SQL：是用于访问和处理数据库的标准的计算机语言。SQL是种结构化的查询语言，他能够让我们使用数据库。

1. SQL面向数据库执行查询
2. SQL可以从数据库中取回数据（select）
3. SQL可在数据库中插入新的记录（insert into）
4. SQL可更新数据库中的数据（update）
5. SQL可从数据库中删除记录（delete）
6. SQL可创建数据库 （create）
7. SQL可在数据库中创建新表
8. SQL可在数据库中创建存储过程
9. SQL可在数据库中创建视图
10. SQL可以设置表、存储过程和视图的权限

SQL主要分为两个部分：数据操作语言（DML）和数据定义语言（DDL）。

DML：查询和更新指令

DDL：创建和删除等指令

关系型数据库：

一个数据库由多张表构成的，那么每张表有不重复烦人名称（表名），而且每个表是有很多列组成的。

属性：一张关系表中的一个列就是一个属性，一个列又有他的名称和类型。

元组：一张关系表中的一行数据就是元组，也称为记录。

键：键是为在元组中快速查找设置的。键有主键和候选键之分。

第一范式：指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，既实体中的某个属性不能有多个值或者不能有重复的属性。

1. 元组中的每个属性只能包含一个值。【每个单元格必须只能有一个值】
2. 关系中的每个元组必须包含相同的数量【每行都有相同数量的列】
3. 关系中的每个元组一定不能相同【一个表中不能有相同的行】

第二范式：

1. 满足第二范式必须先满足第一范式
2. 要求数据表中的每一行记录必须可以被唯一的区分（通常需要为表加添加主键）

第三范式：

1. 必须先满足第二范式
2. 要求一个数据库表中不包含已在其他表中已包含的非主关键字关键字信息（也就是外键）

外键：主要是用来约束用于预防破坏表之间连接的动作

索引：主要是在不读取全表的时候加速查询。分为聚集索引和非聚集索引。

聚集索引只要的特点就是调整索引表顺序的同时也会调整原表的顺序。当创建主键时自动创建聚集索引。一个表只能有一个主键，同样一个表也只能有一个聚集索引。一个聚集索引最多有16列。

非聚集索引：就是调整索引表的同时不会改变原表的顺序。当创建唯一约束时自动创建非聚集索引。一个非聚集索引表最多有249列。

约束：约束分为行约束、列约束和引用约束。行约束和列约束是针对单表的，引用约束是针对两个表的。

行约束：索引、唯一约束、主键（不为空，记录不能重复）、自动增长（针对int类型）

列约束：check约束、not null约束、default约束

引用约束：外键约束

SQL Server数据类型：

|  |  |  |
| --- | --- | --- |
| Character字符串 | | |
| 数据类型 | 描述 | 存储 |
| char(n) | 固定长度的字符串。最多8000个字符 | n |
| varchar(n) | 可变长度的字符串。最多8000个字符 | n |
| varchar(max) | 可变长度的字符串，最多1073741824个字符 | n |
| text | 可变长度的字符串。最多2GB个字符数据 | n |

|  |  |
| --- | --- |
| Unicode字符串 | |
| 数据类型 | 描述 |
| nchar(n) | 固定长度的Unicode数据。最多4000个字符 |
| nvarchar(n) | 可变长度的Unicode数据。最多4000个字符 |
| nvarchar(max) | 可变长度的Unicode数据。最多534870912个字符 |
| ntext | 可变长度的Unicode数据。最多2GB个字符数据 |

|  |  |
| --- | --- |
| Binary类型 | |
| 数据类型 | 描述 |
| bit | 运行0、1或null |
| binary(n) | 固定长度的二进制数据。最多8000字节 |
| varbinary(n) | 可变长度的二进制数据。最多8000个字节 |
| varbinary(max) | 可变长度的二进制数据。最多2GB字节 |
| image | 可变长度的二进制数据。最多2GB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number类型 | | | | |
| 数据类型 | 描述 | | | 存储 |
| tinyint | 允许从0到255的所有数字 | | | 1字节 |
| smallint | 允许从-32768到32747的所有数字 | | | 2字节 |
| int | 允许从-2147483648到2147483647的所有数字 | | | 4字节 |
| bigint | 允许从 -10^38 +1 到 10^38 -1 之间的数字。 | | | 8字节 |
| decimal(p,s) | p参数指示可以存储的最大位数（小数点左侧和右侧）。p必须是1到38之间的值。默认是18。s参数指示小数点右侧存储的最大位数。S必须是0到p之间的值。默认是0.允许从 -10^38 +1 到 10^38 -1 之间的数字。 | | | 5-17个字节 |
| numeric(p,s) | p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | | | 5-17个字节 |
| smallmoney | 介于-214,748.3648 和214,748.3647 之间的货币数据 | | | 4字节 |
| money | 介于 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之间的货币数据 | | | 8字节 |
| float(n) | 从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据。参数 n 指示该字段保存 4 字节还是 8 字节。float(24) 保存 4 字节，而 float(53) 保存 8 字节。n 的默认值是 53。 | | | 4或8个字节 |
| real | 从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据。 | | | 4字节 |
| Date类型 | | | | |
| 数据类型 | | 描述 | 存储 | |
| datetime | | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 3.33 毫秒。 | 8 bytes | |
| datetime2 | | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | 6-8 bytes | |
| smalldatetime | | 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | 4bytes | |
| date | | 仅存储日期。从 0001 年 1 月 1 日 到 9999 年 12 月 31 日。 | 3bytes | |
| time | | 仅存储时间。精度为 100 纳秒 | 3-5bytes | |
| datetimeoffset | | 与datetime2 相同，外加时区偏移 | 8-10bytes | |
| timestamp | | 存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量 |  | |

--创建数据库 主语句是CREATE DATABASE

--指令是CREATE DATABASE databasename;

CREATE DATABASE testdb01;

--在SQL Server中一个数据库中包含有三种文件：一种是.mdf，一种是.ndf还有.ldf。

--mdf是主数据文件，ndf是次数据文件，ldf是日志文件

--创建一个指定数据文件的数据库

CREATE DATABASE testdb02

ON(

    NAME=学生,

--数据文件名

    FILENAME="C:\data\学生.mdf",

    --数据文件的物理位置

   SIZE=10MB,

    --初始大小，默认是MB

    --MAXSIZE=20MB,

    --最大大小

    MAXSIZE=UNLIMITED,

    --最大大小没有限制

    FILEGROWTH=1MB

    --文件自增长大小

);

--上面create database是主语句，on是子语句

CREATE DATABASE testdb03

ON(

    NAME=Course,

    FILENAME='C:\data\course.mdf',

   SIZE=10MB,

   MAXSIZE=20MB,

    FILEGROWTH=10%

   --以百分之十增长

),

(

    NAME=Teach,

    FILENAME='C:\data\teach.ndf',

    --通过文件的扩展名指定次数据文件

    SIZE=10MB,

    MAXSIZE=20MB,

    FILEGROWTH=10%

)

LOG ON(

    NAME=DB\_LOG,

    FILENAME='C:\data\DB\_LOG.LDF',

    SIZE=10MB,

    MAXSIZE=20MB,

    FILEGROWTH=10%

);

--删除数据库

--DROP DATABASE database\_name

DROP DATABASE testdb01;

--修改数据库

--主语句是alter database databasename

--给testdb02增加一个数据文件

ALTER DATABASE testdb02

ADD FILE(

    NAME=教师,

    FILENAME='C:\data\教师.ndf',

    SIZE=10MB,

    MAXSIZE=20MB,

    FILEGROWTH=10%

);

--修改刚添加的文件教师.ndf

ALTER DATABASE testdb02

MODIFY FILE(

    NAME=教师,

    --这个必须是存在的文件名

    FILENAME='C:\data\教师.ndf',

    SIZE=13MB,

    --更改后的大小必须必之前的大

    MAXSIZE=10MB,

    --这个也是大小只能比之前的大

    FILEGROWTH=3MB

);

--删除刚刚增加的文件

ALTER DATABASE testdb02

REMOVE FILE 教师;

--修改数据库名

ALTER DATABASE testdb02

MODIFY NAME=demodb02;

--使用数据库，在数据库中建表这些操作之前要先使用数据库

--USE databasename;

USE testdb03;

CREATE DATABASE TestDB;

USE TestDB;

--建一个最简单的表

CREATE TABLE test01(

   ID int,

   Sname nvarchar(100),

    grade varchar(50)

)

--空约束，NOT NULL不可以为空

CREATE TABLE Student(

    ID int NOT NULL,

    Sname nvarchar(50),

    score real

)

