# 实验 1 DBMS 的安装和使用

姓名：段皞一
学号：3190105359
专业：计算机科学与技术

## 一 实验目的

1. 通过安装某个数据库管理系统，初步了解 DBMS 的运行环境。
2. 了解 DBMS 交互界面、图形界面和系统管理工具的使用。
3. 搭建实验平台。

## 二 实验平台

1. 操作系统： Windows
2. 数据库管理系统：SQL Server 或 MySQL

## 三 实验内容和要求

### 3.1 认识数据库

数据库是按照数据结构来组织、存储和管理数据的仓库，是存储在一起的相关数据的集合，其优点主要体现在以下几个方面。其优点主要有以下几个方面：

减少数据的冗余度，节省数据的存储空间；

具有较高的数据独立性和易扩充性；

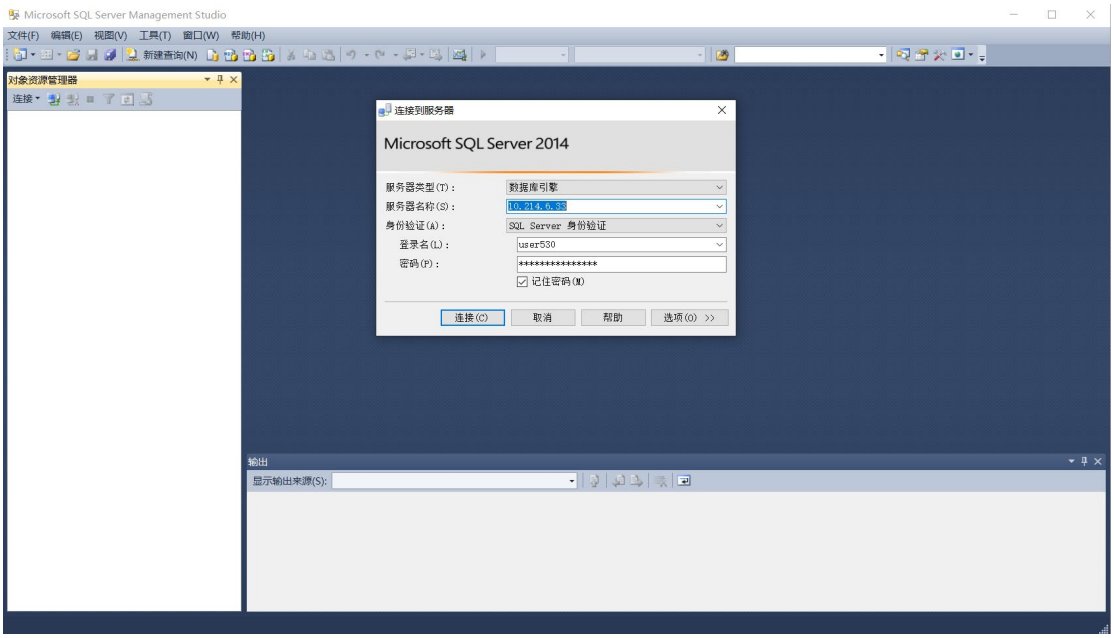
实现数据资源的充分共享。

### 3.2 根据某个 DBMS 的安装说明等文档，安装 DBMS

#### 3.2.1 下载安装 Microsoft SQL Server 2014

该版本的 DBMS 环境在浙江大学正版软件管理系统中即可下载。下载完毕后，需进行一些初步的配置，然后就可通过该平台初步了解 DBMS 的运行环境。

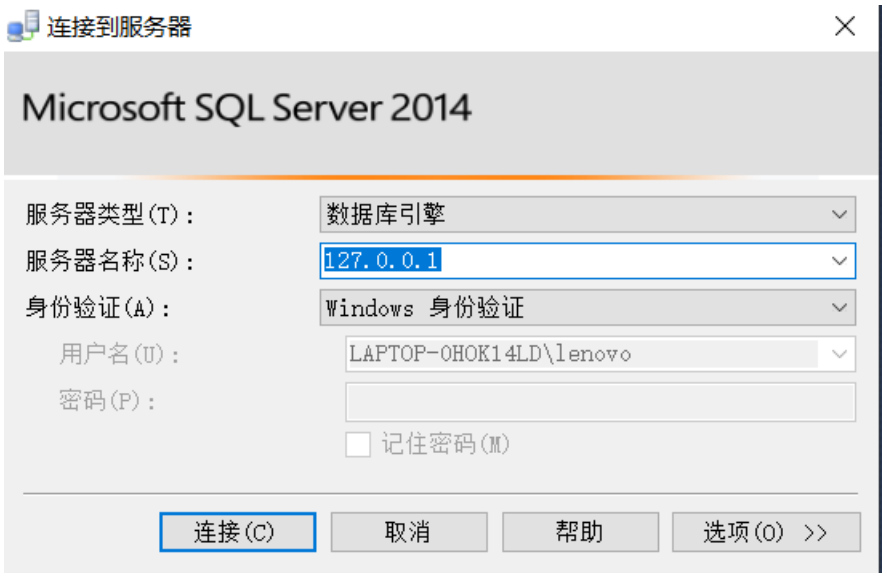
打开 Microsoft SQL Server Management Studio,可以出现如下初始界面。



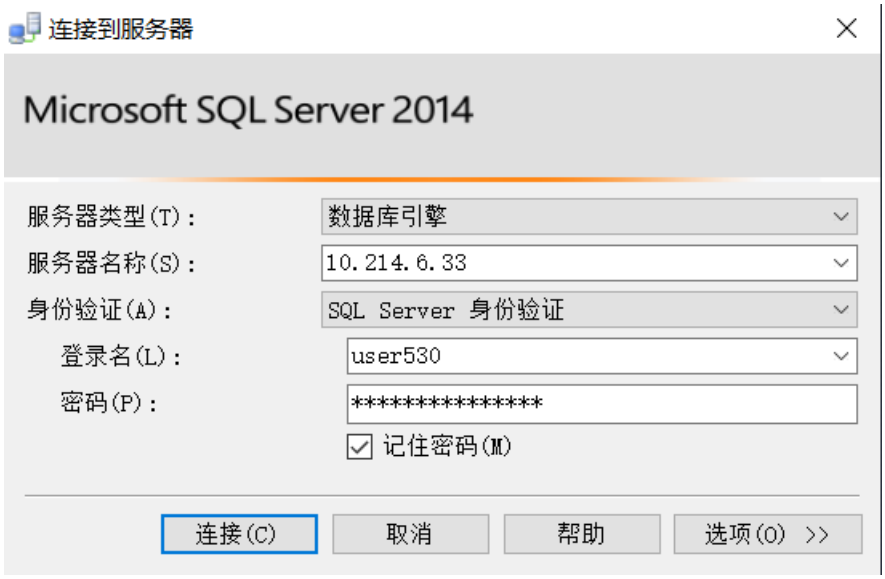

### 3.2.2 连接到服务器

需要连接到服务器，才能进行后续的实验。可以连接两种类型的服务器，本地以及外来服务器。

若要链接本地服务器，只需在服务器名称中输入本机的 ip 地址或者 127.0.0.1 或者 localhost,身份验证选择 Windows 身份验证，左键点击连接即可进行本地服务器的连接。

