

一、MySQL课程内容

作者：千锋-Mask

版本：1.0

版权：千锋研究院

数据库介绍

- 数据库概念
- 术语介绍

MySQL数据库

- 下载、安装、配置、卸载
- MySQL客户端工具的安装及使用

SQL 结构化查询语言

- 什么是SQL
- SQL操作数据（CRUD操作：添加、查询、修改、删除）

SQL 高级

- 存储过程
- 索引
- 触发器、视图

数据库事务

- 什么是事务
- 事务特性ACID
- 事务隔离级别
- 事务管理

数据库设计

- 数据库设计步骤
- 数据库设计范式
- E-R图

数据库引擎

- MySql体系结构
- 储存引擎介绍
- 常见引擎
- InnoDB逻辑存储结构

SQL优化

- 插入优化
- 排序优化

MySql锁

- 全局锁
- 行级锁
- 表级锁

MySql日志

- 错误日志
- 二进制日志

运维

- 主从复制
- 读写分离

二、数据库介绍

数据库概念

数据库是“按照数据结构来组织、存储和管理数据的仓库”。是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合。数据库软件还提供了高效的增加\删除\修改\查询数据的解决方案

数据库，就是存放数据的仓库

数据库（DataBase，简称DB）是长期存储在计算机内部有结构的、大量的、共享的数据集合。

- 长期存储：持久化
- 有结构：
 - 类型：数据库不仅可以存放数据，而且存放的数据还是有类型的
 - 关系：存储数据与数据之间的关系
- 大量：大多数数据库都是文件系统的，也就是说存储在数据库中的数据实际上就是存储在磁盘的文件中
- 共享：多个应用程序可以通过数据库实现数据的共享

为什么需要用数据库

其实通过io技术对数据进行增删改查,实际上相当于自己写了一个数据库软件.但功能简单，执行效率底下，而且每个项目都开发数据操作相关代码致使开发效率低下，数据库就是当前计算机软件开发行业中，对大量数据管理的通用解决方案，学习数据库就是学习如何和数据库软件进行交流，SQL语言就是程序员和数据库软件进行沟通的语言。

关系型数据库与非关系型数据库

数据库可以分为关系型和非关系型数据库：

- 关系型数据库

关系型数据库，采用了关系模型来组织数据的存储，以行和列的形式存储数据并记录数据与数据之间的关系——将数据存储在表格中，可以通过建立表格与表格之间的关联来维护数据与数据之间的关系。

学生信息---- 学生表

班级信息---- 班级表

- 非关系型数据库

非关系型数据库，采用键值对的模型来存储数据，只完成数据的记录，不会记录数据与数据之间的关系。

在非关系型数据库中基于其特定的存储结构来解决一些大数据应用的难题。

NoSQL(Not only SQL)数据库来指代非关系型数据库。

常见的数据库产品

关系型数据库产品

- MySQL 免费（最常用）

- MariaDB
- Percona Server

- PostgreSQL

- Oracle 收费（常用）

- SQL Server
- Access
- Sybase
- 达梦数据库

非关系型数据库产品

- 面向检索的列式存储 Column-Oriented

- HBase (Hadoop子系统)
- BigTable (Google)

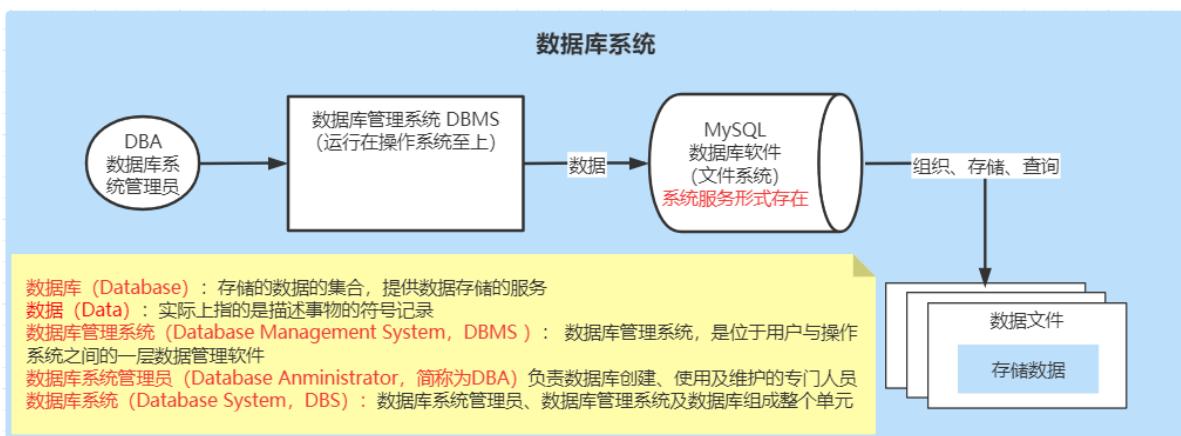
- 面向高并发的缓存存储Key-Value

- Redis
- MemcacheDB

- 面向海量数据访问的文档存储 Document--Oriented

- MongoDB
- CouchDB

数据库术语



- **数据库 (Database)** : 存储的数据的集合, 提供数据存储的服务
- **数据 (Data)** : 实际上指的是描述事物的符号记录
- **数据库管理系统 (Database Management System, DBMS)** : 数据库管理系统, 是位于用户与操作系统之间的一层数据管理软件
- **数据库系统管理员 (Database Administrator, 简称为DBA)** : 负责数据库创建、使用及维护的专门人员
- **数据库系统 (Database System, DBS)** : 数据库系统管理员、数据库管理系统及数据库组成整个单元

三、MySQL数据库环境准备

MySQL下载、安装、配置、卸载，安装DBMS、使用DBMS

MySQL版本及下载

版本

- MySQL 是Oracle公司提供的免费的关系型数据库， 官网 <https://www.mysql.com/>
- MySQL 目前的最新版本为 8.0.x，在企业项目中主流版本： 5.0 --- 5.5 --- 5.6 --- 5.7 --- 8.0.x
 - 5.x --- 常用版本为5.6或者5.7
 - 8.x --- 常用版本为8.0.x
- MySQL 8.x新特性
 - 性能：官方8.x比5.7速度要快2倍
 - 支持NoSQL（非关系型）存储：5.7开始提供了对NoSQL的支持，8.0.x做了更进一步的改进
 - 窗口函数（提供了一种新的查询方式）
 - 索引：隐藏索引、降序索引
 - 可用性、可靠性

下载

- 官网下载：<https://dev.mysql.com/downloads/installer/>
 - 具体步骤：

The screenshot shows the MySQL website's main navigation bar at the top, followed by a large banner for "MySQL HeatWave". The banner highlights "One MySQL Database service for OLTP, OLAP, and ML". Below the banner, there are sections for "HeatWave ML", "Faster Performance", and "Lower Total Cost of Ownership". Buttons for "Try Free" and "Technical Guides" are visible. The main content area contains sections for "MySQL Newsletter", "MySQL Enterprise Edition", "MySQL Cluster CGE", and "MySQL Community Downloads". The "MySQL Community Downloads" section lists various MySQL packages, with "MySQL Installer for Windows" highlighted by a red border.

MySQL HeatWave

One MySQL Database service for OLTP, OLAP, and ML

HeatWave ML

- Native, In-Database Machine Learning
- Build ML Models using SQL Commands
- Automate the ML Lifecycle
- 25x Faster than Redshift ML

Faster Performance

- 5400x Faster than Amazon RDS
- 1400x Faster than Amazon Aurora
- 6.5x Faster than Amazon Redshift AQUA
- 7x Faster than Snowflake

Lower Total Cost of Ownership

- 2/3 the cost of Amazon RDS
- 1/2 the cost of Amazon Aurora
- 1/2 the cost of Amazon Redshift AQUA
- 1/5 the cost of Snowflake

[Try Free](#) | [Technical Guides](#)

MySQL Newsletter

[Subscribe »](#)
[Archive »](#)

Free Webinars

Oracle CloudWorld
Announcements: What's New with MySQL
Tuesday, December 13, 2022
Connecting MySQL HeatWave with External Data Sources with Oracle Data Integration
On Demand

Benefits of MySQL Enterprise Edition
On Demand

More »

Contact Sales

USA: +1-866-221-0634
Canada: +1-866-221-0634

Germany: +49 89 143 01280
France: +33 1 57 60 83 57
Italy: +39 02 249 59 120
UK: +44 207 553 8447

MySQL Enterprise Edition

MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools and technical support for MySQL.

[Learn More »](#)
[Customer Download »](#)
[Trial Download »](#)

MySQL Cluster CGE

MySQL Cluster is a real-time open source transactional database designed for fast, always-on access to data under high throughput conditions.

- MySQL Cluster
- MySQL Cluster Manager
- Plus, everything in MySQL Enterprise Edition

[Learn More »](#)
[Customer Download » \(Select Patches & Updates Tab, Product Search\)](#)
[Trial Download »](#)

MySQL Community Downloads

- MySQL Yum Repository
- MySQL APT Repository
- MySQL SUSE Repository
- MySQL Community Server
- MySQL Cluster
- MySQL Router
- MySQL Shell
- MySQL Operator
- MySQL NDB Operator
- MySQL Workbench
- MySQL Installer for Windows
- MySQL for Visual Studio
- C API (libmysqlclient)
- Connector/C++
- Connector/J
- Connector/NET
- Connector/Node.js
- Connector/ODBC
- Connector/Python
- MySQL Native Driver for PHP
- MySQL Benchmark Tool
- Time zone description tables
- Download Archives

需要注册oracle

服务器在国外，下载速度....

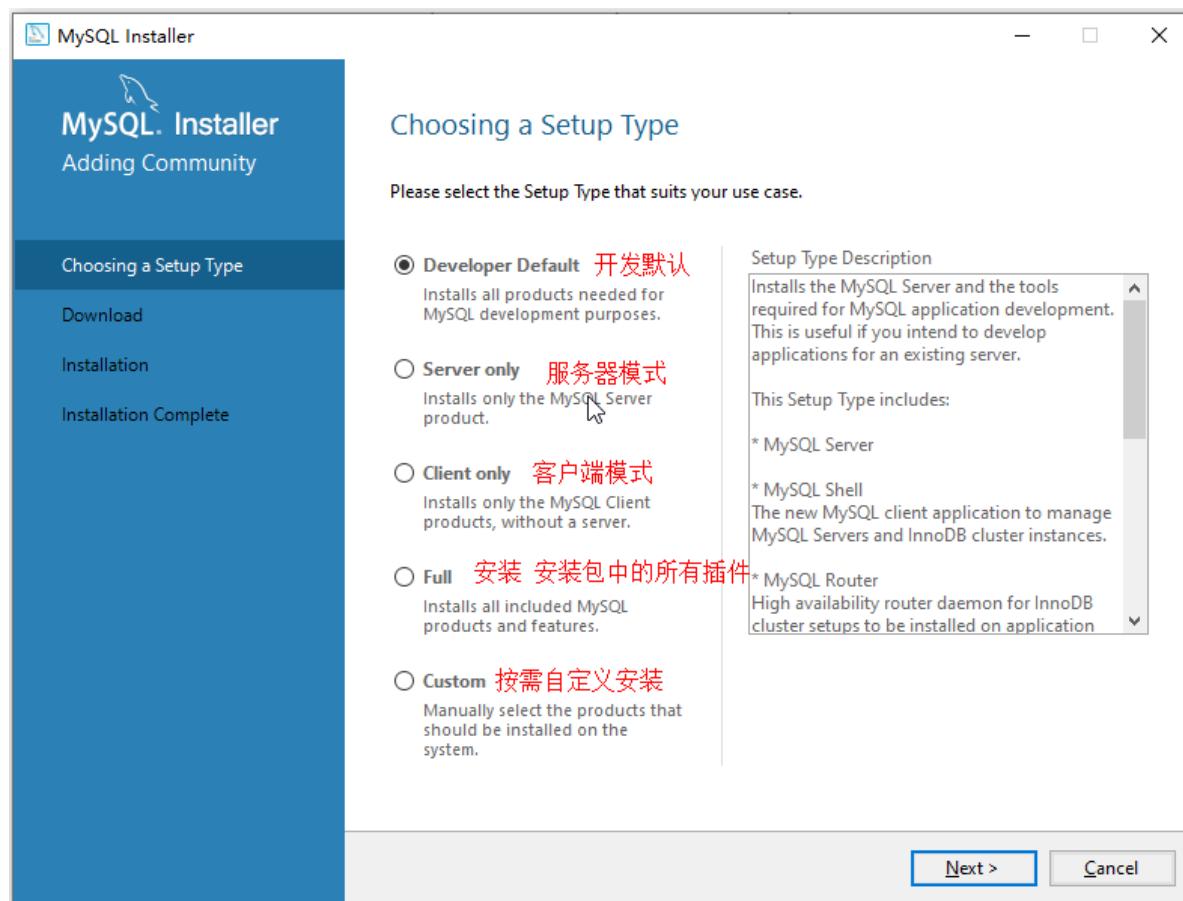
- 镜像下载：<https://www.filehorse.com/download-mysql-64/download/>

The screenshot shows the filehorse.com website's search results page for "mysql". The search bar at the top contains "mysql". Below it, the main content area is titled "Search Results for \"mysql\"". There are four search results listed:

- dbExpress driver for MySQL 8.0.2** - September, 27th 2022 - 10.9 MB - Trial. Description: Provides direct high performance access to MySQL database server. Download button.
- dbForge Studio for MySQL Professional 9.1.21** - August, 31st 2022 - 116 MB - Trial. Description: IDE for MySQL database development, management, and administration. Download button.
- dotConnect for MySQL Professional 9.0.0** - July, 1st 2022 - 74.5 MB - Trial. Description: Development framework with a number of innovative technologies. Download button.
- MySQL 8.0.31.0 (32-bit)** - October, 11th 2022 - 432 MB - Open Source. Description: The world's most popular open-source relational database management system. Download button.
- MySQL 8.0.31.0 (64-bit)** - October, 11th 2022 - 432 MB - Open Source. Description: The world's most popular open-source relational database management system. Download button.

A red box highlights the first two results: the dbExpress driver and dbForge Studio.

MySQL 安装



选择 Developer Default 模式安装

此模式MySQL会安装它认为的开发人员需要的常用组件；在安装这些组件时需要对应的环境依赖，我们要暂停，先去安装依赖的环境：

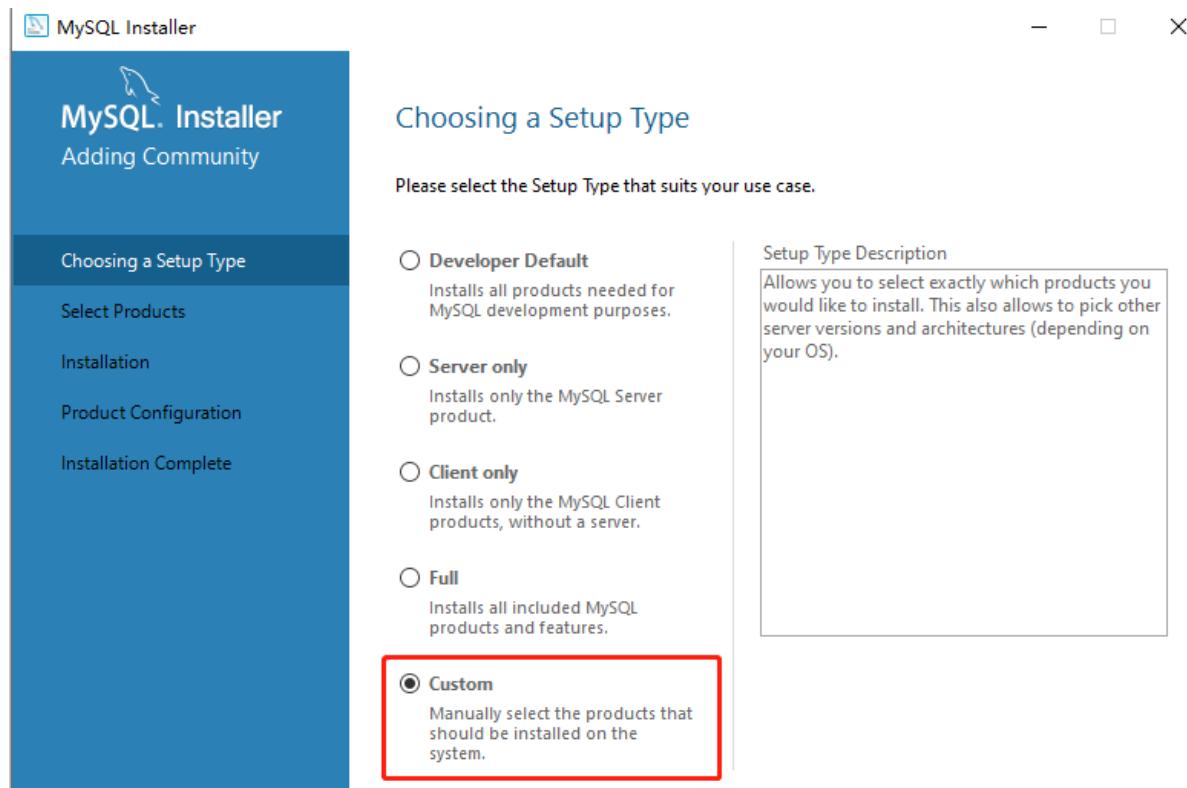
例如：Microsoft Visual C++ 2019 Redistributable Package (x64) is not installed. Latest binary compatible version will be installed if agreed to resolve this requirement.