-- 主键约束

--PRIMARY KEY 约束唯一标识数据库表中的每条记录。

--主键必须包含唯一的值。

--主键列不能包含 NULL 值。

--每个表应该都一个主键，并且每个表只能有一个主键。

CREATE TABLE Student01(

    ID int PRIMARY KEY,

    Sname nvarchar(50),

    score real

)

--唯一约束，一个表中只能有一个主键约束，但是可以有多个约束。

--Primary Key自动拥有唯一约束

CREATE TABLE Student02(

    ID int PRIMARY KEY NOT NULL,

    Sname nvarchar(50) UNIQUE,

    score real

)

--默认约束

CREATE TABLE Student03(

    ID int PRIMARY KEY NOT NULL,

    Sname nvarchar(50) UNIQUE,

    score real,

    address nvarchar(200) DEFAULT '钓鱼岛是中国的'

)

--自动增长约束

CREATE TABLE Student04(

    ID int PRIMARY KEY NOT NULL IDENTITY(1001,1),

    Sname nvarchar(50) UNIQUE,

    score real

)

USE TestDB;

--昨天我们讲的创建约束都是在定义列时直接建立的，今天我们在介绍一个指令CONSTRAINT

--用CONSTRAINT语句建立主键、唯一、默认、检查约束

CREATE TABLE Stu(

    Sno int,

    Sname varchar(50),

    Saddr varchar(100) CONSTRAINT DF\_Saddr DEFAULT '北京市',

    Sage int,

    --用constraint建立约束

    CONSTRAINT PK\_Sno PRIMARY KEY (Sno),

    CONSTRAINT UN\_Sname UNIQUE (Sname DESC),

    CONSTRAINT CK\_Sage CHECK (Sage>15)

);

--建立外键约束

CREATE TABLE Scourse(

    Cno int,

    Cname varchar(50),

    Tno int,

    Sno int FOREIGN KEY REFERENCES Stu(Sno)

    --用constraint建立外键约束

    --CONSTRAINT FK\_Scoures\_Stu FOREIGN KEY(Sno)

        --REFERENCES Stu(Sno)

    --这里要注意Stu表中的Sno列一定要是主键哦^\_^

    --还有主键列和外键列的这个列类型要一致哦

);

--再建一个表，下面的例子用

CREATE TABLE Teacher(

    Tno int,

    Tname varchar(50),

    Tage int,

    CONSTRAINT PK\_Teacher PRIMARY KEY(Tno)

);

--修改表，前面修改数据库我们用的是ALTER DATABASE，我们这里也是用ALTER

--增加列

ALTER TABLE Teacher

    ADD Taddr varchar(200);

--修改列

ALTER TABLE Teacher

    ALTER COLUMN Taddr varchar(100);

--删除列

ALTER TABLE Teacher

   DROP COLUMN Taddr;

--增加约束，和上面用constraint建立约束一样的

--这里就增加一个外键了

ALTER TABLE Scourse

    ADD CONSTRAINT FK\_Teacher FOREIGN KEY(Tno)

        REFERENCES Teacher(Tno);

--删除约束

ALTER TABLE Scourse

    DROP FK\_Teacher;

--搜狗输入法在SQL Server里好难用啊，总变成全角的

--这里再增加一个默认约束，注意一下是怎么指定列的?

ALTER TABLE Teacher

    ADD CONSTRAINT DF\_Tage DEFAULT 35 FOR Tage;

--删除表

DROP TABLE Scourse;

--在使用数据库之前，要先USE哦

USE TestDB;

--创建表，然后填入数据

CREATE TABLE Course(

   Cno char(4) PRIMARY KEY,

   Cname char(50),

    ClassHour char(4),

    Credit int

);

--插入数据使用INSERT INTO语句

INSERT INTO Course VALUES(1001,'软件测试基础',45,4);

--插入成功后，下面的消息框会提示“（1行受影响）”

--插入一条记录，只提供部分列的值

INSERT INTO Course(Cno,Cname) VALUES(1002,'数据库基础');

--大家看下下面这条指令可以正确执行吗？

INSERT INTO Course(Cname,Credit) VALUES('Java程序设计',4);

--答案肯定是不能了，为什么呢？

--因为Cno列有主键约束，所以不可以将NULL值插入Cno列

--我们在插入几条记录

INSERT INTO Course VALUES(1003,'Linux基础',40,4);

INSERT INTO Course(Cno,Cname) VALUES(1004,'自动化测试基础');

INSERT INTO Course VALUES(1005,'白盒测试基础',40,4);

INSERT INTO Course(Cno,Cname) VALUES(1006,'Python语言');

--下面来看一下怎么修改记录，修改记录就是更新表

--我们用update指令

UPDATE Course

    SET Cname='Loadrunner';

--执行这条语句，表中所有的Cname都会变成Loadrunner了

--那么可不可以使用这条语句更改Sno的值呢？

UPDATE Course

    SET Cno=1005;

--执行这条语句，会提示违反了主键约束，Cno不能有重复的

--修改满足条件的记录

UPDATE Course

    SET Cname='白盒测试技术' WHERE Cno=1005;

--Cno为1005的Cname列又变为白盒测试技术了

--WHERE子句在查询时还会经常使用的

--更改Cno>1003的Credit为Credit+1

UPDATE Course

    SET Credit=Credit+1 WHERE Cno>1004;

--删除记录

DELETE Course

    WHERE Cno=1005;

--删除全部记录

DELETE Course;

USE TestDB;

--先建个表

CREATE TABLE Book(

    bId int NOT NULL IDENTITY(0001,1),

    bName char(50),

    bPrice real,

    bPress char(100)

);

--建立索引

CREATE INDEX IN\_book ON Book(bId ASC);

--ASC是升序

--DESC是降序

--用两列建立索引

CREATE INDEX IN\_book1 ON Book(bId ASC,bPrice DESC);

--建立聚集索引

CREATE CLUSTERED INDEX IN\_book2 ON Book(bId DESC);

--建立非聚集索引

CREATE NONCLUSTERED INDEX IN\_book3

    ON Book(bId DESC);

--建立唯一索引

CREATE UNIQUE INDEX IN\_book4

    ON Book(bID ASC);

--大家看一下这个索引，能不能建立？

CREATE INDEX IN\_book ON Book(bId,bPrice DESC);

--失败了吧，因为我们前面建过一个名为IN\_book的索引了

--如果我们想要建一个索引，如果存在就删除它，建立新的，该怎么办呢？

CREATE INDEX IN\_book ON Book(bId,bPrice DESC)

    WITH DROP\_EXISTING;

--修改索引

--禁用IN\_book索引

ALTER INDEX IN\_book

    ON Book DISABLE;

--启用索引

ALTER INDEX IN\_book

    ON Book REBUILD;

--删除索引

DROP INDEX Book.IN\_book2;

USE TeachingDB;

--查询Student中的所有信息

SELECT \* FROM StudentInfo;

查询student中的学生的Sname和Saddress信息

SELECT Sname,Saddress FROM StudentInfo;

--查询student中的学生的Sname和Saddress信息，并设定别名

SELECT Sname AS '姓名',Saddress AS '地址' FROM StudentInfo;

SELECT Sname '姓名',Saddress '地址' FROM StudentInfo;

SELECT Sname 姓名,Saddress 地址 FROM StudentInfo;

SELECT Sname AS 姓名,Saddress AS 地址 FROM StudentInfo;

--这里的四种方法都行，但是建议用第一种方法

--查询student表中的前5行信息，用top

SELECT TOP 5 Sname,Sage FROM StudentInfo;

--查询student表中前10%的信息

SELECT TOP 10 PERCENT \* FROM StudentInfo;

--这个表中共有30条记录，所以这个结果显示前3条信息

--那么查询倒数5行信息呢？SQL中没有down之类的语句，大家思考一下这个问题

--下面用下面这个表看一下

SELECT \* FROM StudentCourse;

--这个表中的Sno有重复的项，共60条记录啊

--那么怎么去掉重复项呢？

SELECT DISTINCT Sno FROM StudentCourse;

--现在就剩30条了

--查询地址为四川绵阳的学生

SELECT \* FROM StudentInfo

    WHERE Saddress='四川绵阳';

--查询年龄大于19岁的学生信息

SELECT \* FROM StudentInfo

    WHERE Sage>19;

--查询年龄在18和20之间的

SELECT \* FROM StudentInfo

    WHERE Sage>18 AND Sage<20;

SELECT \* FROM StudentInfo

    WHERE Sage BETWEEN 18 AND 20;

