SQL SERVER使用规范

一. 数据库命名规范 3

1. 数据库 3

2. 数据对象 3

3. 字段 3

4. 索引 4

5. 视图 4

二. 表结构 4

1. 字段选择基本准则 4

2. 常见的字段类型 4

3. 常见的字段类型选择： 5

4. 时间戳字段：datachange\_createtime和datachange\_lasttime 5

三. 约束与索引 6

1. 主键 6

2. 不允许使用外键 6

3. NULL属性 6

4. 索引设计准则 6

5. 聚集索引建议 7

四. SQL查询规范 7

1. SQL查询禁令 7

2. SQL查询限制 9

3. SQL查询常用优化 10

五. 架构设计 11

1. 读写分离 12

2. schema解耦 12

3. 数据生命周期 12

4. 分区或分表 12

5. 禁止新增复制分发链路 13

# 一. 数据库命名规范

## 1. 数据库

数据库的定义原则上对应在子系统一级，在不能满足实际需求的情况下也可以对应到具体的应用程序一级：

命名：采用“子系统名称&DB ”或“应用程序名称&DB ”，去除子系统或应用程序名称中的 “.” 符号。命名长度不可超过20个字符。

举例：

酒店产品子系统数据库：HtlProductDB

酒店产品子系统产品房态房价数据库：HtlProductRoomInfoDB

## 2. 数据对象

* 实体表：

命名：表名长度必须不超过30个字符，表命名只能使用26个英文字母、下划线，并且每个单词首字母大写其余小写

举例：O\_orders

* 临时表 ：

命名： \_del+ 当天时间 + 原表名

举例：\_del201103030O\_orders

* 分类状态位表

命名：统一使用 Dictionary

举例：Dictionary

* 订单迁移备份中转表

命名：子系统名\_相关业务名\_TransferOrderID\_Change

举例：Payment\_Casher\_TransferOrderID\_Change

* 订单相关表

订单相关表必须有OrderID字段，避免大量join

## 3. 字段

* 不同系统相同含义的字段使用同一命名，并且两者所有属性必须一致

命名： 字段命名长度必须不超过20个字符 ，只能使用26个英文字母、数字、下划线，并且每个单词首字母大写其余小写

举例： OrderDate

* 当遇到主键字段是自增类型的建议使用表名+ID 的形式

举例： O\_orders 主键： OrderID

* 当字段表示为是否的意思时，建议使用 Is+XXX 的形式

举例：是否删除IsDeleted，是否可用IsEnabled

* 字段命名中不允许出现保留词

## 4. 索引

主键索引使用PK\_表名命名，不可使用联合主键

非聚集索引必须使用idx\_表名\_x

举例： idx\_O\_Orders\_OrderID

## 5. 视图

视图命名：v\_相关内容命名

举例：v\_Orders

# 二. 表结构

## 1. 字段选择基本准则

在满足业务需求的前提下，字段类型越短越好。

* 更少的存储空间
* 更少的磁盘IO
* 更少的网络流量

## 2. 常见的字段类型

* 常见的字段类型—精确数值

|  |  |  |
| --- | --- | --- |
| 数 据 类 型 | 描 述 | 存 储 空 间 |
| bit | 0、1或Null | 1字节(8位) |
| tinyint | 0～255之间的整数 | 1字节 |
| smallint | –32768～32767之间的整数 | 2字节 |
| int | –2147483648～2147483647之间的整数 | 4字节 |
| bigint | –9223372036854775808～ 9223372036854775807 之间的整数 | 8字节 |
| numeric(p,s)或decimal(p,s) | –1038＋1～1038–1之间的数值 | 最多17字节 |
| money | –922337203685477.5808～922337203685477.5807 | 8字节 |
| smallmoney | –214748.3648～214748.3647 | 4字节 |

* 常见的字段类型—近似数值

|  |  |  |
| --- | --- | --- |
| 数 据 类 型 | 描 述 | 存 储 空 间 |
| float[(n)] | –1.79E＋308~–2.23E–308, 0, 2.23E–308~1.79E＋308 | N< =24－4字节 N> 24－8字节 |
| real() | –3.40E＋38~–1.18E–38, 0, 1.18E–38~3.40E＋38 | 4字节 |

