传智播客 www.itcast.cn —高级软件人才实作培训专家

SQL进阶

讲师：伍湖

CASE函数用法1

- 等值判断，相当于switch case

```
CASE expression
 WHEN value1 THEN returnvalue1
 WHEN value2 THEN returnvalue2
 WHEN value3 THEN returnvalue3
ELSE defaultreturnvalue
END
```

- 例子：论坛中用户的等

```
case level
   when 1 then '骨灰'
   when 2 then '大虾'
   when 3 then '菜鸟'
end
```

- 相当于Switch，只能等值判断。

CASE函数用法2

```
CASE
    WHEN condition1 THEN returnvalue1
    WHEN condition 2 THEN returnvalue2
    WHEN condition 3 THEN returnvalue3
    ELSE defaultreturnvalue
END
```

相当于if...else if...else....（可以进行区间判断）

例子：百分制转换为素质教育

注意：then后面返回的数据类型要一致， returnvalue1、returnvalue2、 returnvalue3的数据类型必须一致。

练习1

- 表中有A B C三列,用SQL语句实现：当A列大于B列时选择A列否则选择B列，当B列大于C列时选择B列否则选择C列。

- 在订单表中，统计每个销售员的总销售金额，列出销售员名、总销售金额、称号（>6000金牌，>5500银牌,>4500铜牌，否则普通）

  Id,Name,TotalMoney 表名：Sale（id,name,salemoney）

  1, 小张 ，10000

  2, 小王， 7000

  3, 小刘， 200

练习2

单号 金额
Rk1 10
Rk2 20
Rk3 -30
Rk4 -10
将上面的表输出为如下的格式:
单号 收入 支出
Rk1 10 0
Rk2 20 0
Rk3 0 30
Rk4 0 10

```
select  case
          when 金额 >0 then 金额
          else 0
           end as 收入,
         case
          when 金额 <0 then abs(金额)
          else 0
          end as 支出
from 表
```

子查询

- 把一个查询的结果在另一个查询中使用就叫子查询。(将一个查询语句做为一个结果集供其他SQL语句使用)
- 例如:
  - Select * from (select col1,col2 from tab) as t
- 子查询基本分类:
  - 独立子查询
    - 子查询可以独立运行
  - 相关子查询
    - 子查询中引用了父查询中的结果
- In、exists、not in、not exists

子查询

- 就像使用普通的表一样，被当作结果集的查询语句被称为子查询。所有可以使用表的地方几乎都可以使用子查询来代替。 SELECT * FROM (SELECT * FROM student where sAge<30) as t
- 重做Union all的一道题。要求在一个表格中查询出学生的英语最高成绩、最低成绩、平均成绩.
  - select (select max(english) from score),(select min(english) from score),(select avg(english) from score)
- 只有返回且仅返回一行、一列数据的子查询才能当成单值子查询。下面的是错误的：SELECT 1 ,2,(SELECT english FROM score)
- 查询高二二班的所有学生
  - select * from student where sClassId = (select cId from class where cName='高二二班')
- 子查询在以下条件时必须只能是一个列的一个值：
  - 当子查询跟随在=、!=、<、<=、>、>= 之后，或子查询用作表达式

子查询

- 如果子查询是多行**单列**的子查询，这样的子查询的结果集其实是一个集合。可以使用in关键字代替=号
- 查询高一一班和高二一班的所有学生
  - select * from student where sClassId in
  - (select cId from class where cName='高一一班' or cName='高二一班')
  - 等价于：备注1.
- 查询刘关张的成绩
  - select studentId,english from score where studentId in
  - (select sId from student where sName='刘备' or sName = '关羽' or sName='张飞')
- 快速实现删除多个学生
  - delete from student where sId in
  - (select sId from student where sName='刘备' or sName = '关羽' or sName='张飞')

- 数据库中分页的实现,见备注。
  - 每页5条，数据，查询第4页的数据。
- 上面是sql 2000以前的实现方式。SQLServer2005后增加了Row_Number函数简化实现。
- 限制结果集。返回第3行到第5行的数据（ROW_NUMBER 不能用在where子句中，所以将带行号的执行结果作为子查询，就可以将结果当成表一样用了）：
  - select * from
  - (select row_number() over (order by sId asc) as num,* from student) as s
  - where s.num between 6 and 10 order by sId asc

