SQL常用知识点总纲

2014年9月12日 18:00

一、知识点总纲

Level	关键字	说明	备注	示例
1	Primary key	主键	概念理解	
1	Foreign Key	外键	概念理解	
1	Index	索引/聚焦索引/唯一索引	1、索引分类及区别 2、索引与优化 3、索引与死锁	
1	Check	约束/唯一约束		
1	Identity	自增长		
1	ComputeField	计算字段	1、固定值 2、计算公式	
3	Cursor	游标		
2	@Table	表变量		
2	#Table	临时表	1、创建 2、生命周期	
1	Convert/Cast	类型转换函数		
1	NOLOCK	忽略锁	1、概念 2、何时需要 3、与锁的关系	
1	DECLARE	申明变量	1、初始值不可少	DECLARE @s NVARCHAR(MAX)=''
1	View	视图	1、单表视图 2、多表视图 3、视图的索引穿透性	
1	SELECT	查询	1、单表查询 2、多表查询 3、子查询 4、基表概念	
1	Where	条件	1、In 2、Like 3、Where条件与索引的关系	
1	SELECT INTO		1、IDENTITY 2、identity重复问题	
2	CREATE TABLE @temp	创建表变量/临时表		
1	Update	新增	1、单表更新 2、多表更新 3、Update中的@变量应用	
1	Delete	删除	1、单表删除 2、多表删除	
2	Begin Tran/Commit Tran RollBack Tran	事务	1、概念理解 2、事务与锁	
1	NULL	空		
1	0x	空值		
1	ISNULL			
1	NULLIF			
1	CASE			
2	UNION/UNION ALL	联合	1、两者的区别	
2	Timestamp		1、概念理解 2、并发处理	
3	CheckSum			

1	DateDiff	日期相减		
1	DatePart	日期部分		
1	DateAdd	日期相加		
3	EXEC	动态执行SQL		
3	Sp_executesql	动态执行SQL	1、output参数	
1	INNER JOIN	关联	Join与On的关系	
1	LEFT JOIN	左连接		
1	Group By	汇总		
1	Having	汇总后条件		
1	Order BY	排序		
1	MIN	取最小值		
1	MAX	取最大值		
1	SUM	取汇总值		
1	AVG	取平均值		
1	@@RowCount	全局上一影响行数		
1	IN/NOT IN	不在…范围中		
3	Exists/NOT Exists	不存在		
3	CROSS APPLY			
3	Outer apply			
3	RowNumber	行号排序计算		
3	Rank	行号排序计算		
4	Pivot	纵转横		
4	UnPivot	横转纵		
3	fnpbConvertStringToTable	字符串转表		
3	fnpbConvertStringToTwoFieldsTable	字符串转表(两字段)		
3	fnpbConcatString	字符串联		
3	fnpbConcatStringEx	字符串联		
3	sppbNewIden	生成Iden		
3	sppbNewBillNo	生成单据号		
5	For xml	SQL生成XML		
5	Xml.nodes			
5	Xml.Query			
5	Xml.value			
6	with递归查询			
6	常规递归查询			
5	#Table逐行Update			
2	GOTO			
2	TRY..Catch			
3	Sp_rename	对象更名		
3	ExtendProperty	扩展属性		
3	dvColumns	字段列表		
3	dvColumnsEx	字段列表		
3	dvTables	表列表		
3	dvIndex	索引列表		
3	dvCheck	约束列表		
3	dvForeignKeys	外键列表		
1	OBJECT_ID	对象ID		
3	vwpbLockInfo	死锁监测		

3	Lock	死锁	1、死锁的形成机制 2、锁的类型 3、锁与索引的关系 4、如何规避死锁	
2	RAISERROR			
3	FormatString			
3	UniqueValidate	字段唯一校验		
5	sppbPivot	纵转横		
6	SQL.debug	SQL代码调试		
3	COALESCE	返回非空值		COALESCE(a,b,c,d)
5	Grant/Revoke/Deny	授权		
5	sppbQueryDBUserGrantRight	查询授权		
5	sppbBackupDBObjectGrantRight	备份授权		
5	sppbRestoreDBObjectGrantRight	还原授权		
5	FileStream			

二、示例代码与说明

1. Order By
   a. 排序
   b. 正序排序：SELECT * FROM dbo.saUser A(NOLOCK) ORDER BY A.sUserNo
   c. 倒序排序：SELECT * FROM dbo.saUser A(NOLOCK) ORDER BY A.sUserNo DESC
   d. 混合排序：SELECT * FROM dbo.saUser A(NOLOCK) ORDER BY A.iClassId DESC,A.sUserNo
   e. 简写语法：SELECT * FROM dbo.saUser A(NOLOCK) ORDER BY 2 DESC,3
      i. 代表按返回的第二列的值倒序排序+第三列的值正序排序
      ii. 简写语法，不允许出现在正式代码中，仅调试时可使用
2. Identity
   a. 设定字段为自增长
   b. 参数说明：参数 1 为初始值，参数 2 为递增量
      i. IDENTITY(0,1)代表从 0 开始，每次递增 1
      ii. IDENTITY(1,1)代表从 1 开始，每次递增 1
   c. 适用场景：
      i. 创建表/临时表时

```
CREATE TABLE #Order(iIden INT IDENTITY(0,1))
```

      ii. SELECT INTO

```
SELECT iRowId=Identity(INT,0,1),A.*
INTO #User
FROM dbo.saUser A(NOLOCK)
WHERE A.bUsable=1
```

      iii. 注意：
         1) 单个表中只允许出现一个Identity字段
         2) 上例中如果saUser表中原本就有Identity字段，则会报错，上面sql需修改为如下：

```
--iIden原本为Identity，所以需转换后才不与iRowId冲突
SELECT iRowId=Identity(INT,0,1)
,iIden=Convert(INT,A.iIden),A.sUserNo,A.sUserName
INTO #User
FROM dbo.saUser A(NOLOCK)
WHERE A.bUsable=1
```

