第六章 基础SQL语言

PART 01
SQL
语言概述

PART 02
数据定义语言

PART 03
数据查询语句

PART 04
数据更新语句

PART 05
视图

PART 06
数据定义
中的完整性约束

PART 07
索引

- SEQUEL：最早的SQL原型是IBM的研究人员在20世纪70年代开发的，该原型被命名为SEQUEL。
- SQL-86：第一个SQL标准SQL-86由美国ANSI于1986年颁布。随后，ISO于1987年采纳它为国际标准。
- SQL-92：在1992年，ISO和ANSI又共同颁布了新的SQL标准，称之为SQL-92，又称为SQL2，对SQL-86中存在的不足进行改进。
- SQL-1999：1999年新颁布的SQL：1999（又称为SQL3）标准中增加了一些额外的特征以支持面向对象的数据管理。
- SQL-2003：第四版的SQL标准是2003年颁布的SQL：2003。
- SQL-2016：这是SQL数据库查询语言标准的第八次修订版本。

- 数据定义功能，DDL（Data Definition Language）

提供命令定义关系模式、索引、视图，包括创建CREATE，修改ALTER以及删除DROP等命令。

- 数据操纵功能，DML（Data Manipulation Language）

提供命令对存储在数据库中的数据进行查询和修改操作，包括查询数据SELECT、插入数据INSERT、更新数据UPDATE、删除数据DELETE等命令。

- 数据控制命令，DCL（Data Control Language）

提供对关系和视图的授权命令、事务的控制命令以及并发控制中的加锁操作等。

- 综合统一
  - 集DDL、DML、DCL的功能于一体
  - 数据操作符统一
- 高度非过程化
  - 只需提出“做什么”，而无需指明“怎么做”
  - 无需了解存取路径，存取路径的选择以及SQL语句的操作过程由系统自动完成
- 面向集合的操作方式
  - 操作对象和结果均为集合
- 可独立使用又可嵌入主语言使用
  - 图形化用户接口GUI
  - 终端输入命令
  - 嵌入到JAVA，C++等高级语言中

数据类型	申明
数字类型	NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, DOUBLE, REAL
字符串类型	CHAR, VARCHAR, TEXT
二进制串类型	BINARY, VARBINARY, BLOB
布尔类型	BOOLEAN
日期时间类型	DATE, TIME, DATETIME, TIMESTAMP
时间间隔类型	INTERVAL
XML类型	XML

函数	返回值
CHAR_LENGTH(string)	字符串长度
LOWER(string)	将字符串全部转换为小写字母
UPPER(string)	将字符串全部转换为大写字母
SUBSTRING(source,n,len)	取子串，从第n个字符开始，长度为len
CURRENT_DATE()	当前日期
MAX(column)	指定列的最大值
MIN(column)	指定列的最小值
AVG(column)	指定列的平均值
SUM(column)	指定列的总和

2. 数据库基本结构定义

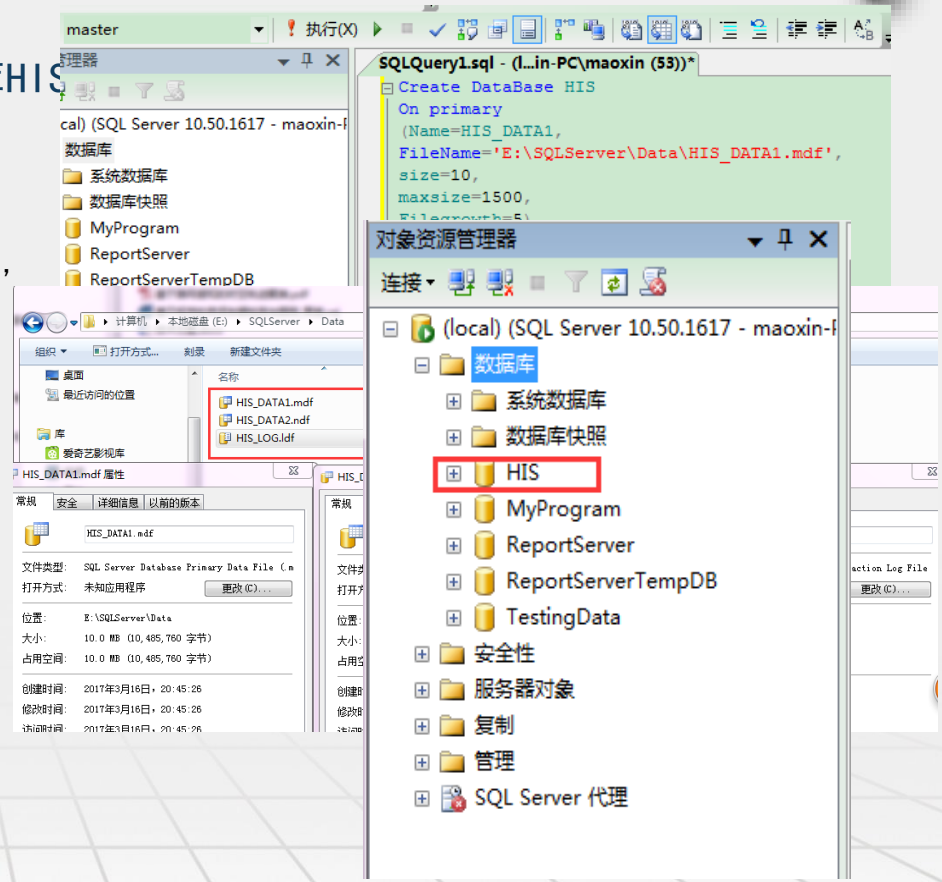
- 数据库是包含多个对象的集合，包括基表、视图、索引、存储过程、与数据库安全性有关的控制机制及其他对象。
- 数据库的创建一般由DBA来完成。ISO标准并没有对如何创建数据库进行详细规定，因此数据库的创建过程在不同的商用数据库系统中差异较大。
- 在SQL SERVER中使用一组操作系统文件来映射数据库，数据库中的所有数据和对象（如表、存储过程、触发器和视图）都存储在三类操作系统文件中：
  - 第一类是主文件，扩展名为mdf。包含数据库的启动信息及数据信息；
  - 第二类是次要文件或从文件，扩展名为ndf。含有主文件以外的所有数据；
  - 第三类是事务日志，扩展名为ldf。包含用于恢复数据库的日志信息。

- 创建数据库语法形式如下：

```
CREATE DATABASE <数据库名>
[<On  Primary>
    ([Name = 系统使用的逻辑名],
    [Filename = 完全限定的NT Server文件名],
    [Size = 文件的初始大小],
    [MaxSize = 最大的文件尺寸],
    [FileGrowth = 系统的扩展文件量])…]
    [<Log On>
    ([Name = 系统使用的逻辑名],
    [Filename = 完全限定的NT Server文件名],
    [Size = 文件的初始大小],
    [FileGrowth = 系统的扩展文件量])]
```

- 例：指定参数，创建医院信息系统数据库HIS

```
CREATE DATABASE HIS
ON Primary
( NAME = HIS_DATA1,
    FILENAME = 'd:\data\ HIS_DATA1.mdf',
    SIZE = 10,
    MAXSIZE = 1500,
    FILEGROWTH = 5 )
( NAME = HIS_DATA2,
    FILENAME 'd:\data\ HIS_DATA2.ndf',
    SIZE = 10,
    MAXSIZE = 500,
    FILEGROWTH = 5 )
LOG ON
( NAME = HIS_LOG,
    FILENAME = 'd:\data\ HIS_LOG.ldf',
    SIZE = 5MB,
    MAXSIZE = 500MB,
    FILEGROWTH = 5MB )
```



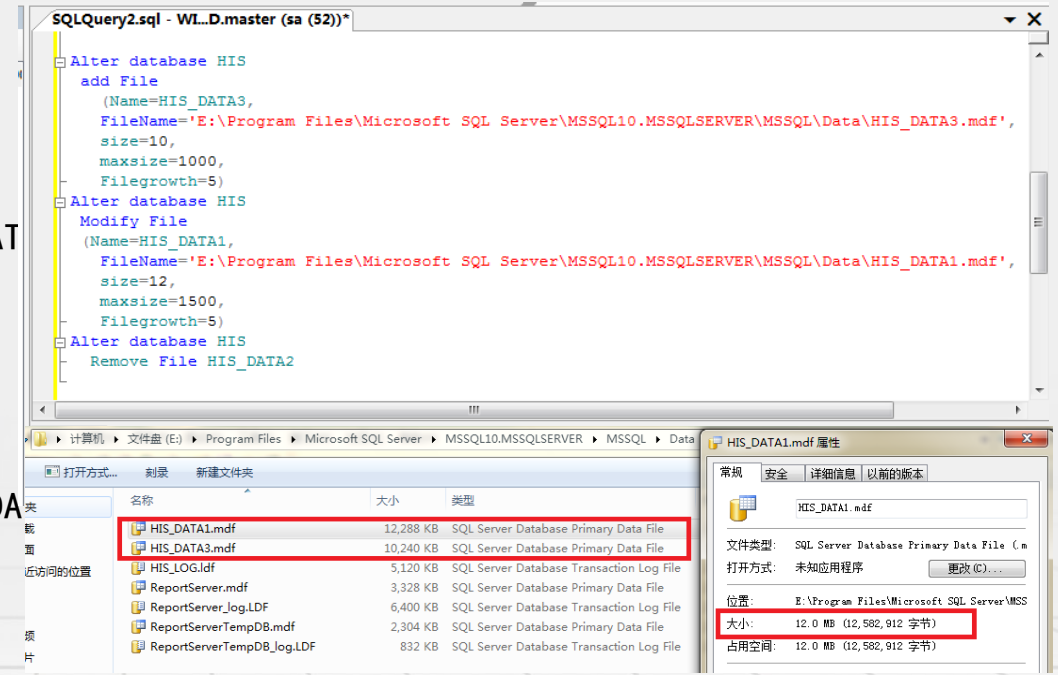
- 修改数据库
  - 扩充数据库的数据或事务日志存储空间；
  - 收缩分配给数据库的数据或事务日志空间；
  - 添加或删除数据和事务日志文件；
  - 更改数据库的配置设置；
  - 更改数据库名称；
  - 更改数据库的所有者等。

- 修改数据库语法如下：

```
ALTER DATABASE <数据库名>
[<Add File>
(<Name = 系统使用的逻辑名>,
[Filename = 完全限定的NT Server文件名],
[Size = 文件的初始大小],
[MaxSize = 最大的文件尺寸],
[FileGrowth = 系统的扩展文件量])…]
[<Modify File>
(<Name = 系统使用的逻辑名>,
[Filename = 完全限定的NT Server文件名],
[Size = 文件的初始大小],
[MaxSize    = 最大的文件尺寸],
[FileGrowth = 系统的扩展文件量])…]
[<Remove File> <系统使用文件的逻辑名>,…]
[<Add Log File>
(<Name = 系统使用的逻辑名>,
[Filename = 完全限定的NT Server文件名],
[Size = 文件的初始大小],
[MaxSize = 最大的文件尺寸],
[FileGrowth = 系统的扩展文件量])…]
```

- 例：在医院信息系统数据库HIS中增加一个新数据库文件HIS_DATA3，同时要修改原数据库文件HIS_DATA1的最大文件尺寸为1500MB，并且要删除医院信息系统数据库HIS的次要文件HIS_DATA2。

```
ALTER DATABASE HIS
Add File
( NAME = HIS_DATA3,
    FILENAME = 'd:\data\HIS_DAT
    SIZE = 10,
    MAXSIZE = 1000,
    FILEGROWTH = 5 )
Modify File
( NAME = HIS_DATA1,
    FILENAME = 'd:\data\ HIS_DA
    SIZE = 12,
    MAXSIZE = 1500,
    FILEGROWTH = 5 )
Remove File HIS_DATA2
```



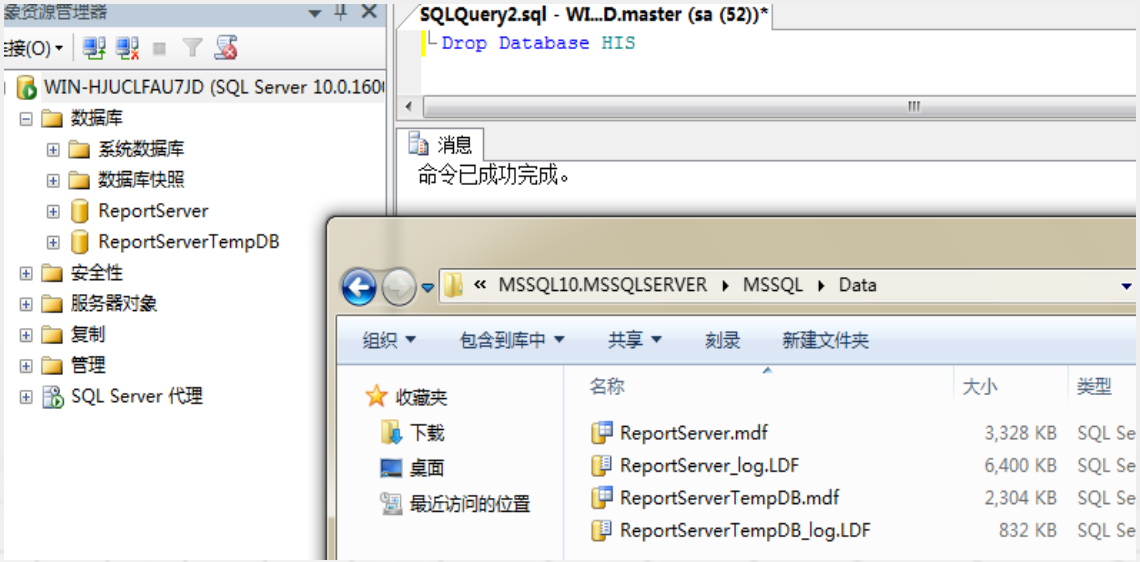
- 删除数据库
  - 当不再需要数据库，或者数据库数据被移到其他数据库或服务器时，即可删除该数据库。
  - 数据库删除之后，该数据库的文件及其数据都从服务器的磁盘删除。
  - 不能删除系统数据库msdb，master，model和tempdb。

