# 数据库

## 数据库的基本概念及操作

### 概念

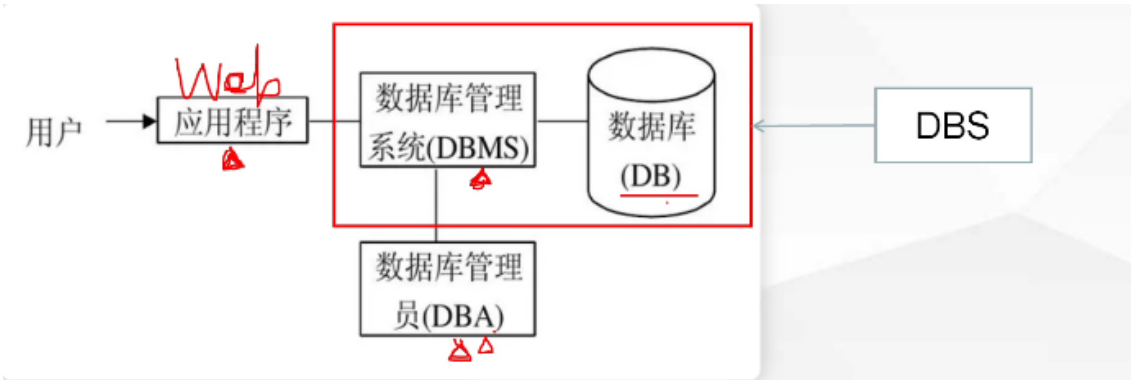
- 数据库的定义

  数据库(Database,简称DB)是**长期储存**在计算机内、**有组织**的、**可共享**的**大量**数据的集合。

- 数据库的基本特征

  1. 数据按一定的数据模型组织、描述和储存
  2. 可为各种用户共享
  3. 冗余度较小
  4. 数据独立性较高
  5. 易扩展

- 数据(Data):的定义
  - 定义：描述事物的符号记录，是数据库中存储的基本对象
  - 种类：文字、图形、图象、声音。
- 数据的记录：计算机中表示和存储数据的一种格式或一种方法
  - 例子：(李明，男，1998，江苏，计算机系，2017)
  - 语义：学生姓名、性别、出生年月、籍贯、所在系别、入学时间
- 数据库管理系统
  - 什么是DBMS(数据库管理系统-数据库的核心)
    - 位于用户与操作系统之间的一层数据管理软件。
    - 是系统软件，是一个大型复杂的软件系统
- DBMS的用途
  - 科学地组织和存储数据、高效地获取和维护数据
- 数据库系统(Database System,简称DBS)
  - 定义：是指在计算机系统中引入数据库后的系统构成。常常把**数据库系统**简称为数据库。
  - 构成：由数据库、数据库管理系统（及其开发工具）、应用程序、数据库管理员（和用户)构成。
  - 注：DBS(数据库系统)=DB(数据库)+DBMS(数据库管理系统)
- 数据库系统(DBS)数据库(DB)、数据库管理系统(DBMS)的关系




- 数据库在计算机系统中的位置


- 数据库系统发展
  - 人工管理阶段(40年代中-50年代中)
  - 文件系统阶段(50年代末-60年代中)
    - 数据共享性差，冗余度大
    - 数据独立性差
  - 数据库系统阶段(60年代末-现在)
    - 数据共享性高，冗余度低且容易扩充
    - 高度的物理独立性 和 一定的逻辑独立性
    - 整体结构化
    - 数据独立性高

## SQLServer2012简介

- 版本
  - √企业版(Enterprise)
  - √商业智能(Business intelligence)
  - √标准版(Standard)
  - √Web版
- SQLServer2012数据库相关概念
  - 4个系统数据库（安装完数据库后自动安装）
    - master数据库：系统最重要的数据库，记录了所有系统信息。如所有的登录信息、系统设置信息、SQL Server的初始化信息和其它系统数据库及用户数据库的相关信息。
    - model数据库：是模板数据库，为新建立的数据库提供模板和原型。
    - tempdb数据库：是一个临时数据库，它为所有的临时表、临时存储过程及其它临时操作提供存储空间。
    - msdb数据库：是代理服务数据库，为其警报、任务调度和记录操作员的操作提供存储空间。
  - 数据库文件和日志
    1. **主数据文件**：包含数据库的启动信息，用户数据和对象存储在此文件中，扩展名为**.mdf(有且只有一个)**
    2. **次要数据文件**：可选，扩展名为**.ndf(可以没有可以有多个)**
    3. **事务日志文件**：保存用于恢复数据库的日志信息，扩展名**.ldf(至少包含一个)**

