一、基础

1、说明：创建数据库

CREATE DATABASE database-name

2、说明：删除数据库

drop database dbname3、说明：备份 sql server

--- 创建 备份数据的 device

USE master

EXEC sp_addumpdevice 'disk', 'testBack',

'c:\mssql7backup\MyNwind_1.dat'

--- 开始 备份

BACKUP DATABASE pubs TO testBack

4、说明：创建新表

create table tabname(col1 type1 [not null] [primary key],col2 type2 [not null],..)

根据已有的表创建新表：

A：create table tab_new like tab_old (使用旧表创建新表)

B：create table tab_new as select col1,col2… from tab_old definition only5、说明：删除新表

drop table tabname

6、说明：增加一个列

Alter table tabname add column col type 注：列增加后将不能删除。DB2 中列加上后数据类型也不能改变，唯一能改变的是增加 varchar 类型的长度。

7、说明：添加主键： Alter table tabname add primary key(col)

说明：删除主键： **Alter table tabname drop primary key(col) 8、说明：创建索引：**

create [unique] index idxname on tabname(col….)

删除索引：**drop index idxname**

注：索引是不可更改的，想更改必须删除重新建。

9、说明：创建视图：create view viewname as select statement

删除视图：drop view viewname

10、说明：几个简单的基本的 sql 语句

选择：select * from table1 where 范围

插入：insert into table1(field1,field2) values(value1,value2)

删除：delete from table1 where 范围**更新：**update table1 set field1=value1 where 范围

查找：select * from table1 where field1 like ’%value1%’ ---like 的语法很精妙，查资料!

排序：select * from table1 order by field1,field2 [desc]

总数：select count as totalcount from table1

求和：select sum(field1) as sumvalue from table1

平均：select avg(field1) as avgvalue from table1

最大：select max(field1) as maxvalue from table1

最小：select min(field1) as minvalue from table1

11、说明：几个高级查询运算词

A： UNION 运算符

UNION 运算符通过组合其他两个结果表（例如 TABLE1 和 TABLE2）并消去表中任何重复行而派生出一个结果表。当 ALL 随 UNION 一起使用时（即 UNION ALL），不消除重复行。两种情况下，派生表的每一行不是来自 TABLE1 就是来自 TABLE2。

B： EXCEPT 运算符

EXCEPT 运算符通过包括所有在 TABLE1 中但不在 TABLE2 中的行并消除所有重复行而派生出一个结果表。当 ALL 随 EXCEPT 一起使用时 (EXCEPT ALL)，不消除重复行。

C： INTERSECT 运算符

INTERSECT 运算符通过只包括 TABLE1 和 TABLE2 中都有的行并消除所有重复行而派生出一个结果表。当 **ALL** 随 INTERSECT 一起使用时 (INTERSECT ALL)，不消除重复行。

注：使用运算词的几个查询结果行必须是一致的。

12、说明：使用外连接

A、**left （outer） join**：

左外连接（左连接）：结果集几包括连接表的匹配行，也包括左连接表的所有行。

SQL: select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

B：right （outer） join:

右外连接(右连接)：结果集既包括连接表的匹配连接行，也包括右连接表的所有行。

C：full/cross （outer） join：

全外连接：不仅包括符号连接表的匹配行，还包括两个连接表中的所有记录。

12、分组:Group by:

一张表，一旦分组 完成后，查询后只能得到组相关的信息。

组相关的信息：（统计信息） count,sum,max,min,avg 分组的标准)

在 **SQLServer** 中分组时：不能以 **text,ntext,image** 类型的字段作为分组依据

在 **selecte** 统计函数中的字段，不能和普通的字段放在一起；

13、对数据库进行操作：

分离数据库： **sp_detach_db;**附加数据库：**sp_attach_db** 后接表明，附加需要完整的路径名

14.如何修改数据库的名称:

sp_renamedb 'old_name', 'new_name'

二、提升

1、说明：复制表(只复制结构,源表名：a 新表名：b) (Access 可用)

法一：select * into b from a where 1<>1（仅用于 SQlServer）法二：select top 0 * into b from a

2、说明：拷贝表(拷贝数据,源表名：a 目标表名：b) (Access 可用)

insert into b(a, b, c) select d,e,f from b;

3、说明：跨数据库之间表的拷贝(具体数据使用绝对路径) (Access 可用)
insert into b(a, b, c) select d,e,f from b in '具体数据库' where 条件
例子：..from b in '"&Server.MapPath(".")&"\data.mdb" &"' where..

4、说明：子查询(表名 1：a 表名 2：b)
select a,b,c from a where a IN (select d from b) 或者: select a,b,c from a where a IN (1,2,3)

5、说明：显示文章、提交人和最后回复时间
select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b

6、说明：外连接查询(表名 1：a 表名 2：b)
select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

7、说明：在线视图查询(表名 1：a)
select * from (SELECT a,b,c FROM a) T where t.a > 1;

8、说明：between 的用法,between 限制查询数据范围时包括了边界值,not between 不包括
select * from table1 where time between time1 and time2
select a,b,c, from table1 where a not between 数值 1 and 数值 2

9、说明：in 的使用方法
select * from table1 where a [not] in ('值 1','值 2','值 4','值 6')

10、说明：两张关联表，删除主表中已经在副表中没有的信息
delete from table1 where not exists (select * from table2 where table1.field1=table2.field1)

11、说明：四表联查问题：
select * from a left inner join b on a.a=b.b right inner join c on a.a=c.c inner join d on a.a=d.d where .....

12、说明：日程安排提前五分钟提醒
SQL: select * from 日程安排 where datediff('minute',f 开始时间,getdate())>5

13、说明：一条 sql 语句搞定数据库分页 select top 10 b.* from (select top 20 主键字段,排序字段 from 表名 order by 排序字段 desc) a,表名 b where b.主键字段 = a.主键字段 order by a.排序字段**具体实现：**关于数据库分页：

```
declare @start int,@end int

@sql    nvarchar(600)
```

```
    set @sql=' select top' +str(@end-@start+1)+' +from T where rid not
in(select top' +str(@str-1)+' Rid from T where Rid>-1)'

    exec sp_executesql @sql
```

注意：在 top 后不能直接跟一个变量，所以在实际应用中只有这样的进行特殊的处理。Rid 为一个标识列，如果 top 后还有具体的字段，这样做是非常有好处的。因为这样可以避免 top 的字段如果是逻辑索引的，查询的结果后实际表中的不一致（逻辑索引中的数据有可能和数据表中的不一致，而查询时如果处在索引则首先查询索引）

14、说明：前 10 条记录

```
select top 10 * form table1 where 范围
```

15、说明：选择在每一组 b 值相同的数据中对应的 a 最大的记录的所有信息(类似这样的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)

```
select a,b,c from tablename ta where a=(select max(a) from tablename tb where
tb.b=ta.b)
```

16、说明：包括所有在 TableA 中但不在 TableB 和 TableC 中的行并消除所有重复行而派生出一个结果表

```
(select a from tableA ) except (select a from tableB) except (select a from tableC)
```

17、说明：随机取出 10 条数据

```
select top 10 * from tablename order by newid()
```

18、说明：随机选择记录

```
select newid()
```

19、说明：删除重复记录

1),

```
delete from tablename where id not in (select max(id) from tablename group by
col1,col2,...)
```

2),

```
select distinct * into temp from tablename
 delete from tablename
 insert into tablename select * from temp
```

评价： 这种操作牵连大量的数据的移动，这种做法不适合大容量但数据操作 **3),**例如：在一个外部表中导入数据，由于某些原因第一次只导入了一部分，但很难判断具体位置，这样只有在下一次全部导入，这样也就产生好多重复的字段，怎样删除重复字段

```
alter table tablename
--添加一个自增列
add  column_b int identity(1,1)
 delete from tablename where column_b not in(
```

select max(column_b) from tablename group by column1,column2,...)
alter table tablename drop column column_b

20、说明：列出数据库里所有的表名
select name from sysobjects where type='U' // U 代表用户

21、说明：列出表里的所有的列名
select name from syscolumns where id=object_id('TableName')

22、说明：列示 type、vender、pcs 字段，以 type 字段排列，case 可以方便地实现多重选择，类似 select 中的 case。
select type,sum(case vender when 'A' then pcs else 0 end),sum(case vender when 'C' then pcs else 0 end),sum(case vender when 'B' then pcs else 0 end) FROM tablename group by type
显示结果：
type vender pcs
电脑 A 1
电脑 A 1
光盘 B 2
光盘 A 2
手机 B 3
手机 C 3

23、说明：初始化表 table1

TRUNCATE TABLE table1

24、说明：选择从 10 到 15 的记录
select top 5 * from (select top 15 * from table order by id asc) table_别名 order by id desc

三、技巧

1、1=1，1=2 的使用，在 SQL 语句组合时用的较多

"where 1=1" 是表示选择全部 "where 1=2"全部不选，
如：
```
if @strWhere !=''
begin
set @strSQL = 'select count(*) as Total from [' + @tblName + '] where ' +
@strWhere
end
else
```

```
begin
set @strSQL = 'select count(*) as Total from [' + @tblName + ']'
end
```

我们可以直接写成

错误！未找到目录项。

set @strSQL = 'select count(*) as Total from [' + @tblName + '] where 1=1 安定 '+ @strWhere 2、收缩数据库

```
--重建索引
DBCC REINDEX
DBCC INDEXDEFRAG
--收缩数据和日志
DBCC SHRINKDB
DBCC SHRINKFILE
```

3、压缩数据库

```
dbcc shrinkdatabase(dbname)
```

4、转移数据库给新用户以已存在用户权限

```
exec sp_change_users_login 'update_one','newname','oldname'
go
```

5、检查备份集

```
RESTORE VERIFYONLY from disk='E:\dvbbs.bak'
```

6、修复数据库

```
ALTER DATABASE [dvbbs] SET SINGLE_USER
GO
DBCC CHECKDB('dvbbs',repair_allow_data_loss) WITH TABLOCK
GO
ALTER DATABASE [dvbbs] SET MULTI_USER
GO
```