- 删除数据库的命令格式为：

DROP DATABASE 需要删除的数据库名

- 例：删除数据库HIS

DROP DATABASE HIS



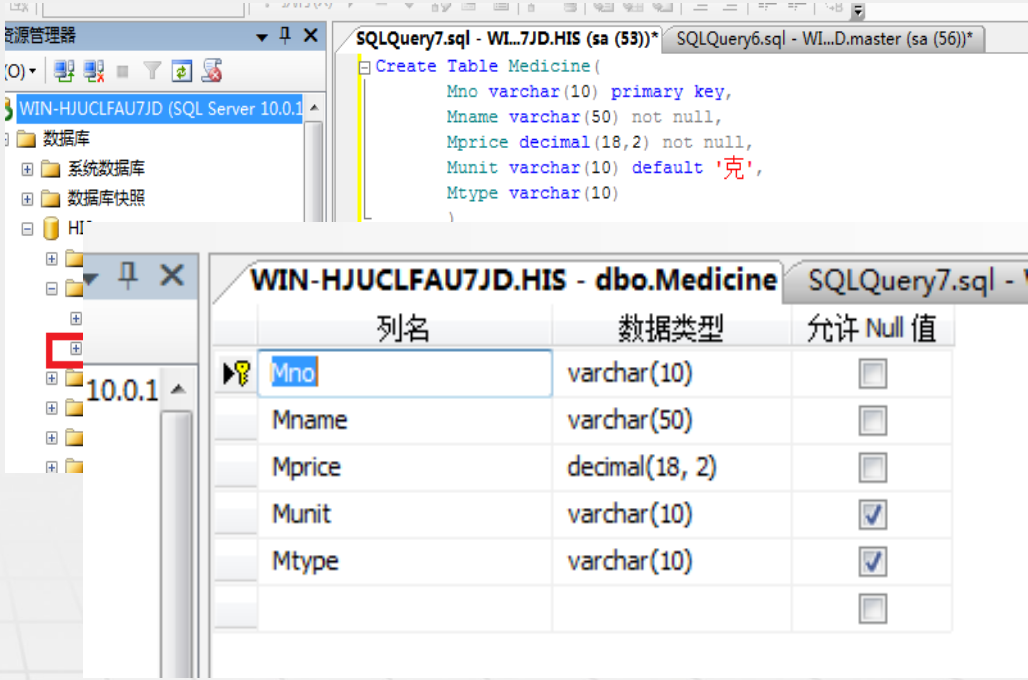
创建表的语法

```
CREATE TABLE  基表名（
    <列名1> <列类型> <列约束>,
    <列名2> <列类型> <列约束>,
    …
    <列名n> <列类型> <列约束>,
    < 表约束> | [ { PRIMARY KEY | UNIQUE } [ ,...] ]
）
```

表名形式为：[数据库名[.拥有者.]]表名

- 示例：在HIS数据库中，建立药品基本信息表Medicine如下：

```
CREATE TABLE Medicine (
    Mno VARCHAR(10) PRIMARY KEY,
    Mname VARCHAR(50) NOT NULL,
    Mprice DECIMAL(18,2) NOT NULL,
    Munit VARCHAR(10) DEFAULT '克',
    Mtype VARCHAR(10)
)
```



- 修改表的语法

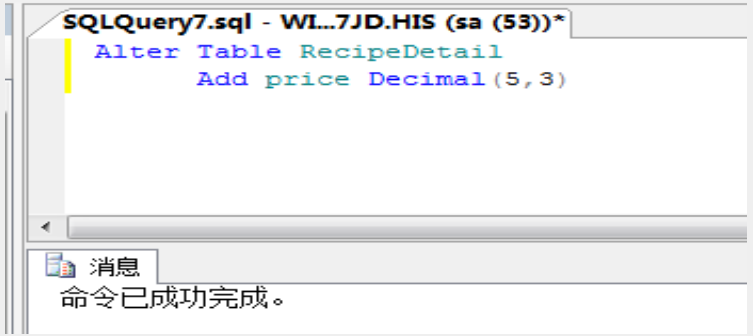
```
ALTER TABLE 〈基表名〉
    [ ALTER COLUMN <列名> <数据类型>],
    [ ADD  <新列名> <数据类型> <约束规则>],
    [ DROP  <列名>],
    [ DROP  <约束规则>];
```

- <表名>：要修改的基本表
- ADD子句：增加新列和新的完整性约束条件
- DROP子句：删除指定的完整性约束条件
- ALTER子句：用于修改列名和数据类型

- 修改表示例：在医院信息系统的数据库中，如果医院的某些药品价钱随着市场供求在不断调整，不同阶段的处方药品价钱不一样，因此在处方明细表RecipeDetail需要增加一列存储药品单价。

```
ALTER TABLE RecipeDetail

ADD Price Decimal （ 5,3 ）
```



- 删除表的语法如下：

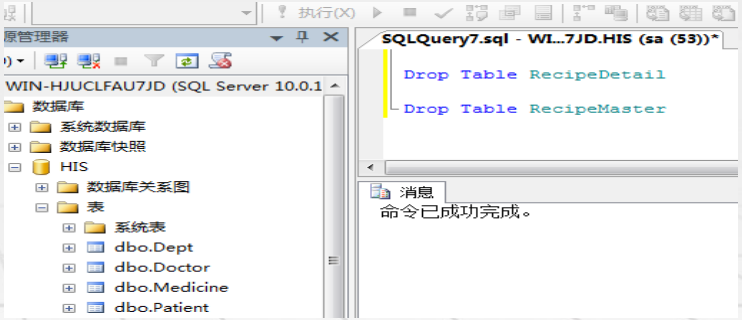
```
DROP TABLE <表名> [RESTRICT|CASCADE];
```

  - RESTRICT：拥有表的对象（Check、Foreign Key、视图、触发器、存储过程、函数等）时禁止删除；
  - CASCADE：级联删除表的所有对象

- 删除表示例： 删除RecipeDetail和RecipeMaster表：

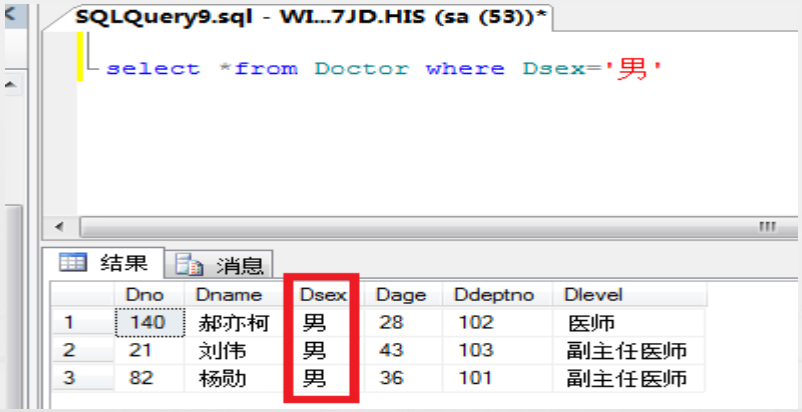
```
DROP TABLE RecipeDetail;

DROP TABLE RecipeMaster;
```



- 查询语句示例
  - 查询医生基本信息表中所有男医生的基本信息

```
SELECT * FROM Doctor
WHERE Dsex='男'
```



	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	140	郝亦柯	男	28	102	医师
2	21	刘伟	男	43	103	副主任医师
3	82	杨勋	男	36	101	副主任医师

- SQL查询语句的结构特征，即SELECT-FROM-WHERE形式

  - SELECT子句说明哪些属性列显示输出
  - FROM子句给出查询的表名
  - WHERE子句是一个条件子句，就像关系代数中的选择条件，返回满足条件的元组。

查询语句基本结构

```
SELECT [DISTINCT|ALL]{*|属性列表达式[AS 新的属性列名][,…] }
FROM <基表名|视图名>[别名][,…]
[INTO <新表名>]
[WHERE <行选择条件>]
[GROUP BY <分组依据列>]
[HAVING <分组选择条件>]
[ORDER BY <排序依据列>[ASC|DESC]];
```

- 查询所有列
  - 例：查询医生的所有信息

```
SELECT * FROM Doctor
```

  - 等价于：

```
SELECT Dno,Dname,Dsex,Dage,Ddeptno,Dlevel,Dsalary FROM Doctor
```

	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel	Dsalary
1	140	郝亦伟	男	28	102	医师	1800.00
2	21	刘伟	男	43	103	副主任医师	2800.00
3	368	罗晓	女	27	102	主治医师	2000.00
4	73	邓英超	女	43	201	主任医师	3200.00
5	82	杨勋	男	36	101	副主任医师	2800.00

- 查询指定列
  - 例：查询所有医生的姓名、编号及所在科室

```
SELECT Dname,Dno ,Ddeptno FROM Doctor
```

	Dname	Dno	Ddeptno
1	郝亦伟	140	102
2	刘伟	21	103
3	罗晓	368	102
4	邓英超	73	201
5	杨勋	82	101

- 查询经过计算的列
  - 例：查询医生的姓名和出生年份

```
SELECT Dname,2023-Dage FROM Doctor
```

	Dname	（无列名）
1	郝亦柯	1982
2	刘伟	1967
3	罗晓	1983
4	邓英超	1967
5	杨勋	1974

  - 例：查询医生的姓名及年薪

```
SELECT Dname,Dsalary*12 FROM Doctor
```

	Dname	(无列名)
1	郝亦柯	24000.00
2	王军	39600.00
3	刘伟	48000.00
4	罗晓	42000.00
5	邓英超	60000.00
6	杨勋	52800.00

  - 列别名

```
SELECT Dname,Dsalary*12 年薪 FROM Doctor
```

- 查询不重复的元组 DISTINCT
  - 在Doctor表中查询医院的所有科室编号，要求不重复显示

```
SELECT DISTINCT Ddeptno FROM Doctor
```

	Ddeptno
1	101
2	102
3	103
4	201

- 选择满足条件的元组
  - 例：查询医生基本信息表中年龄在 40 岁以上医生的姓名和专业职称

```
SELECT Dname 医生姓名,Dlevel 专业职称
FROM Doctor
WHERE  Dage>40
```

	医生姓名	专业职称
1	刘伟	副主任医师
2	邓英超	主任医师

- WHERE子句中的运算符
  - 比较运算符

    = 相等　　！= 不等于
    < >不等于　　< = 小于等于
    < 小于　　> = 大于等于
    > 大于
  - 逻辑运算符

    AND　　OR　　NOT
  - 集合运算符

    IN　　NOT IN
  - 确定范围运算符

    BETWEEN…AND　　NOT BETWEEN…AND
  - 字符串比较运算符

    _（下画线）：匹配任意一个字符。
    %（百分号）：匹配任意长度的字符
    []:查询一定范围内的数据
    escape<换码字符>：转义符，本身含有%或_

- 比较运算
  - 例：查询年龄在40岁以下的医生信息

```
SELECT * FROM Doctor
WHERE Dage<40
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	82	杨勋	男	36	104	35
2	140	郝亦柯	男	28	101	1
3	368	罗晓	女	27	103	4

- 逻辑运算AND、OR和NOT
  - 例：查询年龄在40岁以下的男医生信息

```
SELECT * FROM Doctor
WHERE Dsex=‘男’ AND Dage<40
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	82	杨勋	男	36	104	35
2	140	郝亦柯	男	28	101	1

- 确定集合运算
  - 例：查询部门编号为102，103和201的医生信息。

```
SELECT * FROM Doctor
WHERE Ddeptno IN ('102','103','201')
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	368	罗晓	女	27	103	4

  - 例：查询既不是102科室，也不是201科室的医生信息

```
SELECT * FROM Doctor
WHERE Ddeptno NOT IN ('102','201')
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	21	刘伟	男	43	104	1
2	73	邓英超	女	43	105	33
3	82	杨勋	男	36	104	35
4	140	郝亦柯	男	28	101	1
5	368	罗晓	女	27	103	4