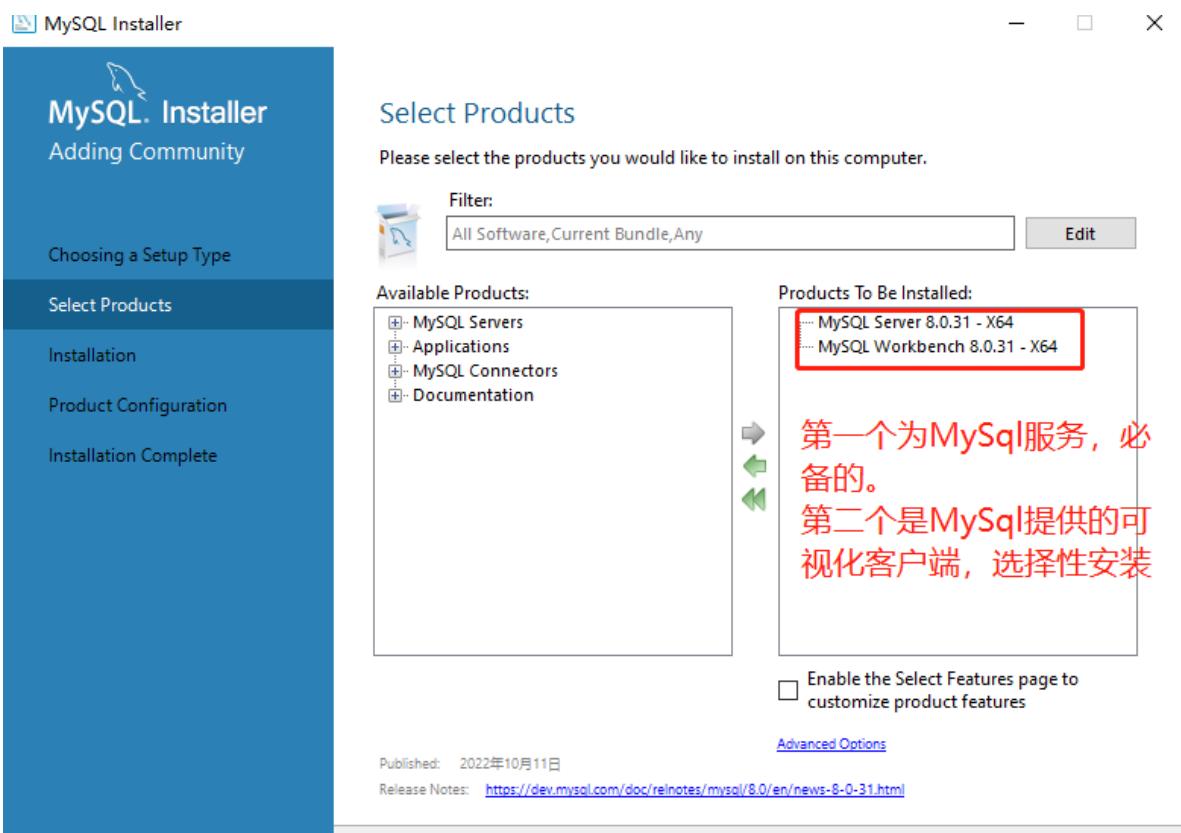
安装：



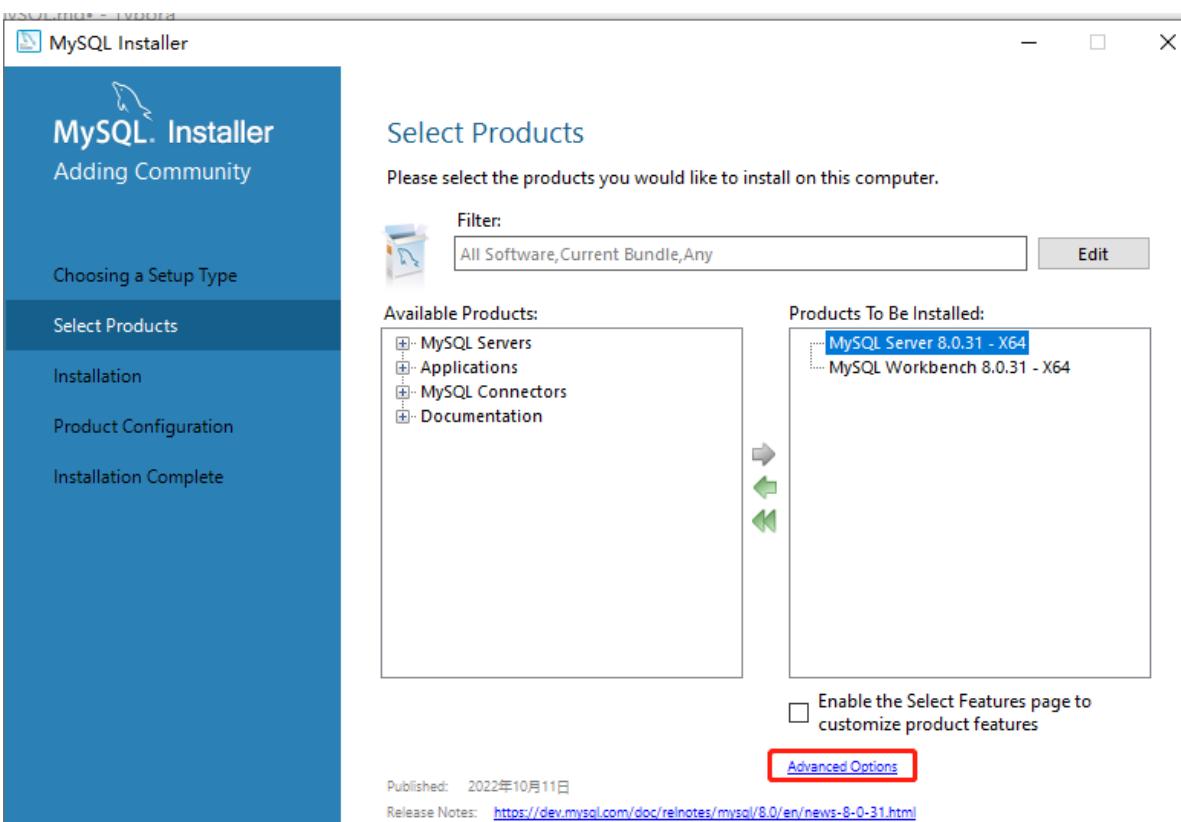
注意：这样的安装方式比较麻烦，而且容易出现问题，所以我们可以选择自定义安装

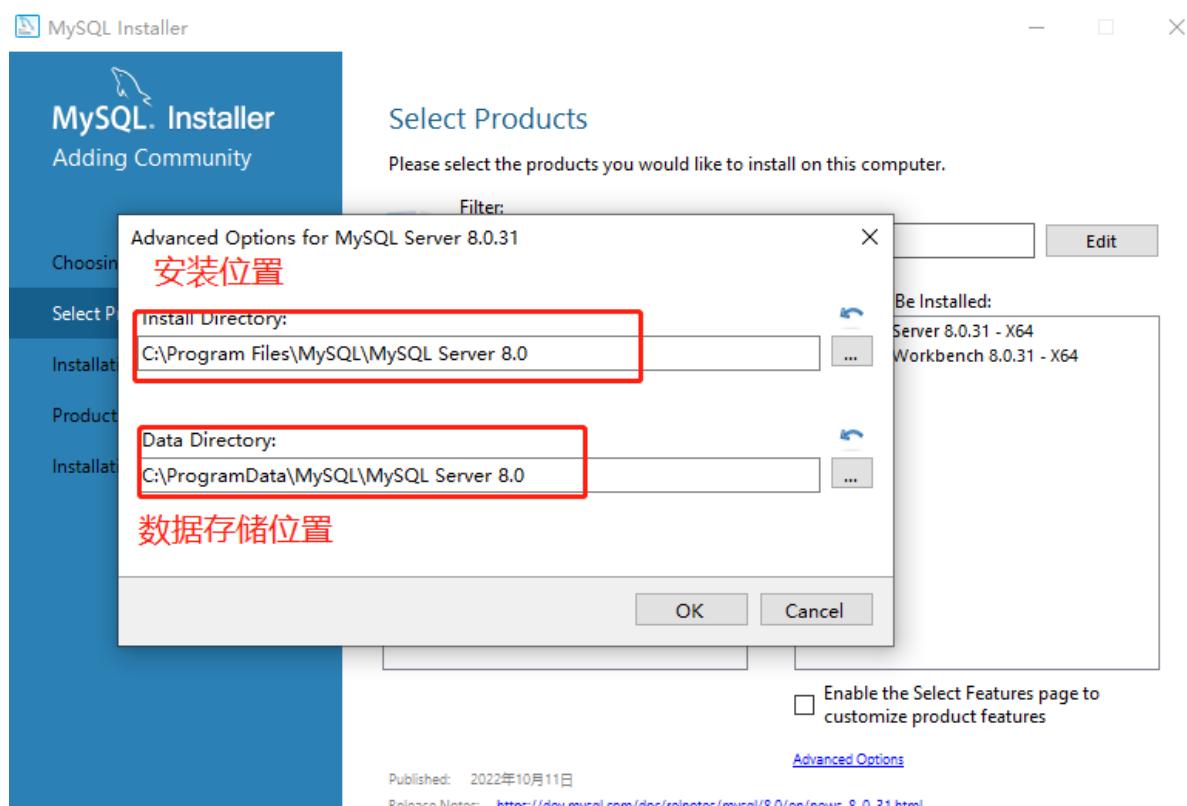
选择自定义 Custom 安装



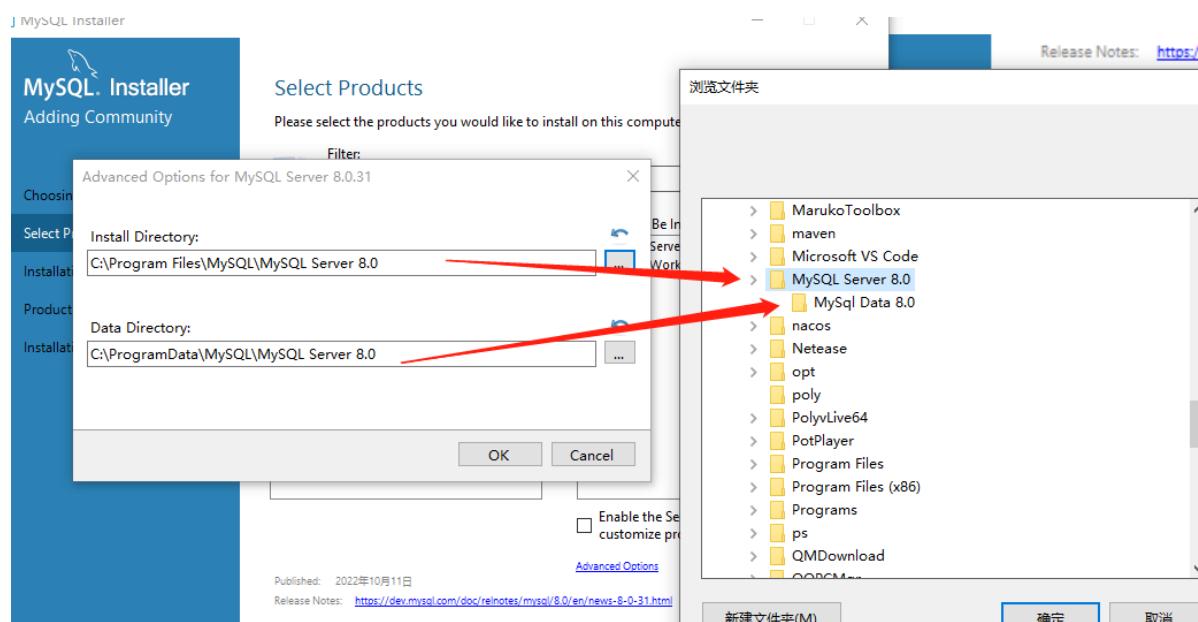


如果想更改MySQL的安装位置，可以在选择具体安装内容以后，选择以下选项更改安装位置





以下是我的安装路径

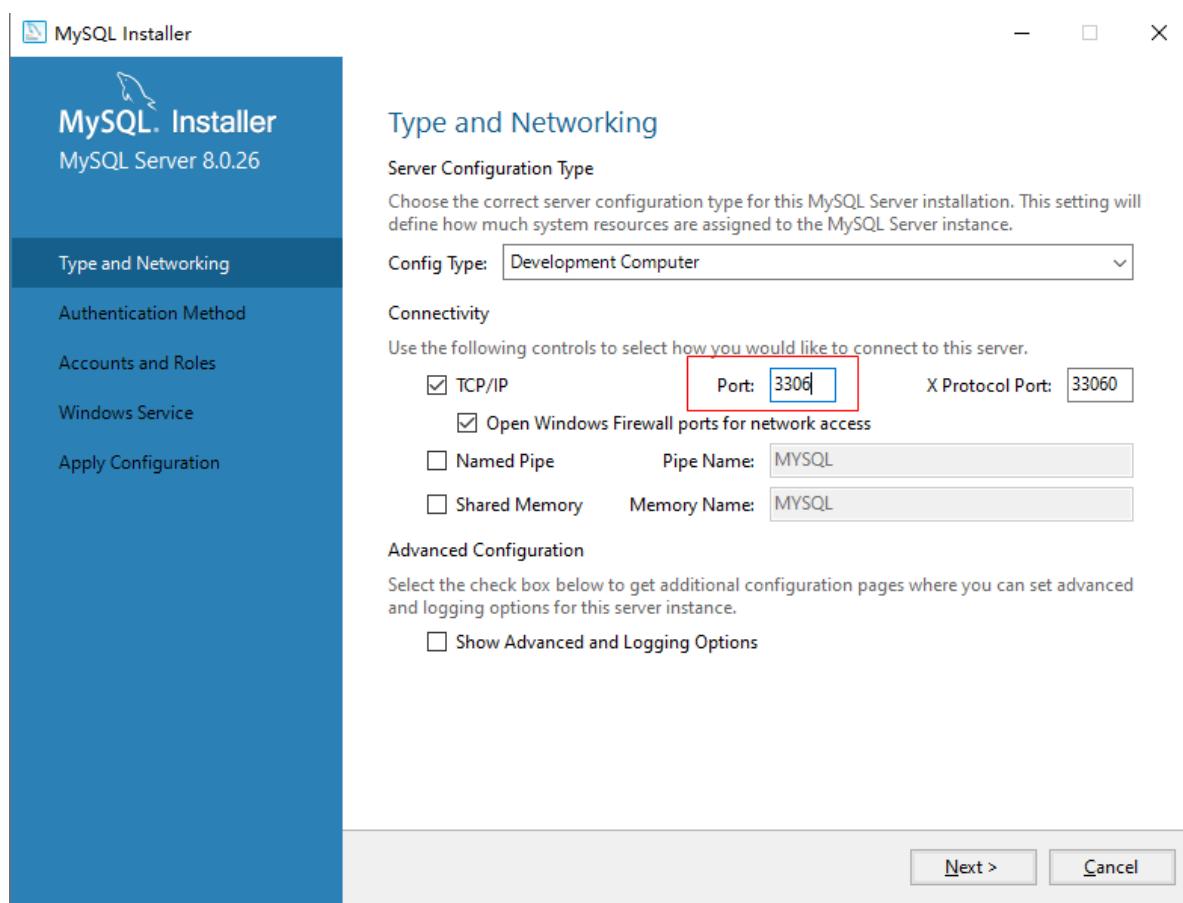


这里点击ok会弹出提示，直接选择OK即可，再往下选择Next时也会弹出提示，选择YES即可

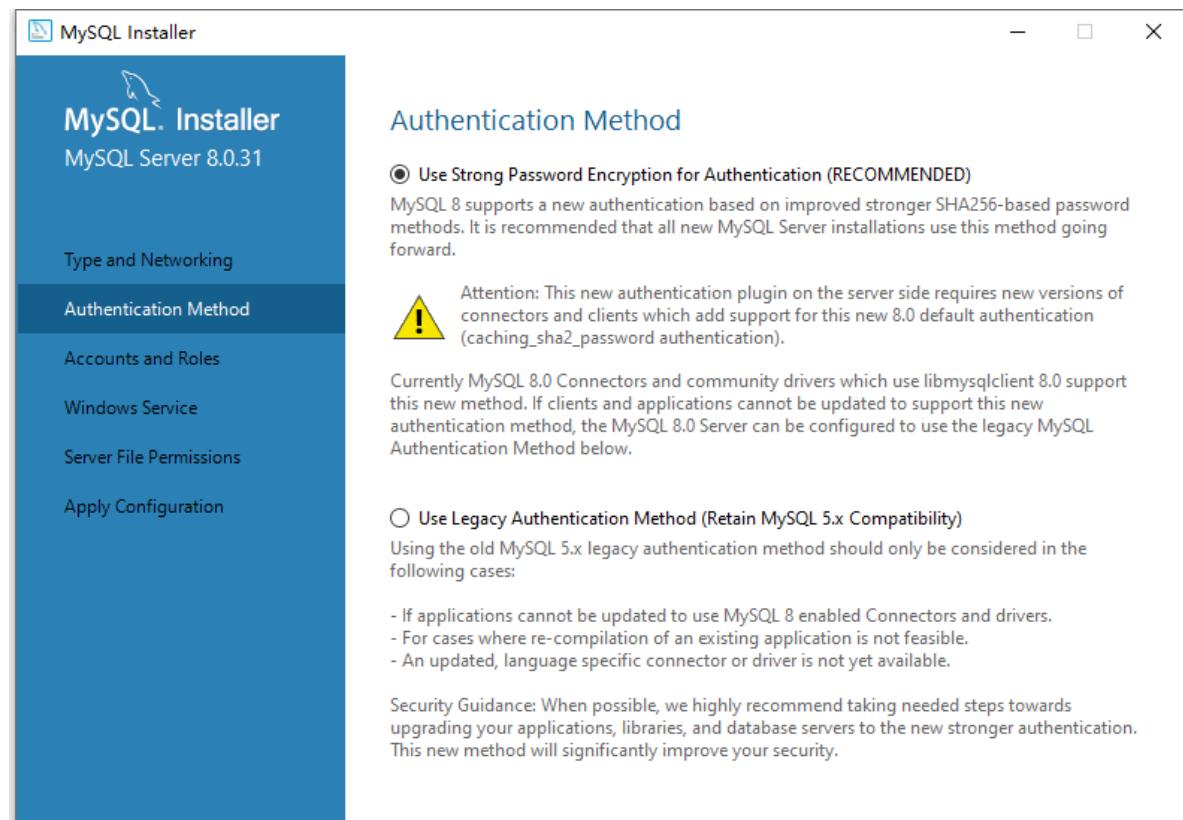
MySQL配置

端口配置

注意：端口号一般情况下默认（3306）即可，但是如果被占用那么就要修改端口号（老师本人电脑有很多数据库，所以这里我将端口号改为3307）



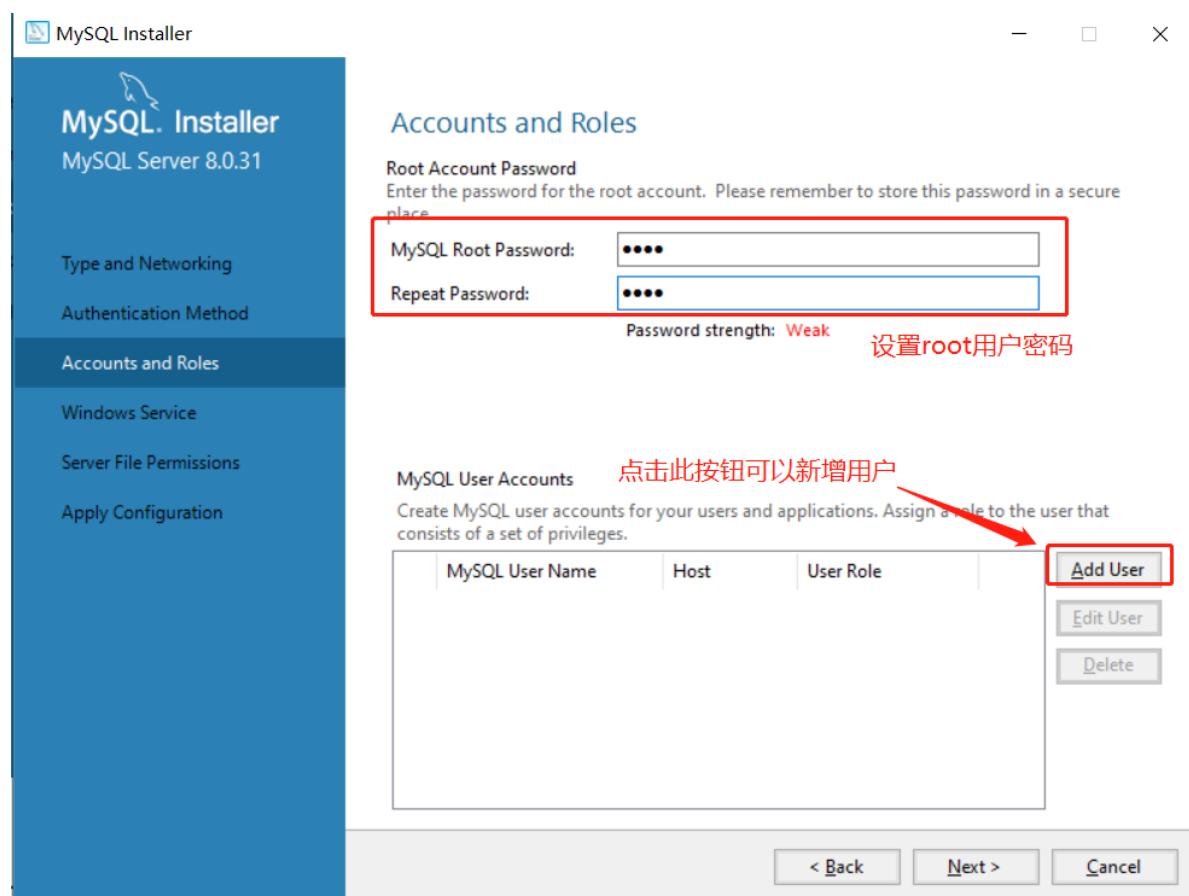
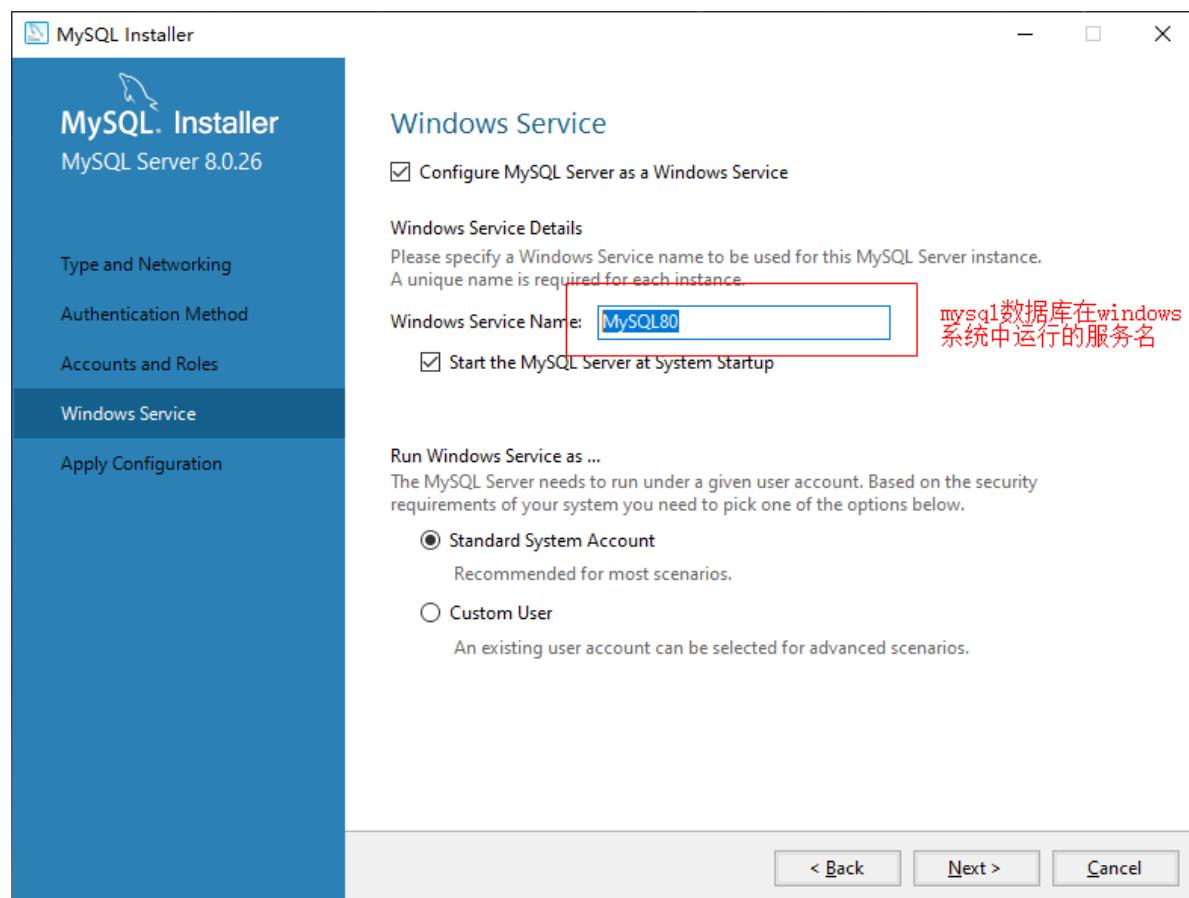
点击Next以后，以下页面默认即可

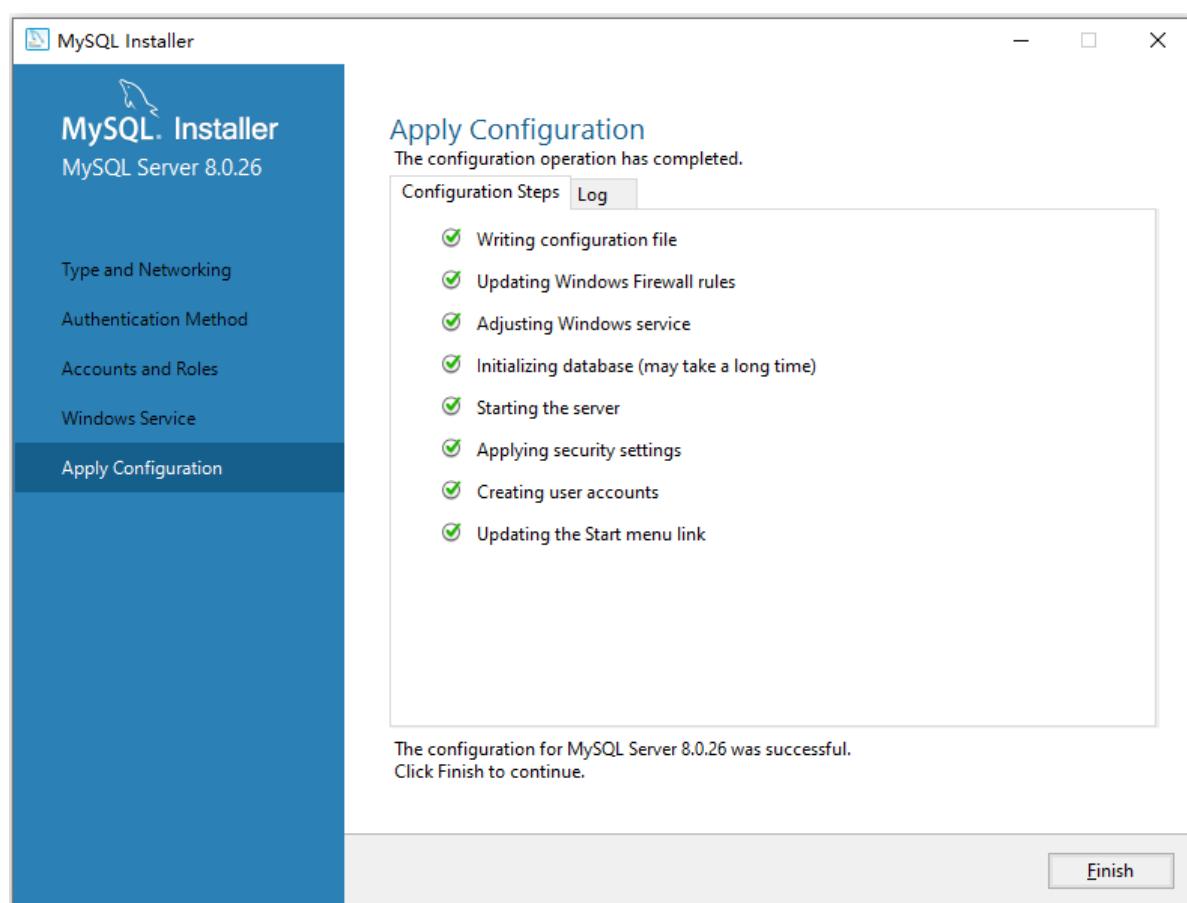
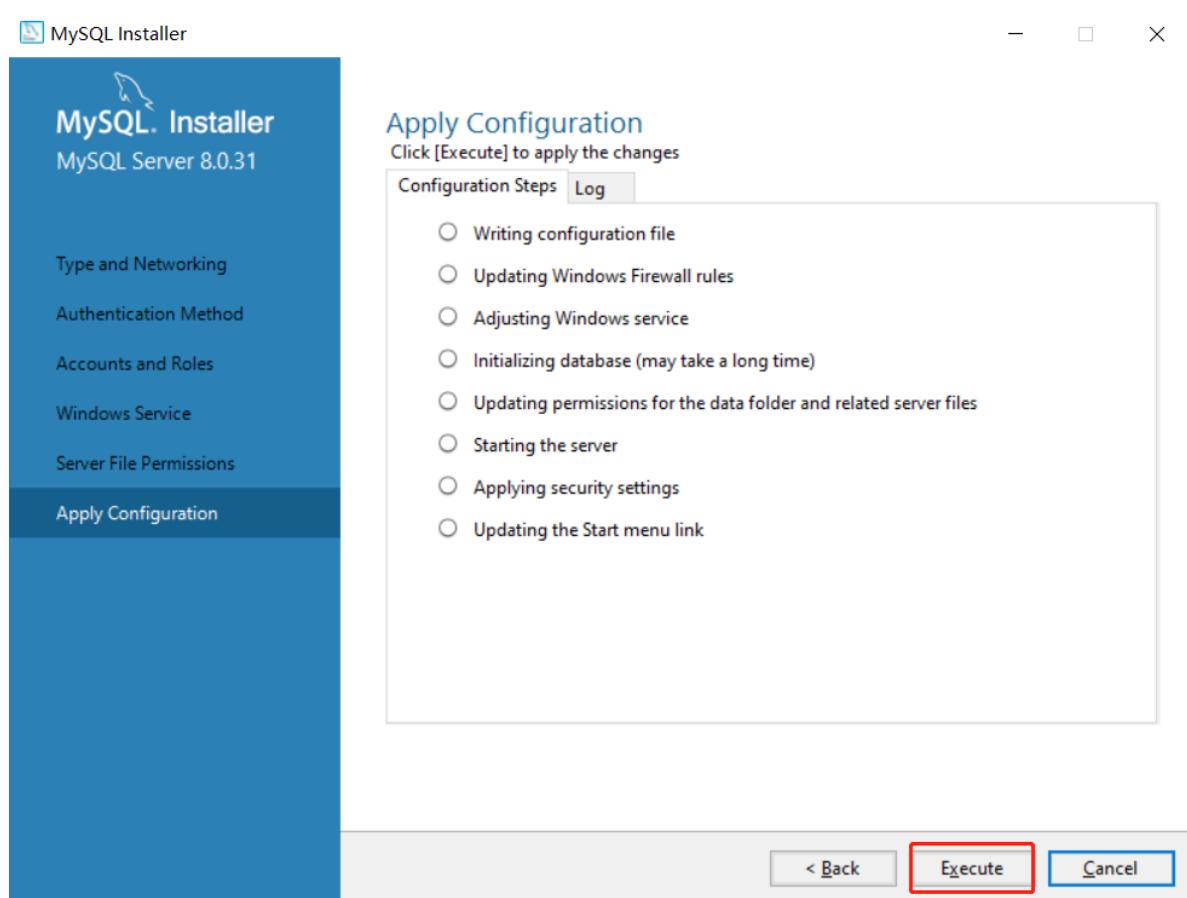


账号密码设置

密码设置可以自行决定，如果密码强度过低，会出现警告，当然也可以设置，比如我设置的密码为：root。

注意：密码设置之后一定不要忘记，否则很麻烦

**服务名称**

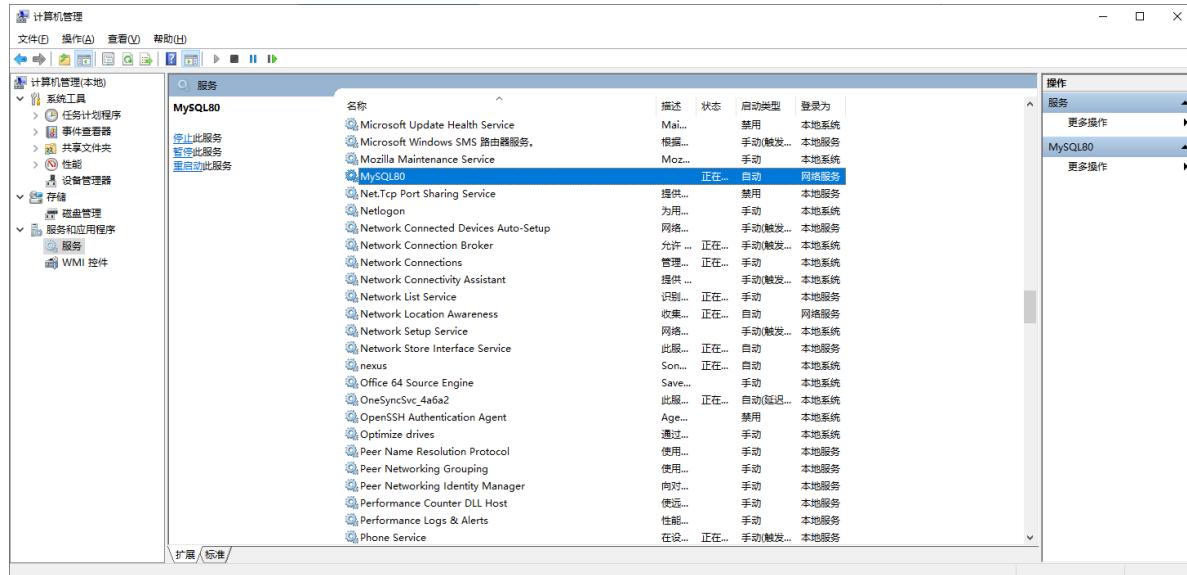


MySQL 服务的启动与停止

MySQL是以服务的形式运行在系统中

计算机管理窗口

此电脑 --- 右键 --- 管理



windows命令行

打开命令行： `win + R` --- 输入 `cmd` 回车

以管理员身份打开命令行： `win+s` ---- 输入 `cmd` ----选择 以管理员身份运行

```
管理员: 命令提示符
Microsoft Windows [版本 10.0.19043.1165]
(c) Microsoft Corporation。保留所有权利。

C:\WINDOWS\system32>net stop mysql80
MySQL80 服务正在停止。
MySQL80 服务已成功停止。

C:\WINDOWS\system32>net start mysql80
MySQL80 服务正在启动。
MySQL80 服务已经启动成功。

C:\WINDOWS\system32>
```

MySQL卸载

- 关闭服务

```
## 管理员身份启动 cmd 命令行
net stop MySQL80
```

- 卸载软件

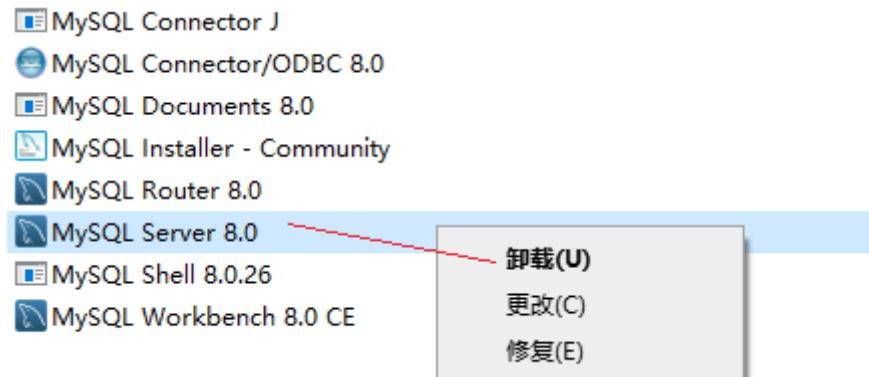
- 打开控制面板



- 点击“程序和功能”



- 卸载MySQL



- 删除目录

- MySQL的安装目录: C:\Program Files (x86)\MySQL (默认)
- MySQL的数据文件目录(默认隐藏): C:\ProgramData\MySQL (默认, 如果不允许删除, 强制删除)

- 删除注册表

- 打开注册表: win+r --- 输入 regedit --- 回车
- 删除 HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\MySQL80 (新版的 MySQL卸载之后, 会自动删除)
- 删除搜索 mysql 的相关项 (非必须)

四、MySQL的管理工具

当完成数据库的安装之后, mysql是以服务的形式运行在windows/linux系统, 用户是通过DBMS工具来对MySQL进行操作的, 当我们安装完成MySQL之后默认安装了mysql Command line client, 此工具是一个命令行形式的工具, 通常我们会单独安装可视化的DBMS工具:

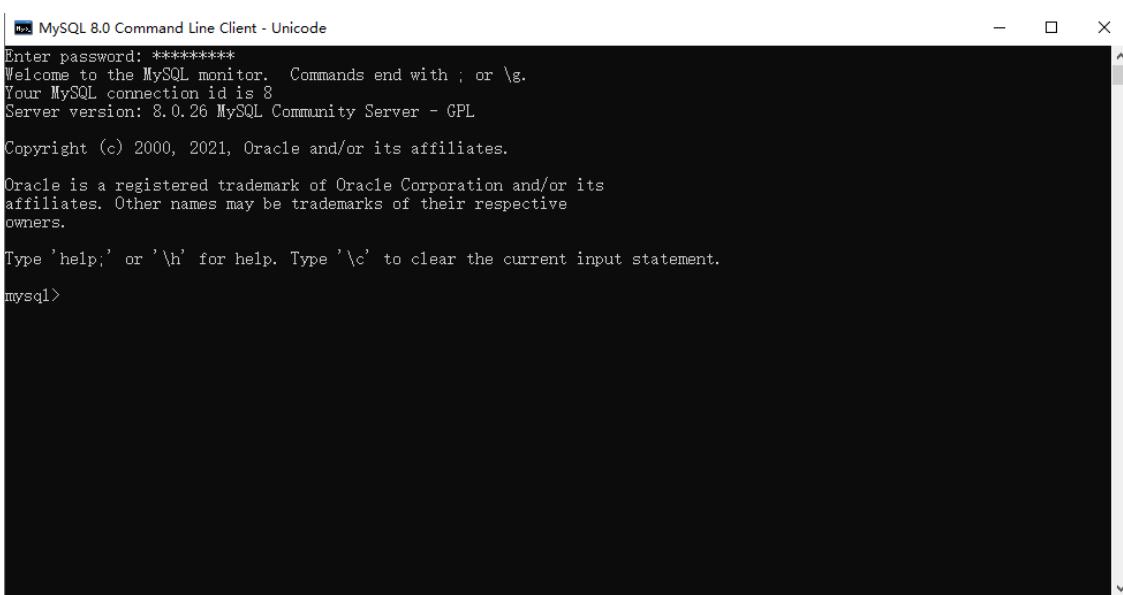
常用工具比如 (以下工具都是收费的, 如果想要使用请自行购买) :

- SQLyog
- Navicat for MySQL

当然这里也有免费的DBMS工具, 比如DBeaver

MySQL Command line Client使用

- 打开MySQL Command line Client: 开始菜单 --- MySQL --- MySQL 8.0 Command line Client
- 连接MySQL: 输入密码 即可 (如果密码错误或者mysql服务没有启动, 窗口会闪退)