7、日志清除

```
SET NOCOUNT ON
DECLARE @LogicalFileName sysname,
 @MaxMinutes INT,
 @NewSize INT


USE tablename -- 要操作的数据库名
SELECT  @LogicalFileName = 'tablename_log', -- 日志文件名
```

```
@MaxMinutes = 10, -- Limit on time allowed to wrap log.
 @NewSize = 1  -- 你想设定的日志文件的大小(M)

Setup / initialize
DECLARE @OriginalSize int
SELECT @OriginalSize = size
 FROM sysfiles
 WHERE name = @LogicalFileName
SELECT 'Original Size of ' + db_name() + ' LOG is ' +
 CONVERT(VARCHAR(30),@OriginalSize) + ' 8K pages or ' +
 CONVERT(VARCHAR(30),(@OriginalSize*8/1024)) + 'MB'
 FROM sysfiles
 WHERE name = @LogicalFileName
CREATE TABLE DummyTrans
 (DummyColumn char (8000) not null)


DECLARE @Counter    INT,
 @StartTime DATETIME,
 @TruncLog   VARCHAR(255)
SELECT @StartTime = GETDATE(),
 @TruncLog = 'BACKUP LOG ' + db_name() + ' WITH TRUNCATE_ONLY'

DBCC SHRINKFILE (@LogicalFileName, @NewSize)
EXEC (@TruncLog)
-- Wrap the log if necessary.
WHILE @MaxMinutes > DATEDIFF (mi, @StartTime, GETDATE()) -- time has not
expired
 AND @OriginalSize = (SELECT size FROM sysfiles WHERE name =
@LogicalFileName)
 AND (@OriginalSize * 8 /1024) > @NewSize
 BEGIN -- Outer loop.
SELECT @Counter = 0
 WHILE   ((@Counter < @OriginalSize / 16) AND (@Counter < 50000))
 BEGIN -- update
 INSERT DummyTrans VALUES ('Fill Log') DELETE DummyTrans
 SELECT @Counter = @Counter + 1
 END
 EXEC (@TruncLog)
 END
SELECT 'Final Size of ' + db_name() + ' LOG is ' +
 CONVERT(VARCHAR(30),size) + ' 8K pages or ' +
 CONVERT(VARCHAR(30),(size*8/1024)) + 'MB'
 FROM sysfiles
```

```
 WHERE name = @LogicalFileName
DROP TABLE DummyTrans
SET NOCOUNT OFF
```

8、说明：更改某个表

```
exec sp_changeobjectowner 'tablename','dbo'
```

9、存储更改全部表

```
CREATE PROCEDURE dbo.User_ChangeObjectOwnerBatch
@OldOwner as NVARCHAR(128),
@NewOwner as NVARCHAR(128)
AS

DECLARE @Name     as NVARCHAR(128)
DECLARE @Owner    as NVARCHAR(128)
DECLARE @OwnerName    as NVARCHAR(128)

DECLARE curObject CURSOR FOR
select 'Name'     = name,
   'Owner'     = user_name(uid)
from sysobjects
where user_name(uid)=@OldOwner
order by name

OPEN    curObject
FETCH NEXT FROM curObject INTO @Name, @Owner
WHILE(@@FETCH_STATUS=0)
BEGIN
if @Owner=@OldOwner
begin
   set @OwnerName = @OldOwner + '.' + rtrim(@Name)
   exec sp_changeobjectowner @OwnerName, @NewOwner
end
-- select @name,@NewOwner,@OldOwner

FETCH NEXT FROM curObject INTO @Name, @Owner
END

close curObject
deallocate curObject
GO
```

10、SQL SERVER 中直接循环写入数据

```
declare @i int
set @i=1
while @i<30
begin
    insert into test (userid) values(@i)
    set @i=@i+1
end
```

案例：

有如下表，要求就裱中所有沒有及格的成績，在每次增長 **0.1** 的基礎上，使他們剛好及格：

```
   Name     score
   Zhangshan   80
   Lishi       59
   Wangwu      50
   Songquan    69
```

```
while((select min(score) from tb_table)<60)
begin
update tb_table set score =score*1.01
where score<60
if  (select min(score) from tb_table)>60
 break
 else
   continue
```

end

数据开发-经典

1.按姓氏笔画排序:

```
Select * From TableName Order By CustomerName Collate
Chinese_PRC_Stroke_ci_as //从少到多
```

2.数据库加密:select encrypt('原始密码')

```
select pwdencrypt('原始密码')
select pwdcompare('原始密码','加密后密码') = 1--相同；否则不相同 encrypt('原始密码
')
select pwdencrypt('原始密码')
select pwdcompare('原始密码','加密后密码') = 1--相同；否则不相同
```

3.取回表中字段:

```
declare @list varchar(1000),
@sql nvarchar(1000)
select @list=@list+','+b.name from sysobjects a,syscolumns b where a.id=b.id
and a.name='表 A'
set @sql='select '+right(@list,len(@list)-1)+' from 表 A'
exec (@sql)
```

4.查看硬盘分区:

```
EXEC master..xp_fixeddrives
```

5.比较 A,B 表是否相等:

```
if (select checksum_agg(binary_checksum(*)) from A)
     =
    (select checksum_agg(binary_checksum(*)) from B)
print '相等'
else
print '不相等'
```

6.杀掉所有的事件探察器进程:

```
DECLARE hcforeach CURSOR GLOBAL FOR SELECT 'kill '+RTRIM(spid) FROM
master.dbo.sysprocesses
WHERE program_name IN('SQL profiler',N'SQL 事件探查器')
EXEC sp_msforeach_worker '?'
```

7.记录搜索:

开头到 **N** 条记录 Select Top N * From 表----------

N 到 M 条记录(要有主索引 ID)

Select Top M-N * From 表 Where ID in (Select Top M ID From 表) Order by ID Desc

---

N 到结尾记录 Select Top N * From 表 Order by ID Desc

案例例如 1:一张表有一万多条记录,表的第一个字段 RecID 是自增长字段，写一个 SQL 语句，找出表的第 31 到第 40 个记录。

```
select top 10 recid from A where recid not in(select top 30 recid from A)
```

分析：如果这样写会产生某些问题，如果 recid 在表中存在逻辑索引。

select top 10 recid from A where……是从索引中查找，而后面的 select top 30 recid from A 则在数据表中查找，这样由于索引中的顺序有可能和数据表中的不一致，这样就导致查询到的不是本来的欲得到的数据。

解决方案

1，用 order by select top 30 recid from A order by ricid 如果该字段不是自增长，就会出现问题

2，在那个子查询中也加条件：select top 30 recid from A where recid>-1

例 2：查询表中的最后以条记录，并不知道这个表共有多少数据,以及表结构。

```
set @s = 'select top 1 * from T where pid not in (select top ' + str(@count-1) + ' pid from T)'

print @s exec sp_executesql @s
```

9：获取当前数据库中的所有用户表

```
select Name from sysobjects where xtype='u' and status>=0
```

10：获取某一个表的所有字段

```
select name from syscolumns where id=object_id('表名')

select name from syscolumns where id in (select id from sysobjects where type = 'u' and name = '表名')
```

两种方式的效果相同

11：查看与某一个表相关的视图、存储过程、函数 select a.*

```
from sysobjects a, syscomments b where a.id = b.id and b.text like '%表名%'
```

12：查看当前数据库中所有存储过程

select name as 存储过程名称 from sysobjects where xtype='P'

13：查询用户创建的所有数据库 select * from master..sysdatabases D where sid not in(select sid from master..syslogins where name='sa')

或者

select dbid, name AS DB_NAME from master..sysdatabases where sid <> 0x01

14：查询某一个表的字段和数据类型

select column_name,data_type from information_schema.columns
where table_name = '表名'

15：不同服务器数据库之间的数据操作

--创建链接服务器

exec sp_addlinkedserver 'ITSV ', ' ', 'SQLOLEDB ', '远程服务器名或 ip 地址 '

exec sp_addlinkedsrvlogin 'ITSV ', 'false ',null, '用户名 ', '密码 '

--查询示例

select * from ITSV.数据库名.dbo.表名

--导入示例

select * into 表 from ITSV.数据库名.dbo.表名

--以后不再使用时删除链接服务器

exec sp_dropserver 'ITSV ', 'droplogins '

--连接远程/局域网数据(openrowset/openquery/opendatasource)

--1、openrowset

--查询示例

select * from openrowset('SQLOLEDB ', 'sql 服务器名 '; '用户名 '; '密码 ',数据库名.dbo.表名)

--生成本地表

```
select * into 表 from openrowset( 'SQLOLEDB ', 'sql 服务器名 '; '用户名 '; '密码 ',数据库名.dbo.表名)


--把本地表导入远程表

insert openrowset( 'SQLOLEDB ', 'sql 服务器名 '; '用户名 '; '密码 ',数据库名.dbo.表名)

select *from 本地表

--更新本地表

update b

set b.列 A=a.列 A

 from openrowset( 'SQLOLEDB ', 'sql 服务器名 '; '用户名 '; '密码 ',数据库名.dbo.表名)as a inner join 本地表 b

on a.column1=b.column1

--openquery 用法需要创建一个连接

--首先创建一个连接创建链接服务器

exec sp_addlinkedserver   'ITSV ', ' ', 'SQLOLEDB ', '远程服务器名或 ip 地址 '

--查询

select *

FROM openquery(ITSV,  'SELECT *  FROM 数据库.dbo.表名 ')

--把本地表导入远程表

insert openquery(ITSV,  'SELECT *  FROM 数据库.dbo.表名 ')

select * from 本地表

--更新本地表

update b
```

```
set b.列 B=a.列 B

FROM openquery(ITSV, 'SELECT * FROM 数据库.dbo.表名 ') as a

inner join 本地表 b on a.列 A=b.列 A



--3、opendatasource/openrowset

SELECT  *

FROM  opendatasource( 'SQLOLEDB ', 'Data Source=ip/ServerName;User ID=登陆名;Password=密码 ' ).test.dbo.roy_ta

--把本地表导入远程表

insert opendatasource( 'SQLOLEDB ', 'Data Source=ip/ServerName;User ID=登陆名;Password=密码 ').数据库.dbo.表名

select * from 本地表
```

SQL Server 基本函数

SQL Server 基本函数

1.字符串函数 长度与分析用

1,datalength(Char_expr) 返回字符串包含字符数,但不包含后面的空格

2,substring(expression,start,length) 取子串，字符串的下标是从"1"，start 为起始位置，length 为字符串长度，实际应用中以 len(expression)取得其长度

3,right(char_expr,int_expr) 返回字符串右边第 int_expr 个字符，还用 **left** 于之相反

4,isnull(*check_expression* , *replacement_value*)如果 check_expression 為空，則返回 replacement_value 的值，不為空，就返回 check_expression 字符操作类

5,Sp_addtype 自定義數據類型
例如：EXEC sp_addtype birthday, datetime, 'NULL'

6,set nocount {on|off}

使返回的结果中不包含有关受 Transact-SQL 语句影响的行数的信息。如果存储过程中包含的一些语句并不返回许多实际的数据，则该设置由于大量减少了网络流量，因此可显著提高性能。

SET NOCOUNT 设置是在执行或运行时设置，而不是在分析时设置。

SET NOCOUNT 为 ON 时，不返回计数（表示受 Transact-SQL 语句影响的行数）。

SET NOCOUNT 为 OFF 时，返回计数

常识

在 SQL 查询中：from 后最多可以跟多少张表或视图： 256

在 SQL 语句中出现 Order by,查询时，先排序，后取

在 SQL 中，一个字段的最大容量是 8000，而对于 nvarchar(4000),由于 nvarchar 是 Unicode 码。

SQLServer2000 同步复制技术实现步骤

一、 预备工作

1.发布服务器,订阅服务器都创建一个同名的 windows 用户,并设置相同的密码,做为发布快照文件夹的有效访问用户

--管理工具

--计算机管理

--用户和组

--右键用户

--新建用户

--建立一个隶属于 administrator 组的登陆 windows 的用户（SynUser）

2.在发布服务器上,新建一个共享目录,做为发布的快照文件的存放目录,操作:

我的电脑--D:\ 新建一个目录,名为: PUB

--右键这个新建的目录

--属性--共享

--选择"共享该文件夹"