3. Update
   a. 单表Update
      i.

```
UPDATE dbo.saUser
SET bUsable=1
WHERE iIden=1
```

      ii.

```
UPDATE A
SET bUsable=1
FROM dbo.saUser A
WHERE A.iIden=1
```

      iii. 正式代码中统一用方式 2 语法格式

b. 多表Update
   i. UPDATE A
      SET bUsable=1
      FROM dbo.saUser A
      JOIN dbo.pbCompany B(NOLOCK) ON B.iIden=A.iCompanyId
      WHERE B.iIden=1 AND A.sUserNo='caizh'
   ii. UPDATE A
      SET sUserName=B.sName
      FROM dbo.saUser A
      JOIN dbo.pbUser B(NOLOCK) ON B.sUserNo=A.sUserNo
      WHERE A.bUsable=1 AND B.bUsable=1
   iii. 注意：多表更新时，需要更新的表不添加NOLOCK，其它查询表要添加NOLOCK

4. UPDATE中变量的应用
   a. Update语法中的SET中允许使用@变量，执行顺序是先执行@变量的计算，再执行字段的写入
   b. 示例:要求将年龄大于２０岁的学生的年龄+１
      i. UPDATE A
      SET iAge+=1
      FROM dbo.Student A
      WHERE A.iAge>20
      ii. UPDATE A
      SET @i=A.iAge
      ,iAge=@i+1
      FROM dbo.saUser A
      WHERE A.iAge>20
      iii. UPDATE A
      SET iAge=@i+1
      ,@i=A.iAge
      FROM dbo.saUser A
      WHERE A.iAge>20
      iv. 注意：以上三段ＳＱＬ效果完全等同
   c. 问题1：请取出#File表中的文件名的串联
      i. SELECT sFile='D:\Work\DB\A.sql'
      ii. INTO #File
      iii. INSERT INTO #File
      iv. SELECT sFile='D:\Work\DB\Procedure\B.cs'
      v. 要求使用一条Update语句，去除文件名中的路径，执行后表中两行数据分别为：A.sql,B.cs
      vi. 答案如下：

```
DECLARE @i INT=0,@s NVARCHAR(MAX)=''
UPDATE #File
SET @s=REVERSE(sFile)
,@i=CHARINDEX('\',@s)
,@s=REVERSE(LEFT(@s,@i-1))
,sFile=@s
```

   d. 问题２：出库摊销问题

5. Cursor游标
   a. 游标提供一种数据的单行循环处理方式
   b. 示例代码如下：

```
DECLARE @iIden INT=0,@sUserNo NVARCHAR(50)='',@sUserName NVARCHAR(50)=''
DECLARE curUser CURSOR
FOR
    SELECT TOP 5 iIden,sUserNo,sUserName
    FROM dbo.saUser A(NOLOCK)
    WHERE A.bUsable=1
    ORDER BY A.iIden

OPEN curUser
FETCH NEXT FROM curUser INTO @iIden,@sUserNo,@sUserName
WHILE @@FETCH_STATUS=0
BEGIN
    SELECT @iIden,@sUserNo,@sUserName
    FETCH NEXT FROM curUser INTO @iIden,@sUserNo,@sUserName
```

```
END
CLOSE curUser
DEALLOCATE curUser
```

c. 替代方法 1 ：临时表+Identity的循环

```
DECLARE @iIden INT=0,@sUserNo NVARCHAR(50)='',@sUserName NVARCHAR(50)=''
DECLARE @i INT=1,@iCount INT=0

SELECT TOP 5 iRowId=IDENTITY(INT,1,1)
,A.iIden,A.sUserNo,A.sUserName
INTO #User
FROM dbo.saUser A(NOLOCK)
WHERE A.bUsable=1
ORDER BY A.iIden

SET @iCount=@@RowCount
WHILE @i<=@iCount
BEGIN
    SELECT @iIden=A.iIden,@sUserNo=A.sUserNo,@sUserName=A.sUserName
    FROM #User A
    WHERE A.iRowId=@i

    SELECT @iIden,@sUserNo,@sUserName

    SET @i+=1
END
DROP TABLE #User
```

d. 替代方法 2 ：字符串联数据的循环，此方法仅适用于对单个字符串中的多个值进行循环处理的应用场景

```
SET @sBillId+=','

DECLARE @iStart INT=1
DECLARE @iEnd INT= CHARINDEX(',',@sBillId)

WHILE @iEnd>0
BEGIN
    SET @iBillId=Convert(INT,SUBSTRING(@sBillId,@iStart,@iEnd-@iStart))

    IF @iBillId>0
    BEGIN
        --循环内可取到单个值做任意处理
        --EXEC dbo.sppbBillOperate_SendToAudit @iBillTypeId=@iBillTypeId,@iBillId=@iBillId
            --,@iUserId=@iUserId,@sUserNo=@sUserNo,@iBillTableId=@iBillTableId
    END

    SET @iStart=@iEnd+1
    SET @iEnd=CHARINDEX(',',@sBillId,@iStart)
END
```