## 3. 常见的字段类型选择：

* 字符类型建议采用varchar/nvarchar数据类型
  + 满足数据量增长
  + 减少维护成本
* 金额货币建议采用money数据类型
* 科学计数建议采用numeric数据类型
* 自增长标识建议采用bigint数据类型
* 时间类型建议采用为datetime数据类型
* 禁止使用text、ntext、image老的数据类型
  + 这些类型后续SQL版本将不再支持
* 禁止使用XML数据类型
  + XML数据解析会消耗大量CPU和内存资源
* 禁止使用varchar(max)、nvarchar(max)等数据类型

## 4. 时间戳字段：datachange\_createtime和datachange\_lasttime

* 新建表中的时间戳字段设计时必须指定默认属性：GetDate() NOT NULL
* 已有表新增时间戳字段设计时必须指定默认属性： GetDate() NULL
  + 方便BI取数，方便用户行为分析
  + 方便增量建CACHE
  + 实际业务也大量需要这两个数据

# 三. 约束与索引

## 1. 主键

* 每张表必须有主键，用于强制实体完整性
* 单表只能有一个主键（不允许为空及重复数据）
* 尽量使用单字段主键
* 建议使用聚集类型主键

## 2. 不允许使用外键

* 外键增加了表结构变更及数据迁移的复杂性
* 外键对插入，更新的性能有影响，需要检查主外键约束
* 数据一致性由程序控制

## 3. NULL属性

* 新加的表，所有字段禁止NULL
  + 允许NULL值，会增加应用程序的复杂性。必须得增加特定的逻辑代码，以防止出现各种意外的bug
* 旧表新加字段，需要允许为NULL
  + 避免全表数据更新 ，长期持锁阻塞导致阻塞

## 4. 索引设计准则

* 应该对 WHERE 子句中经常使用的列创建索引
* 应该对经常用于连接表的列创建索引
* 应该对 ORDER BY 子句中经常使用的列创建索引
* 不应该对小型的表（仅使用几个页的表）创建索引，这是因为完全表扫描操作可能比使用索引执行的查询快
* 单表索引数不超过6个
* 不要给选择性低的字段建单列索引
  + 为了维持Btree会带来大量的数据移动
  + 数据库引擎内部更新聚集索引动作会转换为一次删除和一次插入
  + 适合创建索引的字段：ORDERID、UID等
* 充分利用唯一约束
  + 唯一索引给SQL Server提供了确保某一列绝对没有重复值的信息，当查询分析器通过唯一索引查找到一条记录则会立刻退出，不会继续查找索引
* 索引包含的字段不超过5个（包括include列）
  + 索引加快了查询速度，但是却会影响写入性能
  + 一个表的索引应该结合这个表相关的所有SQL综合创建，尽量合并
  + 组合索引的原则是，过滤性越好的字段越靠前，更新频繁的字段放入include列
  + 索引过多不仅会增加编译时间，也会影响数据库选择最佳执行计划
* 聚集索引建议
  + 字段长度小
  + 字段发生的更改小
  + 字段值递增，如自增长，时间字段
  + 字段值唯一（这个是必要条件）
  + 尽量使用单字段主键
  + 根据业务选择有意义的主键
  + 不要修改聚集索引
    - 为了维持Btree会带来大量的数据移动
    - 数据库引擎内部更新聚集索引动作会转换为一次删除和一次插入

# 四. SQL查询规范

## 1. SQL查询禁令

* 禁止在数据库做复杂运算
  + XML解析
  + 字符串相似性比较
  + 字符串搜索（Charindex）
  + 复杂运算在程序端完成
* 禁止使用SELECT \*
  + 减少内存消耗和网络带宽
  + 给查询优化器有机会从索引读取所需要的列
  + 表结构变化时容易引起查询出错
* 禁止在索引列上使用函数或计算
  + 在where子句中,如果索引是函数的一部分,优化器将不再使用索引而使用全表扫描