- 关闭MySQL Command line Client：输入`exit`指令回车即可退出

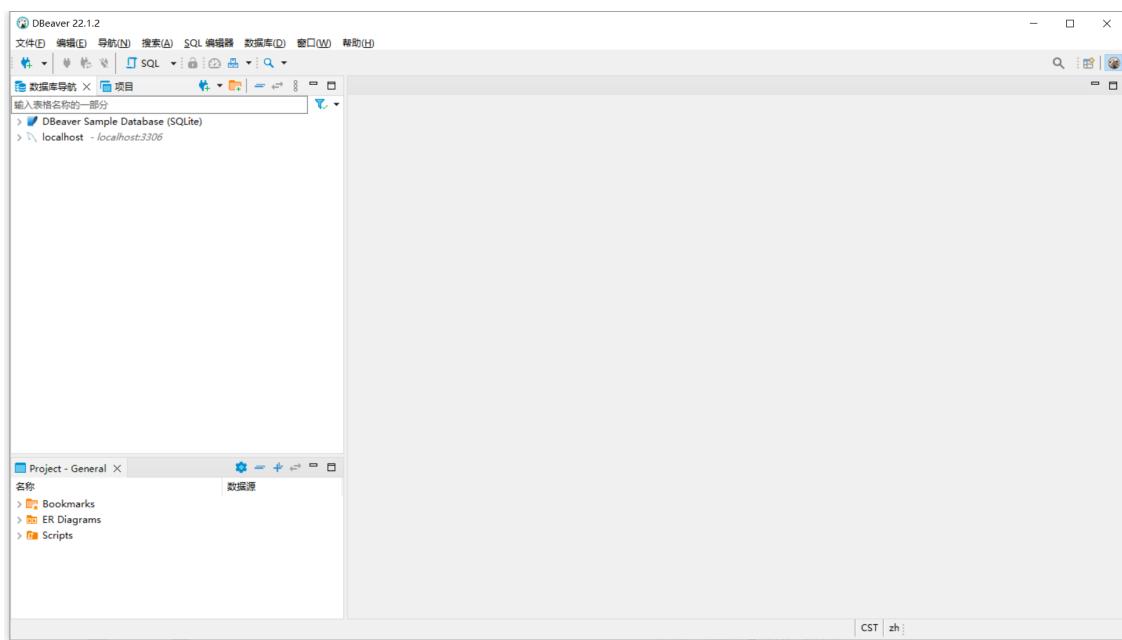
可视化工具DBeaver使用

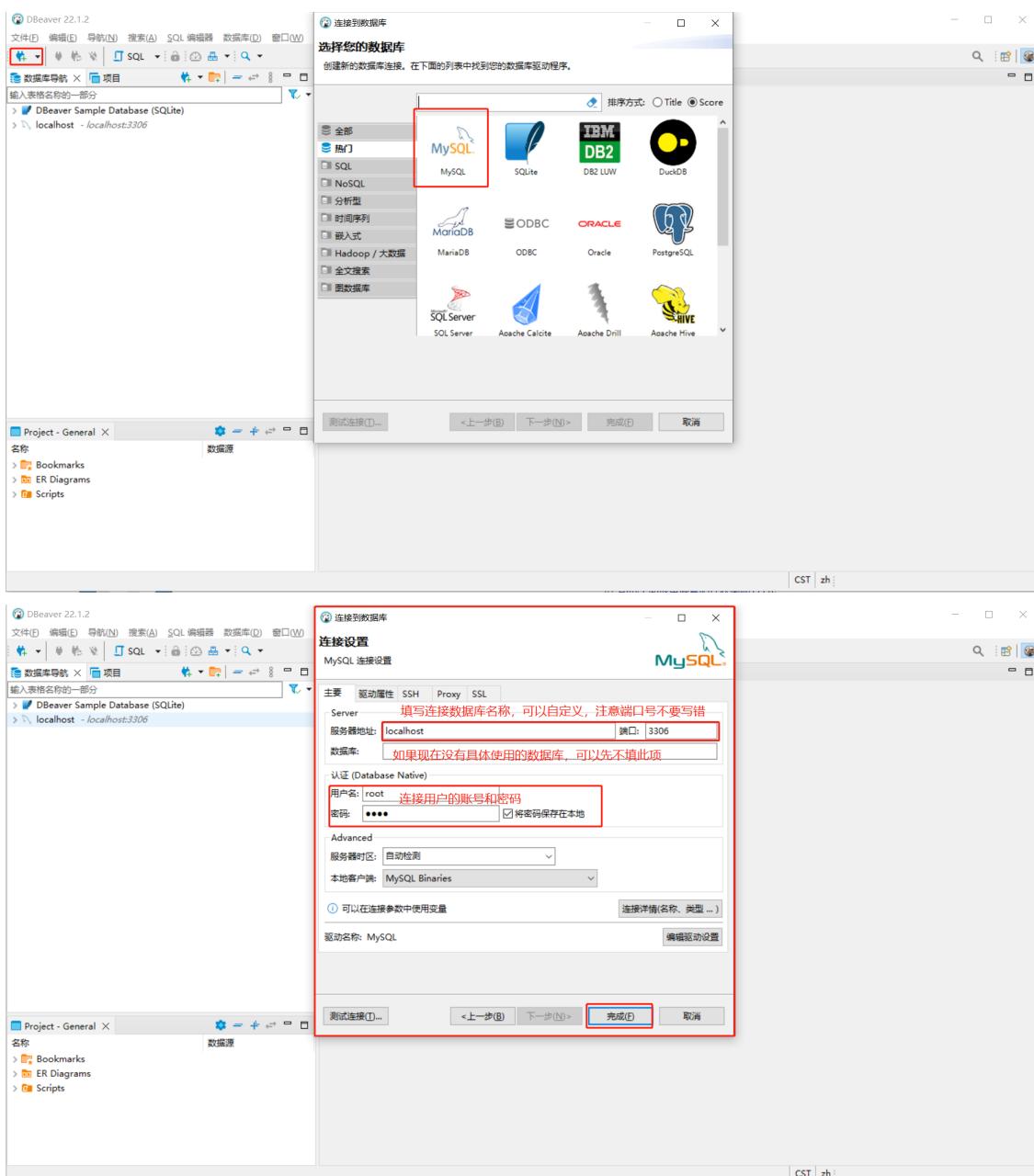
DBeaver工具下载及安装

傻瓜式安装

创建连接

- 打开DBeaver工具
- 创建连接：





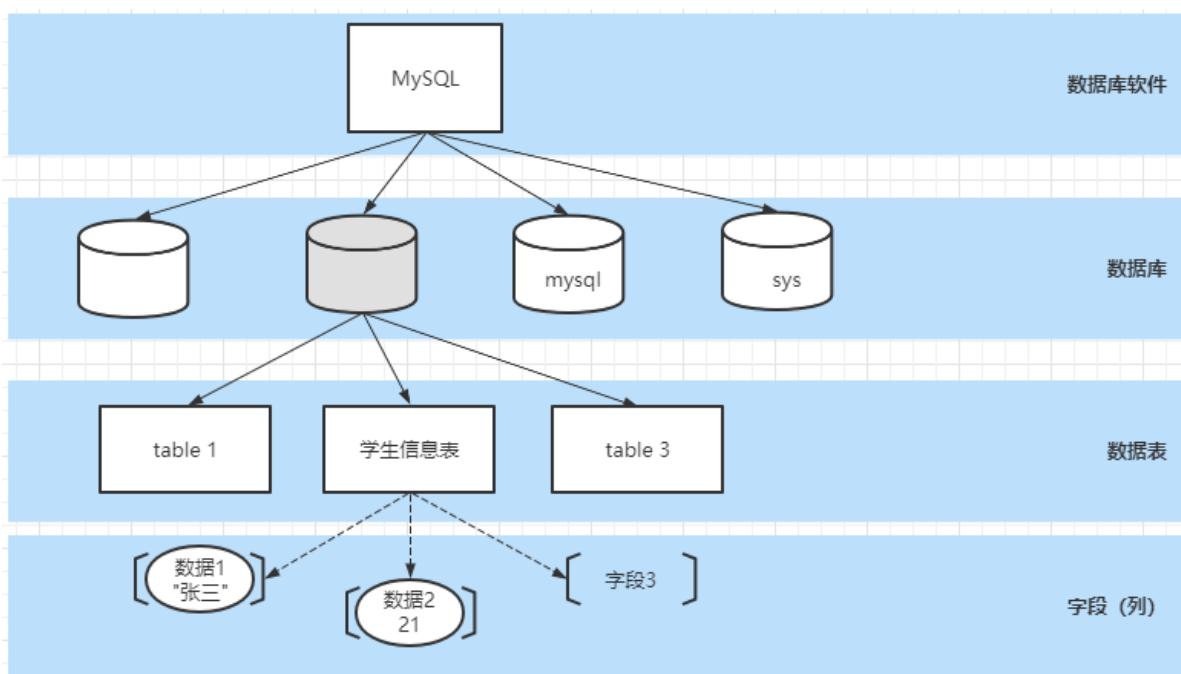
五、MySQL逻辑结构

MySQL可以存储数据，但是存储在MySQL中的数据需要按照特定的结构进行存储

学生 ----- 学校

数据 ----- 数据库

逻辑结构



记录/元组

学号	姓名	性别	年龄	身高	体重	地址
101	张三	男	21	177	65	湖北武汉
102	李四	女	20	165	45	湖北武汉

表格中的行——一条记录（元组）

六、SQL 结构化查询语言

SQL概述

SQL (Structured Query Language) 结构化查询语言，用于存取、查询、更新数据以及管理关系型数据库系统

SQL发展

- SQL是在1981年由IBM公司推出，一经推出基于其简洁的语法在数据库中得到了广泛的应用，成为主流数据库的通用规范
- SQL由ANSI组织确定规范
- 在不同的数据库产品中遵守SQL的通用规范，但是也对SQL有一些不同的改进，形成了一些数据库的专有指令
 - MySQL: limit
 - SQLServer : top
 - Oracle: rownum

SQL分类

根据SQL指令完成的数据库操作的不同，可以将SQL指令分为四类：

- **DDL Data Definition Language 数据定义语言**
 - 用于完成对数据库对象（数据库、数据表、视图、索引等）的创建、删除、修改
- **DML Data Manipulation Language 数据操作/操纵语言**
 - 用于完成对数据表中的数据的添加、删除、修改操作
 - 添加：将数据存储到数据表
 - 删除：将数据从数据表移除
 - 修改：对数据表中的数据进行修改
- **DQL Data Query Language 数据查询语言**
 - 用于将数据表中的数据查询出来
- **DCL Data Control Language 数据控制语言**
 - 用于完成事务管理等控制性操作

SQL基本语法

在MySQL Command Line Client 或者navicat等工具中都可以编写SQL指令

- SQL指令不区分大小写
- 每条SQL表达式结束之后都以 ; 结束
- SQL关键字之间以 空格 进行分隔
- SQL之间可以不限制换行（可以有空格的地方就可以有换行）

DDL 数据定义语言

DDL-数据库操作

使用DDL语句可以创建数据库、查询数据库、修改数据库、删除数据库

查询数据库

```
## 显示当前mysql中的数据库列表
show databases;

## 显示指定名称的数据库创建的SQL指令
show create database db_test;
```

创建数据库

```

## 创建数据库 db_test表示创建的数据库名称，可以自定义
create database db_test ;

## 创建数据库，当指定名称的数据库不存在时执行创建
create database if not exists db_test;

## 在创建数据库的同时指定数据库的字符集（字符集：数据存储在数据库中采用的编码格式
utf8 gbk）
create database db_test character set utf8;

```

修改数据库 修改数据库字符集

```

## 修改数据库的字符集
alter database db_test character set utf8; # utf8 gbk

```

删除数据库 删除数据库时会删除当前数据库中所有的数据表以及数据表中的数据

```

## 删除数据库
drop database db_test;

## 如果数据库存在则删除数据库
drop database if exists db_test;

```

使用/切换数据库

```
use db_test
```

DDL-数据表操作

创建数据表

数据表实际就是一个二维的表格，一个表格是由多列组成，表格中的每一类称之为表格的一个字段

stu_num char(8) not null	stu_name varchar(20) not null	stu_gender char(2) not null	stu_age int not null	stu_tel char(11) not null unique	stu_qq varchar(11) unique	
20210101	张三	男	20	13030303300	null	
20210102	李四	女	20	13030303300		

```
create table students(
    stu_num char(8) not null unique,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null,
    stu_tel char(11) not null unique,
    stu_qq varchar(11) unique
);
```

在数据库中创建列名如果需要多个单词，请用_来区分，因为SQL语句不区分大小写

在小括号中定义列名，需要通过,分割，切记最后一个没有逗号

char(8): 不可变字符串（定长，如果字符串长度不够，会自动补0）

varchar(20): 可变字符串（一个字符占2个字节）

null not: 约束不能为空

unique: 约束不可重复

```
create table students(
    stu_num char(8) not null unique,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null,
    stu_tel char(11) not null unique,
    stu_qq varchar(11) unique
);
```

查询全部数据表

```
show tables;
```

查询表结构

```
desc students;
```

删除数据表

```
## 删除数据表
drop table students;

## 当数据表存在时删除数据表
drop table if exists students;
```

修改数据表

修改表先关内容的关键字为**alter**

```
## 修改表名
alter table <tableName> rename to <newTableName>;

## 数据表也是有字符集的，默认字符集和数据库一致，当然也可以通过以下语句进行修改
alter table <tableName> character set utf8;

## 添加列（字段）
alter table <tableName> add <columnName> varchar(200);

## 修改列（字段）的列表和类型
alter table <tableName> change <oldColumnName> <newColumnName> <type>;

## 只修改列（字段）类型
alter table <tableName> modify <columnName> <newType>;

## 删除列（字段）
alter table <tableName> drop <columnName>;
```

MySQL数据类型

数据类型，指的是数据表中的列中支持存放的数据的类型

数值类型

在mysql中有多种数据类型可以存放数值，不同的类型存放的数值的范围或者形式是不同的

注意：前三种数字类型我们在实际研发中心用的很少，一般整数类型我们会直接使用int/integer，如果过大会使用bigint，同样浮点数类型默认会使用double

类型	内存空间大小	范围	说明
tinyint	1byte	有符号 -128~127 无符号 0~255	特小型整数（年龄）
smallint	2byte (16bit)	有符号 -32768 ~ 32767 无符号 0~65535	小型整数
mediumint	3byte	有符号 -2^31 ~ 2^31 - 1 无符号 0~2^32-1	中型整数
int/integer	4byte		整数
bigint	8byte		大型整数
float	4byte		单精度
double	8byte		双精度
decimal	一般情况：第一参数 +2		decimal(10,2) 表示数值一共有10位 小数位有2位

字符串类型

存储字符序列的类型

注意：在数据库中存储图片或者视频音频等内容，一般是存储，文件在服务器上的路径，当然如果非要存储就需要将图片等数据转成二进制进行存储，所以blob类型是可以存储所有类型的，但是前提是需要转换成二进制，所以此类型用的很少。

longtext类型一般用于varchar类型储存不下的时候。

常用类型也就是char和varchar

类型	字符长度	说明

类型	字符长度	说明
char	0~255 字节	定长字符串，最多可以存储255个字符；当我们指定数据表字段为char(n)此列中的数据最长为n个字符，如果添加的数据少于n，则补'\u0000'至n长度
varchar	0~65536 字节	可变长度字符串，此类型的类最大长度为65535
tinyblob	0~255 字节	存储二进制字符串
blob	0~65535	存储二进制字符串
mediumblob	0~1677215	存储二进制字符串
longblob	0~4294967295	存储二进制字符串
tinytext	0~255	文本数据（字符串）
text	0~65535	文本数据（字符串）
mediumtext	0~1677215	文本数据（字符串）
longtext	0~4294967295	文本数据（字符串）

日期类型

在MySQL数据库中，我们可以使用字符串来存储时间，但是如果我们需要基于时间字段进行查询操作（查询在某个时间段内的数据）就不便于查询实现

类型	格式	说明
date	2021-09-13	日期，只存储年月日
time	11:12:13	时间，只存储时分秒
year	2021	年份
datetime	2021-09-13 11:12:13	日期+时间，存储年月日时分秒
timestamp	20210913 111213	日期+时间（时间戳）

字段约束

约束介绍

在创建数据表的时候，指定的对数据表的列的数据限制性的要求（对表的列中的数据进行限制）

为什么要给表中的列添加约束呢？

- 保证数据的有效性
- 保证数据的完整性
- 保证数据的正确性

字段常见的约束有哪些呢？

- 非空约束 (not null)：限制此列的值必须提供，不能为null
- 唯一约束 (unique)：在表中的多条数据，此列的值不能重复
- 主键约束 (primary key)：非空+唯一，能够唯一标识数据表中的一条数据
- 外键约束 (foreign key)：建立不同表之间的关联关系

非空约束

限制数据表中此列的值必须提供

- 创建表：设置图书表的 book_name not null

```
create table books(
    book_isbn char(4),
    book_name varchar(10) not null,
    book_author varchar(6)
);
```

- 添加数据：

book_isbn	book_name	book_author
1001	Java	老王
(Null)	C++	老李头
(Null)	Python	(Null)
1004	(Null)	老张

在添加数据时，如果book_isbn、book_author
不给值，可以默认为null。
如果book_name不给值则提示添加错误

唯一约束

在表中的多条数据，此列的值不能重复

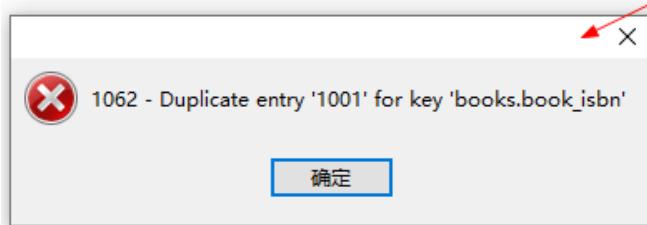
- 创建表：设置图书表的book_isbn为 unique

```
create table books(
    book_isbn char(4) unique,
    book_name varchar(10) not null,
    book_author varchar(6)
);
```

- 添加数据：

book_isbn	book_name	book_author
(Null)	Java	张安
(Null)	Python	李四
1001	C++	老王
1001	PHP	老李

当我们添加数据时，如果给定的book_isbn的值已经存在，则显示如下提示：



主键约束

主键——就是数据表中记录的唯一标识，在一张表中只能有一个主键（主键可以是一个列，也可以是多个列的组合）

当一个字段声明为主键之后，添加数据时：

- 此字段数据不能为null
- 此字段数据不能重复

创建表时添加主键约束

```
create table books(
    book_isbn char(4) primary key,
    book_name varchar(10) not null,
    book_author varchar(6)
);
```

或者

```
create table books(
    book_isbn char(4),
    book_name varchar(10) not null,
    book_author varchar(6),
    primary key(book_isbn)
);
```

删除数据表主键约束

```
alter table books drop primary key;
```

创建表之后添加主键约束

```

## 创建表时没有添加主键约束
create table books(
    book_isbn char(4),
    book_name varchar(10) not null,
    book_author varchar(6)
);

## 创建表之后添加主键约束
alter table books modify book_isbn char(4) primary key;

```

主键自动增长

在我们创建一张数据表时，如果数据表中有列可以作为主键（例如：学生表的学号、图书表的isbn）我们可以直接将这个列为主键；

当有些数据表中没有合适的列作为主键时，我们可以额外定义一个与记录本身无关的列（ID）作为主键，此列数据无具体的含义主要用于标识一条记录，在mysql中我们可以将此列定义为int，同时设置为自动增长，当我们向数据表中新增一条记录时，无需提供ID列的值，它会自动生成。

定义此列的原因，在于后期我们使用数据，对于数据进行CRUD的时候，可以根据此ID字段来完成一些操作。

定义主键自动增长

- 定义int类型字段自动增长：`auto_increment`

```

create table types(
    type_id int primary key auto_increment,
    type_name varchar(20) not null,
    type_remark varchar(100)
);

```

注意：自动增长从1开始，每添加一条记录，自动增长的列会自定+1，当我们把某条记录删除之后再添加数据，自动增长的数据也不会重复生成（自动增长只保证唯一性、不保证连续性）

联合主键

联合主键——将数据表中的多列组合在一起设置为表的主键

students				courses		
stu_num	stu_name	stu_gender	stu_age	course_id	course_name	course_xf
101	张三	男	21	1	Java	4
102	李四	女	20	2	C++	3
grades				stu_num='101' 且 course_id=1		
stu_num	course_id	score				
101	1	59				
101	2	60				
102	1	78				
102	2	59				

如上的案例中，students为学员、courses为课程表、grades表为分数表。此时成绩表中没有任何一列可以作为主键的存在，因为不管是stu_num还是course_id包括score都不能作为唯一标识的存在，所以此时我们就需要使用到联合主键，所以需要确定stu_num和course_id一并作为唯一标识

定义联合主键

```
create table grades(
    stu_num char(8),
    course_id int,
    score int,
    primary key(stu_num, course_id)
);

## 以下写法不可
create table grades(
    stu_num char(8) primary key,
    course_id int primary key,
    score int
);
```

注意：在实际企业项目的数据库设计中，联合主键使用频率并不高；当一个张数据表中没有明确的字段可以作为主键时，我们可以额外添加一个ID字段作为主键。

外键约束

在多表关联部分讲解

DML 数据操纵语言

用于完成对数据表中数据的插入、删除、修改操作

```
create table students(
    stu_num char(8) primary key,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null,
    stu_tel char(11) not null unique,
    stu_qq varchar(11) unique
);
```

插入数据

语法

```
insert into <tableName>(columnName, columnName....)
values(value1, value2....);
```

示例

```
## 向数据表中添加数据，完整格式
insert into stus(stu_num,stu_name,stu_gender,stu_age,stu_tel,stu_qq)
values('20230101','张三','男',21,'13030303300','555555');

## 向数据表中指定的列添加数据（不允许为空的列必须提供数据）
insert into stus(stu_num,stu_name,stu_gender,stu_age,stu_tel)
values('20230102','李四','男',21,'13030303300');

## 数据表名后的字段名列表顺序可以不与表中一致，但是values中值的顺序必须与表名后
## 字段名顺序对应
insert into stus(stu_num,stu_name,stu_age,stu_tel,stu_gender)
values('20230103','王五',20,'13030303302','女');

## 当要向表中的所有列添加数据时，数据表名后面的字段列表可以省略，但是values中的
## 值的顺序要与数据表定义的字段保持一致；
insert into stus values('20230104','赵六','男',21,'13030303304','666666');

## 不过在项目开发中，即使要向所有列添加数据，也建议将列名的列表显式写出来（增强SQL
## 的稳定性）
insert into stus(stu_num,stu_name,stu_gender,stu_age,stu_tel,stu_qq)
values('20230105','小明','男',21,'13030303304','666666');
```

删除数据

从数据表中删除满足特定条件（所有）的记录

语法

```
delete from <tableName> [where conditions];
```

实例

```
## 删除学号为20230102的学生信息
delete from stus where stu_num='20230102';

## 删除年龄大于20岁的学生信息(如果满足where子句的记录有多条，则删除多条记录)
delete from stus where stu_age>20;

## 如果删除语句没有where子句，则表示删除当前数据表中的所有记录(敏感操作)
delete from stus;
```

修改数据

对数据表中已经添加的记录进行修改

语法

```
update <tableName> set columnName=value [where conditions]
```

示例

```
## 将学号为20230105的学生姓名修改为“孙七”（只修改一列）
update stus set stu_name='孙七' where stu_num='20230105';

## 将学号为20230103的学生 性别修改为“男”，同时将QQ修改为 777777（修改多列）
update stus set stu_gender='男',stu_qq='777777' where stu_num='20230103';

## 根据主键修改其他所有列
update stus set stu_name='韩梅
梅',stu_gender='女',stu_age=18,stu_tel='13131313311',stu_qq='999999' where
stu_num='20230102';

## 如果update语句没有where子句，则表示修改当前表中所有行（记录）
update stus set stu_name='Tom';
```

DQL 数据查询语言

从数据表中提取满足特定条件的记录

- 单表查询
- 多表联合查询

查询基础语法

```
## select 关键字后指定要显示查询到的记录的哪些列
select columnName1[,columnName2, columnName3...] from <tableName> [where
conditions];

## 如果要显示查询到的记录的所有列，则可以使用 * 替代字段名列表（在项目开发中不
建议使用*）
select * from stus;
```

where 子句

在删除、修改及查询的语句后都可以添加where子句（条件），用于筛选满足特定的添加的数据进行删除、修改和查询操作。

```
delete from tableName where conditions;
update tabeName set ... where conditions;
select .... from tableName where conditions;
```

条件关系运算符

```
## = 等于
select * from stus where stu_num = '20230101';

## != <> 不等于
select * from stus where stu_num != '20230101';
select * from stus where stu_num <> '20230101';

## > 大于
select * from stus where stu_age>18;

## < 小于
select * from stus where stu_age<20;

## >= 大于等于
select * from stus where stu_age>=20;

## <= 小于等于
select * from stus where stu_age<=20;

## between and 区间查询      between v1 and v2      [v1,v2]
select * from stus where stu_age between 18 and 20;
```

条件逻辑运算符

在where子句中，可以将多个条件通过逻辑运算(and or not)进行连接，通过多个条件来筛选要操作的数据。

```

## and 并且 筛选多个条件同时满足的记录
select * from stus where stu_gender='女' and stu_age<21;

## or 或者 筛选多个条件中至少满足一个条件的记录
select * from stus where stu_gender='女' or stu_age<21;

## not 取反
select * from stus where stu_age not between 18 and 20;

```

LIKE 子句

在where子句的条件中，我们可以使用like关键字来实现模糊查询

语法

```
select * from tableName where columnName like 'reg';
```

- 在like关键字后的reg表达式中
 - % 表示任意多个字符 【%o% 包含字母o】
 - _ 表示任意一个字符 【_o% 第二个字母为o】

示例

```

# 查询学生姓名包含字母o的学生信息
select * from stus where stu_name like '%o%';

# 查询学生姓名第一个字为`张`的学生信息
select * from stus where stu_name like '张%';

# 查询学生姓名最后一个字母为o的学生信息
select * from stus where stu_name like '%y';

# 查询学生姓名中第二个字母为o的学生信息
select * from stus where stu_name like '_o%';

```

对查询结果的处理

设置查询的列

声明显示查询结果的指定列

```
select columnName1,columnName2,... from stus where stu_age>20;
```

计算列

对从数据表中查询的记录的列进行一定的运算之后显示出来

```
## 出生年份 = 当前年份 - 年龄
select stu_name,2021-stu_age from stus;
```

stu_name	2021-stu_age
omg	2000
韩梅梅	2003
Tom	2001
Lucy	2000
Polly	2000
Theo	2004

as 字段取别名

我们可以为查询结果的列名去一个语义性更强的别名（如下案例中`as`关键字也可以省略）

```
select stu_name,2021-stu_age as stu_birth_year from stus;
```

stu_name	stu_birth_year
omg	2000
韩梅梅	2003
Tom	2001
Lucy	2000
Polly	2000
Theo	2004

```
select stu_name as 姓名,2021-stu_age as 出生年份 from stus;
```

姓名	出生年份
omg	2000
韩梅梅	2003
Tom	2001
Lucy	2000
Polly	2000
Theo	2004

distinct 消除重复行

从查询的结果中将重复的记录消除 `distinct`

```
select stu_age from stus;
```

```

|   18 |
|   20 |
|   21 |
|   21 |
|   17 |
+-----+
select distinct stu_age from stus;
+-----+
| stu_age |
+-----+
|   21 |
|   18 |
|   20 |
|   17 |
+-----+

```

排序 - order by

将查询到的满足条件的记录按照指定的列的值升序/降序排列

语法

```
select * from tableName where conditions order by columnName asc|desc;
```

- order by columnName 表示将查询结果按照指定的列排序
 - asc 按照指定的列升序（默认如果是默认的可以省略asc关键字）
 - desc 按照指定的列降序

实例

```
# 单字段排序
select * from stus where stu_age>15 order by stu_gender desc;
+-----+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel    | stu_qq |
+-----+-----+-----+-----+-----+-----+
| 20210101 | omg       | 男         | 21     | 13030303300 | NULL    |
| 20210103 | Tom        | 男         | 20     | 13030303302 | 777777 |
| 20210105 | Polly      | 男         | 21     | 13030303304 | 666666 |
| 20210106 | Theo        | 男         | 17     | 13232323322 | NULL    |
| 20210102 | 韩梅梅     | 女         | 18     | 13131313311 | 999999 |
| 20210104 | Lucy        | 女         | 21     | 13131323334 | NULL    |
+-----+-----+-----+-----+-----+-----+
# 多字段排序：先满足第一个排序规则，第一个排序规则完成之后，在按照第二个排序规则执行
(以下案例就是先按照性别升序排列，然后再降序排序相同性别的年龄)
select * from stus where stu_age>15 order by stu_gender asc,stu_age desc;
+-----+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel    | stu_qq |
+-----+-----+-----+-----+-----+-----+

```

stu_num	stu_name	stu_gender	stu_age	stu_tel	stu_qq
20210104	Lucy	女	21	13131323334	NULL
20210102	韩梅梅	女	18	13131313311	999999
20210101	omg	男	21	13030303300	NULL
20210105	Polly	男	21	13030303304	666666
20210103	Tom	男	20	13030303302	777777
20210106	Theo	男	17	13232323322	NULL

聚合函数

SQL中提供了一些可以对查询的记录的列进行计算的函数——聚合函数

- count 统计函数
- max 最大值
- min 最小值
- sum 求和
- avg 平均值
- `count()` 统计函数，统计满足条件的指定字段值的个数（记录数）

```
# 统计学生表中学生总数
select count(stu_num) from stus;
+-----+
| count(stu_num) |
+-----+
|          7   |
+-----+


# 统计学生表中性别为男的学生总数
select count(stu_num) from stus where stu_gender='男';
+-----+
| count(stu_num) |
+-----+
|          5   |
+-----+
```

- `max()` 计算最大值，查询满足条件的记录中指定列的最大值

```
select max(stu_age) from stus;
+-----+
| max(stu_age) |
+-----+
|      21      |
+-----+


select max(stu_age) from stus where stu_gender='女';
+-----+
| max(stu_age) |
+-----+
|      21      |
+-----+
```

- `min()` 计算最小值，查询满足条件的记录中指定列的最小值

```
select min(stu_age) from stus;
+-----+
| min(stu_age) |
+-----+
|      14      |
+-----+


select min(stu_age) from stus where stu_gender='女';
+-----+
| min(stu_age) |
+-----+
|      18      |
+-----+
```

- `sum()` 计算和，查询满足条件的记录中 指定的列的值的总和

```
# 计算所有学生年龄的总和
select sum(stu_age) from stus;
+-----+
| sum(stu_age) |
+-----+
|      133     |
+-----+


# 计算所有性别为男的学生的年龄的综合
select sum(stu_age) from stus where stu_gender='男';
+-----+
| sum(stu_age) |
+-----+
|      94      |
+-----+
```

- `avg()` 求平均值，查询满足条件的记录中 计算指定列的平均值

```

select avg(stu_age) from stus;
+-----+
| avg(stu_age) |
+-----+
|      19.0000 |
+-----+

select avg(stu_age) from stus where stu_gender='男';
+-----+
| avg(stu_age) |
+-----+
|      18.8000 |
+-----+

```

日期函数 和 字符串函数

日期函数

当我们向日期类型的列添加数据时，可以通过字符串类型赋值（字符串的格式必须为 yyyy-MM-dd hh:mm:ss）

如果我们想要获取当前系统时间添加到日期类型的列，可以使用 `now()` 或者 `sysdate()`

示例：

首先我们现在现有表格的基础之上添加一列 入学时间

```
alter table students add stu_enrollment datetime;
```

```

desc stus;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| stu_num | char(8) | NO  | PRI | NULL    |       |
| stu_name | varchar(20) | NO  |     | NULL    |       |
| stu_gender | char(2) | YES |     | NULL    |       |
| stu_age | int    | NO  |     | NULL    |       |
| stu_tel | char(11) | NO  | UNI | NULL    |       |
| stu_qq | varchar(11) | YES | UNI | NULL    |       |
| stu_enrollment | datetime | YES |     | NULL    |       |
+-----+-----+-----+-----+-----+

```

```

# 通过字符串类型 给日期类型的列赋值
insert into
stus(stu_num,stu_name,stu_gender,stu_age,stu_tel,stu_qq,stu_enrollment)
values('20200108','张小三','女',20,'13434343344','123111','2022-09-01
09:00:00');

# 通过now()获取当前时间

```

```

insert into
stus(stu_num,stu_name,stu_gender,stu_age,stu_tel,stu_qq,stu_enrollment)
values('20210109','张小四','女',20,'13434343355','1233333',now());

# 通过sysdate()获取当前时间
insert into
stus(stu_num,stu_name,stu_gender,stu_age,stu_tel,stu_qq,stu_enrollment)
values('20210110','李雷','男',16,'13434343366','123333344',sysdate());

# 通过now和sysdate获取当前系统时间
mysql> select now();
+-----+
| now() |
+-----+
| 2022-11-29 16:22:19 |
+-----+

mysql> select sysdate();
+-----+
| sysdate() |
+-----+
| 2022-11-29 16:22:26 |
+-----+

```

字符串函数

就是通过SQL指令对字符串进行处理

示例：

```

# concat(column1,column2,...) 拼接多列
select concat(stu_name,'-',stu_gender) from stus;
+-----+
| concat(stu_name,'-',stu_gender) |
+-----+
| 韩梅梅-女 |
| Tom-男 |
| Lucy-女 |
| 林涛-男 |
+-----+

# upper(column) 将字段的值转换成大写
mysql> select upper(stu_name) from stus;
+-----+
| upper(stu_name) |
+-----+
| 韩梅梅 |
| TOM |
| LUCY |
| POLLY |
| THEO |
| 林涛 |
+-----+