       **提示：虽然SQL Server2012不强制这3种类型文件必须使用带mdf、ndf和ldf扩展名，但使用它们指出文件类型是个良好的文件命名习惯。**
       **注：以上几种文件都放在文件组中，主文件和日志文件必须有，其中主文件只能有一个，其他类型文件可以有多个**

- 文件组
  - 文件组分**主文件组**和**用户定义的文件组。**
  - 主文件组（系统自动创建）
    主文件组包含**主数据文件(.mdf)、次数据库文件（如果没有为次数据文件分配其他的文件组)**和任何没有明确指派给其它文件组的其它文件。
  - 用户定义的文件组
    用户定义文件组是在create database或alter database语句中，使用FILEGROUP关键字指定的文件组。
    **说明：日志文件不存在于任何文件组中。**

# 数据库操作

- 创建数据库 语法

```
CREATE DATABASE 数据库名;
on primary  -- 主数据文件  只能有一个
(
name = mydb,   -- 名字 -- 自定义最少有名字
filename ='D:\data\mydb.mdf',  -- 位置 文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =unlimited -- 增长方式 百分比 绝对值 unlimited表示不限制
),
filegroup 文件组名字 ( -- 次数据文件 可以定义一个用户定义的文件组
name = mydb_sec,   -- 名字
filename ='D:\data\mydb_sec.ndf',  -- 位置文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =10% -- -- 增长方式 百分比 绝对值
) -- 没有逗号
log on    -- 日志文件 最少一个
(
name = mydb_log, -- 名字
filename ='D:\data\mydb_log.ldf', -- 位置
size =2,  -- 大小
maxsize=10, -- 最大大小
filegrowth =1 -- 每次增长多少
)
```

  - **maxsize:指定文件可增长到的最大值。如果没有指定，则文件可以不断增长直到充满磁盘 如果是unlimited表示不限制**
  - **filegrowth:指定文件每次增加容量的大小当指定数据为“0“时，表示文件不增长**
  - **filegroup:定义文件组写在次数据前面 后名接用户自定义文件组名字**
  - 创建最简单的数据库

```
create database 数据库名称
```

  - 最简单的数据库语句，默认创建一个主数据文件一个日志文件
    - 主数据文件默认名称为**数据库名.mdf**
    - 日志文件默认名称为**数据库名_log.ldf**
- 例子

```
create database MyDb
on primary (
size=20,
name = MyDb,
maxsize=unlimited, -- unlimited 最大尺寸
filegrowth=10%,
filename='D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb.mdf'
),
(
name=MyDb_sec,
size=20,
maxsize=200, -- unlimited 最大尺寸
filegrowth=10%,
filename='D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb_sec.ndf'
)
log on
(
name=MyDb_log,
size=20,
maxsize=200, -- unlimited 最大尺寸
filegrowth=10%,
filename='D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb_log.ldf'
)
```

- 修改数据库
- 修改数据库名称 语法

```
alter database 数据库名 modify name=数据库新名字
```

- 例子

```
alter database MyDb modify name=mydd
```

- 增加数据文件 语法

```
alter database 数据库名  add file(
name = mydb,    -- 名字 -- 自定义最少有名字
filename ='D:\data\mydb.mdf',  -- 位置 文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =10% -- 增长方式 百分比 绝对值
)
```

- 增加日志文件 语法

```
alter database 数据库名  add log file(
name = mydb,    -- 名字 -- 自定义最少有名字
filename ='D:\data\mydb.mdf',  -- 位置 文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =10% -- 增长方式 百分比 绝对值
)
```

- 例子

```
alter database mydb add file(add log file)( -- 次数据日志
name = mydb,    -- 名字 -- 自定义最少有名字
filename ='D:\data\\',  -- 位置 文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =10% -- 增长方式 百分比 绝对值
)
```

- 修改文件（数据或日志）语法

```
alter database 数据库名 modify file(
name = mydb,    -- 名字 -- 自定义最少有名字
filename ='D:\data\mydb.mdf',  -- 位置 文件名
size=5,  -- 初始大小
maxsize= 20,  -- 最大大小
filegrowth =10% -- 增长方式 百分比 绝对值
)
```

- 例子

```
alter database mydb modify file(
name=mydb_data,
size=5MB
)
```

- 删除(移除)文件(数据或日志) 语法

```
alter database 数据库名 remove file 数据文件逻辑名-- 只能逻辑名
```

- 例子

```
alter database mydb remove file mydb_data
```

- 增加文件组

```
alter database 数据库名 add filegroup 文件组名
```

- 例子

```
alter database mydb add filegroup Fgroup
alter database test3
add file
```

```
(
name=test3_data2,
filename='d:\SQL\test3 data2.ndf'
size=10
),
(
name=test3_data3,
filename='d:\SQL\test3 data3.ndf',
size=5
)
to filegroup FGroup -- to filegroup 把新添加的次要数据文件放到FGroup组
```