举例：假设在字段Col1上建有一个索引，则下列场景将无法使用到索引

ABS[Col1]=1

[Col1]+1>9

[Col1] LIKE ‘%abc’

举例：假设在字段Col1上建有一个索引，则下列场景将可以使用到索引：

[Col1]=3.14

[Col1]>100

[Col1] BETWEEN 0 AND 99

[Col1] LIKE ‘abc%’

[Col1] IN(2,3,5,7)

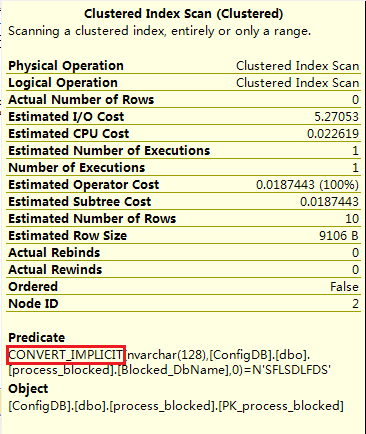
* LIKE查询尽量使用前缀查询
  + like只能使用前缀索引，因此 ：

col like "abc%" 能用上索引

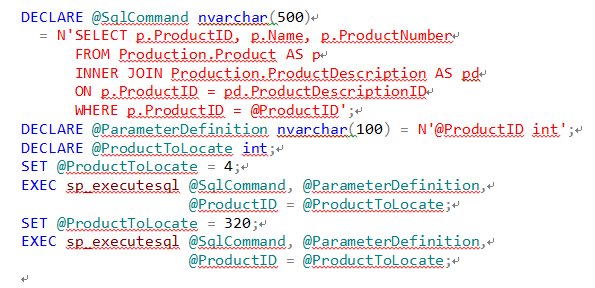
col like "%abc%" 不能用上索引

col like "%abc" 不能用上索引

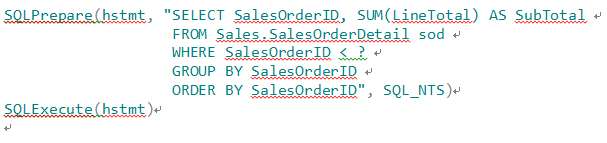
* 禁止使用游标
  + 关系数据库适合集合操作，也就是对由WHERE子句和选择列确定的结果集作集合操作，游标是提供的一个非集合操作的途径。一般情况下，游标实现的功能往往相当于客户端的一个循环实现的功能。
  + 游标是把结果集放在服务器内存，并通过循环一条一条处理记录，对数据库资源（特别是内存和锁资源）的消耗是非常大的。
* 禁止使用触发器
  + 不要使用触发器(Trigger), 在存储过程中实现触发器逻辑
* 禁止在查询里指定索引
  + 随着数据的变化查询语句指定的索引性能可能并不最佳
  + 索引对应用应是透明的，如指定的索引被删除将会导致查询报错，不利于排障
  + 新建的索引无法被应用立即使用，必须通过发布代码才能生效
* 禁止变量/参数/关联字段类型与字段类型不一致
  + 避免类型转换额外消耗的CPU，引起的大表scan尤为严重



* + C#中的string数据类型对应的字符集为UNICODE，在数据库中显示为N’ABC’,如数据库字段为Varchar/Char，则会产生额外的CPU消耗
  + C#中的anistring数据类型对应的字符集为非UNICODE，在数据库中显示为’ABC’，如数据库字段为Varchar/Char，则不会产生额外的CPU消耗
* 禁止非参数化查询
  + 以下方式可以对查询SQL进行参数化：
    - sp\_executesql：