```

```
+-----+
# lower(column) 将指定列的值转换成小写
mysql> select lower(stu_name) from stus;
+-----+
| lower(stu_name) |
+-----+
| 韩梅梅          |
| tom            |
| Lucy           |
| polly          |
| theo           |
+-----+

# substring(column,start,len) 从指定列中截取部分显示 start从1开始
参数8代表第八位数字，参数4代表截取几位
mysql> select stu_name,substring(stu_tel,8,4) from stus;
+-----+
| stu_name | substring(stu_tel,8,4) |
+-----+
| 韩梅梅   | 3311                |
| Tom      | 3302                |
| Lucy     | 3334                |
+-----+
```

分组查询 - group by

分组——就是将数据表中的记录按照指定的类进行分组

语法

语法中加[]的是可有可无的，group by一般和having一起使用

```
select 分组字段/聚合函数
from 表名
[where 条件]
group by 分组列名 [having 条件]
[order by 排序字段]
```

- `select` 后使用`*`显示对查询的结果进行分组之后，显示每组的第一条记录（这种显示通常是没有意义的）
- `select` 后通常显示分组字段和聚合函数(对分组后的数据进行统计、求和、平均值等)
- 语句执行属性：
 1 先根据where条件从数据库查询记录
 2 group by对查询记录进行分组
 3 执行having对分组后的数据进行筛选

示例

```
# 先对查询的学生信息按性别进行分组（分成了男、女两组），然后再分别统计每组学生的
个数
select stu_gender,count(stu_num) from stus group by stu_gender;
```

```
+-----+
| stu_gender | count(stu_num) |
+-----+
| 女          |      4 |
| 男          |      5 |
+-----+
```

先对查询的学生信息按性别进行分组（分成了男、女两组），然后再计算每组的平均年龄

```
select stu_gender,avg(stu_age) from stus group by stu_gender;
+-----+
| stu_gender | avg(stu_age) |
+-----+
| 女          | 19.7500 |
| 男          | 18.2000 |
+-----+
```

先对学生按年龄进行分组（分了16、17、18、20、21、22六组），然后统计各组的学生数量，还可以对最终的结果排序

```
select stu_age,count(stu_num) from stus group by stu_age order by stu_age;
+-----+
| stu_age | count(stu_num) |
+-----+
|    16   |      2 |
|    17   |      1 |
|    18   |      1 |
|    20   |      3 |
|    21   |      1 |
|    22   |      1 |
+-----+
```

查询所有学生，按年龄进行分组，然后分别统计每组的人数，再筛选当前组人数>1的组，再按年龄升序显示出来

```
select stu_age,count(stu_num)
from stus
group by stu_age
having count(stu_num)>1
order by stu_age;
+-----+
| stu_age | count(stu_num) |
+-----+
|    16   |      2 |
|    20   |      3 |
+-----+
```

查询性别为'男'的学生，按年龄进行分组，然后分别统计每组的人数，再筛选当前组人数>1的组，再按年龄升序显示出来

```
mysql> select stu_age,count(stu_num)
-> from stus
-> where stu_gender='男'
-> group by stu_age
```

```

-> having count(stu_num)>1
-> order by stu_age;
+-----+-----+
| stu_age | count(stu_num) |
+-----+-----+
|     16   |          2 |
|     20   |          2 |
+-----+-----+

```

分页查询 - limit

当数据表中的记录比较多的时候，如果一次性全部查询出来显示给用户，用户的可读性/体验性就不太好，因此我们可以将这些数据分页进行展示。

语法

```

select ...
from ...
where ...
limit param1,param2

```

- param1 int, 表示获取查询语句的结果中的第一条数据的索引（索引从0开始）
- param2 int, 表示获取的查询记录的条数（如果剩下的数据条数<param2，则返回剩下的所有记录）

案例

对数据表中的学生信息进行分页显示，总共有10条数据，我们每页显示3条

总记录数 count 10

每页显示 pageSize 3

总页数： pageCount = count%pageSize==0 ? count/pageSize : count/pageSize +1;

```

# 查询第一页:
select * from stus [where ...] limit 0,3;           (1-1)*3

# 查询第二页:
select * from stus [where ...] limit 3,3;           (2-1)*3

# 查询第三页:
select * from stus [where ...] limit 6,3;           (3-1)*3

# 查询第四页:
select * from stus [where ...] limit 9,3;           (4-1)*3

# 如果在一张数据表中:
# pageNum表示查询的页码
# pageSize表示每页显示的条数
# 通用分页语句如下:

```

```
select * from <tableName> [where ...] limit (pageNum-1)*pageSize,pageSize;
```

注意：SQL语句书写和执行顺序

书写：select、form、where、group by、having、order by、limit

执行：from、where、group by、having、select、order by、limit

(后续还会学到join，那么它的优先级要高于form)

七、数据表的关联关系

关联关系介绍

MySQL是一个关系型数据库，不仅可以存储数据，还可以维护数据与数据之间的关系——通过在数据表中添加字段建立外键约束

学生信息表				
stu_num	stu_name	stu_gender	stu_age	cid(外键)
1001	张三	男	20	2
1002	李四	女	19	3

班级信息表		
class_id	chass_name	class_remark
1	Java2105	...
2	Java2106	...
3	Python2105	...

数据与数据之间的关联关系分为四种：

- 一对关联
- 一对多关联
- 多对一关联
- 多对多关联

一对关联

人 --- 身份证 一个人只有一个身份证、一个身份证只对应一个人

学生 --- 学籍 一个学生只有一个学籍、一个学籍也对应唯一的一个学生

用户 --- 用户详情 一个用户只有一个详情、一个详情也只对应一个用户

方案1： 主键关联——两张数据表中主键相同的数据为相互对应的数据

用户基本信息表			
user_id	login_name	login_pwd	last_login_time
1	admin	666666	2021/9/15
2	javazhang	888888	2021/9/16
3	ergouzi	999999	2021/9/17

用户详情表					
detail_id	real_name	gender	age	addr	tel
1	张三	男	22	湖北武汉	130...
2	王小明	女	25	北京	131...
3	刘小花	女	21	广州	132...

方案2： 唯一外键 —— 在任意一张表中添加一个字段添加外键约束与另一张表主键关联，并且将外键列添加唯一约束



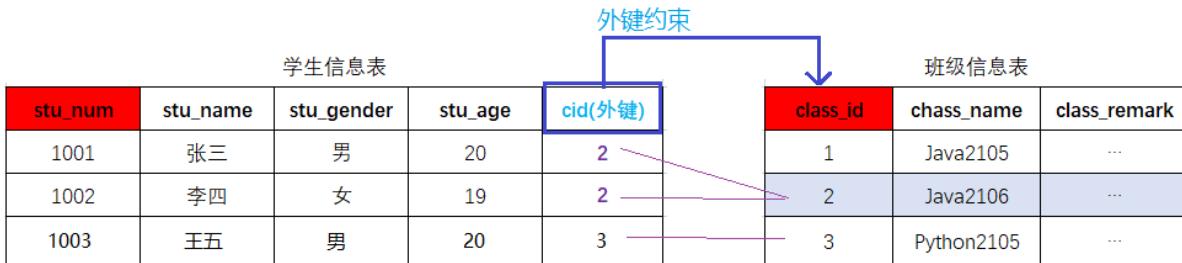
一对多与多对一

班级 --- 学生 (一对多) 一个班级包含多个学生

学生 --- 班级 (多对一) 多个学生可以属于同一个班级

图书 --- 分类 商品 ---- 商品类别

方案：在多的一端添加外键，与一的一端主键进行关联



多对多关联

学生 --- 课程 一个学生可以选择多门课、一门课程也可以由多个学生选择

会员 --- 社团 一个会员可以参加多个社团、一个社团也可以招纳多个会员

方法：额外创建一张关系表来维护多对多关联——在关系表中定义两个外键，分别与两个数据表的主键进行关联

students

stu_num	stu_name	stu_gender	stu_age
101	张三	男	21
102	李四	女	20
103	王五	男	20

courses

course_id	course_name	course_xf
1	Java	4
2	C++	3
3	Python	4

关系表

snum(外键)	cid(外键)	score
103	1	59
103	3	72
102	1	-1

王五选择Java课程
王五选择Python课程
李四选择Java课程

外键约束

外键约束——将一个列添加外键约束与另一张表的主键(唯一列)进行关联之后，这个外键约束的列添加的数据必须要在关联的主键字段中存在

这里新建一个新的数据库，为了方便后续的测试，数据库名称为：db_test2

```
create database db_test2;
use db_test2;
```

案例：学生表 与 班级表 （在学生表中添加外键与班级表的主键进行关联）

stu_num	stu_name	stu_gender	stu_age	cid
1001	张三	男	20	2
1002	李四	女	18	3

class_id	class_name	class_remark
1	Java2201	
2	Java2202	
3	Java2203	

创建原则，先创建不包含外键的表也就是班级表

1. 先创建班级表

```
create table classes(
    class_id int primary key auto_increment,
    class_name varchar(40) not null unique,
    class_remark varchar(200)
);
```

2. 创建学生表（在学生表中添加外键与班级表的主键进行关联）

```
# 【方式一】在创建表的时候，定义cid字段，并添加外键约束
# 由于cid列要与classes表的class_id进行关联，因此cid字段类型和长度要与
# class_id一致
create table students(
```

```

stu_num char(8) primary key,
stu_name varchar(20) not null,
stu_gender char(2) not null,
stu_age int not null,
cid int unique, #如果需要一对一关系，那么需要添加unique约束
constraint FK_STUDENTS_CLASSES foreign key(cid) references
classes(class_id)
# constraint(关键字) FK_STUDENTS_CLASSES(约束名称) foreign key(cid)
# (外键约束+具体字段) references classes(class_id)(关联的表的具体字段)
);

# 【方式二】先创建表，再添加外键约束
create table students(
    stu_num char(8) primary key,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null,
    cid int
);
# 在创建表之后，为cid添加外键约束
#      修改学生表          添加约束          约束名称          外键约束（具
体字段）  关联classes表的class_id列
alter table students add constraint FK_STUDENTS_CLASSES foreign
key(cid) references classes(class_id);

# 删除外键约束
alter table students drop foreign key FK_STUDENTS_CLASSES;

```

3. 向班级表添加班级信息

```

insert into classes(class_name,class_remark) values('Java2104','...');
insert into classes(class_name,class_remark) values('Java2105','...');
insert into classes(class_name,class_remark) values('Java2106','...');
insert into classes(class_name,class_remark)
values('Python2106','...');

select * from classes;
+-----+-----+-----+
| class_id | class_name | class_remark |
+-----+-----+-----+
| 1 | Java2104 | ... |
| 2 | Java2105 | ... |
| 3 | Java2106 | ... |
| 4 | Python2106 | ... |
+-----+-----+-----+

```

4. 向学生表中添加学生信息

```

insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20210102','李斯','女',20, 4);

# 添加学生时，设置给cid外键列的值必须在其关联的主表classes的class_id列存在
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20210103','王五','男',20, 6);

```

外键约束-级联

当学生表中存在学生信息关联班级表的某条记录时，就不能对班级表的这条记录进行修改ID和删除操作，如下：

```

mysql> select * from classes;
+-----+-----+-----+
| class_id | class_name | class_remark |
+-----+-----+-----+
|      1 | Java2104 | ...          | # 班级表中class_id=1的班级信息
被学生表中的记录关联了
|      2 | Java2105 | ...          | # 我们就不能修改Java2104的
class_id，并且不能删除
|      3 | Java2106 | ...          |
|      4 | Python2106 | ...          |
+-----+-----+-----+

mysql> select * from students;
+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | cid   |
+-----+-----+-----+-----+-----+
| 20210101 | 张三 | 男 | 18 | 1 |
| 20210102 | 李四 | 男 | 18 | 1 |
| 20210103 | 王五 | 男 | 18 | 1 |
| 20210104 | 赵柳 | 女 | 18 | 2 |
+-----+-----+-----+-----+

mysql> update classes set class_id=5 where class_name='Java2104';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
constraint fails (`db_test2`.`students`, CONSTRAINT `FK_STUDENTS_CLASSES`
FOREIGN KEY (`cid`) REFERENCES `classes` (`class_id`))

mysql> delete from classes where class_id=1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
constraint fails (`db_test2`.`students`, CONSTRAINT `FK_STUDENTS_CLASSES`
FOREIGN KEY (`cid`) REFERENCES `classes` (`class_id`))

```

如果一定要修改Java2104的班级ID，该如何实现呢？

- 将引用Java2104班级id的学生记录中的cid修改为NULL

- 在修改班级信息表中Java2104记录的 class_id
- 将学生表中cid设置为NULL的记录的cid重新修改为 Java2104这个班级的新的id

```
1 update students set cid=NULL where cid=1; # 结果如下:
```

stu_num	stu_name	stu_gender	stu_age	cid
20210101	张三	男	18	NULL
20210102	李四	男	18	NULL
20210103	王五	男	18	NULL
20210104	赵柳	女	18	2

```
2 update classes set class_id=5 where class_name='Java2104'; # 结果如下
```

class_id	class_name	class_remark
2	Java2105	...
3	Java2106	...
4	Python2106	...
5	Java2104	...

3 update students set cid=5 where cid IS NULL; # 结果如下（null值需要通过IS关键字判断）

stu_num	stu_name	stu_gender	stu_age	cid
20210101	张三	男	18	5
20210102	李四	男	18	5
20210103	王五	男	18	5
20210104	赵柳	女	18	2

我们可以使用级联操作来实现：

1. 在添加外键时，设置级联修改 和 级联删除

```
# 删除原有的外键
alter table students drop foreign key FK_STUDENTS_CLASSES;

# 重新添加外键，并设置级联修改和级联删除
alter table students add constraint FK_STUDENTS_CLASSES foreign
key(cid) references classes(class_id) ON UPDATE CASCADE ON DELETE
CASCADE;
# ON UPDATE CASCADE ON DELETE CASCADE 表示添加修改和删除的级联操作
```

2. 测试级联修改：

```
# 班级信息
```

```
+-----+-----+-----+
```

```

| class_id | class_name | class_remark |
+-----+-----+-----+
|     2 | Java2105 | ...      |
|     3 | Java2106 | ...      |
|     4 | Python2106 | ...      |
|     5 | Java2104 | ...      |
+-----+-----+-----+
# 学生信息
+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | cid   |
+-----+-----+-----+-----+-----+
| 20210101 | 张三     | 男        | 18       | 5    |
| 20210102 | 李四     | 男        | 18       | 5    |
| 20210103 | 王五     | 男        | 18       | 5    |
| 20210104 | 赵柳     | 女        | 18       | 2    |
+-----+-----+-----+-----+
# 直接修改Java2104的class_id,关联Java2104这个班级的学生记录的cid也会同步修改
update classes set class_id=1 where class_name='Java2104';

# 班级信息
+-----+-----+-----+
| class_id | class_name | class_remark |
+-----+-----+-----+
|     2 | Java2105 | ...      |
|     3 | Java2106 | ...      |
|     4 | Python2106 | ...      |
|     1 | Java2104 | ...      |
+-----+-----+-----+
# 学生信息
+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | cid   |
+-----+-----+-----+-----+-----+
| 20210101 | 张三     | 男        | 18       | 1    |
| 20210102 | 李四     | 男        | 18       | 1    |
| 20210103 | 王五     | 男        | 18       | 1    |
| 20210104 | 赵柳     | 女        | 18       | 2    |
+-----+-----+-----+-----+

```

3. 测试级联删除

```

# 删除class_id=1的班级信息，学生表引用此班级信息的记录也会被同步删除
delete from classes where class_id=1;

+-----+-----+-----+
| class_id | class_name | class_remark |
+-----+-----+-----+
|     2 | Java2105 | ...      |
|     3 | Java2106 | ...      |
|     4 | Python2106 | ...      |
+-----+-----+-----+

```

stu_num	stu_name	stu_gender	stu_age	cid
20210104	赵柳	女	18	2

八、连接查询

通过对DQL的学习，我们可以很轻松的从一张数据表中查询出需要的数据；在企业的应用开发中，我们经常需要从多张表中查询数据（例如：我们查询学生信息的时候需要同时查询学生的班级信息），可以通过连接查询从多张数据表提取数据：

在MySQL中可以使用join实现多表的联合查询——连接查询，join按照其功能不同分为三个操作：

- inner join 内连接
- left join 左连接
- right join 右连接

数据准备

创建新的数据库db_test2

```
create database db_test2;
use db_test2;
```

创建班级信息表 和 学生信息表

```
create table classes(
    class_id int primary key auto_increment,
    class_name varchar(40) not null unique,
    class_remark varchar(200)
);

create table students(
    stu_num char(8) primary key,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null,
    cid int,
    constraint FK_STUDENTS_CLASSES foreign key(cid) references
    classes(class_id) ON UPDATE CASCADE ON DELETE CASCADE
);
```

添加数据

添加班级信息

```
# Java2204 包含三个学生信息
insert into classes(class_name,class_remark) values('Java2204','...');

# Java2205 包含两个学生信息
insert into classes(class_name,class_remark) values('Java2205','...');

# 以下两个班级在学生表中没有对应的学生信息
insert into classes(class_name,class_remark) values('Java2206','...');
insert into classes(class_name,class_remark) values('Python2205','...');
```

添加学生信息

```
# 以下三个学生信息 属于 class_id=1 的班级 (Java2204)
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20220101','张三','男',20,1);
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20220102','李四','女',20,1);
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20220103','王五','男',20,1);

# 以下三个学生信息 属于 class_id=2 的班级 (Java2205)
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20220104','赵柳','女',20,2);
insert into students(stu_num,stu_name,stu_gender,stu_age,cid)
values('20220105','孙七','男',20,2);

# 小红和小明没有设置班级信息
insert into students(stu_num,stu_name,stu_gender,stu_age)
values('20220106','小红','女',20);
insert into students(stu_num,stu_name,stu_gender,stu_age)
values('20220107','小明','男',20);
```

内连接 INNER JOIN

语法

```
select ... from tableName1 inner join tableName2 ON 匹配条件 [where 条件];
```

笛卡尔积

- 笛卡尔积 (A集合&B集合)：使用A中的每个记录依次关联B中每个记录，笛卡尔积的总数 =A总数*B总数
- 如果直接执行 `select ... from tableName1 inner join tableName2;` 会获取两种数据表中的数据集合的笛卡尔积 (依次使用tableName1 表中的每一条记录去匹配 tableName2的每条数据)

内连接条件

两张表使用inner join连接查询之后生产的笛卡尔积数据中很多数据都是无意义的，我们如何消除无意义的数据呢？——添加两张进行连接查询时的条件

- 使用 `on` 设置两张表连接查询的匹配条件

```
-- 使用where设置过滤条件：先生成笛卡尔积再从笛卡尔积中过滤数据（效率很低）
```

```
select * from students INNER JOIN classes where students.cid =  
classes.class_id;
```

```
-- 使用ON设置连接查询条件：先判断连接条件是否成立，如果成立两张表的数据进行组合生成一条结果记录
```

```
select * from students INNER JOIN classes ON students.cid =  
classes.class_id;
```

- 结果：只获取两张表中匹配条件成立的数据，任何一张表在另一种表如果没有找到对应匹配则不会出现在查询结果中（例如：小红和小明没有对应的班级信息，Java2206和Python2206没有对应的学生）。

左连接 LEFT JOIN

需求：请查询出所有的学生信息，如果学生有对应的班级信息，则将对应的班级信息也查询出来

左连接：显示左表中的所有数据，如果在右表中存在与左表记录满足匹配条件的数据，则进行匹配；如果右表中不存在匹配数据，则显示为Null

```
# 语法
```

```
select * from leftTable LEFT JOIN rightTable ON 匹配条件 [where 条件];
```

```
-- 左连接：显示左表中的所有记录
```

```
select * from students LEFT JOIN classes ON students.cid =  
classes.class_id;
```

stu_num	stu_name	stu_gender	stu_age	cid	class_id	class_name	class_remark
20210101	张三	男	20	1	1	Java2104	...
20210102	李四	女	20	1	1	Java2104	...
20210103	王五	男	20	1	1	Java2104	...
20210104	赵柳	女	20	2	2	Java2105	...
20210105	孙七	男	20	2	2	Java2105	...
20210106	小红	女	20	(Null)	(Null)	(Null)	(Null)
20210107	小明	男	20	(Null)	(Null)	(Null)	(Null)

右连接 RIGHT JOIN

-- 右连接：显示右表中的所有记录

```
select * from students RIGHT JOIN classes ON students.cid =
classes.class_id;
```

stu_num	stu_name	stu_gender	stu_age	cid	class_id	class_name	class_remark
20210101	张三	男	20	1	1	Java2104	...
20210102	李四	女	20	1	1	Java2104	...
20210103	王五	男	20	1	1	Java2104	...
20210104	赵柳	女	20	2	2	Java2105	...
20210105	孙七	男	20	2	2	Java2105	...
(Null)	(Null)	(Null)	(Null)	(Null)	3	Java2106	...
(Null)	(Null)	(Null)	(Null)	(Null)	4	Python2105	...

数据表别名

首先我们可以先将两张表stu_name和class_name字段名称都改为name

```
alter table students rename column stu_name to name;
alter table classes rename column class_name to name;
```

如果在连接查询的多张表中存在相同名字的字段，我们可以使用表名.字段名来进行区分

```
select students.name,classes.name
from students
INNER JOIN classes
ON students.cid = classes.class_id;
```

如果表名太长则不便于SQL语句的编写，我们可以使用数据表别名

使用示例：

```
select s.name,c.name
from students s
INNER JOIN classes c
ON s.cid = c.class_id;
```

子查询/嵌套查询

子查询 — 先进行一次查询，第一次查询的结果作为第二次查询的源/条件（第二次查询是基于第一次的查询结果来进行的）

子查询返回单个值-单行单列

案例1：查询班级名称为 'Java2204' 班级中的学生信息 (只知道班级名称，而不知道班级ID)

- 传统的方式：

```
-- a. 查询 Java2204 班的班级编号
select class_id from classes where class_name='Java2204';

-- b. 查询此班级编号下的学生信息
select * from students where cid = 1;
```

- 子查询：

```
-- 如果子查询返回的结果是一个值（单列单行），条件可以直接使用关系运算符 (= != ....)
select * from students where cid = (select class_id from classes where
class_name='Java2205');
```

子查询返回多个值-多行单列

案例2：查询所有 Java 班级中的学生信息

- 传统的方式：

```
-- a. 查询所有 Java 班的班级编号
select class_id from classes where class_name LIKE 'Java%';
+-----+
| class_id |
+-----+
|      1    |
|      2    |
|      3    |
+-----+
```

```
-- b. 查询这些班级编号中的学生信息 (union 将多个查询语句的结果整合在一起)
select * from students where cid=1
UNION
select * from students where cid=2
UNION
select * from students where cid=3;
```

- 子查询

```
-- 如果子查询返回的结果是多个值（单列多行），条件使用 IN / NOT IN
select * from students where cid IN (select class_id from classes where
class_name LIKE 'Java%');
```

子查询返回多个值-多行多列

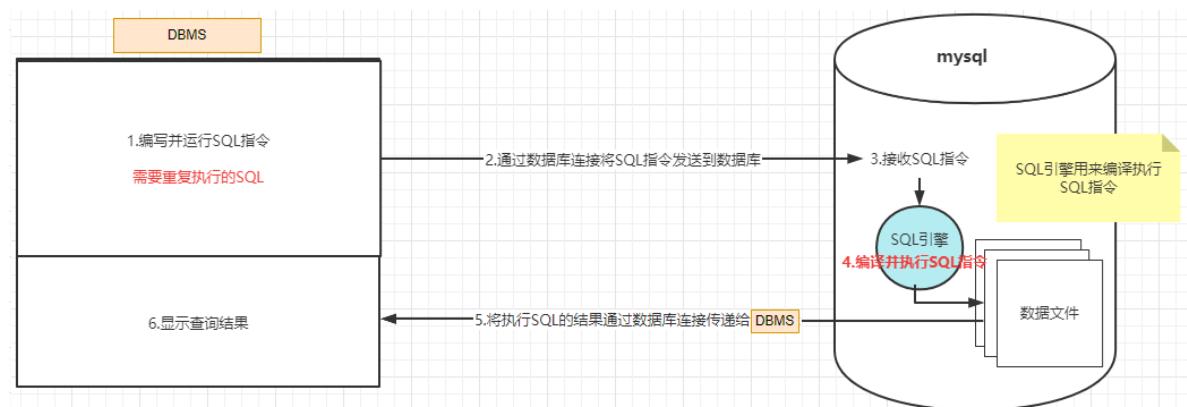
案例3：查询cid=1的班级中性别为男的学生信息

```
-- 多条件查询:  
select * from students where cid=1 and stu_gender='男';  
  
-- 子查询:先查询cid=1班级中的所有学生信息, 将这些信息作为一个整体虚拟表(多行多列)  
--          再基于这个虚拟表查询性别为男的学生信息('虚拟表'需要别名)  
select * from (select * from students where cid=1) t where  
t.stu_gender='男';
```

九、存储过程

存储过程介绍

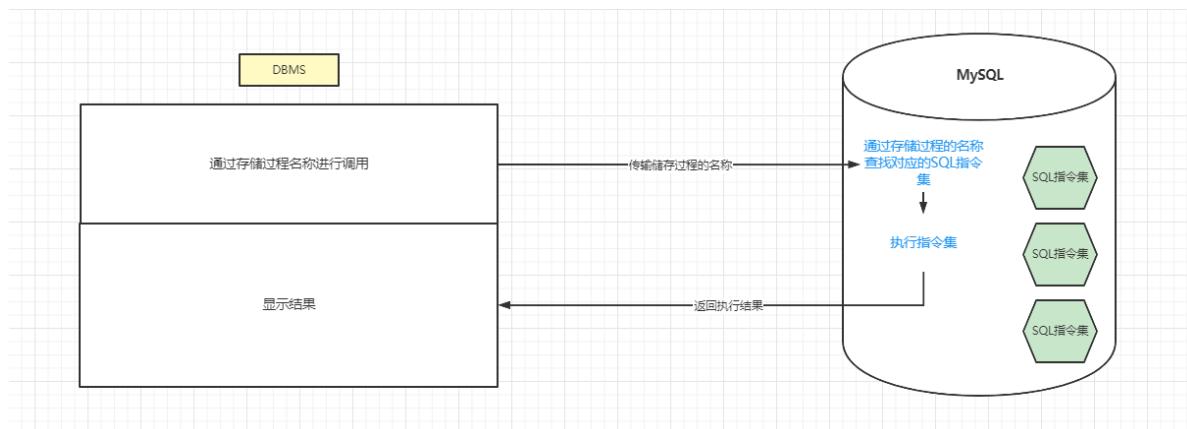
SQL指令执行过程



从SQL执行执行的流程中我们分析存在的问题：

1. 如果我们需要重复多次执行相同的SQL, SQL指令都需要通过连接传递到MySQL, 并且需要经过编译和执行的步骤;
2. 如果我们需要连续执行多个SQL指令, 并且第二个SQL指令需要使用第一个SQL指令执行的结果作为参数;

存储过程的介绍



存储过程：

将能够完成特定功能的SQL指令进行封装（SQL指令集），编译之后存储在数据库服务器上，并且为之取一个名字，客户端可以通过名字直接调用这个SQL指令集，获取执行结果。

存储过程优缺点分析

存储过程优点：

1. SQL指令无需客户端编写，通过网络传送，可以节省网络开销，同时避免SQL指令在网络传输过程中被恶意篡改保证安全性；
2. 存储过程经过编译创建并保存在数据库中的，执行过程无需重复的进行编译操作，对SQL指令的执行过程进行了性能提升；
3. 存储过程中多个SQL指令之间存在逻辑关系，支持流程控制语句（分支、循环），可以实现更为复杂的业务；

存储过程的缺点：

1. 存储过程是根据不同的数据库进行编译、创建并存储在数据库中；当我们需要切换到其他的数据库产品时，需要重写编写针对新数据库的存储过程；
2. 存储过程受限于数据库产品，如果需要高性能的优化会成为一个问题；
3. 在互联网项目中，如果需要数据库的高（连接）并发访问，使用存储过程会增加数据库的连接执行时间（因为我们将复杂的业务交给了数据库进行处理）

创建存储过程

存储过程创建语法

```
-- 语法 [为参数部分，与java类似，可以定义参数，也可以不定义参数]：
create procedure <proc_name>([IN/OUT args])
begin
    -- SQL
end;
```

示例

```
-- 创建一个存储过程实现加法运算： Java语法中，方法是有参数和返回值的
--                                         存储过程中，是有输入参数 和 输出参数的
create procedure proc_test1(IN a int,IN b int,OUT c int)
begin
    -- set代表的是定义变量的意思
    SET c = a+b;
end;
```

调用存储过程

```
-- 调用存储过程
-- 定义变量@m
set @m = 0;
-- 调用存储过程，将3传递给a，将2传递给b，将@m传递给c
call proc_test1(3,2,@m);
-- 显示变量值（dual系统表，无需创建，定义变量的值都会在这里）
select @m from dual;
```

存储过程中变量的使用

存储过程中的变量分为两种：局部变量 和 用户变量

定义局部变量

局部变量：定义在存储过程中的变量，只能在存储过程内部使用

- 局部变量定义语法

```
-- 局部变量要定义在存储过程中，而且必须定义在存储过程开始
declare <attr_name> <type> [default value];
```

- 局部变量定义示例：

```
create procedure proc_test2(IN a int,OUT r int)
begin
    declare x int default 0; -- 定义x int类型，默认值为0
    declare y int default 1; -- 定义y
    set x = a*a;
    set y = a/2;
    set r = x+y;
end;
```

定义用户变量

用户变量：相当于全局变量，定义的用户变量可以通过 `select @attrName from dual` 进行查询

```
-- 用户变量会存储在mysql数据库的数据字典中（dual）
-- 用户变量定义使用set关键字直接定义，变量名要以@开头
set @n=1;
```

给变量设置值

- 无论是局部变量还是用户变量，都是使用 `set` 关键字修改值

```
set @n=1;
call proc_test2(6,@n);
select @n from dual;
```

将查询结果赋值给变量

在存储过程中使用select..into..给变量赋值

```
-- 查询学生数量
-- 注意在储存过程中使用SQL语句需要将结果赋值给变量，那么就需要使用into关键字来进行赋值
create procedure proc_test3(OUT c int)
begin
    select count(stu_num) INTO c from students; -- 将查询到学生数量赋值给c
end;

-- 调用存储过程
call proc_test3(@n);
select @n from dual;
```