- 确定范围
  - 例：查询年龄在35-40岁之间的医生信息

```
SELECT * FROM Doctor
WHERE Dage BETWEEN 35 AND 40
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	82	杨勋	男	36	104	35

- 字符串比较
  - 例：查询药品名中含有“葡萄糖”的药品信息。

```
SELECT * FROM Medicine
WHERE Dname LIKE  '%葡萄糖%'
```

	Mno	GSno	Mname	Mprice	Munit	Mtype
1	314418	NULL	普硝唑葡萄糖针	12	瓶	西药

  - 例：查询姓名中第二个字为“英” 的医生，返回其姓名和编号。

```
SELECT Dname,Dno
FROM Doctor
WHERE Dname LIKE  '_英%'
```

	Dname	Dno
1	邓英超	73

- 测试是否为NULL
  - 空（NULL）在数据库里有特殊的含义，它表示未定义或不确定的值。用运算符 IS 或 IS NOT 来测试是否为NULL值。
  - 例：查询还没有分配部门的医生

```
SELECT * FROM Doctor
where Ddeptno IS NULL
```

  - 例：查询已分配部门的医生

```
SELECT * FROM Doctor
where Ddeptno IS NOT NULL
```

- ORDER BY 子句

```
ORDER BY 表达式1 [ASC | DESC]  [,表达式2  [ASC | DESC] [, … n]]
```

- 例：在医生基本信息表中，按部门编号升序及年龄降序查询医生信息

```
SELECT *
FROM Doctor
ORDER BY Ddeptno ASC,Dage DESC
```

	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	82	杨勋	男	36	101	副主任医师
2	140	郝亦柯	男	28	102	医师
3	368	罗晓	女	27	102	主治医师
4	21	刘伟	男	43	103	副主任医师
5	73	邓英超	女	43	201	主任医师

- TOP限制结果集
  - 语法格式

```
SELECT TOP n [PERCENT] [WITH TIES] 列名1 [,列名2,…列名n]
FROM 表名
```

  - TOP谓词放在SELECT子句的后面（如果有DISTINCT，则在DISTINCT之后），查询列表的前面。
  - 使用TOP谓词时通常会与ORDER BY 子句一起使用，以表达前几名的含义。当使用WITH TIES（包括并列值）时，要求必须使用ORDER BY。

- TOP子句实例
  - 例：查询医院年龄最大的三个医生姓名，年龄

```
SELECT TOP 3 Dname,Dage
FROM Doctor
ORDER BY Dage DESC
```

	Dname	Dage
1	刘伟	43
2	邓英超	43
3	杨勋	36

- TOP子句实例
  - 查询医院年龄最大的医生姓名，年龄：

```
SELECT TOP 1 WITH TIES Dname,Dage
FROM Doctor
ORDER BY Dage DESC
```

	Dname	Dage
1	刘伟	43
2	邓英超	43

```
SELECT TOP 1 Dname,Dage
FROM Doctor
ORDER BY Dage DESC
```

	Dname	Dage
1	刘伟	43

- 统计查询
  - 聚集函数实现对数据集的统计查询，包括求和、求平均值、最大值、最小值等
  - 聚集函数一般要忽略NULL值，不对NULL值进行操作
  - 除MIN，MAX，COUNT3个函数适合于任何数据类型外，其余的聚集函数一般都要求是数值型。

- 使用格式：

〈聚集函数名〉(DISTINCT|ALL 表达式)

函数名称	函数功能
SUM	返回选取结果集合中所有值的总和
COUNT	返回选取的结果集合中所有记录行的数目
MAX	返回选取结果集中所有值的最大值
MIN	返回选取结果集中所有值的最小值
AVG	返回选取结果集中所有值的平均值

- MAX，MIN，SUM,AVG等函数的用法
  - 例1：查询Medicine表中Mprice字段的最大值和最小值

```
SELECT MAX(Mprice), MIN(Mprice)
 FROM Medicine
```

	(无列名)	(无列名)
1	46.00	0.13

  - 例2：统计医生的平均年龄

```
SELECT AVG(Dage)
 FROM Doctor
```

	(无列名)
1	34

  - 例3：统计就诊表中就诊费用的总额

```
SELECT SUM(Rfee)
 FROM Diagnosis
```

	(无列名)
1	31

- COUNT函数的用法
  - COUNT函数用来计算表中记录的个数或者列中值的个数，计算内容由SELECT语句指定。
  - COUNT（*），计算表中行的总数，即使表中行的数据为NULL，也被计入在内。
  - COUNT（column），计算column列包含的行的数目，如果该列中某行数据为null，则该行不计入统计总数。
  - 例：统计医生总数和有医生的部门数量

```
SELECT  COUNT(*), COUNT(Ddeptno)
FROM Doctor
```

	(无列名)	(无列名)
1	6	6

- 分组查询GROUP BY
  - GROUP BY子句主要是对数据表分组后进行统计查询
  - GROUP BY子句必需和聚集函数一起使用
  - 例：按部门编号统计各部门的医生数量

```
SELECT Ddeptno 部门编号,count(Dno)人数
FROM Doctor
GROUP BY Ddeptno
```

	部门编号	人数
1	101	1
2	102	2
3	103	1
4	201	1

- 分组条件子句HAVING
  - 当需要选出符合条件的分组统计值时，在GROUP BY子句之后，可使用HAVING子句实现
  - HAVING子句通常和GROUP BY子句一起使用。如果不使用GROUP BY子句，则HAVING子句的作用与WHERE子句一样
  - 例：按部门统计男医生的平均年龄不超过40岁的部门编号

```
SELECT Ddeptno 部门编号,AVG(Dage) 平均年龄
FROM Doctor
WHERE Dsex='男'
GROUP By Ddeptno
HAVING AVG(Dage)<=40
```

	部门编号	平均年龄
1	102	28
2	101	36

- HAVING子句与WHERE子句对比

  - HAVING子句和WHERE子句的相似之处在于，它也定义搜索条件。但与WHERE子句不同的是，HAVING子句是对分组记录进行筛选，而WHERE子句与单行的记录行有关。

  - 如果查询语句中同时出现WHERE子句GROUP BY子句，则先执行WHERE子句，然后执行HAVING子句。

- 集合运算
  - UNION运算符实现集合并运算

```
查询语句1
UNION  [ALL]
查询语句2
```

  - EXCEPT运算符实现集合差运算

```
查询语句1
EXCEPT
查询语句2
```

  - INTERSECT运算符实现集合交运算

```
查询语句1
INTERSECT
查询语句2
```

- UNION运算

- 例：在医院数据库中，为了提高系统处理效率，需要定期对患者的诊断信息归档，假定患者诊断归档信息表为HisDiagnosis，当前诊断表是Diagnosis。医生要查询患者“刘景”的近期和历史诊断信息，以便分析患者的病因。

```
SELECT DGno 诊断号,Dname 医生姓名,Symptom 症状,Diagnosis 诊断,DGtime 时间
FROM Diagnosis Diag,Doctor Doc,Patient P
WHERE DiagB.Dno=Doc.Dno AND P.Pno=DiagB.Pno AND P.Pname='刘景'
UNION
SELECT DGno,Dname,Symptom,Diagnosis,DGtime
FROM HisDiagnosis Diag,Doctor Doc,Patient P
WHERE Diag.Dno=Doc.Dno AND P.Pno=Diag.Pno AND P.Pname='刘景'
```

	诊断号	医生姓名	症状	诊断	时间
1	3265	杨勋	胃溃疡	螺杆菌感染	2007-07-23 10:59:42.000

- INTERSECT运算
- 例：在医院数据库中，查找同时拿到就诊单和处方单的患者。

```
SELECT Pname
FROM Patient
WHERE Pno IN
     ( SELECT Pno
       FROM Diagnosis
       INTERSECT
       SELECT Pno
       FROM RecipeMaster
      );
```

	Pname
1	刘景
2	陈禄
3	曾华
4	傅伟相
5	张珍
6	李秀

```
SELECT Pname
FROM Patient
WHERE Pno IN
     ( SELECT Pno
       FROM Diagnosis
      )
      AND Pno IN
      ( SELECT Pno
        FROM RecipeMaster
      );
```

- EXCEPT运算
- 例：在医院数据库中，查找只拿到处方单没有拿到就诊单的患者，返回患者编号。

```
SELECT Pno
FROM RecipeMaster
EXCEPT
SELECT Pno
FROM Diagnosis
```

	Pno
1	481

```
SELECT Pno
FROM Patient
WHERE Pno in
    ( SELECT Pno
      FROM RecipeMaster
    )
    AND Pno NOT IN
   ( SELECT Pno
     FROM Diagnosis
   );
```

- 查询中集合运算的使用原则
- 集合运算符两端的查询结果具有相同的模式
- ORDER BY子句只能在语句的结尾处使用，不能在构成语句的各个查询中使用
- GROUP BY和HAVING子句只能在各个查询中使用，他们不能用于影响最终结果集

连接查询

- 为了查询分散在多个表中的数据，则可以连接多个表之后再进行查询
- 同时涉及多个表的查询，称为连接查询。用来连接两个表的条件称为连接条件或连接谓词。
- 连接可以两个表及多个表进行操作，也可以对同一个表操作。连接主要包括内连接，外连接以及自连接。

- 内连接是最常用的连接类型，也是在不明确指明连接类型的情况下默认的连接类型。
- 内连接可以通过在WHERE子句中指定连接条件来完成 （SQL89）
- 内连接也可以通过JOIN子句完成（SQL92）

- 在WHERE子句中的内连接
  - 连接格式：FROM 表1,表2 WHERE <连接条件>
  - 连接条件指定两个表按照什么条件进行连接

    [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>。
  - 例：查询开出处方的医生信息

```
SELECT Rno,Pno,D.Dno,Dname,Dsex,Dage,Ddeptno,Dlevel
FROM RecipeMaster R,Doctor D
WHERE R.Dno = D.Dno
```

	Rno	Pno	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	1282317	481	140	郝亦柯	男	28	102	医生
2	1282872	201	21	刘伟	男	43	103	副主任医师
3	1283998	161	82	杨勋	男	36	101	副主任医师
4	1284041	181	82	杨勋	男	36	101	副主任医师
5	1284256	501	73	邓英超	女	43	201	主任医师

- INNER JOIN ON 运算符
  - 格式:FROM 表1 [INNER] JOIN 表2 ON <连接条件>
  - 连接条件同WHERE子句的连接条件相同，可以是等值连接或者是非等值连接。
  - 例：查询开出处方的医生信息

```
SELECT Rno, Pno, D.Dno, Dname, Dsex, Dage, Ddeptno, Dlevel
 FROM RecipeMaster R JOIN Doctor D ON R.Dno = D.Dno
```

	Rno	Pno	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	1282317	481	140	郝亦柯	男	28	102	医生
2	1282872	201	21	刘伟	男	43	103	副主任医师
3	1283998	161	82	杨勋	男	36	101	副主任医师
4	1284041	181	82	杨勋	男	36	101	副主任医师
5	1284256	501	73	邓英超	女	43	201	主任医师

- INNER JOIN USING运算符

  - 格式:FROM 表1 [INNER] JOIN 表2 USING(列名)
  - 当使用USING语句来指明连接条件时，只有连接的两个表中名字相同的列的列名才可以作为USING语句的连接列名
  - 例：查询开出处方的医生信息

```
SELECT Rno, Pno, D.Dno, Dname,  Dsex, Dage, Ddeptno, Dlevel
FROM RecipeMaster R JOIN Doctor D USING(Dno)
```

	Rno	Pno	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	1282317	481	140	郝亦柯	男	28	102	医生
2	1282872	201	21	刘伟	男	43	103	副主任医师
3	1283998	161	82	杨勋	男	36	101	副主任医师
4	1284041	181	82	杨勋	男	36	101	副主任医师
5	1284256	501	73	邓英超	女	43	201	主任医师

- 自然连接
  - 格式:FROM 表1 NATURAL JOIN 表2
  - 自然连接是一种特殊的等值连接
  - 自然连接要求两个关系表中进行比较的必须是相同的属性列，无须添加ON或者USING子句，并且在结果中消除重复的属性列。
  - 例：查询开出处方的医生信息

```
SELECT Rno, Pno, D.Dno, Dname, Dsex, Dage, Ddeptno, Dlevel
FROM RecipeMaster R NATURAL JOIN Doctor D
```

	Rno	Pno	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
1	1282317	481	140	郝亦柯	男	28	102	医生
2	1282872	201	21	刘伟	男	43	103	副主任医师
3	1283998	161	82	杨勋	男	36	101	副主任医师
4	1284041	181	82	杨勋	男	36	101	副主任医师
5	1284256	501	73	邓英超	女	43	201	主任医师

- 自连接
  - 自连接是一种特殊的内连接。它指的是相互连接的表在物理上为同一张表，但是可以在逻辑上分为两张表。
  - 使用自连接时必须为两个表取别名，使之逻辑上成为两张表。可以把自连接理解为同一张表的两个副本之间的连接，使用不同别名来区别副本，处理过程与不同表之间的连接相同。
  - 例：在医院部门表中，需要医院的各部门名称和上级部门名称

