**迪亮信息数据库规范**



**上海迪亮信息科技有限公司**

**数据库规范**

2013年09月05日

V 1.1

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **说明：如形成文件、变更内容和变更范围** | **日期** | **变更人** |
| 1.0 | 初稿 | 2013/09/05 | 周建城 |
| 1.1 | 排版 | 2014/03/24 | 周建城 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. 目的 8](#_Toc366583487)

[2. 命名规范 9](#_Toc366583488)

[2.1 总则 9](#_Toc366583489)

[2.2 相关对象标识 9](#_Toc366583490)

[2.3 内容标识 10](#_Toc366583491)

[2.4 命名规则 10](#_Toc366583492)

[3 数据库编码规范 12](#_Toc366583493)

[3.1 编码规范总则 12](#_Toc366583494)

[3.1.1 文件结构 12](#_Toc366583495)

[3.1.2 排版 12](#_Toc366583496)

[3.1.3 注释 13](#_Toc366583497)

[3.2 SQL开发规范 14](#_Toc366583498)

[3.2.1 书写 15](#_Toc366583499)

[3.2.2 注释 17](#_Toc366583500)

[3.2.3 存储过程和函数 19](#_Toc366583501)

[3.2.4 触发器 24](#_Toc366583502)

[3.3 优化 24](#_Toc366583503)

[3.3.1 SQL语句原则 24](#_Toc366583504)

[3.3.2 索引创建原则 25](#_Toc366583505)

[3.3.3 充分利用索引 26](#_Toc366583506)

[3.3.4 Like的使用 29](#_Toc366583507)

[3.4 编码阶段 29](#_Toc366583508)

[3.4.1 尽可能少的范围数据 29](#_Toc366583509)

[3.4.2 尽可能少的重复操作 30](#_Toc366583510)

[3.4.3 注意事务和锁 31](#_Toc366583511)

[3.4.4 注意临时表和表变量的用法 32](#_Toc366583512)

[3.4.5 注意子查询的用法 33](#_Toc366583513)

[3.4.6 注意连接条件的用法 35](#_Toc366583514)

[3.4.7 查看执行计划及成本 36](#_Toc366583515)

[3.4.8 索引的使用及维护 36](#_Toc366583516)

[3.4.9 索引的选择技术 40](#_Toc366583517)

[3.4.10 索引的维护 41](#_Toc366583518)

[4. 数据库设计规范 43](#_Toc366583519)

[4.1 逻辑数据库规范化 43](#_Toc366583520)

[4.2 数据库设计原则 43](#_Toc366583521)

[4.2.1 规范命名 43](#_Toc366583522)

[4.2.2 控制字段的引用 44](#_Toc366583523)

[4.2.3 库表重复控制 44](#_Toc366583524)

[4.2.4 并发控制 44](#_Toc366583525)

[4.2.5 必要的讨论 44](#_Toc366583526)

[4.2.6 数据小组的审核 44](#_Toc366583527)

[4.3 数据库设计技巧 45](#_Toc366583528)

[4.3.1 分类拆分数据量大的表 45](#_Toc366583529)

[4.3.2 索引设计 45](#_Toc366583530)

[4.3.3 数据操作的优化 46](#_Toc366583531)

[4.3.4 数据库参数的调整 46](#_Toc366583532)

[4.3.5 必要的工具 47](#_Toc366583533)

[4.3.6 避免长事务 47](#_Toc366583534)

[4.3.7 避免或简化排序 47](#_Toc366583535)

[4.3.8 消除对大型表行数据的顺序存取 48](#_Toc366583536)

[4.3.9 避免相关子查询 49](#_Toc366583537)

[4.3.10 避免困难的正规表达式 49](#_Toc366583538)

[4.3.11 使用临时表加速查询 50](#_Toc366583539)

[4.3.12 适当超前 50](#_Toc366583540)

[5 数据库对象 51](#_Toc366583541)

[5.1 范围 51](#_Toc366583542)

[5.2 数据表 51](#_Toc366583543)

[5.2.1 主键 51](#_Toc366583544)

[5.2.2 字段 51](#_Toc366583545)

[5.3 完整性约束 52](#_Toc366583546)

[5.4 触发器 53](#_Toc366583547)

[5.4.1 使用范围 53](#_Toc366583548)

[5.5 视图 54](#_Toc366583549)

[5.5.1 用途 54](#_Toc366583550)

[5.5.2 使用范围 54](#_Toc366583551)

[5.6 函数 55](#_Toc366583552)

[5.6.1 用途 55](#_Toc366583553)

[5.6.2 适用范围 55](#_Toc366583554)

[5.7 存储过程 55](#_Toc366583555)

[5.7.1 用途 55](#_Toc366583556)

[5.7.2 事务处理 56](#_Toc366583557)

[5.7.3 错误处理 56](#_Toc366583558)

[5.7.4 游标 56](#_Toc366583559)

[6. 性能优化 57](#_Toc366583560)

[6.1 索引 57](#_Toc366583561)

[6.2 分区视图 57](#_Toc366583562)

[6.3 合理的冗余 57](#_Toc366583563)

[6.4 主键的设计 58](#_Toc366583564)

[6.5 外键的设计 58](#_Toc366583565)

[6.6 字段的设计 59](#_Toc366583566)

[6.7 索引的设计 59](#_Toc366583567)

1. **目的**

对数据库设计提出要求并制定规范和标准，以便在系统设计及软件编程过程中参考。

在一个设计团队中，采用统一的设计规范，以保证开发出高可管理性的高质量系统、能够快速的进行开发、以及减少维护代码的时间。

规范SQL的编写格式，提高SQL语句的可读性，共享性和执行效率。

1. **命名规范**
   1. **总则**
2. 命名对象包括文件、类、过程、函数、模块、变量。
3. 尽量使用英文。在英文不好表达其含义时，可使用汉语拼音。
4. 名称应能反映所代表对象的实际内容和作用。可使用多个词组合，但不要使用人名、项目组名。
5. 只可使用字母、数字和连字符‘\_’，但第一个字符必须是字母。
6. 多个词组合较长时可使用缩写，但尽可能使用标准的缩写。在一个系统中要建立缩语表以规范缩写的使用。
7. 命名时请考虑名字的唯一性和含义的准确性，不得使用非常相近的名字表示几个不同含义的对象。
8. 名称中字母的大小写在实施细则中规定，在同一个项目中必须统一。

**2.2 相关对象标识**

与特定内容相关的一组数据库对象，赋予一个指定的3字符的英文标识，便于识别和查找。如与客户信息CustomerInfo相关的数据库对象，则用cui标识；如与账户费用AccountExpense相关数据库对象，则用ace标识。