--注意观察这两个区别？范围有区别的哦

--查询学生地址为四川自贡和四川成都的学生信息

SELECT \* FROM StudentInfo

    WHERE Saddress='四川自贡' OR Saddress='四川成都';

--查询学生地址不是自贡的学生信息

SELECT \* FROM StudentInfo

    WHERE NOT(Saddress='四川自贡');

SELECT \* FROM StudentInfo

    WHERE Saddress!='四川自贡';

SELECT \* FROM StudentInfo

    WHERE Saddress<>'四川自贡';

--查询Sno以02开头的学生信息

SELECT \* FROM StudentInfo

    WHERE Sno LIKE '02%';

--查询Sname中带有小字的学生信息

SELECT \* FROM StudentInfo

    WHERE Sname LIKE '%[小]%';

SELECT \* FROM RankPayment;

--查询平均payment

SELECT AVG(payment) FROM RankPayment;

select \* from courseinfo;

--统计所有的行数

SELECT COUNT(\*) FROM CourseInfo;

--查询coursehour最大的课程信息

SELECT MAX(CouresHour) FROM CourseInfo;

--查询coursehour最小的课程信息

SELECT min(CouresHour) FROM CourseInfo;

--查询coureshour大于平均课时的课程信息

SELECT \* FROM CourseInfo

    WHERE CouresHour>avg(coureshour);

USE commoditysell;

SELECT \* FROM EmporiumSell

--排序子句：Order By

--任务：查询ProductInfo表中所有产品的ProdName、UnitPrice，按UnitPrice降序排列

SELECT \* FROM ProductInfo

    ORDER BY UnitPrice DESC;

--查询最贵的2种电视机

SELECT TOP 2 \* FROM ProductInfo

    WHERE ProdName LIKE '%电视机%'

    ORDER BY UnitPrice DESC;

--用这个就可以解决我们昨天查询后5条的信息了吧。

--任务：查询SalerInfo中SalerName、SalerAge、SalerSex、Salary，按SaleAge升序、Salary降序

SELECT \* FROM SalerInfo

    ORDER BY SalerAge ASC,Salary DESC;

--分组子句：Group By

--任务：简单分组：查询按SalerAge分组的员工的平均Salary

--大家看一下这个会正确的执行吗？

SELECT \* FROM SalerInfo

    GROUP BY SalerAge;

SELECT Salary FROM SalerInfo

    GROUP BY SalerAge;

--上面这两条都出现了同样的问题，就是“因为该列没有包含在聚合函数或 GROUP BY 子句中”

--如何解决呢，就是我们下面的两条语句

SELECT avg(Salary) FROM SalerInfo

   GROUP BY SalerAge;

SELECT SalerAge,Salary FROM SalerInfo

    GROUP BY SalerAge,Salary;

--通过上述的例子要记住一点：

--使用group by子句，查询的列必须出现在group by子句中，后者使用聚合函数

--任务：查询按各年龄段的员工人数

SELECT SalerAge,count(SalerId) FROM SalerInfo

    GROUP BY SalerAge;

SELECT SalerAge,count(\*) FROM SalerInfo

    GROUP BY SalerAge

        HAVING avg(Salary)>1500;

上面是我们昨天剩的内容，下面开始今天的内容。

[sql] view plain copy

---------------------------------------------------------------------------------------------------------------------

--            多表查询

---------------------------------------------------------------------------------------------------------------------

USE commoditysell;

--交叉连接

SELECT \* FROM ProductInfo;--这个表共有9列

SELECT \* FROM EmporiumSell;--这个表共有51列

SELECT \* FROM ProductInfo CROSS JOIN EmporiumSell;

--这个结果是459列，就是9\*51了，所以交叉连接就是A表的一条记录对应B表的所有记录

--这里执行的是笛卡尔积

--我们使用where子句看一下

SELECT \* FROM ProductInfo CROSS JOIN EmporiumSell

    WHERE ProductInfo.ProdID=EmporiumSell.ProdID;

--这下就是51条了

--可以使用表的别名

SELECT \* FROM ProductInfo pit CROSS JOIN EmporiumSell est

    WHERE pit.ProdID=est.ProdID;

--内连接（自然连接）

--写法一（推荐）

SELECT \* FROM ProductInfo INNER JOIN EmporiumSell

    ON ProductInfo.ProdID=EmporiumSell.ProdID;

--写法二

SELECT \* FROM ProductInfo,EmporiumSell

    WHERE ProductInfo.ProdID=EmporiumSell.ProdID;

--我们发现这个和上面的带where的cross一样的，这里他们的确是一样的，他们

--的性能也是一样的，但是推荐使用inner join

--cross join是老语法里面的了

--左外连接left outer join和left join一样

SELECT \* FROM ProductInfo LEFT OUTER JOIN EmporiumSell

    ON ProductInfo.ProdID=EmporiumSell.ProdID;

--大家观察下这个结果，ProdID的顺序是连续9个050105,然后接着连续9个050201……

--右外连接

SELECT \* FROM ProductInfo RIGHT OUTER JOIN EmporiumSell

    ON ProductInfo.ProdID=EmporiumSell.ProdID;

--这个不是刚才的排序了哦

SELECT \* FROM EmporiumSell;

--全外连接

SELECT \* FROM ProductInfo FULL OUTER JOIN EmporiumSell

    ON ProductInfo.ProdID=EmporiumSell.ProdID

--这个和左连接时一样的

<span style="font-size:18px;" deep="5">USE commoditysell;

--我们先来看一道题来复习一下上次学习的内容，上次我们学习了多表查询的连接

--查询销售总量大于20的产品的信息

--我们来看，这个题目要求我们查询的是产品信息，那么产品信息是在ProductInfo这个表里吧

--我们肯定要用这个表了。又要是销售总量大于20，这个信息又在EmporiumSell这个表里，所以

--这个就是多表查询了，

SELECT \* FROM EmporiumSell;

SELECT \* FROM ProductInfo;

SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd

    LEFT JOIN Emporiumsell em ON pd.prodid=em.prodid

    WHERE em.sellamout>20;

SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd

    RIGHT JOIN Emporiumsell em ON pd.prodid=em.prodid

    WHERE em.sellamout>20;

SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd

    INNER JOIN Emporiumsell em ON pd.prodid=em.prodid

    WHERE em.sellamout>20;

--查询单价大于“060101”产品的单价的产品信息

--这个题目的查询的信息都在一个表了啊？他自己和自己连接可以吗？我们来看一下

SELECT \* FROM ProductInfo pd1

    LEFT JOIN ProductInfo pd2

    ON pd2.ProdID='060101' AND pd1.UnitPrice>pd2.UnitPrice;

--哦，这个结果里有很多NULL哦，怎么回事

--我们来看一下这个连接时怎么连的，ProductInfo表里共有9行记录，那么自己连接自己，先来看第一个

--条件，ProdID是'060101'，也就是把060101的所有信息都连接到这个表里，也就是ProductInfo表

--里每条记录后面都多了060101的这条记录的信息，而且还有一个条件pd1.UnitPrice>pd2.UnitPrice;

--所以，不满足这条记录的，后面060101的信息就是NULL了，怎么解决呢，把NULL屏蔽掉就可以了

SELECT \* FROM ProductInfo pit1

    RIGHT OUTER JOIN ProductInfo pit2

    ON pit2.ProdID='060101' AND pit1.UnitPrice>pit2.UnitPrice

    WHERE pit1.ProdID IS NOT NULL;

--这样就好了吧

--那有没有更好的方法呢，看下下面的方法

SELECT \* FROM ProductInfo

    WHERE UnitPrice>(SELECT UnitPrice FROM ProductInfo WHERE ProdID='060101');

--哦，这个结果比之前的好了啊，这个就是我们接下来要介绍的子查询了

--通过这个例子我们可以看出，子查询就是在查询里还有查询了

--再来看两个题目看一下查询

--查询单价大于平均单价的产品信息

--哦，这个不是很简单吗，用一个avg函数就行了啊

SELECT \* FROM ProductInfo

    WHERE UnitPrice>avg(UnitPrice);

--哦哦，报错了啊，这个就是我们前面说的了，where里不能包含聚合函数了

--我们可以用前面讲的使用having子句，也可以用今天的子查询

SELECT \* FROM ProductInfo

   WHERE UnitPrice>(SELECT avg(UnitPrice) FROM ProductInfo);

--我们前面用的子查询的表都是外查询的表，可不可以是其他表呢，当然可以了

--任务：查询已销售商品的厂家的所有信息（返回列表）

SELECT \* FROM ProductInfo

   WHERE ProdID IN (SELECT DISTINCT ProdID FROM EmporiumSell);