```
SELECT First.DeptName 部门名称,Second.DeptName 上级部
FROM  Dept First ,Dept Second
WHERE First.ParentDeptNo=Second.DeptNo
```

	部门名称	上级部门
1	门诊部	xx医院
2	社区医疗部	xx医院
3	消化内科	门诊部
4	急诊内科	门诊部
5	门内三诊室	门诊部
6	家庭病床病区	社区医疗部

- 外连接
  - 外连接分为左外连接，右外连接和全外连接三种。
  - 左外连接（LEFT OUTER JOIN）:输出左表的所有记录相关列值，右表输出与左表匹配的记录，如果没有与左表匹配的记录，则使用NULL匹配输出。
    FROM 表1 LEFT OUTER JOIN 表2 ON <连接条件>
  - 右外连接（RIGHT OUTER OUT）:输出右表的所有记录相关列值，左表输出与右表匹配的记录，如果没有与右表匹配的记录，则使用NULL匹配输出。
    FROM 表1 RIGHT OUTER JOIN 表2 ON <连接条件>
  - 全外连接(FULL OUTER JOIN):为左外连接和右外连接的并集，返回左表跟右表中的所有行。
    FROM 表1 FULL OUTER JOIN 表2 ON <连接条件>

- 左外连接
  - 例：查询医院的各部门名称和该部门医生姓名

```
SELECT DeptName 部门名称,Dname 医生姓名
 FROM Dept LEFT OUTER JOIN Doctor ON Dept.DdeptNo=Doctor.Ddeptno
```

	部门名称	医生姓名
1	XX医院	NULL
2	门诊部	NULL
3	社区医疗部	NULL
4	消化内科	郝亦柯
5	急诊内科	NULL
6	门内三诊室	罗晓
7	家庭病床病区	NULL

- 右外连接

- 例：查询每种药品名对应的处方编号。

```
SELECT Rno 处方编号,Mname 药物名称
 FROM RecipeDetail RIGHT OUTER JOIN medicine ON RecipeDetail.Mno = Medicine.Mno
```

	处方编号	药物名称
1	NULL	卡托普利片
2	NULL	普萘唑葡萄糖针
3	16	骨石通颗粒
4	32	心胃止痛胶囊
5	NULL	阿奇霉素胶囊
6	NULL	L-谷氨酰胺胶囊
7	NULL	盐酸雷尼替丁胶囊
8	47	胃立康片
9	NULL	复方雷尼替丁胶囊
10	89	依诺沙星注射液
11	NULL	蒲公英胶囊

课堂练习

◆查询每一门课的间接先修课（即先修课的先修课）

```
SELECT  FIRST.Cno, SECOND.Cpno
FROM  Course  FIRST, Course  SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

FIRST

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学	<NULL>	2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	<NULL>	2
7	PA_CAL语言	6	4

SECOND

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学	<NULL>	2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	<NULL>	2
7	PA_CAL语言	6	4

间接先修课

Cno	Cpno
1	7
3	5
4	NULL
5	6
7	NULL

- 嵌套查询基本概念
  - 例1：查询‘102’部门中，工资高于4000元的医生，返回其姓名、工资和所在部门编号。

```
SELECT Dname 姓名,Dsalary 工资, DeptNo 部门编号
FROM Doctor
WHERE DeptNo= ‘102’ and DSalary>4000;
```

  - 例2：查询‘102’部门中，工资高于该部门平均工资的医生，返回其姓名、工资和所在部门编号。

```
SELECT Dname 姓名,Dsalary 工资, DeptNo 部门编号
FROM Doctor
WHERE DeptNo= ‘102’ and DSalary>
    ( SELECT AVG(Dsalary)
      FROM Doctor
      WHERE DeptNo= ‘102’ );
```

- 嵌套查询基本概念
  - 在查询语句的条件子句部分含有SELECT查询语句；
  - 外层的查询被称为主查询（或父查询）；
  - 内层的SELECT查询子句被称为子查询；
  - 子查询的结果作为主查询条件的一部分；
  - 嵌套查询可分为不相关子查询和相关子查询。

- 嵌套查询语法格式

```
SELECT <查询列表>
[ INTO <新表名> ]
FROM <基表名|视图名> [ 别名 ] ……
WHERE <列名或列表达式> <比较运算符>
( SELECT <查询列>
FROM <基表名|视图名> [ 别名 ] ……
WHERE <条件表达式>
[ GROUP BY <分组内容>]
[ HAVING <组内条件>] )
[ GROUP BY <分组内容>]
[ HAVING <组内条件>]
[ ORDER BY <排序列名>[ ASC | DESC ])
```

- 不相关子查询
  - 子查询的查询条件不依赖于主查询
  - 执行顺序从嵌套层次最内层子查询开始执行，每个子查询在其直接外查询处理之前执行，查询返回结果作为父查询的查询条件，最后执行最外层的主查询
  - 例：查询‘102’部门中，工资高于该部门平均工资的医生，返回其姓名、工资和所在部门编号。

```
SELECT Dname 姓名,Dsalary 工资, DeptNo 部门编号
 FROM Doctor
 WHERE DeptNo= ‘102’ and  DSalary>
      ( SELECT AVG(Dsalary)
        FROM Doctor
        WHERE DeptNo= ‘102’ );
```

	姓名	工资	部门编号
1	朱自鸣	5000.00	102
2	李陵	5000.00	102

- 相关子查询
  - 依赖于主查询的子查询，即子查询的条件子句中含有主查询中表的相关信息。
  - 先执行外层查询，然后再执行内层查询。由外层查询的值决定内层查询的结果，内层查询的执行次数由外层查询的结果决定。
  - 例：查询各部门中，工资高于该部门平均工资的医生，返回其姓名、工资和所在部门编号。

```
SELECT Dname 姓名,Dsalary 工资, DeptNo 部门编号
FROM Doctor D1
WHERE DSalary>
    ( SELECT AVG(Dsalary)
      FROM Doctor D2
      WHERE D1.DeptNo=D2.DeptNo);
```

	姓名	工资	部门编号
1	王军	3300.00	101
2	杨勋	4400.00	101
3	李陵	5000.00	102
4	朱自鸣	5000.00	102
5	王维国	5000.00	103
6	邓英超	5000.00	201

- 运算符IN和NOT IN
  - IN表示某元素值是属于集合，而NOT IN则表示元素不属于集合
  - 例：查询开过药品“肾石通颗粒”的医生信息

```
SELECT * FROM Doctor
 WHERE Dno IN
      (SELECT Dno from RecipeMaster
       WHERE Rno IN
            (SELECT Rno fromRecipeDetail
             WHERE Mno IN
                  (SELECT Mno FROM Medicine
                   WHERE Mname= ‘肾石通颗粒’
                   )
             )
      );
```

	Dno	Dname	Dsex	Dage	Ddeptno	Tno
1	368	罗晓	女	27	103	4

- 运算符EXISTS和NOT EXISTS
  - 如果EXISTS运算符限定的子查询有查询记录返回，那么该条件为真，否则为假。
  - NOT EXISTS运算符限定的子查询返回的记录集为空，那么该条件为真，否则为假。
  - 例：查询给姓名为“刘景”的患者开过处方的医生：

```
SELECT Dno 医生编号,Dname 姓名,Dsex 性别,Dage 年龄,Dlevel 职称
 FROM Doctor
 WHERE EXISTS
      ( SELECT *
        FROM RecipeMaster
        WHERE RecipeMaster.Dno=Doctor.Dno AND EXISTS
          ( SELECT *
            FROM Patient
            WHERE Patient.Pname=’刘景’ AND
Patient.Pno=RecipeMaster.Pno
          )
      );
```

	医生编号	姓名	性别	年龄	职称编号
1	368	罗晓	女	27	4

- 运算符ANY和ALL
  - ANY运算符是检查在子查询结果集中是否满足给定的条件。如果子查询的结果集中至少有一个值满足条件，则比较运算结果为真，否则为假。
  - ALL运算符是检查在子查询结果集中所有值是否都满足给定的条件。只有当结果集的所有值均满足给定的条件，则比较运算结果为真，否则为假。
  - 在ANY和ALL一般需要=，!=，>，<，<=或>=等比较运算符配合使用。

- 运算符ANY和ALL
  - 例：查询比至少一位女医生年龄大的男医生姓名和年龄

```
SELECT Dname 姓名,Dage 年龄
FROM Doctor
WHERE Dsex='男' AND Dage >ANY
     ( SELECT Dage
       FROM Doctor
       WHERE Dsex= '女'
      );
```

	姓名	年龄
1	刘伟	43
2	杨勋	36
3	郝亦柯	28

```
  SELECT Dname 姓名,Dage 年龄
FROM Doctor
WHERE Dsex='男' AND Dage >
     ( SELECT MIN(Dage)
       FROM Doctor
       WHERE Dsex= '女'
      );
```

- 数据插入（insert）
  - 插入元组

    INSERT INTO <基表名>[(<属性列表>)] VALUES (<属性值列表>)

  - 插入子查询的结果

    INSERT INTO <基表名>[(<属性列表>)]
    子查询

- 例：在医院数据库中，需要向医生信息表中插入('145','王军','男',28,'101','医师')记录

```
INSERT INTO Doctor(Dno,Dname,Dsex,Dage,Ddeptno,Dlevel)
 VALUES('145','王军','男',28,'101,'医师')
```

```
INSERT NTO Doctor
VALUES('145','王军','男',28,'101','医师')
```

WIN-HJUCLFAU7JD.HIS - dbo.Doctor | SQLQuery1.sql - WI...7JD.HIS (sa (55))*

Dno	Dname	Dsex	Dage	Ddeptno	Dlevel
140	郝亦柯	男	28	102	医师
145	王军	男	28	101	医师
21	刘伟	男	43	103	副主任医师
368	罗晓	女	27	102	主治医师
73	邓英超	女	43	201	主任医师
82	杨勋	男	36	101	副主任医师
NULL	NULL	NULL	NULL	NULL	NULL

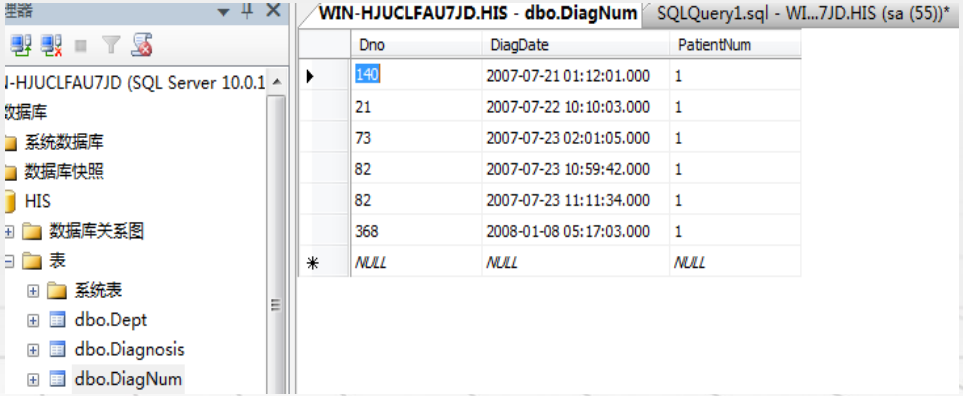
- 例：在医院数据库中，统计每个医生每天诊断的患者数量，并把结果存入数据库。

首先要建立一个新表DiagNum，包含医生编码、诊断日期和患者数量：

```
CREATE TABLE DiagNum
 (Dno VARCHAR(10) NOT NULL,
  DiagDate DATETIME,
  PatientNum INT)
```

然后插入：

```
INSERT INTO DiagNum (Dno, DiagDate, PatientNum)
SELECT Dno,Rdatatime,COUNT(DGno)
FROM RecipeMaster
GROUP BY Dno,Rdatatime
```



- 数据修改
  - 修改语句语法

UPDATE <基表名>
SET <属性列名>=<表达式>[,<属性列名>=<表达式>,….]
[WHERE <行选择条件>]

  - 例：在医院数据库中，将编号为“421”的患者的社会保障号修改为“20073425”

UPDATE Patient SET Pino='20073425'
WHERE Pno='421'

WIN-HJUCLFAU7JD.HIS - dbo.Patient | SQLQuery1.sql - WI...7JD.HIS (sa (55))*

Pno	Pname	Psex	Page	Pino	Pid
161	刘景	男	66	120167699	6781121941030...
181	陈禄	男	69	120400180	5461021938081...
201	曾华	女	75	080092007	1231111932090...
421	傅伟相	男	60	070023515	4901021947051...
481	张珍	女	54	120043267	3451121953120...
501	李秀	女	71	069201544	3341111936041...
NULL	NULL	NULL	NULL	NULL	NULL

