连接

https://blog.csdn.net/qq_36961226/article/details/112802896?ops_request_misc=&request_id=&biz_id=102&utm_term=mysql%E8%A1%A8%E8%BF%9E%E6%8E%A5%E7%9A%84%E5%87%A0%E7%A7%8D%E6%96%B9%E5%BC%8F%E5%9B%BE%E8%A1%A8&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-3-112802896.142^v13^pc_search_result_control_group,157^v14^new_3&spm=1018.2226.3001.4187

sqlserver和mysql的区别

递增语句

mysql的递增语句是AUTO_INCREMENT，而mssql是identity(1,1)

```
create table table1(id int identity,t varchar(500))
//即从1开始递增，每次自增1。
```

mysql不支持nchar,nvarchar,ntext类型

mysql支持enum,和set类型，sql server不支持

enum其实就是sqlserver中check

set和enum的区别：

https://blog.csdn.net/weixin_43054397/article/details/91048642?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216550429091678143542626%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=165504290916781435426262&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-91048642-null-null.142^v13^pc_search_result_control_group,157^v14^new_3&utm_term=mysql%E4%B8%ADenum%E5%92%8Cset%E7%9A%84%E5%8C%BA%E5%88%AB&spm=1018.2226.3001.4187

l enum类型的字段类似于单选按钮的功能，一个enum类型的数据最多可以包含65535个元素。

l set 类型的字段类似于复选框的功能，一个set类型的数据最多可以包含64个元素。

限制查询

Mysql使用谓词limit查询某几行记录，SQL SERVER使用top，语法格式如下。

select字段列表

from数据源

limit [start,]length; 下标start是第start+1行

没写start的话默认是0

select top 10 //查看前10行

from数据源

查询区别

分组总运算

ith rollup:对各组进行总运算时，需要在分组后面加上一条汇总记录，可以通过with rollup实现。

例： 查询每个课程所修读的学生人数以及全部课程总的修读学生人数

SELECT Cno,COUNT(Sno)

FROM SC

GROUP BY Cno

WITH ROLLUP;

Cno	COUNT(Sno)
001	11
002	12
003	12
004	4
(Null)	39

group_concat():按照分组字段，将另一个字段的值（NULL值除外）使用逗号连接起来。

例：查询每个系老师的名字，要求同一个系的老师名字在一行全部列出来。

SELECT DNo,group_concat(TName) 系里教师名字

FROM teacher

GROUP BY DNo

DNo	系里教师名字
01	张一,张二,张三,张四,张五,张六,张七
02	李一,李二,李三,李四
03	王一,王二,王三,王四
04	孙一,孙二,孙三,孙四,孙五

正则表达式

正则表达式由一些普通字符和一些元字符构成，普通字符包括大写字母、小写字母和数字，而元字符具有特殊的含义。

元字符	说明
.	匹配任何单个的字符
^	匹配字符串开始的部分
$	匹配字符串结尾的部分
[字符集合或数字集合]	匹配方括号内的任何字符，可以使用'-'表示范围。例如[abc]匹配字符串"a"、"b"或"c"。[a-z]匹配任何字母，而[0-9]匹配任何数字
[^字符集合或数字集合]	匹配除了方括号内的任何字符
字符串1|字符串2	匹配字符串1或者字符串2
*	表示匹配0个或多个在它前面的字符。例如x*表示0个或多个x字符，.*表示匹配任何数量的任何字符；
+	表示匹配1个或多个在它前面的字符。如a+表示1个或多个a字符
?	表示匹配0个或1个在它前面的字符。如a?表示0个或1个a字符。
字符串{n}	字符串出现n次
字符串{m,n}	字符串至少出现m次，最多出现n次

例： 检索以"j"开头，以"程序设计"结尾的课程信息，可以使用下面的SQL语句。

select * from course where course_name regexp '^j.*程序设计$';

例：检索学生联系方式中以15开头或者18开头，且后面跟着9位数字的学生信息，可以使用下面的SQL语句。

select * from student where student_contact regexp '^1[58][0-9]{9}';

mysql表有两种存储引擎，默认是innodb，

InnoDB和MyISAM的区别

l InnoDB 支持事务，MyISAM 不支持事务。这是 MySQL 将默认存储引擎从 MyISAM 变成 InnoDB 的重要原因之一；

l InnoDB 支持外键，而 MyISAM 不支持。对一个包含外键的 InnoDB 表转为 MYISAM 会失败；

l InnoDB 是聚集索引，MyISAM 是非聚集索引。

l InnoDB 不保存表的具体行数，执行 select count(*) from table 时需要全表扫描。而MyISAM 用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

l InnoDB 最小的锁粒度是行锁，MyISAM 最小的锁粒度是表锁。一个更新语句会锁住整张表，导致其他查询和更新都会被阻塞，因此并发访问受限。这也是 MySQL 将默认存储引擎从 MyISAM 变成 InnoDB 的重要原因之一

create table test(

....

)ENGINE=INNODB;

数据库的创建

create database <数据库名字>

创建数据库时设置字符编码

使用 create database 数据库名 character set utf8; 创建数据库并设置数据库的字符编码。

```
create database MyDB_two character set utf8;
character set 可以缩写成 charset ，效果是一样的。
```

查看数据库

使用 show create database 数据库名; 显示数据库的创建信息。

```
show create database MyDB_one;
show create database MyDB_two;
```

显示当前数据库 select database();

```
select database();
```

修改数据库

```
alter database MyDB_one character set utf8;
```

进入或者切换数据库

使用 use 数据库名 进入或切换数据库。

```
use MyDB_one
```

数据库的删除

drop database <数据库名字>

约束类型

约束类型有六种，primary key　//主键可以由多个列组成，主键一定是not null

not null

default

check

unique

foreign key

主键是能确定一条记录的唯一标识，最多只能有一个

外键用于与另一张表的关联，是能够确定另一张表记录的字段，用于保持数据的一致性，可以有多个

唯一键确定了数据的唯一性

表的创建

create table <表名>

(<列名> <数据类型> [列级完整性约束条件],

……

<列名> <数据类型> [列级完整性约束条件]

[表级完整性约束条件]

);

举例

create table biao

(

学号 char(5) not null, //默认为null

unique,

default '男',

primary key

primary key（学号）

check（学号 between 15 and 25）

check (学号 in('男','女'))

foreign key(学号) references 另一个表名(学号) on delete cascade //可在后面补充on update cascade/action/set null

on delete no action

on delete set null

);

外键在删除以及更新时候cascade串联

删除：删除主表自动删除从表，删除从表，主表不变

更新：更新主表自动更新从表，更新从表，主表不变

而no action则删除以及更新时候

删除：从表记录不存在，主表才可以删除，删除从表，主表不变

更新：从表记录不存在，主表才可以更新，更新从表，主表不变

set null

删除：删除主表自动更新从表为null，删除从表，主表不变

更新：更新主表自动更新从表为null，更新从表，主表不变

数据类型

字符串

char(n)　固定长度为n的字符串

varchar(n) 可变长度的字符串，长度最多为n

text 可变长度的字符串，最多为2GB字符数据

nchar(n)，nvarchar(n)，ntext　Unicode数据，每一个存储单位占两个字节

数值类型

bigint

int

smallint

tinyint

float(n) n为用于存储float数值尾数的位数，以科学记数法表示

real 浮点型，使用4字节存储

decimal(p[,s])　定点数，由p位数字(不包括符号，小数点)组成，小数后面有s位数字

numertic(p[,s]) 定点数，由p位数字(不包括符号，小数点)组成，小数后面有s位数字

货币数值

smallmoney　-214,748.3648----214,748.3647

money　-922,337,203685,477.5808----922,337,203,685,477.5807

binary类型

bit　允许0,1,或null

image 图像类型，可变长度的二进制数据，最多2GB

日期类型

datetime　年月日分秒 精度为3.33毫秒

datetime2 年月日分秒 精度为100纳秒

smalldatetime　1900年1月1日到2079年6月6日　精度为一分钟

datetimeoffset 与datetime2相同，外加时区偏移

timestamp 时间戳，存储唯一的数字，每当创建或者修改某行时，该数字会更新

表的更新

增加属性列

alter table <表名> add <新列名> <数据类型>[完整性约束条件]