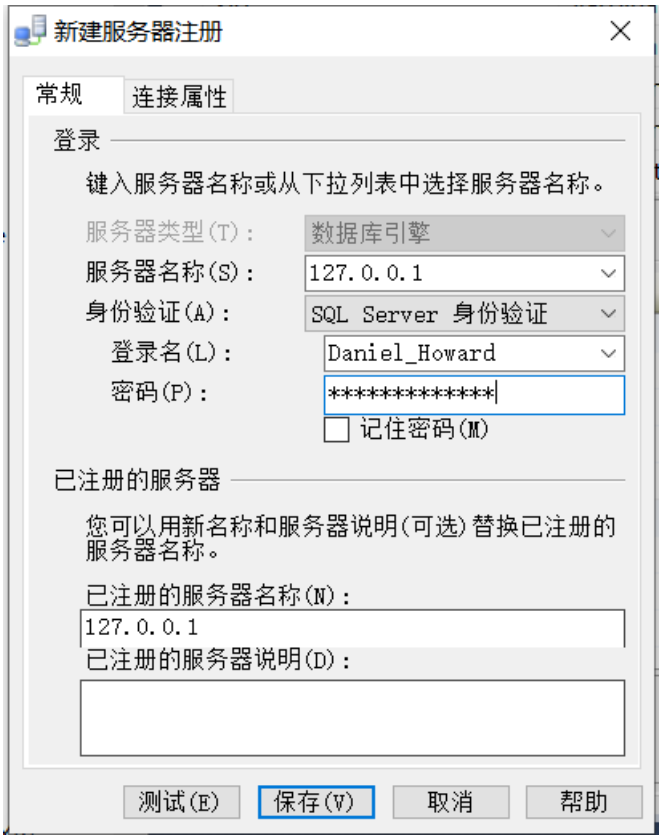



若要连外来的服务器，这也是较为常用的场景，在服务器名称中输入学校的服务器 ip 地址，并进行 SQL Server 验证。




## 3.3 了解 DBMS 的用户管理

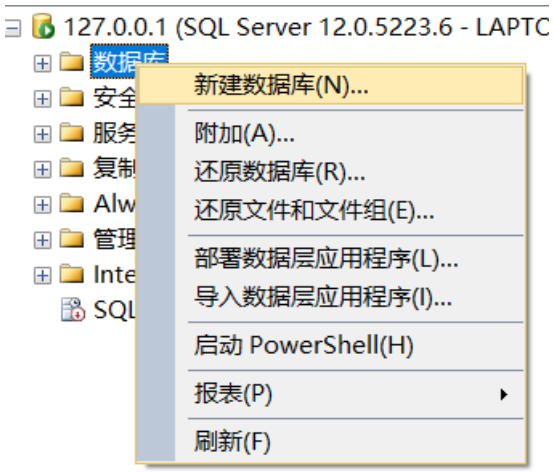
### 3.3.1 注册



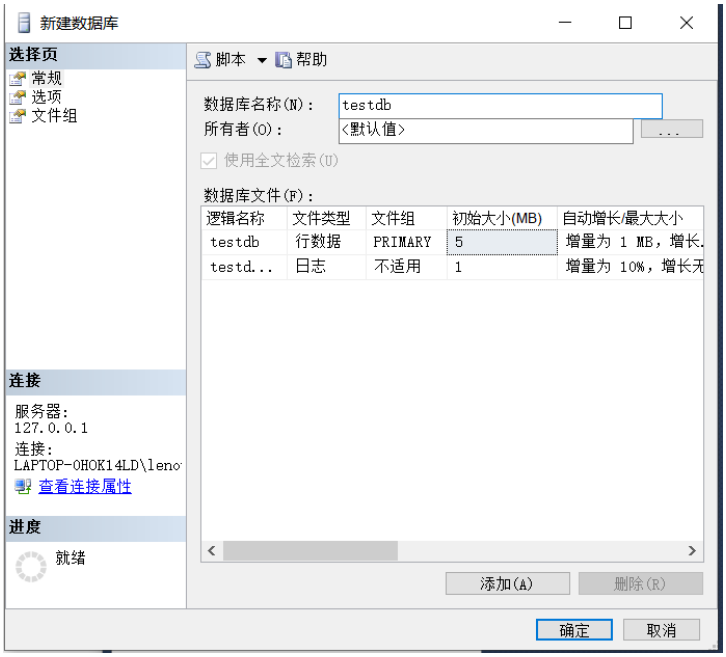

### 3.3.2 新建数据库

#### 3.3.2.1 图形界面方法进行数据库的创建

在“对象资源管理器”中，右键单击数据库，选择“新建数据库”。




接下来在弹出的新建数据库窗口中，可以对数据库进行一系列的初始配置，数据库建成之后也可以打开这个窗口，进行后期的修改与调整。



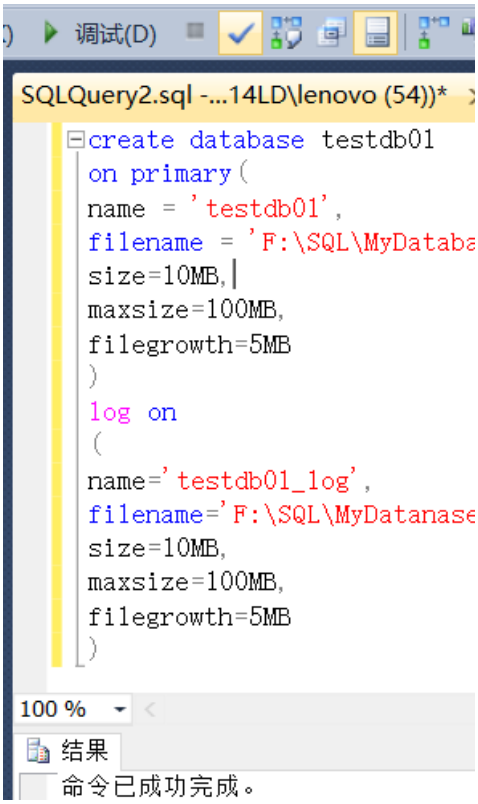

### 3.3.2.2 脚本方法创建数据库

单击新建查询按钮，输入以下脚本：

```
create database testdb01
on primary(
name = 'testdb01',
filename = 'F:\SQL\MyDatabase\testdb01.mdf',
size=10MB,
maxsize=100MB,
filegrowth=5MB
)
log on
(
name='testdb01_log',
```

```
filename='F:\SQL\MyDatanase\testdb01_log.ldf',
size=10MB,
maxsize=100MB,
filegrowth=5MB
)
```