用户变量使用注意事项

因为用户变量相当于全局变量，可以在SQL指令以及多个存储过程中共享，在开发中建议尽量少使用用户变量，用户变量过多会导致程序不易理解、难以维护。

存储过程的参数

MySQL存储过程的参数一共有三种：IN \ OUT \ INOUT

IN 输入参数

输入参数——在调用存储过程中传递数据给存储过程的参数（在调用的过程必须为具有实际值的变量或者字面值）

```
-- 创建存储过程：添加学生信息
create procedure proc_test4(IN snum char(8),IN sname varchar(20), IN gender
char(2), IN age int, IN cid int, IN remark varchar(255))
begin
    insert into students(stu_num,stu_name,stu_gender,stu_age,cid,remark)
    values(snum,sname,gender,age,cid,remark);
end;

call proc_test4('20220108','小丽','女',20,1,'aaa');
```

OUT 输出参数

输出参数——将存储过程中产生的数据返回给过程调用者，相当于Java方法的返回值，但不同的是一个存储过程可以有多个输出参数

```
-- 创建存储过程，根据学生学号，查询学生姓名
create procedure proc_test5(IN snum char(8),OUT sname varchar(20))
begin
    select stu_name INTO sname from students where stu_num=snum;
end;

set @name='';
call proc_test5('20220107',@name);
select @name from dual;
```

INOUT 输入输出参数

注意：此方式不建议使用，一般我们输入就用 IN 输出就用OUT，此参数代码可读性低，容易混淆。

```
create procedure proc_test6(INOUT str varchar(20))
begin
    select stu_name INTO str from students where stu_num=str;
end;

set @name='20220108';
call proc_test6(@name);
select @name from dual;
```

存储过程中流程控制

在存储过程中支持流程控制语句用于实现逻辑的控制

分支语句

- if-then-else

```
-- 单分支：如果条件成立，则执行SQL
if conditions then
    -- SQL
end if;

-- 如果参数a的值为1，则添加一条班级信息
create procedure proc_test7(IN a int)
begin
    if a=1 then
        insert into classes(class_name,remark)
values('Java2209','test');
    end if;
end;
```

```
-- 双分支：如果条件成立则执行SQL1，否则执行SQL2
if conditions then
    -- SQL1
else
    -- SQL2
```

```

end if;

-- 如果参数a的值为1，则添加一条班级信息；否则添加一条学生信息
create procedure proc_test7(IN a int)
begin
    if a=1 then
        insert into classes(class_name,remark)
values('Java2209','test');
    else
        insert into
students(stu_num,stu_name,stu_gender,stu_age,cid,remark)
values('20220110','小花','女',19,1,'...');
    end if;
end;

```

- case

```

-- case
create procedure proc_test8(IN a int)
begin
case a
when 1 then
    -- SQL1 如果a的值为1 则执行SQL1
    insert into classes(class_name,remark) values('Java2210','wahaha');
when 2 then
    -- SQL2 如果a的值为2 则执行SQL2
    insert into
students(stu_num,stu_name,stu_gender,stu_age,cid,remark)
values('20220111','小刚','男',21,2,'...');
else
    -- SQL (如果变量的值和所有when的值都不匹配，则执行else中的这个SQL)
    update students set stu_age=18 where stu_num='20220110';
end case;
end;

```

循环语句

- while

concat()函数用于将两个字符串连接起来，形成一个单一的字符串

```
-- while
create procedure proc_test9(IN num int)
begin
    declare i int;
    set i = 0;
    while i<num do
        -- SQL
        insert into classes(class_name,remark) values( CONCAT('Java',i)
,'.....');
        set i = i+1;
    end while;
end;

call proc_test9(4);
```

- repeat

```
-- repeat
create procedure proc_test10(IN num int)
begin
    declare i int;
    set i = 1;
    repeat
        -- SQL
        insert into classes(class_name,remark) values( CONCAT('Python',i)
,'.....');
        set i = i+1;
    until i > num end repeat;
end;

call proc_test10(4);
```

- loop (注意如果需要停止循环，需要通过if来进行结束条件的判断)

```
-- loop
create procedure proc_test11(IN num int)
begin
    declare i int ;
    set i =0;
    myloop:loop
        -- SQL
        insert into classes(class_name,remark) values( CONCAT('HTML',i)
,'.....');
        set i = i+1;
        # 结束循环的条件
        if i=num then
            # 离开循环
            leave myloop;
        end if;
    end loop;
```

```
end;

call proc_test1(5);
```

存储过程管理

查询存储过程

存储过程是属于某个数据库的，也就是说当我们将存储过程创建在某个数据库之后，只能在当前数据库中调用此存储过程。

查询存储过程：查询某个数据库中有哪些存储过程

```
-- 根据数据库名，查询当前数据库中的存储过程
show procedure status where db='db_test2';

-- 查询存储过程的创建细节
show create procedure db_test2.proc_test1;
```

修改存储过程

修改存储过程指的是修改存储过程的特征/特性

```
alter procedure <proc_name> 特征1 [特征2 特征3 ....]
```

存储过程的特征参数：

- `CONTAINS SQL` 表示子程序包含 SQL 语句，但不包含读或写数据的语句
- `NO SQL` 表示子程序中不包含 SQL 语句
- `READS SQL DATA` 表示子程序中包含读数据的语句
- `MODIFIES SQL DATA` 表示子程序中包含写数据的语句
- `SQL SECURITY { DEFINER |INVOKER }` 指明谁有权限来执行
 - `DEFINER` 表示只有定义者自己才能够执行
 - `INVOKER` 表示调用者可以执行
- `COMMENT 'string'` 表示注释信息

```
alter procedure proc_test1 READS SQL DATA;
```

删除存储过程

```
-- 删除存储过程
-- drop 删除数据库中的对象 数据库、数据表、列、存储过程、视图、触发器、索引....
-- delete 删除数据表中的数据
drop procedure proc_test1;
```

游标

问题：如果我们要创建一个存储过程，需要返回查询语句查询到的多条数据，该如何实现呢？

游标的概念

游标可以用来依次取出查询结果集中的每一条数据——逐条读取查询结果集中的记录

游标的使用步骤

1、声明游标

- 声明游标语法：

```
DECLARE cursor_name CURSOR FOR select_statement;
```

- 实例

```
declare mycursor cursor for select class_id,class_name from classes;
```

2、打开游标

- 语法

```
open mycursor;
```

3、使用游标

- 使用游标：提取游标当前指向的记录（提取之后，游标自动下移）

```
fetch mycursor into cid,cname;
```

4、关闭游标

```
CLOSE mycursor;
```

游标使用案例

MySQL中，`concat_ws()` 函数 用来通过指定符号，将2个或多个字段拼接在一起，返回拼接后的字符串。

```
create procedure proc_test12(out result varchar(200))
begin
    # 游标变量
    declare cid int;
    # 游标变量
    declare cname varchar(20);
    # 计数变量
    declare num int;
    # 计数变量
```

```

declare i int;
# 每条数据
declare str varchar(100);
# 查询语句执行之后返回的是一个结果集（多条记录），使用游标遍历查询结果集
declare mycursor cursor for select class_id,class_name from classes;
# 记录总数据量
select count(*) into num from classes;
# 打开游标
open mycursor;
set i = 0;
# 开始遍历游标
while i<num do
    # 提取游标中的数据，并将结果赋值给游标变量
    fetch mycursor into cid,cname;
    set i = i+1;
    # set str=concat_ws('~',cid,cname); 不同的写法
    select concat_ws('~',cid,cname) into str;
    set result = concat_ws(',',result,str);
end while;

close mycursor;
end;

# 案例测试
set @r = '';
call proc_test12(@r);
select @r from dual;

```

十、触发器

触发器的介绍

触发器，就是一种特殊的存储过程。触发器和存储过程一样是一个能够完成特定功能、存储在数据库服务器上的SQL片段，但是触发器无需调用，当对数据表中的数据执行DML操作时自动触发这个SQL片段的执行，无需手动调用。

在MySQL,只有执行insert\delete\update操作才能触发触发器的执行。

触发器的使用

案例说明

```

-- 学生信息表
create table students(
    stu_num char(4) primary key,
    stu_name varchar(20) not null,
    stu_gender char(2) not null,
    stu_age int not null
);

```

-- 学生信息操作日志表

```
create table stulogs(
    id int primary key auto_increment,
    time TIMESTAMP,
    log_text varchar(200)
);
```

```
-- 当向students表中添加学生信息时，同时要在 stulogs表中添加一条操作日志
insert into students(stu_num,stu_name,stu_gender,stu_age) values('1004','夏利','女',20);
-- 手动进行记录日志
insert into stulogs(time,log_text) values(now(),'添加1004学生信息');
```

案例：当向学生信息表添加、删除、修改学生信息时，使用触发器自定进行日志记录

创建触发器

语法

```
create trigger tri_name
<before|after>                      -- 定义触发时机
<insert|delete|update>                -- 定义DML类型
ON <table_name>
for each row                           -- 声明为行级触发器（只要操作一条记录就触发
触器执行一次）
sql_statement                          -- 触发器操作
```

-- 创建触发器：当学生信息表发生添加操作时，则向日志信息表中记录一条日志

```
create trigger tri_test1
after insert on students
for each row
insert into stulogs(time,log_text) values(now(), concat('添
加',NEW.stu_num,'学生信息'));
```

查看触发器

```
show triggers;
```

测试触发器

- 我们创建的触发器是在students表发生insert操作时触发，我们只需执行学生信息的添加操作

```
-- 测试1：添加一个学生信息，触发器执行了一次
insert into students(stu_num,stu_name,stu_gender,stu_age) values('1005','小明','男',20);

-- 测试2：一条SQL指令添加了2条学生信息，触发器就执行了2次
insert into students(stu_num,stu_name,stu_gender,stu_age) values('1006','小刚','男',20),('1007','李磊','男',20);
```

删除触发器

```
drop trigger tri_test1;
```

NEW与OLD

触发器用于监听对数据表中数据的insert、delete、update操作，在触发器中通常处理一些DML的关联操作；我们可以使用 `NEW` 和 `OLD` 关键字在触发器中获取触发这个触发器的DML操作的数据

- `NEW`：在触发器中用于获取insert操作添加的数据、update操作修改后的记录
- `OLD`：在触发器中用于获取delete操作删除前的数据、update操作修改前的数据

NEW

- insert操作中：`NEW`表示添加的新记录

```
create trigger tri_test1
after insert on students
for each row
insert into stulogs(time,log_text) values(now(), concat('添加',NEW.stu_num,'学生信息'));
```

- update操作中：`NEW` 表示修改后的数据

```
-- 创建触发器：在监听update操作的触发器中，可以使用NEW获取修改后的数据
create trigger tri_test2
after update on students for each row
insert into stulogs(time,log_text) values(now(), concat('修改学生信息为：',NEW.stu_num,NEW.stu_name));
```

OLD

- delete操作中：`OLD`表示删除的记录

```
create trigger tri_test3
after delete on students for each row
insert into stulogs(time,log_text) values(now(), concat('删除',OLD.stu_num,'学生信息'));
```

- update操作中：`OLD`表示修改前的记录

```

create trigger tri_test2
after update on students for each row
insert into stulogs(time,log_text) values(now(), concat('将学生姓名从
【',OLD.stu_name,'】修改为【',NEW.stu_name,'】'));

```

触发器使用总结

优点

- 触发器是自动执行的，当对触发器相关的表执行响应的DML操作时立即执行；
- 触发器可以实现表中的数据的级联操作（关联操作），有利于保证数据的完整性；
- 触发器可以对DML操作的数据进行更为复杂的合法性校验（比如校验学员年龄）

缺点

- 使用触发器实现的业务逻辑如果出现问题将难以定位，后期维护困难；
- 大量使用触发器容易导致代码结构杂乱，增加了程序的复杂性；
- 当触发器操作的数据量比较大时，执行效率会大大降低。

使用建议

- 在互联网项目中，应避免使用触发器；
- 对于并发量不大的项目可以选择使用存储过程，但是在互联网引用中不提倡使用存储过程
(原因：存储过程时将实现业务的逻辑交给数据库处理，一则增减了数据库的负载，二则不利于数据库的迁移)

十一、视图

视图的概念

视图，就是由数据库中一张表或者多张表根据特定的条件查询出得数据构造成得虚拟表

视图的作用

- **安全性：**如果我们直接将数据表授权给用户操作，那么用户可以CRUD数据表中所有数据，加入我们想要对数据表中的部分数据进行保护，可以将公开的数据生成视图，授权用户访问视图；用户通过查询视图可以获取数据表中公开的数据，从而达到将数据表中的部分数据对用户隐藏。
- **简单性：**如果我们需要查询的数据来源于多张数据表，可以使用多表连接查询来实现；我们通过视图将这些连表查询的结果对用户开放，用户则可以直接通过查询视图获取多表数据，操作更便捷。

创建视图

语法

```

create view <view_name>
AS
select_statement

```

实例

- 实例1：

```
-- 创建视图实例1：将学生表中性别为男的学生生成一个视图
create view view_test1
AS
select * from students where stu_gender='男';

-- 查询视图
select * from view_test1;
```

- 示例2：

```
-- 创建视图示例2：查询班级编号为2的班级的学员信息
create view view_test2
AS
select s.stu_num,s.stu_name,s.stu_gender,s.stu_age,c.class_name
from classes c right join students s
on c.class_id=s.cid
where c.class_id=2;

-- 查询视图
select * from view_test2;
```

视图数据的特性

视图是虚拟表，查询视图的数据是来源于数据表的。当对视图进行操作时，对原数据表中的数据是否由影响呢？

查询操作：如果在数据表中添加了新的数据，而且这个数据满足创建视图时查询语句的条件，通过查询视图也可以查询出新增的数据；当删除原表中满足查询条件的数据时，也会从视图中删除。

新增数据：如果在视图中添加数据，数据会被添加到原数据表

删除数据：如果从视图删除数据，数据也将从原表中删除

修改操作：如果通过修改数据，则也将修改原数据表中的数据

视图的使用建议：对复杂查询简化操作，并且不会对数据进行修改的情况下可以使用视图。

查询视图结构

```
-- 查询视图结构
desc view_test2;
```

修改视图

```
-- 方式1
OR REPLACE (替换)
create OR REPLACE view view_test1
AS
select * from students where stu_gender='女';

-- 方式2
alter (直接进行修改)
alter view view_test1
AS
select * from students where stu_gender='男';
```

删除视图

- 删除数据表时会同时删除数据表中的数据，删除视图时不会影响原数据表中的数据

```
-- 删除视图
drop view view_test1;
```

十二、索引

数据库是用来存储数据，在互联网应用中数据库中存储的数据可能会很多(大数据)，数据表中数据的查询速度会随着数据量的增长逐渐变慢，从而导致响应用户请求的速度变慢——用户体验差，我们如何提高数据库的查询效率呢？

数据准备

```
create table tb_testindex(
    fid int primary key,
    sid int unique,
    tid int,
    name varchar(20),
    remark varchar(20)
);

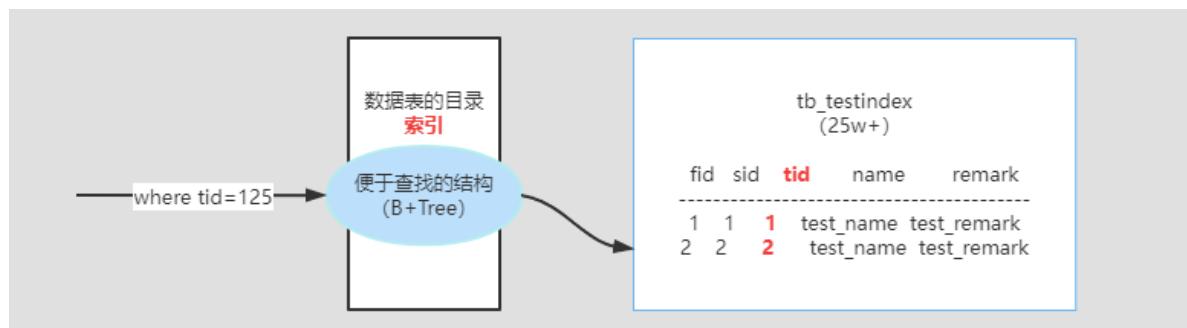
create procedure proc_readydata()
begin
    declare i int default 1;
    while i<=5000000 do
        insert into tb_testindex(fid,sid,tid,name,remark)
            values(i,i,i,'test_name','text_remark');
        set i = i+1;
    end while;
end;
```

索引的介绍

索引，就是用来提高数据表中数据的查询效率的。

索引，就是将数据表中某一列/某几列的值取出来构造成便于查找的结构进行存储，生成数据表的目录。

当我们进行数据查询的时候，则先在目录中进行查找得到对应的数据的地址，然后再到数据表中根据地址快速的获取数据记录，避免全表扫描。



索引的分类

MySQL中的索引，根据创建索引的列的不同，可以分为：

- 主键索引：在数据表的主键字段创建的索引，这个字段必须被primary key修饰，每张表只能有一个主键
- 唯一索引：在数据表中的唯一列创建的索引(unique)，此列的所有值只能出现一次，可以为NULL
- 普通索引：在普通字段上创建的索引，没有唯一性的限制
- 组合索引：两个及以上字段联合起来创建的索引

说明：

1. 在创建数据表时，将字段声明为主键（添加主键约束），会自动在主键字段创建主键索引；
2. 在创建数据表时，将字段声明为唯一键（添加唯一约束），会自动在唯一字段创建唯一索引；

创建索引

唯一索引

```
-- 创建唯一索引：创建唯一索引的列的值不能重复
-- create unique index <index_name> on 表名(列名);
create unique index index_test1 on tb_testindex(tid);
```

普通索引

```
-- 创建普通索引：不要求创建索引的列的值的唯一性
-- create index <index_name> on 表名(列名);
create index index_test2 on tb_testindex(name);
```

组合索引

```
-- 创建组合索引
-- create index <index_name> on 表名(列名1,列名2...);
create index index_test3 on tb_testindex(tid,name);
```

全文索引(了解即可)

MySQL 5.6 版本新增的索引，可以通过此索引进行全文检索操作，因为MySQL全文检索不支持中文，因此这个全文索引不被开发者关注，在应用开发中通常是通过搜索引擎（数据库中间件）实现全文检索

```
create fulltext index <index_name> on 表名(字段名);
```

索引使用

索引创建完成之后无需调用，当根据创建索引的列进行数据查询的时候，会自动使用索引；

组合索引需要根据创建索引的所有字段进行查询时触发（例如：tid=250000 and name='aaa'）。

- 在命令行窗口中可以查看查询语句的查询规划：

```
explain select * from tb_testindex where tid=250000\G;
```

```
mysql> explain select * from tb_testindex where tid=250000\G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: tb_testindex
    partitions: NULL
       type: const
possible_keys: index_test1,index_test3
         key: index_test1
      key_len: 5
        ref: const
       rows: 1
  filtered: 100.00
     Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

查看索引

```
-- 命令行
show create table tb_testindex\G;
```

```
mysql> show create table tb_testindex\G;
***** 1. row *****
  Table: tb_testindex
Create Table: CREATE TABLE `tb_testindex` (
  `fid` int NOT NULL,
  `sid` int DEFAULT NULL,
  `tid` int DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `remark` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`fid`),
  UNIQUE KEY `sid`(`sid`),
  UNIQUE KEY `index_test1`(`tid`),
  KEY `index_test2`(`name`),
  KEY `index_test3`(`tid`, `name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

```
-- 查询数据表的索引
show indexes from tb_testindex;

-- 查询索引
show keys from tb_testindex;
```

删除索引

```
-- 删除索引：索引是建立在表的字段上的，不同的表中可能会出现相同名称的索引
--           因此删除索引时需要指定表名
drop index index_test3 on tb_testindex;
```

索引的使用总结

优点

- 索引大大降低了数据库服务器在执行查询操作时扫描的数据量，提高查询效率
- 索引可以避免服务器排序、将随机IO变成顺序IO

缺点

- 索引是根据数据表列的创建的，当数据表中数据发生DML操作时，索引页需要更新；
- 索引文件也会占用磁盘空间；

注意事项

- 数据表中数据不多时，全表扫面可能更快吗，不要使用索引；
- 数据量大但是DML操作很频繁时，不建议使用索引；
- 不要在数据重复度高的列上创建索引（性别）；
- 创建索引之后，要注意查询SQL语句的编写，避免索引失效。

十三、数据库事务

数据库事务介绍

- 我们把完成特定的业务的多个数据库DML操作步骤称之为一个事务
- 事务，就是完成同一个业务的多个DML操作

- 取钱业务：张三去银行取出 1000
- 操作 1：张三的帐号 -1000
- 操作 2：张三的拿到 +1000

数据库事务特性

ACID特性，高频面试题

原子性 (Atomicity)：一个事务中的多个DML操作，要么同时执行成功，要么同时执行失败

一致性 (Consistency)：事务执行之前和事务执行之后，数据库中的数据是一致的，完整性和一致性不能被破坏

隔离性 (Isolation)：数据库允许多个事务同时执行（张三去银行取钱的同时允许李四同时取钱），多个并行的事务之间不能相互影响

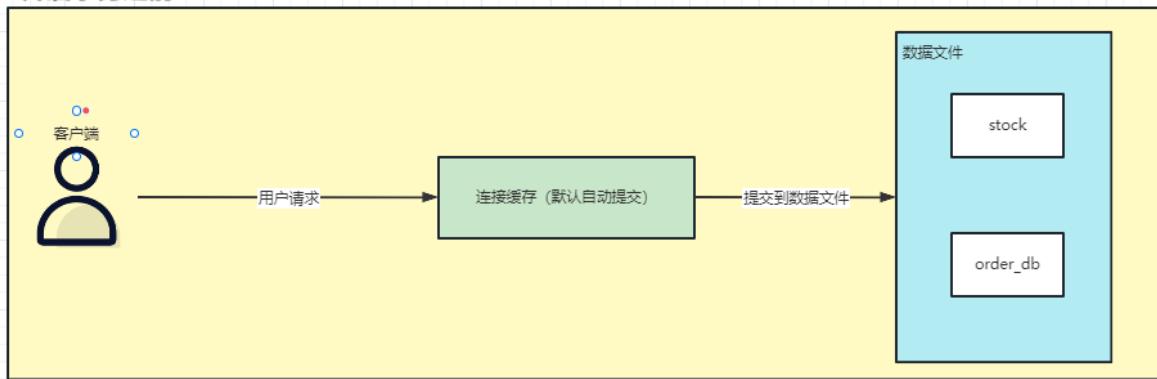
持久性 (Durability)：事务完整之后，对数据库的操作是永久的

MySQL事务管理

自动提交

- 在MySQL中，默认DML指令的执行时自动提交的，当我们执行一个DML指令之后，自动同步到数据库中

开启事务之前

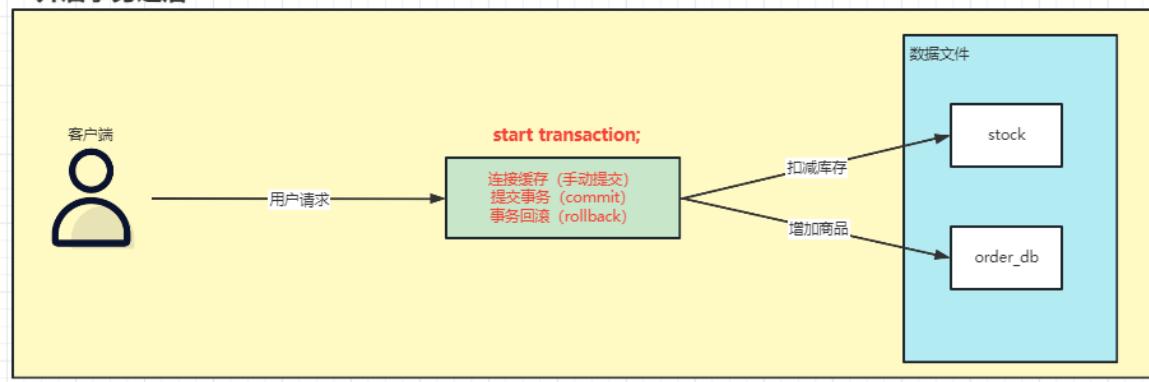


事务管理

开启事务，就是关闭自动提交

- 在开始事务第一个操作之前，执行 `start transaction` 开启事务
- 依次执行事务中的每个DML操作
- 如果在执行的过程中的任何位置出现异常，则执行 `rollback` 回滚事务
- 如果事务中所有的DML操作都执行成功，则在最后执行 `commit` 提交事务

开启事务之后



```

create database db_test3;

use db_test3;
# 库存表
create table stock(
    id int primary key auto_increment,
    name varchar(200),
    num int not null
)
# 订单表
create table order_db(
    id int primary key auto_increment,
    name varchar(200) not null,
    price double,
    num int
);

insert into stock(name,num) values('鼠标',10);
insert into stock(name,num) values('键盘',20);
insert into stock(name,num) values('耳机',30);

# 开启事务
start transaction;

# 操作1: 扣减库存
update stock set num = num-1 where name = '鼠标';

select aaa; # 此处会执行失败

# 操作2: 新增订单
insert into order_db(name,price,num) values('鼠标',20.5,1);

# 事务回滚: 清除缓存中的操作, 撤销当前事务已经执行的操作
 rollback;

# 提交事务: 将缓存中的操作写入数据文件
 commit;

```

事务隔离级别

数据库允许多个事务并行，多个事务之间是隔离的、相互独立的；如果事务之间不相互隔离并且操作同一数据时，可能会导致数据的一致性被破坏。

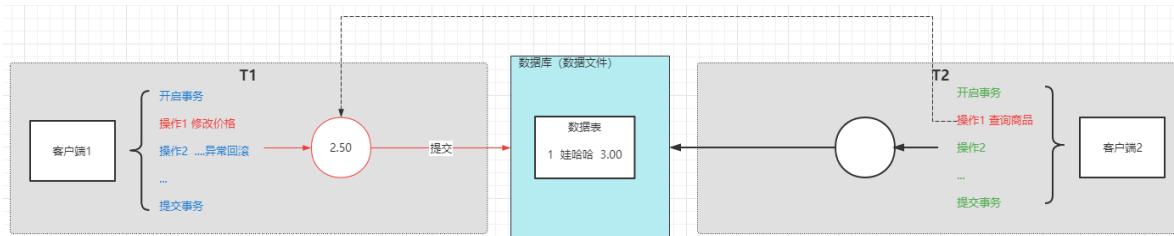
MySQL数据库事务隔离级别：

1. 读未提交
2. 读已提交
3. 可重复读
4. 串行化

读未提交 (read uncommitted)

T2可以读取T1执行但未提交的数据；可能会导致出现脏读

脏读，一个事务读取到了另一个事务中未提交的数据

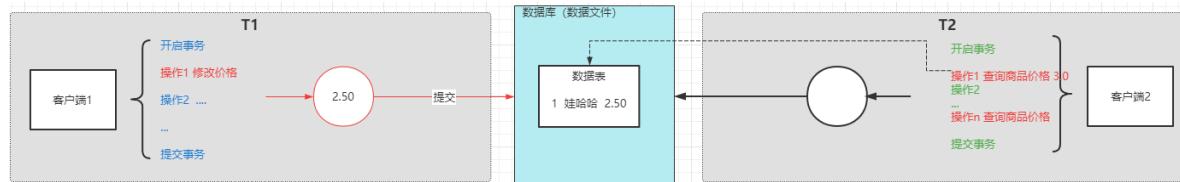


读已提交 (read committed)

T2只能读取T1已经提交的数据；避免了脏读，但可能会导致不可重复度（虚读）

不可重复度（虚读）：在同一个事务中，两次查询操作读取到数据不一致

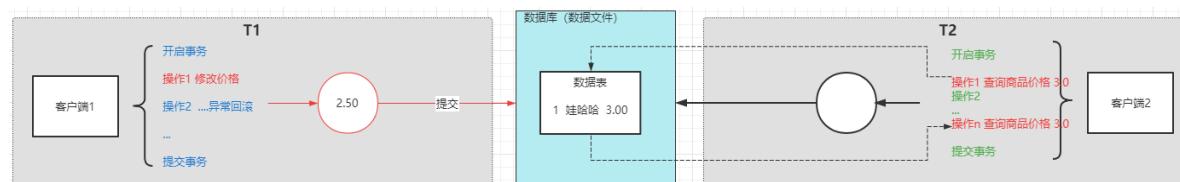
例如：T2进行第一次查询之后在第二次查询之前，T1修改并提交了数据，T2进行第二次查询时读取到的数据和第一次查询读取到的数据不一致。



可重复读 (repeatable read)

T2执行第一次查询之后，在事务结束之前其他事务不能修改对应的数据；避免了不可重复读(虚读)，但可能会导致幻读

幻读，T2对数据表中的数据进行修改然后查询，在查询之前T1向数据表中新增了一条数据，就导致T2以为修改了所有数据，但却查询出了与修改不一致的数据（T1事务新增的数据）



串行化(serializable)

同时只允许一个事务对数据表进行操作；避免了脏读、虚读、幻读问题

隔离级别	脏读	不可重复读(虚读)	幻读
read uncommitted	√	√	√
read committed	✗	√	√
repeatable read	✗	✗	√
serializable	✗	✗	✗

设置数据库事务隔离级别

我们可以通过设置数据库默认的事务隔离级别来控制事务之间的隔离性；

也可以通过客户端与数据库连接设置来设置事务间的隔离性（在应用程序中设置--Spring）；

MySQL数据库默认的隔离级别为 可重复读

- 查看MySQL数据库默认的隔离级别

```
-- 在MySQL8.0.3 之前
select @@tx_isolation;

-- 在MySQL8.0.3 之后
select @@transaction_isolation;
```

- 设置MySQL默认隔离级别

```
set session transaction isolation level <read committed>;
```

十四、数据库设计

数据库设计就是根据需求文档的描述将需求转成数据库的存储结构的过程，软件的开发一般都是分别从页面设计和数据库设计开始开发的，合理的数据库设计（数据表之间的关系）能够完成数据的存储，同时能够方便的提取应用系统所需的数据。

在数据库设计的流程上，我们通常根据需求画出数据的ER图，然后在通过ER图生成数据库的建库脚本。（Entity Relational）

ER图，所谓的ER图就是**数据库关系图**

为什么我们使用ER图来实现数据库设计的设计呢？

1. 可见即可得. 使用ER图可以通过图形的方式展示表与表直接的关系
2. 可以根据设置的数据库,方便生成不同的数据库的SQL建库脚本
3. 可以快速的生成数据库文档

所谓的数据库设计，就是通过ER图，根据需求给数据库建表结构

数据库设计流程

数据库是为应用系统服务的，数据库存储什么样的数据也是由应用系统来决定的。

数据库设计其本质就是根据需求设计**标识表**，包括表的字段，表与表之前的关系，这里有一个名词就是**标识表**

标识表注意事项

- 所谓标识表，就是根据需求创建表
- 标识表分为实体表与业务表
- 实体表：就是记录需求中描述为一个对象（名词）的表。如：用户、商品、订单、管理员、角色等
- 业务表：就是记录在需求中描述为一个业务行为（动词）的表：收藏、关注、等（大部分是中间表）
- 虽然没有强制的规定先标识实体表还是业务表，但通常创建标识表时会先创建标识实体表，再创建标识业务表，因为业务表一般是用于标识实体表与另一个实体的多对多的关系的。

具体步骤

1. 根据应用系统的功能，分析数据实体(实体，就是要存储的数据对象)

电商系统：商品、用户、订单....