6. 表变量@Table
   a. 表变量其实就是一个临时的数据集，它只存在于内存中，速度比临时表更快，但同时占用内存也更多
   b. 表变量与变量一样，是有生命周期的，它的生命周期就存在于定义此表变量的过程/ 函数中
   c. 表变量的可见范围为当前过程，不能像变量一样传递到下一个过程中，或EXEC的动态SQL中，仅能在当前过程中使用
   d. 表变量定义语法如下:
      i. DECLARE @User Table(iIden INT,sUserNo NVARCHAR(20))
7. 临时表#Table
   a. 临时表是临时使用的数据集表，它存在于tempDB中
   b. 临时表的生命周期为当前会话(Session连接)，即当前连接断开时，所有的临时表会被自动清空
   c. 临时表的可见范围为创建后的当前会话中所有代码，可在创建之后的其它过程中或是EXEC动态SQL中使用
   d. 临时表创建的常见方法
      i. CREATE TABLE，需自行定义每个字段的类型

```
CREATE TABLE #User(iIden INT,sUserNo NVARCHAR(20))
```

      ii. SELECT INTO
         1) 从表查询出来的字段，无需定义类型，会自动取原表中的类型及长度

2) 附加出来的新的空字段，则需通过Convert的方式来定义类型及长度
3) SELECT iRowId=Identity(INT,0,1)
,A.iIden,A.sUserNo
,B.sCompanyName
,sStatus=CONVERT(NVARCHAR(50),'')
INTO #User
FROM dbo.saUser A(NOLOCK)
JOIN dbo.pbCompany B(NOLOCK) ON B.iIden=A.iCompanyId
WHERE A.bUsable=1

e. 临时表生命周期示例
i. 示例ＳＱＬ

```
CREATE PROCEDURE dbo.sppbEXEC0
AS
BEGIN
    CREATE TABLE #User(iIden INT,sUserNo NVARCHAR(20))
    EXEC dbo.sppbEXEC1
    CREATE TABLE #Role(iIden INT)
    EXEC dbo.sppbEXEC2
    DROP TABLE #User
    EXEC dbo.sppbEXEC3
    DECLARE @sql NVARCHAR(MAX)='UPDATE #Role SET bUsable=1'
    EXEC(@sql)
END
```

ii. 说明
1) sppbEXEC1中可直接使用临时表#User
2) sppbEXEC2中可直接使用临时表#User、#Role
3) sppbEXEC3中可使用#Role，但不可使用#User，因为临时表#User已被销毁
4) 假设sppbEXEC1中又调用了sppbEXEC4，则sppbEXEC4也可使用#User
5) sppbEXEC1等过程中定义的临时表，在sppbEXEC0中不可使用
6) EXEC动态ＳＱＬ中可直接使用#Role临时表
iii. 临时表在过程间传递时，可通过过程sppbCheckTempTableSchema来校验临时表是否满足需要
1) 如上例中的sppbEXEC1中可使用下面的ＳＱＬ来校验传入的#User是否正确

```
EXEC dbo.sppbCheckTempTableSchema @sTableName='#User'
,@sColumnDefine='iIden INT,sUserNo NVARCHAR(20)'
```

2) 当传入的#User与定义的字段名、类型、长度不符时，会直接RAISERROR报错

8. 类型转换函数Convert/Case
a. Convert与Cast功能基本相同，只是语法略有差异
b. SELECT CONVERT(INT,'100')
c. SELECT CAST('100' AS INT)
d. SELECT CONVERT(NVARCHAR,GETDATE(),120)
9. NOLOCK
a. 忽略锁标签，查询时添加此关键字，说明忽略当前写入锁，允许读取脏数据
b. 添加此关键字，查询的性能将大大增加
c. 要求常规查询时，都必须添加NOLOCK关键字
d. 多表更新时，要更新的表不添加NOLOCK关键字(详见Update节点说明)
e. NOLOCK在并发操作时的作用说明
i. 用户Ａ在执行UPDATE dbo.saUser SET bUsable=1 WHERE A.iIden=100
ii. 用户Ｂ在执行UPDATE dbo.saUser SET bUsable=1 WHERE A.iIden=101
iii. 用户Ｃ执行查询：SELECT * FROM dbo.saUser A(NOLOCK) WHERE A.iIden=100
1) 忽略锁，查询直接返回，速度快
iv. 用户Ｄ执行查询：SELECT * FROM dbo.saUser A WHERE A.iIden=100
1) 未忽略锁，可能会引发查询等待
2) 若iIden上有索引，查询需等待用户Ａ执行完成后才返回(等待行锁解锁)
3) 若iIden上无索引，查询需等待用户Ａ、Ｂ都执行完成后才返回(等待表锁解锁)

10. View视图
a. 视图View是表的一种非存储数据展现方式(极少使用的存储型视图除外)
b. 视图命名全部以vw开头
c. 单表视图
i. CREATE VIEW dbo.vwsmUser
AS
SELECT iUserId=A.iIden,A.sUserNo,A.sUserName,A.iType

```
,sType=CASE A.iType WHEN 1 THEN '系统用户' ELSE '普通用户' END
FROM dbo.saUser A(NOLOCK)
WHERE A.bUsable=1
```

ii. 单表视图在程序开发中可当成单表使用，如同表一样的Update、Delete

1)
```
UPDATE A
SET iType=0
FROM dbo.vwsmUser A
WHERE A.iUserId=0
```
2)
```
DELETE A FROM dbo.vwsmUser A WHERE A.iUserId=0
```

d. 多表视图

i.
```
CREATE VIEW dbo.vwsdOrder
AS
    SELECT A.iIden,A.sOrderNo
    ,B.iMaterialId,B.sMaterialNo,B.sMaterialName,B.nQty
    FROM dbo.sdOrderHdr A(NOLOCK)
    JOIN dbo.sdOrderDtl B(NOLOCK) ON B.iHdrId=A.iIden
    WHERE A.bUsable=1 AND A.iBillStatus=2
```

ii. 多表视图只用于查询，不能直接更新、删除(有Instead of Trigger的除外，但不建议这样处理)

iii. 多表视图的索引穿透性

1) 多表视图的查询时的where条件中字段与视图内部定义的where条件字段，地位等同
2) 以上面的视图为例SELECT * FROM dbo.vwsdOrder WHERE sOrderNo='001'，此时sOrderNo若为sdOrderHdr的索引字段，则会直接通过此索引在表sdOrderHdr中进行查找，而不是先生成视图数据，再在此基础上查找sOrderNo='001'