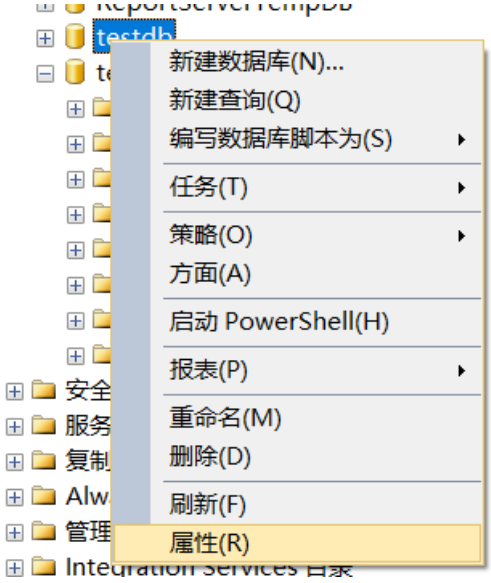
点击分析按钮，代码无误以后，点击执行按钮，完成命令。



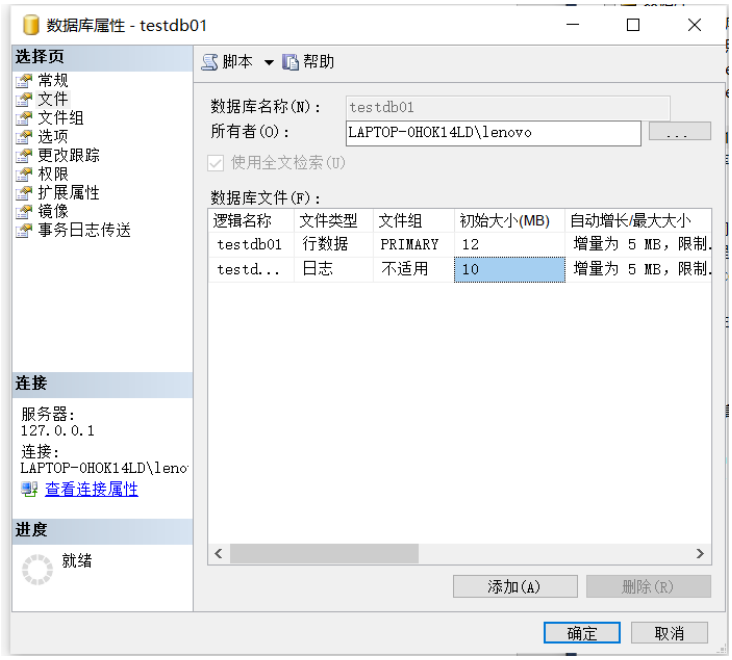

### 3.3.3 数据库属性的更改

#### 3.3.3.1 图形界面方法进行数据库属性的修改

选中要更改属性的数据库，右键选择属性按钮。



在弹出的对话框中就可以进行数据库的属性的修改了。




值得注意的是，修改的时候，存储路径不能够进行修改的，但是数据库的名字依然进行修改。

### 3.3.3.2 脚本方法进行数据库属性的修改

一下是一个脚本的样例，可以进行数据库大小，开销上限，名称，增长指数等参数进行修改。

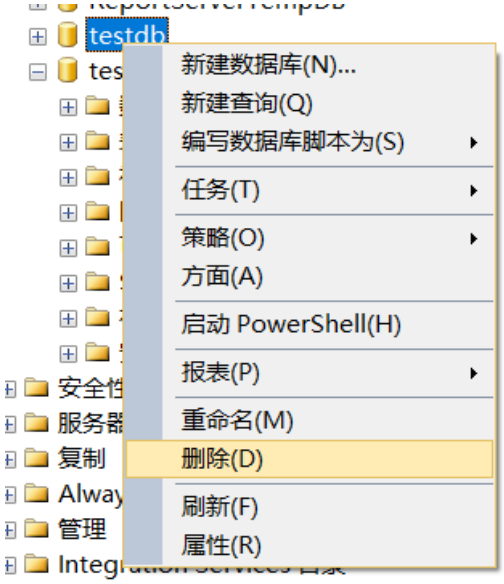
```
ALTER DATABASE TESTDB01
MODIFY NAME = TESTDB01;
ALTER DATABASE TESTDB
MODIFY FILE(
NAME=TESTDT,
SIZE=20MB,
MAXSIZE=80MB,
FILEGROWTH=5MB
);

EXEC SP_HELPDB TESTDB;
```

## 3.3.4 数据库的删除

### 3.3.4.1 界面方法进行数据库的删除操作

右键数据库，选择删除按钮。




### 3.3.4.2 脚本删除操作命令

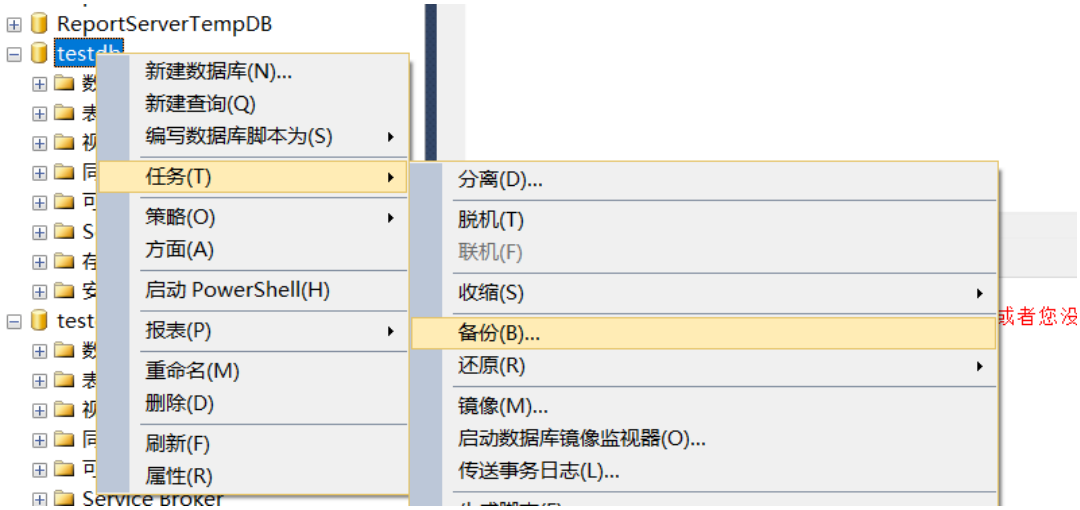
相关代码如下：

```
DROP DATABASE Sales, NewSales;
```