删除属性列

alter table <表名> drop column <列名>

修改属性列

alter table <表名> alter column <列名><类型>

查看所有约束

exec sp_helpconstraint <表名>

补充定义主键

alter table <表名> [constraint <约束名>] add primary key(列名)

删除约束

alter table <表名> drop constraint <约束名>

增加约束

alter table <表名> add [constraint <约束名>]约束类型　//不加约束名则系统生成

表的删除

drop table <表名>

索引的建立和删除

creat [unique]　[cluster|nocluster]　index <索引名> on <表名>(<表名>[asc|desc] [,<表名>[asc|desc]]……)

索引可以建在该表的一列或多列上，各列名之间用逗号分隔。每个列名后面还可以用排序次序指定索引值的排列次序，包括asc(升序)和desc(降序)两种，缺省值为asc。unique表明此索引的每一个索引值只对应唯一的一个元组。

索引类型：唯一索引，聚簇索引，非聚簇索引

唯一索引：不允许其中任何两行具有相同值的索引

聚簇索引：其物理存放顺序与主键顺序一致。因为数据只有一个物理存放顺序，所以一个表只有一个聚簇索引。默认主键为簇索引，所以一般建立的都是非簇索引。

非聚簇索引：非聚集索引并不改变其所在表的物理结构，而是额外生成一个聚集索引的B树结构，叶节点的信息也是按聚簇索引的键值按顺序存储，叶节点的信息存储的并不是实际的数据，是相应数据对象的存放地址指针。

查看索引

exec sp_helpindex <表名>

删除索引

drop index 索引名

查表

格式为：

select [all|distinct]<列名>[，<列名>]...

from <表名或视图名>[，<表名或视图名>]...

[where<逻辑表达式>]

[group by <列名1>]

[having <逻辑表达式>]

[order by <列名2>[asc|desc]];

选全部列则用*

目标列表达式不仅可以是算术表达式，还可以是字符串常量，函数，如果要给新的列起名字，则<新列名>=...

查询满足条件的元组

查询满足指定条件的元组可以通过where句实现，where子句常用查询表达如下

比较　=,>,<,>=,<=,!=,<>,!>,!<,not+上述比较

确定范围 between and，not between and

确定集合 in，not in

字符匹配 like，not like

空值　　is null，is not null

多重条件(逻辑运算)　and，or，not

字符匹配

[not] like '<匹配串>' [escpae '<换码字符>']

%表示任意长度

_表示单个字符

对查询结果排序

从上往下升序 order by <列名>asc

从上往下降序 order by desc，<列名> asc/desc //再次排序

聚集函数

(f). 聚集函数

- 计数
  COUNT（[DISTINCT|ALL] *）
  COUNT（[DISTINCT|ALL] <列名>）
- 计算总和
  SUM（[DISTINCT|ALL] <列名>）
- 计算平均值
  AVG（[DISTINCT|ALL] <列名>）
- 最大最小值
  MAX（[DISTINCT|ALL] <列名>）
  MIN（[DISTINCT|ALL] <列名>）

指定DISTINCT短语，则表示在计算时先要取消指定列中的重复值。如果不指定DISTINCT短语或指定ALL短语（ALL为缺省值），则表示不取消重复值。

计数

count（[all|distinct]*）

count（[all|distinct]<列名>）

总和

sum（[all|distinct]<列名>）

最大值最小值

max（[all|distinct]<列名>）

min（[all|distinct]<列名>）

指定distinct短语，则表示在计算时先要取消指定列中的重复值，如果不指定distinct短语或指定all短语（all为缺省值），则表示不取消重复值

from注意：集合函数只能用在select子句和having子句中

分组

group by

将表中的各行按一列或多列取值相等的原则进行分组

例如学生以及学生成绩，将学生名字相同分成组，则用聚集函数sum（[all|distinct]<列名>）则可以算总和

having和where子句的区别

作用对象不同

where子句作用于基表或视图，从中选择满足条件的元组

having短语作用于组，从中选择满足条件的组

多表查询

from <表一>，<表二>

嵌套查询

嵌套到where

查询修读课程号为001的所有学生姓名

```
select s.sn
from s
where s.s# in
  (select sc.s#
from sc
where sc.c#='001'
)
```

例34：查询不修读课程号为001的所有学生的姓名。

```
SELECT S.Sn
FROM S
WHERE S.S# NOT IN
     (SELECT SC.S#
      FROM SC
      WHERE SC.C#='001')
```

常见错误：

```
SELECT S.Sn
FROM S, SC
WHERE S.S# =SC.S# AND SC.C#!='001'
```

右边错了是因为例如小明 001 小明 002 小明 001，就删除了小明001，但实际上小明选了001还是把小明名字打上去了。因为笛卡尔积中有多个小明多个课程。

凡是多个项目中有了一个就不要则只能用in了

form中嵌套要起别名

查询CS系中年龄在23岁以下的学生姓名

```
SELECT S1.Sn
FROM
 (SELECT *
    FROM S
   WHERE Sa<=23) AS S1
WHERE S1.SD='CS'
```

连接

自连接

一个表与其自己进行连接

需要给表起别名以示区别

由于属性名都是同名属性，因此必须用别名前缀，as可以省略，另外列名也可以用as，例如以下as 1

举例

SELECT SC1.S# as 1

FROM SC AS SC1,SC AS SC2

WHERE SC1.C#=SC2.C# AND SC2.S#='00008';

表名 as 别名

别名.属性名

A	B	C
A1	B1	5
A1	B2	6
A2	B3	8
A2	B4	12

B	E
B1	3
B2	7
B3	10
B3	2
B5	2

笛卡尔积

A	m.B	C	n.B	E
A1	B1	5	B1	3
			B2	7
			B3	10
			B3	2
			B5	2
A1	B2	6	B1	3
			B2	7
			B3	10
			B3	2
			B5	2
A2	B3	8	B1	3
			B2	7
			B3	10
			B3	2
			B5	2
A2	B4	12	B1	3
			B2	7
			B3	10
			B3	2
			B5	2

内连接（等值连接）

相当于上图中m表和n表中返回m.b=n.b的列

即select*

from m

inner join n on m.b=n.b

等同于

select*

from m，n

where m.b=n.b

左外连接

left join 或者 left outer join

不仅仅是联接列所匹配的行（内连接），如果坐标的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表均为空值