--这里的IN就是在……里的，相当于连续的OR了

--下面一个综合的题目，这个数据库是我们前面用过的了

--任务：对TeachingDB库进行查询，统计年龄大于平均年龄的教师的姓名（Tname）、

--性别（Tsex）、年龄（Tage）、职称（Trank）、所授课程名（Cname）、学分（Credit）、

--课时数（Coursehour）、选修该课程的学生的姓名（Sname），按照教师年龄降序排列。

--大家自己思考一下，下面是答案，这个不是唯一答案了，有很多种了

SELECT tin.Tname,tin.Tsex,tin.Tage,tin.Trank,cin.Cname,cin.Credit,cin.CouresHour,sn.Sname

    FROM TeacherInfo tin

        RIGHT JOIN Teach t on tin.Tno=t.Tno

       RIGHT JOIN CourseInfo cin ON t.Cno=cin.Cno

        RIGHT JOIN StudentCourse sc ON cin.Cno=sc.Cno

        RIGHT JOIN StudentInfo sn ON sn.Sno=sc.Sno

    WHERE Tage>(SELECT avg(Tage) FROM TeacherInfo) AND

        cin.Cno in (SELECT cin.Cno

           FROM TeacherInfo tin

                right join Teach t on tin.Tno=t.Tno

               right join CourseInfo cin ON t.Cno=cin.Cno

            WHERE Tage>(SELECT avg(Tage) FROM TeacherInfo))

    ORDER BY tin.Tage DESC

</span>

首先要知道的是我们现在学习的是关系型数据库。我们直观的看，一个数据库由多张表构成的，那么每张表有不重复的名称（表名），而且每个表是有多列组成的。可以这么理解一张表就是一个关系。关于表的还有一些概念要交代一下，第一个就是属性，一张关系表中的一个列就是一个属性，一个列又有它的名称和类型。元组，一张关系表中的一行数据就是元组，也成为记录。还有一个就是键，键是为在元组中快速查找设置的。键有主键和候选键之分。

今天要介绍的还有数据库的规范化，也就是第一范式、第二范式和第三范式。

第一范式：

–指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性  
（1）元组中的每个属性只能包含一个值  
（2）关系中的每个元组必须包含相同的数量  
（3）关系中的每个元组一定不能相同

第一条就是说每个单元格必须只有一个值，下面这个表就不符合要求了吧

|  |  |
| --- | --- |
| 序号 | 编号 |
| 1 | A101，A102 |
| 2 | B101 |

第二条就是每行都有相同数量的列，第三条就是一个表中不能有相同的行。

第二范式：

–满足第二范式必须先满足第一范式

–要求数据表中的每一行记录必须可以被唯一的区分

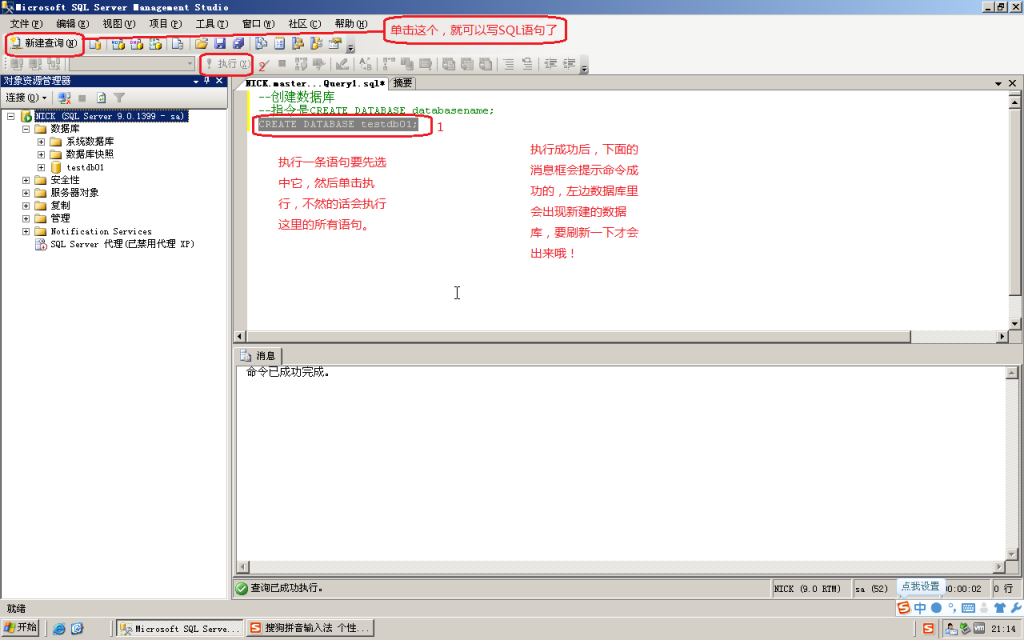
–通常需要为表添加主键

第三范式

–必须先满足第二范式

–要求一个数据库表中不包含已在其他表中已包含的非主关键字信息，也就是外键

* SQL 是用于访问和处理数据库的标准的计算机语言。SQL是种结构化的查询语言，它能够让我们使用数据库。那么SQL能做什么呢？
* SQL 面向数据库执行查询
* SQL 可从数据库取回数据
* SQL 可在数据库中插入新的纪录
* SQL 可更新数据库中的数据
* SQL 可从数据库删除记录
* SQL 可创建新数据库
* SQL 可在数据库中创建新表
* SQL 可在数据库中创建存储过程
* SQL 可在数据库中创建视图
* SQL 可以设置表、存储过程和视图的权限
* SQL主要分为两部分：数据操作语言 (DML) 和 数据定义语言 (DDL)。除此还有数据控制语言（DCL）。我们重点介绍DML和DDL。
* 查询和更新指令构成了 SQL 的 DML 部分，创建和删除等指令构成了DDL部分。
* 在具体介绍语言之前，我们先要知道我们的一条SQL语句是由主语句和子语句构成的，子语句是不能独立运行的。
* 在SQL Server 2005中打开SQL Server Management Studio，然后单击新建查询就可以输入SQL语句了。SQL Server中使用的是Transact-SQL。
* 还有一点要注意的是，在SQL Server中如果要执行这条语句，要选中它然后点执行。不然的话会执行所有的语句的。



* 废话少说我们就来学习关于数据库操作的语句吧。
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8159467" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8159467" \o "copy)
* --创建数据库 主语句是CREATE DATABASE
* --指令是CREATE DATABASE databasename;
* CREATE DATABASE testdb01;
* --在SQL Server中一个数据库中包含有三种文件：一种是.mdf，一种是.ndf还有.ldf。
* --mdf是主数据文件，ndf是次数据文件，ldf是日志文件
* --创建一个指定数据文件的数据库
* CREATE DATABASE testdb02
* ON(
* NAME=学生,
* --数据文件名
* FILENAME="C:\data\学生.mdf",
* --数据文件的物理位置
* SIZE=10MB,
* --初始大小，默认是MB
* --MAXSIZE=20MB,
* --最大大小
* MAXSIZE=UNLIMITED,
* --最大大小没有限制
* FILEGROWTH=1MB
* --文件自增长大小
* );
* --上面create database是主语句，on是子语句
* --指定主数据文件、次数据文件和日志文件
* CREATE DATABASE testdb03
* ON(
* NAME=Course,
* FILENAME='C:\data\course.mdf',
* SIZE=10MB,
* MAXSIZE=20MB,
* FILEGROWTH=10%
* --以百分之十增长
* ),
* (
* NAME=Teach,
* FILENAME='C:\data\teach.ndf',
* --通过文件的扩展名指定次数据文件
* SIZE=10MB,
* MAXSIZE=20MB,
* FILEGROWTH=10%
* )
* LOG ON(
* NAME=DB\_LOG,
* FILENAME='C:\data\DB\_LOG.LDF',
* SIZE=10MB,
* MAXSIZE=20MB,
* FILEGROWTH=10%
* );
* --删除数据库
* --DROP DATABASE database\_name
* DROP DATABASE testdb01;
* --修改数据库
* --主语句是alter database databasename
* --给testdb02增加一个数据文件
* ALTER DATABASE testdb02
* ADD FILE(
* NAME=教师,
* FILENAME='C:\data\教师.ndf',
* SIZE=10MB,
* MAXSIZE=20MB,
* FILEGROWTH=10%
* );
* --修改刚添加的文件教师.ndf
* ALTER DATABASE testdb02
* MODIFY FILE(
* NAME=教师,
* --这个必须是存在的文件名
* FILENAME='C:\data\教师.ndf',
* SIZE=13MB,
* --更改后的大小必须必之前的大
* MAXSIZE=10MB,
* --这个也是大小只能比之前的大
* FILEGROWTH=3MB
* );
* --删除刚刚增加的文件
* ALTER DATABASE testdb02
* REMOVE FILE 教师;
* --修改数据库名
* ALTER DATABASE testdb02
* MODIFY NAME=demodb02;
* --使用数据库，在数据库中建表这些操作之前要先使用数据库
* --USE databasename;
* USE testdb03;
* SQL Server 数据类型
* Character 字符串：

|  |  |  |
| --- | --- | --- |
| * 数据类型 | * 描述 | * 存储 |
| * char(n) | * 固定长度的字符串。最多 8,000 个字符。 | * n |
| * varchar(n) | * 可变长度的字符串。最多 8,000 个字符。 |  |
| * varchar(max) | * 可变长度的字符串。最多 1,073,741,824 个字符。 |  |
| * text | * 可变长度的字符串。最多 2GB 字符数据。 |  |