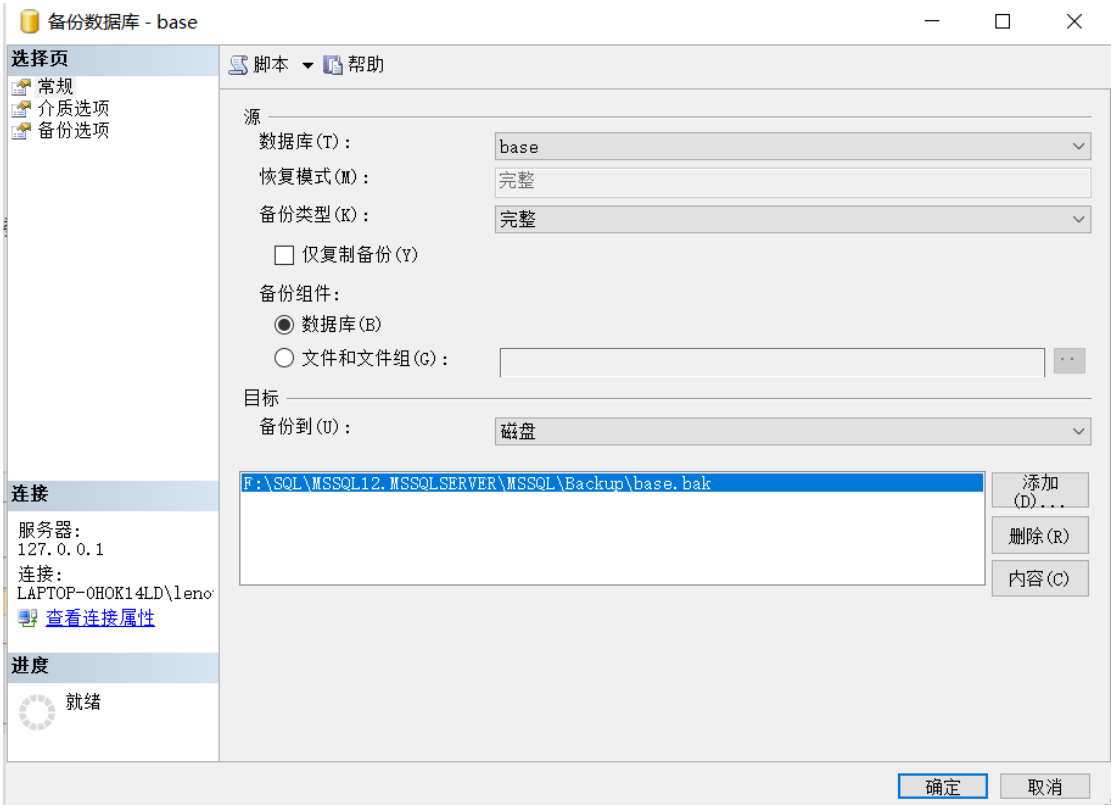
## 3.3.5 数据库的备份与还原

通过还原一组备份然后恢复数据库的策略，能够最大限度地降低灾难性数据丢失的风险。还原方式主要分为差异备份和完整备份两种。

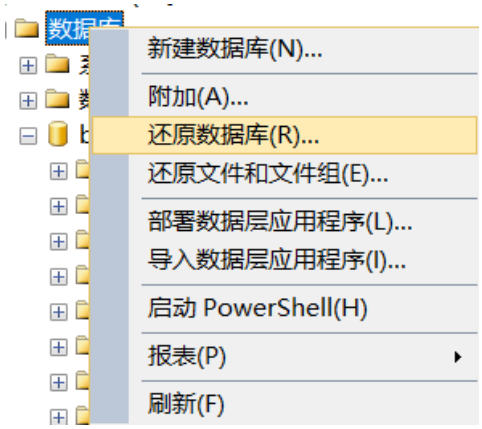
选择需要进行备份的数据库，右键选择“任务”->”备份”



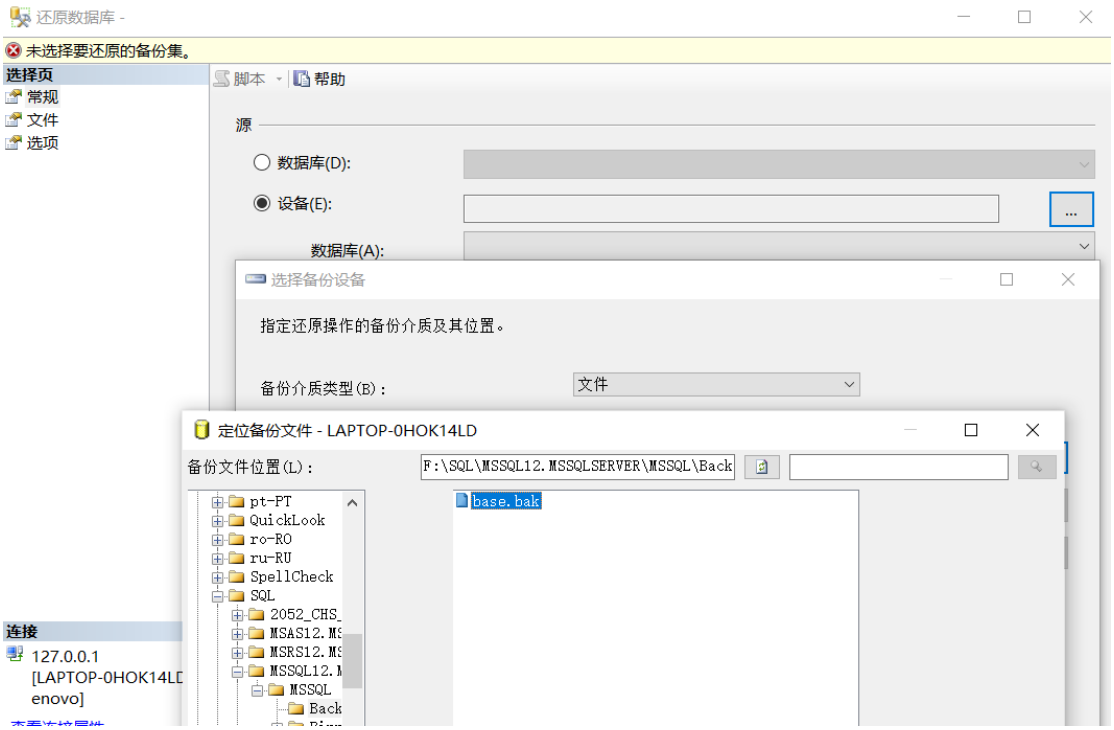

在弹出的对话框中，配置备份的属性，包括选择备份的类型、备份的位置等等。

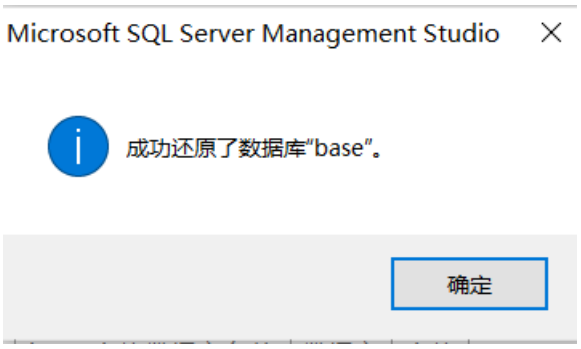



根据备份的文件信息可以进行数据库的还原。具体操作如下：选择相应的数据库，右键选择还原数据库。




在弹出的窗口中找到目标备份文件，即可进行数据库的还原。



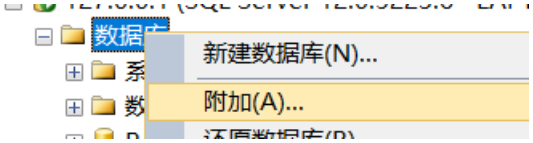




### 3.3.6 数据库分离和附加

数据库的分离是指将整个数据库文件进行拷贝，附加是将数据库文件完整的导入目标的服务器。与数据库的备份与还原相比，由于不产生中间的备份文件，所以移动过程中要耗费更多的存储空间。

分离的方法是，右键所选的数据库，点击“任务”->“分离”。


附加的方法是，右键数据库，点击附加，然后倒入目标的数据库文件路径即可。




## 3.4 SQL Server 下的数据类型

### 3.4.1 数字类型

| 数据类型 | 输出 |
| --- | --- |
| bigint | 8 byte |
| int | 4 byte |
| smallint | 2 byte |
| tinyint | 1 byte |
| float | 取决于 n 的值 |