--通过"权限"按纽来设置具体的用户权限,保证第一步中创建的用户(SynUser) 具有对该文件夹的所有权限

--确定

3.设置 SQL 代理(SQLSERVERAGENT)服务的启动用户(发布/订阅服务器均做此设置)

开始--程序--管理工具--服务

--右键 SQLSERVERAGENT

--属性--登陆--选择"此账户"

--输入或者选择第一步中创建的 windows 登录用户名（SynUser）

--"密码"中输入该用户的密码

4.设置 SQL Server 身份验证模式,解决连接时的权限问题(发布/订阅服务器均做此设置)

企业管理器

--右键 SQL 实例--属性
--安全性--身份验证
--选择"SQL Server 和 Windows"
--确定
5.在发布服务器和订阅服务器上互相注册
企业管理器
--右键 SQL Server 组
--新建 SQL Server 注册...
--下一步--可用的服务器中,输入你要注册的远程服务器名 --添加
--下一步--连接使用,选择第二个"SQL Server 身份验证"
--下一步--输入用户名和密码（SynUser）
--下一步--选择 SQL Server 组,也可以创建一个新组
--下一步--完成
6.对于只能用 IP,不能用计算机名的,为其注册服务器别名(此步在实施中没用到）
(在连接端配置,比如,在订阅服务器上配置的话,服务器名称中输入的是发布服务器的 IP)
开始--程序--Microsoft SQL Server--客户端网络实用工具
--别名--添加
--网络库选择"tcp/ip"--服务器别名输入 SQL 服务器名
--连接参数--服务器名称中输入 SQL 服务器 ip 地址
--如果你修改了 SQL 的端口,取消选择"动态决定端口",并输入对应的端口号
二、 正式配置
1、配置发布服务器
打开企业管理器，在发布服务器（B、C、D）上执行以下步骤:
(1) 从[工具]下拉菜单的[复制]子菜单中选择[配置发布、订阅服务器和分发]
出现配置发布和分发向导
(2)［下一步］选择分发服务器 可以选择把发布服务器自己作为分发服务器或者其他 sql 的服务器（选择自己）
(3)［下一步］设置快照文件夹
采用默认\\servername\Pub
(4)［下一步］自定义配置
可以选择:是,让我设置分发数据库属性启用发布服务器或设置发布设置
否,使用下列默认设置（推荐）
(5)［下一步］设置分发数据库名称和位置 采用默认值
(6)［下一步］启用发布服务器 选择作为发布的服务器
(7)［下一步］选择需要发布的数据库和发布类型
(8)［下一步］选择注册订阅服务器
(9)［下一步］完成配置
2、创建出版物
发布服务器 B、C、D 上
(1)从[工具]菜单的[复制]子菜单中选择[创建和管理发布]命令
(2)选择要创建出版物的数据库，然后单击[创建发布]

(3)在[创建发布向导]的提示对话框中单击[下一步]系统就会弹出一个对话框。对话框上的内容是复制的三个类型。我们现在选第一个也就是默认的快照发布(其他两个大家可以去看看帮助)
(4)单击[下一步]系统要求指定可以订阅该发布的数据库服务器类型,
SQLSERVER 允许在不同的数据库如 orACLE 或 ACCESS 之间进行数据复制。
但是在这里我们选择运行"SQL SERVER 2000"的数据库服务器
(5)单击[下一步]系统就弹出一个定义文章的对话框也就是选择要出版的表
注意: 如果前面选择了事务发布 则再这一步中只能选择带有主键的表
(6)选择发布名称和描述
(7)自定义发布属性 向导提供的选择:
是 我将自定义数据筛选,启用匿名订阅和或其他自定义属性
否 根据指定方式创建发布 (建议采用自定义的方式)
(8)[下一步] 选择筛选发布的方式
(9)[下一步] 可以选择是否允许匿名订阅
1)如果选择署名订阅,则需要在发布服务器上添加订阅服务器
方法: [工具]->[复制]->[配置发布、订阅服务器和分发的属性]->[订阅服务器] 中添加
否则在订阅服务器上请求订阅时会出现的提示:改发布不允许匿名订阅
如果仍然需要匿名订阅则用以下解决办法
[企业管理器]->[复制]->[发布内容]->[属性]->[订阅选项] 选择允许匿名请求订阅
2)如果选择匿名订阅,则配置订阅服务器时不会出现以上提示
(10)[下一步] 设置快照 代理程序调度
(11)[下一步] 完成配置
当完成出版物的创建后创建出版物的数据库也就变成了一个共享数据库
有数据
srv1.库名..author 有字段:id,name,phone,
srv2.库名..author 有字段:id,name,telphone,adress

要求:
srv1.库名..author 增加记录则 srv1.库名..author 记录增加
srv1.库名..author 的 phone 字段更新, 则 srv1.库名..author 对应字段 telphone 更新
--*/

--大致的处理步骤
--1.在 srv1 上创建连接服务器,以便在 srv1 中操作 srv2,实现同步
```
exec sp_addlinkedserver 'srv2','','SQLOLEDB','srv2 的 sql 实例名或 ip'
exec sp_addlinkedsrvlogin 'srv2','false',null,'用户名','密码'
go
```
--2.在 srv1 和 srv2 这两台电脑中,启动 msdtc(分布式事务处理服务),并且设置为自动启动
。我的电脑--控制面板--管理工具--服务--右键 Distributed Transaction Coordinator--属性--启动--并将启动类型设置为自动启动

```
go
```

--然后创建一个作业定时调用上面的同步处理存储过程就行了

企业管理器
--管理
--SQL Server 代理
--右键作业
--新建作业
--"常规"项中输入作业名称
--"步骤"项
--新建
--"步骤名"中输入步骤名
--"类型"中选择"Transact-SQL 脚本(TSQL)"
--"数据库"选择执行命令的数据库
--"命令"中输入要执行的语句: exec p_process
--确定
--"调度"项
--新建调度
--"名称"中输入调度名称
--"调度类型"中选择你的作业执行安排
--如果选择"反复出现"
--点"更改"来设置你的时间安排

然后将 SQL Agent 服务启动,并设置为自动启动,否则你的作业不会被执行

设置方法:
我的电脑--控制面板--管理工具--服务--右键 SQLSERVERAGENT--属性--启动类型--选择"自动启动"--确定.

--3.实现同步处理的方法 2,定时同步

--在 srv1 中创建如下的同步处理存储过程

```
create proc p_process
as
```

--更新修改过的数据

```
update b set name=i.name,telphone=i.telphone
from srv2.库名.dbo.author b,author i
where b.id=i.id and
(b.name <> i.name or b.telphone <> i.telphone)
```

```
--插入新增的数据
insert srv2.库名.dbo.author(id,name,telphone)
select id,name,telphone from author i
where not exists(
select * from srv2.库名.dbo.author where id=i.id)

--删除已经删除的数据(如果需要的话)
delete b
from srv2.库名.dbo.author b
where not exists(
select * from author where id=b.id)
go
```

五、使用 SQL 访问 MySQL 数据库

5.1、增加数据

insert 语句可以用来将一行或多行数据插到数据库表中，使用的一般形式如下：

Insert into 表名(字段列表) values (值列表);

insert [into] 表名 [(列名 1, 列名 2, 列名 3, ...)] values (值 1, 值 2, 值 3, ...);
insert into students values(NULL, "张三", "男", 20, "18889009876");

有时我们只需要插入部分数据，或者不按照列的顺序进行插入，可以使用这样的形式进行插入：

insert into students (name, sex, age) values("李四", "女", 21);

5.2、查询数据

select 语句常用来根据一定的查询规则到数据库中获取数据，其基本的用法为：

select 字段名 from 表名称 [查询条件];

查询学生表中的所有信息：select * from students;

查询学生表中所有的 name 与 age 信息：select name, age from students;

也可以使用通配符 * 查询表中所有的内容，语句: select * from students;

5.2.1、表达式与条件查询

where 关键词用于指定查询条件，用法形式为: select 列名称 from 表名称 where 条件;

以查询所有性别为女的信息为例，输入查询语句: select * from students where sex="女";

where 子句不仅仅支持 "where 列名 = 值" 这种名等于值的查询形式，对一般的比较运算的运算符都是支持的，例如 =、>、<、>=、<、!= 以及一些扩展运算符 is [not] null、in、like 等等。 还可以对查询条件使用 or 和 and 进行组合查询，以后还会学到更加高级的条件查询方式，这里不再多做介绍。

示例:

查询年龄在 21 岁以上的所有人信息: select * from students where age > 21;

查询名字中带有 "王" 字的所有人信息: select * from students where name like "%王%";

查询 id 小于 5 且年龄大于 20 的所有人信息: select * from students where id<5 and age>20;

5.2.2、聚合函数

获得学生总人数：select count(*) from students

获得学生平均分：select avg(mark) from students

获得最高成绩：select max(mark) from students

获得最低成绩：select min(mark) from students

获得学生总成绩：select sum(mark) from students

5.3、删除数据

delete from 表名 [删除条件];

删除表中所有数据：delete from students;

删除 id 为 10 的行: delete from students where id=10;

删除所有年龄小于 88 岁的数据: delete from students where age<88;

5.4、更新数据

update 语句可用来修改表中的数据，基本的使用形式为:

update 表名称 set 列名称=新值 where 更新条件;

Update 表名 set 字段=值 列表 更新条件

使用示例:

将 id 为 5 的手机号改为默认的"-": update students set tel=default where id=5;

将所有人的年龄增加 1: update students set age=age+1;

将手机号为 13723887766 的姓名改为 "张果"，年龄改为 19: update students set name="张果", age=19 where tel="13723887766";

5.5、修改表

alter table 语句用于创建后对表的修改，基础用法如下:

5.5.1、添加列

基本形式: alter table 表名 add 列名 列数据类型 [after 插入位置];

示例:

在表的最后追加列 address: alter table students add address char(60);

在名为 age 的列后插入列 birthday: alter table students add birthday date after age;

5.5.2、修改列

基本形式: alter table 表名 change 列名称 列新名称 新数据类型;

示例:

将表 tel 列改名为 phone: alter table students change tel phone char(12) default "-";

将 name 列的数据类型改为 char(9): alter table students change name name char(9) not null;

5.5.3、删除列

基本形式: alter table 表名 drop 列名称;

示例:

删除 age 列: alter table students drop age;

5.5.4、重命名表

基本形式: alter table 表名 rename 新表名;

示例:

重命名 students 表为 temp: alter table students rename temp;

5.5.5、删除表

基本形式: drop table 表名;

示例: 删除 students 表: drop table students;

5.5.6、删除数据库

基本形式: drop database 数据库名;

示例: 删除 lcoa 数据库: drop database lcoa;