## 2.3 内容标识

与对象表示的内容相关的名称。尽量使用描述性的、完整的、能够描述每个对象的用途的名字；如果单词很长，可使用缩写。

用英文单词或多个单词组合（尽量避免使用汉语拼音，拒绝使用汉字）命名，每单词首字母大写。

用缩写时要使用标准的或通用的写法，同时要建立缩写词典以便标准化。

对于专业词汇，要建立词汇词典，以便统一和识别。

## 2.4 命名规则

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据库对象 | 前缀 | 相关对象标识 | 内容标识 | 后缀 |
| 数据库 |  |  | <内容标识> |  |
| 数据表 |  | <相关对象标识>\_ | <内容标识> | \_[<后缀>]① |
| 字段 |  | <相关对象标识>\_ | <内容标识> |  |
| 视图 | v\_ | <相关对象标识>\_ | <内容标识> |  |
| 函数 | f\_ | <相关对象标识>\_ | <内容标识> |  |
| 存储过程 | p\_ | <相关对象标识>\_ | <内容标识> |  |
| 触发器 | tr\_ | <表名> | <内容标识> |  |
| 索引 | Ix\_ | <表名> | <内容标识> |  |
| 主键 | pk\_ | <表名> | <内容标识> |  |
| 外键 | fk\_ | <表名> | <内容标识> |  |

注：

1. 只在需要时使用，用以表示同名表间的差别。如主表\_main，明细表\_detail，年度\_2000/\_2010。

② 不要使用数据库系统的保留词。对于可能在多个表中出现的，或常用的名称，要建立字段名词典，统一数据类型和长度，以便标准化。

# 3 数据库编码规范

## 3.1 编码规范总则

通过建立软件编码规范，形成代码约定，提高程序的可读性、一致性、可维护性，共享性和继承性，保证程序质量和可靠性，延长程序的生命周期。

### 3.1.1 文件结构

每个程序文件由标题、主体内容和附加说明三部分组成。

（1）标题：文件最前面的注释说明，其内容主要包括：程序名、作者、程序目的/功能、版本信息、版本号、生成日期、修改日志、依赖性等。必要时应有更详尽的说明。

（2）主体内容：为文件源代码部分基本上由预处理语句、类型定义、变量定义、实现等部分组成。

（3）附加说明：文件末尾的补充说明，如参考资料等，若内容不多也可放在标题部分的最后。

### 3.1.2 排版

（1）要采用缩进风格编写，缩进使用半角空格键，每级缩进的空格数为4个。

（2）每个程序块或语句块的起止符（如C/C++语言的大括号“{”和“}”，SQL子查询的“（”和“）”）应各独占一行并且与包含的语句有相同的缩进。

（3）函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格。

（4）较长的语句要分成多行书写，一般每行为120字符。长语句要在低优先级操作符处划分新行，操作符放在新行之首。划分出的新行相对首行要缩进。不得在单词或词组中间分行。

（5）独立的程序块之间、变量说明之后、或是程序说明跟主体之间必须加空行。

（6）一行只写一条语句，不允许把多个短语句写在一行中。

### 3.1.3 注释

为什么要使用注释：

方便代码的阅读和维护（修改），但不会增加可执行文件的大小和处理时间；

我们在维护代码时，多数时间不是编写或修改代码，而是在理解原有代码（有相当部分是自己编写的）。而好的注释可以大大减少理解代码的时间。

（1）注释的原则是有助于对程序的阅读理解，注释不宜太多也不能太少，注释语言必须易懂、简洁、含义准确，防止注释二义性。

（2）函数或程序的头部应进行注释，列出：函数或程序名、功能说明、参数说明、作者、创建时间、修改人、修改日期。

（3）边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。没有用的注释要及时删除。

（4）避免在注释中使用缩写，特别是非常用缩写。如果使用缩写，应对缩写进行必要的说明。

（5）注释应与其描述的代码邻近，可放在多行代码的上方或单行代码的行尾，不可放在下方；与被解释的代码之间不得有空行；如放于上方则需与其上面的代码用空行隔开。

（6）注释与所描述内容使用相同的缩进。

（7）避免在一行代码或表达式的中间插入注释。

（8）当代码段较长，特别是多重嵌套时，可在程序块的结束行右方加注释标记，以表明某程序块的结束。

（9）使用中文注释。

## 3.2 SQL开发规范

规范SQL的编写格式，提高SQL语句的可读性，共享性和执行效率。

大部分业务系统需要与数据库进行交互，与数据库交互的主要方式就是SQL语句，编写规范的SQL语句不但利于阅读，而且被数据库重复使用的几率也较大，执行效率相对较高，编写的好的SQL与编写的差的SQL在执行性能上可能会差几倍甚至几千几万倍，因此养成好的SQL编写规范对于提高项目质量及提高开发人员自身素质有着潜在的极大的影响。

### 3.2.1 书写

SQL书写遵守如下规范：

在同一个项目中，为了最大限度实现SQL的共享，要求书写SQL语句时大小写要一致，为了阅读方便和统一起见，所有SQL语句全部小写（如SQL谓词，字段名，表名等）,常量除外，常量可以按需要书写。

举例：下面两个相同的语句除常量外都要统一起来。

1) select name from emp;

2) select 'NAME' from emp where emp\_no='QD001';

SQL语句尽可能放在一行，若SQL太长放在一行中影响阅读时可分多行，但要保持缩进一致，缩进可用TAB或者空格，但TAB数和空格数最好一致。

SQL语句中，各谓词之间以空格分割的，尽量保持空格数量一致，即若用一个空格分割，则全部都用一个空格分割，便于数据库能够共享。

能使用绑定变量的，尽量使用绑定变量，尤其是在前台程序中。

对下面列出的情况，慎重使用绑定变量：

1）列值倾斜严重，如：某一状态列大部分值是‘1’，只有极少数值为‘2’，这种情况不宜用绑定变量，而应该用常量，便于数据库使用柱状图统计信息。

2）日期时间列。

总之：书写SQL的目标是若SQL的用途是一样的，则SQL应该完全一致，包括空格，大小写。下面的语句由于写法不完全相同，数据库会理解为4条不同的语句从而导致重复编译，降低了性能。

1）select name from emp where emp\_no='111';

2）Select name From emp Where emp\_no='111';

3）select Name from EMP where emp\_no='111';

4）select name from emp where EMP\_NO='111';

下面的语句，由于语句规范，可以只编译一次。

1）select name from emp where emp\_no='111';

2）select name from emp where emp\_no='222';

3）select name from emp where emp\_no=@emp\_no;

4）select name from emp where emp\_no=:emp\_no;