### 3.4.2 时间类型

| 数据类型 | 输出 |
| --- | --- |
| time | 12:35:29.1234567 |
| date | 2021-3-15 |
| smalldatetime | 2021-3-15 12:35:00 |
| datetime | 2021-3-15 12:35:29.123 |
| datetime2 | 2021-3-15 12:35:29.1234567 |

### 3.4.3 字符串的类型

char[n]

varchar[(n|max)]

支持 Unicode 编码，所以每个字符占用两个字节:

nchar[n]

nvarchar[n|max]

## 3.5 表的创建和操作

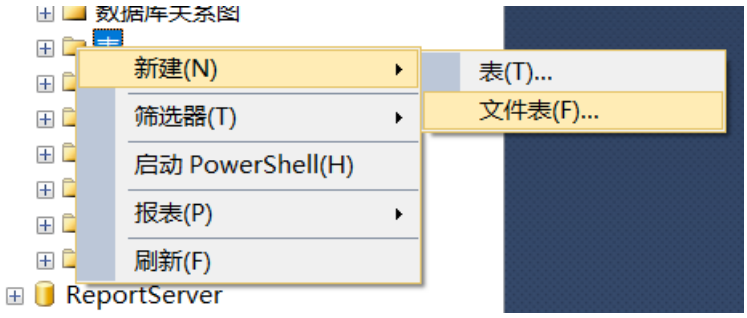
### 3.5.1 新建表

若要创建表，必须提供表的名称和该表中每个列的名称和数据类型，指出每个列中是否允许空值。

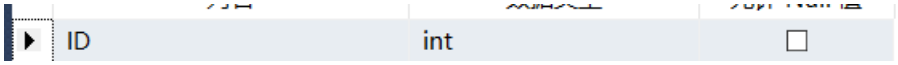
此外，大多数表中都有一个主键，由表的一列或者多列组成，主键始终是唯一的。

#### 3.5.1.1 图形界面进行创建

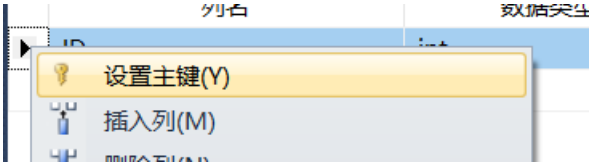
右键数据库栏下的“表”，点击新建“表”



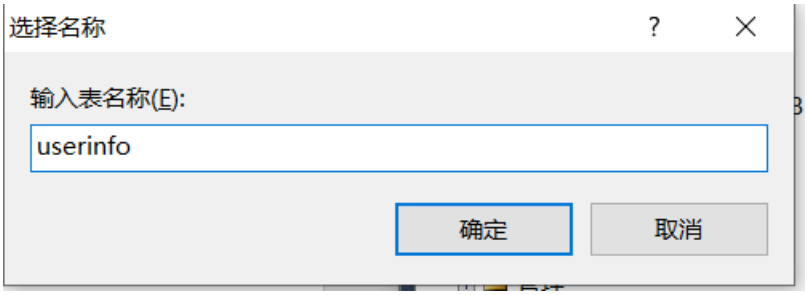

在弹出的窗口下，可以对各个字段进行属性的初始化。




点击一个字段，右键可以将其设为该表的主键。




保存时，可以对表进行命名。



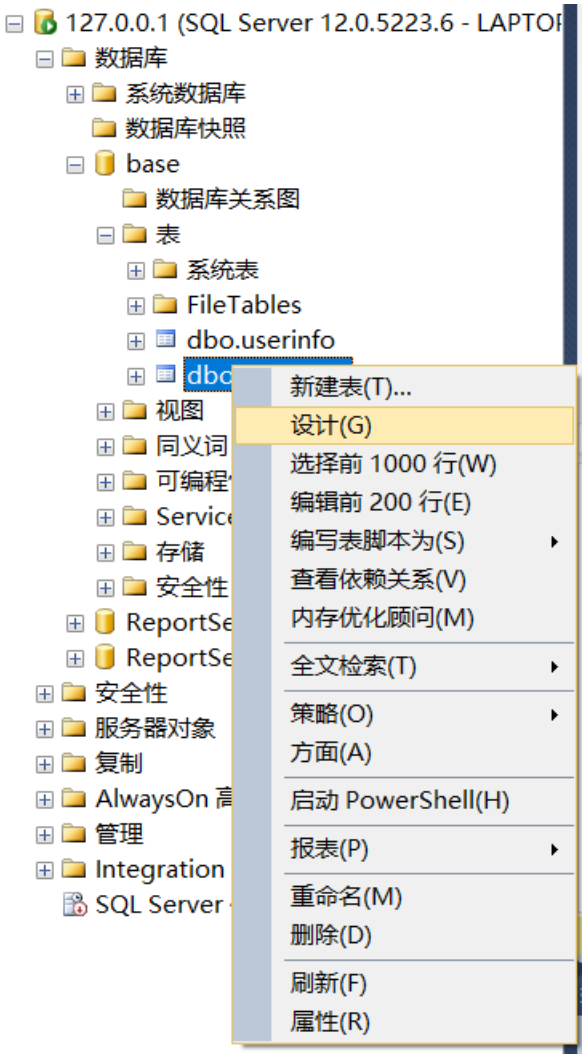

### 3.5.1.2 使用 SQL 语言进行表的创建

```
create table userinfo2
(ID int primary key not null,
name varchar(20) not null
);
```

## 3.5.2 修改、删除表结构

### 3.5.2.1 操作界面进行表的修改

右键相应的表，点击“设计”。




在弹出的对话框中，可以对表的各个字段的属性进行调整。


### 3.5.2.2 SQL 语句进行

使用 SQL 语句进行类型长度的修改。比如将 varchar(10)改成 varchar(100)。

```
alter table userinfo
alter column name varchar(100);
```

使用 SQL 语句进行字段类型的修改。

```
alter table userinfo
alter column age float;
```

添加是否可以为空。

```
alter table userinfo
alter column age float not null;
```

添加主键。

```
alter table userinfo
add constraint KID primary key(ID);
```

新建字段。

```
alter table userinfo
add grade varchar(10);
```

first：第一个位置；

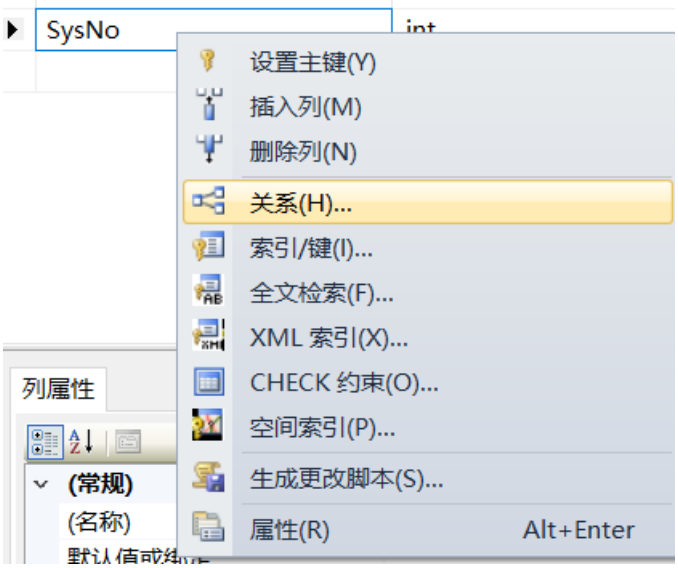
after：在哪个字段之后；默认在最后一个字段的后面。

```
alter table userinfo
add newkey FIRST;
```