5.5.7、一千行 MySQL 笔记

```
/* 启动 MySQL */
net start mysql

/* 连接与断开服务器 */
mysql -h 地址 -P 端口 -u 用户名 -p 密码

/* 跳过权限验证登录 MySQL */
mysqld --skip-grant-tables
-- 修改 root 密码
密码加密函数 password()
update mysql.user set password=password('root');

SHOW PROCESSLIST -- 显示哪些线程正在运行
SHOW VARIABLES --

/* 数据库操作 */ ------------------
-- 查看当前数据库
    select database();
-- 显示当前时间、用户名、数据库版本
    select now(), user(), version();
```

```
-- 创建库
    create database[ if not exists] 数据库名 数据库选项
    数据库选项：
        CHARACTER SET charset_name
        COLLATE collation_name
-- 查看已有库
    show databases[ like 'pattern']
-- 查看当前库信息
    show create database 数据库名
-- 修改库的选项信息
    alter database 库名 选项信息
-- 删除库
    drop database[ if exists] 数据库名
        同时删除该数据库相关的目录及其目录内容

/* 表的操作 */ ------------------
-- 创建表
    create [temporary] table[ if not exists] [库名.]表名 ( 表的结构定义 )[ 表选项]
        每个字段必须有数据类型
        最后一个字段后不能有逗号
        temporary 临时表，会话结束时表自动消失
        对于字段的定义：
            字段名 数据类型 [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'string']
-- 表选项
    -- 字符集
        CHARSET = charset_name
        如果表没有设定，则使用数据库字符集
    -- 存储引擎
        ENGINE = engine_name
        表在管理数据时采用的不同的数据结构，结构不同会导致处理方式、提供的特性操作等不同
        常见的引擎：InnoDB MyISAM Memory/Heap BDB Merge Example CSV MaxDB Archive
        不同的引擎在保存表的结构和数据时采用不同的方式
        MyISAM 表文件含义：.frm 表定义，.MYD 表数据，.MYI 表索引
        InnoDB 表文件含义：.frm 表定义，表空间数据和日志文件
        SHOW ENGINES -- 显示存储引擎的状态信息
        SHOW ENGINE 引擎名 {LOGS|STATUS} -- 显示存储引擎的日志或状态信息
    -- 数据文件目录
        DATA DIRECTORY = '目录'
    -- 索引文件目录
```

```
        INDEX DIRECTORY = '目录'
    -- 表注释
        COMMENT = 'string'
    -- 分区选项
        PARTITION BY ... (详细见手册)
-- 查看所有表
    SHOW TABLES[ LIKE 'pattern']
    SHOW TABLES FROM 表名
-- 查看表机构
    SHOW CREATE TABLE 表名     (信息更详细)
    DESC 表名 / DESCRIBE 表名 / EXPLAIN 表名 / SHOW COLUMNS FROM 表名
[LIKE 'PATTERN']
    SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
-- 修改表
    -- 修改表本身的选项
        ALTER TABLE 表名 表的选项
        EG:    ALTER TABLE 表名 ENGINE=MYISAM;
    -- 对表进行重命名
        RENAME TABLE 原表名 TO 新表名
        RENAME TABLE 原表名 TO 库名.表名     (可将表移动到另一个数据库)
        -- RENAME 可以交换两个表名
    -- 修改表的字段机构
        ALTER TABLE 表名 操作名
        -- 操作名
            ADD[ COLUMN] 字段名         -- 增加字段
                AFTER 字段名             -- 表示增加在该字段名后面
                FIRST                  -- 表示增加在第一个
            ADD PRIMARY KEY(字段名)     -- 创建主键
            ADD UNIQUE [索引名] (字段名)-- 创建唯一索引
            ADD INDEX [索引名] (字段名)     -- 创建普通索引
            ADD
            DROP[ COLUMN] 字段名         -- 删除字段
            MODIFY[ COLUMN] 字段名 字段属性         -- 支持对字段属性
进行修改，不能修改字段名(所有原有属性也需写上)
            CHANGE[ COLUMN] 原字段名 新字段名 字段属性         -- 支持
对字段名修改
            DROP PRIMARY KEY     -- 删除主键(删除主键前需删除其
AUTO_INCREMENT 属性)
            DROP INDEX 索引名     -- 删除索引
            DROP FOREIGN KEY 外键     -- 删除外键

-- 删除表
    DROP TABLE[ IF EXISTS] 表名 ...
-- 清空表数据
```

```
    TRUNCATE [TABLE] 表名
-- 复制表结构
    CREATE TABLE 表名 LIKE 要复制的表名
-- 复制表结构和数据
    CREATE TABLE 表名 [AS] SELECT * FROM 要复制的表名
-- 检查表是否有错误
    CHECK TABLE tbl_name [, tbl_name] ... [option] ...
-- 优化表
    OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
-- 修复表
    REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
[QUICK] [EXTENDED] [USE_FRM]
-- 分析表
    ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...



/* 数据操作 */ ------------------
-- 增
    INSERT [INTO] 表名 [(字段列表)] VALUES (值列表)[, (值列表), ...]
        -- 如果要插入的值列表包含所有字段并且顺序一致，则可以省略字段
列表。
        -- 可同时插入多条数据记录!
        REPLACE 与 INSERT 完全一样，可互换。
    INSERT [INTO] 表名 SET 字段名=值[, 字段名=值, ...]
-- 查
    SELECT 字段列表 FROM 表名[ 其他子句]
        -- 可来自多个表的多个字段
        -- 其他子句可以不使用
        -- 字段列表可以用*代替，表示所有字段
-- 删
    DELETE FROM 表名[ 删除条件子句]
        没有条件子句，则会删除全部
-- 改
    UPDATE 表名 SET 字段名=新值[, 字段名=新值] [更新条件]

/* 字符集编码 */ ------------------
-- MySQL、数据库、表、字段均可设置编码
-- 数据编码与客户端编码不需一致
SHOW VARIABLES LIKE 'character_set_%'    -- 查看所有字符集编码项
    character_set_client         客户端向服务器发送数据时使用的编码
    character_set_results         服务器端将结果返回给客户端所使用的编
码
    character_set_connection     连接层编码
```

```
SET 变量名 = 变量值
    set character_set_client = gbk;
    set character_set_results = gbk;
    set character_set_connection = gbk;
SET NAMES GBK;    -- 相当于完成以上三个设置
-- 校对集
    校对集用以排序
    SHOW CHARACTER SET [LIKE 'pattern']/SHOW CHARSET [LIKE 'pattern'] 查看所有字符集
    SHOW COLLATION [LIKE 'pattern']        查看所有校对集
    charset 字符集编码          设置字符集编码
    collate 校对集编码          设置校对集编码

/* 数据类型（列类型） */ ------------------
1. 数值类型
-- a. 整型 ----------
    类型              字节          范围（有符号位）
    tinyint          1 字节     -128 ~ 127          无符号位：0 ~ 255
    smallint     2 字节     -32768 ~ 32767
    mediumint     3 字节     -8388608 ~ 8388607
    int              4 字节
    bigint          8 字节

    int(M)     M 表示总位数
    - 默认存在符号位，unsigned 属性修改
    - 显示宽度，如果某个数不够定义字段时设置的位数，则前面以 0 补填，zerofill 属性修改
        例：int(5)     插入一个数'123'，补填后为'00123'
    - 在满足要求的情况下，越小越好。
    - 1 表示 bool 值真，0 表示 bool 值假。MySQL 没有布尔类型，通过整型 0 和 1 表示。常用 tinyint(1)表示布尔型。

-- b. 浮点型 ----------
    类型                  字节          范围
    float(单精度)          4 字节
    double(双精度)     8 字节
    浮点型既支持符号位 unsigned 属性，也支持显示宽度 zerofill 属性。
        不同于整型，前后均会补填 0.
    定义浮点型时，需指定总位数和小数位数。
        float(M, D)          double(M, D)
        M 表示总位数，D 表示小数位数。
        M 和 D 的大小会决定浮点数的范围。不同于整型的固定范围。
        M 既表示总位数（不包括小数点和正负号），也表示显示宽度（所有显示符号均包括）。
```

支持科学计数法表示。
浮点数表示近似值。

-- c. 定点数 ----------
decimal -- 可变长度
decimal(M, D) M 也表示总位数，D 表示小数位数。
保存一个精确的数值，不会发生数据的改变，不同于浮点数的四舍五入。
将浮点数转换为字符串来保存，每 9 位数字保存为 4 个字节。

2. 字符串类型
-- a. char, varchar ----------
char 定长字符串，速度快，但浪费空间
varchar 变长字符串，速度慢，但节省空间
M 表示能存储的最大长度，此长度是字符数，非字节数。
不同的编码，所占用的空间不同。
char,最多 255 个字符，与编码无关。
varchar,最多 65535 字符，与编码有关。
一条有效记录最大不能超过 65535 个字节。
utf8 最大为 21844 个字符，gbk 最大为 32766 个字符，latin1 最大为 65532 个字符
varchar 是变长的，需要利用存储空间保存 varchar 的长度，如果数据小于 255 个字节，则采用一个字节来保存长度，反之需要两个字节来保存。
varchar 的最大有效长度由最大行大小和使用的字符集确定。
最大有效长度是 65532 字节，因为在 varchar 存字符串时，第一个字节是空的，不存在任何数据，然后还需两个字节来存放字符串的长度，所以有效长度是 64432-1-2=65532 字节。
例：若一个表定义为 CREATE TABLE tb(c1 int, c2 char(30), c3 varchar(N)) charset=utf8; 问 N 的最大值是多少？ 答：(65535-1-2-4-30*3)/3

-- b. blob, text ----------
blob 二进制字符串（字节字符串）
tinyblob, blob, mediumblob, longblob
text 非二进制字符串（字符字符串）
tinytext, text, mediumtext, longtext
text 在定义时，不需要定义长度，也不会计算总长度。
text 类型在定义时，不可给 default 值

-- c. binary, varbinary ----------
类似于 char 和 varchar，用于保存二进制字符串，也就是保存字节字符串而非字符字符串。
char, varchar, text 对应 binary, varbinary, blob.

3. 日期时间类型
一般用整型保存时间戳，因为 PHP 可以很方便的将时间戳进行格式化。

```
    datetime    8 字节    日期及时间        1000-01-01 00:00:00 到
9999-12-31 23:59:59
    date        3 字节    日期              1000-01-01 到 9999-12-31
    timestamp    4 字节    时间戳          19700101000000 到 2038-01-19
03:14:07
    time        3 字节    时间              -838:59:59 到 838:59:59
    year        1 字节    年份              1901 - 2155

datetime     “YYYY-MM-DD hh:mm:ss”
timestamp     “YY-MM-DD hh:mm:ss”
             “YYYYMMDDhhmmss”
             “YYMMDDhhmmss”
             YYYYMMDDhhmmss
             YYMMDDhhmmss
date         “YYYY-MM-DD”
             “YY-MM-DD”
             “YYYYMMDD”
             “YYMMDD”
             YYYYMMDD
             YYMMDD
time         “hh:mm:ss”
             “hhmmss”
             hhmmss
year         “YYYY”
             “YY”
             YYYY
             YY

4. 枚举和集合
-- 枚举(enum) ----------
enum(val1, val2, val3...)
    在已知的值中进行单选。最大数量为 65535.
    枚举值在保存时，以 2 个字节的整型(smallint)保存。每个枚举值，按保存
的位置顺序，从 1 开始逐一递增。
    表现为字符串类型，存储却是整型。
    NULL 值的索引是 NULL。
    空字符串错误值的索引值是 0。

-- 集合（set） ----------
set(val1, val2, val3...)
    create table tab ( gender set('男', '女', '无') );
    insert into tab values ('男, 女');
    最多可以有 64 个不同的成员。以 bigint 存储，共 8 个字节。采取位运算的
形式。
```