子查询练习

- 使用Row_number over()实现分页
- 查询年龄比“廖杨”大的学员，显示这些学员的信息
- 查询二期班开设的课程
- 查询参加最近一次“office”考试成绩最高分和最低分

表连接Join

- 案例1：查询所有学生的姓名、年龄及所在班级
- 案例2：查询年龄超过20岁的学生的姓名、年龄及所在班级
- 案例3：查询学生姓名、年龄、班级及成绩
- 案例4：查询所有学生(参加及未参加考试的都算)及成绩
- 案例5：请查询出所有没有参加考试（在成绩表中不存在的学生）的学生的姓名。
- Inner Join、Left Join、Right Join
- (*)Cross Join 交叉连接 笛卡尔积 其它连接的基础

	sName	sAge	cName
1	曹操	22	高一三班
2	夏侯惇	22	高一三班
3	华佗	50	高一三班
4	甄姬	18	高一三班

	sName	sAge	cName	english
1	曹操	22	高一三班	100
2	夏侯惇	22	高一三班	60
3	华佗	50	高一三班	NULL
4	甄姬	18	高一三班	80
5	诸葛亮	18	高二三班	NULL

- 练习1：查询所有英语及格的学生姓名、年龄及成绩
- 练习2：查询所有参加考试的(english分数不为null)学生姓名、年龄及成绩
- 练习3：查询所有学生(参加和未参加考试)的学生姓名、年龄、成绩，如果没有参加考试显示缺考,如果小于60分显示不及格

	sName	sAge	(无列名)
1	曹操	22	100
2	夏侯惇	22	60
3	华佗	50	缺考
4	甄姬	18	80

视图概述

- 回顾数据怎么存储的
- **视图**是一张**虚拟表**，它表示一张表的部分数据或多张表的综合数据，其结构和数据是建立在对表的查询基础上
- **视图**在操作上和**数据表**没有什么区别，但两者的差异是其**本质是不同**:数据表是实际存储记录的地方，然而视图并不保存任何记录。
- 相同的**数据表**，根据不同用户的不同需求，可以创建不同的视图（不同的查询语句）
- 视图的目的是方便查询，所以一般情况下不对视图进行增改，不能删
- 优点:
  - 筛选表中的行
  - 防止未经许可的用户访问敏感数据
  - 降低数据库的复杂程度

- 普通视图
  - 并不存储数据（虚拟表），访问的是真实表中的数据
- 使用视图注意事项：
  - 1.视图中的查询不能使用order by，除非指定了top语句。
    - 视图被认为是一个虚拟表，表是一个集合，是不能有顺序的。而order by 则返回的是一个有顺序的，是一个游标。
    - 在视图中使用select top percent + order by 问题。
  - 如果指定列名，则列名必须唯一（使用*不考虑）
  - create view vw_name as 后不能跟begin end.
- （*）索引视图
  - 在视图上创建唯一聚集索引
  - 数据会保存在数据库中而不是引用表中的数据。

视图

- create view vw_tab
- as
- select top 100 percent * from tab order by col1 desc
  - 以上语句执行完毕以后查询select * from vw_tab也不会排序
  - 与select * from vw_tab order by col1 desc不一样
  - top已经取得了上次order by 的结果前几个结果
  - 并且top输出的结果没有再次排序
  - 所以无法保证输出的结果是desc排序后的结果
- 视图的增改查

局部变量_先声明再赋值

声明局部变量

DECLARE @变量名 数据类型

DECLARE @name varchar(20)

DECLARE @id int

赋值

SET @变量名 =值 --set用于普通的赋值

SELECT @变量名 = 值 --用于从表中查询数据并赋值，，可以一次给多个变量赋值

例如：

```
SET @name= '张三'
SET @id = 1
SELECT @name = sName FROM student WHERE sId=@id
```