11. SELECT查询

a. SELECT语句执行顺序

i. FROM-->WHERE-->GROUP BY-->HAVING-->SELECT-->ORDER BY

b. 单表查询

```
SELECT A.iIden,A.sUserNo
FROM dbo.saUser A(NOLOCK)
WHERE A.bUsable=1
```

c. 多表查询

```
SELECT A.iIden,A.sUserNo
,B.iRoleId
FROM dbo.saUser A(NOLOCK)
JOIN dbo.saUserRole B(NOLOCK) ON B.iUserId=A.iIden
WHERE A.bUsable=1
```

d. 子查询

i. 查找姓黄的老师班上的同学名称

1)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
WHERE B.sTeacher LIKE '黄%'
```
2)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
WHERE A.iClassId IN (SELECT iIden FROM dbo.Class A1(NOLOCK) WHERE A1.sTeacher LIKE '黄%')
```
3)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
WHERE EXISTS(SELECT TOP 1 1 FROM dbo.Class A1(NOLOCK)
                WHERE A1.iIden=A.iClassId AND A1.sTeacher LIKE '黄%')
```
4) 以上三条SQL完全等同
5) 注意：使用EXISTS、NOT EXISTS时，内外查询一定要有相关联的条件(如A1.iIden=A.iClassId)

ii. 查找班上没有姓黄的老师的班级的所有学生名称

1)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
LEFT JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
WHERE B.sTeacher LIKE '黄%' AND B.iIden IS NULL
```
2)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
LEFT JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId AND B.sTeacher LIKE '黄%'
WHERE B.iIden IS NULL
```
3)
```
SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
```

WHERE A.iClassId NOT IN (SELECT iIden FROM dbo.Class A1(NOLOCK) WHERE A1.sTeacher LIKE '黄%')

4) SELECT A.iClassId,A.sName
   FROM dbo.Student A(NOLOCK)
   WHERE NOT EXISTS(SELECT TOP 1 1 FROM dbo.Class A1(NOLOCK)
   WHERE A1.iIden=A.iClassId AND A1.sTeacher LIKE '黄%')
5) 以上四条SQL中2、３、４正确且完全等同
6) Sql1错误之处说明
   a) Sql1执行时会先执行A LEFT JOIN B，Where条件sTeacher LIKE '黄%'对针对A LEFT JOIN B的结果集进行筛选的，但A Left JOIN B WHERE B.iIden IS NULL的结果集为空，是没有sTeacher LIKE '黄%'的
   b) Sql2中的sTeacher LIKE '黄%'是针对Class B表进行过滤的，过滤之后再实现A LEFT JOIN B，所以正确

e. 基表
   i. 基表是多表查询中使用的多表绘制的实体类图中最底层的一张表
   ii. 基表在一个查询中必须是唯一的一张表
   iii. 基表对象就是最终返回的对象，即基表的行数＝最终返回行数(如学生JOIN班级，学生为基表，学生表中有多少条记录，就有多少个学生，也即select返回多少行数据)
   iv. 一个查询中出现多个基表的sql，一定是错误的
   v. 查询中出现的基表，与题目要求的目标基表不一致，也一定是错误的
   vi. 当目标基表在实体类图中出现在实际基表的上层的时候，可以通过distinct、group by、或子查询的方式来解决

12. 条件筛选Where/ON
   a. 筛选条件与索引
      i. 查找Iden=100的学生名称
         1) SELECT A.sName
            FROM dbo.Student A(NOLOCK)
            WHERE A.iIden=100
         2) 说明：当A.iIden字段上有索引时，则会通过iIden字段索引在表中进行查找
            当A.iIden字段上无索引时，则整表扫描，逐行扫描直到找到iIden=100的行为止
         3) 结论：最常用的查询条件一定要添加索引，尤其是表中行数较大时，否则整表扫描将会非常慢
      ii. 查找01班年龄大于２０岁的学生名称
         1) SELECT A.sName
            FROM dbo.Student A(NOLOCK)
            JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
            WHERE A.iAge>20 AND B.sNo='001'
         2) 执行顺序
            a) 根据iAge>20在Student学生表中查找，得到结果集X1
            b) 根据No='001'在Class班级表中查找，得到结果集X2
            c) 执行X1 JOIN X2
            d) 执行Select
   b. LIKE与%
      i. 查找所有姓黄的学生
         1) SELECT A.*
            FROM dbo.Student A(NOLOCK)
            WHERE A.sName LIKE '黄%'
         2) 说明：sName字段上有索引时，会根据sName字段索引进行查找
      ii. 查找所有名称中有“明”字的学生
         1) SELECT A.*
            FROM dbo.Student A(NOLOCK)
            WHERE A.sName LIKE '%明%'
         2) 说明：无论sName字段上是否有索引，都会导致整表扫描
         3) 结论：=、In、LIKE 'XX%'索引都有效，但LIKE '%X'会导致索引无效
   c. Where与ON的区别
      i. Where条件是应用于JOIN之后，ON是应用于JOIN之时或之前
      ii. 对INNER JOIN来说，条件放在Where中还是ON时，效果是等同的
      iii. 但对LEFT JOIN来说，条件放在Where中还是ON时，效果是完全不同的
      iv. 查找姓黄的老师班上的同学名称
         1) SELECT A.iClassId,A.sName
            FROM dbo.Student A(NOLOCK)
            JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
            WHERE B.sTeacher LIKE '黄%'
         2) SELECT A.iClassId,A.sName
            FROM dbo.Student A(NOLOCK)

JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId AND B.sTeacher LIKE '黄%'

3) 说明：Sql1、Sql2完全等同

v. 查找班上没有姓黄的老师的班级的所有学生名称

1) SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
LEFT JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
WHERE B.sTeacher LIKE '黄%' AND B.iIden IS NULL

2) SELECT A.iClassId,A.sName
