：日程安排提前五分钟提醒

SQL: select \* from 日程安排 where datediff(minute,f开始时间,getdate())>5

13、说明：一条sql 语句搞定数据库分页

select top 10 b.\* from (select top 20 主键字段,排序字段 from 表名 order by 排序字段 desc) a,表名 b where b.主键字段 = a.主键字段 order by a.排序字段

14、说明：前10条记录

select top 10 \* form table1 where 范围

15、说明：选择在每一组b值相同的数据中对应的a最大的记录的所有信息(类似这样的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)

select a,b,c from tablename ta where a=(select max(a) from tablename tb where tb.b=ta.b)

16、说明：包括所有在 TableA 中但不在 TableB和TableC 中的行并消除所有重复行而派生出一个结果表

(select a from tableA except (select a from tableB) except (select a from tableC)

17、说明：随机取出10条数据

select top 10 \* from tablename order by newid()

18、说明：随机选择记录

select newid()

19、说明：删除重复记录

Delete from tablename where id not in (select max(id) from tablename group by col1,col2,...)

20、说明：列出数据库里所有的表名

select name from sysobjects where type=U

21、说明：列出表里的所有的

select name from syscolumns where id=object\_id(TableName)

22、说明：列示type、vender、pcs字段，以type字段排列，case可以方便地实现多重选择，类似select 中的case。

select type,sum(case vender when A then pcs else 0 end),sum(case vender when C then pcs else 0 end),sum(case vender when B then pcs else 0 end) FROM tablename group by type

显示结果：

type vender pcs

电脑 A 1

电脑 A 1

光盘 B 2

光盘 A 2

手机 B 3

手机 C 3

23、说明：初始化表table1

TRUNCATE TABLE table1

24、说明：选择从10到15的记录

select top 5 \* from (select top 15 \* from table order by id asc) table\_别名 order by id desc

三、技巧

1、1=1，1=2的使用，在SQL语句组合时用的较多

“where 1=1” 是表示选择全部 “where 1=2”全部不选，

如：

if @strWhere !=

begin

set @strSQL = select count(\*) as Total from [ + @tblName + ] where + @strWhere

end

else

begin

set @strSQL = select count(\*) as Total from [ + @tblName + ]

end

我们可以直接写成

set @strSQL = select count(\*) as Total from [ + @tblName + ] where 1=1 安定 + @strWhere

2、收缩数据库

--重建索引

DBCC REINDEX

DBCC INDEXDEFRAG

--收缩数据和日志

DBCC SHRINKDB

DBCC SHRINKFILE

3、压缩数据库

dbcc shrinkdatabase(dbname)

4、转移数据库给新用户以已存在用户权限

exec sp\_change\_users\_login update\_one,newname,oldname

go

5、检查备份集

RESTORE VERIFYONLY from disk=E:dvbbs.bak

6、修复数据库

Alter DATABASE [dvbbs] SET SINGLE\_USER

GO

DBCC CHECKDB(dvbbs,repair\_allow\_data\_loss) WITH TABLOCK

GO

Alter DATABASE [dvbbs] SET MULTI\_USER

GO

7、日志清除

SET NOCOUNT ON

DECLARE @LogicalFileName sysname,

@MaxMinutes INT,

@NewSize INT

USE tablename -- 要操作的数据库名

Select @LogicalFileName = tablename\_log, -- 日志文件名

@MaxMinutes = 10, -- Limit on time allowed to wrap log.

@NewSize = 1 -- 你想设定的日志文件的大小(M)

-- Setup / initialize

DECLARE @OriginalSize int

Select @OriginalSize = size

FROM sysfiles

Where name = @LogicalFileName

Select Original Size of + db\_name() + LOG is +

CONVERT(VARCHAR(30),@OriginalSize) + 8K pages or +

CONVERT(VARCHAR(30),(@OriginalSize\*8/1024)) + MB

FROM sysfiles

Where name = @LogicalFileName

Create TABLE DummyTrans

(DummyColumn char (8000) not null)

DECLARE @Counter INT,

@StartTime DATETIME,

@TruncLog VARCHAR(255)

Select @StartTime = GETDATE(),

@TruncLog = BACKUP LOG + db\_name() + WITH TRUNCATE\_ONLY

DBCC SHRINKFILE (@LogicalFileName, @NewSize)

EXEC (@TruncLog)

-- Wrap the log if necessary.

WHILE @MaxMinutes > DATEDIFF (mi, @StartTime, GETDATE()) -- time has not expired

AND @OriginalSize = (Select size FROM sysfiles Where name = @LogicalFileName)

AND (@OriginalSize \* 8 /1024) > @NewSize

BEGIN -- Outer loop.

Select @Counter = 0

WHILE ((@Counter < @OriginalSize / 16) AND (@Counter < 50000))

BEGIN -- update

Insert DummyTrans VALUES (Fill Log)

Delete DummyTrans

Select @Counter = @Counter + 1

END

EXEC (@TruncLog)

END

Select Final Size of + db\_name() + LOG is +

CONVERT(VARCHAR(30),size) + 8K pages or +

CONVERT(VARCHAR(30),(size\*8/1024)) + MB

FROM sysfiles

Where name = @LogicalFileName

Drop TABLE DummyTrans

SET NOCOUNT OFF

8、说明：更改某个表

exec sp\_changeobjectowner tablename,dbo

9、存储更改全部表

Create PROCEDURE dbo.User\_ChangeObjectOwnerBatch

@OldOwner as NVARCHAR(128),

@NewOwner as NVARCHAR(128)

AS

DECLARE @Name as NVARCHAR(128)

DECLARE @Owner as NVARCHAR(128)

DECLARE @OwnerName as NVARCHAR(128)

DECLARE curObject CURSOR FOR

select Name = name,

Owner = user\_name(uid)

from sysobjects

where user\_name(uid)=@OldOwner

order by name

OPEN curObject

FETCH NEXT FROM curObject INTO @Name, @Owner

WHILE(@@FETCH\_STATUS=0)

BEGIN

if @Owner=@OldOwner

begin

set @OwnerName = @OldOwner + . + rtrim(@Name)

exec sp\_changeobjectowner @OwnerName, @NewOwner

end

-- select @name,@NewOwner,@OldOwner

FETCH NEXT FROM curObject INTO @Name, @Owner

END

close curObject

deallocate curObject

GO

10、SQL SERVER中直接循环写入数据

declare @i int

set @i=1

while @i<30

begin

insert into test (userid) values(@i)

set @i=@i+1

end