WIN-HJUCLFAU7JD.HIS - dbo.Patient | SQLQuery1.sql - WI...7JD.HIS (sa (55))*

Pno	Pname	Psex	Page	Pino	Pid
161	刘景	男	66	120167699	6781121941030...
181	陈禄	男	69	120400180	5461021938081...
201	曾华	女	75	080092007	1231111932090...
421	傅伟相	男	60	20073425	4901021947051...
481	张珍	女	54	120043267	3451121953120...
501	李秀	女	71	069201544	3341111936041...
NULL	NULL	NULL	NULL	NULL	NULL

- 带子查询的修改语句
  - 例：将消化内科所有医生的工资上浮10%

```
UPDATE Doctor
SET Dsalary = Dsalary+Dsalary*0.1
WHERE Ddeptno IN
      (SELECT Ddeptno
       FROM Dept
       WHERE DeptName='消化内科')
```

结果 消息

	Dno	Dname	Dsex	Dage	Ddeptno	Dlevel	Dsalary
1	140	郝亦柯	男	28	102	医生	2000.00
2	145	王军	男	28	101	医师	3000.00
3	21	刘伟	男	43	103	副主任医师	4000.00
4	368	罗晓	女	27	102	主治医师	3500.00
5	73	邓英超	女	43	201	主任医师	5000.00
6	82	杨勋	男	36	101	副主任医师	4000.00

- 数据删除的语法
  - 删除语句格式

```
DELETE FROM <基表名>
[WHERE<行选择条件>]
```

  - 例1：在医院数据库中，将编号为“421”的患者从系统中删除

```
DELETE FROM Patient WHERE Pno='421'
```

  - 例2：在医院数据库中，将姓名为“刘景”的患者收费记录从系统中删除

```
DELETE FROM Fee
WHERE Pno in (
      SELECT Pno
      FROM Patient
      WHERE Pname='刘景')
```

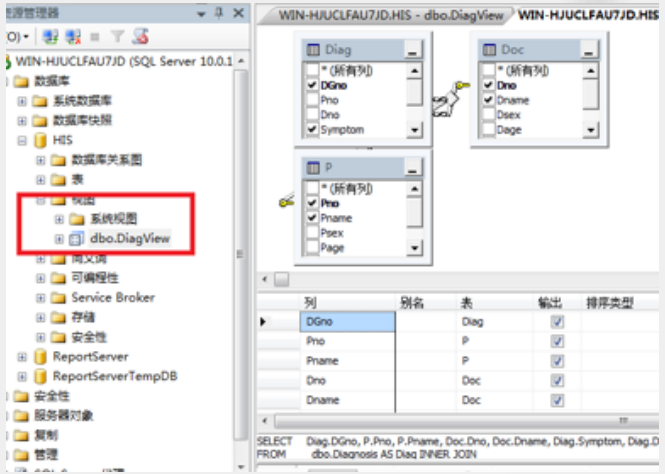
- 视图的基本概念
  - 视图是一个虚表，即视图所对应的数据不进行实际存储
  - 数据库中只存储视图的定义，在对视图的数据进行操作时，数据库系统根据视图的定义去操作与视图相关联的基本表

- 视图的优点
  - 视图能简化用户的操作
  - 提高数据的安全性
  - 保证数据的完整性

- 定义视图的语法

```
CREATE VIEW <视图名> [(视图列表)]
AS <子查询>
[ WITH CHECK OPTION ]
```

- 例：为消化内科诊断的患者信息建立一个视图




```
CREATE VIEW DiagView
AS
SELECT DGno,P.Pno,Pname,Doc.Dno,Symptom,Diagnosis,DGtime
FROM Diagnosis Diag,Doctor Doc,Patient P
WHERE Diag.Dno=Doc.Dno
AND P.Pno=Diag.Pno
AND Doc.Ddeptno IN(SELECT Ddeptno FROM Dept WHERE DeptName='消化内科')
```

- 创建视图示例：
  - 为消化内科医生信息建立一个视图。

```
CREATE VIEW DocView_1
AS

SELECT Dno,Dname,Dsex,Dage,Dlevel,Dsalary
 FROM Doctor
 WHERE Ddeptno= '101'
 with check option
```

- 修改操作：DBMS自动加上Ddeptno= ‘101'的条件
- 删除操作：DBMS自动加上Ddeptno= ‘101'的条件
- 插入操作：DBMS自动检查Ddeptno属性值是否为 ‘101'
  - 如果不是，则拒绝该插入操作
  - 如果没有提供Ddeptno属性值，则自动定义Ddeptno为‘101'

- 查询视图
  - 例：查询视图DiagView中的记录

```
SELECT * FROM DiagView
```

```
select * from DiagView
```

100 %

结果 消息

	DGno	Pno	Pname	Dno	Symptom	Diagnosis	DGtime
1	3265	161	刘景	82	胃溃疡	螺杆菌感染	2007-07-23 10:59:42.000
2	3308	181	陈禄	82	消化不良	胃病	2007-07-23 11:11:34.000

更新视图原则

- 由于视图是不存储数据的虚表，数据是来自其他基表部分数据，因此，对视图的更新最终是对基表的更新。
- SQL语言标准规定：只能对直接定义在一个基表上的视图进行插入、修改、删除等更新操作，对定义在多个基表或其它视图之上的视图，数据库管理系统不允许进行更新操作。
- 例如：在医院数据库中，创建了医生为患者的诊断信息视图。该视图为不可修改视图。

```
CREATE VIEW DiagView
AS
SELECT DGno,P.Pno,Pname,Doc.Dno,Dname,
        Symptom,Diagnosis,DiagDateTime
FROM Diagnosis Diag , Doctor Doc, Patient P
WHERE Diag.Dno=Doc.Dno AND P.Pno=Diag.Pno
```

更新视图

- 尽管视图数据只来源于一个基表，如果SELECT语句含有GROUP BY、DISTINCT或聚集函数等，除可执行删除操作外，不能进行插入或修改操作。
- 例如：在医院数据库中，如果需要统计每位医生每天诊断工作量，建立如下视图，该视图除可执行删除操作外，不能进行插入或修改操作。

```
CREATE VIEW DiagNum (Dno, DiagDate, PatientNum)
AS
SELECT Dno,Rdatetime,COUNT(DGno)
FROM RecipeMaster
GROUP BY Dno,Rdatetime
```

更新视图

- 如果视图中包含由表达式计算的列，则不允许进行更新操作。
- 例如：在药品信息表中，假如为药品单价提高15%后建立的药品价格视图，则不能修改该视图中的药品单价。

```
CREATE VIEW MedicineNewPrice(Mno,Mname ,Newprice ,Munit,Mtype)
AS
   SELECT Mno,Mname ,Mprice*1.15,Munit,Mtype
   FROM Medicine
   WHERE Mprice*1.15>=30
```

更新视图

- 尽管视图满足上述3个条件，如果该视图中没有包含基表的所有NOT NULL列，则不能对该视图进行插入操作。
- 原因是，对视图的插入实际是对基表的插入操作，当视图没有包含基表的所有NOT NULL列时，在向视图进行插入时，系统默认为NULL，这与定义中的NOT NULLL相矛盾，因此系统就会拒绝插入并给出错误提示。

- 更新视图的方法
- 例：在医院数据库中，创建了男医生视图，该视图为可更新视图

```
CREATE VIEW MaleDoctor
AS
SELECT *
FROM Doctor Doc
WHERE Dsex= ‘男’
WITH CHECK OPTION
```

更新视图，104部门的男医生年龄增加一岁

```
UPDATE MaleDoctor
SET Dage=Dage+1
WHERE Ddeptno= ‘104’
```

等价于

```
UPDATE Doctor
SET Dage=Dage+1
WHERE Ddeptno= ‘104’ AND Dsex= ‘男’
```

- 删除视图的语法

DROP VIEW 要删除的视图名

- 例：删除视图

```
DROP VIEW DiagView

DROP VIEW MaleDoctor
```

完整性约束概述

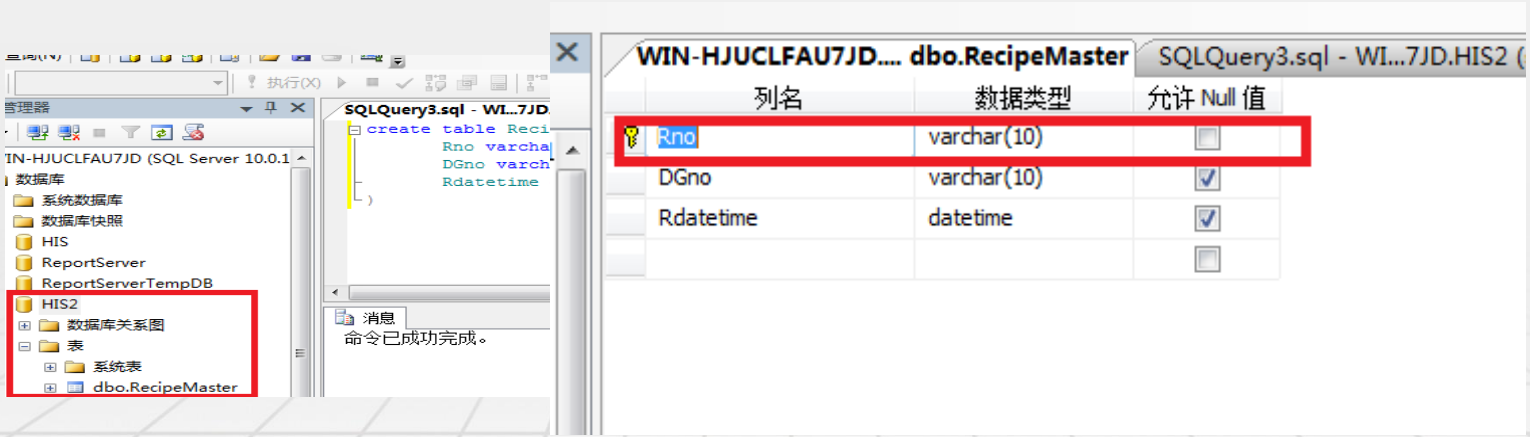
- 数据库系统是对现实世界的真实反映，用户在进行数据库访问的过程中，有很多原因可能导致更新数据出现错误，因此保护存储数据的一致性和正确性很有必要。
- 完整性约束是加在数据库模式上的一个具体条件，它规定什么样的数据能够存储到数据库系统当中。
- DBA或用户定义完数据模式后，就指明在数据库中的所有模式应满足的完整性约束条件。

- 完整性约束的分类
  - 按照完整性约束条件作用的对象分：
    - 类型约束
    - 属性约束
    - 关系变量约束
    - 数据库约束
  - 按照完整性约束条件声明时的位置分：
    - 列级约束
    - 表级约束

- 主键约束
  - 一个基本表只能有一个主键。在使用CREATE TABLE语句定义基本表时，可以有两种方法定义主键：
    - 在一个属性的类型定义完毕后，直接在后面加上PRIMARY KEY。
    - 在所有属性定义完毕后，增加一个PRIMARY KEY的声明，指出主键包含哪些属性。
  - PRIMARY KEY子句中的每个属性的取值都必须是NOT NULL

- 例：Primary Key约束

```
CREATE TABLE RecipeMaster{
Rno VARCHAR(10) PRIMARY KEY,
DGno VARCHAR (10) ,
Rdatetime  DATETIME
}
```



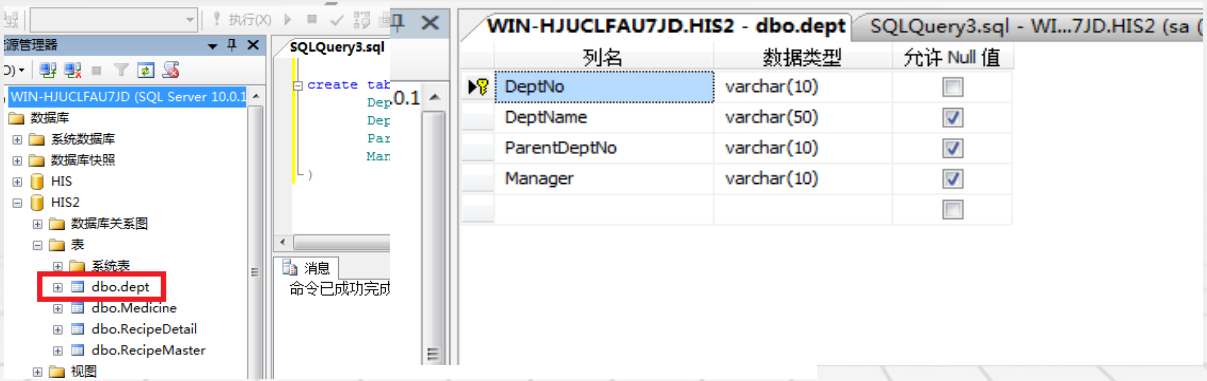
- 例：表级约束的定义
  - PRIMARY KEY表级约束

```
CREATE TABLE RecipeDetail{
Rno VARCHAR(10),
Mno VARCHAR(10) NOT NULL,
Mamount DECIMAL(18,0),
PRIMARY KEY(Rno,Mno)
}
```