****

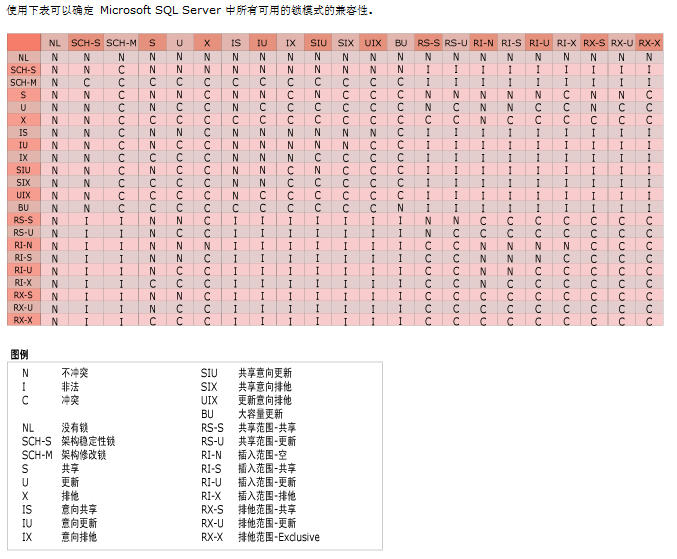
* + - Prepared Queries：



* + - Stored procedures：

## 2. SQL查询限制

* 限制JOIN个数
  + 单个SQL语句的表JOIN个数不能超过5个
  + 过多的JOIN个数会导致查询分析器走错执行计划
  + 过多JOIN在编译执行计划时消耗很大
* 限制SQL语句长度及IN子句个数
  + 在 IN 子句中包括数量非常多的值（数以千计）可能会消耗资源并返回错误 8623 或 8632，要求IN子句中条件个数限制在100个以内
* 限制TVP包含的行数
  + 行数限制在200以内
* 限制大事务操作
  + 只在数据需要更新时开始事务，减少资源锁持有时间
  + 增加事务异常捕获预处理机制
  + 禁止使用数据库上的分布式事务
* 限制递归查询层级
  + 使用 MAXRECURSION 来防止不合理的递归 CTE 进入无限循环
* 限制大量数据查询，使用分页或TOP
  + 合理限制记录返回数，避免IO、网络带宽出现瓶颈
* 除非业务特殊需要，SELECT语句都必须加上NOLOCK
  + 指定允许脏读。不发布共享锁来阻止其他事务修改当前事务读取的数据，其他事务设 置的排他锁不会阻碍当前事务读取锁定数据。允许脏读可能产生较多的并发操作，但其代价是读取以后会被其他事务回滚的数据修改。这可能会使您的事务出错，向用户显示从未提交过的数据，或者导致用户两次看到记录（或根本看不到记录）



* 关闭影响的行计数信息返回
  + 在SQL语句中显示设置Set Nocount On，取消影响的行计数信息返回，减少网络流量

## 3. SQL查询常用优化

* NOT EXISTS替代NOT IN
  + 对于NOT IN与NOT EXISTS，并非等价替换，只有当子查询中不含有NULL值时，二者才会产生同样的结果；若子查询中有NULL值，NOT IN将返回不正确的空集结果，而NOT EXISTS将返回正确的结果。
  + 因为NOT IN (Value1, Value2, NULL)，将返回NOT TRUE或者NOT UNKNOWN(NOT UNKNOWN 等同于UNKNOWN，还是false)，都是非TRUE条件，所以始终返回空集
  + 对于NOT IN, NOT EXISTS，考虑通过外连接，并判断为空来实现，连接的查询条件通过索引查找
* 使用UNION ALL替换UNION
  + UNION会对SQL结果集去重排序，增加CPU、内存等消耗
* 临时表与表变量
  + 临时表具有统计信息，可以在临时表上建立索引，而表变量没有统计信息，不能建索引。临时表对大数据能做更多优化，但维护成本较高，因此临时表适合于返回大的结果集；表变量适合于小的结果集
* 使用本地变量选择中庸执行计划
  + 在存储过程或查询中，访问了一张数据分布很不平均的表格，这样往往会让存储过程或查询使用了次优甚至于较差的执行计划上，造成High CPU及大量IO Read等问题，使用本地变量防止走错执行计划。
  + 采用本地变量的方式，SQL在编译的时候是不知道这个本地变量的值，这时候SQL会根据表格里数据的一般分布，“猜测”一个返回值。不管用户在调用存储过程或语句的时候代入的变量值是多少，生成的计划都是一样的。这样的计划一般会比较中庸一些，不一定是最优的计划，但一般也不会是最差的计划
  + 如果查询中本地变量使用了不等式运算符，查询分析器使用了一个简单的 30% 的算式来预估