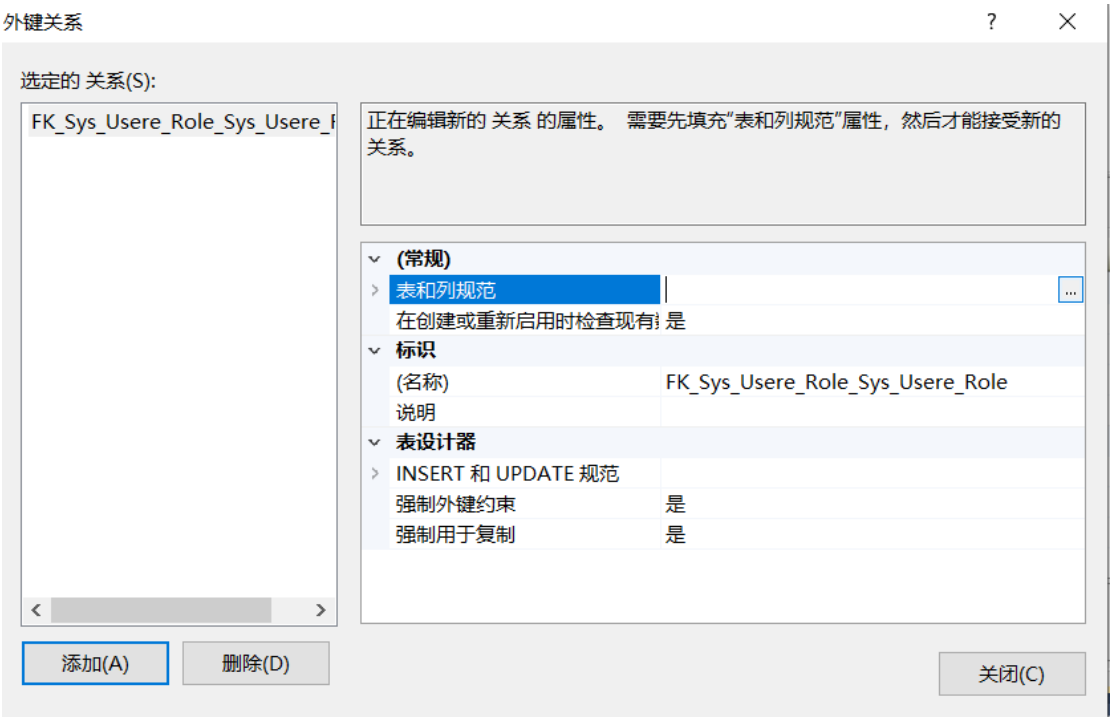
### 3.5.3 定义主键外键

#### 3.5.3.1 使用界面方法定义主键和外键

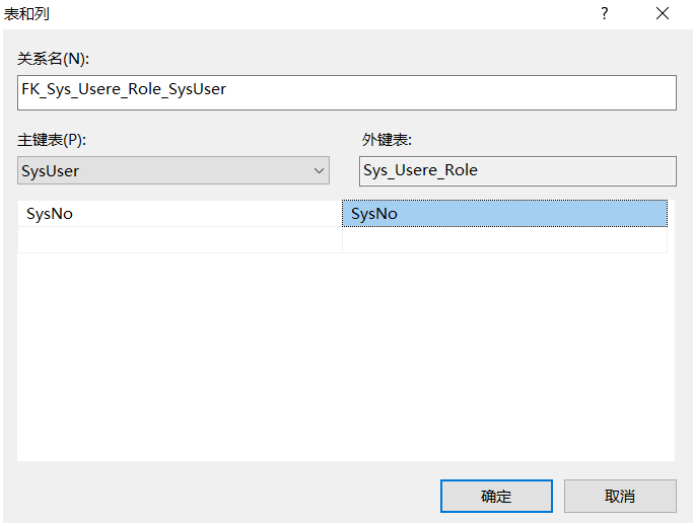
选择关系。




在弹出的对话框中添加关系。




选择主键表和外键表，并指定各自的主键和外键。这样两个表之间就构建了联系。




### 3.5.3.2 SQL 语句方法定义主键和外键

```
alter table sys_usere_role
add constraint FK_user_role foreign key(RoleSysNo) references
sys_role(sysno);
```

## 3.6 一些常用的 SQL 语句

### 3.6.1 新增表记录

新增表记录的时候，我们可以插入单行数据，也可以插入多行数据。

在讲述如何新增表记录之前，先了解一下如何查询表，这是一个基础的 SQL 语句。我们输入以下代码查询表的内容：

```
select * from [dbo].[userinfo];
```