当创建表时，SET 成员值的尾部空格将自动被删除。

/* 选择类型 */
-- PHP 角度
1. 功能满足
2. 存储空间尽量小，处理效率更高
3. 考虑兼容问题

-- IP 存储 ----------
1. 只需存储，可用字符串
2. 如果需计算，查找等，可存储为 4 个字节的无符号 int，即 unsigned
    1) PHP 函数转换
        ip2long 可转换为整型，但会出现携带符号问题。需格式化为无符号的整型。
        利用 sprintf 函数格式化字符串
        sprintf("%u", ip2long('192.168.3.134'));
        然后用 long2ip 将整型转回 IP 字符串
    2) MySQL 函数转换(无符号整型，UNSIGNED)
        INET_ATON('127.0.0.1') 将 IP 转为整型
        INET_NTOA(2130706433) 将整型转为 IP

/* 列属性（列约束） */ ------------------
1. 主键
    - 能唯一标识记录的字段，可以作为主键。
    - 一个表只能有一个主键。
    - 主键具有唯一性。
    - 声明字段时，用 primary key 标识。
        也可以在字段列表之后声明
            例：create table tab (id int, stu varchar(10), primary key (id));
    - 主键字段的值不能为 null。
    - 主键可以由多个字段共同组成。此时需要在字段列表后声明的方法。
        例：create table tab (id int, stu varchar(10), age int, primary key (stu, age));

2. unique 唯一索引（唯一约束）
    使得某字段的值也不能重复。

3. null 约束
    null 不是数据类型，是列的一个属性。
    表示当前列是否可以为 null，表示什么都没有。

null，允许为空。默认。
not null，不允许为空。
insert into tab values (null, 'val');
-- 此时表示将第一个字段的值设为 null，取决于该字段是否允许为 null

4. default 默认值属性
当前字段的默认值。
insert into tab values (default, 'val'); -- 此时表示强制使用默认值。
create table tab (add_time timestamp default current_timestamp);
-- 表示将当前时间的时间戳设为默认值。
current_date, current_time

5. auto_increment 自动增长约束
自动增长必须为索引（主键或 unique）
只能存在一个字段为自动增长。
默认为 1 开始自动增长。可以通过表属性 auto_increment = x 进行设置，或 alter table tbl auto_increment = x;

6. comment 注释
例：create table tab (id int) comment '注释内容';

7. foreign key 外键约束
用于限制主表与从表数据完整性。
alter table t1 add constraint `t1_t2_fk` foreign key (t1_id) references t2(id);
-- 将表 t1 的 t1_id 外键关联到表 t2 的 id 字段。
-- 每个外键都有一个名字，可以通过 constraint 指定

存在外键的表，称之为从表（子表），外键指向的表，称之为主表（父表）。

作用：保持数据一致性，完整性，主要目的是控制存储在外键表（从表）中的数据。

MySQL 中，可以对 InnoDB 引擎使用外键约束：
语法：
foreign key (外键字段) references 主表名 (关联字段) [主表记录删除时的动作] [主表记录更新时的动作]
此时需要检测一个从表的外键需要约束为主表的已存在的值。外键在没有关联的情况下，可以设置为 null.前提是该外键列，没有 not null。

可以不指定主表记录更改或更新时的动作，那么此时主表的操作被拒绝。

如果指定了 on update 或 on delete：在删除或更新时，有如下几个操作可以选择：

1. cascade，级联操作。主表数据被更新（主键值更新），从表也被更新（外键值更新）。主表记录被删除，从表相关记录也被删除。

2. set null，设置为 null。主表数据被更新（主键值更新），从表的外键被设置为 null。主表记录被删除，从表相关记录外键被设置成 null。但注意，要求该外键列，没有 not null 属性约束。

3. restrict，拒绝父表删除和更新。

注意，外键只被 InnoDB 存储引擎所支持。其他引擎是不支持的。

```
/* 建表规范 */ ------------------
    -- Normal Format, NF
        - 每个表保存一个实体信息
        - 每个具有一个 ID 字段作为主键
        - ID 主键 + 原子表
    -- 1NF, 第一范式
        字段不能再分，就满足第一范式。
    -- 2NF, 第二范式
        满足第一范式的前提下，不能出现部分依赖。
        消除符合主键就可以避免部分依赖。增加单列关键字。
    -- 3NF, 第三范式
        满足第二范式的前提下，不能出现传递依赖。
        某个字段依赖于主键，而有其他字段依赖于该字段。这就是传递依赖。
        将一个实体信息的数据放在一个表内实现。


/* select */ ------------------

select [all|distinct] select_expr from -> where -> group by [合计函数]
-> having -> order by -> limit

a. select_expr
    -- 可以用 * 表示所有字段。
        select * from tb;
    -- 可以使用表达式（计算公式、函数调用、字段也是个表达式）
        select stu, 29+25, now() from tb;
    -- 可以为每个列使用别名。适用于简化列标识，避免多个列标识符重复。
        - 使用 as 关键字，也可省略 as.
        select stu+10 as add10 from tb;

b. from 子句
    用于标识查询来源。
```

```
    -- 可以为表起别名。使用 as 关键字。
        select * from tb1 as tt, tb2 as bb;
    -- from 子句后，可以同时出现多个表。
        -- 多个表会横向叠加到一起，而数据会形成一个笛卡尔积。
        select * from tb1, tb2;

c. where 子句
    -- 从 from 获得的数据源中进行筛选。
    -- 整型 1 表示真，0 表示假。
    -- 表达式由运算符和运算数组成。
        -- 运算数：变量（字段）、值、函数返回值
        -- 运算符：
            =, <=>, <>, !=, <=, <, >=, >, !, &&, ||,
            in (not) null, (not) like, (not) in, (not) between and, is
(not), and, or, not, xor
            is/is not 加上 ture/false/unknown，检验某个值的真假
            <=>与<>功能相同，<=>可用于 null 比较

d. group by 子句, 分组子句
    group by 字段/别名 [排序方式]
    分组后会进行排序。升序：ASC，降序：DESC

    以下[合计函数]需配合 group by 使用：
    count 返回不同的非 NULL 值数目    count(*)、count(字段)
    sum 求和
    max 求最大值
    min 求最小值
    avg 求平均值
    group_concat 返回带有来自一个组的连接的非 NULL 值的字符串结果。组内
字符串连接。

e. having 子句，条件子句
    与 where 功能、用法相同，执行时机不同。
    where 在开始时执行检测数据，对原数据进行过滤。
    having 对筛选出的结果再次进行过滤。
    having 字段必须是查询出来的，where 字段必须是数据表存在的。
    where 不可以使用字段的别名，having 可以。因为执行 WHERE 代码时，可
能尚未确定列值。
    where 不可以使用合计函数。一般需用合计函数才会用 having
    SQL 标准要求 HAVING 必须引用 GROUP BY 子句中的列或用于合计函数中的列。

f. order by 子句，排序子句
    order by 排序字段/别名 排序方式 [,排序字段/别名 排序方式]...
    升序：ASC，降序：DESC
```

支持多个字段的排序。

g. limit 子句，限制结果数量子句
仅对处理好的结果进行数量限制。将处理好的结果的看作是一个集合，按照记录出现的顺序，索引从 0 开始。
limit 起始位置, 获取条数
省略第一个参数，表示从索引 0 开始。limit 获取条数

h. distinct, all 选项
distinct 去除重复记录
默认为 all, 全部记录

/* UNION */ ------------------
将多个 select 查询的结果组合成一个结果集合。
SELECT ... UNION [ALL|DISTINCT] SELECT ...
默认 DISTINCT 方式，即所有返回的行都是唯一的
建议，对每个 SELECT 查询加上小括号包裹。
ORDER BY 排序时，需加上 LIMIT 进行结合。
需要各 select 查询的字段数量一样。
每个 select 查询的字段列表(数量、类型)应一致，因为结果中的字段名以第一条 select 语句为准。

/* 子查询 */ ------------------
- 子查询需用括号包裹。
-- from 型
from 后要求是一个表，必须给子查询结果取个别名。
- 简化每个查询内的条件。
- from 型需将结果生成一个临时表格，可用以原表的锁定的释放。
- 子查询返回一个表，表型子查询。
select * from (select * from tb where id>0) as subfrom where id>1;
-- where 型
- 子查询返回一个值，标量子查询。
- 不需要给子查询取别名。
- where 子查询内的表，不能直接用以更新。
select * from tb where money = (select max(money) from tb);
-- 列子查询
如果子查询结果返回的是一列。
使用 in 或 not in 完成查询
exists 和 not exists 条件
如果子查询返回数据，则返回 1 或 0。常用于判断条件。
select column1 from t1 where exists (select * from t2);
-- 行子查询

```
        查询条件是一个行。
        select * from t1 where (id, gender) in (select id, gender from
t2);
        行构造符：(col1, col2, ...) 或 ROW(col1, col2, ...)
        行构造符通常用于与对能返回两个或两个以上列的子查询进行比较。

    -- 特殊运算符
    != all()    相当于 not in
    = some()    相当于 in。any 是 some 的别名
    != some()    不等同于 not in，不等于其中某一个。
    all, some 可以配合其他运算符一起使用。


/* 连接查询(join) */ ------------------
    将多个表的字段进行连接，可以指定连接条件。
-- 内连接(inner join)
    - 默认就是内连接，可省略 inner。
    - 只有数据存在时才能发送连接。即连接结果不能出现空行。
    on 表示连接条件。其条件表达式与 where 类似。也可以省略条件（表示条
件永远为真）
    也可用 where 表示连接条件。
    还有 using，但需字段名相同。 using(字段名)

    -- 交叉连接 cross join
        即，没有条件的内连接。
        select * from tb1 cross join tb2;
-- 外连接(outer join)
    - 如果数据不存在，也会出现在连接结果中。
    -- 左外连接 left join
        如果数据不存在，左表记录会出现，而右表为 null 填充
    -- 右外连接 right join
        如果数据不存在，右表记录会出现，而左表为 null 填充
-- 自然连接(natural join)
    自动判断连接条件完成连接。
    相当于省略了 using，会自动查找相同字段名。
    natural join
    natural left join
    natural right join

select info.id, info.name, info.stu_num, extra_info.hobby,
extra_info.sex from info, extra_info where info.stu_num =
extra_info.stu_id;

/* 导入导出 */ ------------------
```

```
select * into outfile 文件地址 [控制格式] from 表名;    -- 导出表数据
load data [local] infile 文件地址 [replace|ignore] into table 表名 [控制格式];    -- 导入数据
    生成的数据默认的分隔符是制表符
    local 未指定，则数据文件必须在服务器上
    replace 和 ignore 关键词控制对现有的唯一键记录的重复的处理
-- 控制格式
fields    控制字段格式
默认：fields terminated by '\t' enclosed by '' escaped by '\\'
    terminated by 'string'    -- 终止
    enclosed by 'char'        -- 包裹
    escaped by 'char'        -- 转义
    -- 示例：
        SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'
        FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        FROM test_table;
lines    控制行格式
默认：lines terminated by '\n'
    terminated by 'string'    -- 终止

/* insert */ ------------------
select 语句获得的数据可以用 insert 插入。

可以省略对列的指定，要求 values () 括号内，提供给了按照列顺序出现的所有字段的值。
    或者使用 set 语法。
    insert into tbl_name set field=value,...;

可以一次性使用多个值，采用(), (), ();的形式。
    insert into tbl_name values (), (), ();

可以在列值指定时，使用表达式。
    insert into tbl_name values (field_value, 10+10, now());
可以使用一个特殊值 default，表示该列使用默认值。
    insert into tbl_name values (field_value, default);

可以通过一个查询的结果，作为需要插入的值。
    insert into tbl_name select ...;

可以指定在插入的值出现主键(或唯一索引)冲突时,更新其他非主键列的信息。
    insert into tbl_name values/set/select on duplicate key update 字段=值, …;
```

```
/* delete */ ------------------
DELETE FROM tbl_name [WHERE where_definition] [ORDER BY ...] [LIMIT
row_count]
```