右外连接

right join 或者 right outer join

不仅仅是联接列所匹配的行（内连接），如果坐标的某行在左表中没有匹配行，则在相关联的结果集行中左表的所有选择列表均为空值

外连接

full join 或者 full outer join

也就是内连接+左连接补上的行+右连接补上的行

exists

查询为空就假，查询存在一个！就真，存在给列名无实际意义则用*

exists难点

由于只有exists没有全称量程for all，则要将全部转化为存在

查询所学课程包含了学号为00003号学生所选课程的所有学号改成

不存在一门课，学生00003号学生选了，而其他学生没选。

对于难题使用NOT EXISTS查询选修了课程学分为 4 的全部课程的学生的姓名；
我们可以这么理解
可以读成：
学生表中这样的学生姓名
不存在课程表中学分为4的课程
选课表中的学生没选到这门课

```
select  SName
from Student
where not exists
(
select *
from course
where CCredits=4 and not exists
(
select *
from sc
where sc.CNo=course.CNo and sc.SNo=Student.SNo
)
)
```

学生表中这样的学生姓名
对应select Sname from Student
不存在课程表中学分为4的课程
对应where not exists (select * from course where CCredits=4
选课表中的学生没选到这门课
对应and not exists (select * from sc where sc.CNo=course.CNo and sc.SNo=Student.Sno

选课表中的学生没选到这门课，没选到就是not exists。选课表中的学生就是 sc.SNo=Student.SNo，这门课就sc.CNo=course.CNo

并操作union

表一 union 表二，并集，去重。 union all 不去重

相当于where中逻辑一or逻辑二

表一　intersect 表二，交集

相当于where中逻辑一and逻辑二，但与and有一个区别，但逻辑一与逻辑二指同一个东西举例数字等于1或者等于2时候，则不能用and，则用intersect。

集合a-集合b

则用except，举例18岁的人减去男生，即18岁的女生

any和all

用在where中

any（相当于max（））

查询其他系中比信息系任意一个学生年龄小的学生姓名和年龄

```
SELECT Sn,Sa
FROM S
WHERE Sa<ANY(SELECT Sa
    FROM S
    WHERE Sd='IS')
  AND Sd <>'IS';
```

即元素a小于任意一个元素a（带有限制的a）

也就是小于最大的那个

all（相当于min（））

查询其他系中比信息系所有学生年龄都小的学生姓名及年龄。

```
SELECT Sn,Sa
FROM S
WHERE Sa<ALL(SELECT Sa
    FROM S
    WHERE Sd='IS')
  AND Sd <>'IS';
```

即元素a小于所有的元素a（带有限制的a）

也就是小于最小的那个

数据更新

删除

alter　from <表名> where <条件>

删除计算机系全体学生的选课记录

delete

from s，sc

where s.s#=sc.s#and s.sd='cs'

是不对的，会将学生系统中计算机系的学生也删除了

应该

delete

from sc

where s# in（SELECT S#

　　FROM S

　　WHERE SD='CS'）

删除全部数据truncate和delete的区别

truncate table <表名>

delete from <表名>

truncate是清空所有的数据，速度快，不可回滚，删除整张表后重建

delete逐行删除数据，每一步删除都有日志记录，可以回滚数据

插入

insert

into <表名>[<列名>[,<列名>]...]

values（<常量>[,<常量>]...）<子查询>

into可指定列，也可不指定，指定列则输入指定分量就好，不指定的则所有分量都要输入

子查询即将查询结果的列插入

例如

假设SC#表和SC表结构一样，将001这门课的学生选课信息插入到SC#。

INSERT INTO SC#

SELECT *

FROM SC

WHERE C#='001'

利用旧表创建新表

select

into

from

where

多了个into，即into<表名>，注意如果该表不存在，则插入时候会自动创建一个合适的表，

若要造一个一模一样的空表则where写成1=0就好了

数据修改

update<基本表名>

set<列名>=表达式[，<列名>=表达式...]

[from<数据来自哪个表>]

where<逻辑条件>

视图

视图是由若干基于经映像语句而构成的标，是一种导出表，这种表本身不存在于数据库内，在库中知识保留其构造定义（即映像语句）。只有在实际操作时，才将它与操作语句结合转化成对基本表的操作，因此视图也称为虚表virtual table。视图除了在更新方面有较大的限制和在查询时有个别限制外，可以像基本表一样，进行各种数据操作。

创建视图

create view<视图名>([<列名>][,<列名>])

as(映像语句)

[with check option]

映像语句其实就不含有order by子句的select语句

重点wtih check option

wtih check option表示用视图进行更新插入删除操作，即update，insert，delete操作要保证更新的元组满足视图定义中的谓语条件，也就是where语句。即映像语句中的条件表达式。

不加wtih check option则随意更新插入删除。

注意更新插入删除的是基本表，而不是视图表，但是更新插入删除的基本表数据也会映射到是图标。

举例

CREATE VIEW　S_C_G(Sn,Cn,G)

AS SELECT S.Sn,C.Cn,SC.G

FROM S,C,SC

WHERE S.S#=SC.S# AND SC.C#=C.C#;

如下，用*号不指定列的缺点在于修改基表的结构，基表与视图的映像关系被破坏，导致视图不能正确工作

CREATE VIEW CS_S(S#,Sn,Sd,Sa)

AS SELECT *

FROM S

WHERE Sd='CS'

WITH CHECK OPTION;

撤销视图

drop view<视图名>

视图查询

视图可以像基本表一样查询。

视图更新

视图更新本质是更新基本表

规则：一个视图如果是从多个表连接操作导出，对视图执行更新操作，每次只能影响一个表，不能对多个表同时影响。

创建视图用了分组和聚集函数则不允许对视图进行更新操作。

视图作用

视图能够简化用户的操作

视图使用户能以多种角度看待同一数据

视图对重构数据库提供了一定程度的逻辑独立性

视图能对机密数据提供安全保护

适当的利用视图可以更清晰的表达查询

数据库安全

问题的提出

数据库特点在于数据共享，数据共享必然带来数据库的安全性问题。数据库数据不能是无条件的共享。

自主存取控制方法grant和revoke

创建用户

使用存储过程创建数据库用户

创建登录用户，密码和默认数据库

exec sp_addlogin ‘用户名1’，’密码’，‘数据库‘

创建数据库账号

exec sp_grantdbaccess 用户名1　//授予数据库可接触

使用create login创建

创建登录用户，密码和默认数据库

create login 用户名2 with password='密码'，default_database=数据库名字

创建数据库账号

create user 用户名2 for login 用户名2 with default_schema=dbo;

授予语句

grant <权限>[,<权限>]

[on<对象类型><对象名>]

to<用户>[,<用户>]

[with grant option]

授予用户名1对sc表有对sno列的select和update操作

grant select,update(sno)

on sc

to 用户名1

若是不加sno，则是对所有列有权限

with grant option

指定：可以再授予

没有指定：不能传播

创建表授权

创建表授权予给user1

grant creat table to user1

回收

revoke <权限>[,<权限>]

[on<对象类型><对象名>]

from<用户>[,<用户>]

举例从用户user1手中收回关系Student上的查询和修改权，并且是级联收回

注意是级联收回！！！

revoke select，update

on Student

from user1

cascade

数据库角色

角色是权限的集合

角色的创建

create role<角色名>

角色授权

grant<权限>[,<权限>]　　权限有select（可以括号具体到某一列）

on<对象类型><对象名>

to<角色>[,<角色>]

将一个角色授予给其他用户或角色!

exec sp_addrolemember<角色1>,<用户>[,<角色2>]