### 3.2.2 注释

#### 3.2.2.1 开头注释

所有的过程、函数、触发器、包都应该在开头有注释，

注释中要列出对象名称；完成功能简述；调用模块，调用时机；

创建日期；作者信息；历次修改日期；历次修改人；历次修改原因和其它作者认为重要的内容，在每次修改的注释之间建一空行。

举例如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

名称：P\_WO\_MULTI\_REPAIR\_JOB

功能：多次维修判定程序

调用：自动统计月结前，后台调用

作者：pekah

創建时间：2013-09-06

修改人：pekah

修改时间: 2013-09-06

修改内容：重写部分SQL优化性能

修改人：pekah

修改日期：2013-09-06

修改内容：增加判断条件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### 3.2.2.2块注释

对于复杂的语句块，必须提供块注释，清晰描述该语句块的功能逻辑、数据结构以及算法；

块注释应该和它们所描述的代码具有一样的缩进格式；

块注释之首应该有一个空行，用于把块注释和代码分割开来；

块注释结构如下所示：

/\*

计算配送能力系数：

派工系数＝a\*b+c\*d+e\*f

a:考评成绩，取xx的考评成绩

c:承担度度，根据当前遗留量和总服务能力计算所得，

c=当前遗留配送量/总配送能力

...

\*/

#### 3.2.2.3 行注释

对于复杂的SQL语句，必须提供行注释，清晰描述该SQL语句功能以及目的；行注释结构如下所示：

--计算动态能力得分,当负荷率超高时，可能出现负值，注意处理。

添加注释时要注意：注释中包含GO 命令时会生成一个错误消息。

### 3.2.3 存储过程和函数

#### 3.2.3.1 命名规范

存储过程/函数的命名（代码）使用以下方式：

存储过程名= p\_ + 模块代码 + 存储过程代码

函数名= f\_ + 模块代码 + 函数代码

存储过程以表现其功能的简洁语言进行命名，定义好后，放置于每一个Diagram的特定区域（一般置于右边部分的上边）。

#### 3.2.3.2存储过程/函数头定义规范

包括过程/函数名称定义、参数定义、注释说明、变量定义四部分。

参数定义

所有的参数必须显示指出变量类型，对于返回参数，必须指出其方向；

参数变量定义格式为：“a“ + 参数类型简写 ＋ “\_” ＋ 参数名称，其中a是单词argument(参数)的第一个字母，参数类型如下：

文本型：s

日期时间型：d

数字型：n

整形：i

布尔型：b

举例：as\_name 表示文本型

ad\_birthdate 表示日期时间型

an\_age 表示数字型

注释说明

见前面描述。

#### 3.2.3.3 变量定义规范

1). 变量的命名

除游标变量以外，所有的变量命名采用此方式：

变量名= “v\_”+ 变量名称

游标变量采用：

变量名= “cur”+ 变量名(第一个字母大写)。

其中变量名称由代表变量确切含意的单词代码组合而成，每一个单词的首字母根据阅读的方便性决定是否需要大写。

举例：

变量名：v\_EmpName

游标名：curNetlist

2). 变量类型

定义变量的类型时，尽量采用显示定义的方式。尽量将变量的定义分开来，并进行格式化，以便程序代码的阅读。例如：

v\_UserName CHAR(10)

v\_SEX CHAR(1)

3). 变量注释

原则上要求对于一般变量定义时必须加上注释，如变量有特定的值范围，则必须显示描述各确切值及其含意说明；

4). 存储过程/函数体定义规范

在过程/函数的定义中，最重要的是过程代码的严谨性和可读性，主要包括以下的注意事项：

a. 如无特殊需要，SQL中涉及的所有内容都要小写。

b. 原则上要求所有的SQL语句必须在其前面加上注释，对于IF/CASE等流程控制语句，必须在语句前/后说明控制处理和可能的流程方向；

c. 所有的赋值语句要求变量与运算符之间要有空格。

如：v\_Count := v\_Count + 1，并保持适当的对齐；

d. 尽量避免复杂SQL，尤其是关联多个大表的SQL，对于需要关联多个大表的SQL最好分解成小的SQL分步处理，避免出现性能问题，对于复杂的语句块之间，要求中间加入空行；

e. 所有可能的返回结果，必须在过程体中显示定义和说明，并在注释中说明。在其它的过程、代码中调用本过程/函数时，必须在其代码中处理所有的各种可能的返回结果；

f. 锁定数据时，尽可能只锁定要操作的数据行，避免锁定整个表，限制使用local table等DDL语句，若要对数据表、数据行加锁时，需要考虑由此导致的并发操作失败的处理；

g. 所有的过程/函数代码在编写之前，必须要有设计原型及其说明；

5). 存储过程/函数错误定义

a). 所有的DML语句必须考虑死锁、并发、主关键字不唯一等的出错异常处理，应该对所有可能出现的异常进行捕获、编码处理，并注明异常的内容；

b). 异常处理有两种需要考虑的问题，一是事务的完整性；二是错误的出错日志及返回处理。所有异常的根据其类型和等级进行以下的处理：

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **内容** | **处理** |  |
| 系统级 | 数据库级发出的底层的异常，如主关键字不唯一冲突等； | 返回定义错误代码为其它错误，完全回退事务，登录异常日志； |  |
| 应用级 | 应用系统定义并发出的异常，如指定记录不存在； | 定义应用级错误代码，根据实际情况处理事务； |  |

6). 存储过程/函数使用原则

a). 存储过程功能尽量保持独立、复用，但尽量避免嵌套调用，充分考虑其性能体现；

b). 使用前台代码调用存储过程时，必须考虑返回传入参数的有效性、代码的处理及展示、事务完整性处理；

c). 直接使用参数作为变量处理，不要在存储过程中定义参数同样含义的变量使用，避免由于值的不同导致性能问题。

### 3.2.4 触发器

1). 命名规范

触发器不同类型的命名结构如下

|  |  |  |
| --- | --- | --- |
| 操作 | 时机 | 命名结构 |
| Insert | After | ai\_表名 |
| Update | After | au\_表名 |
| Delete | After | ad\_表名 |