* Unicode 字符串：

|  |  |  |
| --- | --- | --- |
| * 数据类型 | * 描述 | * 存储 |
| * nchar(n) | * 固定长度的 Unicode 数据。最多 4,000 个字符。 |  |
| * nvarchar(n) | * 可变长度的 Unicode 数据。最多 4,000 个字符。 |  |
| * nvarchar(max) | * 可变长度的 Unicode 数据。最多 536,870,912 个字符。 |  |
| * ntext | * 可变长度的 Unicode 数据。最多 2GB 字符数据。 |  |

* Binary 类型：

|  |  |  |
| --- | --- | --- |
| * 数据类型 | * 描述 | * 存储 |
| * bit | * 允许 0、1 或 NULL |  |
| * binary(n) | * 固定长度的二进制数据。最多 8,000 字节。 |  |
| * varbinary(n) | * 可变长度的二进制数据。最多 8,000 字节。 |  |
| * varbinary(max) | * 可变长度的二进制数据。最多 2GB 字节。 |  |
| * image | * 可变长度的二进制数据。最多 2GB。 |  |

* Number 类型：

|  |  |  |
| --- | --- | --- |
| * 数据类型 | * 描述 | * 存储 |
| * tinyint | * 允许从 0 到 255 的所有数字。 | * 1 字节 |
| * smallint | * 允许从 -32,768 到 32,767 的所有数字。 | * 2 字节 |
| * int | * 允许从 -2,147,483,648 到 2,147,483,647 的所有数字。 | * 4 字节 |
| * bigint | * 允许介于 -9,223,372,036,854,775,808 和 9,223,372,036,854,775,807 之间的所有数字。 | * 8 字节 |
| * decimal(p,s) | * 固定精度和比例的数字。允许从 -10^38 +1 到 10^38 -1 之间的数字。 * p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。 * s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | * 5-17 字节 |
| * numeric(p,s) | * 固定精度和比例的数字。允许从 -10^38 +1 到 10^38 -1 之间的数字。 * p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。 * s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | * 5-17 字节 |
| * smallmoney | * 介于 -214,748.3648 和 214,748.3647 之间的货币数据。 | * 4 字节 |
| * money | * 介于 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之间的货币数据。 | * 8 字节 |
| * float(n) | * 从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据。参数 n 指示该字段保存 4 字节还是 8 字节。float(24) 保存 4 字节，而 float(53) 保存 8 字节。n 的默认值是 53。 | * 4 或 8 字节 |
| * real | * 从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据。 | * 4 字节 |

* Date 类型：

|  |  |  |
| --- | --- | --- |
| * 数据类型 | * 描述 | * 存储 |
| * datetime | * 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 3.33 毫秒。 | * 8 bytes |
| * datetime2 | * 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | * 6-8 bytes |
| * smalldatetime | * 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | * 4 bytes |
| * date | * 仅存储日期。从 0001 年 1 月 1 日 到 9999 年 12 月 31 日。 | * 3 bytes |
| * time | * 仅存储时间。精度为 100 纳秒。 | * 3-5 bytes |
| * datetimeoffset | * 与 datetime2 相同，外加时区偏移。 | * 8-10 bytes |
| * timestamp | * 存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量。 |  |

* 为了保证数据的完整性，我们对列还有一些约束。约束分为行约束、列约束和引用约束。行约束和列约束是针对单表的，引用约束是针对两个表的。
* 行约束：索引、唯一约束、主键（不为空；记录不能重复）、自动增长（针对int类型）
* 列约束：check约束、NOT NULL约束、Default约束
* 引用约束：外键约束
* 好了，我们开始学习指令吧。
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8163515" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8163515" \o "copy)
* CREATE DATABASE TestDB;
* USE TestDB;
* --建一个最简单的表
* CREATE TABLE test01(
* ID int,
* Sname nvarchar(100),
* grade varchar(50)
* )
* --空约束，NOT NULL不可以为空
* CREATE TABLE Student(
* ID int NOT NULL,
* Sname nvarchar(50),
* score real
* )
* -- 主键约束
* --PRIMARY KEY 约束唯一标识数据库表中的每条记录。
* --主键必须包含唯一的值。
* --主键列不能包含 NULL 值。
* --每个表应该都一个主键，并且每个表只能有一个主键。
* CREATE TABLE Student01(
* ID int PRIMARY KEY,
* Sname nvarchar(50),
* score real
* )
* --唯一约束，一个表中只能有一个主键约束，但是可以有多个约束。
* --Primary Key自动拥有唯一约束
* CREATE TABLE Student02(
* ID int PRIMARY KEY NOT NULL,
* Sname nvarchar(50) UNIQUE,
* score real
* )
* --默认约束
* CREATE TABLE Student03(
* ID int PRIMARY KEY NOT NULL,
* Sname nvarchar(50) UNIQUE,
* score real,
* address nvarchar(200) DEFAULT '钓鱼岛是中国的'
* )
* --自动增长约束
* CREATE TABLE Student04(
* ID int PRIMARY KEY NOT NULL IDENTITY(1001,1),
* Sname nvarchar(50) UNIQUE,
* score real
* )
* 先来看一下外键，外键主要是用来约束用于预防破坏表之间连接的动作。我们用一个例子来说明一下：
* "Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * Id\_P | * LastName | * FirstName | * Address | * City |
| * 1 | * Adams | * John | * Oxford Street | * London |
| * 2 | * Bush | * George | * Fifth Avenue | * New York |
| * 3 | * Carter | * Thomas | * Changan Street | * Beijin |

* "Orders" 表：

|  |  |  |
| --- | --- | --- |
| * Id\_O | * OrderNo | * Id\_P |
| * 1 | * 77895 | * 3 |
| * 2 | * 44678 | * 3 |
| * 3 | * 22456 | * 1 |
| * 4 | * 24562 | * 1 |