输出变量的值

- SELECT 以表格的方式输出,可以同时输出多个变量
- PRINT 以文本的方式输出，一次只能输出一个变量的值

  - SELECT @name,@id
  - PRINT @name
  - PRINT @id
  - print @name,@id --错误！！

- 使用变量实现：
  - 查询参加最近一次“office”考试成绩最高分和最低分

变量种类

变量分为：

- 局部变量：
  - 局部变量必须以标记@作为前缀，如@Age int
  - 局部变量：先声明，再赋值
- 全局变量（系统变量）：
  - 全局变量必须以标记@@作为前缀，如@@version
  - 全局变量由系统定义和维护，我们只能读取，不能修改全局变量的值

全局变量

变量	含义
@@ERROR	最后一个T-SQL错误的错误号
@@IDENTITY	最后一次插入的标识值
@@LANGUAGE	当前使用的语言的名称
@@MAX_CONNECTIONS	可以创建的同时连接的最大数目
@@ROWCOUNT	受上一个SQL语句影响的行数
@@SERVERNAME	本地服务器的名称
@@TRANSCOUNT	当前连接打开的事务数
@@VERSION	SQL Server的版本信息

IF ELSE

```
IF(条件表达式)
  BEGIN --相当于C#里的{
    语句1
      ……
  END --相当于C#里的}
ELSE
 BEGIN
    语句1
    ……
  END
```

例：

计算平均分数并输出，如果平均分数超过60分输出成绩最高的三个学生的成绩，否则输出后三名的学生

```
declare @avg float
select @avg = avg(english) from score
print '平均分数' + convert(varchar(20),@avg)
if(@avg > 60)
Begin
        print '前三名 ‘
        select top 3 sName,english from student inner join score on student.sId=score.studentId order by english desc
        End
Else
begin
        print '后三名'
        select top 3 sName,english from student inner join score on student.sId=score.studentId order by english asc
end
```

WHILE循环

```
WHILE(条件表达式)
  BEGIN --相当于C#里的{
    语句
    ……
    continue --退出本次循环
    BREAK    --退出整个这一层循环
  END --相当于C#里的}
```

--计算1-100之间所有奇数的和。

例:

如果english不及格的人超过半数(考试题出难了)，则给每个人增加2分,循环加，直到不及格的人数少于一半。

把所有未及格的人的成绩都加及格，超过100的按100计算

事务-为什么需要事务

如，转账问题：

假定钱从A转到B，至少需要两步：

- A的资金减少
- 然后B的资金相应增加

•update bank set balance=balance-1000 where cid='0001'

•update bank set balance=balance + 1000 where cid='0002'

--查看结果。

SELECT * FROM bank

注意约束：金额不能小于10

会否出问题？

什么是事务(Transaction)

- 事务：同生共死
- 指访问并可能更新数据库中各种数据项的一个程序执行单元(unit)--也就是由多个sql语句组成，必须作为一个整体执行 ＡＣＩＤ
- 这些sql语句作为一个整体一起向系统提交，要么都执行、要么都不执行
- 语法步骤：
  - 开始事务：BEGIN TRANSACTION　　开启事务
  - 事务提交：COMMIT TRANSACTION　--提交操作
  - 事务回滚：ROLLBACK TRANSACTION --取消操作
- 判断某条语句执行是否出错：
  - 全局变量@@ERROR;
  - @@ERROR只能判断当前一条T-SQL语句执行是否有错，为了判断事务中所有T-SQL语句是否有错，我们需要对错误进行累计；

  例：SET @errorSum=@errorSum+@@error

存储过程

- **存储过程---就像数据库中运行方法(函数)**
- 和C#里的方法一样，由存储过程名/存储过程参数组成/可以有返回结果。
- 前面学的if else/while/变量/insert/select 等，都可以在存储过程中使用
- **优点：**
  执行速度更快 – 在数据库中保存的存储过程SQL语句都是编译过的
  允许模块化程序设计 – 类似方法的复用
  提高系统安全性 – 防止SQL注入
  减少网络流通量 – 只要传输 存储过程的名称
- 系统存储过程
  - 由系统定义，存放在master数据库中
  - 名称以“sp_”开头或”xp_”开头,自定义的存储过程可以以usp_开头。
- **自定义存储过程**
  - 由用户在自己的数据库中创建的存储过程