2). 触发器编写规范

请参见存储过程/函数编写规范。

3). 异常

觸發器处理必须考虑可能出现的错误，一是数据库或操作系统底层错误；二是由于业务逻辑造成的错误；三是无反应或超时；四是输入参数不在指定范围内。

设计所有的过程时，必须考虑出错后的恢复。

## 3.3 优化

### 3.3.1 SQL语句原则

1). Where子句尽量避免使用函数；

2). 避免在ORDER BY子句中使用表达式；

3). 慎用游标；

4). 大小写规范统一，变量绑定统一，避免重复编译；

5). 尽可能少的返回结果集行的数量

6). 避免使用select \* 语句；

7). 减少结果集中的列的数量；

8). 视图嵌套使用不能超过3层；

9). 不要使用没有意义的列作为聚集索引列，例如，加1自增列；

10). 避免隐式类型转换，例如字符型一定要用’’，数字型一定不要使用’’；

11). 查询语句一定要有范围的限定，避免全表扫描操作；

12). 合理对大表进行分区；

13). 慎用DISTINCT关键字；

14). 慎用UNION关键字，可以用OR替代；

15). 使用top 1替count(\*)来判断是否存在记录；

### 3.3.2 索引创建原则

1). 同一索引中的组成列最好不要超过3列。

2). 把经常一起出现的字段组合在一起，组成组合索引，组合索引的字段顺序与主键一样，也需要把最常用的字段放在前面，把重复率低的字段放在前面。

3). 根据使用频率决定哪些字段需要建立索引，选择经常作为连接条件、筛选条件、聚合查询、排序的字段作为索引的候选字段。

4). 根据数据量决定哪些表需要增加索引，数据量小的可以只有主键。

5). 若某列中有大量的值是空值，可以建立索引。

6). 要对值分布较宽的列建立索引。

7). 若表主要用来查询，则可按需要建立索引，若对表操作主要是UPDATE，则尽可能少建索引。

8). 不要对值较窄的列建立索引，如性别

9). 不要索引较小的表（如表不足1000行）

10). 若某列的值大部分是a，少数是别的值（如b,c,d…），且经常以该列的其它值(如b,c,d…)为查询条件，则可以将值a设为空值，并在此列上创建索引。

### 3.3.3 充分利用索引

1). 函数、表达式使用

在where语句中，尽量避免在运算符左边对列进行函数或者表达式操作，容易引起全表扫描，要尽可能将操作移至运算符右边。

2). IN/OR子句使用

IN、OR、NOT IN SQLServer2005数据库可以分析出应该根据索引查找。属于2005版本的新特性。

3). !=或<>操作符子句使用

!=或<>操作符可以用INDEXSEEK查找的，可以正常使用。

4). 不要对索引字段进行运算

例如：

select id from t where num/2=100;

应改为:

select id from t where num=100\*2;

select id from t where num/2=num1;

如果num有索引应改为:

select id from t where num=num1\*2;

如果num1有索引则不应该改。

5). 不要对索引字段进行格式转换

日期字段的例子：

where convert(varchar(10),createdate,120)='2008-08-15';

应改为：

where createdate>='2008-08-15' and createdate <'2008-08-16';

ISNULL转换的例子：

where isnull(字段,'')<>''

应改为：

where 字段 <> ''

where isnull(字段,'')=''不应修改

where isnull(字段,'F')='T'

应改为：

where 字段='T'

where isnull(字段,'F')<>'T'不应修改

6). 不要对索引字段使用函数

Where left(name,3)='ABC'

或者

where substring(name,1,3)='ABC'

应改为:

where name like 'ABC%'

日期查询的例子：

where datediff(day,日期,'2005-11-30')=0

应改为：

where 日期>='2005-11-30' and 日期<'2005-12-1'

where datediff(day,日期,'2005-11-30')>0

应改为：

where 日期<'2005-11-30'

where datediff(day,日期,'2005-11-30')>=0

应改为：

where 日期<'2005-12-01'

where datediff(day,日期,'2005-11-30')<0

应改为：

where 日期>='2005-12-01'

where datediff(day,日期,'2005-11-30')<=0

应改为：

where 日期>='2005-11-30'

7). 不要对索引字段进行多字段连接

例如：

where fname+'.'+lname='H.Y'

应改为:

where fname='H' and lname='Y'

### 3.3.4 Like的使用

对索引列避免使用like ‘%xx’， 应该使用like ‘xx%’。设计数据结构时就应该考虑这个问题，不要出现必须要采用like ‘%xx’才能满足业务需要的情形。

## 3.4 编码阶段

### 3.4.1 尽可能少的范围数据

1). 横向来看，不要写select \*的语句，而是选择你需要的字段。

2). 纵向来看，合理写where子句，不要写没有where的SQL语句。

3). 注意select into后的where子句，因为select into把数据插入到临时表，这个过程会锁定一些系统表，如果这个where子句返回的数据过多或者速度太慢，会造成系统表长期锁定，诸塞其他进程。

4). 对于聚合查询，可以用having子句进一步限定返回的行。

### 3.4.2 尽可能少的重复操作

1). 控制同一语句的多次执行，特别是一些基础数据的多次执行是很多程序员很少注意的。

2). 减少多次的数据转换，也许需要数据转换是设计的问题，但是减少次数是程序员可以做到的。

3). 杜绝不必要的子查询和连接表，子查询在执行计划一般解释成外连接，多余的连接表带来额外的开销。

4). 合并对同一表同一条件的多次UPDATE，比如

update employee set fname='H' where EMP\_ID='b001';

update employee set lname='Y' where EMP\_ID='b001';

这两个语句应该合并成以下一个语句

update employee set fname='H', lname='Y' where EMP\_ID='b001';

5). UPDATE操作不要拆成DELETE操作+INSERT操作的形式，虽然功能相同，但是性能差别是很大的。

6). 不要写一些没有意义的查询，比如

select \* from employee where 1=2;

### 3.4.3 注意事务和锁

事务是数据库应用中和重要的工具，它有原子性、一致性、隔离性、持久性这四个属性，很多操作我们都需要利用事务来保证数据的正确性。在使用事务中我们需要做到尽量避免死锁、尽量减少阻塞。具体以下方面需要特别注意：

1). 事务操作过程要尽量小，能拆分的事务要拆分开来。

2). 事务操作过程不应该有交互，因为交互等待的时候，事务并未结束，可能锁定了很多资源。

3). 事务操作过程要按同一顺序访问对象。

4). 提高事务中每个语句的效率，利用索引和其他方法提高每个语句的效率可以有效地减少整个事务的执行时间。

5). 尽量不要指定锁类型和索引，SQL SERVER允许我们自己指定语句使用的锁类型和索引，但是一般情况下，SQL SERVER优化器选择的锁类型和索引是在当前数据量和查询条件下是最优的，我们指定的可能只是在目前情况下更有，但是数据量和数据分布在将来是会变化的。

### 3.4.4 注意临时表和表变量的用法

在复杂系统中，临时表和表变量很难避免，关于临时表和表变量的用法，需要注意：

1). 如果语句很复杂，连接太多，可以考虑用临时表和表变量分步完成。

2). 如果需要多次用到一个大表的同一部分数据，考虑用临时表和表变量暂存这部分数据。

3). 如果需要综合多个表的数据，形成一个结果，可以考虑用临时表和表变量分步汇总这多个表的数据。

4). 其他情况下，应该控制临时表和表变量的使用。

5). 关于临时表和表变量的选择，很多说法是表变量在内存，速度快，应该首选表变量，但是在实际使用中发现，这个选择主要考虑需要放在临时表的数据量，在数据量较多的情况下，临时表的速度反而更快。