按照条件删除

指定删除的最多记录数。Limit

可以通过排序条件删除。order by + limit

支持多表删除，使用类似连接语法。
delete from 需要删除数据多表 1，表 2 using 表连接操作 条件。

```
/* truncate */ ------------------
TRUNCATE [TABLE] tbl_name
```

清空数据
删除重建表

区别：
1，truncate 是删除表再创建，delete 是逐条删除
2，truncate 重置 auto_increment 的值。而 delete 不会
3，truncate 不知道删除了几条，而 delete 知道。
4，当被用于带分区的表时，truncate 会保留分区

```
/* 备份与还原 */ ------------------
```

备份，将数据的结构与表内数据保存起来。
利用 mysqldump 指令完成。

-- 导出
1. 导出一张表
    mysqldump -u 用户名 -p 密码 库名 表名 > 文件名(D:/a.sql)
2. 导出多张表
    mysqldump -u 用户名 -p 密码 库名 表 1 表 2 表 3 > 文件名(D:/a.sql)
3. 导出所有表
    mysqldump -u 用户名 -p 密码 库名 > 文件名(D:/a.sql)
4. 导出一个库
    mysqldump -u 用户名 -p 密码 -B 库名 > 文件名(D:/a.sql)

可以-w 携带备份条件

-- 导入
1. 在登录 mysql 的情况下：
    source 备份文件

2. 在不登录的情况下
    mysql -u用户名 -p密码 库名 < 备份文件

/* 视图 */ ----------
什么是视图：
    视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。
    视图具有表结构文件，但不存在数据文件。
    对其中所引用的基础表来说，视图的作用类似于筛选。定义视图的筛选可以来自当前或其它数据库的一个或多个表，或者其它视图。通过视图进行查询没有任何限制，通过它们进行数据修改时的限制也很少。
    视图是存储在数据库中的查询的 sql 语句，它主要出于两种原因：安全原因，视图可以隐藏一些数据，如：社会保险基金表，可以用视图只显示姓名，地址，而不显示社会保险号和工资数等，另一原因是可使复杂的查询易于理解和使用。

-- 创建视图
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW view_name [(column_list)] AS select_statement
    - 视图名必须唯一，同时不能与表重名。
    - 视图可以使用 select 语句查询到的列名，也可以自己指定相应的列名。
    - 可以指定视图执行的算法，通过 ALGORITHM 指定。
    - column_list 如果存在，则数目必须等于 SELECT 语句检索的列数

-- 查看结构
    SHOW CREATE VIEW view_name

-- 删除视图
    - 删除视图后，数据依然存在。
    - 可同时删除多个视图。
    DROP VIEW [IF EXISTS] view_name ...

-- 修改视图结构
    - 一般不修改视图，因为不是所有的更新视图都会映射到表上。
    ALTER VIEW view_name [(column_list)] AS select_statement

-- 视图作用
    1. 简化业务逻辑
    2. 对客户端隐藏真实的表结构

-- 视图算法(ALGORITHM)
    MERGE   合并
        将视图的查询语句，与外部查询需要先合并再执行！

TEMPTABLE　　临时表
　　将视图执行完毕后，形成临时表，再做外层查询！
UNDEFINED　　未定义(默认)，指的是 MySQL 自主去选择相应的算法。

/* 事务(transaction) */ ------------------
事务是指逻辑上的一组操作,组成这组操作的各个单元,要不全成功要不全失败。
- 支持连续 SQL 的集体成功或集体撤销。
- 事务是数据库在数据晚自习方面的一个功能。
- 需要利用 InnoDB 或 BDB 存储引擎，对自动提交的特性支持完成。
- InnoDB 被称为事务安全型引擎。

-- 事务开启
START TRANSACTION; 或者 BEGIN;
开启事务后，所有被执行的 SQL 语句均被认作当前事务内的 SQL 语句。
-- 事务提交
COMMIT;
-- 事务回滚
ROLLBACK;
如果部分操作发生问题，映射到事务开启前。

-- 事务的特性
1. 原子性（Atomicity）
　　事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
2. 一致性（Consistency）
　　事务前后数据的完整性必须保持一致。
　　- 事务开始和结束时，外部数据一致
　　- 在整个事务过程中，操作是连续的
3. 隔离性（Isolation）
　　多个用户并发访问数据库时,一个用户的事务不能被其它用户的事物所干扰，多个并发事务之间的数据要相互隔离。
4. 持久性（Durability）
　　一个事务一旦被提交，它对数据库中的数据改变就是永久性的。

-- 事务的实现
1. 要求是事务支持的表类型
2. 执行一组相关的操作前开启事务
3. 整组操作完成后，都成功，则提交；如果存在失败，选择回滚，则会回到事务开始的备份点。

-- 事务的原理
利用 InnoDB 的自动提交(autocommit)特性完成。

普通的 MySQL 执行语句后，当前的数据提交操作均可被其他客户端可见。
而事务是暂时关闭“自动提交”机制，需要 commit 提交持久化数据操作。

-- 注意
1. 数据定义语言（DDL）语句不能被回滚，比如创建或取消数据库的语句，和创建、取消或更改表或存储的子程序的语句。
2. 事务不能被嵌套

-- 保存点
SAVEPOINT 保存点名称 -- 设置一个事务保存点
ROLLBACK TO SAVEPOINT 保存点名称 -- 回滚到保存点
RELEASE SAVEPOINT 保存点名称 -- 删除保存点

-- InnoDB 自动提交特性设置
SET autocommit = 0|1; 0 表示关闭自动提交，1 表示开启自动提交。
- 如果关闭了，那普通操作的结果对其他客户端也不可见，需要 commit 提交后才能持久化数据操作。
- 也可以关闭自动提交来开启事务。但与 START TRANSACTION 不同的是，
SET autocommit 是永久改变服务器的设置，直到下次再次修改该设置。(针对当前连接)
而 START TRANSACTION 记录开启前的状态，而一旦事务提交或回滚后就需要再次开启事务。(针对当前事务)

/* 锁表 */
表锁定只用于防止其它客户端进行不正当地读取和写入
MyISAM 支持表锁，InnoDB 支持行锁
-- 锁定
LOCK TABLES tbl_name [AS alias]
-- 解锁
UNLOCK TABLES

/* 触发器 */ ----------
触发程序是与表有关的命名数据库对象，当该表出现特定事件时，将激活该对象
监听：记录的增加、修改、删除。

-- 创建触发器
CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW trigger_stmt
参数：
trigger_time 是触发程序的动作时间。它可以是 before 或 after，以指明触发程序是在激活它的语句之前或之后触发。

trigger_event 指明了激活触发程序的语句的类型
INSERT：将新行插入表时激活触发程序
UPDATE：更改某一行时激活触发程序
DELETE：从表中删除某一行时激活触发程序
tbl_name：监听的表，必须是永久性的表，不能将触发程序与 TEMPORARY 表或视图关联起来。
trigger_stmt：当触发程序激活时执行的语句。执行多个语句，可使用 BEGIN...END 复合语句结构

```
-- 删除
DROP TRIGGER [schema_name.]trigger_name
```

可以使用 old 和 new 代替旧的和新的数据
更新操作，更新前是 old，更新后是 new.
删除操作，只有 old.
增加操作，只有 new.

```
-- 注意
    1. 对于具有相同触发程序动作时间和事件的给定表，不能有两个触发程序。


-- 字符连接函数
concat(str1[, str2,...])

-- 分支语句
if 条件 then
    执行语句
elseif 条件 then
    执行语句
else
    执行语句
end if;

-- 修改最外层语句结束符
delimiter 自定义结束符号
    SQL 语句
自定义结束符号

delimiter ;         -- 修改回原来的分号

-- 语句块包裹
begin
    语句块
end
```

-- 特殊的执行
1. 只要添加记录，就会触发程序。
2. Insert into on duplicate key update 语法会触发：
    如果没有重复记录，会触发 before insert, after insert;
    如果有重复记录并更新，会触发 before insert, before update, after update;
    如果有重复记录但是没有发生更新，则触发 before insert, before update
3. Replace 语法 如果有记录，则执行 before insert, before delete, after delete, after insert

/* SQL 编程 */ ----------

--// 局部变量 ----------
-- 变量声明
    declare var_name[,...] type [default value]
    这个语句被用来声明局部变量。要给变量提供一个默认值，请包含一个 default 子句。值可以被指定为一个表达式，不需要为一个常数。如果没有 default 子句，初始值为 null。

-- 赋值
    使用 set 和 select into 语句为变量赋值。

    - 注意：在函数内是可以使用全局变量（用户自定义的变量）

--// 全局变量 ----------
-- 定义、赋值
set 语句可以定义并为变量赋值。
set @var = value;
也可以使用 select into 语句为变量初始化并赋值。这样要求 select 语句只能返回一行，但是可以是多个字段，就意味着同时为多个变量进行赋值，变量的数量需要与查询的列数一致。
还可以把赋值语句看作一个表达式，通过 select 执行完成。此时为了避免=被当作关系运算符看待，使用:=代替。（set 语句可以使用= 和 :=）。
select @var:=20;
select @v1:=id, @v2=name from t1 limit 1;
select * from tbl_name where @var:=30;

select into 可以将表中查询获得的数据赋给变量。
    -| select max(height) into @max_height from tb;

-- 自定义变量名

为了避免 select 语句中，用户自定义的变量与系统标识符（通常是字段名）冲突，用户自定义变量在变量名前使用@作为开始符号。

```
@var=10;
```

- 变量被定义后，在整个会话周期都有效（登录到退出）

```
--// 控制结构 ----------
-- if 语句
if search_condition then
    statement_list
[elseif search_condition then
    statement_list]
...
[else
    statement_list]
end if;

-- case 语句
CASE value WHEN [compare-value] THEN result
[WHEN [compare-value] THEN result ...]
[ELSE result]
END


-- while 循环
[begin_label:] while search_condition do
    statement_list
end while [end_label];
```