系统存储过程

系统存储过程	说明
sp_databases	列出服务器上的所有数据库。
sp_helpdb	报告有关指定数据库或所有数据库的信息
sp_renamedb	更改数据库的名称
sp_tables	返回当前环境下可查询的对象的列表
sp_columns	回某个表列的信息
sp_help	查看某个表的所有信息
sp_helpconstraint	查看某个表的约束
sp_helpindex	查看某个表的索引
sp_stored_procedures	列出当前环境中的所有存储过程。
sp_password	添加或修改登录帐户的密码。
sp_helptext	显示默认值、未加密的存储过程、用户定义的存储过程、触发器或视图的实际文本。

创建自定义存储过程

- 定义存储过程的语法

```
CREATE  PROC[EDURE]  存储过程名
@参数1  数据类型 = 默认值,
@参数n  数据类型 = 默认值 OUTPUT
AS
 declare @num int ---方法里面定义的局部变量
  SQL语句
```

- 参数说明：
  - 参数可选（有默认值）
  - 参数分为输入参数、输出参数
  - 输入参数允许有默认值，可以不传入值---与C#一样
- EXEC 过程名 [参数]

调用带参数的存储过程

无参数的存储过程调用：

- **Exec usp_upGrade**

有参数的存储过程两种调用法：

- **EXEC usp_upGrade2 60,55 ---按次序**
- **EXEC usp_upGrade2 @english=55,@math=60 --参数名**

参数有默认值时：

- **EXEC usp_upGrade2 --都用默认值**
- **EXEC usp_upGrade2 1 --第一个用默认值**
- **EXEC usp_upGrade2 1,5 --不用默认值**
- **若想英语的用默认值，数学的及格分数改了怎么办？**
  指定参数名

编写存储过程

- 作业：考试题出难了，降低及格分数线60,(将该题目封装为一个存储过程。
  - 条件：及格人数 大于 总人数一半
  - 结果：输出最后的分数线
- 作业：编写存储过程usp_upGrade
- 要求传入参数:@pass float
  - 给没及格的人提分，一直加到及格人数大于总人数一半
  - 结果：输出调整之后的学员姓名及分数

存储过程中使用输出参数

- 如果希望在加分的过程中想输出总共加了多少次分
- 输出参数关键字：OUTPUT
- declare **@a** int
- exec **usp_pp @canshu= @a** output
- print **@a**

通过ado.net执行存储过程

- 实现分页的存储过程（*）
- 案例：使用存储过程、事务、winform实现转账。
- 编写存储过程版的SQLHelper

补充

- 问题：
- new SqlParameter(“参数名”，“值”);
- new SqlParameter(“@id”,0);有问题?
- 报错信息：参数化查询 ‘(@tid bigint)insert into tblT1 values(@tid)’ 需要参数 ‘@tid’，但未提供该参数。
- 2.多层开发的时候（多个项目），app.config文件的位置。

触发器

- 触发器的作用：
  - 自动化操作，减少了手动操作以及出错的几率。
- 触发器是一种特殊类型的存储过程，它不同于前面介绍过的一般的存储过程。【在SQL内部把触发器看做是存储过程但是不能传递参数】
- 一般的存储过程通过存储过程名称被直接调用，而触发器主要是通过事件进行触发而被执行。
- 触发器是一个功能强大的工具，在表中数据发生变化时自动强制执行。触发器可以用于SQL Server约束、默认值和规则的完整性检查，还可以完成难以用普通约束实现的复杂功能。
- 那究竟何为触发器？在SQL Server里面也就是对某一个表的一定的操作，触发某种条件，从而执行的一段程序。触发器是一个特殊的存储过程。

常见的触发器

- DML触发器
  - Insert、delete、update（不支持select）
  - after触发器(for)、instead of触发器（不支持before触发器）
- (*)DDL触发器
  - Create table、create database、alter、drop….

inserted表与deleted表

- inserted表与deleted表是干什么的?
- inserted表包含新数据
  - insert、update触发器会用到
- deleted表包含旧数据
  - delete、update触发器会用到