6). 关于临时表产生使用SELECTINTO和CREATETABLE+INSERTINTO的选择，我们做过测试，一般情况下，SELECTINTO会比CREATETABLE+INSERTINTO的方法快很多，但是SELECTINTO会锁定TEMPDB的系统表SYSOBJECTS、SYSINDEXES、SYSCOLUMNS，在多用户并发环境下，容易阻塞其他进程，所以我的建议是，在并发系统中，尽量使用CREATETABLE+INSERTINTO，而大数据量的单个语句使用中，使用SELECTINTO。

7). 注意排序规则，用CREATETABLE建立的临时表，如果不指定字段的排序规则，会选择TEMPDB的默认排序规则，而不是当前数据库的排序规则。如果当前数据库的排序规则和TEMPDB的排序规则不同，连接的时候就会出现排序规则的冲突错误。一般可以在CREATETABLE建立临时表时指定字段的排序规则为DATABASE\_DEFAULT来避免上述问题。

### 3.4.5 注意子查询的用法

子查询是一个 SELECT查询，它嵌套在 SELECT、INSERT、UPDATE、DELETE语句或其它子查询中。任何允许使用表达式的地方都可以使用子查询。

子查询可以使我们的编程灵活多样，可以用来实现一些特殊的功能。但是在性能上，往往一个不合适的子查询用法会形成一个性能瓶颈。

如果子查询的条件中使用了其外层的表的字段，这种子查询就叫作相关子查询。相关子查询可以用IN、NOT IN、EXISTS、NOT EXISTS引入。

关于相关子查询，应该注意：

1. NOT IN、NOT EXISTS的相关子查询可以改用LEFT JOIN代替写法。比如：

select pub\_name from publishers where pub\_id not in

(select pub\_id from titles where where type='BUSINESS');

可以改写成：

select a.pub\_name from publishers a

left join titles b on b.type='BUSINESS' and a.pub\_id=b.pub\_id;

select title from titles

where not exists (select title\_id from sales where title\_id=titles.title\_id);

可以改写成：

select title

from titles left join sales on sales.title\_id=titles.title\_id

where sales.title\_id is not null;

2). 如果保证子查询没有重复，IN、EXISTS的相关子查询可以用INNER JOIN代替。比如：

select pub\_name from publlishers where pub\_id in (select pub\_id from titles where type='BUSINESS');

可以改写成：

select distinct a.pub\_name from publishers a inner join titles b on b.type='BUSINESS' and a.pub\_id= b.pub\_id;

3). 不要用 count(\*) 的子查询判断是否存在记录，最好用LEFT JOIN或者EXISTS，比如有人写这样的语句：

select job\_desc from jobs where (select count(\*) from employee where job\_id=jobs.job\_id)=0;

应该改成：

select jobs.job\_desc from jobs left join employee

on employee.job\_id=jobs.job\_id

where employee.emp\_id is not null;

select job\_desc from jobs

where (select count(\*) from employee where job\_id=jobs.job\_id)<>0;

应该改成：

select job\_desc from jobs

where exists (select 1 from employee where job\_id=jobs.jobs\_id);

### 3.4.6 注意连接条件的用法

多表连接的连接条件对索引的选择有着重要的意义，所以我们在写连接条件的时候需要特别的注意。

1). 多表连接的时候，连接条件必须写全，宁可重复，不要缺漏。

2). 连接条件尽量使用聚集索引

3). 注意ON部分条件和WHERE部分条件的区别

### 3.4.7 查看执行计划及成本

养成良好的编码习惯，每写完一个SQL语句，就查看执行计划及成本分析。在大表上尽可能避免全表扫描及聚集索引扫描。

### 3.4.8 索引的使用及维护

在良好的数据库设计基础上，能有效地使用索引是SQL Server取得高性能的基础，SQL Server采用基于代价的优化模型，它对每一个提交的有关表的查询，决定是否使用索引或用哪一个索引。因为查询执行的大部分开销是磁盘I/O，使用索引提高性能的一个主要目标是避免全表扫描，因为全表扫描需要从磁盘上读表的每一个数据页，如果有索引指向数据值，则查询只需读几次磁盘就可以了。所以如果建立了合理的索引，优化器就能利用索引加速数据的查询过程。但是，索引并不总是提高系统的性能，在增、删、改操作中索引的存在会增加一定的工作量，因此，在适当的地方增加适当的索引并从不合理的地方删除次优的索引，将有助于优化那些性能较差的SQL Server应用。

#### 3.4.8.1 聚簇索引(clustered indexes)的使用

聚簇索引是一种对磁盘上实际数据重新组织以按指定的一个或多个列的值排序。由于聚簇索引的索引页面指针指向数据页面，所以使用聚簇索引查找数据几乎总是比使用非聚簇索引快。每张表只能建一个聚簇索引，并且建聚簇索引需要至少相当该表120%的附加空间，以存放该表的副本和索引中间页。建立聚簇索引的思想是：

1). 大多数表都应该有聚簇索引或使用分区来降低对表尾页的竞争，在一个高事务的环境中，对最后一页的封锁严重影响系统的吞吐量。

2). 在聚簇索引下，数据在物理上按顺序排在数据页上，重复值也排在一起，因而在那些包含范围检查(between、<、<=、>、>=)或使用group by或order by的查询时，一旦找到具有范围中第一个键值的行，具有后续索引值的行保证物理上毗连在一起而不必进一步搜索，避免了大范围扫描，可以大大提高查询速度。

3). 在一个频繁发生插入操作的表上建立聚簇索引时，不要建在具有单调上升值的列(如IDENTITY)上，否则会经常引起封锁冲突。

4). 在聚簇索引中不要包含经常修改的列，因为码值修改后，数据行必须移动到新的位置。

5). 选择聚簇索引应基于where子句和连接操作的类型。

聚簇索引的侯选列是：

1). 主键列,该列在where子句中使用并且插入是随机的。

2). 按范围存取的列，如pri\_order > 100 and pri\_order < 200。

3). 在group by或order by中使用的列。

4). 不经常修改的列。

5). 在连接操作中使用的列。

#### 3.4.8.2 非聚簇索引(nonclustered indexes)的使用

SQL Server缺省情况下建立的索引是非聚簇索引，由于非聚簇索引不重新组织表中的数据，而是对每一行存储索引列值并用一个指针指向数据所在的页面。换句话说非聚簇索引具有在索引结构和数据本身之间的一个额外级。一个表如果没有聚簇索引时,可有250个非聚簇索引。每个非聚簇索引提供访问数据的不同排序顺序。在建立非聚簇索引时，要权衡索引对查询速度的加快与降低修改速度之间的利弊。另外，还要考虑这些问题：