- 删除数据库 语法

```
drop database 数据库名
```

- 例子

```
drop database mydb
```

# 管理维护数据库

## 分离附加数据库

- 分离数据库 语法 （系统数据库master、empdb、moel不可分离）

```
execute sp_detach_db 数据库名
```

例子

```
execute sp_detach_db mydb
```

- 附加数据库 语法

```
execute sp_attach_db 数据库名 '文件位置(需要加文件名)'
```

例子

```
execute sp_attach_db mydb ,'D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb_log.bak'
```

- 注：通过分离和附加数据库可以实现SQL Server数据库**文件存储位置的改变**（移植数据库）
- 备份 语法

```
backup database 数据库名 to disk = '文件位置(需要加文件名)'
```

例子

```
backup database mydb to disk = 'D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb_log.bak'
```

- 还原 语法

```
restore  database 数据库名 from disk = '文件位置(需要加文件名)'
```

例子

```
restore  database 数据库名 from disk = 'D:\EXE\BC\sql server\sql server
sl\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyDb_log.bak'
```

# 数据表操作

- 表和表结构：每个数据库包含了若干个表。表是SQL Server中最主要的数据库对象，它是用来**存储数据**的一种**逻辑结构。表由行和列组成**，因此也称为**二维表。**表是在日常工作和生活中经常使用的一种**表示数据及其关系**的形式
- 下面简单介绍与表有关的几个概念：
  1. 表结构。组成表的各列的名称及数据类型，统称为表结构。
  2. 记录。每个表包含了若干行数据，它们是表的"值"，表中的一行称为一个记录。
  3. 字段。表中的一列称为字段。例如，表3.1中表结构为（学号，姓名，性别，出生时间，专业，总学分，备注），包含7个字段，由5个记录组成。
  4. 空值。空值(NULL)通常表示未知、不可用或将在以后添加的数据。若一个列允许为空值，则向表中输入记录值时可不为该列给出具体值；而一个列若不允许为空值，则在输入时必须给出具体值。
  5. 关键字。若表中记录的某一字段或字段组合能唯一标识记录，则称该字段或字段组合为候选关键字(Candidate key)。若一个表有多个候选关键字，则选定其中一个为主关键字（Primary key),也称为主键。
  6. **外码(不能是当前表的主码)一定会参照另一个表的主码**
- 创建表 语法

```
create table 表名(
字段1 字段1类型 列级完整性约束条件,
字段2 字段2类型 列级完整性约束条件,
字段3 字段3类型 列级完整性约束条件,
...
字段n 字段n类型 列级完整性约束条件
)
```

- 注：列级完整性约束条件如下：
  1. primary key:指定该字段为主键（不为空且唯一）
  2. null / not null:指定的字段允许/不允许为空，如果没有约束条件，则默认为NULL(列级约束)
  3. unique :指定字段取值唯一，即每条记录的指定字段值不能重复 (允许列中有一个空值)
  4. default<默认值>：指定设置字段的默认值。（列级约束）
  5. check<条件表达式>：对输入值检验，拒绝接受不满足条件的值。只看非空约束 （不是null的）
  6. identity(1,1)自增长（第一个值,步长）不能直接添加数据自动增长

     如果想强制执行 set identity_insert 表名 on
  7. foreign key 外键 注意字段类型必须一样

     例子

```
foreign key reference 表名(字段)
```

8. 第二条和第四条是列级约束其他的是列级约束和表级约束都可以　　列级约束是 只能针对当列约束

   表级约束可以多列

- 例子

```
create table student(
 sno char(8) not null primary key,
 sanme char(10) not null,
 grender char(2) null default '男' check(grender='女' or grender='男'),
 sbirth date,
 email char(30) unique, -- 唯一约束
 major char(20),
 chedit int check(chedit>=0 and chedit<120),
 remark varchar(500)
)
```

- 注意：一个主键可以由多个字段构成

```
create table student(
 sno char(8) ,
 sanme char(10),
 grender char(2) null default '男' check(grender='女' or grender='男'),
 sbirth date,
 email char(30) unique, -- 唯一约束
 major char(20),
 chedit int check(chedit>=0 and chedit<120),
 remark varchar(500),
  primary key(sno,sanme),  --联合主键
  foreign key(cno) references sc(sno)
)
```