教务管理系统：学生、课程、成绩...

2. 提取实体的数据项 (数据项，就是实体的属性)

商品(商品名称、商品图片、商品描述...)

用户(姓名、登录名、登录密码...)

3. 根据数据库设计三范式规范视图的数据项

检查实体的数据项是否满足数据库设计三范式

如果实体的数据项不满足三范式，可能会导致数据的冗余，从而引起数据维护困难、破坏数据一致性等问题

三大范式，就是用于数据库设计，标识字段的时候使用的

4. 绘制E-R图

实体关系图，直观的展示实体与实体之间的关系

5. 数据库建模

PowerDesigner

PDMan

6. 建库建表

编写SQL指令创建数据库、数据表

7. 添加测试数据，SQL测试

数据库设计三范式

第一范式：要求数据表中的字段（列）不可再分

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

以下表不满足第一范式（联系方式还可以分解）

姓名	性别	年龄	联系方式		
			手机号	qq	邮箱
不符合第一范式					

将细分的列作为单独的一列：

姓名	性别	年龄	手机号	qq	邮箱

第二范式：不存在非关键字段对关键字段的部分依赖

第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

以下表不满足第二范式

学生成绩表，此张表中还存在数据冗余，因为有重复的数据

学号、课程编号，是这张表的联合主键，我们把学号、课程编号称之为**关键字段**
除了关键字段以外的其他字段称之为**非关键字段**

将每个关键字段列出来\关键字段的组合也列出来，依次检查每个非关键字段，来构建新的表，以下就是上面这张表具体拆分出来的表

学号	姓名	性别	学生信息表
1001	张三	男	
1002	李四	女	
课程编号	课程名称		课程信息表
1	Java		
2	C++		
学号	课程编号	成绩	成绩信息表
1001	1	61	
1001	2	59	
1002	1	99	
1002	2	77	

第三范式：不存在非关键字段之间的传递依赖

确保每列都和主键列直接相关，而不是间接相关，比如以下的案例中就出现了间接关系，因为我们确认了学员编号就可以确认学员的班级编号，同时班级名称又依赖着班级编号，不满足第三范式

以下数据表不满足第三范式

学员编号为关键字段，但是班级名称可以直接通过班级编号来确认，而班级编号可以通过学员编号来确认，所以出现了传递依赖

班级名称 依赖 班级编号 依赖 学员编号

学员编号	姓名	性别	年龄	手机号	qq	邮箱	班级编号	班级名称

所以以上的情况我们可以进行拆分，将关键字段和被依赖的非关键字段分别作为主键，依次检查所有的非关键字段的依赖关系，分出多个表，通过班级编号作为外键，来关联学员表，而不是在学员表中出现班级相关的信息

学员编号	姓名	性别	年龄	手机号	qq	邮箱	班级编号
1001	小明	男	20	138xxxxxxxx	5555	1234@qq.com	1
1002	小李	女	18	139xxxxxxxx	6666	2134@qq.com	2
班级编号	班级名称						
1	1班						
2	2班						
3	3班						

注意：三大范式只是一般设计数据库的基本理念，可以建立冗余较小、结构合理的数据库。如果有特殊情况，当然要特殊对待，数据库设计最重要的是看需求跟性能，**需求>性能>表结构（范式）**。所以不能一味的去追求范式建立数据库。

例子：很多系统都要记录日志，而日志里面必须要包括用户的信息，如果严格按照三大范式，日志的用户信息必须是从用户表中获得，日志每天都会出现巨量的数据。如果关联用户表查询，整理日志时会导致用户表的访问大大被拖慢。

所以，我们会将用户的信息直接写在日志表里面。在日志表中写用户的的信息，明显违反了第二范式，基于查询的性能的需要，一般日志表的用户信息是冗余。

日志id	用户名	登录时间	登出时间
1	张三	2023/1/2	2023/2/4

小结：数据库设计总体上要符合三大范式，但是基于业务需求和性能要求，有时候可以有少许的冗余。数据设计冗余，**设计者必须要说明原因(项目需求需要)**

数据库建模 (E-R图)

E-R (Entity-Relationship) 实体关系图，用于直观的体现实体与实体之间的关联关系

数据库建模

E-R图实际上就是数据模建模的一部分：

- E-R 图 数据表设计 建库建表
- PDMan建模工具

下载安装PDMan: <http://www.pdman.cn/#/>



数据库建模

极简易用

数据库建模过程精细提炼，化繁为简，省去不必要的操作，只留下最需要的，直截了当的展现给用户。

自带案例

自带参考案例，以耳熟能详的[学生信息管理]为原型参考，让用户能够快速了解PDMan。

下载

选择版本



选择下载方式 (免费即可)

致下载用户

各位用户好：

PDManner自开源并提供免费下载以来，受到广大用户喜爱，非常感谢各位的支持。因为用户量大，给我们的流量带来巨大的挑战，我们团队自己出钱购买CDN以及流量加速的方式已经无法持续。

因此我们提供以下下载方案供你选择

下载方案一：关注公众号，获取免费下载码（CDN高速下载）



下载方案二：扫码付费加速（CDN高速下载）



下载方案三：一元以上随喜（超高速下载）



安装：傻瓜式安装方式，下一步即可（可以设置安装位置）

创建项目

设计一个学生成绩管理系统

1. 首先有学生，学生必须包括登录、禁用功能
2. 基于安全性的考虑，学生的身份信息要单独存储。
3. 一个学生有多门成绩
4. 一个学生可以有多个老师，一个老师也可以教多个学生

具体设计：

1. 标识表：学生、学生身份信息、成绩、老师
2. 标识字段
3. 设计表与表之间的关系

学生：学生id、学生姓名、学生账号、学生密码、学生状态

学生身份信息：身份证号、地址、电话、学生id（外键）

成绩表：id、科目、成绩、学生id（外键）

老师信息表：老师id、姓名、账号、密码、电话

业务表：老师和学员关系表：学生id、老师id

PdMan

1、创建项目

新建项目

* 项目名	stuScore
* 保存位置	E:\pdsouce
描述	
图标	像素为64*64，大小为10KB以内，可直接使用base64

确定 **取消**

选择关系表，创建新的关系图（ER图）这里连接对象选择字段，就可以通过字段进行连接数据

编辑关系图

* 关系图代码	stu.1
关系图名称	学生成绩系统物理关系图
连线对象	字段
所属主题域	

确定 **取消**

点击新建表，创建数据表

The screenshot shows a software interface for managing databases. At the top, there's a toolbar with various icons: Save, Refresh, New Empty Table (highlighted with a red box), Group, Shape, Mind Map, Align, Quick Tools, Import, Export, Settings, and Database. Below the toolbar, a search bar contains the text '学生成...'. The main area is titled 'TABLE_1' and shows a table definition for 'tb_student'. The table has five columns: 'stu_id', 'stu_name', 'stu_username', 'stu_password', and 'stu_status'. Each column has its data type (INT or VARCHAR), length (50), and decimal places (0) specified. There are also checkboxes for primary key, not null, and auto-increment.

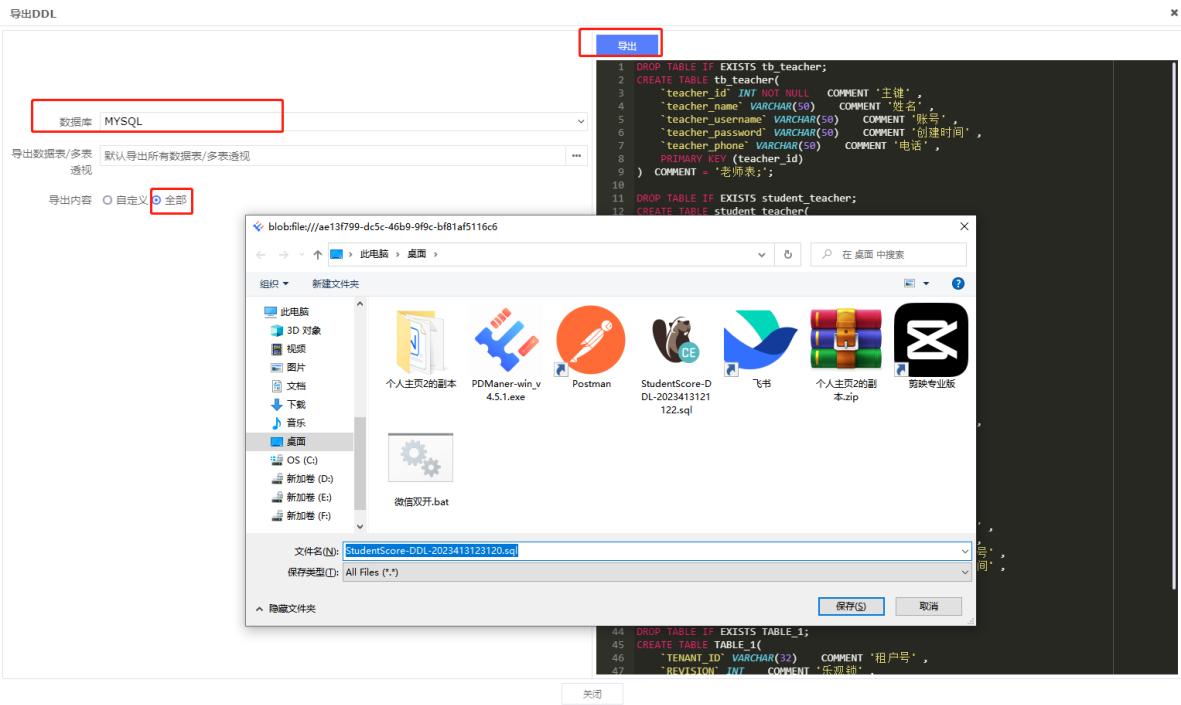
ER图



导出SQL脚本

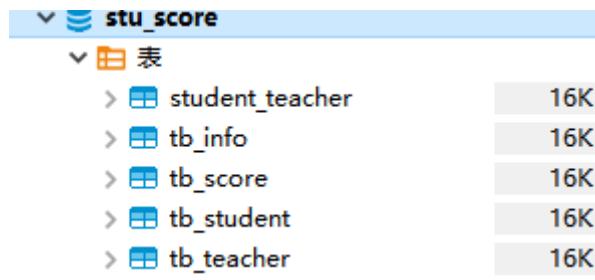


选择具体导出内容以及SQL脚本位置



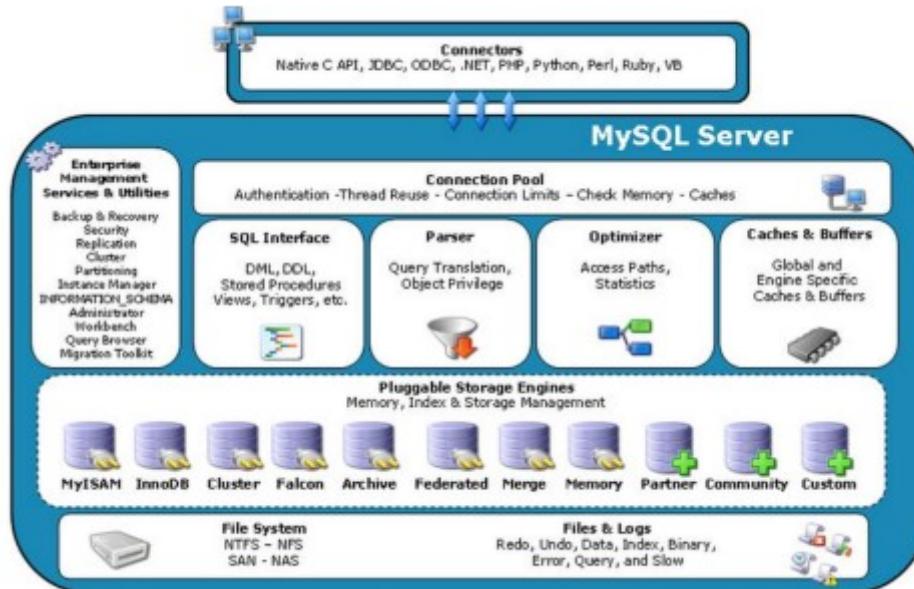
执行导出脚本

在MySQL中创建一个新的库，运行脚本即可



十五、存储引擎

MySQL体系结构



MySQL体系结构共分为4层：

1. 连接层：最上层是一些客户端和连接服务，包括大多数基于客户端/服务端工具实现的类似于TCP/IP的通信，主要功能是完成一些类似于连接处理、授权认证、安全方案等，在该层上还引入线程池的概念，为通过认证接入的客户端提供线程。
2. 服务层：这一层主要完成了大多数的核心服务功能，如SQL接口，SQL解析器，查询优化器、缓存、内置函数执行，所有跨存储引擎的功能也在这一层实现，比如DML、DDL语句封装、存储过程、视图等，都是在这一层完成的。
3. 引擎层：存储引擎层，存储引擎是真正负责MySQL中数据的储存和提取，服务器通过API和储存引擎进行通信。不同的引擎有不同的功能，我们可以根据自己的需求，来选取合适的存储引擎，并且数据库中的索引是在存储引擎层实现的。
4. 存储层：数据存储层，主要是将数据（日志、索引、二进制日志、错误日志、查询日志、慢查询日志等）存储在文件系统之上（存储到磁盘中），并可以完成与引擎的交互。MySQL它的架构可以在多种不同场景中应用，并发挥良好作用主要体现在储存引擎上，它是可插拔的储存引擎。

存储引擎

存储引擎就是存储数据、建立索引、更新/查询数据等技术的实现方案，引擎是基于表的，所以存储引擎也可以被称之为表类型。我们可以在创建表的时候，来指定选择的存储引擎，如果不指定则使用默认的引擎（InnoDB）。

为什么会有多种不同的引擎那？

因为引擎在我们生活中就是车、飞机、船等工具的发动机，也就是核心部件，不同的交通工具需要使用合适的引擎，所以MySQL中的引擎也是同样的道理。

默认引擎查看

```
use db_test3;
show create table order_db;
```

显示效果：

ABC Table	ABC Create Table
order_db	rice` double DEFAULT NULL, `num` int DEFAULT NULL, PRIMARY KEY (`id`) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

查询MySQL支持的所有引擎

```
show engines;
```

Engine	Support	Comment	Transactions	XA	Savepoints
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	[NULL]	[NULL]	[NULL]
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
ndbinfo	NO	MySQL Cluster system information storage engine	[NULL]	[NULL]	[NULL]
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
ndbcluster	NO	Clustered, fault-tolerant tables	[NULL]	[NULL]	[NULL]

- InnoDB：支持事务，具有提交，回滚和崩溃恢复能力，事务安全；
- MyISAM：不支持事务和外键，访问速度快；
- Memory：利用内存创建表，访问速度非常快，因为数据在内存，而且默认使用Hash索引，但是一旦关闭，数据就会丢失；
- Archive：归档类型引擎，仅能支持insert和select语句；
- Csv：以CSV文件进行数据存储，由于文件限制，所有列必须强制指定not null，另外CSV引擎也不支持索引和分区，适合做数据交换的中间表；
- BlackHole：黑洞，只进不出，进来消失，所有插入数据都不会保存；
- Federated：可以访问远端MySQL数据库中的表。一个本地表，不保存数据，访问远程表内容；
- MRG_MyISAM：一组MyISAM表的组合，这些MyISAM表必须结构相同，Merge表本身没有数据，对Merge操作可以对一组MyISAM表进行操作。

常见的引擎如下

1. InnoDB 默认
2. MyISAM MySQL早期版本默认引擎
3. MEMORY 数据存储在内存中，通常作用于缓存或者临时表

创建指定引擎的数据表

```
create table mytable(
    id int,
    name varchar(10)
)engine = MyISAM;

④ create table mytable(
    id int,
    name varchar(10)
)engine = MyISAM;
```

Table	Create Table
mytable	CREATE TABLE `mytable` (`id` int DEFAULT NULL, `name` varchar(10) DEFAULT NULL) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

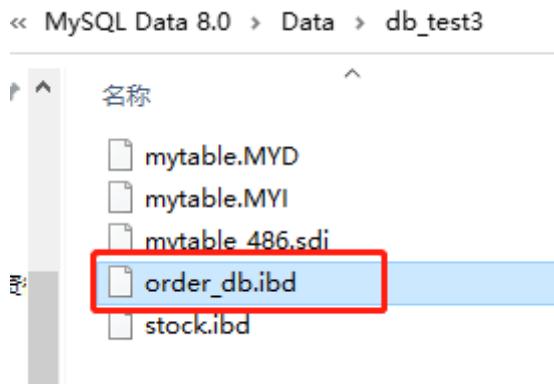
InnoDB

InnoDB是MySQL5.5版本以后默认的存储引擎，它是一种高可靠和高性能的通用储存引擎，它的特点如下：

- 支持事务。
- 行级锁，提高并发访问性能。
- 能缓存索引，也能够缓存数据。
- 支持外键约束。
- 5.6版本之后支持全文索引

在InnoDB中，表都是按照主键顺序组织存放的，称为索引组织表。在每张表中都有一个主键，如果创建表时没有显示的指定主键，那么InnoDB会判断表中是否有唯一索引，有就将此唯一索引作为主键，没有就自动创建6字节大小的指针作为主键。

InnoDB中的每一张表都对应一个表空间中，表空间由段、区、页组成。也就是创建每张表都会有一个(表名).ibd文件，那么只会存放每张表的数据、索引、表结构等内容。



我们可以通过一个系统参数来验证一下这个表空间，也就是一张表有一个独立的表空间：
innodb_file_per_table

```
show variables like 'innodb_file_per_table'; //查看此属性是否开启
```

show variables like 'innodb_file_per_table';	
<input type="text"/> inn_variables 1 × 输出	
variables like 'innodb_file_per_table'	输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)
Variable_name	Value
innodb_file_per_table	ON

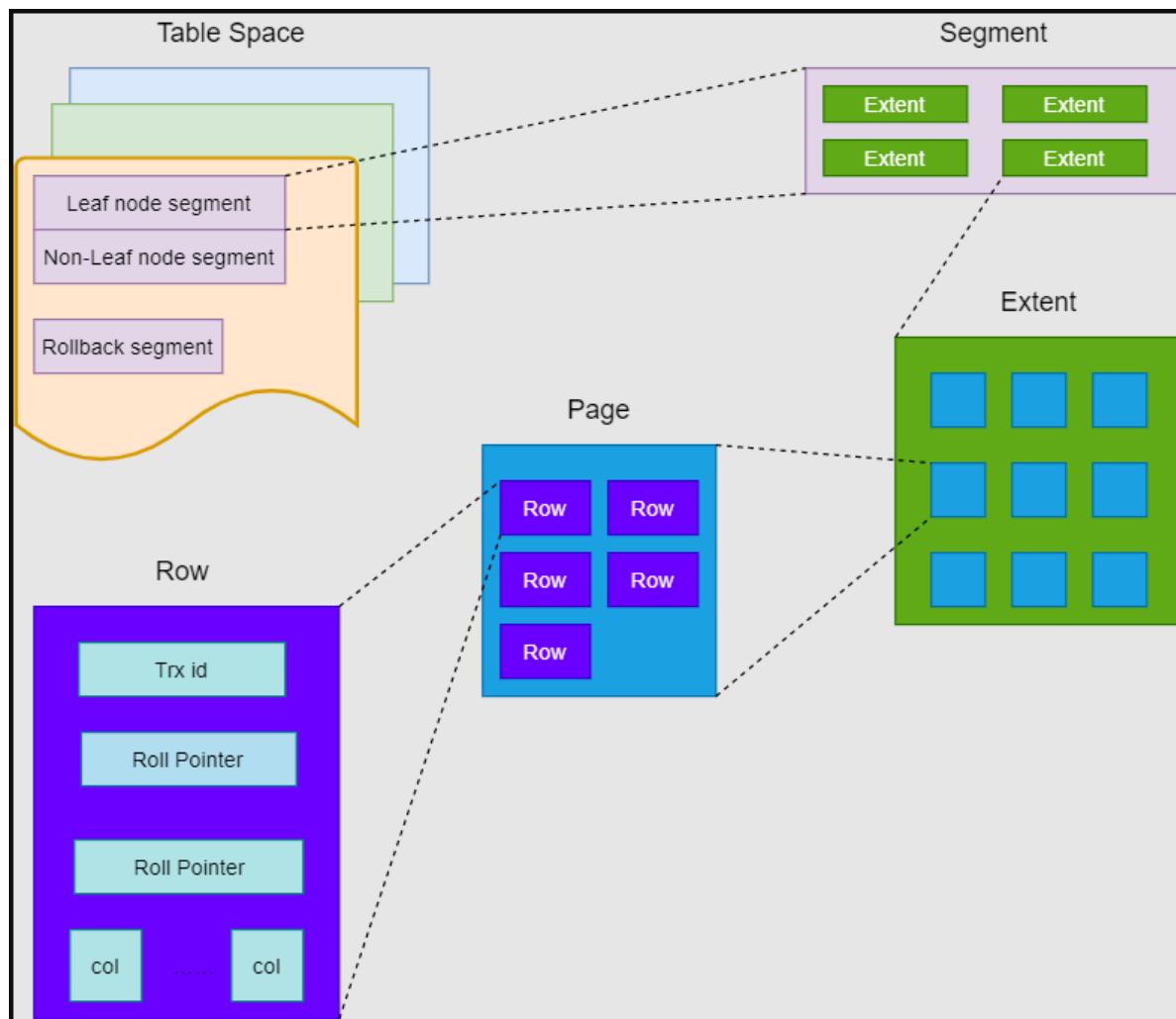
InnoDB逻辑存储结构

表由各个段组成。一个区默认大小为1M，可以存放64个页，每个页默认大小16KB。

- TableSpace：表空间，InnoDB存储引擎逻辑结构的最高层，.ibd文件其实就是表空间文件，在表空间中可以包含多个Segment段。
- Segment：段，表空间是由各个段组成的，常见的段有数据段、索引段、回滚段等。InnoDB中对于段的管

理，都是引擎自身完成，不需要人为对其控制，一个段中包含多个区。

- Extent：区，区是表空间的单元结构，每个区的大小为1M。默认情况下，InnoDB存储引擎页大小为16K，即一个区中一共有64个连续的页。
- Page：页，页是组成区的最小单元，页也是InnoDB存储引擎磁盘管理的最小单元，每个页的大小默认认为16KB。为了保证页的连续性，InnoDB存储引擎每次从磁盘申请4-5个区。
- Row：行，InnoDB存储引擎是面向行的，也就是说数据是按行进行存放的。



InnoDB使用场景

由InnoDB引擎的特性，我们可以简单归纳出InnoDB引擎适用的场景，主要有以下几点：

- 1、需要事务支持的业务。
- 2、有高并发需求的业务。
- 3、数据读写及更新都比较频繁的场景。

MyISAM

MyISAM是MySQL早期的默认引擎

特点如下

- 不支持事务
- 表级锁

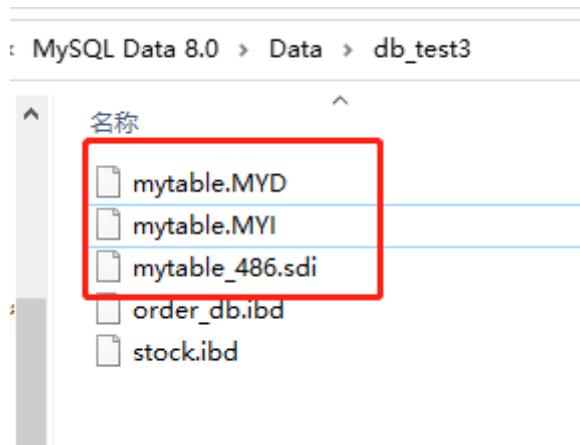
- 读写互相阻塞：在MyISAM类型的表中，既不可以在向数据表中写入数据的同时另一个会话也向该表中写入数据，也不允许其他的会话读取该表中的数据。只允许多个会话同时读取该数据表中的数据。
- 只缓存索引，不缓存数据：所谓缓存，就是指数据库在访问磁盘数据时，将更多的数据读取进入内存，这样可以使得当访问这些数据时，直接从内存中读取而不是再次访问硬盘。MyISAM可以通过key_buffer_size缓存索引，以减少磁盘I/O，提升访问性能。但是MyISAM数据表并不会缓存数据。
- 读取速度较快，占用资源少
- 不支持外键
- 支持全文索引

文件

xxx.sdi：存储表结构信息

xxx.MYD：储存数据

xxx.MYI：储存索引



MyISAM使用场景

- 1、不需要事务支持的场景。
- 2、读取操作比较多，写入和修改操作比较少的场景。
- 3、数据并发量较低的场景。
- 4、硬件条件比较差的场景。
- 5、在配置数据库读写分离场景下，MySQL从库可以使用MyISAM索引。

InnoDB引擎与MyISAM引擎的区别？

1. InnoDB引擎，支持事务，而MyISAM不支持。
2. InnoDB引擎，支持行锁和表锁，而MyISAM仅支持表锁，不支持行锁。
3. InnoDB引擎，支持外键，而MyISAM是不支持的

Memory

Memory引擎的表数据存储在内存中，所以导致会受到硬件问题、断电问题影响（一旦服务器关闭，数据就会消失），只能将Memory引擎的表作为临时表或者缓存来使用。

特点

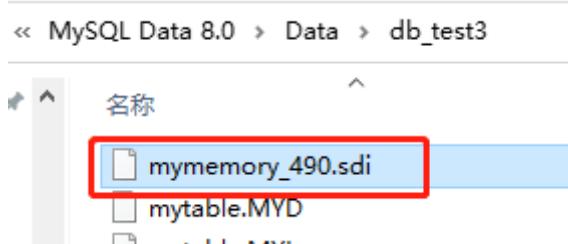
- 读写速度快，可大大提高查询效率

- 支持事务处理
- 内存限制，因为数据只能储存在内存中，所以数据不能过大，否则占用了大量的内存空间会影响性能

文件

```
create table myMemory(
    id int,
    name varchar(10)
)engine = Memory;
```

xxx.sdi：储存表结构信息



使用场景总结

实际上，目前最常用的就是InnoDB存储引擎，因为MyISAM与Memory的使用场景其实大部分都被NoSQL（非关系型数据）系列的数据库替代了，替代MyISAM的是MongoDB，而Memory被Redis替代了

十六、SQL优化

在应用系统开发初期，由于开发数据库数据比较少，对于查询SQL语句，复杂视图的编写等体会不出SQL语句各种写法的性能优劣，但是如果将应用系统提交实际应用后，随着数据库中数据的增加，系统的响应速度就成为目前系统需要解决的最主要的问题之一。系统优化中一个很重要的方面就是SQL语句的优化。对于海量数据，劣质SQL语句和优质SQL语句之间的速度差别可以达到上百倍，可见对于一个系统不是简单地能实现其功能就可，而是要写出高质量的SQL语句，提高系统的可用性。

插入优化

插入表中数据我们肯定会使用到Insert语句，那么如果想要插入多条数据，我们目前的做法就是写多条插入语句，比如

```
insert into stu values(1,'tom');
insert into stu values(2,'jerry');
insert into stu values(3,'zhangsan');
....
```

那么以上的这种执行方式如果在数据量比较庞大的时候，必然会影响执行的效率，因为你每执行一次insert就会与数据库服务器建立连接，进行网络数据传输，多次执行势必效率是比较慢的。

优化方案

批量插入，在一条insert语句中插入多条数据，但是如果数据量过大，也不能完全使用一条语句，建议数据量为一次性插入1000条以下的数据

```
insert into sut values(1,'tom'),(2,'jerry')....;
```

如果数据量过大，可以使用拆分为多条insert即可

```
insert into sut values(1,'tom'),(2,'jerry')....;
insert into sut values(1,'tom'),(2,'jerry')....;
insert into sut values(1,'tom'),(2,'jerry')....;
....
```

但是这里还要注意一个问题，因为MySQL本身是自动执行事务的，这也就意味着每次执行一条insert语句，就会进行一次开启事务和提交事务，那么会产生频繁的提交事务，所以我们在批量数据插入的时候，可以采用手动提交事务

```
start transaction;
insert into sut values(1,'tom'),(2,'jerry')....;
insert into sut values(1,'tom'),(2,'jerry')....;
insert into sut values(1,'tom'),(2,'jerry')....;
.....
commit;
```

还有一种优化手段就是，主键顺序插入，因为如果主键的顺序插入高于乱序插入（一般情况下都是顺序）

排序优化

其实在MySQL中的排序，要尽量避免使用Using filesort：文件排序（效率低）

在SQL优化中，我们希望尽可能不要出现文件排序，因为出现了文件排序意味着没有使用到索引构建好的排序，而是需要在内存中对字段进行重新排序，排序的过程是计算的过程比较消耗CPU。

多字段排序要尽量遵循**最左前缀原则**，而且不要对一个字段升序对另一个字段降序，否则也会使用到Using filesort

最左前缀原则： MySQL最左前缀原则是指在使用索引时，只有从左到右的索引列才能被用到，也就是说如果索引是 (a,b,c)，那么查询条件必须是a、a,b、a,b,c才能使用这个索引。

案例

我们来使用db_test数据库中的students表，首先我们先清空其中设置过的索引，然后我们来进行排序操作的，当然我们需要使用到explain（执行计划）来进行查看排序语句

```
explain select stu_num,stu_name,stu_tel,stu_age from students where
stu_name='tom' order by stu_age;
```

```
explain select stu_num,stu_name,stu_tel from students where stu_name='tom' order
```

id	abc select_type	abc table	abc partitions	abc type	abc possible_keys	abc key	abc key_len	abc ref	123 rows	123 filtered	abc Extra
1	SIMPLE	students	[NULL]	ALL	[NULL]	[NULL]	[NULL]	[NULL]	40	10	Using where; Using filesort

这里很明显能看出使用的是Using filesort（文件排序），效率是比较低的，所以这里我们就要进行优化

创建索引：针对 stu_name 和 stu_age 字段创建索引

```
create index index_students_stu_name_age on students(stu_name,stu_age);
```