1、索引需要使用多少空间。

2、合适的列是否稳定。

3、索引键是如何选择的，扫描效果是否更佳。

4、是否有许多重复值。

对更新频繁的表来说，表上的非聚簇索引比聚簇索引和根本没有索引需要更多的额外开销。对移到新页的每一行而言，指向该数据的每个非聚簇索引的页级行也必须更新，有时可能还需要索引页的分理。从一个页面删除数据的进程也会有类似的开销，另外，删除进程还必须把数据移到页面上部，以保证数据的连续性。所以，建立非聚簇索引要非常慎重。非聚簇索引常被用在以下情况:

1、某列常用于集合函数(如Sum,....)。

2、某列常用于join,order by,group by。

3、查寻出的数据不超过表中数据量的20%。

#### 3.4.8.3 覆盖索引(covering indexes)的使用

覆盖索引是指那些索引项中包含查寻所需要的全部信息的非聚簇索引，这种索引之所以比较快也正是因为索引页中包含了查寻所必须的数据,不需去访问数据页。如果非聚簇索引中包含结果数据,那么它的查询速度将快于聚簇索引。 但是由于覆盖索引的索引项比较多,要占用比较大的空间。而且update操作会引起索引值改变。所以如果潜在的覆盖查询并不常用或不太关键，则覆盖索引的增加反而会降低性能。

### 3.4.9 索引的选择技术

索引的有无，建立方式的不同将会导致不同的查询效果，选择什么样的索引基于用户对数据的查询条件,这些条件体现于where从句和join表达式中。一般来说建立索引的思路是：

1). 主键时常作为where子句的条件，应在表的主键列上建立聚簇索引，尤其当经常用它作为连接的时候。

2). 有大量重复值且经常有范围查询和排序、分组发生的列，或者非常频繁地被访问的列，可考虑建立聚簇索引。

3). 经常同时存取多列，且每列都含有重复值可考虑建立复合索引来覆盖一个或一组查询，并把查询引用最频繁的列作为前导列，如果可能尽量使关键查询形成覆盖查询。

4). 如果知道索引键的所有值都是唯一的，那么确保把索引定义成唯一索引。

5). 在一个经常做插入操作的表上建索引时，使用fillfactor(填充因子)来减少页分裂，同时提高并发度降低死锁的发生。如果在只读表上建索引，则可以把fillfactor置为100。

6.) 在选择索引键时，设法选择那些采用小数据类型的列作为键以使每个索引页能够容纳尽可能多的索引键和指针，通过这种方式，可使一个查询必须遍历的索引页面降到最小。此外，尽可能地使用整数为键值，因为它能够提供比任何数据类型都快的访问速度。

### 3.4.10 索引的维护

某些不合适的索引影响到SQL Server的性能,随着应用系统的运行,数据不断地发生变化,当数据变化达到某一个程度时将会影响到索引的使用。这时需要用户自己来维护索引。索引的维护包括：

#### 3.4.10.1重建索引

随着数据行的插入、删除和数据页的分裂，有些索引页可能只包含几页数据，另外应用在执行大块I/O的时候，重建非聚簇索引可以降低分片，维护大块I/O的效率。重建索引实际上是重新组织B-树空间。在下面情况下需要重建索引：

1). 数据和使用模式大幅度变化。

2). 排序的顺序发生改变。

3). 要进行大量插入操作或已经完成。

4). 使用大块I/O的查询的磁盘读次数比预料的要多。

5). 由于大量数据修改，使得数据页和索引页没有充分使用而导致空间的使用超出估算。

6). dbcc检查出索引有问题。

当重建聚簇索引时,这张表的所有非聚簇索引将被重建。

#### 3.4.10.2索引统计信息的更新

当在一个包含数据的表上创建索引的时候，SQL Server会创建分布数据页来存放有关索引的两种统计信息：分布表和密度表。优化器利用这个页来判断该索引对某个特定查询是否有用。但这个统计信息并不动态地重新计算。这意味着，当表的数据改变之后，统计信息有可能是过时的，从而影响优化器追求最有工作的目标。因此，在下面情况下应该运行update statistics命令：

1). 数据行的插入和删除修改了数据的分布。

2). 对用truncate table删除数据的表上增加数据行。

3). 修改索引列的值。

# 4. 数据库设计规范

## 4.1 逻辑数据库规范化

一般来说，逻辑数据库设计需要满足规范化的前3级标准：

第1范式：没有重复的组或多值的列；

第2范式：每个非关键字段必须依赖于主关键字，不能依赖于一个组合式主关键字的某些组成部分；

第3范式：一个非关键字段不能依赖于另一个非关键字段。

遵守这些规则的数据库设计会产生较少的列和更多的表，因而也就减少了数据冗余，也减少了用于存储数据的页

没有冗余的数据库设计可以做到。但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是：在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

## 4.2 数据库设计原则

### 4.2.1 规范命名

所有的库名、表名、域名必须遵循统一的命名规则，并进行必要说明，以方便设计、维护、查询

### 4.2.2 控制字段的引用

在设计时，可以选择适当的数据库设计管理工具，以方便开发人员的分布式设计和数据小组的集中审核管理。采用统一的命名规则，如果设计的字段已经存在，可直接引用；否则，应重新设计。

### 4.2.3 库表重复控制

在设计过程中，如果发现大部分字段都已存在，开发人员应怀疑所设计的库表是否已存在。通过对字段所在库表及相应设计人员的查询，可以确认库表是否确实重复。

### 4.2.4 并发控制

设计中应进行并发控制，即对于同一个库表，在同一时间只有一个人有控制权，其他人只能进行查询。

### 4.2.5 必要的讨论

数据库设计完成后，数据小组应于相关人员进行讨论，通过讨论来熟悉数据库，从而对设计中存在的问题进行控制或从中获取数据库设计的必要信息。

### 4.2.6 数据小组的审核

库表的定版、修改最终都要通过数据小组的审核，以保证符合必要的要求。

## 4.3 数据库设计技巧

### 4.3.1 分类拆分数据量大的表

对于经常使用的表（如某些参数表或代码对照表），由于其使用频率很高，要尽量减少表中的记录数量。例如，通过业务类别将数据拆分，则可以大大提高查询效率。

### 4.3.2 索引设计

对于大的数据库表，合理的索引能够提高整个数据库的操作效率。

在索引设计中，索引字段应挑选重复值较少的字段；在对建有复合索引的字段进行检索时，应注意按照复合索引字段建立的顺序进行。

例如，如果对一个5万多条记录的订单表以日期和订单号为序建立复合索引，由于在该表中日期的重复值接近整个表的记录数，用订单号进行查询所用的时间接近3秒；而如果以订单号为索引字段建立索引进行相同的查询，所用时间不到１秒。因此在大型数据库设计中，只有进行合理的索引字段选择，才能有效提高整个数据库的操作效率。