- UNIQUE约束
  - UNIQUE约束用于指明某一列或多个列的组合上的取值必须唯一
  - 在一个关系中，PRIMARY KEY只有一个，而UNIQUE可以声明多个
  - PRIMARY KEY要求属性取值不能为NULL，而UNIQUE允许属性取空值，允许多个空值同时存在

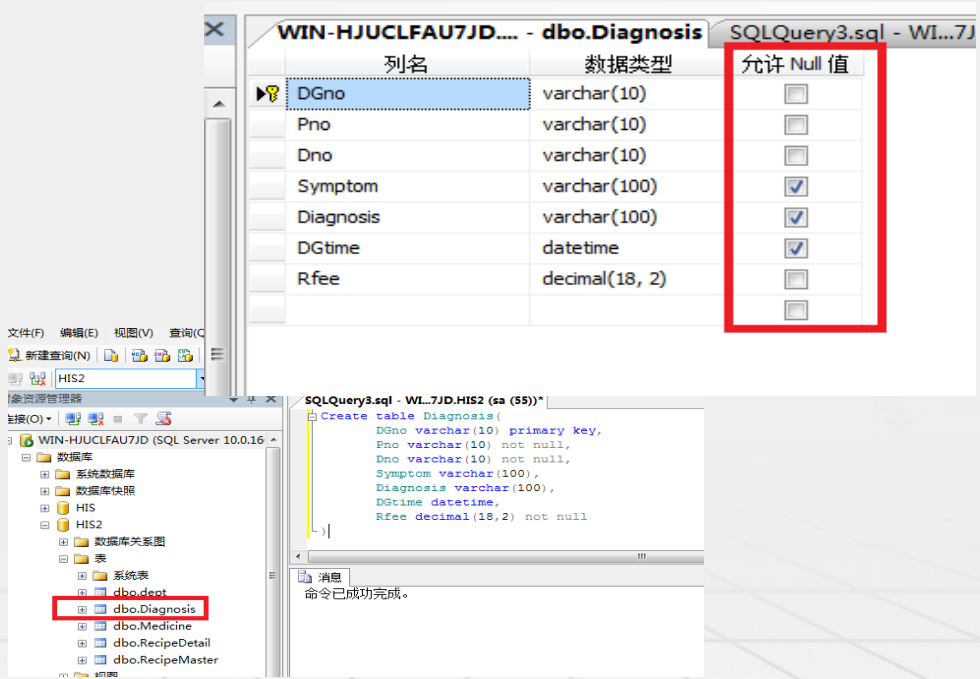
- 例：UNIQUE约束

```
CREATE TABLE Dept{
DeptNo VARCHAR(10) PRIMARY KEY,
DeptName VARCHAR(50) UNIQUE,
ParentDeptNo VARCHAR(10),
Manager VARCHAR(10)
}
```



- NOT NULL约束

```
CREATE TABLE Diagnosis{
DGno VARCHAR(10) PRIMARY KEY,
Pno VARCHAR(10) NOT NULL,
Dno VARCHAR(10) NOT NULL,
Symptom VARCHAR(100),
Diagnosis VARCHAR(100),
DGtime DATETIME,
Rfee DECIMAL(18,2) NOT NULL
}
```



- CHECK约束
  - CHECK子句的通常应用是保证属性值满足指定的条件。CHECK子句括号内的条件可以是取值的简单限制。一个表中可以定义多个CHECK约束，在多个字段上定义CHECK约束时，则必须将CHECK约束定义为表级约束。CHECK约束不能包含子查询。
  - CHECK子句中的条件可以涉及关系表中的其他属性、元组。每当关系中插入一新元组或有元组被修改，CHECK约束中的条件都会被立即进行检查，若条件为假，更新操作被拒绝。

- 例：CHECK约束

```
CREATE TABLE Doctor{
Dno VARCHAR(10),
Dname VARCHAR(50) NOT NULL,
Dsex VARCHAR(2),
Dage INT,
Ddeptno VARCHAR(10),
Dlevel VARCHAR(50),
Dsalary DECIMAL(18,2),
PRIMARY KEY(Dno),
CHECK( Dsex IN (‘男’, ‘女’)),
CHECK( Dage > 0 AND Dage <60)
}
```

- Foreign Key约束
  - 设R和S是两个基本关系；
  - F是关系R的一个或一组属性，但不是关系R的码；
  - 如果F与关系S的主码K相对应，则称F是关系R的外码（Foreign-Key）；
  - R中每个元组在F上的值必须为：①或者取空值；②或者等于S中某个元组的主码值。

- 外码的声明
  - 若外码为单属性，则可在属性名称、类型声明之后，用REFERENCES指出被参照的关系、属性。形式为：

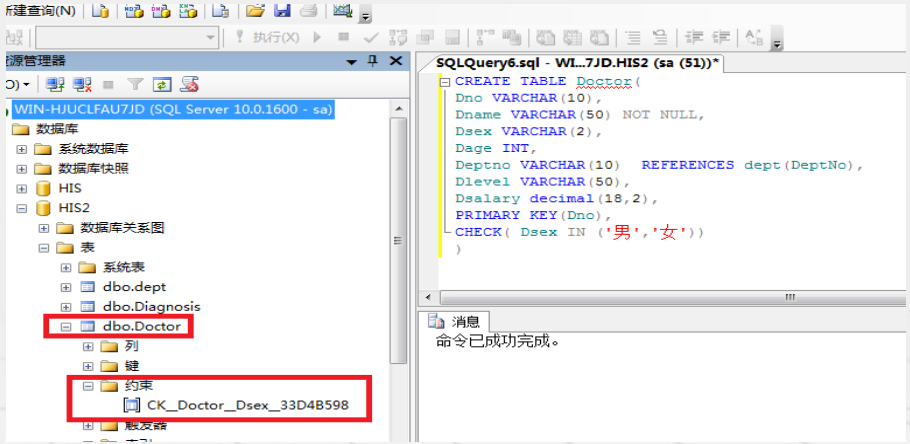
REFERENCES <被参照表表名>（<属性名>）

  - 在属性列描述之后，将一个或多个属性列声明为外码。形式为：

FOREIGN KEY（<属性名>）REFERENCES <被参照表表名>（<属性名>）

- 示例 ：Foreign Key的定义

```
CREATE TABLE Doctor ( Dno VARCHAR(10),
                      Dname VARCHAR(50) NOT NULL,
                      Dsex VARCHAR(2),
                      Dage INT,
                      Ddeptno VARCHAR(10)  REFERENCES Dept(DeptNo),
                      Dlevel VARCHAR(50),
                      Dsalary DECIMAL(18,2),
                      PRIMARY KEY(Dno),
                      CHECK( Dsex IN ('男','女'))
    )
```



- 示例 ：Foreign Key的定义

```
CREATE TABLE Doctor （ Dno VARCHAR(10),
                Dname VARCHAR(50) NOT NULL,
                Dsex VARCHAR(2),
                Dage INT,
                Ddeptno VARCHAR(10),
                Dlevel VARCHAR(50),
                Dsalary DECIMAL(18,2),
                PRIMARY KEY(Dno),
                CHECK( Dsex IN （‘男’，‘女’)),
                FOREIGN KEY Ddeptno REFERENCES Dept(DeptNo)
    )

                CONSTRAINT fk_Deptno FOREIGN KEY Ddeptno REFERENCES Dept(DeptNo)
```

RDBMS实现参照完整性时需要考虑的问题：

- 外码是否可以接受空值的问题
- 在被参照关系中删除元组时的问题
- 在参照关系中插入元组时的问题
- 修改关系中主码的问题

- 受限策略（RESTRICTED）
  - 系统的默认方式。
  - 当出现违背参照完整性规则的更新操作请求时，系统拒绝执行该操作。

- 例：受限策略示例

思考：在RecipeDetail中新增、修改Mno分量，在Medicine中删除、修改Mno的分量值时，系统如何处理？（教材P162）

```
CREATE TABLE RecipeDetail{
Rno VARCHAR(10),
Mno VARCHAR(10) NOT NULL,
Mamount DECIMAL(18,0),
PRIMARY KEY(Rno,Mno),
FOREIGN KEY(Mno) REFERENCES Medicine(Mno)
}
```

- 置空策略（SET-NULL）
  - 依照参照完整性规则，外码是可以取空值的。
  - 但具体能否取空值，要根据应用环境的语义来定。

- 例：置空策略示例

若用户删除 Dept表中某一行元组（设该元组的主码DeptNo的值为“内科”），则 Doctor 表中外码Ddeptno中所有“内科”均被更新为空值NULL。

- 定义置空策略

```
CREATE TABLE Doctor
 ( Dno VARCHAR(10) PRIMARY KEY,
  Dname VARCHAR(50) NOT NULL,
  Dsex VARCHAR(2),
  Dage INT,
  Dlevel VARCHAR(50),
  Dsalary DECIMAL(18,2),
  Ddeptno VARCHAR(10) REFERENCES Dept(DeptNo) ON DELETE SET NULL
 );
```

- 级联策略（CASCADE）
  - 不用拒绝用户操作请求的处理方式。
  - 连带处理参照数据。
- 例：级联策略示例

若用户删除 Dept表中某一行元组（设该元组的主码DeptNo的值为“内科”），则 Doctor 表中外码Ddeptno所有值为“内科”的元组均被同步删除。

- 定义级联策略

```
CREATE TABLE Doctor
 ( Dno VARCHAR(10) PRIMARY KEY,
  Dname VARCHAR(50) NOT NULL,
  Dsex VARCHAR(2),
  Dage INT,
  Dlevel VARCHAR(50),
  Dsalary DECIMAL(18,2),
  Ddeptno VARCHAR(10) REFERENCES Dept(DeptNo) ON DELETE CASCADE
 );
```

- 域约束
  - 通过CREATE DOMAIN可以定义一个新的域；
  - 通过对域进行约束可以达到对属性列取值的约束；
  - SQL92用一个特殊的关键字VALUE表示域的一个值。
  - 统一约束，便于修改。
- 定义域

```
CREATE DOMAIN SexVal CHAR(2)
CHECK (VALUE IN('男', '女'));
```

– 使用域

```
CREATE TABLE Patient{
Pno VARCHAR(10),
Pname VARCHAR(50) NOT NULL,
Psex SexVal,
Page INT,
Pino VARCHAR(50),
Pid VARCHAR(18),
PRIMARY KEY(Pno)}
```

- 表级约束只关联一张表。虽然在使用CHECK语句时其表达式中可以引用其他关系表，但表级约束要求关联的表是非空的。
- 当约束条件涉及两张、甚至更多的关系表时，这种表级约束的缺陷就显现出来，不能满足需要。
- SQL断言可以解决多张关系表关联的约束定义。
- 中的前面介绍的域约束和参照完整性约束是断言的特殊形式，它们容易检测并适用于很多数据库应用。
- 当创建断言时，系统要检测其有效性。如果断言有效，则以后只有不破坏断言的数据库修改才被允许。如果断言比较复杂，则检测会带来相当大的开销。因此，使用断言应该特别小心。

- 断言格式

CREATE ASSERTION <断言名> CHECK<谓词>

- 断言示例

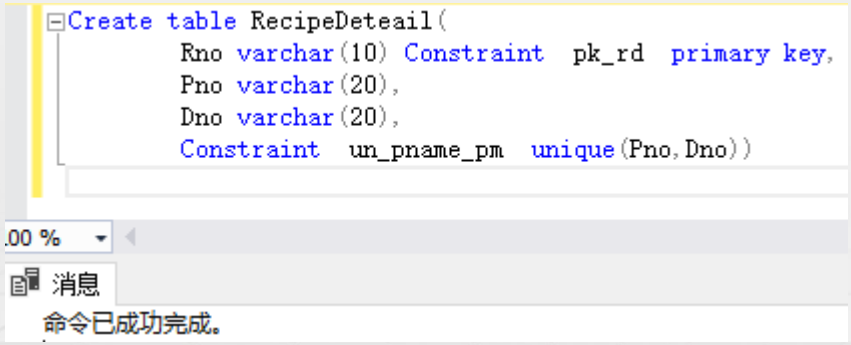
```
Create assertion salarycheck check(
Not exists(
     Select * from Doctor x
     Where Dsalary >= some ( select Dsalary from Doctor
        y
     Where x.Deptno=y.Deptno and y.Dno =(
          Select Manager from Dept
          Where x.Deptno =Dept.Deptno)
          )
)
```

- 任何时候都可以添加、修改、删除约束。
- 为了对约束进行修改、删除，有必要对约束命名。
- 约束命名：CONSTRAINT关键字后跟约束名称

```
CREATE TABLE RecipeDetail{
Rno VARCHAR(10),
Mno VARCHAR(10) CONSTRAINT notnullMno  NOT NULL,
Mamount DECIMAL(18,0),
CONSTRAINT pkRecipeDetailRnoMno PRIMARY
  KEY(Rno,Mno),
CONSTRAINT fkRecipeDetailMnoMedicine FOREIGN KEY
   (Mno) REFERENCES Medicine(Mno)
ON DELETE CASCADE
ON UPDATE CASCADE
}
```