创建成功之后，根据 stu_age 和 stu_tel 字段进行升序排序

```
explain select stu_num,stu_name,stu_tel,stu_age from students where stu_name='tom' order by stu_age;
```

提高效率：

```
lect stu_num,stu_name,stu_tel from students where stu_name='tom' order by
```

id	abc select_type	abc table	abc partitions	abc type	abc possible_keys	abc key	abc key_len	abc ref	123 rows	123 filtered	abc Extra
1	SIMPLE	students	[NULL]	ref	idx_students_num_age	idx_students_num_age	82	const	27	100	NULL

违反最左前缀原则

```
explain select stu_num,stu_name,stu_tel,stu_age from students order by stu_age;
```

```
explain select stu_num,stu_name,stu_tel,stu_age from students order by stu_age;
```

id	abc select_type	abc table	abc partitions	abc type	abc possible_keys	abc key	abc key_len	abc ref	123 rows	123 filtered	abc Extra
1	SIMPLE	students	[NULL]	ALL	[NULL]	[NULL]	[NULL]	[NULL]	40	100	Using filesort

在 MySQL8 版本中，支持降序索引（反向扫描）

```
explain select stu_num,stu_name,stu_tel,stu_age from students where stu_name='tom' order by stu_age desc;
```

```
explain select stu_num,stu_name,stu_tel,stu_age from students where stu_name='tom'
```

id	abc select_type	abc table	abc partitions	abc type	abc possible_keys	abc key	abc key_len	abc ref	123 rows	123 filtered	abc Extra
1	SIMPLE	students	[NULL]	ref	index_students_stu_name_index_stu	index_students_stu_name_index_stu	82	const	27	100	Backward index scan

关于 SQL 优化以及底层原理的内容，可以搜索千锋教育：千锋教育_MySQL 数据库高级教程，超详细 mysql 优化和原理分析，MySQL 优化项目教程

网址：https://www.bilibili.com/video/BV1h64y1y77i?p=1&vd_source=929e8428d83d4c51be5a1939d0d88870

十七、MySQL锁

简介

锁是计算机用以协调多个进程间并发访问同一共享资源的一种机制。MySQL中为了保证数据访问的一致性与有效性等功能，实现了锁机制，MySQL中的锁是在服务层或者存储引擎层实现的，以下我们所讲的所有锁的内容都是应用在InnoDB引擎中的。

锁解决的问题

锁是用来解决并发事务的访问问题，我们已经知道事务并发执行时可能带来的各种问题，最大的一个难点是：一方面要最大程度的利用数据库的并发访问，另外一方面还要确保每个用户能以一致的方式读取和修改数据，尤其是一个事务进行读取操作，另一个同时进行修改操作的情况下。

一个事务进行读取操作，另一个进行改动操作，我们前边说过，这种情况下可能发生脏读、不可重复读、幻读的问题。

锁的分类

MySQL中的锁是按照粒度分的，分为以下三种

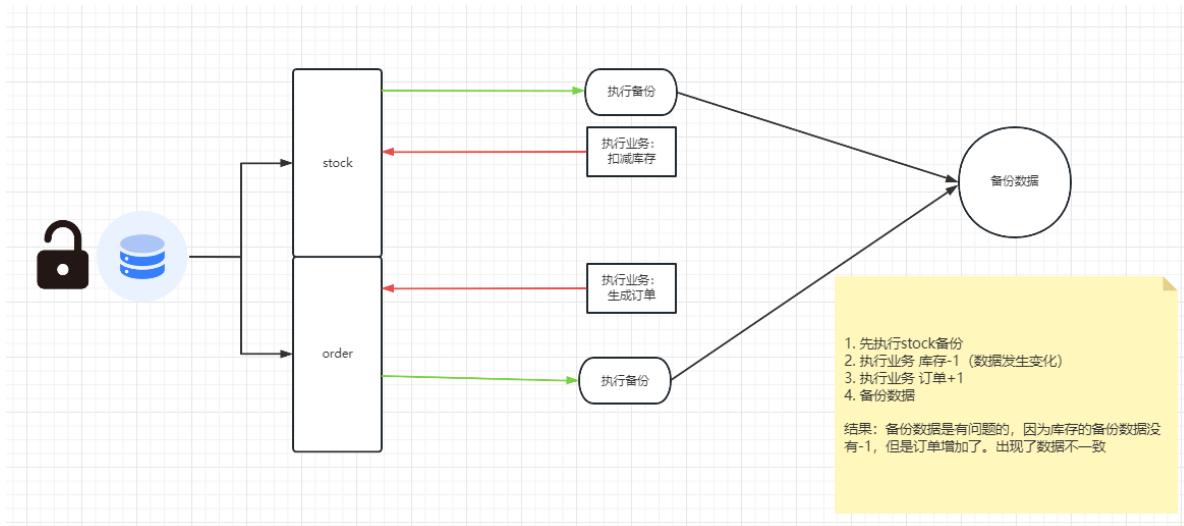
1. 全局锁：锁定数据库中的所有表
2. 表级锁：每次操作锁住操作表
3. 行级锁：每次操作锁住操作的行数据

全局锁

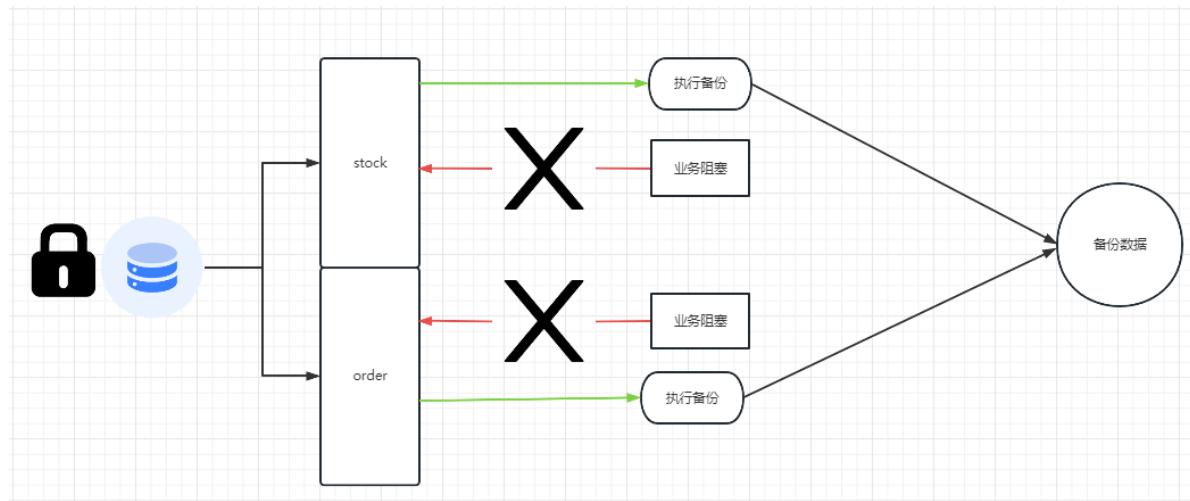
全局锁就是对整个数据库实例加锁，使整个库处于只读状态。使用该命令之后，数据更新语句、数据定义语句和更新类事务的提交语句等操作都会被阻塞。

典型的应用场景就是做全库逻辑备份，把所有表全部锁定，从而获取一致性视图，保证数据的完整性。

我们可以举一个例子：



以上案例中就出现了数据备份时，数据不一致的情况，那么如果加上全局锁，就可以解决



对数据库进行逻辑备份之前，先对整个数据库加上全局锁，此时数据库所有的表都变成了只读状态，所以DDL、DML全部都处于阻塞状态，但是可以执行DQL语句，而数据备份就是查询操作。

那么数据在进行逻辑备份的过程中，数据库中的数据就是不会发生变化的，这样就保证了数据的一致性和完整性。

具体使用

MySQL中可以通过一下语句添加全局锁

```
flush tables with read lock;
```

执行数据备份，利用MySQL提供的工具 mysqldump

```
mysqldump -uroot -proot 数据库名 > 具体存入脚本名称.sql
```

备份之后，解锁指令(释放全局锁)

```
unlock tables;
```

演示

我们来备份db_test数据库，并且一起演示一下全局锁的效果

1、首先我们在dbeaver中输入全局锁指令

```
flush tables with read lock;
```

2、然后我们可以通过命令行在开启一个MySQL的命令行客户端，来查看是否能够执行插入或者修改等DDL或者是DML的操作

```

MySQL 8.0 Command Line Client
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use db_test
Database changed
mysql> update students set stu_name='jerry' where stu_num=20230104;

```

此时执行的是修改操作，但是无法真正的完成，因为全局锁已经开启，此时无法执行修改操作，程序阻塞。

3、完成备份，将db_test表中的数据备份到D盘db_test.sql文件中，但是要注意此时我们使用的mysqldump是MySQL提供的工具，但并不是SQL指令，所以需要在win系统黑窗口中编写

注意：我这里的-P3307是指定端口号的，因为我电脑上有两个数据库，现在使用MySQL8.0端口为3307，如果不设置会导致找默认的3306端口

```
mysqldump -uroot -proot -P3307 test_db > D://db1.sql
```

```
C:\WINDOWS\system32>mysqldump -uroot -proot -P3307 test_db > D://db1.sql
mysqldump: [Warning] Using a password on the command line interface can be insecure.
mysqldump: Got error: 1049: Unknown database 'test_db' when selecting the database
C:\WINDOWS\system32>
```

解锁

```
unlock tables ;
```

此时再去执行修改操作，就可以正常完成了

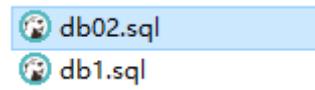
```
mysql> update students set stu_name='jerry' where stu_num=20230104;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

全局锁缺点

1. 在备份期间不能执行更新操作，业务基本上就会停摆。
2. 如果有主从复制+读写分离的结构，在备份期间从库不能执行主库同步过来的二进制日志(binlog)，会导致主从延迟。

官方自带的逻辑备份工具mysqldump，当mysqldump使用参数-single-transaction的时候，会启动一个事务，确保拿到一致性视图，也就是不加锁完成数据备份并且保证数据一致性。

```
mysqldump --single-transaction -uroot -proot -P3307 db_test > D://db02.sql
```



表级锁

表级锁，它的粒度为锁定整张表，每次操作锁住一张表，并且发生锁冲突的概率最高，并发度最低

在InnoDB、MyISAM等常见储存引擎中支持

表级锁还可以细分为三类：

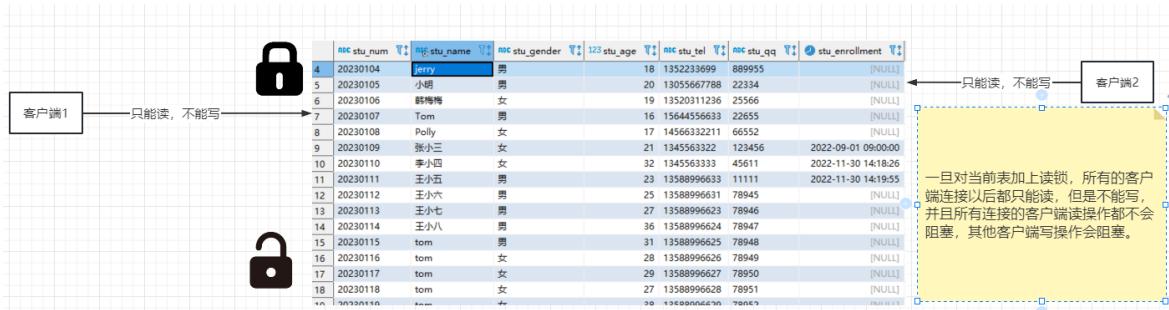
1. 表锁
2. 元数据锁(meta data lock, MDL)
3. 意向锁

表锁

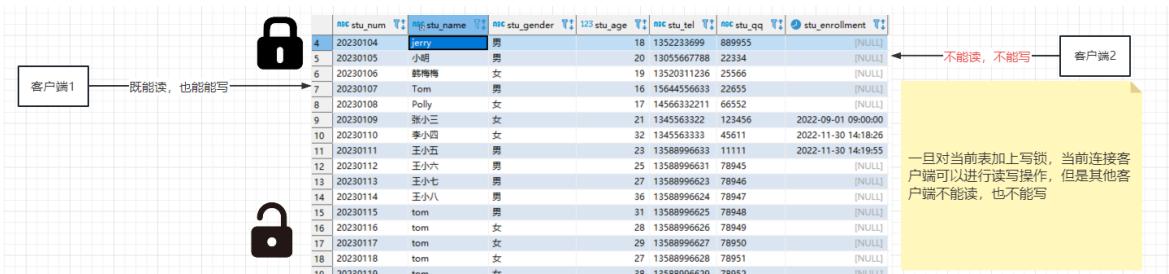
表锁还可以分为两类

1. 表共享读锁(read lock)
2. 表独占写锁(write lock)

读锁：



写锁



具体用法

加锁

```
lock tables 表名(可以是多张表) read/write
```

解锁（释放锁）

`unlock tables` 或 直接断开客户端连接（关闭客户端）

演示

我们来使用db_test库中的books表

	book_id	book_name	book_author
1	1	草房子	曹老师
2	3	Java	Mask

演示读锁

通过两个客户端来进行演示，客户端1连接以后，开启读锁，来看客户端2是否能够读取数据和写入数据，当然也可以在查看客户端1的读与写

```
lock tables books read;
unlock tables;
```

The screenshot shows two MySQL command-line clients. Client 1 (left) has a read lock on the 'books' table. Client 2 (right) attempts to update the 'book_name' column for book_id=1, but receives an error message: "Table 'books' was locked with a READ lock and can't be updated".

```

MySQL> use db_test;
Database changed
MySQL> lock tables books read;
MySQL> select * from books;
+----+-----+-----+
| book_id | book_name | book_author |
+----+-----+-----+
| 1 | 草房子 | 曹老师 |
| 2 | Java | Mask |
+----+-----+-----+
2 rows in set (0.00 sec)

MySQL> update books set book_name='JavaWeb' where book_id=1;
ERROR 1099 (HY000): Table 'books' was locked with a READ lock and can't be updated
MySQL>由于有读锁所以无法完成修改操作

```

```

MySQL> use db_test;
Database changed
MySQL> select * from books;
+----+-----+-----+
| book_id | book_name | book_author |
+----+-----+-----+
| 1 | 草房子 | 曹老师 |
| 2 | Java | Mask |
+----+-----+-----+
2 rows in set (0.00 sec)

MySQL> update books set book_name='高冷演讲' where book_id=1;
阻塞修改操作

```

当客户端1 释放锁以后

The screenshot shows the same two MySQL command-line clients. After client 1 releases the read lock, client 2 is able to successfully update the 'book_name' column for book_id=1.

```

MySQL> use db_test;
Database changed
MySQL> unlock tables;
MySQL>由于释放锁资源
MySQL>

```

```

MySQL> use db_test;
Database changed
MySQL> select * from books;
+----+-----+-----+
| book_id | book_name | book_author |
+----+-----+-----+
| 1 | 草房子 | 曹老师 |
| 2 | Java | Mask |
+----+-----+-----+
2 rows in set (0.00 sec)

MySQL> update books set book_name='JavaWeb' where book_id=1;
Query OK, 1 row affected (2 min 1.49 sec)
Rows matched: 1  Changed: 1  Warnings: 0
MySQL>阻塞解除, 执行修改成功

```

演示写锁

```
lock tables books write;
unlock tables;
```

MySQL 8.0 Command Line Client

```
mysql> use db_test
Database changed
mysql> lock tables books read;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='JavaWeb' where book_id = 3;
ERROR 1099 (HY000): Table 'books' was locked with a READ lock and can't be updated
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> lock tables books write; 加写锁
Query OK, 0 rows affected (0.00 sec)

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='MySQL' where book_id = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

MySQL 8.0 Command Line Client

```
mysql> Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 24
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use db_test;
Database changed
mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='高效演讲' where book_id=1;
Query OK, 1 row affected (2 min 1.48 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='MySQL' where book_id = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update books set book_name='Spring' where book_id = 3;
Query OK, 1 row affected (1 min 24.11 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

释放锁阻塞结束

MySQL 8.0 Command Line Client

```
mysql> use db_test
Database changed
mysql> lock tables books read;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='JavaWeb' where book_id = 3;
ERROR 1099 (HY000): Table 'books' was locked with a READ lock and can't be updated
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> lock tables books write;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='MySQL' where book_id = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

MySQL 8.0 Command Line Client

```
mysql> Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 24
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use db_test;
Database changed
mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='高效演讲' where book_id=1;
Query OK, 1 row affected (2 min 1.48 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from books;
+----+----+----+
| book_id | book_name | book_author |
+----+----+----+
| 1 | 高效演讲 | 曹老师 |
| 3 | Java | 马士兵 |
+----+----+----+
2 rows in set (0.00 sec)

mysql> update books set book_name='Spring' where book_id = 3;
Query OK, 1 row affected (1 min 24.11 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

表锁：元数据锁

元数据锁 (meta data lock, MDL)

元数据：简单理解为表结构

元数据锁实际上是系统自动控制的，不需要显示使用，当我们访问一张表的时候会自动加锁，它的主要作用就是维护表结构的数据一致性，当表中有活动事务存在时，是不允许操作元数据写入的

也就是如果有一张表有存在未提交的事务，不允许修改表结构，其目的就是为了避免DML与DDL冲突。

在MySQL5.5中引入了元数据锁 (MDL)，当一张表中数据进行增删改查时，加MDL读锁 (共享锁)，当表结构进行改变操作时，加MDL写锁 (排他锁)

简单理解，其实就是在每一个客户端中针对一张表做事务的处理DML操作，系统就会自动加上MDL读锁 (共享的)，另外的客户端读写DML操作同一张表操作是不受影响的，但是如果另外一张表要修改表结构也就是DDL操作时会自动加上MDL写锁，这个锁是一个排他锁，也就会导致我们第二个客户端无法修改表结构，阻塞发生，直到第一个客户端提交事务为止。

表锁：意向锁

意向锁是为了避免DML执行时，行锁与表锁的冲突，可以让表锁无需检查每行数据是否加锁，直接使用意向锁减少表锁的检查。

具体案例：

无意向锁：需要给当前表添加表锁，需要从头检查数据到尾，看是否有行锁，会不会产生冲突，然后再来添加

客户端1									客户端2								
begin.. 开启事务，执行 DML 语句时，会对涉及到的行添加行锁									lock tables students read/write								
update...	自动加行锁	4	20230104	jerry	男	18	1352233699	889955	[NULL]	5	20230105	小明	男	20	13055667788	22334	[NULL]
		6	20230106	薛梅梅	女	19	13520311236	25566	[NULL]	7	20230107	Tom	男	16	15644556633	22655	[NULL]
		8	20230108	Polly	女	17	14566332211	66552	[NULL]	9	20230109	张小三	女	21	1345563322	123456	2022-09-01 09:00:00
		10	20230110	李小四	女	32	13455633333	45611		11	20230111	王小五	男	23	13588996633	11111	2022-11-30 14:19:55
		12	20230112	王小六	男	25	13588996631	78945	[NULL]	13	20230113	王小七	男	27	13588996623	78946	[NULL]
		14	20230114	王小八	男	36	13588996624	78947	[NULL]	15	20230115	tom	男	31	13588996625	78948	[NULL]
		16	20230116	tom	女	28	13588996626	78949	[NULL]	17	20230117	tom	女	29	13588996627	78950	[NULL]
		18	20230118	tom	女	27	13588996628	78951	[NULL]	19	20230119	tom	女	29	13588996629	78952	[NULL]

有意向锁：加上意向锁以后，当其他客户端需要加表锁的时候，会根据意向锁判断是否可以成功添加，而不需要逐行判断了

意向锁判断依据就是是否兼容，如果是兼容的锁就可以直接加上，如果是排它锁就会阻塞，直到行锁释放

客户端1									客户端2								
begin.. 开启事务，执行 DML 语句时，会对涉及到的行添加行锁，同时对这个表加上意向锁									lock tables students read/write								
update...	自动加行锁	7	20230107	Tom	男	16	15644556633	22655	[NULL]	8	20230108	Polly	女	17	14566332211	66552	[NULL]
		9	20230109	张小三	女	21	1345563322	123456	2022-09-01 09:00:00	10	20230110	李小四	女	32	13455633333	45611	2022-11-30 14:18:26
		11	20230111	王小五	男	23	13588996633	11111		12	20230112	王小六	男	25	13588996631	78945	[NULL]
		13	20230113	王小七	男	27	13588996623	78946	[NULL]	14	20230114	王小八	男	36	13588996624	78947	[NULL]
		15	20230115	tom	男	31	13588996625	78948	[NULL]	16	20230116	tom	女	28	13588996626	78949	[NULL]
		17	20230117	tom	女	29	13588996627	78950	[NULL]	18	20230118	tom	女	27	13588996628	78951	[NULL]
		19	20230119	tom	女	29	13588996629	78952	[NULL]								

意向锁使用

意向锁分为两种类型：

- 意向共享锁(IS): 由语句select ... lock in share mode添加。与表锁共享锁(read)兼容，与表锁排他锁(write)互斥。
- 意向排他锁(IX): 由insert、update、delete、select...for update添加。与表锁共享锁(read)及排他锁(write)都互斥，意向锁之间不会互斥。

事务一旦提交，意向共享锁、意向排他锁，都会自动释放。

共享锁案例

我们现在需要给students表添加事务，并执行查询操作，同时使用lock in share mode (共享锁)，来添加行锁和意向共享锁

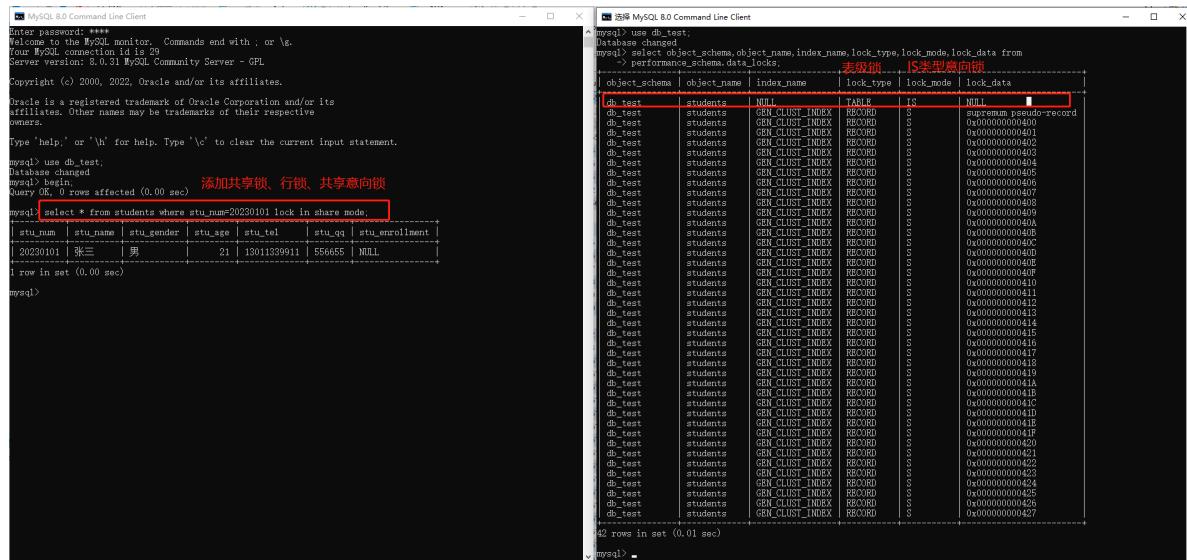
```

begin;
select * from students where stu_num=20230101 lock in share mode;//添加行
锁、和共享意向锁
-----  

## 此SQL可以查看意向锁及行锁加锁情况
select object_schema,object_name,index_name,lock_type,lock_mode,lock_data
from
performance_schema.data_locks;

```

注意lock_type属性，其中Table表示表级锁 RECORD表示行锁



```

mysql> use db_test;
Database changed
mysql> select * from students where stu_num=20230101 lock in share mode;
+----+----+----+----+----+----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel | stu_enrollment |
+----+----+----+----+----+----+
| 20230101 | 张三 | 男 | 21 | 13011339911 | 556655 | NULL |
+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql> select * from students where stu_num=20230101 lock in share mode;
+----+----+----+----+----+----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel | stu_enrollment |
+----+----+----+----+----+----+
| 20230101 | 张三 | 男 | 21 | 13011339911 | 556655 | NULL |
+----+----+----+----+----+----+
1 row in set (0.00 sec)

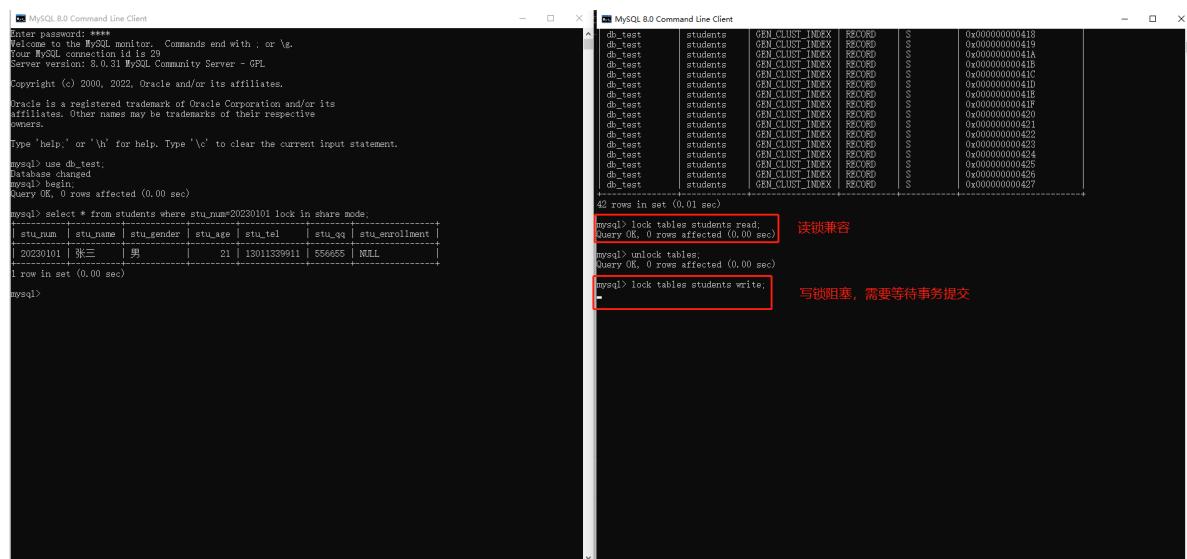
mysql>

```

object_schema	object_name	index_name	lock_type	lock_mode	lock_data
db_test	students	NULL	TABLE	IS	NULL
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000400
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000401
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000402
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000403
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000404
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000405
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000406
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000407
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000408
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000409
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040A
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040B
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040C
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040D
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040E
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000040F
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000410
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000411
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000412
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000413
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000414
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000415
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000416
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000417
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000418
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000419
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041A
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041B
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041C
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041D
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041E
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041F
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000420
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000421
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000422
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000423
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000424
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000425
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000426
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000427

现在我们再来给第二个客户端添加读锁与写锁

```
# read锁兼容 write会阻塞
lock tables students read/write;
```



```

mysql> use db_test;
Database changed
mysql> select * from students where stu_num=20230101 lock in share mode;
+----+----+----+----+----+----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel | stu_enrollment |
+----+----+----+----+----+----+
| 20230101 | 张三 | 男 | 21 | 13011339911 | 556655 | NULL |
+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql> select * from students where stu_num=20230101 lock in share mode;
+----+----+----+----+----+----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel | stu_enrollment |
+----+----+----+----+----+----+
| 20230101 | 张三 | 男 | 21 | 13011339911 | 556655 | NULL |
+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql>

```

object_schema	object_name	index_name	lock_type	lock_mode	lock_data
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000418
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000419
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041A
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041B
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041C
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041D
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041E
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x00000000041F
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000420
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000421
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000422
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000423
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000424
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000425
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000426
db_test	students	GEM_CLUST_INDEX	RECORD	S	0x000000000427

事务提交以后

```

mysql> select * from students where stu_num=20230101 lock in share mode;
+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel |
+-----+-----+-----+-----+-----+
| 20230101 | 张三 | 男 | 21 | 13011339911 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> update students set stu_age=18 where stu_num=20230101;
Query OK, 0 rows affected (0.00 sec) 提交事务，行锁和意向锁自动释放

mysql>

```

最后关闭锁即可

排它锁案例

当我们执行insert、update、delete等语句时就会自动添加行锁，并且会添加意向排它锁，此时另外的客户端无论是写还是读都会是阻塞状态

```

begin;
## 执行update语句会自动添加行锁和意向排它锁
update students set stu_age=18 where stu_num=20230101;

```

```

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from students where stu_num=20230101 lock in share mode;
+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel |
+-----+-----+-----+-----+-----+
| 20230101 | 张三 | 男 | 21 | 13011339911 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update students set stu_age=18 where stu_num=20230101;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>

```

此时我们在第二个客户端中添加read、write锁，都会阻塞，直到第一个客户端提交事务为止

```

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from students where stu_num=20230101 lock in share mode;
+-----+-----+-----+-----+-----+-----+
| stu_num | stu_name | stu_gender | stu_age | stu_tel | stu_enrollment |
+-----+-----+-----+-----+-----+-----+
| 20230101 | 张三 | 男 | 21 | 13011339911 | 556655 | NULL |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update students set stu_age=18 where stu_num=20230101;
Query OK, 1 row affected (16.54 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql>