FROM dbo.Student A(NOLOCK)
LEFT JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId AND B.sTeacher LIKE '黄%'
WHERE B.iIden IS NULL

3) 说明：sql1与sql2完全不同

4) Sql1是先执行A LEFT JOIN再执行B.sTeacher LIKE '黄%' AND B.iIden IS NULL，所以得到结果行数＝0

5) Sql2是先执行Class表中搜索(B.sTeacher LIKE '黄%')，得到结果集X，再A LEFT JOIN X

13. Delete
    a. 删除表中的数据
    b. 单表删除
        i. DELETE FROM dbo.saUser WHERE iUserId=0
        ii. DELETE A FROM dbo.saUser A WHERE A.iUserId=0
    c. 多表删除(多表关联，单表删除)
        i. DELETE A
        ii. FROM dbo.Student A
        iii. JOIN dbo.Class B(NOLOC) ON B.iIden=A.iClassId
        iv. WHERE B.sNo='001'
14. 返回非空值ISNULL、COALESCE
    a. SELECT ISNULL(@i,@j)--仅支持两个参数
        i. @i=NULL则返回@j
        ii. @i不等于NULL则返回@i
    b. SELECT COALESCE(@i,@j,@k,@L)--支持若干个参数
        i. @i不为空则返回@i，否则下一步
        ii. @j不为空则返回@j，否则下一步
        iii. @K不为空则返回@K，否则下一步
        iv. @L不为空则返回@L，否则下一步
        v. 依此类推
15. NULLIF
    a. 判断相等则返回NULL，否则返回原值
    b. DECLARE @i INT=1
    c. SELECT NULLIF(@i,2)--@i=1<>2，返回1
    d. SET @i=2
    e. SELECT NULLIF(@i,2)--@i=2，返回NULL
16. CASE
    a. 同开发语言中的switch
    b. SELECT sType=CASE A.iType WHEN 1 THEN '系统用户' WHEN 2 THEN '常规用户' ELSE '其它用户' END
    FROM dbo.saUser A(NOLOCK)
    c. SELECT sType=CASE WHEN A.iAge<16 THEN '少年' WHEN A.iAge<45 THEN '青年' ELSE '老年' END
    FROM dbo.saUser A(NOLOCK)
17. UNION/UNION ALL联合
    a. 将多个数据集联合成为一个数据集
    b. 要联合的多个数据集的列信息必须一致
    c. 返回结果的列名，以第一个数据集的列名为准
    d. UNION将在多个数据集中进行去重处理，UNION ALL直接联合返回
    e. SELECT A.iIden,A.sName,sSex='男'
    FROM dbo.Student A(NOLOCK)
    WHERE A.Sex='男'
    UNION ALL
    SELECT A.iIden,A.sName,[性别]='女'
    FROM dbo.Student A(NOLOCK)
    WHERE A.Sex='女'
    f. 上例中返回的结果集的列名为：iIden、sSex(第二个数据集中指定的列名无效)
18. 日期计算函数
    a. DateDiff两个日期相减

i. SELECT DATEDIFF(DAY,GETDATE(),GETDATE()+11)

b. DATEPART取到指定日期的指定部分
   i. SELECT DATEPART(YEAR,GETDATE())
   ii. SELECT DATEPART(MONTH,GETDATE())
   iii. SELECT DATEPART(DAY,GETDATE())

c. DATEADD日期相加
   i. SELECT DATEADD(DAY,2,GETDATE())

19. OBJECT_ID
   a. 判断ＤＢ中指定的对象是否存在
   b. 此对象可以是表、视图、存储过程、函数等
   c. SELECT OBJECT_ID('saUser')
   d. IF OBJECT_ID('sppbLogLockHistory') IS NOT NULL
      EXEC dbo.sppbLogLockHistory @bReturnCurrLockInfo=1

20. RAISERROR/TRY CATCH
   a. RAISERROR用于抛出异常错误信息
   b. TRY CATCH用于捕获代码中的异常，之后可自行控制是否继续抛出异常
   c. 固定语法RAISERROR(N'出错信息',16,1,文本替换参数)
   d. 无参数时的用法
      i. RAISERROR(N'出错啦',16,1)
   e. 有参数时用法
      i. %s代表nvarchar类型
      ii. %d代表INT类型
      iii. RAISERROR(N'单据号[%s]不存在，或版本号[%d]不正确，请检查',16,1,'001',2)
   f. 正常代码中的RAISERROR只负责抛出异常，与程序开发不同的是，RAISERROR不会导致代码跳转
      i.
```
DECLARE @i INT=1
PRINT '1.@i='+CONVERT(NVARCHAR,@i)
IF @i=1
BEGIN
    RAISERROR('出错啦',16,1)
    PRINT '2.@i='+CONVERT(NVARCHAR,@i)
    SET @i+=1
END
PRINT '3.@i='+CONVERT(NVARCHAR,@i)
```
      ii. 说明：RAISERROR之后的代码还是执行了，输出@i=2
   g. 在TRY CATCH块中的RAISERROR，则抛出异常会立即跳转到CATCH块中
      i.
```
DECLARE @i INT=1
BEGIN TRY
    PRINT '1.@i='+CONVERT(NVARCHAR,@i)
    RAISERROR('出错啦',16,1)
    SET @i+=1
    PRINT '2.@i='+CONVERT(NVARCHAR,@i)
END TRY
BEGIN CATCH
    PRINT '3.@i='+CONVERT(NVARCHAR,@i)
END CATCH
```
      ii. 说明:PRINT1后因为RAISERROR所以直接跳转到PRINT3，PRINT2未执行

21. GROUP BY
   a. 按条件汇总
   b. GROUP BY中出现哪些对象的属性，则基表＝这些汇总对象中的基表对象
   c. 查询每个班的总学生数
      i. SELECT B.sName,iCount=COUNT(*)
      ii. FROM dbo.Student A(NOLOCK)
      iii. JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
      iv. GROUP BY A.iClassId,B.sName
   d. 查询每个班的学生姓氏个数(相同姓氏只算一个姓氏)
      i. SELECT B.sName,iCount=COUNT(DISTINCT LEFT(A.sName,1)
      ii. FROM dbo.Student A(NOLOCK)
      iii. JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
      iv. GROUP BY B.iIden,B.sName

22. 排名函数Row_Number/Rank/DENSE_RANK
   a. 用于对数据集中的每行计算一个顺序号
   b. Row_Number

i. 对分组内的每行返回一个唯一且连续的顺序号,排序字段值相等的行也会返回不同的顺序号
ii. 如A B两行并列第一时，返回的是A=1,B=2,C=3

c. Rank
i. 对分组内的每行返回一个顺序号，排序字段值相等时返回相同的顺序号，且下一顺序号跳号
ii. 如A B两行并列第一时，返回的是A=1,B=1,C=3

d. DENSE_RANK
i. 对分组内的每行返回一个连续顺序号，排序字段值相等时返回相同的顺序号，且下一顺序号不跳号
ii. 如A B两行并列第一时，返回的是A=1,B=1,C=2

e. 示例