- 如果需要在循环内提前终止 while 循环，则需要使用标签；标签需要成对出现。

```
    -- 退出循环
        退出整个循环 leave
        退出当前循环 iterate
        通过退出的标签决定退出哪个循环


--// 内置函数 ----------
-- 数值函数
abs(x)              -- 绝对值 abs(-10.9) = 10
format(x, d)     -- 格式化千分位数值 format(1234567.456, 2) =
1,234,567.46
```

```
ceil(x)              -- 向上取整 ceil(10.1) = 11
floor(x)          -- 向下取整 floor (10.1) = 10
round(x)          -- 四舍五入去整
mod(m, n)          -- m%n m mod n 求余 10%3=1
pi()              -- 获得圆周率
pow(m, n)          -- m^n
sqrt(x)              -- 算术平方根
rand()              -- 随机数
truncate(x, d)     -- 截取 d 位小数

-- 时间日期函数
now(), current_timestamp();      -- 当前日期时间
current_date();                     -- 当前日期
current_time();                     -- 当前时间
date('yyyy-mm-dd hh:ii:ss');     -- 获取日期部分
time('yyyy-mm-dd hh:ii:ss');     -- 获取时间部分
date_format('yyyy-mm-dd hh:ii:ss', '%d %y %a %d %m %b %j');     -- 格式
化时间
unix_timestamp();                  -- 获得 unix 时间戳
from_unixtime();                  -- 从时间戳获得时间

-- 字符串函数
length(string)             -- string 长度, 字节
char_length(string)         -- string 的字符个数
substring(str, position [,length])         -- 从 str 的 position 开始,取
length 个字符
replace(str ,search_str ,replace_str)     -- 在 str 中用 replace_str 替换
search_str
instr(string ,substring)     -- 返回 substring 首次在 string 中出现的位置
concat(string [,...])     -- 连接字串
charset(str)             -- 返回字串字符集
lcase(string)             -- 转换成小写
left(string, length)     -- 从 string2 中的左边起取 length 个字符
load_file(file_name)     -- 从文件读取内容
locate(substring, string [,start_position])     -- 同 instr,但可指定开
始位置
lpad(string, length, pad)     -- 重复用 pad 加在 string 开头,直到字串长度
为 length
ltrim(string)             -- 去除前端空格
repeat(string, count)     -- 重复 count 次
rpad(string, length, pad)     --在 str 后用 pad 补充,直到长度为 length
rtrim(string)             -- 去除后端空格
strcmp(string1 ,string2)     -- 逐字符比较两字串大小
```

```
-- 流程函数
case when [condition] then result [when [condition] then result ...] [else result] end   多分支
if(expr1,expr2,expr3)  双分支。

-- 聚合函数
count()
sum();
max();
min();
avg();
group_concat()

-- 其他常用函数
md5();
default();


--// 存储函数，自定义函数 ----------
-- 新建
    CREATE FUNCTION function_name (参数列表) RETURNS 返回值类型
        函数体

    - 函数名，应该合法的标识符，并且不应该与已有的关键字冲突。
    - 一个函数应该属于某个数据库,可以使用 db_name.funciton_name 的形式执行当前函数所属数据库，否则为当前数据库。
    - 参数部分，由"参数名"和"参数类型"组成。多个参数用逗号隔开。
    - 函数体由多条可用的 mysql 语句，流程控制，变量声明等语句构成。
    - 多条语句应该使用 begin...end 语句块包含。
    - 一定要有 return 返回值语句。

-- 删除
    DROP FUNCTION [IF EXISTS] function_name;

-- 查看
    SHOW FUNCTION STATUS LIKE 'partten'
    SHOW CREATE FUNCTION function_name;

-- 修改
    ALTER FUNCTION function_name 函数选项


--// 存储过程，自定义功能 ----------
-- 定义
```

存储存储过程 是一段代码（过程），存储在数据库中的 sql 组成。
一个存储过程通常用于完成一段业务逻辑，例如报名，交班费，订单入库等。
而一个函数通常专注与某个功能，视为其他程序服务的，需要在其他语句中调用函数才可以，而存储过程不能被其他调用，是自己执行 通过 call 执行。

```
-- 创建
CREATE PROCEDURE sp_name (参数列表)
    过程体
```

参数列表：不同于函数的参数列表，需要指明参数类型
IN，表示输入型
OUT，表示输出型
INOUT，表示混合型

注意，没有返回值。

```
/* 存储过程 */ ------------------
```

存储过程是一段可执行性代码的集合。相比函数，更偏向于业务逻辑。
调用：CALL 过程名

```
-- 注意
```

- 没有返回值。
- 只能单独调用，不可夹杂在其他语句中

```
-- 参数
IN|OUT|INOUT 参数名 数据类型
IN          输入：在调用过程中，将数据输入到过程体内部的参数
OUT         输出：在调用过程中，将过程体处理完的结果返回到客户端
INOUT     输入输出：既可输入，也可输出

-- 语法
CREATE PROCEDURE 过程名 (参数列表)
BEGIN
    过程体
END
```

```
/* 用户和权限管理 */ ------------------
```

用户信息表：mysql.user

```
-- 刷新权限
FLUSH PRIVILEGES
-- 增加用户
CREATE USER 用户名 IDENTIFIED BY [PASSWORD] 密码(字符串)
```

- 必须拥有 mysql 数据库的全局 CREATE USER 权限，或拥有 INSERT 权限。

- 只能创建用户，不能赋予权限。
- 用户名，注意引号：如 'user_name'@'192.168.1.1'
- 密码也需引号，纯数字密码也要加引号
- 要在纯文本中指定密码，需忽略 PASSWORD 关键词。要把密码指定为由 PASSWORD()函数返回的混编值，需包含关键字 PASSWORD

```
-- 重命名用户
RENAME USER old_user TO new_user
-- 设置密码
SET PASSWORD = PASSWORD('密码')     -- 为当前用户设置密码
SET PASSWORD FOR 用户名 = PASSWORD('密码')     -- 为指定用户设置密码
-- 删除用户
DROP USER 用户名
-- 分配权限/添加用户
GRANT 权限列表 ON 表名 TO 用户名 [IDENTIFIED BY [PASSWORD] 'password']
```

- all privileges 表示所有权限
- *.* 表示所有库的所有表
- 库名.表名 表示某库下面的某表

```
-- 查看权限
SHOW GRANTS FOR 用户名
```

-- 查看当前用户权限

SHOW GRANTS; 或 SHOW GRANTS FOR CURRENT_USER; 或 SHOW GRANTS FOR CURRENT_USER();

```
-- 撤消权限
REVOKE 权限列表 ON 表名 FROM 用户名
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 用户名     -- 撤销所有权限
-- 权限层级
-- 要使用 GRANT 或 REVOKE，您必须拥有 GRANT OPTION 权限，并且您必须用于您正在授予或撤销的权限。
全局层级：全局权限适用于一个给定服务器中的所有数据库，mysql.user
    GRANT ALL ON *.*和 REVOKE ALL ON *.*只授予和撤销全局权限。
数据库层级：数据库权限适用于一个给定数据库中的所有目标，mysql.db, mysql.host
    GRANT ALL ON db_name.*和 REVOKE ALL ON db_name.*只授予和撤销数据库权限。
表层级：表权限适用于一个给定表中的所有列，mysql.talbes_priv
    GRANT ALL ON db_name.tbl_name 和 REVOKE ALL ON db_name.tbl_name 只授予和撤销表权限。
列层级：列权限适用于一个给定表中的单一列，mysql.columns_priv
    当使用 REVOKE 时，您必须指定与被授权列相同的列。
-- 权限列表
ALL [PRIVILEGES]     -- 设置除 GRANT OPTION 之外的所有简单权限
ALTER     -- 允许使用 ALTER TABLE
ALTER ROUTINE     -- 更改或取消已存储的子程序
CREATE     -- 允许使用 CREATE TABLE
```

CREATE ROUTINE -- 创建已存储的子程序
CREATE TEMPORARY TABLES -- 允许使用 CREATE TEMPORARY TABLE
CREATE USER -- 允许使用 CREATE USER, DROP USER, RENAME USER 和 REVOKE ALL PRIVILEGES。
CREATE VIEW -- 允许使用 CREATE VIEW
DELETE -- 允许使用 DELETE
DROP -- 允许使用 DROP TABLE
EXECUTE -- 允许用户运行已存储的子程序
FILE -- 允许使用 SELECT...INTO OUTFILE 和 LOAD DATA INFILE
INDEX -- 允许使用 CREATE INDEX 和 DROP INDEX
INSERT -- 允许使用 INSERT
LOCK TABLES -- 允许对您拥有 SELECT 权限的表使用 LOCK TABLES
PROCESS -- 允许使用 SHOW FULL PROCESSLIST
REFERENCES -- 未被实施
RELOAD -- 允许使用 FLUSH
REPLICATION CLIENT -- 允许用户询问从属服务器或主服务器的地址
REPLICATION SLAVE -- 用于复制型从属服务器（从主服务器中读取二进制日志事件）
SELECT -- 允许使用 SELECT
SHOW DATABASES -- 显示所有数据库
SHOW VIEW -- 允许使用 SHOW CREATE VIEW
SHUTDOWN -- 允许使用 mysqladmin shutdown
SUPER -- 允许使用 CHANGE MASTER, KILL, PURGE MASTER LOGS 和 SET GLOBAL 语句,mysqladmin debug 命令;允许您连接(一次),即使已达到 max_connections。
UPDATE -- 允许使用 UPDATE
USAGE -- “无权限”的同义词
GRANT OPTION -- 允许授予权限

/* 表维护 */
-- 分析和存储表的关键字分布
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE 表名 ...
-- 检查一个或多个表是否有错误
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
-- 整理数据文件的碎片
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...

/* 杂项 */ ----------
1. 可用反引号（`）为标识符（库名、表名、字段名、索引、别名）包裹，以避免与关键字重名！中文也可以作为标识符！
2. 每个库目录存在一个保存当前数据库的选项文件 db.opt。
3. 注释：

单行注释 # 注释内容
多行注释 /* 注释内容 */
单行注释 -- 注释内容 (标准 SQL 注释风格，要求双破折号后加一空格符（空格、TAB、换行等）)

4. 模式通配符：
_ 任意单个字符
% 任意多个字符，甚至包括零字符
单引号需要进行转义 \'