- After触发器：
  - 在语句执行完毕之后触发
  - 按语句触发，而不是所影响的行数，无论所影响为多少行，只触发一次。
  - 只能建立在常规表上，不能建立在视图和临时表上。(*)
  - 可以递归触发，最高可达32级。
  - update(列)，在update语句触发时，判断某列是否被更新，返回布尔值。
  - 介绍如何编写after触发器。
- instead of触发器
  - 用来替换原本的操作
  - 不会递归触发
  - 可以在约束被检查之前触发
  - 可以建在临时表和视图上（*）
  - 介绍如何编写instead of 触发器

常用语法

```
CREATE TRIGGER triggerName ON 表名
after(for)（for与after都表示after触发器） | instead of
 UPDATE|INSERT|DELETE（insert,update,delete）
AS
begin
…
end
```

触发器-插入

```
CREATE TRIGGER tr_updateStudent ON score  --相当于外键检查约束
after INSERT -- 后置的新增触发器
AS
Begin
    declare @sid int,@scoreid int--定义两个变量
    select @sid = sId,@ scoreid=id from inserted--获得新增行的数据
    if exists(select * from student where sid=@sid)--判断分数学员是否存在
        print  '插入成功'
    else --如果不存在，则把更新增成功的分数记录给删除掉
        delete from score where sid = @scoreId
End
Insert into score (studentId,english) values(100,100)
```

触发器-删除

```
CREATE TRIGGER tr_deleteStudent ON student
for delete
AS
begin
    insert into backupStudent select * from deleted
End

    Delete from student where sId=1
```

触发器使用建议：

- 尽量避免在触发器中执行耗时操作，因为触发器会与SQL语句认为在同一个事务中。（事务不结束，就无法释放锁。）
- 避免在触发器中做复杂操作，影响触发器性能的因素比较多（如：产品版本、所使用架构等等），要想编写高效的触发器考虑因素比较多（编写触发器容易，编写复杂的高性能触发器难！）。

索引Index(*)

- **全表扫描**：对数据进行检索（select）效率最差的是全表扫描，就是一条条的找。
- 如果没有目录，查汉语字典就要一页页的翻，而有了目录只要查询目录即可。为了提高检索的速度，可以为经常进行查询的列添加索引，相当于创建目录。
- 创建索引的方式，在表设计器中点击右键，选择“索引/键”→添加→在列中选择索引包含的列。
- 使用索引能提高查询效率，但是索引也是占据空间的，而且添加、更新、删除数据的时候也需要同步更新索引，因此会降低Insert、Update、Delete的速度。只在经常检索的字段上(Where)创建索引。
- （*）即使创建了索引，仍然有可能全表扫描，比如like、函数、类型转换等。
- --删除索引
- drop index **T8.IX_T8_tage**