* “Persons" 表中的 "Id\_P" 列是 "Persons" 表中的 PRIMARY KEY。"Orders" 表中的 "Id\_P" 列是 "Orders" 表中的 FOREIGN KEY。通过建立外键就把两个表建立起了关联，这对查询很有用的。
* OK，我们在来看一下索引。索引主要是在不读取全表的时候加速查询。SQL Server中主要分为两种：聚集索引和非聚集索引。聚集索引主要特点就是调整索引表顺序的同时也会调整原表的顺序。当创建主键时自动创建聚集索引。一个表只能有一个主键，同样一个表也只能有一个聚集索引。一个聚集索引表最多有16列。非聚集索引就是调整表索引表的同时不会改变原表的顺序。当创建唯一约束时自动创建非聚集索引。一个非聚集索引表最多有249列。
* 唯一索引表示表中任何两个记录的索引值都不同的。
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8167717" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8167717" \o "copy)
* USE TestDB;
* --昨天我们讲的创建约束都是在定义列时直接建立的，今天我们在介绍一个指令CONSTRAINT
* --用CONSTRAINT语句建立主键、唯一、默认、检查约束
* CREATE TABLE Stu(
* Sno int,
* Sname varchar(50),
* Saddr varchar(100) CONSTRAINT DF\_Saddr DEFAULT '北京市',
* Sage int,
* --用constraint建立约束
* CONSTRAINT PK\_Sno PRIMARY KEY (Sno),
* CONSTRAINT UN\_Sname UNIQUE (Sname DESC),
* CONSTRAINT CK\_Sage CHECK (Sage>15)
* );
* --建立外键约束
* CREATE TABLE Scourse(
* Cno int,
* Cname varchar(50),
* Tno int,
* Sno int FOREIGN KEY REFERENCES Stu(Sno)
* --用constraint建立外键约束
* --CONSTRAINT FK\_Scoures\_Stu FOREIGN KEY(Sno)
* --REFERENCES Stu(Sno)
* --这里要注意Stu表中的Sno列一定要是主键哦^\_^
* --还有主键列和外键列的这个列类型要一致哦
* );
* --再建一个表，下面的例子用
* CREATE TABLE Teacher(
* Tno int,
* Tname varchar(50),
* Tage int,
* CONSTRAINT PK\_Teacher PRIMARY KEY(Tno)
* );
* --修改表，前面修改数据库我们用的是ALTER DATABASE，我们这里也是用ALTER
* --增加列
* ALTER TABLE Teacher
* ADD Taddr varchar(200);
* --修改列
* ALTER TABLE Teacher
* ALTER COLUMN Taddr varchar(100);
* --删除列
* ALTER TABLE Teacher
* DROP COLUMN Taddr;
* --增加约束，和上面用constraint建立约束一样的
* --这里就增加一个外键了
* ALTER TABLE Scourse
* ADD CONSTRAINT FK\_Teacher FOREIGN KEY(Tno)
* REFERENCES Teacher(Tno);
* --删除约束
* ALTER TABLE Scourse
* DROP FK\_Teacher;
* --搜狗输入法在SQL Server里好难用啊，总变成全角的
* --这里再增加一个默认约束，注意一下是怎么指定列的?
* ALTER TABLE Teacher
* ADD CONSTRAINT DF\_Tage DEFAULT 35 FOR Tage;
* --删除表
* DROP TABLE Scourse;
* --在使用数据库之前，要先USE哦
* USE TestDB;
* --创建表，然后填入数据
* CREATE TABLE Course(
* Cno char(4) PRIMARY KEY,
* Cname char(50),
* ClassHour char(4),
* Credit int
* );
* --插入数据使用INSERT INTO语句
* INSERT INTO Course VALUES(1001,'软件测试基础',45,4);
* --插入成功后，下面的消息框会提示“（1行受影响）”
* --插入一条记录，只提供部分列的值
* INSERT INTO Course(Cno,Cname) VALUES(1002,'数据库基础');
* --大家看下下面这条指令可以正确执行吗？
* INSERT INTO Course(Cname,Credit) VALUES('Java程序设计',4);
* --答案肯定是不能了，为什么呢？
* --因为Cno列有主键约束，所以不可以将NULL值插入Cno列
* --我们在插入几条记录
* INSERT INTO Course VALUES(1003,'Linux基础',40,4);
* INSERT INTO Course(Cno,Cname) VALUES(1004,'自动化测试基础');
* INSERT INTO Course VALUES(1005,'白盒测试基础',40,4);
* INSERT INTO Course(Cno,Cname) VALUES(1006,'Python语言');
* --下面来看一下怎么修改记录，修改记录就是更新表
* --我们用update指令
* UPDATE Course
* SET Cname='Loadrunner';
* --执行这条语句，表中所有的Cname都会变成Loadrunner了
* --那么可不可以使用这条语句更改Sno的值呢？
* UPDATE Course
* SET Cno=1005;
* --执行这条语句，会提示违反了主键约束，Cno不能有重复的
* --修改满足条件的记录
* UPDATE Course
* SET Cname='白盒测试技术' WHERE Cno=1005;
* --Cno为1005的Cname列又变为白盒测试技术了
* --WHERE子句在查询时还会经常使用的
* --更改Cno>1003的Credit为Credit+1
* UPDATE Course
* SET Credit=Credit+1 WHERE Cno>1004;
* --删除记录
* DELETE Course
* WHERE Cno=1005;
* --删除全部记录
* DELETE Course;
* 索引主要是在不读取全表的时候加速查询。索引就好比我们书的目录，它能帮助我们快速的找到要找的内容。SQL Server中主要分为两种：聚集索引和非聚集索引。聚集索引主要特点就是调整索引表顺序的同时也会调整原表的顺序。当创建主键时自动创建聚集索引。一个表只能有一个主键，同样一个表也只能有一个聚集索引。一个聚集索引表最多有16列。非聚集索引就是调整表索引表的同时不会改变原表的顺序。当创建唯一约束时自动创建非聚集索引。一个非聚集索引表最多有249列。这个聚集索引就好比我们新华字典的按拼音检索的那个目录，非聚集索引就好比那个按部首检索了。
* 唯一索引表示表中任何两个记录的索引值都不同的。
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8170751" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8170751" \o "copy)
* USE TestDB;
* --先建个表
* CREATE TABLE Book(
* bId int NOT NULL IDENTITY(0001,1),
* bName char(50),
* bPrice real,
* bPress char(100)
* );
* --建立索引
* CREATE INDEX IN\_book ON Book(bId ASC);
* --ASC是升序
* --DESC是降序
* --用两列建立索引
* CREATE INDEX IN\_book1 ON Book(bId ASC,bPrice DESC);
* --建立聚集索引
* CREATE CLUSTERED INDEX IN\_book2 ON Book(bId DESC);
* --建立非聚集索引
* CREATE NONCLUSTERED INDEX IN\_book3
* ON Book(bId DESC);
* --建立唯一索引
* CREATE UNIQUE INDEX IN\_book4
* ON Book(bID ASC);
* --大家看一下这个索引，能不能建立？
* CREATE INDEX IN\_book ON Book(bId,bPrice DESC);
* --失败了吧，因为我们前面建过一个名为IN\_book的索引了
* --如果我们想要建一个索引，如果存在就删除它，建立新的，该怎么办呢？
* CREATE INDEX IN\_book ON Book(bId,bPrice DESC)
* WITH DROP\_EXISTING;
* --修改索引
* --禁用IN\_book索引
* ALTER INDEX IN\_book
* ON Book DISABLE;
* --启用索引
* ALTER INDEX IN\_book
* ON Book REBUILD;
* --删除索引
* DROP INDEX Book.IN\_book2;
* 今天我们学习SQL知识，我们今天开始我们的重点select查询语句。
* 今天用到的知识：SQL通配符和SQL函数。
* 先来看通配符：

|  |  |
| --- | --- |
| * 通配符 | * 描述 |
| * % | * 替代一个或多个字符 |
| * \_ | * 仅替代一个字符 |
| * [charlist] | * 字符列中的任何单一字符 |
| * [^charlist] * 或者 * [!charlist] | * 不在字符列中的任何单一 |

* SQL函数：

|  |  |
| --- | --- |
| * 函数 | * 描述 |
| * [AVG(column)](http://blog.csdn.net/xc5683/article/details/sql_func_avg.asp.htm" \o "SQL AVG 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回某列的行数 |
|  |  |
|  |  |
|  |  |
| * [COUNT(column)](http://blog.csdn.net/xc5683/article/details/sql_func_count.asp.htm" \o "SQL COUNT 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回某列的行数（不包括NULL值） |
| * [COUNT(\*)](http://blog.csdn.net/xc5683/article/details/sql_func_count_ast.asp.htm" \o "SQL COUNT(*) 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回被选行数 |
| * [COUNT(DISTINCT column)](http://blog.csdn.net/xc5683/article/details/sql_func_count_distinct.asp.htm" \o "SQL COUNT DISTINCT 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回相异结果的数目 |
| * [FIRST(column)](http://blog.csdn.net/xc5683/article/details/sql_func_first.asp.htm" \o "SQL FIRST 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回在指定的域中第一个记录的值（SQLServer2000 不支持） |
| * [LAST(column)](http://blog.csdn.net/xc5683/article/details/sql_func_last.asp.htm" \o "SQL LAST 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回在指定的域中最后一个记录的值（SQLServer2000 不支持） |
| * [MAX(column)](http://blog.csdn.net/xc5683/article/details/sql_func_max.asp.htm" \o "SQL MAX 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回某列的最高值 |
| * [MIN(column)](http://blog.csdn.net/xc5683/article/details/sql_func_min.asp.htm" \o "SQL MIN 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回某列的最低值 |
|  |  |
|  |  |
| * [SUM(column)](http://blog.csdn.net/xc5683/article/details/sql_func_sum.asp.htm" \o "SQL SUM 函数" \t "http://blog.csdn.net/xc5683/article/details/_blank) | * 返回某列的总和 |
|  |  |
|  |  |