```
SELECT iIden=1,iGroup=1,iQty=1
INTO #1

INSERT INTO #1
(iIden,iGroup,iQty)
SELECT 2,1,1
UNION ALL
SELECT 3,1,2
UNION ALL
SELECT 4,2,1
UNION ALL
SELECT 5,2,1
UNION ALL
SELECT 6,2,2

SELECT *
,iRowNumber=ROW_NUMBER() OVER(PARTITION BY iGroup ORDER BY iQty)
,iRank=RANK() OVER(PARTITION BY iGroup ORDER BY iQty)
,iDenseRank=DENSE_RANK()OVER(PARTITION BY iGroup ORDER BY iQty)
FROM #1

DROP TABLE #1
```

结果 消息

	iIden	iGroup	iQty	iRowNumber	iRank	iDenseRank
1	1	1	1	1	1	1
2	2	1	1	2	1	1
3	3	1	2	3	3	2
4	4	2	1	1	1	1
5	5	2	1	2	1	1
6	6	2	2	3	3	2

f. 问题1：查找每班年龄最大的学生姓名（有多个学生年龄并列第一的，任意取一个）
i. 通过排名函数Row_Number函数按每班进行排序，然后得到序号＝1 的就是年龄最大的学生

```
SELECT A.sClassName,A.sStudentName,A.iAge
FROM (
    SELECT sClassName=B1.sName,sStudentName=A1.sName,A1.iAge
    ,iRowNumber=Row_Number() OVER(PARTITION BY A1.iClassId ORDER BY A1.iAge DESC)
    FROM dbo.Student A1(NOLOCK)
    JOIN dbo.Class B1(NOLOCK) ON B1.iIden=A1.iClassId
) A
WHERE A.iRowNumber=1
```

ii. 通过子查询中的ORDER By实现查找每个班中年龄最大的学生的Iden

```
SELECT sClassName=B.sName,sStudentName=A.sName,A.iAge
FROM dbo.Student A(NOLOCK)
JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
WHERE A.iIden=(SELECT TOP 1 A1.iIden FROM dbo.Student A1(NOLOCK)
                WHERE A1.iClassId=A.iClassId
                ORDER BY A1.iAge DESC)
```

iii. 通过CROSS APPLY实现查找每班最大年龄学生Iden

```
SELECT sClassName=A.sName,sStudentName=B.sName,B.iAge
FROM dbo.Class A(NOLOCK)
CROSS APPLY (SELECT TOP 1 A1.sName,A1.iAge FROM dbo.Student A1(NOLOCK)
        WHERE A1.iClassId=A.iIden ORDER BY A1.iAge DESC) B
```

g. 问题2：查找每班年龄最大的学生姓名（有多个学生年龄并列第一的，返回年龄第一的所有学生）
i. 通过排名函数Rank函数按每班进行排序，然后得到序号＝1 的就是年龄最大的学生(可能多个)

```
SELECT A.sClassName,A.sStudentName,A.iAge
```

```
FROM (
    SELECT sClassName=B1.sName,sStudentName=A1.sName,A1.iAge
    ,iRowNumber=Rank() OVER(PARTITION BY A1.iClassId ORDER BY A1.iAge DESC)
    FROM dbo.Student A1(NOLOCK)
    JOIN dbo.Class B1(NOLOCK) ON B1.iIden=A1.iClassId
) A
WHERE A.iRowNumber=1
```

ii. 通过子查询中的ORDER By实现查找每个班中年龄最大的学生的Iden

```
SELECT sClassName=B.sName,sStudentName=A.sName,A.iAge
FROM dbo.Student A(NOLOCK)
JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
WHERE A.iAge=(SELECT TOP 1 A1.iAge FROM dbo.Student A1(NOLOCK)
                WHERE A1.iClassId=A.iClassId
                ORDER BY A1.iAge DESC)
```

iii. 通过Group By先行计算出每班的最大年龄，再查找学生