例子：创建一个角色rol

create role rol

将对关系表Student的查询，更新和插入授予给角色rol

grant select,uppdate,insert

on Student

to rol

将rol角色授予user1

eexec sp_addrolemember rol,user1

增加角色的权限

grant 权限

on 表

to 角色

角色权限的收回

revoke<权限>[,<权限>]

on<对象类型><对象名>

from<角色>[,<角色>]

数据库固有角色

db_owner：在数据库中有全部权限。

db_accessadmin：可以添加或删除用户ID。

db_ddladmin：可以发出ALL DDL操作的所有权。

db_securityadmin：可以管理全部权限、对象所有权、角色和角色成员资格。

db_backupoperator：可以发出DBCC、CHECKPOINT和BACKUP语句。

db_datareader：可以选择数据库内任何用户表中的所有数据。

db_datawriter：可以更改数据库内任何用户表中的所有数据。

db_denydatareader：不能选择数据库内任何用户表中的任何数据。

db_denydatawriter：不能更改数据库内任何用户表中的任何数据。

public：一个特殊的数据库角色，数据库中的每个用户都是其成员。不能将用户、组或其他角色指定给public角色，在每一个数据库中都包含public角色，且不能删除这个角色。

细节

一个用户拥有角色，则角色权限是并集，

若角色a有权限1，2

角色b有权限2，3

用户有角色a和角色b后去掉角色b，权限2不会被撤销。还有角色a权限1，2

应用程序角色

默认情况下，应用程序角色是非活动的，需要用密码激活，当应用程序角色被激活以后，这次服务器连接将暂时失去所有应用于登录账号，数据库用户等权限，而只拥有与应用程序相关的权限，在断开本次连接以后，应用程序失去作用。

创建应用程序角色赋予权限

use <数据库>

go

create application role <角色> with password=<‘密码’>，default_schema=dbo

go

grant <权限>[,<权限>]

on <表>

to <角色>

激活应用程序角色

角色被激活后，拥有应用程序角色权限，原有的权限短暂失去，断开连接后重获原有的权限

exep sp_setapprole @rolename=<‘角色名’>,@password=<'密码'>

审计

什么是审计?

将用户对数据库库的所有操作记录在上面

可利用审计日志找出非法存取数据的人，时间和内容

创建服务器审计

to file：指定输出到审核文件，也可以指定为security log和application log

filepath：审核文件的目录地址

maxsize：单个审核文件的最大容量

max_files：类似于trace，指定rollover允许最多文件数

reserve_disk_space：预先分配审核文件到maxsize

queue_delay：指定事件发生到被强制审核的毫秒间隔，指定为0则为同步审核

on_failure：当审核向上档写入数据失败时，接下来会采取的行为，
continue|shutdown|fail_operation

```
USE master
GO
 CREATE SERVER AUDIT [Audit-AuditTest]
TO FILE
( FILEPATH = 'D:'
 ,MAXSIZE = 50 MB
 ,MAX_FILES = 10
 ,RESERVE_DISK_SPACE = ON)
WITH
( QUEUE_DELAY = 1000
 ,ON_FAILURE = CONTINUE)
```

创建完后默认是不启动审计

启动审计

alter server audit[audit-audittest] wtih (state=on)

统计数据库安全性

任何查询要涉及n足够大以上的查询

任意两个查询的相交数据项不能超过m个

任一用户的查询次数不能超过1+（n-2）/m

实体完整性定义

在列级定义主码

```
CREATE TABLE Student
        (Sno CHAR(9) PRIMARY KEY,
         Sname CHAR(20) NOT NULL,
         Ssex CHAR(2) ,
         Sage SMALLINT,
```

Sdept CHAR(20));

在表级定义主码

```
CREATE TABLE Student
  (Sno CHAR(9),
   Sname CHAR(20) NOT NULL,
   Ssex CHAR(2) ,
   Sage SMALLINT,
   Sdept CHAR(20),
   PRIMARY KEY (Sno)
  );
```

将SC表中的Sno，Cno属性组定义为码

```
  CREATE TABLE SC
    (Sno  CHAR(9) NOT NULL,
     Cno CHAR(4) NOT NULL,
     Grade  SMALLINT,
     PRIMARY KEY (Sno，Cno)  /只能在表级定义主码/
    );
```

实体完整性检查和违约处理

RDBMS按照实体完整性规则自动进行检查。包括：

1检查主码值是否唯一，如果不唯一则拒绝插入或修改

2检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改

检查记录中主码值是否唯一的一种方法是进行全表扫描

索引:为了避免全表扫描， DBMS一般在主码中建立索引，如下图的B+树索引。

参照完整性定义

在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外码

用REFERENCES短语指明这些外码参照哪些表的主码

在表级定义实体完整性

```
  CREATE TABLE SC
    (Sno  CHAR(9) NOT NULL,
    Cno   CHAR(4) NOT NULL,
    Grade  SMALLINT,
    PRIMARY KEY (Sno， Cno)， /在表级定义实体完整性/
```

FOREIGN KEY (Sno) REFERENCES Student(Sno),

/在表级定义参照完整性/

FOREIGN KEY (Cno) REFERENCES Course(Cno)

/在表级定义参照完整性/

);

参照完整性检查和违约处理

被参照表（例如Student）	参照表（例如SC）	违约处理
可能破坏参照完整性	插入元组	拒绝
可能破坏参照完整性	修改外码值	拒绝
删除元组	可能破坏参照完整性	拒绝/级联删除/设置为空值/设置为默认值
修改主码值	可能破坏参照完整性	拒绝/级联修改/设置为空值/设置为默认值

用户定义完整性

属性上的约束条件的定义

CREATE TABLE时定义

列值非空（NOT NULL）

列值唯一（UNIQUE）

检查列值是否满足一个布尔表达式（CHECK）

举例check （列名 in('男','女')）

属性上的约束条件检查和违约处理

插入元组或修改属性的值时，RDBMS检查属性上的约束条件是否被满足，如果不满足则操作被拒绝执行

元组上的约束条件的定义

在CREATE TABLE时可以用CHECK短语定义元组上的约束条件，即元组级的限制

同属性值限制相比，元组级的限制可以设置不同属性之间的取值的相互约束条件

举例

CHECK (Ssex='女' OR Sname **NOT LIKE 'Ms.%')**

*定义了元组中Sname和 Ssex两个属性值之间的约束条件

);

性别是女性的元组都能通过该项检查，因为Ssex='女'成立；

当性别是男性时，要通过检查则名字一定不能以Ms.打头

元组上的约束条件检查和违约处理

插入元组或修改属性的值时，RDBMS检查元组上的约束条件是否被满足

如果不满足则操作被拒绝执行

完整性约束命名子句

添加约束

创建表时候

CREATE TABLE Student

(Sno NUMERIC(6)

CONSTRAINT C1 CHECK (Sno BETWEEN 90000 AND 99999),

Sname CHAR(20)

CONSTRAINT C2 NOT NULL,

...)