可以发现，当前是一个空表。

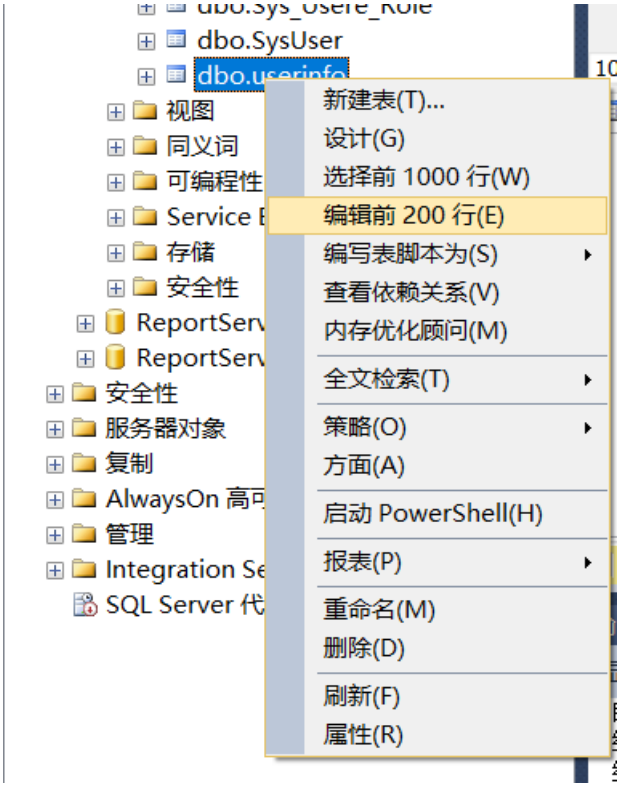
```
select * from [dbo].[userinfo];
```

0 %

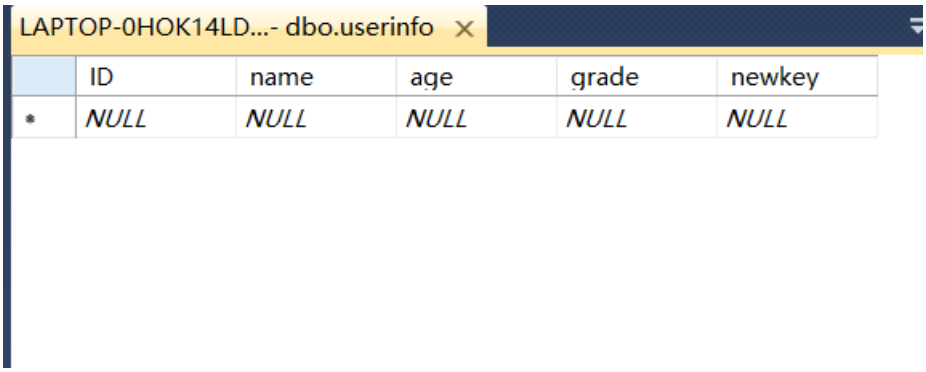
结果 消息

| ID | name | age | grade | newkey |
|---|---|---|---|---|

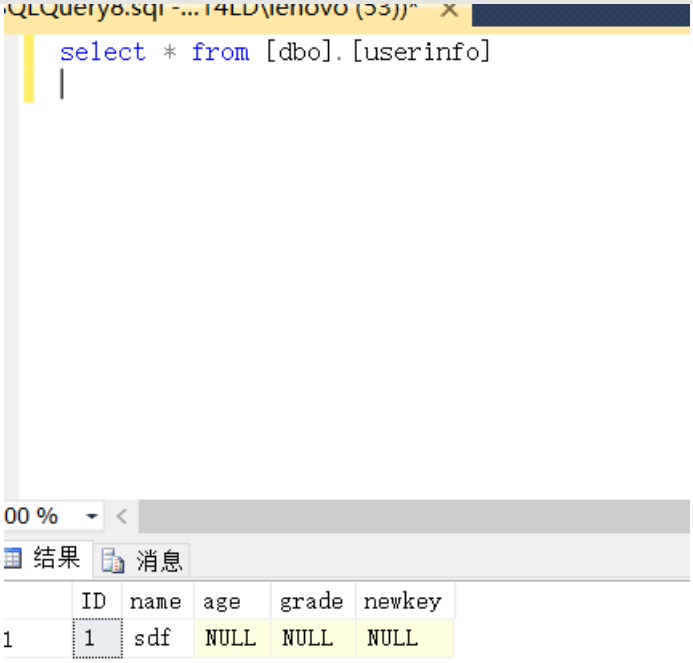
或者还可以从图形界面进行相关的查找。鼠标右键相应的表，选择编辑前 200 行。




可以观察到，当前的表也显示为空。




```
select * from [dbo].[userinfo]
```

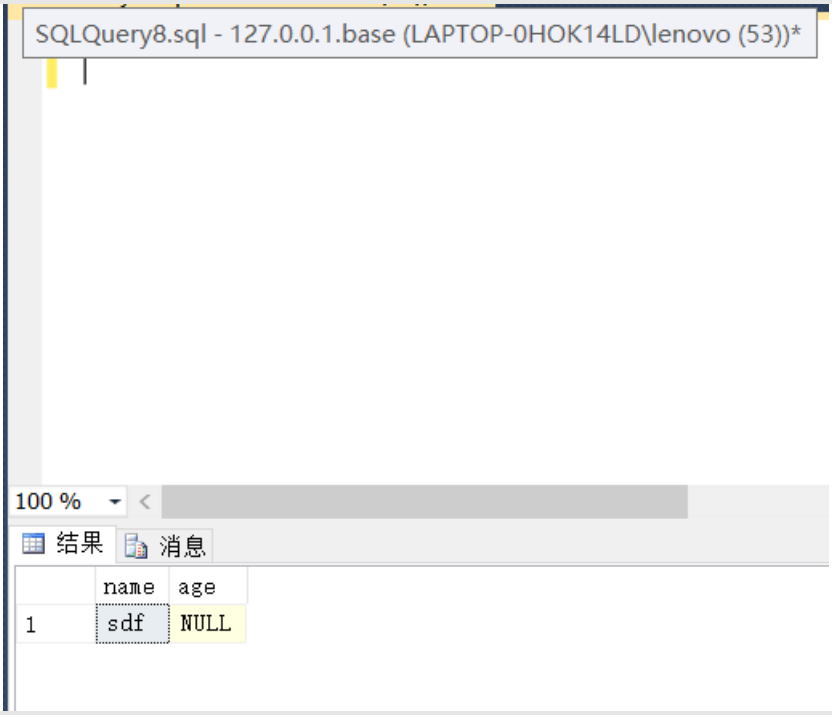



```
select name, age from [dbo].[userinfo]
```

```
select name, age from [dbo].[userinfo]
```


distinct/top 的用法

```
select distinct name, age from [dbo].[userinfo]
```




```
select top 0 name, age from [dbo].[userinfo]
```

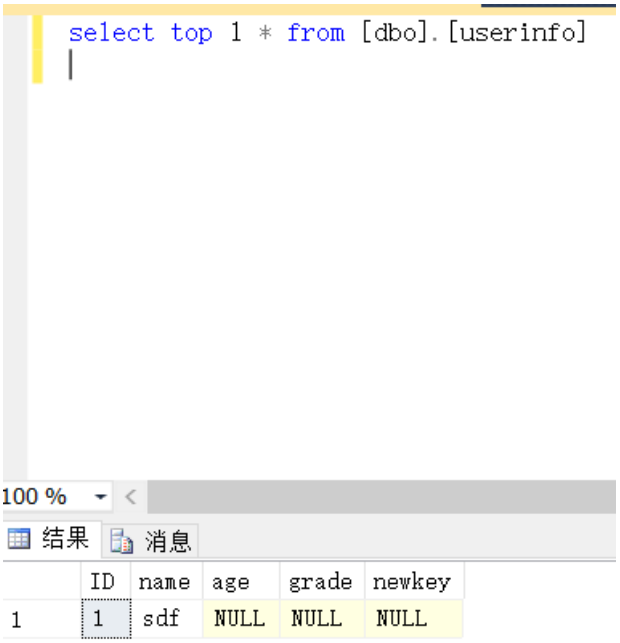
```
select top 0 name, age from [dbo].[userinfo]
```

0 %

结果 消息

| name | age |
|---|---|

```
select top 1 * from [dbo].[userinfo]
```




可以用 SQL 语句进行新增表记录：

```
insert into userinfo(age, grade, ID)
values(15, '一年级', 1031004);
```

注意，要一一对应。

插入多行记录的方法类似，同样需要一一对齐。

```
insert into userinfo(age, grade, ID)
values(15, '一年级', 1031004),
(18, '四年级', 1031005),
(10, '一年级', 1031006),
```

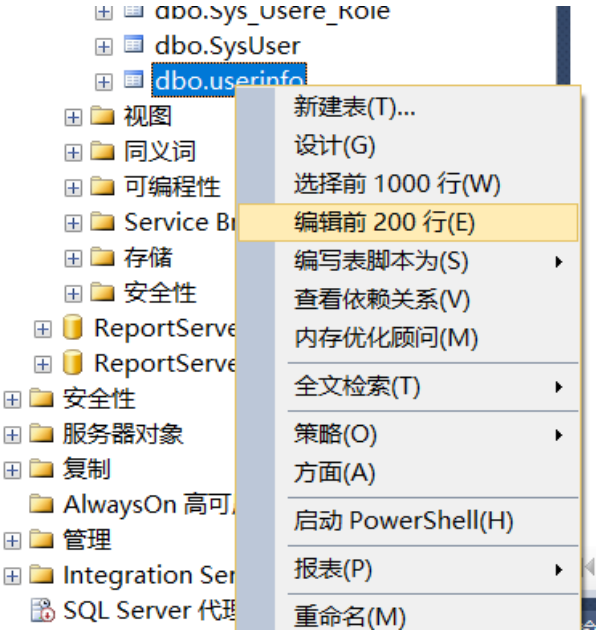
```
(13, '六年级', 1031007),
(90, '二年级', 1031048);
```