- 修改数据表 语法

```
-- 增加字段
alter table 表名 add 新字段名称 数据类型 列级完整性约束条件
-- 修改字段的数据类型
alter table 表名 alter column 字段名称 新数据类型
-- 删除字段
alter table 表名 drop column 字段名称
-- 添加约束
alter table 表名 add constraint 约束名 约束条件 (字段名称)
-- 修改约束
alter table 表名 alter constraint 约束名 约束条件 (字段名称)
-- 删除约束
alter table 表名 drop constraint 约束名
```

- 例子

```
-- 增加字段
alter table student add sql char(30) null
-- 修改字段的数据类型
alter table student alter column sql char(20)
-- 删除字段
alter table student drop column sql
-- 添加约束
alter table student add constraint uq_stu_sq unique (sql)
-- 修改约束
alter table student alter constraint uq_stu_sq unique (sql)
-- 删除约束
alter table student drop constraint uq_stu_sq
```

- 删除表 语法

```
drop table 表名
```

- 例子

```
drop table student
```

- 系统数据类型
  - 整型

| 类型 | 字节数 | 范围 |
| --- | --- | --- |
| bigint(长整型) | 8个字节 | 其取值范围-$2^{63}$~$2^{63}$-1 |
| int（基本整型) | 4个字节 | 其取值范围-$2^{31}$~$2^{31}$-1 |
| smallint（短整型) | 2个字节 | 其取值范围-$2^{15}$~$2^{15}$-1 |
| tinyint(微整型) | 1个字节 | 表示无符号整数，其取值范围0~255 |

  - 精确数值型
    - decimal和numeric两种类型(等价)
    - 格式：decimal(p,s)或者numeric(p,s)
      - p(有效位数，小数点**左右两侧位数之和**)
      - s(小数位数)，s默认值为0，0<=s<=p
    - **decimal(10,6)表示数中共有10位数，其中数部分占4位，小数部分占6。**
  - 近似数值型
    - 包括real和floati两种类型。
    - real:精确到7位小数，数据范围：-3.40E$^{+38}$~3.40E$^{+38}$
    - float:可以精确到第15位小数，数据范围：-1.79E$^{+308}$~1.79E$^{+308}$。
  - 字符数据类型（表示8000字符以内）
    - 字符型包括char、varchar、nchar 和 nvarcharl四种类型。
    - char(n)
      - 存放**固定长度**的个字符数据。若输入字符长度不足n时，则用**空格**补足。1≤n≤8000。

        - char(10),那么不论存储的数据是否达到了10个字节，都要占去10个字节的空间，不足的自动用空格填充。
    - varchar(n)
        - 存放**可变长度**的n个字符数据。若输入字符长度不足n时，则按实际输入长度存储。1≤n≤8000
    - nchar[(n)]
        - 可存储1~4000个定长Unicode字符串，字符串长度在创建时指定；如未指定，默认为nchar(1)。**每个字符占用2bytes存储空间。**
    - nvarchar[(n)]
        - 可存储最大值为4000个字符**可变长**Unicode字符串。可变长Unicode字符串的最大长度在创建时指定，如nvarchar(50),**每个字符占用2 bytes存储空间。**
- 文本数据类型（表示8000字符以上）
    - 当存储的字符数目大于8000时使用文本型，文本型包括text和ntext两种。
    - text
        - 用来存储ASCII编码字符数据，最多可以存储$2^{31}$-1（约20亿）个字符。在定义Text数据类型时，不需要指定数据长度.
    - ntext:
        - 用来存储Unicode编码字符型数据，最多可能存储$2^{30}$-1（约10亿)个字符 其存储长度为实际字符个数的**两倍**，因为Unicode字符用双字节表示。
- 日期数据类型
    - date 日期格式 yyyy-MM-dd(年月日)
    - datetime 日期格式 yyyy-MM-dd HH:mm:ss.fff (年月日时分秒 精确到3.33毫秒)
    - 日期型属性的具体数值需要**加单引号**
- **写T-SOL语句时，日期型属性、char、varchar、nchar、nvarchar、text、ntext类型的属性的具体数值需要加单引号。**

**关系的完整性**

- 关系的完整性（最大限度地保证数据的**正确性**）
- 关系模型的完整性规则是对关系的某种约束条件。
- 关系模型中允许定义3类完整性约束：

1. √实体完整性

   实体完整性规则若属性A是基本关系R的主属性，则属性A不能取空值。
   例如：学生关系“学生学号，姓名，性别，专业号，年龄”中，“学号”
   为主码，则“学号”不能取空值。