在约束条件前添加上约束名，即[constraint <约束名>]，如果没有写则系统自动命名

增加新的约束

alter table <表名> [add constraint]<约束名> <约束条件>

[constraint <约束名>]，如果没有写则系统自动命名

删除约束

alter table <表名> drop constraint<约束名>

触发器

触发器trigger是用户定义在关系表上的一类由事件驱动的特殊过程，由服务器自动激活，可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力

触发器类型

DML触发器：当数据库中发生数据操纵语言(DML)事件时将调用DML触发器。

DDL触发器：当数据库中发生数据定义语言(DDL)事件时将调用DDL触发器，这些语句以CREATE、ALTER、DROP等关键字开头。

我们新学的是dml

创建触发器

定义触发器的语言说明

table|view 可以在表或者视图上定义触发器

with encryption 对创建触发器语句进行加密

after 执行某一操作成功后触发

instead of 触发器表示并不执行所定义操作，而执行触发器本身，即可以在表或者视图中定义instead of 触发器

触发事件 update delete insert

create trigger <触发器名字>

on table|view

[with encryption]

for|after|instead of [delete][,insert][,update]

AS

Sql_statement[...n]

举例

创建触发器，使得student表插入新数据的时候会自动删除。

CREATE TRIGGER s_insert

ON student

AFTER INSERT

AS

DELETE FROM student

WHERE sno in(SELECT sno FROM INSERTED)

deleted表和insert表

被删除表和被插入表

插入的数据在inserted表

删除的数据在deleted表

更新的数据inserted表有新数据，deleted表有旧数据

举例子

在student表中创建一个级联删除触发器，当在student表删除学生信息时，把SC表对应的学生信息删除。

CREATE TRIGGER s_delete

ON student

AFTER DELETE

AS

DELETE FROM SC

WHERE sno in(

  SELECT sno

  FROM DELETED)

错误：先删student表信息，再删SC表，由于先删student表信息会不满足参照完整性，删除不成功，所以触发器不会被触发。.

正确是

```
CREATE TRIGGER s_delete
ON student
INSTEAD OF DELETE
AS
BEGIN
   DELETE FROM SC
   WHERE sno in(  SELECT sno  FROM DELETED)
   DELETE FROM student
   WHERE sno in(SELECT sno FROM DELETED)
END
```

区分语句级触发器和行级触发器

语句级触发器：每个DML语句只触发一次

行级触发器：每行更新操作都会触发一次

例如,假设在TEACHER表上创建了一个AFTER UPDATE触发器。如果表TEACHER有1000行，执行如下语句：

```
 UPDATE TEACHER SET Deptno=5;
```

•如果该触发器为语句级触发器，那么执行完该语句后，触发动作只发生一次

•如果是行级触发器，触发动作将执行1000次

SQL SERVER只有语句级触发器，没有行级触发器

ORACLE有语句级触发器和行级触发器（for each row）

MySQL 只有行级触发器，没有语句级触发器

在sc表中创建一个update触发器，sc表的成绩只能改高不能改低

即执行完所有行的更新或者插入或者删除操作，才执行语句级触发器，被删除表和被插入表有多行数据。则行级触发器是每执行一行的更新插入删除操作，就执行行级触发器，被删除表和被插入表最多只有一行数据。

启用触发器

alter table <表名> enable trigger <触发器名>

默认创建触发器时候已启动

禁用触发器

alter table <表名> disable trigger <触发器名>

删除触发器

drop trigger <触发器名>

变量

全局变量

@变量名

局部变量

声明并创建

declare @变量名 数据类型 set/select @变量名=.....

declare @c char(20)

set @c='Hello world!'

例如

声明一个局部变量，使用SELECT把学号为00001的选课成绩赋予该变量（对比上一题看看结果有什么不同）；

declare @score2 numeric(3, 1)

select/set @score2=(select score from sc where sno=00001)

select @score2

有多个值

但是也不行：子查询返回的值不止一个。当子查询跟随在 =、!=、<、<=、>、>= 之后，或子查询用作表达式时，这种情况是不允许的。

declare @score2 numeric(3, 1)

select @score2=score

from sc where sno=00001

select @score2

只有一个值，最后一个score赋值

注意

局部变量是创建完用完就被系统清除掉。并没有保存在数据库中

转换

convert(所转换的数据类型，值)

goto

goto lv

lv:select * from student where sname='吕小妹'

也就是跳转

case

/case/

select sno=

case sno

when '00001' then '马行空'

when '00002' then '马春花'

when '00003' then '徐铮'

else '商宝震'

end

from student

where sno='00001' or sno='00002' or sno='00003' or sno='00004'

if语句

if [begin end] else [begin end]

while[begin end]

多语句就用begin end相当于{}

判断有！=，=，exists（表）...

标量函数

create function函数名字(@sno char(5)//参数) returns int //返回值

as

begin

函数内容

end

表值函数

内联表值函数

create function函数名字(@sno char(5)//参数) returns table

as 表

多语句表值函数

create function函数名字(@sno char(5)//参数) returns @表名 table（表的格式）

as

begin

insert into @表名

....

return//直接就return

end

数据库结构

关系模式由五部分组成，即它是一个五元组

R（U,D,DOM,F)

R:关系名

U：组成该关系的属性名集合

D：属性组U中属性所来自的域

DOM：属性向域的映像集合

F：属性间数据的依赖关系集合

数据依赖对关系模式的影响

学生的学号（Sno）、所在系（Sdept）

系主任姓名（Mname）、课程名（Cname）

成绩（Grade）

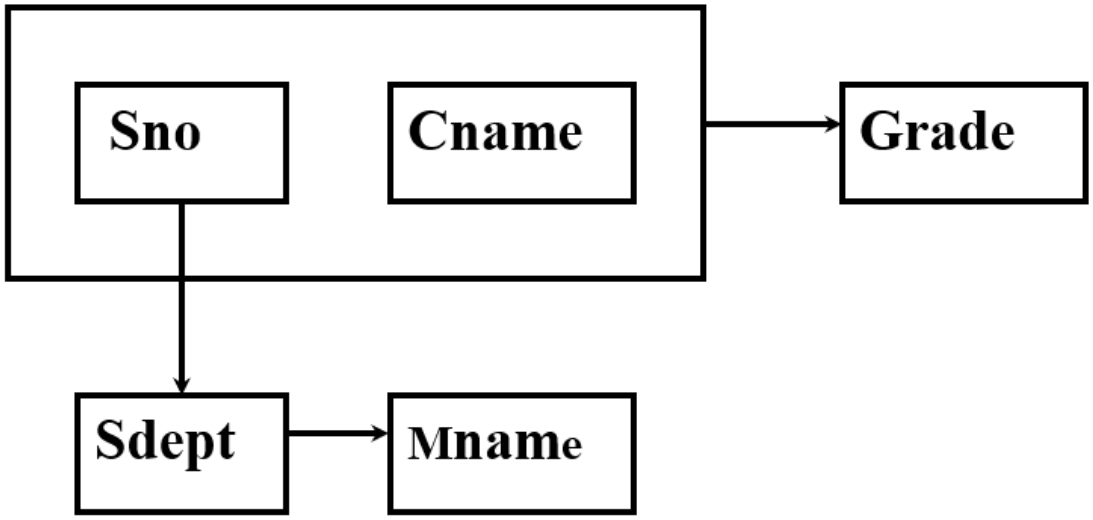
R<U,F>=Student<U,F>

U={*Sno, Sdept, Mname, Cname, Grade* }

F = { Sno → Sdept, Sdept → Mname,

(Sno, Cname) → Grade }

即



学号决定系名，系名决定系主任名字，学号和课程名决定成绩

有以下问题：

数据冗余过大

如所在系以及系主任名字不断重复

更新异常

系主任改变后需要修改多行数据

插入异常

如一个系没有学生，则无法输入该系信息

删除异常

如某个系学生都毕业后，删除该系学生信息的时候需要把系主任信息也删除

解决方法

问题提出：为了好的表数据冗余少，

不存在更新异常插入异常删除异常

原因：由于存在于模式中某些数据依赖引起的

解决：需要分解关系模式消除不合适数据依赖

将Student(Sno, Sdept, Mname, Cname, Grade，Sno → Sdept, Sdept → Mname,

(Sno, Cname) → Grade)