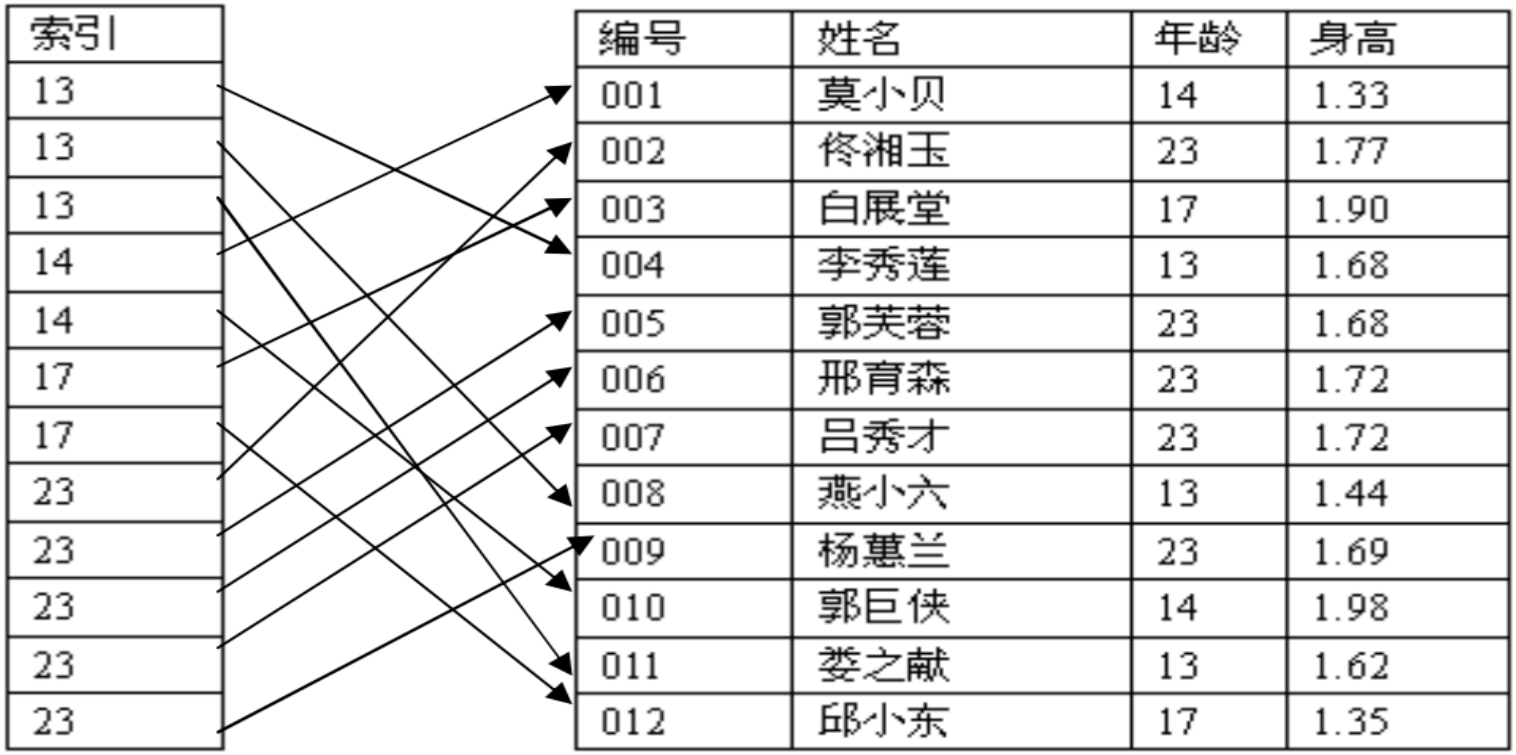
- 索引意味着排序，排序后则可更高效的查询。

无索引的表就是一个无序的行集。比如下面的人员表中有一些数据：

编号	姓名	年龄	身高
001	莫小贝	14	1.33
002	佟湘玉	23	1.77
003	白展堂	17	1.90
004	李秀莲	13	1.68
005	郭芙蓉	23	1.68
006	邢育森	23	1.72
007	吕秀才	23	1.72
008	燕小六	13	1.44
009	杨蕙兰	23	1.69
010	郭巨侠	14	1.98
011	娄之献	13	1.62
012	邱小东	17	1.35

按年龄建索引

如果我们为年龄列创建一个索引，注意这里的索引所采用的值是排序的：



索引
13
13
13
14
14
17
17
23
23
23
23
23

编号	姓名	年龄	身高
001	莫小贝	14	1.33
002	佟湘玉	23	1.77
003	白展堂	17	1.90
004	李秀莲	13	1.68
005	郭芙蓉	23	1.68
006	邢育森	23	1.72
007	吕秀才	23	1.72
008	燕小六	13	1.44
009	杨蕙兰	23	1.69
010	郭巨侠	14	1.98
011	娄之献	13	1.62
012	邱小东	17	1.35

(*) 临时表

- 局部临时表
  - create table #tbName(列信息);
  - 表名前缀#
  - 只在当前会话中有效，不能跨连接访问
  - **作用域范围类似C#：**
    - **如果直接在连接会话中创建的，则当前连接断开后删除，如果是在存储过程中创建的则当前存储过程执行完毕后删除**
- 全局临时表
  - create table ##tbName(列信息);
  - 表名前缀##
  - 多个会话可共享全局临时表
  - 当创建全局临时表的会话断开，并且没有用户正在访问全局临时表时删除
- (*) 表变量：declare @varT1 table(col1 int,col2 char(2));//存储更小量的数据，比临时表有更多的限制。
- 临时数据都存储在tempdb,当服务重新启动的时候，会重建tempdb.