2. √参照完整性

   学生、课程、学生与课程之间的多对多联系选修可以用如下3个关系表示。
   √学生（**学号**，姓名，性别，专业号，年龄)
   √课程（**课程号**，课程名，学分）
   √选修（**学号**，**课程号**，成绩）

3. √用户自定义的完整性

   用户自定义的完整性就是针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必须满足语义要求。例如某个属性必须取唯一值、属性值之间应满足一定的函数关系、某属性的取值范围在0~100之间等。
   √例如，性别只能取“男”或“女”；学生的成绩必须在0~100之间。

# 操作数据表中的数据

# 插入

1. 插入单条记录
   - 语法

     ```
     insert into 表名 [列名清单] values 常量清单
     ```

     如果省略<**列名清单**>，则按<常量清单>顺序为**每个属性**列赋值，即每个属性列上都应该有值
   - 例子

     ```
     insert into student values ('20200101','张三','男','2001-06-
     09','666@QQ.COM','计算机系',100,'备注1')
     ```

   - 注：
     1. 表中不允许为空值的项必须输入
     2. 省略列名清单，则常量清单应于表中的列名顺序一致
2. 插入多条记录
   - 语法

     ```
     insert into 表名 [列名清单] values (常量清单1),(常量清单2),...
     ```

   - 例子

     ```
     insert into student values ('20200102','张三','男','2001-06-
     09','999@QQ.COM','计算机系',100,'备注1'),('20200103','萧炎','男','2001-06-
     09','66666@QQ.COM','计算机系',100,'备注1'),('20200104','刘6','男','2001-06-
     09','666999@QQ.COM','计算机系',100,'备注1'),
     ('20200105','张齐','男','2001-06-09','666999666@QQ.COM','计算机系',100,'备
     注1'),('20200106','钱多多','男','2001-06-09','9996669996666@QQ.COM','计算机
     系',100,'备注1')
     ```

   - 小结
     1. INSERT语句中的INTO可以省略
     2. 如果某些属性列在表名后的列名表没有出现，则新记录在这些列上取空值
     3. 如果没有指明任何列名，则新插入的记录必须在每个属性列上均有值
     4. 字符型或日期型数据必须使用''（单引号）将其括起来
     5. 常量的顺序必须和指定的列名顺序保持一致

# 修改

- 语法

  ```
  update 表名 set 列名1=表达式1 [,列名2=表达式2] [where 条件表达式]
  ```

  说明：
  1. 如果不指定条件，则会修改表中所有记录
  2. 如果要修改多列，则在SET语句后用”，”分割各修改子句
- 例

```
update student set sanme ='刘德华' where sanme='张齐'
update student set chedit=60 where chedit<60
```

## 删除

- 语法

```
delete from 表名  [where 条件表达式]
```

说明:
当无WHERE<条件表达式>时将删除<表>中所有记录，但是，该表结构还在，只是变为了空表

- 例

```
delete from student where sanme='刘德华'
```

## 单表查询

- 语法

```
SELECT all[DISTINCT][TOP N[PERCENT]],字段1,字段2,字段3..[AS 别名] FROM 表名;
```

- 说明
  1. ALL:表示输出所有记录，包括重复记录。默认值为ALL。
  2. DISTINCT:表示在查询结果中去掉重复值。
  3. TOP N:返回查询结果集中的前N行。

     加[PERCENT]返回查询结果集中的前N%行。N的取值范围是0~100。
- 例

```
select * from student
select distinct *  from student
select top 6 *  from student
-- top % 计算结果向上取整数
select top 80 percent  * from student
select sno as '学号'  from student
-- 计算年龄现在减出生日期
select sanme ,year(getdate())-year(sbirth) from student
-- 分数提20%
select sno,chedit*1.2 AS 成绩 from student
```

- 单表有条件查询
- 语法

```
SELECT all[DISTINCT][TOP N[PERCENT]],字段1,字段2,字段3..[AS 别名] FROM 表名
where 条件;
```

- WHERE条件中的运算符

| 查询条件 | 运算符 |
| --- | --- |
| 比较运算符 | =, <, >, <=, >=, != |
| 逻辑运算符 | AND,OR,NOT |
| 范围运算符 | BETWEEN AND,NOT BETWEEN AND |
| 列表运算符 | IN,NOT IN |
| 字符匹配符 | LIKE,NOT LIKE |
| 空值 | IS NULL,IS NOT NULL |