把这个单一模式分成3个关系模式：

S（Sno，Sdept，Sno → Sdept）；

SC（Sno，Cno，Grade，（Sno，Cno）→ Grade）；

DEPT（Sdept，Mname，Sdept→ Mname）

规范化

规范化理论就是用来改造关系模式

X函数确定Y" 或 "Y函数依赖于X"，记作X→Y。（其实就是x确定了，则就能推出y即y=f（x），一个x对应一个y，一个y可对应多个x）

X和Y都可以是多个属性，例如（学号，课程号）→成绩

如果X→Y，但$Y \nsubseteq X$，则称X→Y是非平凡的函数依赖

若X→Y，但$Y \subseteq X$，则称X→Y是平凡的函数依赖

举例

非平凡函数依赖：(Sno, Cno) → Grade

平凡函数依赖：(Sno, Cno) → Sno

（不怎么重要若*X*→*Y*，则*X*称为这个函数依赖的决定属性组，也称为决定因素（Determinant）。

若*X*→*Y*，*Y*→*X*，则记作*X*←→*Y*。

完全函数依赖和部分函数依赖及传递函数依赖

定义**6.2** 在R(U)中，如果X→Y，并且对于X的任何一个真子集X'，都有X' $\not\rightarrow$ Y, 则称Y对X完全函数依赖，记作 $X \xrightarrow{F} Y$。

若X→Y，但Y不完全函数依赖于X，则称Y对X部分函数依赖，记作 $X \xrightarrow{P} Y$。

若*Y*不函数依赖于*X*，则记作*X*→Y。）

[例1] 中(Sno,Cno) $\xrightarrow{F}$ Grade是完全函数依赖，

(Sno,Cno) $\xrightarrow{P}$ Sdept是部分函数依赖

因为Sno →Sdept成立，且Sno是（Sno，Cno）的真子集

定义**6.3** 在R(U)中，如果X→Y，(Y $\not\subseteq$ X) ,Y $\not\rightarrow$ X Y→Z，则称Z对X传递函数依赖。

记为：$X \xrightarrow{传递} Z$

注: 如果Y→X， 即X←→Y，则Z直接依赖于X。

例: 在关系Std(Sno, Sdept, Mname)中，有：

Sno → Sdept，Sdept → Mname

Mname传递函数依赖于Sno

我们平时生活中的逻辑思维都是非平凡的完全函数依赖，

候选码

R<U,F>

K是U的子集或者真子集

K完全函数依赖于U,则K称为R的候选码

（候选码其实就是主键，候选码可以有多个，选一个做主键）

主属性

候选码可能不止一个属性，候选码中的属性称为主属性，不是主属性就是非主属性

全码

整个属性组是码，称为全码，也就是属性中谁都不能决定谁，也就是只有平凡函数依赖，自己决定自己。也就是没有非平方函数依赖。

拆分其实就是利用主键和外键一起提供了表示关系间联系的手段

范式

其实就是对关系模式评级，越高越好

举例：2肯定比1好，2包含1

第一范式(1NF)

第二范式(2NF)

第三范式(3NF)

BC范式(BCNF)

第四范式(4NF)

第五范式(5NF)

举例第二范式包含第一范式

什么是规范化?

一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化

属性不可再分就是第一范式。

第二范式

每一个非主属性完全函数依赖于(候选)码

关系模式 S-L-C(Sno, Sdept, Sloc, Cno, Grade)

Sloc为学生住处，假设每个系的学生住在同一个地方

sno学号sdept系sloc住处cno课程号grade成绩

由非平凡的完全函数依赖

(Sno, Cno) → Grade

Sno → Sdept

Sno → Sloc

Sdept → Sloc

可知道

（sno，cno）就是候选码

主属性是sno，cno

非属性是sdept，sloc，grade

非主属性与(候选)码的函数依赖关系

(Sno, Cno) F Grade　//f是完全函数依赖，p是部分函数依赖

(Sno, Cno) P Sdept

(Sno, Cno) P Sloc

第二范式要求每一个非主属性完全函数依赖于(候选)码

分解

由 (Sno, Cno) F Grade 推出　表一（sno，cno，grade）

(Sno, Cno) P Sdept，(Sno, Cno) P Sloc知道sno f sdept，sno f sloc则表二（sno，sdept，sloc）

拆分其实就是利用主键和外键一起提供了表示关系间联系的手段

这两个表怎么联系起来，sno是表二外键

第三范式

❖ 3NF的定义

定义**6.7** 关系模式*R<U，F>* 中若不存在这样的码*X*、属性组*Y*及非主属性*Z*（$Z \nsubseteq Y$），使得$X \to Y$，$Y \to Z$成立，$Y \not\to X$，则称*R<U，F>* $\in$ *3NF*。

■ 若***R***∈3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码，也就是***R*** $\in$ *2NF*。

每一个非主属性既不部分依赖于码也不传递依赖于码。含有第二范式中每一个非主属性不部分依赖于码也就是每一个非主属性完全函数依赖于(候选)码

由上表二（sno，sdept，sloc）可得

Sno→Sdept

Sdept！→ Sno

Sdept→Sloc

sno传递→sloc

即非主属性对码有传递函数依赖

则分解为表(sno，sdept)和表(sdept,sloc)