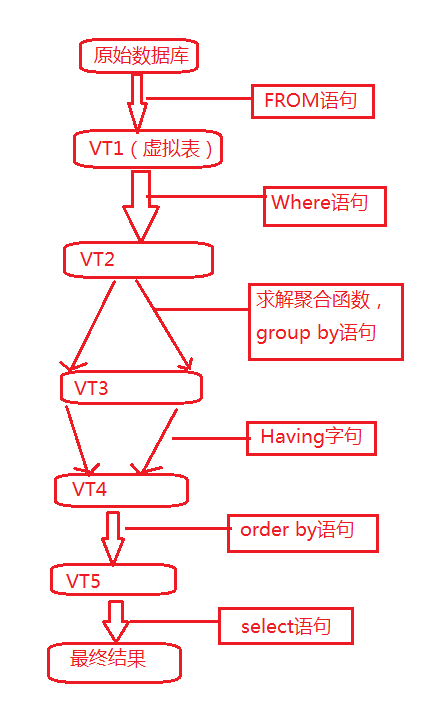
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8176079" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8176079" \o "copy)
* USE TeachingDB;
* --查询Student中的所有信息
* SELECT \* FROM StudentInfo;
* --查询student中的学生的Sname和Saddress信息
* SELECT Sname,Saddress FROM StudentInfo;
* --查询student中的学生的Sname和Saddress信息，并设定别名
* SELECT Sname AS '姓名',Saddress AS '地址' FROM StudentInfo;
* SELECT Sname '姓名',Saddress '地址' FROM StudentInfo;
* SELECT Sname 姓名,Saddress 地址 FROM StudentInfo;
* SELECT Sname AS 姓名,Saddress AS 地址 FROM StudentInfo;
* --这里的四种方法都行，但是建议用第一种方法
* --查询student表中的前5行信息，用top
* SELECT TOP 5 Sname,Sage FROM StudentInfo;
* --查询student表中前10%的信息
* SELECT TOP 10 PERCENT \* FROM StudentInfo;
* --这个表中共有30条记录，所以这个结果显示前3条信息
* --那么查询倒数5行信息呢？SQL中没有down之类的语句，大家思考一下这个问题
* --下面用下面这个表看一下
* SELECT \* FROM StudentCourse;
* --这个表中的Sno有重复的项，共60条记录啊
* --那么怎么去掉重复项呢？
* SELECT DISTINCT Sno FROM StudentCourse;
* --现在就剩30条了

* --查询地址为四川绵阳的学生
* SELECT \* FROM StudentInfo
* WHERE Saddress='四川绵阳';
* --查询年龄大于19岁的学生信息
* SELECT \* FROM StudentInfo
* WHERE Sage>19;
* --查询年龄在18和20之间的
* SELECT \* FROM StudentInfo
* WHERE Sage>18 AND Sage<20;
* SELECT \* FROM StudentInfo
* WHERE Sage BETWEEN 18 AND 20;
* --注意观察这两个区别？范围有区别的哦
* --查询学生地址为四川自贡和四川成都的学生信息
* SELECT \* FROM StudentInfo
* WHERE Saddress='四川自贡' OR Saddress='四川成都';
* --查询学生地址不是自贡的学生信息
* SELECT \* FROM StudentInfo
* WHERE NOT(Saddress='四川自贡');
* SELECT \* FROM StudentInfo
* WHERE Saddress!='四川自贡';
* SELECT \* FROM StudentInfo
* WHERE Saddress<>'四川自贡';
* --查询Sno以02开头的学生信息
* SELECT \* FROM StudentInfo
* WHERE Sno LIKE '02%';
* --查询Sname中带有小字的学生信息
* SELECT \* FROM StudentInfo
* WHERE Sname LIKE '%[小]%';
* SELECT \* FROM RankPayment;
* --查询平均payment
* SELECT AVG(payment) FROM RankPayment;
* select \* from courseinfo;
* --统计所有的行数
* SELECT COUNT(\*) FROM CourseInfo;
* --查询coursehour最大的课程信息
* SELECT MAX(CouresHour) FROM CourseInfo;
* --查询coursehour最小的课程信息
* SELECT min(CouresHour) FROM CourseInfo;
* --查询coureshour大于平均课时的课程信息
* SELECT \* FROM CourseInfo
* WHERE CouresHour>avg(coureshour);
* USE commoditysell;
* SELECT \* FROM EmporiumSell
* --排序子句：Order By

* --任务：查询ProductInfo表中所有产品的ProdName、UnitPrice，按UnitPrice降序排列
* SELECT \* FROM ProductInfo
* ORDER BY UnitPrice DESC;
* --查询最贵的2种电视机
* SELECT TOP 2 \* FROM ProductInfo
* WHERE ProdName LIKE '%电视机%'
* ORDER BY UnitPrice DESC;
* --用这个就可以解决我们昨天查询后5条的信息了吧。
* --任务：查询SalerInfo中SalerName、SalerAge、SalerSex、Salary，按SaleAge升序、Salary降序
* SELECT \* FROM SalerInfo
* ORDER BY SalerAge ASC,Salary DESC;
* --分组子句：Group By
* --任务：简单分组：查询按SalerAge分组的员工的平均Salary
* --大家看一下这个会正确的执行吗？
* SELECT \* FROM SalerInfo
* GROUP BY SalerAge;
* SELECT Salary FROM SalerInfo
* GROUP BY SalerAge;
* --上面这两条都出现了同样的问题，就是“因为该列没有包含在聚合函数或 GROUP BY 子句中”
* --如何解决呢，就是我们下面的两条语句
* SELECT avg(Salary) FROM SalerInfo
* GROUP BY SalerAge;
* SELECT SalerAge,Salary FROM SalerInfo
* GROUP BY SalerAge,Salary;
* --通过上述的例子要记住一点：
* --使用group by子句，查询的列必须出现在group by子句中，后者使用聚合函数
* --任务：查询按各年龄段的员工人数
* SELECT SalerAge,count(SalerId) FROM SalerInfo
* GROUP BY SalerAge;
* SELECT SalerAge,count(\*) FROM SalerInfo
* GROUP BY SalerAge
* HAVING avg(Salary)>1500;
* 上面是我们昨天剩的内容，下面开始今天的内容。
* [sql] [view plain](http://blog.csdn.net/xc5683/article/details/8180051" \o "view plain) [copy](http://blog.csdn.net/xc5683/article/details/8180051" \o "copy)
* ---------------------------------------------------------------------------------------------------------------------
* --            多表查询
* ---------------------------------------------------------------------------------------------------------------------
* USE commoditysell;
* --交叉连接
* SELECT \* FROM ProductInfo;--这个表共有9列
* SELECT \* FROM EmporiumSell;--这个表共有51列
* SELECT \* FROM ProductInfo CROSS JOIN EmporiumSell;
* --这个结果是459列，就是9\*51了，所以交叉连接就是A表的一条记录对应B表的所有记录
* --这里执行的是笛卡尔积
* --我们使用where子句看一下
* SELECT \* FROM ProductInfo CROSS JOIN EmporiumSell
* WHERE ProductInfo.ProdID=EmporiumSell.ProdID;
* --这下就是51条了
* --可以使用表的别名
* SELECT \* FROM ProductInfo pit CROSS JOIN EmporiumSell est
* WHERE pit.ProdID=est.ProdID;
* --内连接（自然连接）
* --写法一（推荐）
* SELECT \* FROM ProductInfo INNER JOIN EmporiumSell
* ON ProductInfo.ProdID=EmporiumSell.ProdID;
* --写法二
* SELECT \* FROM ProductInfo,EmporiumSell
* WHERE ProductInfo.ProdID=EmporiumSell.ProdID;
* --我们发现这个和上面的带where的cross一样的，这里他们的确是一样的，他们
* --的性能也是一样的，但是推荐使用inner join
* --cross join是老语法里面的了
* --左外连接left outer join和left join一样
* SELECT \* FROM ProductInfo LEFT OUTER JOIN EmporiumSell
* ON ProductInfo.ProdID=EmporiumSell.ProdID;
* --大家观察下这个结果，ProdID的顺序是连续9个050105,然后接着连续9个050201……
* --右外连接
* SELECT \* FROM ProductInfo RIGHT OUTER JOIN EmporiumSell
* ON ProductInfo.ProdID=EmporiumSell.ProdID;
* --这个不是刚才的排序了哦
* SELECT \* FROM EmporiumSell;
* --全外连接
* SELECT \* FROM ProductInfo FULL OUTER JOIN EmporiumSell
* ON ProductInfo.ProdID=EmporiumSell.ProdID
* --这个和左连接时一样的
* 他们之间的区别是：
* JOIN: 如果表中有至少一个匹配，则返回行
* LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行
* RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行
* FULL JOIN: 只要其中一个表中存在匹配，就返回行
* <span style="font-size:18px;" deep="5">USE commoditysell;
* --我们先来看一道题来复习一下上次学习的内容，上次我们学习了多表查询的连接
* --查询销售总量大于20的产品的信息
* --我们来看，这个题目要求我们查询的是产品信息，那么产品信息是在ProductInfo这个表里吧
* --我们肯定要用这个表了。又要是销售总量大于20，这个信息又在EmporiumSell这个表里，所以
* --这个就是多表查询了，
* SELECT \* FROM EmporiumSell;
* SELECT \* FROM ProductInfo;
* SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd
* LEFT JOIN Emporiumsell em ON pd.prodid=em.prodid
* WHERE em.sellamout>20;
* SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd
* RIGHT JOIN Emporiumsell em ON pd.prodid=em.prodid
* WHERE em.sellamout>20;
* SELECT pd.ProdID,pd.ProdName,UnitPrice,ProdFact FROM ProductInfo pd
* INNER JOIN Emporiumsell em ON pd.prodid=em.prodid
* WHERE em.sellamout>20;