1. WHERE子句中可以使用逻辑运算符AND、OR和NOT,这3个逻辑运算符可以混合使用。
2. 在WHERE子句中使用BETWEEN关键字查找在某一范围内的数据，也可以使用NOT BETWEEN关键字查找不在某一范围内的数据。

3. 在WHERE子句中使用字符匹配符LIKE或NOT LIKE可以把表达式与字符串进行比较，从而实现对字符串的模糊查询。

   通配符% 表示0或者多个字符

   通配符_ 表示任意一个字符

4. 在WHERE子句中，如果需要确定表达式的取值是否属于某一列表值之一时，就可以使用关键字IN或NOT IN来限定查询条件。
5. 当数据表中的值为NULL时，可以使用IS NULL关键字的WHERE子句进行查询，反之要查询数据表的值不为NULL时，可以使用IS NOT NULL关键字。**注意无法用等于号（=）判断空**

- 例子

```
select * from student WHERE grender='男'
select * from student WHERE chedit>60
select * from student WHERE major='计算机系'
-- 查询计算机系女生的信息。
select * from student where major='计算机系' and grender='女'
-- 查询成绩在90分以上或不及格的学生学号和课号信息。
select * from student where chedit>90 or chedit<60
-- 查询非计算机
select * from student where not major='计算机系'
-- 查询成绩在60~70分之间含(60,70)的学生学号及成绩。
select sno,chedit from student where chedit between 60 and 70
-- 查询所有姓张的学生的个人信息。
select * from student where sanme like '张%'
-- 查询所有名字第2个字带三的学生的个人信息。
select * from student where sanme LIKE '_三%'
-- 查询所有姓张姓唐的学生的个人信息。
select * from student where sanme like'[张唐]%'
-- 查询软件和计算机的学生
select * from student where major in('软件','计算机系')
--查询缺少成绩的学生
select * from student where chedit is  null
-- 不为空
select * from student where chedit is not null
```

# 聚合函数

- SQL Server的聚集函数是综合信息的统计函数，也称为聚合函数或集函数，
    1. 包括计数、求最大值、求最小值、求平均值和求和等。
    2. 聚集函数可作为列标识符出现在SELECT子句的目标列或HAVING子句的条件中。
    3. 在SQL查询语句中，如果有GROUP BY子句，则语句中的函数为分组统计函数；否则，语句中的函数为全部结果集的统计函数。SQL提供的聚集
- 

| 函数 | 说明 |
| --- | --- |
| COUNT(*) | 统计行的个数 |
| COUNT(<列名>) | 统计一列中值的个数 |
| MAX(<列名>) | 计算一列中值的最大值 |
| MIN(<列名>) | 计算一列中的最小值 |
| SUM(<列名>) | 计算一列中值的总和 |
| AVG(<列名>) | 计算一列中值的平均值 |

  注意：**聚集函数忽略空值** 所以用非空项查询， **除了count其他都不能用*号**

- 例子

```
-- 查询学生总数。
select count(*) from student
--查询选修了课程的学生人数。
select count(distinct sno) from sc
--计算男学生平均成绩。
select avg(chedit) from student where grender='男'
--查询学生最高分和最低分。
select max(chedit),min(chedit) from student
```

# 分组与排序

1. 对查询结果集进行分组
    - 使用GROUP BY子句可以将查询结果按照**某一列或多列数据**值进行**分类**，换句话说，就是对查询结果的信息进行**归纳**，以**汇总**相关数据。
      GROUP BY子句把查询结果集中的各行按列名清单进行分组，在这些列上，对应值都相同的记录分在同一组。若无HAVING子句，则各组分别输出；若有HAVING子句，**只有符合HAVING条件**的组才输出。
    - 语法

      ```
      [GROUP BY列名清单][HAVING条件表达式]
      ```

    - 例子：

```
-- 统计各系人数
select sdept, count(*) from student group by sdept
-- 统计表中男女学生人数
select gender, count(*) from student group by gender
-- 统计各系男女学生人数
select sdept, gender, count(*) from student group by gender,sdept
-- 统计各系女学生人数
select sdept, gender, count(*) from student where gender='女' group by
gender,sdept
select sdept, gender, count(*) from student   group by gender,sdept
having gender='女'
-- 查询选修了2门以上课程的学生学号 每个学生的选课门数
select sno, count(*) from sc group by sno having count(*)>2
-- 变式处理：查询被2个以上同学选过的课程号
select cno ,count(*) from sc group by cno having count(*)>2
```

注意：where对基本表table的行为记录做筛选 having针对分组统计的结果进行筛选

聚合函数可以出现在 having order后面

注意分组和聚合函数里面没有的字段不能直接查询因为没进行过处理无法查询会报错

2. 对查询结果集进行排序
   - 用户可以利用ORDER BY子句对查询结果按照一个或多个字段
     进行**升序(ASC)或降序(DESC)排序**，默认值为升序。
   - 语法

```
[ORDER BY<列名1>[ASC|DESC[,<列名2> [ASC|DESC],…
```