满足第三范式的充要条件

若是第三范式则非平凡完全函数依赖X→Y

则X是码或者Y是主属性（满足一个条件就可以）

BC范式

X→Y

X必含有码，也就是每一个决定因素都包含有码

即X是码

而第三范式中X是码或者Y是主属性（满足一个条件就可以）。则

BC包含第三范式

若第三范式中只有一个候选码，则它也是BC范式

多值依赖

全码满足BC范式

它没有两则之间的关系。

X→→Y

X决定了Y一组值

若X→→Y，而Z = φ，(意思就是说X+Y=U，那么肯定X→→Y，没什么意义)则称

X→→Y为平凡的多值依赖

否则称X→→Y为非平凡的多值依赖

对称性

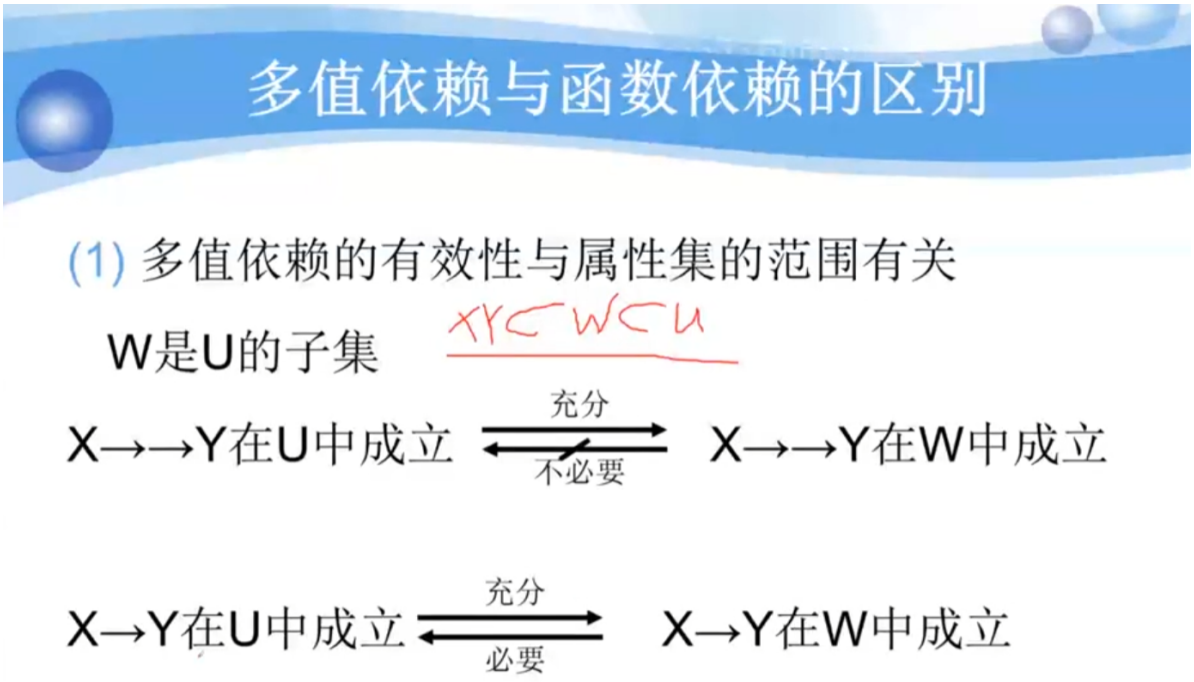
X→→Y，又Z = U - X - Y(剩下的属性组)，则X→→Z

传递性

X→→Y，Y→→Z， 则X→→Z –Y

函数依赖是多值依赖的一种特殊，因为X→Y，一组Y中只有一个

多值依赖和函数依赖的区别



(2)

- 若函数依赖X→Y在R（U）上成立，则对于任何Y' ⊂ Y均有X→Y' 成立
- 多值依赖X→→Y若在R(U)上成立，不能断言对于任何Y' ⊂ Y有X→→Y' 成立

第四范式

非平凡多值依赖X→→Y Y不被X包含，且X含有候选码

X→Y可推出X→→Y，X→→Y推不出X→Y(X→Y是特殊的多值依赖，被包含关系)

若满足第四范式X→→Y Y不被X包含，且X含有候选码，又X→Y可推出X→→Y。则

BC范式中X→Y，X含有候选码。则满足BC范式

举例

表（C,T,B）

C→→T

C→→B

又CTB为全码

存在非平凡的多值依赖C→→T，且C不是码

可拆分为

CT(C, T) ∈ 4NF

CB(C, B) ∈ 4NF

C→→T， C→→B是平凡多值依赖，不存在非平凡多值依赖，所以不用考虑CT候选码中 C→→T,C不是候选码。所以它们是第四范式

第二范式

每一个非主属性完全函数依赖于(候选)码

关系模式 S-L-C(Sno, Sdept, Sloc, Cno, Grade)

Sloc为学生住处，假设每个系的学生住在同一个地方

sno学号sdept系sloc住处cno课程号grade成绩

由非平凡的完全函数依赖

(Sno, Cno) → Grade

Sno → Sdept

Sno → Sloc

Sdept → Sloc

可知道

（sno，cno）就是候选码

主属性是sno，cno

非属性是sdept，sloc，grade

非主属性与(候选)码的函数依赖关系

(Sno, Cno) F Grade　　//f是完全函数依赖，p是部分函数依赖

(Sno, Cno) P Sdept

(Sno, Cno) P Sloc

第二范式要求每一个非主属性完全函数依赖于(候选)码

分解

由 (Sno, Cno) F Grade 推出　表一（sno，cno，grade）

(Sno, Cno) P Sdept，(Sno, Cno) P Sloc知道sno f sdept，sno f sloc则表二（sno，sdept，sloc）

拆分其实就是利用主键和外键一起提供了表示关系间联系的手段

这两个表怎么联系起来，sno是表二外键

第三范式

❖3NF的定义

定义**6.7** 关系模式$R<U，F>$中若不存在这样的码X、属性组Y及非主属性Z（$Z \nsubseteq Y$），使得$X \to Y$，$Y \to Z$成立，$Y \not\to X$，则称$R<U，F> \in 3NF$。

- 若$R \in 3NF$，则每一个非主属性既不部分依赖于码也不传递依赖于码，也就是$R \in 2NF$。

每一个非主属性既不部分依赖于码也不传递依赖于码。含有第二范式中每一个非主属性不部分依赖于码也就是每一个非主属性完全函数依赖于(候选)码

由上表二（sno，sdept，sloc）可得

Sno→Sdept

Sdept！→ Sno

Sdept→Sloc

sno传递→sloc

即非主属性对码有传递函数依赖

则分解为表(sno，sdept)和表(sdept,sloc)