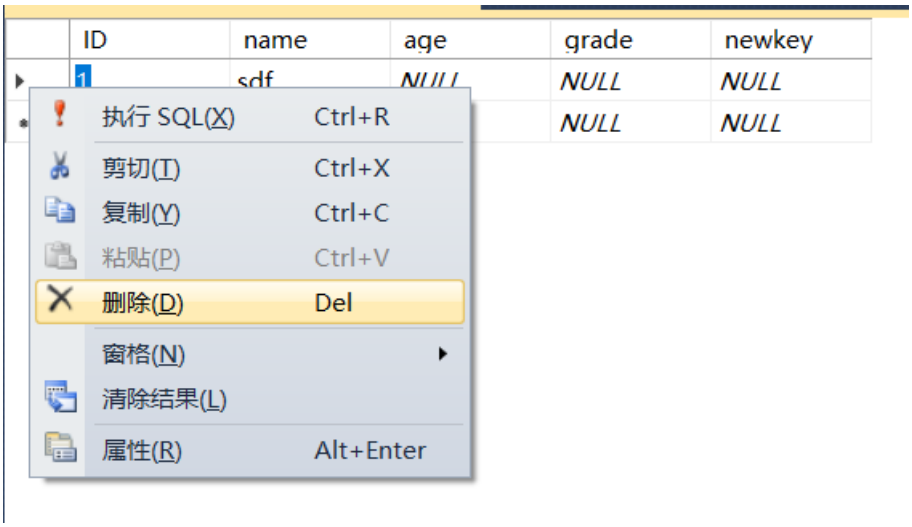
插入其他表的大数据集：

```
insert into userinfo(age, grade, ID)
select age, grade, email from [dbo][bigdata];
```

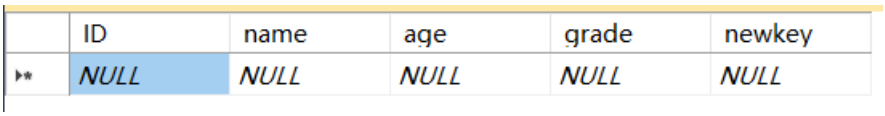
### 3.6.2 SQL Server 删除表记录

选择相应的表格，右键选择“编辑前 200 行”，即可在弹出的窗口中进行表记录的删除操作。




删除完毕：




同样地，也可以使用 SQL 语句进行表记录的删除操作。

```
delete from [dbo].[userinfo]
```

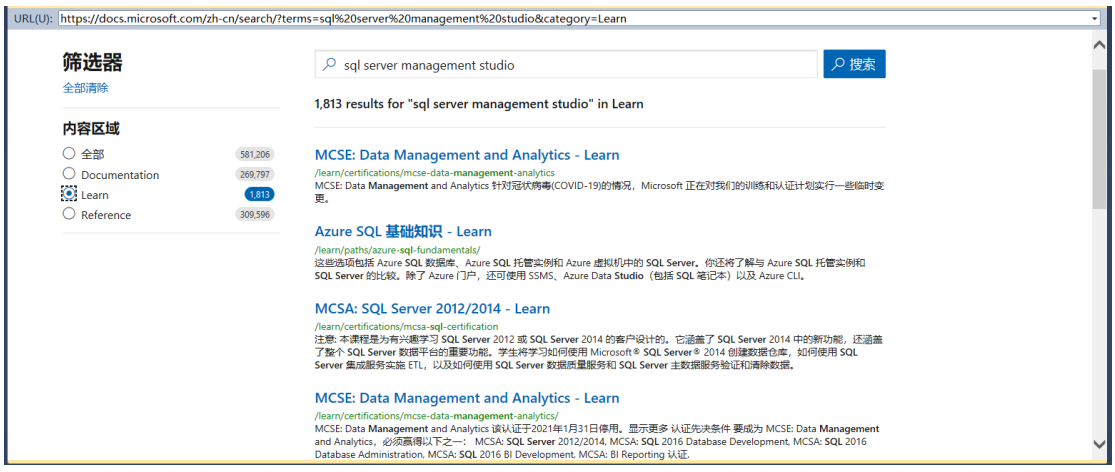
```
where name = 'zhangsan';
```

## 3.7 熟悉在线帮助系统的使用

在 SQL Server Management Studio 窗口中单击“Web 浏览器”，将弹出网络界面。


输入 MSDN 网站的网址，进入微软的在线帮助系统。


可以在搜索栏内搜索相应的问题，在线获取帮助。




可以在线获取很多的帮助文档以及学习的教程。

# Describe types of query plans

100 XP

11 minutes

It is helpful to have a basic understanding of how database optimizers work before taking a deeper dive into execution plan details. SQL Server uses what is known as cost-based query optimizer. The query optimizer calculates a cost for multiple possible plans based on the statistics it has on the columns being utilized, and the possible indexes that can be used for each operation in each query plan. Based on this information, it comes up with a total cost for each plan. Some complex queries can have thousands of possible execution plans. The optimizer does not evaluate every possible plan, but uses heuristics to determine plans that are likely to have good performance. The optimizer will then choose the lowest cost plan of all the plans evaluated for a given query.

Because the query optimizer is cost-based, it is important that it has good inputs for decision making. The statistics SQL Server uses to track the distribution of data in columns and indexes need be kept up to date, or it can cause suboptimal execution plans to be generated. SQL Server automatically updates its statistics as data changes in a table; however, more frequent updates may be needed for rapidly changing data. The engine uses a number of factors when building a plan including compatibility level of the database, row estimates based on statistics and available indexes.

When a user submits a query to the database engine, the following process happens:

1. The query is parsed for proper syntax and a parse tree of database objects is generated if the syntax is correct.
2. The parse tree from Step 1 is taken as input to a database engine component called the Algebrizer for binding. This step validates that columns and objects in the query exist and identifies the data types that are being processed for a given query. This step outputs a query processor tree which is in the input for step 3.
3. Because query optimization is a relatively expensive process in terms of CPU consumption, the database engine

我学到了一些关于 SQL 语句中查询字段的原理和哈希值有关，并且基于此进行查找的优化。

Let's look at an example. Consider the following query:

Transact-SQL 复制

```
SELECT orderdate
 ,avg(salesAmount)
FROM FactResellerSales
WHERE ShipDate = '2013-07-07'
GROUP BY orderdate;
```

In this example SQL Server will check for the existence of the OrderDate, ShipDate, and SalesAmount columns in the table FactResellerSales. If those columns exist, it will then generate a hash value for the query and examine the plan cache for a matching hash value. If there is plan for a query with a matching hash the engine will try to reuse that plan. If there is no plan with a matching hash, it will examine the statistics it has available on the OrderDate and ShipDate columns. The WHERE clause referencing the ShipDate column is what is known as the predicate in this query. If there is a nonclustered index that includes the ShipDate column SQL Server will most likely include that in the plan, if the costs are lower than retrieving data from the clustered index. The optimizer will then choose the lowest cost plan of the available plans and execute the query.