* --查询单价大于“060101”产品的单价的产品信息
* --这个题目的查询的信息都在一个表了啊？他自己和自己连接可以吗？我们来看一下
* SELECT \* FROM ProductInfo pd1
* LEFT JOIN ProductInfo pd2
* ON pd2.ProdID='060101' AND pd1.UnitPrice>pd2.UnitPrice;
* --哦，这个结果里有很多NULL哦，怎么回事
* --我们来看一下这个连接时怎么连的，ProductInfo表里共有9行记录，那么自己连接自己，先来看第一个
* --条件，ProdID是'060101'，也就是把060101的所有信息都连接到这个表里，也就是ProductInfo表
* --里每条记录后面都多了060101的这条记录的信息，而且还有一个条件pd1.UnitPrice>pd2.UnitPrice;
* --所以，不满足这条记录的，后面060101的信息就是NULL了，怎么解决呢，把NULL屏蔽掉就可以了
* SELECT \* FROM ProductInfo pit1
* RIGHT OUTER JOIN ProductInfo pit2
* ON pit2.ProdID='060101' AND pit1.UnitPrice>pit2.UnitPrice
* WHERE pit1.ProdID IS NOT NULL;
* --这样就好了吧
* --那有没有更好的方法呢，看下下面的方法
* SELECT \* FROM ProductInfo
* WHERE UnitPrice>(SELECT UnitPrice FROM ProductInfo WHERE ProdID='060101');
* --哦，这个结果比之前的好了啊，这个就是我们接下来要介绍的子查询了
* --通过这个例子我们可以看出，子查询就是在查询里还有查询了
* --再来看两个题目看一下查询
* --查询单价大于平均单价的产品信息
* --哦，这个不是很简单吗，用一个avg函数就行了啊
* SELECT \* FROM ProductInfo
* WHERE UnitPrice>avg(UnitPrice);
* --哦哦，报错了啊，这个就是我们前面说的了，where里不能包含聚合函数了
* --我们可以用前面讲的使用having子句，也可以用今天的子查询
* SELECT \* FROM ProductInfo
* WHERE UnitPrice>(SELECT avg(UnitPrice) FROM ProductInfo);
* --我们前面用的子查询的表都是外查询的表，可不可以是其他表呢，当然可以了
* --任务：查询已销售商品的厂家的所有信息（返回列表）
* SELECT \* FROM ProductInfo
* WHERE ProdID IN (SELECT DISTINCT ProdID FROM EmporiumSell);
* --这里的IN就是在……里的，相当于连续的OR了
* --下面一个综合的题目，这个数据库是我们前面用过的了
* --任务：对TeachingDB库进行查询，统计年龄大于平均年龄的教师的姓名（Tname）、
* --性别（Tsex）、年龄（Tage）、职称（Trank）、所授课程名（Cname）、学分（Credit）、
* --课时数（Coursehour）、选修该课程的学生的姓名（Sname），按照教师年龄降序排列。
* --大家自己思考一下，下面是答案，这个不是唯一答案了，有很多种了
* SELECT tin.Tname,tin.Tsex,tin.Tage,tin.Trank,cin.Cname,cin.Credit,cin.CouresHour,sn.Sname
* FROM TeacherInfo tin
* RIGHT JOIN Teach t on tin.Tno=t.Tno
* RIGHT JOIN CourseInfo cin ON t.Cno=cin.Cno
* RIGHT JOIN StudentCourse sc ON cin.Cno=sc.Cno
* RIGHT JOIN StudentInfo sn ON sn.Sno=sc.Sno
* WHERE Tage>(SELECT avg(Tage) FROM TeacherInfo) AND
* cin.Cno in (SELECT cin.Cno
* FROM TeacherInfo tin
* right join Teach t on tin.Tno=t.Tno
* right join CourseInfo cin ON t.Cno=cin.Cno
* WHERE Tage>(SELECT avg(Tage) FROM TeacherInfo))
* ORDER BY tin.Tage DESC
* </span>

|  |  |
| --- | --- |
| * 语句 | * 语法 |
| * AND / OR | * SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| * ALTER TABLE (add column) | * ALTER TABLE table\_name  ADD column\_name datatype |
| * ALTER TABLE (drop column) | * ALTER TABLE table\_name  DROP COLUMN column\_name |
| * AS (alias for column) | * SELECT column\_name AS column\_alias FROM table\_name |
| * AS (alias for table) | * SELECT column\_name FROM table\_name  AS table\_alias |
| * BETWEEN | * SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| * CREATE DATABASE | * CREATE DATABASE database\_name |
| * CREATE INDEX | * CREATE INDEX index\_name ON table\_name (column\_name) |
| * CREATE TABLE | * CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, ....... ) |
| * CREATE UNIQUE INDEX | * CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| * CREATE VIEW | * CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| * DELETE FROM | * DELETE FROM table\_name  (Note: Deletes the entire table!!) * or * DELETE FROM table\_name WHERE condition |
| * DROP DATABASE | * DROP DATABASE database\_name |
| * DROP INDEX | * DROP INDEX table\_name.index\_name |
| * DROP TABLE | * DROP TABLE table\_name |
| * GROUP BY | * SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 |
| * HAVING | * SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 HAVING SUM(column\_name2) condition value |
| * IN | * SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| * INSERT INTO | * INSERT INTO table\_name VALUES (value1, value2,....) * or * INSERT INTO table\_name (column\_name1, column\_name2,...) VALUES (value1, value2,....) |
| * LIKE | * SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| * ORDER BY | * SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| * SELECT | * SELECT column\_name(s) FROM table\_name |
| * SELECT \* | * SELECT \* FROM table\_name |
| * SELECT DISTINCT | * SELECT DISTINCT column\_name(s) FROM table\_name |
| * SELECT INTO (used to create backup copies of tables) | * SELECT \* INTO new\_table\_name FROM original\_table\_name * or * SELECT column\_name(s) INTO new\_table\_name FROM original\_table\_name |
| * TRUNCATE TABLE (deletes only the data inside the table) | * TRUNCATE TABLE table\_name |
| * UPDATE | * UPDATE table\_name SET column\_name=new\_value [, column\_name=new\_value] WHERE column\_name=some\_value |
| * WHERE | * SELECT column\_name(s) FROM table\_name WHERE condition |

* 我们这段主要是学习了SQL的增删改查语句，其中查询是我们的重点。我们是以SQL Server 2005来学习的SQL语句，关于SQL Server的图形界面操作，和关于视图、数据库备份还原这些大家下来要下来看一下。
* 关于Oracle，我们都已经学习了SQL语句，那么Oracle其实已经学习了一大半，就接下来自己学习Oracle主要是看一下他的概念，Oracle和SQL Server还是有很大不同的，熟悉他的表空间这些概念，了解他的图形界面，和备份还原等操作。如果想要继续学习可以看一下MLDN李兴华老师的Oracle视频。自己下来看一下《Oracle9i参考手册》。
* 好，今天的总结我主要想要介绍一下SQL语句（主要是Select）执行的顺序，了解了这个就可以知道为什么where里不能有聚合函数了。
* 我画了一个图，画的不太好了IMG_256
* 
* 从这个图里我们就可以看出来，为什么where里不能包含聚合函数了吧，因为SQL是先执行where后执行聚合函数，如果where里含有聚合函数，那么这时聚合函数还没有执行，也就没有结果供where使用了。大家好好消化一下这个图。