### 4.3.3 数据操作的优化

在大型数据库中，如何提高数据操作效率值得关注。例如，每在数据库流水表中增加一笔业务，就必须从流水控制表中取出流水号，并将其流水号的数值加一。正常情况下，单笔操作的反应速度尚属正常，但当用它进行批量业务处理时，速度会明显减慢。

经过分析发现，每次对流水控制表中的流水号数值加一时都要锁定该表，而该表却是整个系统操作的核心，有可能在操作时被其他进程锁定，因而使整个事务操作速度变慢。

对这一问题的解决的办法是，根据批量业务的总笔数批量申请流水号，并对流水控制表进行一次更新，即可提高批量业务处理的速度。

另一个例子是对插表的优化。对于大批量的业务处理，如果在插入数据库表时用普通的insert语句，速度会很慢。其原因在于，每次插表都要进行一次I/O操作，花费较长的时间。改进后，可以用put语句等缓冲区形式等满页后再进行I/O操作，从而提高效率。

对大的数据库表进行删除时，一般会直接用delete语句，这个语句虽然可以进行小表操作，但对大表却会因带来大事务而导致删除速度很慢甚至失败。解决的方法是去掉事务，但更有效的办法是先进行drop操作再进行重建。

### 4.3.4 数据库参数的调整

数据库参数的调整是一个经验不断积累的过程，应由有经验的系统管理员完成。如记录锁的数目太少会造成锁表的失败；逻辑日志的文件数目太少会造成插入大表失败等，这些问题都应根据实际情况进行必要的调整。

### 4.3.5 必要的工具

在整个数据库的开发与设计过程中，可以先开发一些小的应用工具，如自动生成库表的头文件、插入数据的初始化、数据插入的函数封装、错误跟踪或自动显示等，以此提高数据库的设计与开发效率。

### 4.3.6 避免长事务

对单个大表的删除或插入操作会带来大事务，解决的办法是对参数进行调整，也可以在插入时对文件进行分割。对于一个由一系列小事务顺序操作共同构成的长事务，可以由一系列操作完成整个事务，但其缺点是有可能因整个事务太大而使不能完成，或者，由于偶然的意外而使事务重做所需的时间太长。较好的解决方法是，把整个事务分解成几个较小的事务，再由应用程序控制整个系统的流程。这样，如果其中某个事务不成功，则只需重做该事务，因而既可节约时间，又可避免长事务。

### 4.3.7 避免或简化排序

应当尽量简化或避免对大型表进行重复的排序。当能够利用索引自动以适当的次序产生输出时，优化器就避免了排序这个步骤。为了避免不必要的排序，就要正确地增建索引，合理地合并数据库表（尽管有时可能影响表的规范化，但相对于效率的提高是值得的）。如果排序不可避免，那么应当试图简化它，如缩小排序的列的范围等。

### 4.3.8 消除对大型表行数据的顺序存取

在嵌套查询中，表的顺序存取对查询效率可能产生致命的影响。

比如采用顺序存取策略，一个嵌套3层的查询，如果每层都查询1000行，那么这个查询就要查询10亿行数据。避免这种情况的主要方法就是对连接的列进行索引。例如，两个表:学生表(学号、姓名、年龄……)和选课表(学号、课程号、成绩)。如果两个表要做连接，就要在“学号”这个连接字段上建立索引。

我们有时可以使用并集来避免顺序存取。尽管也许在所有的检查列上都有索引，但某些形式的where子句会强迫优化器使用顺序存取，这一点也应注意。

下面的查询将强迫对orders表执行顺序操作:

select \* from orders where

(customer\_num=104 and order\_num>1001) or order\_num=1008;

虽然在customer\_num和order\_num上建有索引，但是在上面的语句中优化器还是使用顺序存取路径扫描整个表。因为这个语句要检索的是分离的行的集合，所以应该改为如下语句，就能利用索引路径处理查询。

select \* from orders where customer\_num=104 AND order\_num>1001

union

select \* from orders where order\_num=1008;

### 4.3.9 避免相关子查询

如果一个列同时在主查询和where子句中出现，很可能当主查询中的列值改变之后，子查询必须重新查询一次。而且查询嵌套层次越多，效率越低，因此应当尽量避免子查询。如果子查询不可避免，那么要在子查询中过滤掉尽可能多的行。

### 4.3.10 避免困难的正规表达式

MATCHES和LIKE关键字支持通配符匹配，技术上叫正规表达式。但这种匹配特别耗费时间。例如: select \* from customer where zipcode like '98\_ \_ \_';

即使在zipcode字段上建立了索引，在这种情况下也还是采用顺序扫描的方式。如果把语句改为select \* from customer where zipcode>'98000';，在执行查询时就会利用索引来查询，显然会大大提高速度。

### 4.3.11 使用临时表加速查询

把表的一个子集进行排序并创建临时表，有时能加速查询。它有助于避免多重排序操作，而且在其他方面还能简化优化器的工作。临时表中的行要比主表中的行少，而且物理顺序就是所要求的顺序，减少了磁盘i/o，所以查询工作量可以得到大幅减少。但要注意，临时表创建后不会反映主表的修改。在主表中数据频繁修改的情况下，注意不要丢失数据。

### 4.3.12 适当超前

计算机技术发展日新月异，数据库的设计必须具有一定前瞻性，不但要满足当前的应用要求，还要考虑未来的业务发展，同时必须有利于扩展或增加应用系统的处理功能。

# 5 数据库对象

## 5.1 范围

本规范涉及的数据库对象包括：数据库、数据表、字段、视图、函数、存储过程、触发器、索引。

## 5.2 数据表

### 5.2.1 主键

每一个数据表必须指定一个主键。如果数据本身没有明显的主键，可增加一个唯一键列作为主键，数据类型可以是uniqueidentifier或自动增长的整数。

### 5.2.2 字段

#### 5.2.2.1 数据类型

字符型字段尽可能使用可变长的字符类型。在长度的设定时要考虑到不可预见性及未来可能的变化，取一个比较宽松的长度，避免长度变化产生的关联改变。

#### 5.2.2.2 计算列

对于类似金额=数量\*单价的形式，如果表中保存了数量和单价，则金额设置为计算列，可保证计算结果的唯一性，又可保证数据的完整性。

#### 5.2.2.3 冗余字段

在如下情况下，允许保留字段冗余，在数据库设计时违反关系数据库范式规定：

只保存代码时，录入界面无法呈现其名称，为方便操作，此时允许在表中加入名称列。但当编码表中名称变更时，要有相应的程序完成名称的同步。在引用该表数据时不使用其名称列，名称取自编码表。如在销售订单中表中同时存在客户的编码和名称列。