   - 例子

```
-- 查询选修了C02号课程的学生的学号、其成绩，查询结果按分数(sc)的降序排列。
select sno,cno,degree from sc where cno='C02' order by degree desc
-- 查询全体学生情况，查询结果按所在系升序排列，同一系中的学生按出生口期除序排列。
select * from student order by sdept asc, sbirth desc
```

# 多表连接查询

1. 内连接
   - 语法

```
SELECT [ALL | DISTINCT][别名]<选项1>[AS<显示列名>][，[别名.<选项2>[AS<显示列名
>]，...]FROM<表名1>[别名1]，<表名2>[别名2[，…]WHERE<连接条件表达式>[AND<条件表
达式>]
```

   注意：若在输出列或条件表达式中出现两个表的公共字段，则在公共、字段名前必须加别名

   - 例子

```
-- 输出所有女学生的学号、姓名、课号及成绩。
select s.sno,s.sname,c.cno, c.degree from student s, sc c where
s.sno=c.sno and s.gender='女'
select  s.sno,s.sname,c.cno, c.degree  from student s inner join sc c on
s.sno=c.sno where s.gender='女'
```

2. 外连接

- 在自然连接中，**只有**在两个表中**匹配**的行才能在结果集中出现。而在外连接中可以**只限制**一个表（**从表**），而对另外一个表**不加限制**（**主表**）。外连接分为**左外连接、右外连接和全外连接。**
- 语法

```
SELECT [ALL | DISTINCT][别名]<选项1>[AS<显示列名>][, [别名.<选项2>[AS<显示列名>], ...]
FROM<表名1>LEFT|RIGHT|FULL[OUTER]JOIN<表名2>
ON<表名1.列1>=<表名2.列2>
```

注意：对于左连接对于主表（left前面是主表）有限制对于从表要进行on的连接右连接同理

- 例子

```
-- 利用左外连接查询每个学生及其选修课的情况。
select s.*,c.cno ,c.degree from student s left join sc c on s.sno= c.sno
-- 右外连
select student.*,cno from sc  right join student  on student.sno= sc.sno
```

3. 嵌套查询-非相关子查询

- 在SQL语言中，一个**SELECT—FROM—WHERE语句称为一个查询块**。将一个查询块嵌套在另一个查询块的WHERE子句或HAVING子句的条件中称为嵌套查询或子查询。
- 非相关嵌套子查询的执行过程为：首先执行子查询，子查询得到的结果集不被显示出来，而是传给外部查询，作为外部查询的条件使用，然后执行外部查询，并显示查询结果。子查询可以多层嵌套。
- 例子

```
-- 查询选修了课程号C02的学生信息
select * from student where sno in(select sno from sc where cno='C02')
```

- 嵌套子查询一般也分为两种：子查询返回单个值和子查询返回一个值列表。

```
-- 查询所有年龄大于平均年龄的学生信息
select * from student where sage > (select avg(sage) from student)
-- 查询与刘晨在同一个系学习的学生信息。
select * from student where sdept in(select sdept from student where
sname='刘晨')
-- 统计选了“数据库”课程的学生的学号、选课门数和平均成绩
select sno,count(*),avg(degree) from sc where sno in(select sno from sc
where cno in(select cno from course where canme='数据库')) group by sno
```

- 带有ANY或ALL操作符的子查询。ANY和ALL操作符在使用时必须和比较运算符一起使用，其格式如下。

```
<字段><比较符>[ANY|ALL<子查询>
```

ALL代表所有 ANY代表其中一个

- 例子

```
-- 查询其他系中比计算机系学生年龄都小的学生。
select * from student where sage<all(select sage
from student where sdept='信息系' ) and sdept <> '信息系'
-- 查询其他系中比计算机系任一个学生年龄小的学生信息。
select * from student where sage <ANY(select sage
from student where sdept='信息系' ) and sdept <> '信息系'
```

- 使用存在量词 EXISTS(相关子查询)