```
SELECT sClassName=B.sName,sStudentName=A.sName,A.iAge
FROM dbo.Student A(NOLOCK)
JOIN dbo.Class B(NOLOCK) ON B.iIden=A.iClassId
JOIN (
    SELECT A1.iClassId,iAge=MAX(A1.iAge)
    FROM dbo.Student A1(NOLOCK)
    GROUP BY A1.iClassId
) C ON C.iClassId=A.iClassId AND C.iAge=A.iAge
```

23. CheckSum/fnpbEncryptionMD5
    a. 校验值函数，用于将多个值通过函数计算得到一个更简单的唯一标识值，从而方便Join时的关联以及查找
    b. CheckSum的计算更快，且支持任意多个任意类型的参数，但计算得到的值不绝对唯一，有小概率下会导致不同值算出来的校验值相同
    c. fnpbEncryptionMD5是标准的SQLCLR封装诉MD5计算函数，将传入的长字符串计算成为32位的字符串，且确保唯一，即MD5值相同时，则说明计算前的原值一定相同
    d. SELECT CHECKSUM('aa','bb','cc')
    e. SELECT dbo.fnpbEncryptionMD5('aa'+'bb'+'cc')
    f. SELECT iCheckSum=CHECKSUM(A.iIden,A.sUserNo,A.sUserName)
       FROM dbo.saUser A(NOLOCK)
    g. 注：Sql中索引的值存储，就是用类似于CheckSum值的方式存储，不管是单字段索引还是多字段索引都是计算成单值存储
24. EXEC/sp_executesql/sppbExecSql
    a. 动态执行SQL
    b. EXEC
        i. 直接执行完整的SQL，无返回值
        ii. DECLARE @sql NVARCHAR(MAX)='UPDATE #Order SET bUsable=1 WHERE iIden='+Convert(NVARCHAR,@iIden)
            EXEC(@sql)
        iii. DECLARE @sql NVARCHAR(MAX)='UPDATE #Order SET bUsable=1 WHERE iIden=:0'
             SET @sql=dbo.fnpbFormatString(@sql,@iIden)
             EXEC(@sql)
    c. sp_executesql
        i. 可传入@参数，也可通过OUTPUT参数实现数据输出
        ii. DECLARE @sql NVARCHAR(MAX)='UPDATE #Order SET bUsable=1 WHERE iIden=@iIden'
            EXEC sp_executesql @sql,N'iIden INT',@iIden
        iii. DECLARE @sql NVARCHAR(MAX)='SELECT TOP 1 @result=sUserName
             FROM dbo.saUser A(NOLOCK)
             WHERE A.iIden=@iIden'
             EXEC sp_executesql @sql,N'iIden INT,@result NVARCHAR(50) OUTPUT',@iIden,@result OUTPUT
        iv. 参数说明：参数1=Sql文本，参数2= 参数说明，参数3=传入的参数值
    d. sppbExecSql
        i. SQLCLR封装函数，可按顺序直接传入参数，无返回值(仿DBManager)
        ii. DECLARE @sql NVARCHAR(MAX)='UPDATE #Order SET bUsable=1 WHERE iIden=:iIden'
            EXEC dbo.sppbExecSql @sql,@iIden
25. 查询A不在B中
    a. 问题：查找所有未选修物理课的学生(班级<－－学生<－－选课－－>课程)
    b. 思路：查找选修了物理课的学生，然后查找不在这些学生范围内的学生
        i. SELECT A.*
           FROM dbo.Student A(NOLOCK)

```
WHERE A.iIden NOT IN (SELECT iStudentId FROM dbo.SelectCourse A1(NOLOCK)
                      JOIN dbo.Course B1(NOLOCK) ON B1.iIden=A1.iCourseId
                      WHERE B1.sName='物理')
```

ii.
```
SELECT A.*
FROM dbo.Student A(NOLOCK)
WHERE NOT EXISTS (SELECT TOP 1 1 FROM dbo.SelectCourse A1(NOLOCK)
                  JOIN dbo.Course B1(NOLOCK) ON B1.iIden=A1.iCourseId
                  WHERE B1.sName='物理' AND A1.iStudentId=A.iIden)
```

iii.
```
SELECT A.*
FROM dbo.Student A(NOLOCK)
LEFT JOIN (
    SELECT iStudentId
    FROM dbo.SelectCourse A1(NOLOCK)
    JOIN dbo.Course B1(NOLOCK) ON B1.iIden=A1.iCourseId
    WHERE B1.sName='物理'
) B ON B.iStudentId=A.iIden
WHERE B.iStudentId IS NULL
```

26. 字符转表函数
    a. fnpbConvertStringToTable
        i. 将单个字符串转成Table(单个Item列)
        ii. 参数1=传入的字符串，参数2=分隔符

```
SELECT A.* FROM dbo.fnpbConvertStringToTable('a,b,c',',') A
```

	Item
1	a
2	b
3	c

    b. fnpbConvertStringToTableEx
        i. 将单个字符串转成Table，返回两列,ID列(顺序号)、Item列
        ii. 参数1=传入的字符串，参数2=分隔符，输出ID、Item列

```
SELECT A.* FROM dbo.fnpbConvertStringToTableEx('a,b,c',',') A
```

	ID	Item
1	1	a
2	2	b
3	3	c

    c. fnpbConvertStringToTwoFieldsTable
        i. 将单个字符串转成Table，返回两列(sField1,sField2)，即每行返回两个Item
        ii. 参数1=传入的字符串，参数2=行分隔符，参数 3 =行内分隔符
        iii. SELECT A.* FROM dbo.fnpbConvertStringToTwoFieldTable('a,b;c,d',';',',') A
        iv.

	sField1	sField2
1	a	b
2	c	d

27. 字符串联函数fnpbConcatString/fnpbConcatStringEx
    a. 将表中的数据串联成单个字符串，fnpbConcatString默认以逗号分隔，fnpbConcatStringEx可指定分隔符
    b. 查找每个学生的选课的所有课程
        i.
```
SELECT A.iIden,A.sStudentName,sCourseList=dbo.fnpbConcatString(C.sCourseName)
FROM dbo.Student A(NOLOCK)
JOIN dbo.SelectCourse B(NOLOCK) ON B.iStudentId=A.iIden
JOIN dbo.Course C(NOLOCK) ON C.iIden=B.iCourseId
GROUP BY A.iIden
```

    c. 查找每个学生的选课的所有课程(按课程名排序)
        i.
```
SELECT A.iIden,A.sStudentName,sCourseList=dbo.fnpbConcatString(DISTINCT C.sCourseName)
FROM dbo.Student A(NOLOCK)
JOIN dbo.SelectCourse B(NOLOCK) ON B.iStudentId=A.iIden
JOIN dbo.Course C(NOLOCK) ON C.iIden=B.iCourseId
GROUP BY A.iIden
```

    d. 查找每个学生的选课的所有课程(按选课时间排序)
        i.
```
SELECT A.iIden,A.sStudentName,sCourseList=dbo.fnpbConcatString(A.sCourseName)
FROM (
    SELECT TOP 100 A.iIden,A.sStudentName,C.sCourseName
    FROM dbo.Student A(NOLOCK)
    JOIN dbo.SelectCourse B(NOLOCK) ON B.iStudentId=A.iIden
    JOIN dbo.Course C(NOLOCK) ON C.iIden=B.iCourseId
    GROUP BY A.iIden
    ORDER BY B.tCreateTime
```

) A

28. CROSS APPLY/OUTER APPLY
    a. A CROSS APPLY B代表以A为基准来进行B的查询，得到的结果行数A的行数*B.行数
    b. OUTER APPLY与CROSS APPLY的区别等同于LEFT JOIN与INNER JOIN的区别
    c. 示例SQL