当两个有所属关系的列，其所属关系会发生变化时，在表中允许同时保留这两列，记录数据发生时的所属关系。如城市和所属销售区域；计算结果值与其相关参数。

## 5.3 完整性约束

对于数据需要人工录入或导入的表，需要附加足够的约束以保证系统数据的完整性。

有如下的方式可采用：

列级约束:

1). 不允许空NOT NULL

2). 缺省值DEFAULT

3). 数据类型（长度，小数位）

4). 检查约束CHECK（列表、like）

5). 唯一值约束UNIQUE

表级约束:

1). 主键约束PRIMARY KEY

2). 关系（外键）约束FOREIGN KEY

3). 规则RULE（列间取值关系）

4). 唯一索引：对于有唯一性要求的非主键列（或列的组合），需要建立唯一索引。

5). 使用触发器：对于较复杂的约束，使用触发器检查和事务回滚完成。

6). 使用存储过程：对于复杂并耗时，不便用触发器实现的，可使用存储过程定时检查，自动或人工排除错误。

## 5.4 触发器

### 5.4.1 使用范围

数据检查和约束：复杂表级约束、表间约束。

记录变更：记录变更，其他程序可以应用进行数据的更新；查询数据的变化历史。

关联更新：对指定的表中的相关列作同步，或进行实时地累计计算

## 5.5 视图

### 5.5.1 用途

弱化使用者与表物理结构间的关联，减少因表结构修改引起的关联对象的维护。

统一数据源。当一种固定的数据组合算法被经常引用时，通过视图可以统一算法、格式，减少算法变化引起关联对象的维护。

增强复用，简化编程。将常用的算法或数据组合固定到视图，编程时引用可简化程序结构，提高变成效率。

### 5.5.2 使用范围

除编辑数据外，对业务数据的引用原则上都要通过视图。

对多处引用的算法或数据组合使用视图，提高变成效率，减少维护量。

对于大数据表，使用分区视图，优化查询性能。

## 5.6 函数

### 5.6.1 用途

需要返回值或列表，便于其他程序引用。

固化程序段便于多次复用。

与视图相比，可使用参数，可使用临时表进行多步骤复杂运算，除返回列表外还可返回值。

与存储过程相比，表函数可以像表一样被引用，值函数可以像常量一样被引用。

### 5.6.2 适用范围

与视图使用同样的目的，但不能用视图实现时。

对重复使用的程序段，提高复用性，统一数据源和算法。

对需要写入永久表的应使用存储过程。

## 5.7 存储过程

### 5.7.1 用途

不需要返回值或列表要其它程序引用，处理结果直接存入表或前台展示。

替代函数编写函数所不能处理的代码。

对重复使用的程序段，提高复用性，统一算法。

适用于处理复杂计算并写入结果表或直接列表显示的情况。

### 5.7.2 事务处理

当对一个以上表进行处理（新增，删除或修改）时，应将这个步骤包含在一个处理事务中，以便出错时回滚，保持数据的完整性。

### 5.7.3 错误处理

对出错信息要统一编码管理，便于准确报告错误及错误的跟踪处理。

### 5.7.4 游标

除非必要请不要使用游标，而使用SELECT字段列表中调用函数的方式代替。

# 6. 性能优化

## 6.1 索引

经常在where字句中引用的列，如果有大数量的不同值可以建索引。只有小数目的不同值的列不宜建索引

聚集索引：因为顺序访问最常见的是按时间范围，所以选择与数据时间相关的列作为聚集索引。不要使用会更新的列。

## 6.2 分区视图

对于数据量巨大的表（暂且规定为百万级），可按年度建表，用分区视图查询。这样做可在不改变代码的前提下加快查询速度。

## 6.3 合理的冗余

完全按照规范化设计的系统几乎是不可能的，除非系统特别的小，在规范化设计后，有计划地加入冗余是必要的。冗余可以是冗余数据库、冗余表或者冗余字段，不同粒度的冗余可以起到不同的作用。冗余可以是为了编程方便而增加，也可以是为了性能的提高而增加。从性能角度来说，冗余数据库可以分散数据库压力，冗余表可以分散数据量大的表的并发压力，也可以加快特殊查询的速度，冗余字段可以有效减少数据库表的连接，提高效率。

## 6.4 主键的设计

主键是必要的，SQL SERVER的主键同时是一个唯一索引，而且在实际应用中，我们往往选择最小的键组合作为主键，所以主键往往适合作为表的聚集索引。聚集索引对查询的影响是比较大的，这个在下面索引的叙述。在有多个键的表，主键的选择也比较重要，一般选择总的长度小的键，小的键的比较速度快，同时小的键可以使主键的B树结构的层次更少。主键的选择还要注意组合主键的字段次序，对于组合主键来说，不同的字段次序的主键的性能差别可能会很大，一般应该选择重复率低、单独或者组合查询可能性大的字段放在前面。

## 6.5 外键的设计

外键是最高效的一致性维护方法，数据库的一致性要求，依次可以用外键、CHECK约束、规则约束、触发器、客户端程序，一般认为，离数据越近的方法效率越高。谨慎使用级联删除和级联更新，级联删除和级联更新作为SQL SERVER 2000当年的新功能，在2005作了保留，应该有其可用之处。因为级联删除和级联更新有些突破了传统的关于外键的定义，功能有点太过强大，使用前必须确定自己已经把握好其功能范围，否则，级联删除和级联更新可能让你的数据莫名其妙的被修改或者丢失。从性能看级联删除和级联更新是比其他方法更高效的方法。

## 6.6 字段的设计

字段是数据库最基本的单位，其设计对性能的影响是很大的。需要注意如下：

1). 数据类型尽量用数字型，数字型的比较比字符型的快很多。

2). 数据类型尽量小，这里的尽量小是指在满足可以预见的未来需求的前提下的。

3). 尽量不要允许NULL，除非必要，可以用NOT NULL+DEFAULT代替。

4). 少用TEXT和IMAGE，二进制字段的读写是比较慢的，而且，读取的方法也不多，大部分情况下最好不用。

5). 自增字段要慎用，不利于数据迁移。

## 6.7 索引的设计

1). 根据数据量决定哪些表需要增加索引，数据量小的可以只有主键。

2). 根据使用频率决定哪些字段需要建立索引，选择经常作为连接条件、筛选条件、聚合查询、排序的字段作为索引的候选字段。

3). 把经常一起出现的字段组合在一起，组成组合索引，组合索引的字段顺序与主键一样，也需要把最常用的字段放在前面，把重复率低的字段放在前面。

4). 一个经常插入更新的表不要加太多索引，因为索引影响插入和更新的速度。

5). 一个多用于查询的表可以适量增加索引，保证查询效率，例如报表系统。