Estimated Rows =(Total Rows \* 30)/100

* + 如果查询中本地变量使用了等式运算符，则查询分析器使用：精确度 \* 表记录总数来预估

Estimated Rows = Density \* Total Rows

* 尽量避免使用OR运算符
  + 对于OR运算符，通常会使用全表扫描，考虑分解成多个查询用UNION/UNION ALL来实现，这里要确认查询能走到索引并返回较少的结果集
* 增加事务异常处理机制
  + 应用程序做好意外处理，及时做Rollback。
  + 设置连接属性 "set xact\_abort on"
* 输出列使用二段式命名格式
  + 二段式命名格式：表名.字段名
  + 有JOIN关系的TSQL，字段必须指明字段是属于哪个表的，否则未来表结构变更后，有可能发生Ambiguous column name的程序兼容错误
* SQL中应添加合理的注释，便于定位故障
  + 注释中标识出AppID或源代码路径，便于出现问题时排障
  + 注释中不应添加当前时间等容易变化的信息，避免SQL Server认为是不同的SQL而反复编译

# 五. 架构设计

## 1. 读写分离

* 设计之初就考虑读写分离，哪怕读写同一个库，有利于快速扩容
* 按照读特征把读分为实时读和可延迟读分别对应到写库和读库
* 读写分离应该考虑在读不可用情况下自动切换到写端

## 2. schema解耦

* 禁止跨库JOIN

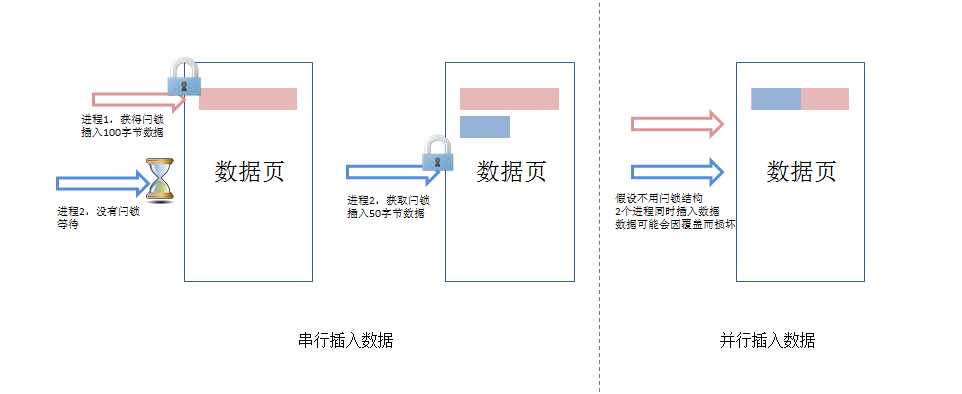
## 3. 数据生命周期

* 根据数据的使用频繁度，对大表定期分库归档
* 主库/归档库物理分离

## 4. 分区或分表

* 日志类型的表应分区或分表
  + 对于大的表格要进行分区，分区操作将表和索引分在多个分区，通过分区切换能够快速实现新旧分区替换，加快数据清理速度，大幅减少IO资源消耗
* 频繁写入的表，需要分区或分表
  + 自增长与Latch Lock

自增长会引发闩锁（Latch Lock），闩锁是sql Server自己内部申请和控制，用户没有办法来干预，用来保证内存里面数据结构的完整性，锁级别是页级锁



* + 减少Latch Lock争用的方法：建立分区表或一张物理表拆分多张物理表

## 5. 禁止新增复制分发链路

* 减少复杂度及维护成本
* 减少故障恢复耗时
* 新版本SQL Server将不再支持复制分发