满足第三范式的充要条件

若是第三范式则非平凡完全函数依赖X→Y

则X是码或者Y是主属性（满足一个条件就可以）

BC范式

X→Y

X必含有码，也就是每一个决定因素都包含有码

即X是码

而第三范式中X是码或者Y是主属性（满足一个条件就可以）。则

BC包含第三范式

若第三范式中只有一个候选码，则它也是BC范式

多值依赖

全码满足BC范式

它没有两则之间的关系。

X→→Y

X决定了Y一组值

若X→→Y，而Z = φ，(意思就是说X+Y=U，那么肯定X→→Y，没什么意义)则称

X→→Y为平凡的多值依赖

否则称X→→Y为非平凡的多值依赖

对称性

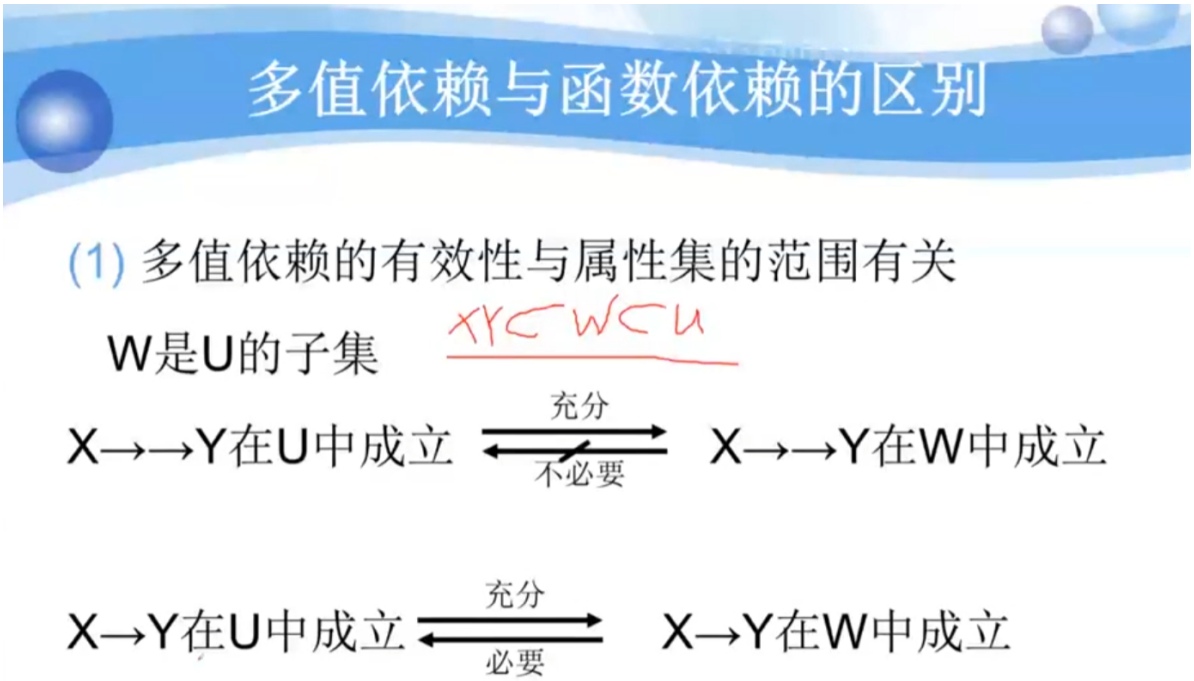
X→→Y，又Z = U - X - Y(剩下的属性组)，则X→→Z

传递性

X→→Y，Y→→Z， 则X→→Z -Y

函数依赖是多值依赖的一种特殊，因为X→Y，一组Y中只有一个

多值依赖和函数依赖的区别



多值依赖与函数依赖的区别

(2)

- 若函数依赖X→Y在R（U）上成立，则对于任何Y' ⊂ Y均有X→Y' 成立
- 多值依赖X→→Y若在R(U)上成立，不能断言对于任何Y' ⊂ Y有X→→Y' 成立

第四范式

非平凡多值依赖X→→Y Y不被X包含，且X含有候选码

X→Y可推出X→→Y，X→→Y推不出X→Y(X→Y是特殊的多值依赖，被包含关系)

若满足第四范式X→→Y Y不被X包含，且X含有候选码，又X→Y可推出X→→Y。则

BC范式中X→Y，X含有候选码。则满足BC范式

举例

表（C,T,B）

C→→T

C→→B

又CTB为全码

存在非平凡的多值依赖C→→T，且C不是码

可拆分为

CT(C, T) ∈ 4NF

CB(C, B) ∈ 4NF

C→→T， C→→B是平凡多值依赖，不存在非平凡多值依赖，所以不用考虑CT候选码中 C→→T,C不是候选码。所以它们是第四范式

判断

注意要

is null

is not null

不要！=null，=null。相当于废话，反正就是错的

<>就是不等于

！=

游标

过程

声明游标

declare 游标名 [属性] cursor

打开游标

open 游标名

关闭贯标

close 游标名

解除游标

deallocate 游标名

属性

游标属性有scroll：可前滚，后滚。insensitive，静态游标，对表改变不会影响游标的值。打开游标时建立在tempdb系统数据库中

没有属性则是只进游标，只能不断下一行

区域

游标区域

declare 游标名 [属性] cursor

for

select-from-where语句

[for read only]　　//不可修改表中内容

移动

游标移动

首个

fetch first from 游标名

后一个

fetch next from 游标名

前一个

fetch prior from 游标名

第n行

fetch absolute n from 游标名

相对于当前游标的第n行

fetch relative n from 游标名

赋值

游标移动且赋值

fetch first from 游标名 into 变量名，变量名

这些变量名对应select语句中属性值

游标全局变量

@@cursor_rows 游标总行数

@@fetch_status 游标状态

0 fetch语句成功，-1 fetch语句失败或此行不在结果集中

-2被提取的的行不存在

fetch next from 游标名 ， 当最后一行不断next依旧是最后一行

游标更新数据

update 表名

set 属性值=

where current of 游标名

armstrong公理系统

自反性

Y属于X，则X→Y

增广性

Z属于U，X→Y，则XZ→YZ

传递性

X→Y,且Y→Z。则X→Z

规则

合并规则，

X→Y,X→Z

X→YZ

A2A3

伪传递规则

X→Y,WY→Z

WX→Z

A2A3

分解规则

X→Y,Z属于Y

X→Z

A1A3

X→*A*1**A**2**…A**k成立的充分必要条件是*X*→*A*i成立（i=*l*，*2*，…，k*）

F的闭包

F={X→Y, Y→Z}

F⁺={

X→φ, Y→φ, Z→φ, XY→φ, XZ→φ, YZ→φ, XYZ→φ,
X→X, Y→Y, Z→Z, XY→X, XZ→X, YZ→Y, XYZ→X,
X→Y, Y →Z, XY→Y, XZ→Y, YZ→Z, XYZ→Y,
X→Z, Y→YZ, XY→Z, XZ→Z, YZ→YZ,XYZ→Z,
X→XY, XY→XY,XZ→XY, XYZ→XY,
X→XZ, XY→YZ,XZ→XZ, XYZ→YZ,
X→YZ, XY→XZ,XZ→XY, XYZ→XZ,
X→ZYZ, XY→XYZ,XZ→XYZ, XYZ→XYZ }

F={X→A1, …… , X→An}的闭包*F*⁺计算是一个NP完全问题

函数依赖闭包

设*F*为属性集*U*上的一组函数依赖，*X* Í*U*， *X*F+ ={ *A* | *X*→*A*能由*F* 根据Armstrong公理导出}，*X*F+称为属性集*X*关于函数依赖集*F* 的闭包

函数依赖闭包(续)

[例1] 已知关系模式$R<U，F>$，其中

$U=\{A，B，C，D，E\}$;

$F=\{AB\rightarrow C，B\rightarrow D，C\rightarrow E，EC\rightarrow B，AC\rightarrow B\}$。

求$（AB）_F^+$。

求解：$（AB）_F^+ =ABCDE$。

最小依赖集

每一个函数依赖集F均等价于一个极小函数依赖集Fm。此Fm称为F的最小依赖集。

极小化过程（续）

例 设有关系模式R(U,F),其中U=ABC，F={A → BC,B → C,A → B,AB → C}，求F_{min}

（1）将F中所有函数依赖的依赖因素写成单属性集形式：

G= {A → B, A → C,B → C,A → B,AB → C}

删除一个A → B，有G= {A → B, A → C,B → C,AB → C}

（2）在G的每一个函数依赖中消除决定因素中的冗余属性：

AB → C可以从A → C中推导出，冗余，删除，则 G= {A → B, A → C,B →C}

（3）在G中消除冗余的函数依赖：

A → C 可以从A → B, B →C中推导出，函数依赖冗余，删除，则 G= {A → B, B →C}

单属性

冗余属性

冗余的函数依赖

存储过程

由于存储过程不像解释执行的SQL语句那样在提出操作请求时才进行语法分析和优化，因而运行效率高，它提供了在服务器端快速执行SQL语句的有效途径。降低了客户机和服务器之间的通信量

创建存储

create procedure 存储名 [局部变量] [输出的局部变量]

as

内容

例如

create procedure test1

as

select student.sno,sname,Score

from student,sc

where student.sno=sc.sno and student.sno=001

create procedure test2 @sno char(5)

create procedure test4 @name varchar(40),@number int output

执行存储过程

execute 存储名

带有参数的存储

excute 存储名 参数值..

带有输出的存储

declare @变量名 数据类型

excute 存储名 参数值，@变量名 output

//会给该局部变量赋值

例如：execute test1

execute test2 '00004'

declare @number int

execute test4 '数据库',@number output

删除存储过程

drop procedure 存储名