约束的创建

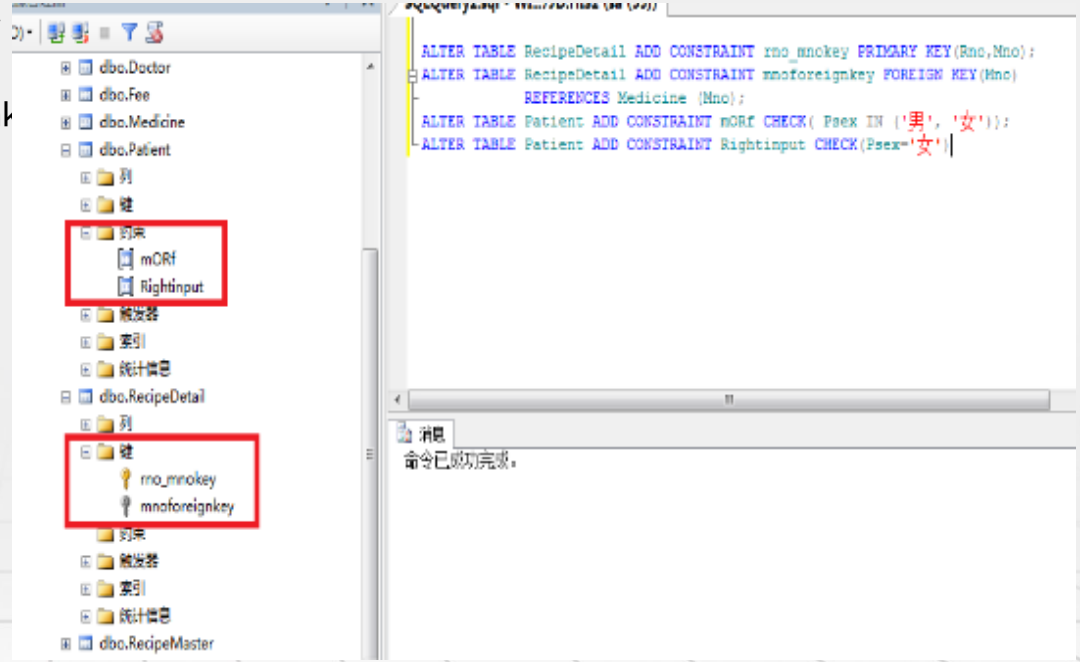
```
Create table RecipeDeteail(
        Rno varchar(10) Constraint  pk_rd  primary key,
        Pno varchar(20),
        Dno varchar(20),
        Constraint  un_pname_pm  unique(Pno,Dno))
```



约束的添加

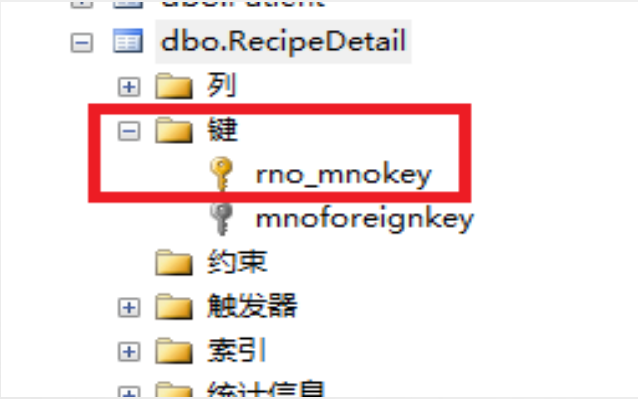
可以通过ALTER TABLE语句为已有表添加各种约束

```
ALTER TABLE RecipeDetail
        ADD CONSTRAINT rno_mnokey
ALTER TABLE RecipeDetail
        ADD CONSTRAINT mnoforeignk
(Mno);
```



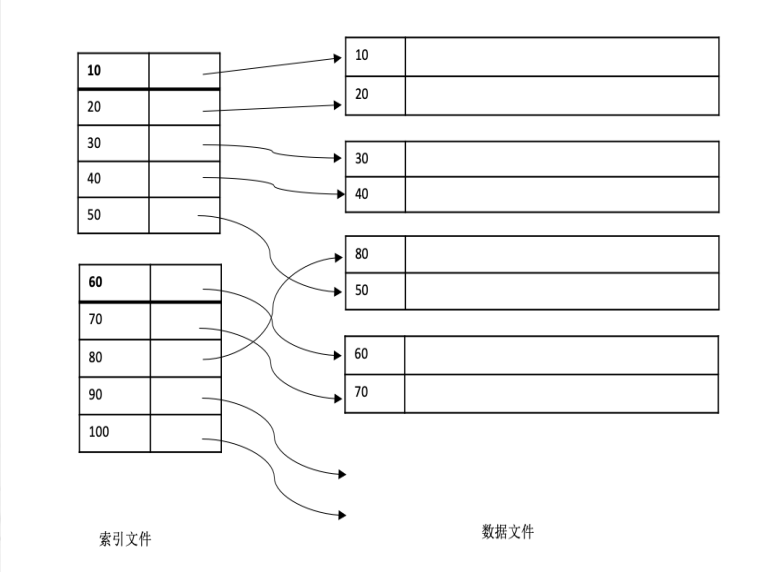
- 例：约束的删除

```
ALTER TABLE RecipeDetail DROP CONSTRAINT rno_mnokey;
```