```
SELECT iGroup=1,sList=CONVERT(NVARCHAR(100),'1,2,3') INTO #1

INSERT INTO #1 VALUES(2,'1,3,5,7')
INSERT INTO #1 VALUES(3,NULL)

SELECT * FROM #1
SELECT A.*,B.Item
FROM #1 A
CROSS APPLY dbo.fnpbConvertStringToTable(A.sList,',') B

SELECT A.*,B.Item
FROM #1 A
OUTER APPLY dbo.fnpbConvertStringToTable(A.sList,',') B

DROP TABLE #1
```

CROSS APPLy执行结果(注意iGroup=3未输出)

	iGroup	sList	Item
1	1	1,2,3	1
2	1	1,2,3	2
3	1	1,2,3	3
4	2	1,3,5,7	1
5	2	1,3,5,7	3
6	2	1,3,5,7	5
7	2	1,3,5,7	7

OUTER APPLY执行结果(iGroup=3输出)

	iGroup	sList	Item
1	1	1,2,3	1
2	1	1,2,3	2
3	1	1,2,3	3
4	2	1,3,5,7	1
5	2	1,3,5,7	3
6	2	1,3,5,7	5
7	2	1,3,5,7	7
8	3	NULL	NULL

29. GOTO
    a. 代码行跳转命令

```
PRINT 100
GOTO EndLine
PRINT 200--此行将会跳过不执行
ENDLine:
    PRINT 'EndLine'
```

消息

```
100
EndLine
```

30. ExtendProperty
    a. SqlServer中的每个对象，都可以拥有扩展属性，如存储过程、函数、视图、表、表字段等
    b. 当对象销毁时，对象所属的扩展属性也会自动消亡
    c. fnpbGetExtendProperty获取对象扩展属性
        i. 参数1=对象名称，参数2=属性名称

```
--获取函数fnpbFormatString的扩展属性iVersion值
SELECT dbo.fnpbGetExtendProperty('fnpbFormatString','iVersion')
--获取表字段的扩展属性iVersion值
SELECT dbo.fnpbGetExtendProperty('sdOrderHdr.iIden','iVersion')
```

    d. sppbSetExtendProperty设置对象扩展属性
        i. 参数1=对象名称，参数2=属性名，参数3=属性值

```
--设置函数fnpbFormatString的扩展属性iVersion值=9
EXEC dbo.sppbSetExtendProperty @sObject='fnpbFormatString',@sName='iVersion',@sValue=9
--设置表字段的扩展属性
EXEC dbo.sppbSetExtendProperty @sObject='sdOrderHdr.iIden',@sName='iVersion',@sValue=9
```

31. 死锁监控
    a. vwpbLockInfo
        i. 当前ＤＢ中当前存在的锁的查询

ii. SELECT * FROM dbo.vwpbLockInfo

b. sppbLogLockHistory
   i. 记录死锁历史
   ii. 查询当前ＤＢ中当前存在的死锁信息，并写入表dbo.pbLockHistory

c. sppbLogLockHistory_Task
   i. 添加后台任务,每10分钟监测一次死锁

32. 字符串格式化函数FormatString/sppbFormatString
   a. fnpbFormatString
      i. fnpbFormatString，参数1=文本,参数2=:0替换值，文本中仅支持:0
      ii. fnpbFormatString2，参数1=文本,参数2=:0替换值，参数3=:1替换值，文本中仅支持:0、:1
      iii. fnpbFormatString3、fnpbFormatString4、fnpbFormatString5与上面相同，只是支持的参数添加
      iv. SELECT dbo.fnpbFormatString('单据号[:0]不存在',1)
      v. SELECT dbo.fnpbFormatString2('单据类型[:0]单据号[:1]不存在',1,'001')
   b. sppbFormatString
      i. 存储过程，利用参数默认值实现任意多个参数的处理

```
DECLARE @s NVARCHAR(MAX)='单据类型[:]单据号[:1]不存在'
EXEC dbo.sppbFormatString @s OUTPUT,1,'001'
SELECT @s
```

33. UniqueValidate唯一性约束
   a. 查找单位表中的单位名称是否唯一
   b.

```
DECLARE @s NVARCHAR(MAX)=''
    SELECT @s=dbo.fnpbConcatString(A.sUnitName)
    FROM dbo.pbUnit A(NOLOCK)
    JOIN dbo.pbUnit B(NOLOCK) ON B.sUnitName=A.sUnitName
    WHERE A.iIden<>B.iIden

    IF @s<>''
    BEGIN
        RAISERROR(N'单位名称[%s]重复，请检查',16,1,@s)
        RETURN
    END
```

   c.

```
DECLARE @s NVARCHAR(MAX)=''
    SELECT @s=dbo.fnpbConcatString(A.sUnitName)
    FROM dbo.pbUnit A(NOLOCK)
    GROUP BY A.sUnitName
    HAVING COUNT(*)>1

    IF @s<>''
    BEGIN
        RAISERROR(N'单位名称[%s]重复，请检查',16,1,@s)
        RETURN
    END
```