5. CMD 命令行内的语句结束符可以为 ";", "\G", "\g"，仅影响显示结果。其他地方还是用分号结束。delimiter 可修改当前对话的语句结束符。
6. SQL 对大小写不敏感
7. 清除已有语句：\c

5.5.8、常用的 SQL

```
/*==============================================================*/
/* DBMS name:      MySQL 5.0                                    */
/* Created on:     2017/3/5 10:29:05                            */
/*==============================================================*/


drop table if exists Address;

drop table if exists ArticleComment;

drop table if exists ArticleType;

drop table if exists Articles;

drop table if exists DictSub;

drop table if exists DictTop;

drop table if exists OrderPdt;

drop table if exists Orders;

drop table if exists ProductComment;

drop table if exists Products;

drop table if exists Users;

/*==============================================================*/
/* Table: Address                                               */
```

```
/*==============================================================*/
create table Address
(
   `AddressId` int not null auto_increment comment '收货地址编号',
   `UserId` int not null comment '用户编号',
   `Province` varchar(50) not null comment '省',
   `City` varchar(50) not null comment '市',
   `County` varchar(50) not null comment '县/区',
   `Street` varchar(300) not null comment '详细地址',
   `RevName` varchar(30) not null comment '收货人姓名',
   `PostCode` varchar(20) comment '邮政编码',
   `Mobile` varchar(50) not null comment '手机',
   `Phone` varchar(50) comment '电话',
   `IsDefault` bool comment '是否为默认地址',
   primary key (AddressId)
);

alter table Address comment '收货地址';

/*==============================================================*/
/* Table: ArticleComment                                        */
/*==============================================================*/
create table ArticleComment
(
   `ArticleCommentId` int not null auto_increment comment '文章评论编
号',
   `ArticleId` int not null comment '文章编号',
   `UserId` int not null comment '用户编号',
   `ArticleCommentContent` varchar(4000) not null comment '文章评论内
容',
   `ArticleCommentDate` timestamp default CURRENT_TIMESTAMP comment '
文章评论时间',
   `ArticleCommentState` int default 1 comment '状态',
   `ArticleRemark` int comment '打分',
   `ArticleCommentReserver1` varchar(4000) comment '备用 1',
   `ArticleCommentReserver2` varchar(4000) comment '备用 2',
   primary key (ArticleCommentId)
);

alter table ArticleComment comment '文章评论';

/*==============================================================*/
/* Table: ArticleType                                           */
/*==============================================================*/
```

```
create table ArticleType
(
   `ArticleTypeId` int not null auto_increment comment '文章栏目编号',
   `ArticleTypeName` varchar(200) comment '文章栏目名称',
   `ArticleTypeState` int default 1 comment '状态',
   `ArticleTypeDesc` varchar(4000) comment '文章栏目描述',
   `ArticleTypePicture` varchar(400) comment '文章栏目图片',
   `ArticleTypeReserve1` varchar(4000) comment '备用 1',
   `ArticleTypeReserve2` varchar(4000) comment '备用 2',
   primary key (ArticleTypeId)
);

alter table ArticleType comment '文章栏目';

/*==============================================================*/
/* Table: Articles                                              */
/*==============================================================*/
create table Articles
(
   `ArticleId` int not null auto_increment comment '文章编号',
   `ArticleTypeId` int not null comment '文章栏目编号',
   `ArticleTitle` varchar(400) not null comment '文章标题',
   `ArticleContent` text comment '文章内容',
   `ArticleDate` timestamp default CURRENT_TIMESTAMP comment '文章发布
时间',
   `ArticleAuthor` varchar(200) comment '文章发布者',
   `ArticleFileName` varchar(100) comment '静态文件名',
   `ArticleThumbNail` varchar(200) comment '缩略图片',
   `ArticleAddition` varchar(200) comment '附件名称',
   `ArticleLevel` int comment '显示的优先级',
   `ArticleIsAllowComment` integer default 1 comment '是否允许评论',
   `ArticleState` int default 1 comment '状态',
   `ArticleHotCount` int comment '点击次数',
   `ArticleReserve1` varchar(4000) comment '备用 1',
   `ArticleReserve2` varchar(4000) comment '备用 2',
   `ArticleReserve3` numeric(8,0) comment '备用 3',
   primary key (ArticleId)
);

alter table Articles comment '文章';

/*==============================================================*/
/* Table: DictSub                                               */
/*==============================================================*/
```

```
create table DictSub
(
   `SubId` int not null auto_increment comment '子项编号',
   `DictId` int not null comment '字典编号',
   `SubName` varchar(200) not null comment '子项名称',
   `SubDesc` varchar(4000) comment '子项描述',
   `SubReserve1` varchar(4000) comment '保留备用 1',
   primary key (SubId)
);

alter table DictSub comment '字典子项';

/*==============================================================*/
/* Table: DictTop                                               */
/*==============================================================*/
create table DictTop
(
   `DictId` int not null auto_increment comment '字典编号',
   `DictName` varchar(100) not null comment '字典名称',
   `DictDesc` varchar(4000) comment '字典描述',
   `DictReserve1` varchar(4000) comment '保留备用',
   primary key (DictId)
);

alter table DictTop comment '字典';

/*==============================================================*/
/* Table: OrderPdt                                              */
/*==============================================================*/
create table OrderPdt
(
   `OrderPdtId` int not null auto_increment comment '订单商品编号',
   `Id` int not null comment '编号',
   `UserId` int not null comment '用户编号',
   `OrderId` int comment '订单号',
   `PdtAmount` int comment '订购数量',
   `PdtPrice` decimal comment '单价',
   `PdtReserve1` varchar(2000) comment '备用 1',
   `PdtReserve2` varchar(4000) comment '备用 2',
   primary key (OrderPdtId)
);

alter table OrderPdt comment '订单商品';
```

```
/*==============================================================*/
/* Table: Orders                                                */
/*==============================================================*/
create table Orders
(
   `OrderId` int not null auto_increment comment '订单号',
   `AddressId` int not null comment '收货地址编号',
   `OrderState` int default 1 comment '订单状态',
   `ExpressNO` varchar(50) comment '快递编号',
   `ExpressName` varchar(50) comment '快递名称',
   `PayMoney` decimal comment '应支付',
   `PayedMoney` decimal comment '已支付',
   `SendInfo` varchar(300) comment '发货人信息',
   `BuyDate` timestamp default CURRENT_TIMESTAMP comment '下单时间',
   `PayDate` datetime comment '支付时间',
   `SendDate` datetime comment '发货时间',
   `ReceivDate` datetime comment '收货时间',
   `OrderMessage` varchar(4000) comment '附言',
   `UserId` integer comment '用户编号',
   `OrderReserve1` varchar(4000) comment '备用 1',
   `OrderReserve2` varchar(4000) comment '备用 2',
   `OrderReserve3` decimal comment '备用 3',
   primary key (OrderId)
);

alter table Orders comment '订单';

/*==============================================================*/
/* Table: ProductComment                                        */
/*==============================================================*/
create table ProductComment
(
   `ProductCommentId` int not null auto_increment comment '商品评论编
号',
   `ProductId` int not null comment '商品编号',
   `UserId` int not null comment '用户编号',
   `ProductCommentContent` varchar(4000) comment '商品评论内容',
   `ProductCommentDate` timestamp default CURRENT_TIMESTAMP comment '
商品评论时间',
   `ProductCommentState` int comment '状态',
   `ProductCommentRemark` int comment '打分',
   `ProductCommentReserve1` varchar(4000) comment '备用 1',
   `ProductCommentReserve2` varchar(4000) comment '备用 2',
   primary key (ProductCommentId)
```

```
);

alter table ProductComment comment '商品评论';

/*==============================================================*/
/* Table: Products                                              */
/*==============================================================*/
create table Products
(
   `Id` int not null auto_increment comment '编号',
   `Name` varchar(200) not null comment '名称',
   `SubIdColor` int not null comment '所属颜色',
   `SubIdBrand` int not null comment '所属品牌',
   `SubIdInlay` int not null comment '所属镶嵌',
   `SubIdMoral` int not null comment '所属寓意',
   `SubIdMaterial` int not null comment '所属种水',
   `SubIdTopLevel` int not null comment '一级分类编号',
   `MarketPrice` decimal comment '市场参考价',
   `MyPrice` decimal not null comment '玉源直销价',
   `Discount` decimal default 1 comment '折扣',
   `Picture` varchar(200) comment '图片',
   `Amount` int comment '库存量',
   `Description` text comment '详细描述',
   `State` int default 1 comment '状态',
   `AddDate` timestamp default CURRENT_TIMESTAMP comment '上货日期',
   `Hang` int comment '挂件',
   `RawStone` int comment '赌石',
   `Size` varchar(200) comment '尺寸',
   `ExpressageName` varchar(100) comment '快递名称',
   `Expressage` decimal comment '快递费',
   `AllowComment` int default 1 comment '是否允许评论',
   `Reserve1` varchar(4000) comment '保留备用 1',
   `Reserve2` varchar(4000) comment '保留备用 2',
   `Reserve3` decimal(0) comment '保留备用 3',
   primary key (Id)
);

alter table Products comment '商品';

/*==============================================================*/
/* Table: Users                                                 */
/*==============================================================*/
create table Users
(
```

```
   `UserId` int not null auto_increment comment '用户编号',
   `UserName` varchar(200) not null comment '用户名',
   `Password` varchar(512) not null comment '密码',
   `Email` varchar(100) not null comment '邮箱',
   `Sex` varchar(10) comment '性别',
   `State` int default 1 comment '状态',
   `RightCode` int comment '权限状态',
   `RegDate` timestamp default CURRENT_TIMESTAMP comment '注册时间',
   `RegIP` varchar(200) comment '注册 IP',
   `LastLoginDate` datetime comment '最近登录时间',
   `UserReserve1` varchar(4000) comment '保留备用 1',
   `UserReserve2` varchar(4000) comment '保留备用 2',
   `UserReserve3` varchar(4000) comment '保留备用 3',
   primary key (UserId)
);

alter table Users comment '用户';

alter table Address add constraint FK_AddressBelongUser foreign key
(UserId)
      references Users (UserId) on delete restrict on update restrict;

alter table ArticleComment add constraint FK_ArticleCommentForArticle
foreign key (ArticleId)
      references Articles (ArticleId) on delete restrict on update
restrict;

alter table ArticleComment add constraint FK_ArticleCommentForUser
foreign key (UserId)
      references Users (UserId) on delete restrict on update restrict;

alter table Articles add constraint FK_ArticleBelongType foreign key
(ArticleTypeId)
      references ArticleType (ArticleTypeId) on delete restrict on
update restrict;

alter table DictSub add constraint FK_BelongDict foreign key (DictId)
      references DictTop (DictId) on delete cascade on update cascade;

alter table OrderPdt add constraint FK_BelongOrder foreign key (OrderId)
      references Orders (OrderId) on delete cascade on update cascade;

alter table OrderPdt add constraint FK_CartForUser foreign key (UserId)
      references Users (UserId) on delete restrict on update restrict;
```

```
alter table OrderPdt add constraint FK_OrderDepProduct foreign key (Id)
      references Products (Id) on delete restrict on update restrict;

alter table Orders add constraint FK_OrderBelongAddress foreign key
(AddressId)
      references Address (AddressId) on delete restrict on update
restrict;

alter table ProductComment add constraint FK_ProductCommentBelongUsers
foreign key (UserId)
      references Users (UserId) on delete restrict on update restrict;

alter table ProductComment add constraint FK_ProductCommentForProduct
foreign key (ProductId)
      references Products (Id) on delete restrict on update restrict;

alter table Products add constraint FK_BelongBrand foreign key
(SubIdMaterial)
      references DictSub (SubId) on delete restrict on update restrict;

alter table Products add constraint FK_BelongColor foreign key
(SubIdBrand)
      references DictSub (SubId) on delete restrict on update restrict;

alter table Products add constraint FK_BelongInlay foreign key
(SubIdInlay)
      references DictSub (SubId) on delete restrict on update restrict;

alter table Products add constraint FK_BelongMaterial foreign key
(SubIdColor)
      references DictSub (SubId) on delete restrict on update restrict;

alter table Products add constraint FK_BelongMoral foreign key
(SubIdTopLevel)
      references DictSub (SubId) on delete restrict on update restrict;

alter table Products add constraint FK_BelongTopLevel foreign key
(SubIdMoral)
      references DictSub (SubId) on delete restrict on update restrict;
```