  执行过程：对外层查询student表的每一条记录，检查内层查询是否为空，若不为空，则输出该记录（先执行外层查询，后执行内层查询）

- 例子

```
-- 查询选修了课程号C02的学生信息
```

# 数据库高级操作

## 视图

- 视图是从一个或者几个**基本表或者视图**中导出的虚拟表，
- 必须使用SQL中的**SELECT**语句来实现。
- 在定义一个视图时，只是把其**定义**存放在数据库中，并**不直接存储视图**对应的**数据**，直到用户使用视图时才去查找对应的数据。
- 主要作用：提供**用户视角**的数据
- 语法

```
CREATE VIEW 视图名字 [(Column [,...n])
[WITH ENCRYPTION]  AS 查询语句
```

select_statement:选择哪些列进入视图的SELECT语句。
WITH ENCRYPTION:对视图的定义进行加密。

注意：视图中的SELECT命令不能包括INTO、ORDER BY等子句。

- 例子

```
-- 创建视图
-- 有条件的视图定义。定义视图v student,查询所有选修C01号
-- 课程的学生的学号(sno)、姓名(sname)、课程名称(cname)和成绩(degree)
create view v_stdent with encryption as select
student.sno,sname,sc.cno,degree from student ,sc,course where
student.sno=sc.sno and sc.cno = course.cno and sc. cno='C01'
-- 使用视图
select * from v_stdent -- 用户视角：提高安全性
-- 定义视图vstudent_count,查询不同性别的学生人数。
create view v_student_count(gender,学生人数) as
select gender,count(*) from student group by gender
-- 使用视图
select * from v_student_count
-- 删除视图
drop view 视图名称
-- 通过视图查询性别人数超过1人
select * from v_student_count where 学生人数>1
```

注意：创建视图中如果有原表中没有的数据需要起别名 通过视图还可以进行筛选

# 索引

1. 索引概念
   - 索引：是SQL Server编排数据的内部方法。它为SQL Server提供一种方法来编排查询数据
   - 索引页：数据库中存储索引的数据页；索引页类似于汉语字(词)典中按拼音或笔画排序的目录页。
   - 索引的作用：通过使用索引，可以大大提高数据库的检索(查询)速度，改善数据库性能。
2. 索引类型
   - 聚集索引(Clustered):表中各行的物理顺序与键值的逻辑（索引）顺序相同，每个表只能有一个；聚集索引适用于范围查询。
   - 非聚集索引(Nonclustered):表中各行的物理顺序与键值的逻辑（索引)顺序不相同索引中包含指向数据存储位置的指针；非聚集索引适合直接匹配单个条件的查询；每个表可以有1~249个。
   - 唯一索引：唯一索引不允许两行具有相同的索引值
   - 注意：聚集索引并不一定是唯一索引，由于SQL SERVER将主键默认定义为聚集索引，事实上，索引是**否唯一与是否聚集是不相关**的，聚集索引可以是唯一索引，也可以是非唯一索引；
3. 语法
   - 创建索引

     ```
     CREATE [UNIQUE][CLUSTERED | NONCLUSTERED]
     INDEX 索引名字
     ON 表名 (column_name...)
     [WITH FILLFACTOR=X]
     ```

     - UNIQUE表示唯一索引，可选
     - CLUSTERED、NONCLUSTERED表示聚集索引还是非聚集索引，可选
     - FILLFACTOR表示填充因子，指定一个0到100之间的值，该值指示索引页填满的空间所占的百分比
   - 删除索引

     ```
     drop index 表名 索引名
     drop index 索引名 on 表名
     ```

   - 例子

     ```
     --为Student表的sno列创建一个唯一性的聚集索引：
     create unique clustered index idx_student_sno on student(sno)
     ```

     - 注意：在执行此命令前先删除原来该表的主关键字属性
     - 提示：主关键字约束默认相当于聚集索引和唯一索引的结合。

# 存储过程

- 创建

  ```
  create procedure 存储过程名字 as 查询语句
  ```

- 使用存储过程

```
execute 存储过程名字
```

- 修改

```
alter procedure 存储过程名字 as 查询语句
```

- 删除

```
drop procedure 存储过程名字
```

注意 procedure可以缩写称 proc

- 例子

```
-- 创建存储过程
-- 例：查询选修了课程号CO2(sc)的学生信息(sno,sname,gender,)student
-- 连接查询
create procedure proc_t1  as select student.*,cno from student,sc where
student.sno=sc.sno and cno='C02'
-- 执行存储过程
execute proc_t1
-- 修改存储过程
alter procedure proc_t1 as select * from student
-- 删除存储过程
drop procedure proc_t1
```

- 存储功能可以带参数 用@加参数名称

  例子

```
-- 例：查询选修了某课程号的学生信息(sno,sname,gender,)student
create procedure proc_t2 @cno char(4) as select student.*,cno from
student,sc where student.sno=sc.sno and cno=@cno
-- 使用
execute proc_t2 'C01'
execute proc_t2  @cno='C01'
-- 修改存储过程
alter proc proc_t1  as select student.*,cno from student,sc where
student.sno=sc.sno and cno='C02'
```