索引的概念

- 索引是关于数据位置信息的关键字表
- 索引是提高查询性能的主要手段
- 索引通常采用采用哈希表或B+树等结构
- 索引由DBMS内部实现，属于内模式范畴



示例

```
select * from Doctor
where DdeptNo='101' and Dage<40
```

- 顺序扫描

直接扫描整张表，对表中所有行进行连续访问，把不满足where条件的行剔除。

- Doctor只在DdeptNo属性上建有索引

首先通过索引项的行指针访问所有在101部门工作的医生。当查询限定在101部门后，在挑选出的101部门的医生中选出Dage属性小于40的记录。

顺序索引

根据值的顺序排序

单级索引

所有的索引字段以及对应的文件位置按顺序一个个地排列出来

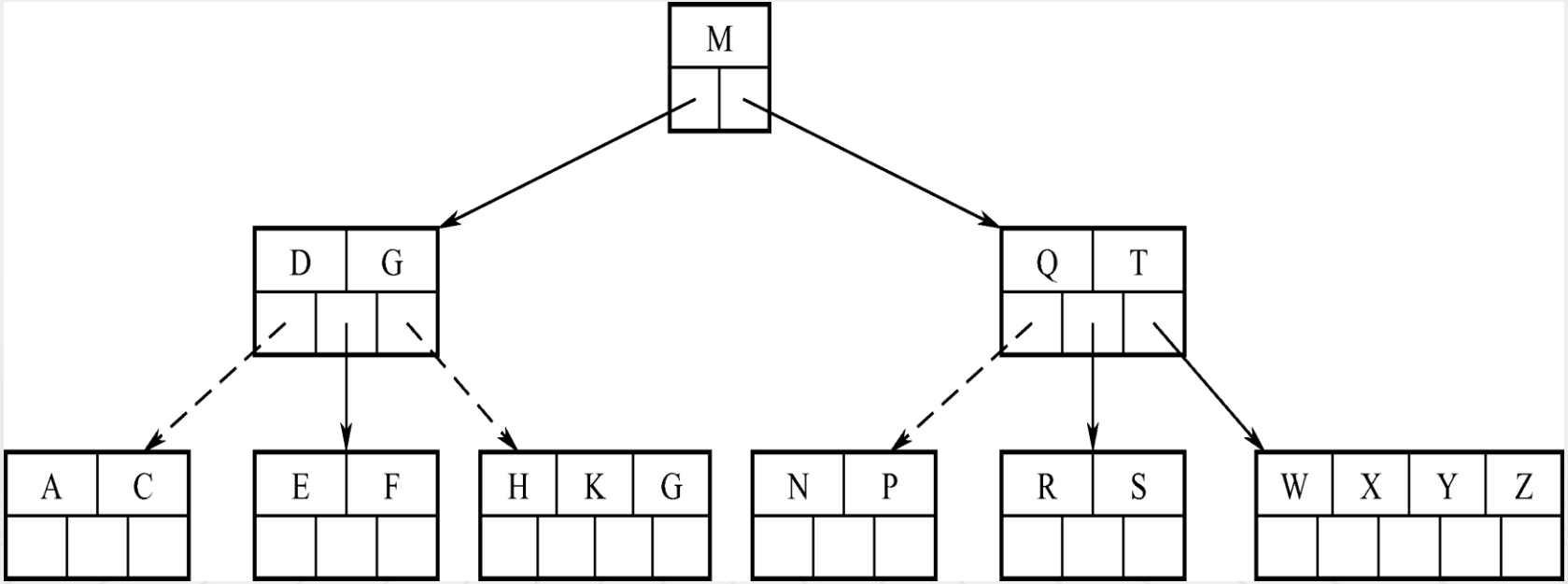
多级索引

在单级索引的基础上再加索引，也就是指向索引的索引

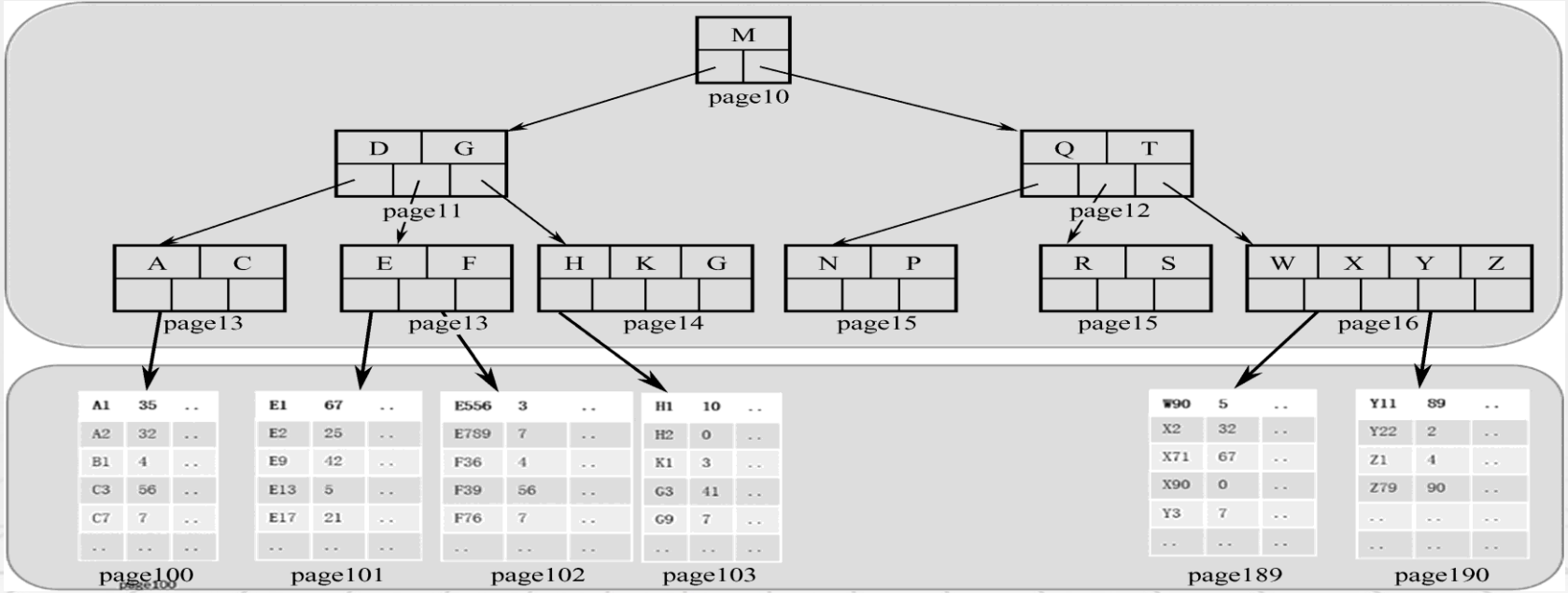
散列索引

将值平均分配到若干散列桶中，通过散列函数定位一个值所属的散列桶

将一个或几个属性列的索引键值按照一种次序存放起来

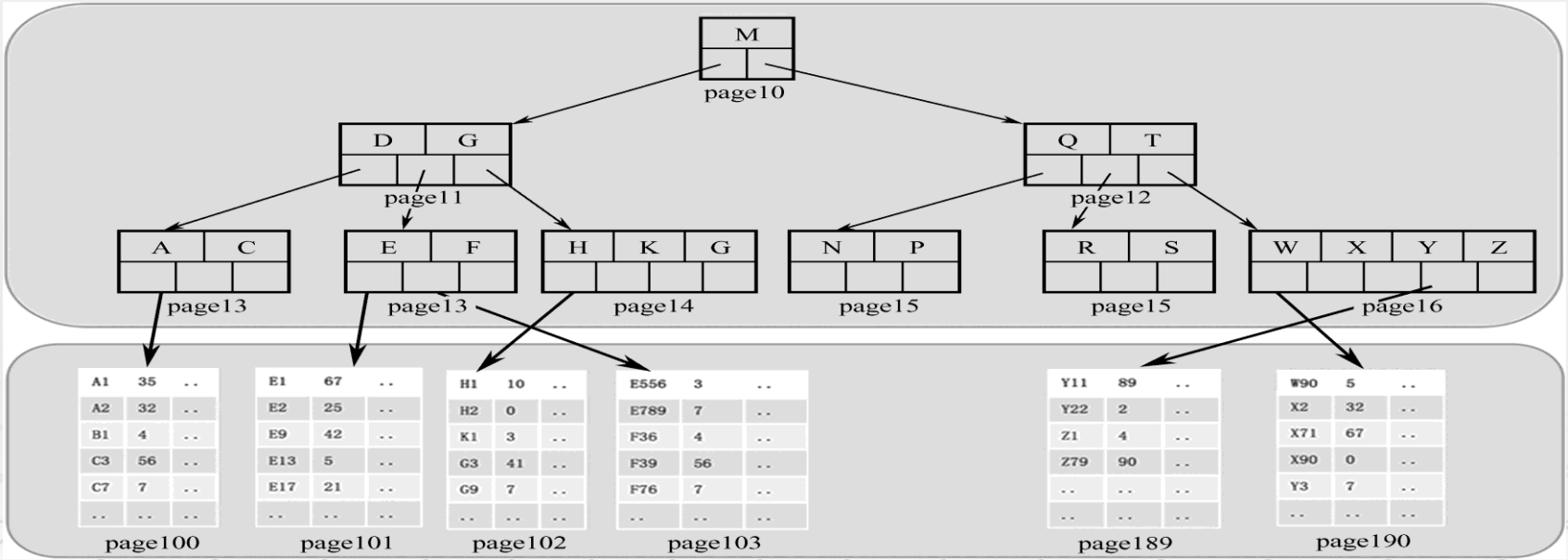


- 索引分类
  - 聚集索引
    - 索引项顺序与表中记录的物理顺序一致
    - 一个基本表上最多只能建立一个聚集索引



非聚集索引

- 数据和索引存储分开存储，索引带有指针指向数据的存储位置
- 索引中的项目按索引键值的顺序存储，索引项顺序与表中记录的物理顺序不必一致

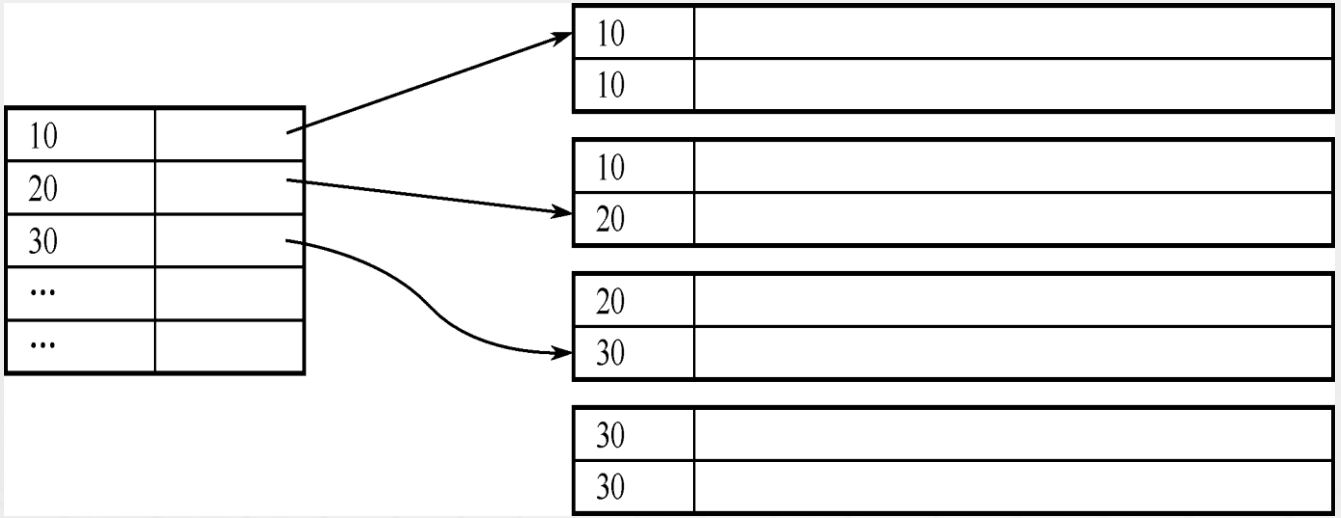


聚集索引与非聚集索引比较

动 作 描 述	聚 集 索 引	非聚集索引
列经常被分组、排序	适用	适用
返回某范围内的数据	适用	不适用
大数目的不同值	不适用	适用
小数目的不同值	适用	不适用
频繁更新列	不适用	适用
主键列	适用	适用
外键列	适用	适用

稠密索引

在顺序文件中，每个键都有一个索引键值与之相对应



稠密索引

- 索引块数量通常比数据块数量少。
- 由于键被排序，我们可以使用二分查找法来查找“80”。若有n个索引块，我们只需查找$\log_2 n$个块。
- 索引文件可能足够小，以至于可以永远存放在主存缓冲区中。如果这样的话，查找键时就只涉及主存访问而不需要执行I/O操作。

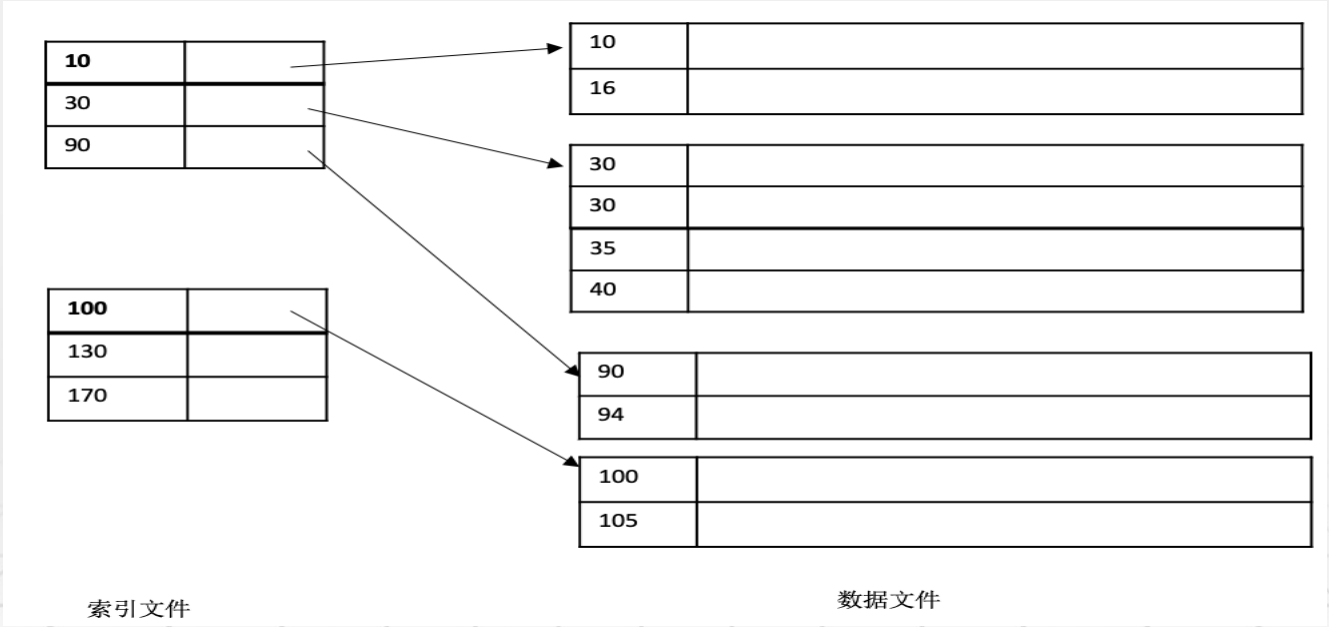
稠密索引

示例

假设一个关系有一百万条记录，大小为4KB的存储块可存放10个这样的记录，则这个关系需要的存储空间超过400MB。若关系的键字段占30B，指针占8B，加上块头所需空间，那么我们可以在一个4KB存储块中存储100个键-指针对。这样一个稠密索引需要1万个存储块，占40MB。我们就有可能为这样一个索引文件分配主存缓冲区。$\log_2 10000$大约是13，采用二分查找法我们只需访问13～14个存储块就可以查找到给定键值。

稀疏索引

只在索引文件中为每个数据块的第一条记录建立索引键值



稀疏索引

稀疏索引相较于稠密索引节省了存储空间，但查找给定值的记录需要更多的时间。

示例

稠密例子中，一百万条记录占用了10个数据存储块，每个索引块存放100个键-指针对，若使用稀疏索引，只需要1000个索引存储块，占用4MB空间。这样大小的文件完全可以存到主存中。

散列索引

通过散列函数进行定位。散列文件组织将数据的某个键作为参数带入散列函数h，计算出一个介于0～B-1的整数（B为桶的数目），之后将记录连接到桶号为h(K)的桶中进行存储。

- 分布是均匀的。散列函数从所有可能的搜索码值集合中为每个桶分配同样数量的搜索码值。
- 分布是随机的。不管搜索码值实际怎么分布，每个桶应分配到的搜索码值数目几乎相同。

示例

假设我们要在Doctor表中求salary的一个散列函数。假设最低工资为2500，最高工资为9500。我们可以使用一个散列函数把这些值分成7个区间：2500～3500，3501～4500，…，8501～9500。对于这个散列函数，搜索码值的分布是均匀的（每个桶里有相同数目的不同的salary值），但不是随机的。若工资在5000～6500之间的记录比在其他区间的记录要多，其结果就是记录的分配是不均匀的，某些桶中的记录数比其他桶的多。如果该函数分布随机，就应使所有的桶拥有大致相等的记录数。

- 选择数据量较大的表建立索引
  - 一般来说，对于数据量较大的表，数据库系统越有机会找到最短路径，索引越能更好地改善响应的时间，越能显示出优势。
  - 索引对于列中的数据多而杂的列是特别有用。例如，在医院信息系统中，如果对患者诊断信息建立索引，速度提高的效果就比较明显。但是，不适宜在性别列上建立索引，因为有大量重复值，对其索引反而会降低查询速度。
  - 对于数据量较小的表最好不要建立索引，因为对小表索引，速度提高不仅不明显，反而会增大系统的开销，除非有特殊需要，要建立唯一索引来加强唯一。

- 建立索引的数量要适量（需要付出代价）
  - 尽管对一个基表可以建立多个索引，提高查询速度，但不宜建立太多的索引，最好不超过3个；
  - 索引要占用磁盘空间；
  - 系统要维护索引结构，维护索引结构系统要花费一定的开销，尤其是经常要插入或删除的表，其维护索引结构的代价是很大的，因此建立索引会减慢插入、修改、删除的执行速度；
  - 用户应该在加快查询速度和降低更新速度之间作出权衡。对于一个仅用来查询的表来讲，建立多个索引是比较合适的，但对更新操作比较频繁的表来讲最好少建立一些索引。

- 选择合适的时机建立索引
  - 通常，建立索引应选择在表中装入数据之后。如果先建立索引后装入数据，则每次插入一行数据都要对索引进行更新，这样会很浪费时间。
  - 但是，如果要保证装入数据的唯一性，则只能以牺牲系统性能为代价，而在装入数据前建立唯一性索引。

- 优先考虑主键列建立索引
  - 当主键包含多列时，最好把数据差异最多的列放在索引命令列表的首位。
  - 如果各列数据种类相近，则最好把经常用到的列放在前面。
  - 最好选择包含大量非重复值的列，如医生编号。
  - 如果只有很少的非重复值，如性别只有男和女，最好不要使用索引查询，此时采用顺序扫描更为有效。

- 创建索引语法格式

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
   INDEX <索引名>
ON < 基表名 | 视图名>( 列名[ ASC | DESC ]
   [ ,...n ] )
```

- 创建索引示例：单列索引
  - 在药品基本信息表中，假如在药品名称上，按照升序创建非聚集索引。

```
CREATE NONCLUSTERED INDEX MedIndex
ON Medicine(Mname ASC)
```

- 创建索引示例：复合索引
  - 在处方详细信息表中，假如在处方编码和药品编码上，创建聚集索引。

```
CREATE CLUSTERED INDEX RDIndex
ON RecipeDetail (Rno ASC,Mno DESC)
```

- 创建索引示例：唯一索引
  - 在医生基本信息表上，假如在医生编码上，创建唯一索引。

```
CREATE UNIQUE INDEX DoctorIndex
ON Doctor (Dno ASC)
```

删除索引语法

```
DROP INDEX 索引名
```

删除索引示例

- 删除DoctorIndex索引。

```
DROP INDEX DoctorIndex
```