42 rows in set (0.00 sec)

mysql> lock tables students read;
阻塞, write也会阻塞

```

行级锁

行锁也称为记录锁，顾名思义，就是锁住某一行（某条记录row）。需要注意的事，MySQL服务器层并没有实行行锁机制，行级锁只在储存引擎实现。

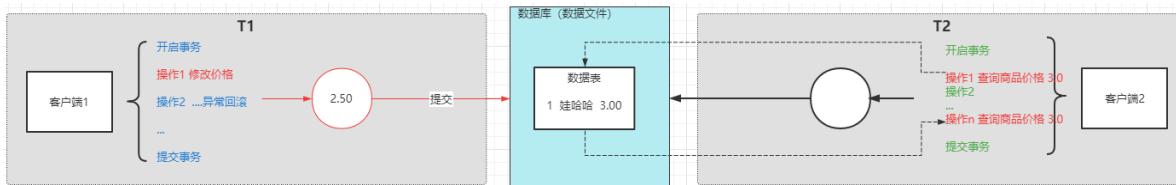
- 优点：锁的力度小，发生锁冲突概率低，可以实现高并发。
- 缺点：对于锁的开销比较大，加锁会比较慢。

InnoDB与MySAM的最大不同有三两点：一是支持事务（TRANSACTION）；二是采用了行级锁；三是InnoDB支持外键。

行级锁分类

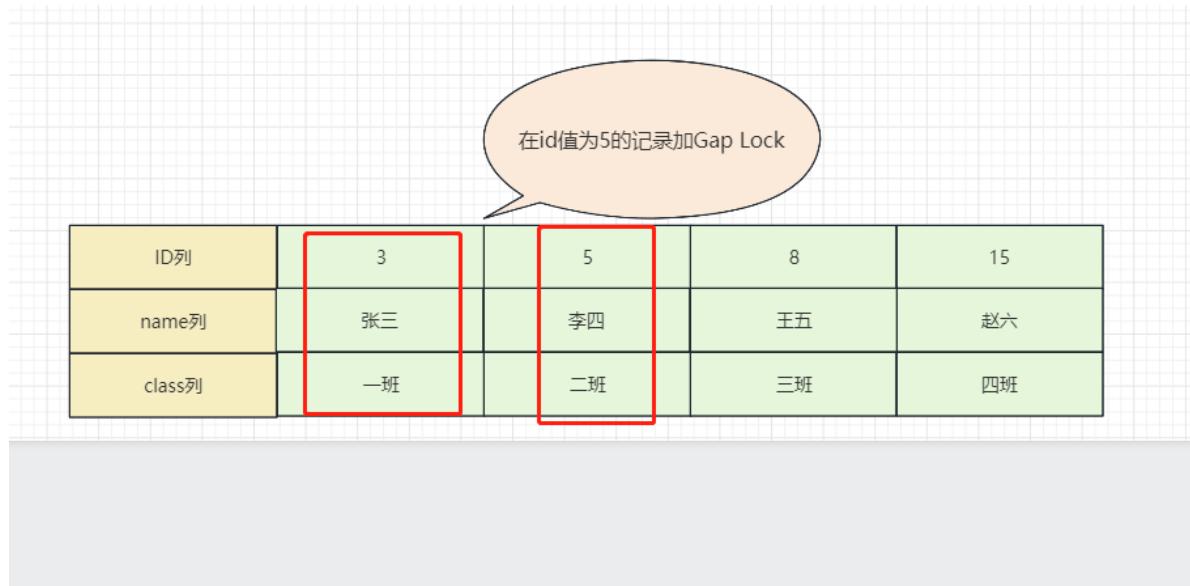
- 行锁、记录锁（Record Lock）：记录锁就是仅仅把一条记录锁上，比如锁住了某张表中id为8的记录，仅仅是锁住了一条记录而已，对其他数据没有影响，防止其他事务对这条数据进行update或者delete操作。
- 间隙锁（Gap Lock）：锁定索引记录间隙（不含该记录），确保索引记录间隙不变，防止其他事务在这个间隙进行insert，产生幻读。实际上这也是MySQL隔离级别可重复读隔离级别下解决幻读的一种方案

幻读：T2对数据表中的数据进行修改然后查询，在查询之前T1向数据表中新增了一条数据，就导致T2以为修改了所有数据，但却查询出了与修改不一致的数据（T1事务新增的数据）



图中id值为5的记录加了Gap锁，意味着不允许别的事务在id值为5的记录前边的间隙插入新记录，其实就是id列的值(3,5)这个区间的新记录是不允许立即插入的。比如，有另外一个事务再想插入一条id值为4的新记录，它定位到该条新记录的下一条记录的id值为5，而这条记录上又有一个Gap锁，所以就会阻塞插入操作，直到拥有这个gap锁的事务提交了之后，id列的值在区间(3,5)中的新记录才可以被插入。

Gap锁的提出仅仅是为了防止插入幻影记录而提出的。虽然有共享Gap锁和独占Gap锁这样的说法，但是它们起到的作用是相同的。而且如果对一条记录加了Gap锁(不论是共享Gap锁还是独占Gap锁)，并不会限制其他事务对这条记录加记录锁或者继续加Gap锁。



3. 临键锁 (Next-Key Lock) : 既想锁住某条记录，又想阻止其他事务在该记录前边的间隙插入新记录，就是行锁和间隙锁组合 (Next-Key Lock) 临键锁，innodb默认的锁就是Next-Key-locks。

行锁/记录锁

行锁的类型

在InnoDB引擎中实现了两种类型的行锁

1. 共享锁 (s) : 允许一个事务读取一行，阻止其他事务获得相同数据的排它锁。简单理解就是共享锁和共享锁兼容和排它锁互斥。
2. 排它锁 (x) : 允许获取排它锁的事务更新数据，阻止其他事务获取相同的排它锁和共享锁。

在锁定机制的实现过程中为了让行级锁定和表级锁定**共存**，InnoDB使用了**意向锁 (表级锁定)**的概念，也就有了意向共享锁和意向排他锁这两种。

意向共享锁可以同时并存多个，但是意向排他锁同时只能有一个存在。所以，可以说InnoDB的锁定模式实际上可以分为四种：

共享锁 (S) , 排他锁 (X) , 意向共享锁 (IS) 和意向排他锁 (IX)

锁模式兼容性

	共享锁 (S)	排它锁 (X)	意向共享锁 (IS)	意向排它锁 (IX)
共享锁 (S)	兼容	冲突	兼容	冲突
排它锁 (X)	冲突	冲突	冲突	冲突
意向共享锁 (IS)	兼容	冲突	兼容	兼容

	共享锁 (S)	排它锁 (X)	意向共享锁 (IS)	意向排它锁 (IX)
意向排它锁 (IX)	冲突	冲突	兼容	兼容

行级锁定实现方式

InnoDB行锁是通过给索引加锁来实现的。所以，只有通过索引条件检索数据，InnoDB才使用行级锁，否则不通过索引条件检索数据，InnoDB将使用表锁。其他注意事项：

- 在不通过索引条件查询的时候，InnoDB使用的是表锁，而不是行锁。
- 当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，另外，不论是使用主键索引、唯一索引或普通索引，InnoDB都会使用行锁来对数据加锁。

常见SQL语句执行时加锁情况

SQL	行锁类型	情况
Insert	排它锁	自动加锁
Update	排它锁	自动加锁
Delete	排它锁	自动加锁
select(普通)	无	不加任何锁
select Lock in share mode	共享锁	手动添加
select for update	排它锁	手动添加

演示

我们可以通过以下SQL来查看意向锁和行锁的加锁情况

```
select * from user where id=3;
-----
select object_schema,object_name,index_name,lock_type,lock_mode,lock_data
from
performance_schema.data_locks;
```

我们就通过db_test库中的user表来进行演示，依旧是打开两个客户端，因为本身此表存在主键，所以自然存在主键索引，那么我们就通过主键索引来检索数据和查看加锁情况

123 id ↑↓	ABC name ↑↓	123 age ↑↓
1 tom		18
3 jerry		20
8 张三		21
11 李四		11
19 王五		19
25 赵六		25

普通select无加锁

The image shows two side-by-side MySQL command-line client windows. Both windows have identical output for the command:

```
mysql> select * from user where id=3;
```

The output is:

id	name	age
3	Jerry	20

Both windows also show the text "普通SQL" (Normal SQL) and "无加锁情况" (No locking situation) in red at the bottom.

手动加锁，通过 lock in share mode

The image shows two side-by-side MySQL command-line client windows. The left window shows the command:

```
select * from user where id=3 lock in share mode;
```

The right window shows the results of a query to view table locks:

```
mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks;
```

object_schema	object_name	index_name	lock_type	lock_mode	lock_data
db_test	user	NULL	TABLE	IS	NULL
db_test	user	PRIMARY	RECORD	S,REC_NOT_GAP	3

Both windows show the text "手动加锁" (Manual Lock) and "表锁：共享意向锁 行锁：共享锁" (Table lock: Shared Intent Lock, Row lock: Shared Lock) in red at the bottom.

同时共享锁与共享锁之间可以兼容，所以在此执行此条语句，也是没有问题的

The image shows a single MySQL command-line client window. It contains the command:

```
select * from user where id=3 lock in share mode;
```

The screenshot shows two MySQL Command Line Client windows side-by-side. Both windows are connected to the same database, db_test.

Left Window (Client 1):

```

mysql> use db_test;
Database changed
mysql> select * from user where id=3;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

mysql> select * from user where id=3 lock in share mode;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

```

Right Window (Client 2):

```

Enter password: ****
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

mysql> use db_test;
Database changed
mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks;
Empty set (0.00 sec)

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from
    > performance_schema.data_locks;
+----+-----+---+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+----+-----+---+
| db_test       | user        | NULL       | TABLE     | IS      | NULL      |
| db_test       | user        | NULL       | RECORD    | S,REC_NOT_GAP | 3          |
+----+-----+---+
2 rows in set (0.00 sec)

mysql> select * from user where id=3 lock in share mode;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

```

A red box highlights the result of the second query in the right window, showing the row with id=3 has a lock type of 'IS' (Share Lock) and a lock mode of 'S,REC_NOT_GAP'. The value '20' is annotated with the text '不冲突' (No Conflict).

最后提交即可

演示共享锁与排它锁之间互斥，现在左侧客户端还是保持加共享行锁的状态，右侧客户端我们来开启一个新的事务，使用update语句，此语句会自动生成互斥锁，看一下是否能够成功执行

update user set name='qf'where id=3;

The screenshot shows two MySQL Command Line Client windows side-by-side. Both windows are connected to the same database, db_test.

Left Window (Client 1):

```

mysql> use db_test;
Database changed
mysql> select * from user where id=3;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

mysql> select * from user where id=3 lock in share mode;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

```

Right Window (Client 2):

```

+----+-----+---+
| id | name | age |
+----+-----+---+
| 3  | Jerry | 20 |
+----+-----+---+
1 row in set (0.00 sec)

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'data_locks' at line 1
mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from
    > performance_schema.data_locks;
+----+-----+---+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+----+-----+---+
| db_test       | user        | NULL       | TABLE     | IS      | NULL      |
| db_test       | user        | NULL       | RECORD    | S,REC_NOT_GAP | 3          |
+----+-----+---+
2 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> update user set name='qf'where id=3;

```

A red box highlights the update statement in the right window. Below the windows, a message states: '因为id=3这行有共享锁的存在所以update会自动添加互斥锁，所以不兼容，导致阻塞' (Because there is a shared lock on this row, the update will automatically add an exclusive lock, so it's incompatible, causing a deadlock).

注意：我们现在对id=3这条数据添加互斥锁会产生阻塞，但是其他的数据不会

排它锁演示，此时我们将两个客户端的操作全部提交commit，也就是释放所有的，此时我们在左侧客户端添加排它锁，使用update语句，右侧也同样使用update语句操作同一条数据，就会出现阻塞，因为排它锁互斥

The screenshot shows two MySQL Command Line Client windows side-by-side. Both windows are connected to the same database, db_test.

Left Window (Client 1):

```

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update user set name='qf' where id=1; 排它锁
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

```

Right Window (Client 2):

```

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update user set name='qf' where id=1; 排它锁
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

```

Both windows show a red box highlighting the update statements. A red box also highlights the 'Rows matched: 0' message in the right window, with the annotation '互斥导致阻塞' (Exclusive lock causes deadlock).

只有左侧客户端commit以后右侧才会结束阻塞。

演示无索引情况（变成表锁）

如果此时在左侧客户端执行以下SQL语句，会生成行锁吗？

```
update user set name='java' where name='qf';
```

```
MySQL> begin;
Query OK, 0 rows affected (0.00 sec)

MySQL> update user set name='java' where name='qf';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

MySQL> update user set name='qf' where id=3;
Query OK, 0 rows affected (0.00 sec)

MySQL> update user set name='java' where name='qf';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

MySQL> update user set name='qf' where id=3;
Query OK, 0 rows affected (0.00 sec)

MySQL>
```

此时使用的name字段没有索引，所以无法使用行锁，变成表锁

此时会导导致操作的并不是同一条数据，因为表锁的存在，锁定了整张表造成了阻塞

此时左侧客户端使用的是name字段作为更新的依据，但是name字段没有索引，所以导致行锁无法生效，变成了表锁，所导致右侧客户端在更新另外一条数据时，也会出现阻塞的情况，因为表锁生效，整表被锁。

间隙锁/临键锁

间隙锁（Gap Lock）：锁定索引记录间隙（不含该记录），确保索引记录间隙不变，防止其他事务在这个间隙进行insert，产生幻读。实际上这也是MySQL隔离级别可重复读隔离级别下解决幻读的一种方案

临键锁（Next-Key Lock）：既想锁住某条记录，又想阻止其他事务在该记录前边的间隙插入新记录，就是行锁和间隙锁组合（Next-Key Lock）临键锁，innodb默认的锁就是Next-Key-locks。

演示

唯一索引等值查询

- 当查询的记录是存在的，next-key lock 会退化成当前记录的「记录锁」。
- 当查询的记录是不存在的，next-key lock 会退化成当前记录所在区间的「间隙锁」。

打开两个客户端，然后用id来查询不存在的数据，查看加锁情况

```
update user set age = 30 where id = 7;

-----
select object_schema,object_name,index_name,lock_type,lock_mode,lock_data
from performance_schema.data_locks;
##阻塞
insert into user values(5,'jerry',10);
```

MySQL 8.0 Command Line Client

```

mysql> select * from user;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 1  | Java | 18 |
| 2  | 张三 | 20 |
| 3  | 张三 | 21 |
| 4  | 李四 | 30 |
| 5  | 李四 | 30 |
| 6  | 李四 | 19 |
| 7  | 李四 | 25 |
+----+-----+---+
7 rows in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update user set age = 30 where id = 7; 唯一索引等值 id=7不存在
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0 Changed: 0 Warnings: 0

mysql>

```

MySQL 8.0 Command Line Client

```

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks;
+-----+-----+-----+-----+-----+-----+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+-----+-----+-----+-----+-----+-----+
| db_test       | user        | NULL       | TABLE    | IX        | NULL      |
| db_test       | user        | PRIMARY    | RECORD   | X_GAP    | S         |
| db_test       | user        | NULL       | TABLE    | IX        | NULL      |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into user values(6,'Jerry',10); 锁住间隙，阻塞
Query OK, 1 row affected (0.00 sec)

mysql>

```

commit提交以后锁释放

非唯一索引等值查询（尽量避免）

- 当查询的记录存在时，除了会加 next-key lock 外，还额外对下一区间加「间隙锁」。

首先先来给age字段添加普通索引

```

create index idx_user_age on user(age);
select * from user where age=10 lock in share mode;

-----
select object_schema,object_name,index_name,lock_type,lock_mode,lock_data
from
performance_schema.data_locks;

```

MySQL 8.0 Command Line Client

```

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from user where age=10 lock in share mode; 加锁
+----+-----+---+
| id | name | age |
+----+-----+---+
| 6  | Jerry | 10 |
+----+-----+---+
1 row in set (0.00 sec)

mysql>

```

MySQL 8.0 Command Line Client

```

4 rows in set (0.00 sec)

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks;
+-----+-----+-----+-----+-----+-----+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+-----+-----+-----+-----+-----+-----+
| db_test       | user        | NULL       | TABLE    | IX        | NULL      |
| db_test       | user        | idx_user_age | RECORD   | X_GAP    | 10, 5     |
| db_test       | user        | PRIMARY    | RECORD   | S_REC_NOT_GAP | 10, 5     |
| db_test       | user        | idx_user_age | RECORD   | S_GAP    | 19, 19    |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>

```

唯一索引范围查询

- 对于给定范围内涉及到的值都加next-key lock，会访问到不满足条件的第一个值为止。

直接查询一个范围，并且添加共享锁，来看看

```
select * from user where id>8 lock in share mode;
```

锁定11,19,25这三条数据，还会锁定它们以上的间隙，并且supremum pseudo-record还会锁定（正无穷），也就是表示这个范围的数据全部锁定，包括上下的间隙也全部锁定。

The screenshot shows two MySQL command-line clients. The left client runs a query to insert a new row into the 'user' table. The right client then runs a query to check locks in the 'performance_schema.data_locks' table. A red box highlights the row for index 'idx_user_age' in the 'db_test' schema, which is locked with type 'S' (Share) and mode 'IS' (Intent Shared), with a lock range from 10 to 19, 19 to 25, and 25 to infinity. This indicates that the entire range of the index is locked.

```
mysql> select * from user;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 1  | Java  | 53 |
| 3  | Java  | 20 |
| 5  | Jerry | 10 |
| 6  | Jerry | 10 |
| 8  | 张三  | 21 |
| 11 | 李四  | 30 |
| 19 | 王五  | 19 |
| 25 | 钱六  | 25 |
+----+-----+---+
3 rows in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from user where id>8 lock in share mode;
+----+-----+---+
| id | name | age |
+----+-----+---+
| 11 | 李四  | 30 |
| 19 | 王五  | 19 |
| 25 | 钱六  | 25 |
+----+-----+---+
3 rows in set (0.00 sec)

mysql>

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from performance_schema.data_locks
-> where object_name='user' and index_name='idx_user_age';
+-----+-----+-----+-----+-----+-----+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+-----+-----+-----+-----+-----+-----+
| db_test       | user        | idx_user_age | RECORD   | IS        | NULL      |
| db_test       | user        | PRIMARY     | RECORD   | S         | 10, 5    |
| db_test       | user        | idx_user_age | RECORD   | S         | 10, 6    |
| db_test       | user        | PRIMARY     | RECORD   | S, REC_NOT_GAP | 6 |
| db_test       | user        | PRIMARY     | RECORD   | S, REC_NOT_GAP | 9 |
| db_test       | user        | idx_user_age | RECORD   | S, GAP    | 19, 19   |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select object_schema,object_name,index_name,lock_type,lock_mode,lock_data from
-> performance_schema.data_locks
+-----+-----+-----+-----+-----+-----+
| object_schema | object_name | index_name | lock_type | lock_mode | lock_data |
+-----+-----+-----+-----+-----+-----+
| db_test       | user        | NULL       | TABLE    | IS        | NULL      |
| db_test       | user        | PRIMARY    | RECORD   | S         | supremum pseudo-record |
| db_test       | user        | PRIMARY    | RECORD   | S         | 11       |
| db_test       | user        | PRIMARY    | RECORD   | S         | 19       |
| db_test       | user        | PRIMARY    | RECORD   | S         | 25       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> 11,19,25都是临键锁
supremum表示正无穷
```

加锁规则：

1. 唯一索引等值查询：

- 当查询的记录是存在的，next-key lock 会退化成 当前记录的「记录锁」。
- 当查询的记录是不存在的，next-key lock 会退化成 当前记录所在区间的「间隙锁」。

2. 非唯一索引等值查询（不建议使用）：

- 当查询的记录存在时，除了会加 next-key lock 外，还额外对上下区间加「间隙锁」。

3. 唯一索引范围查询：

- 对于给定范围内涉及到的值都加next-key lock，会访问到不满足条件的第一个值为止。

锁总结

- 什么是MySQL锁：在并发访问时，解决数据一致性、有效性问题的解决方案，MySQL中的锁是在服务器层或者存储引擎层实现的，我们所讲的所有锁的内容都是应用在InnoDB引擎中的。
- 锁分为三种：全局锁、表级锁、行级锁
- 全局锁：对整个数据库实例加锁，粒度最大，加锁后整个数据库实例处于只读状态，性能较差，一般用于数据库逻辑备份使用
- 表级锁：锁住整张操作的表，粒度也比较大，发生锁冲突概率比较高，具体分为：表锁（具体分为读和写两种锁）、元数据锁（主要作用就是针对DML语句和DDL语句的冲突）、意向锁（主要目的规避行锁与表锁的冲突问题，避免表锁再加锁时逐行扫描行锁，自动无需手动操作）
- 行级锁：锁住对应操作表的具体行数据（针对索引），粒度最小，发生锁冲突的概率最小，具体分为：行锁、间隙锁、临键锁，要注意的是其中共享锁与共享锁是可以兼容的，但是与排它锁，包括排它锁与排它锁之间都是互斥的。

十八、MySQL日志

所谓日志，就是一种将行为动作记录到一个地方，这个地方可以是文件，文本等可存储的载体。Mysql日志就是记录整个mysql从启动，运行，到结束的整个生命周期下的行为。

MySQL中的日志有很多种，主要我们要来了解以下两种

1. 错误日志
2. 二进制日志

错误日志

错误日志包含MySQL启动和关闭的时间记录，还包含诊断消息，比如服务器启动和关闭期间以及在服务器运行期间发生的错误、警告、注释消息，是比较重要的日志之一。

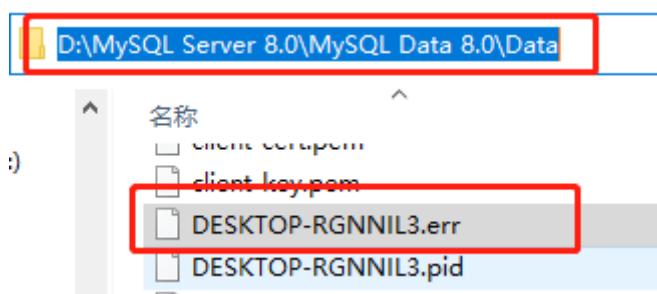
查看日志位置的指令

```
show variables like '%log_error%';
```

```
mysql> show variables like '%log_error%';
+-----+-----+
| Variable_name      | Value
+-----+-----+
| binlog_error_action | ABORT_SERVER
| log_error           | .\DESKTOP-RGNNIL3.err
| log_error_services  | log_filter_internal; log_sink_internal
| log_error_suppression_list | 错误日志位置和名称
| log_error_verbosity | 2
+-----+
5 rows in set, 1 warning (0.01 sec)

mysql>
```

win系统位置：



此日志可以通过编辑工具直接打开进行查看

```

2023-04-14T10:45:30.927097Z 0 [System] [MY-010951] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Ready for connections. Version: '8.0.31' socket:
    port: 3306 MySQL Community Server - GPL.
21 2023-04-14T10:45:30.928731Z 0 [System] [MY-013105] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Normal shutdown.
22 2023-04-14T10:45:30.972185Z 0 [System] [MY-010910] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Shutdown complete (mysqld 8.0.31) MySQL Community
23 2023-04-14T11:45:29.350963Z 0 [Warning] [MY-010915] [Server] 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql modes should be
used with strict mode. They will be merged with strict mode in a future release.
24 2023-04-14T11:45:29.355858Z 0 [System] [MY-010116] [Server] D:\MySQL Server 8.0\bin\mysqld.exe (mysqld 8.0.31) starting as process 9552
25 2023-04-14T11:45:29.474299Z 0 [Warning] [MY-013907] [InnoDB] Deprecated configuration parameters innodb_log_file_size and/or
innodb_log_files_in_group have been used to compute innodb redo log capacity=100663296. Please use innodb redo log capacity instead.
26 2023-04-14T11:45:29.524584Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
27 2023-04-14T11:45:30.244578Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
28 2023-04-14T11:45:30.764099Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
29 2023-04-14T11:45:30.764871Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported
for this channel.
30 2023-04-14T11:45:30.805822Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060
31 2023-04-14T11:45:30.806241Z 0 [System] [MY-010931] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: ready for connections. Version: '8.0.31' socket:
    port: 3307 MySQL Community Server - GPL.
32 2023-04-15T12:23:33.736129Z 0 [System] [MY-013105] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Normal shutdown.
33 2023-04-15T12:23:35.474295Z 0 [Warning] [MY-010909] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Forcing close of thread 18 user: 'root'.
34 2023-04-15T12:23:35.744020Z 0 [Warning] [MY-010909] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Forcing close of thread 12 user: 'root'.
35 2023-04-15T12:23:35.745544Z 0 [Warning] [MY-010909] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Forcing close of thread 13 user: 'root'.
36 2023-04-15T12:23:36.039141Z 0 [System] [MY-010910] [Server] D:\MySQL Server 8.0\bin\mysqld.exe: Shutdown complete (mysqld 8.0.31) MySQL Community
37 2023-04-15T12:24:16.612485Z 0 [Warning] [MY-010915] [Server] 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql modes should be
used with strict mode. They will be merged with strict mode in a future release.
38 2023-04-15T12:24:16.613958Z 0 [System] [MY-010116] [Server] D:\MySQL Server 8.0\bin\mysqld.exe (mysqld 8.0.31) starting as process 26556
39 2023-04-15T12:24:16.654756Z 0 [Warning] [MY-013907] [InnoDB] Deprecated configuration parameters innodb_log_file_size and/or
innodb_log_files_in_group have been used to compute innodb redo log capacity=100663296. Please use innodb redo log capacity instead.
40 2023-04-15T12:24:16.660026Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
41 2023-04-15T12:24:17.103993Z 0 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
42 2023-04-15T12:24:17.413609Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
43 2023-04-15T12:24:17.414299Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported
for this channel.

```

我们只需要去在此电脑->管理->服务->mysql服务，进行关闭就可以查看到日志的改变

二进制日志

二进制日志包含描述数据库更改的“事件”，例如表创建操作或表数据更改，还包含可能已进行更改的语句的事件（例如， DELETE不匹配任何行），但是不包含数据查询（select、 show）语句，MySQL8.0以后的版本此日志默认开启

作用

1. 复制操作(主从复制)，复制源服务器上的二进制日志，这里提供了要发送到副本的数据更改的记录。源将其二进制日志中包含的事件发送到其副本。
2. 数据恢复操作(灾备使用)，恢复备份后，将重新执行备份后记录的二进制日志中的事件，这些事件使数据库从备份点开始更新。

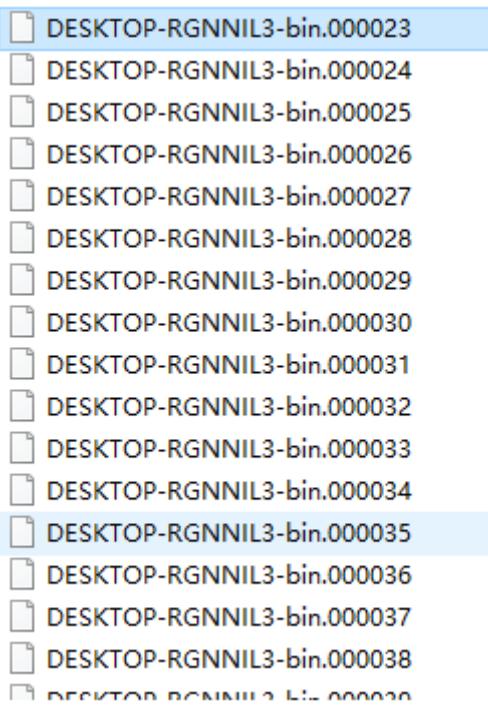
查看位置

```
show variables like '%log_bin%';
```

Variable_name	Value
log_bin	ON
log_bin_basename	D:\MySQL Server 8.0\MySQL Data 8.0\Data\DESKTOP-RGNNIL3-bin
log_bin_index	D:\MySQL Server 8.0\MySQL Data 8.0\Data\DESKTOP-RGNNIL3-bin.index
log_bin_trust_function_creators	OFF
log_bin_use_v1_row_events	OFF 第一个为开启
sql_log_bin	ON 第二个为具体位置

第三个是索引记录着当前日志关联的文件

日志文件名字会因为当前日志文件写满，而重新创建文件编写，所以二进制日志不会只有一个，多个文件就会有多个编号（编号从000001开始）



二进制日志格式

具体格式如下：

日志格式	具体含义
STATEMENT	基于SQL语句的日志记录，记录的是SQL语句，对数据进行修改的SQL都会记录在日志文件中
ROW	基于行的日志记录，记录的是每一行的数据变更（默认）
MIXED	混合了STATEMENT和ROW两种格式，默认采用STATEMENT，在某些特殊情况下会自动转换为STATEMENT

查看当前格式

```
show variables like '%binlog_format';
```

```
mysql> show variables like '%binlog_format';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| binlog_format | ROW  |
+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

那么当前为默认row格式，我们可以做个试验，更新一张表的数据，来查看日志中的具体内容，这里我们就是用user这张表

```
update user set age = 50 where id=1;
```

此时我们要查看最新的二进制日志，但是二进制文件是无法直接查看的，那么我们需要通过二进制查询工具mysqlbinlog来查看，那么具体格式如下：

```
mysqlbinlog [ 参数选项 ] 日志名称
```

参数选项:

- d** 指定数据库名称，只列出指定的数据库相关操作。
 - o** 忽略掉日志中的前n行命令。
 - v** 将行事件(数据变更)重构为SQL语句
 - vv** 将行事件(数据变更)重构为SQL语句，并输出注释信息

我们先来查看一下，注意因为我们当前是在Win系统中，所以使用MySQL指令需要通过cmd，同时为了方便找到日志文件，我们需要在日志所在目录下打开cmd

我的目录: D:\MySQL Server 8.0\MySQL Data 8.0\Data

mysqlbinlog DESKTOP-RGNNIL3-bin.000047

效果如下：

很明显我们无法看懂此文件，是因为我们需要将它转换为sql语句才可以，所以我们需要使用-v参数

具体指令如下：

```
mysqlbinlog -v DESKTOP-RGNNIL3-bin.000047
```

```

BINLOG ,
7Z46ZBMBAAAAAPgAAAIEBAAAAAFUAAAAAAEAB2RiX3R1c3QABHVzZXIAAwMPAwL8AwIBAQACA/z/
AbxZEBI=
7Z46ZB8BAAAAQgAAAMMBAAAAAFUAAAAAAEAAgAD//8AAQAAAQAamF2YRIAAAAAQAAAAQAmF2
YTIAAAChia9n
/*!*/;
## UPDATE db_test . user
## WHERE
##   @1=1          原始
##   @2='java'
##   @3=18
## SET
##   @1=1          更新数据
##   @2='java'
##   @3=50
at 451
#230415 20:56:13 server id 1  end_log_pos 482 CRC32 0xbe55314c  Xid = 25
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;

D:\MySQL Server 8.0\MySQL Data 8.0\Data>

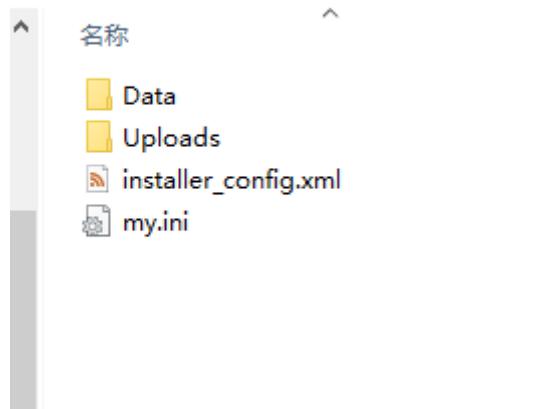
```

修改二进制日志格式

如果我们想修改二进制日志的格式，需要找到MySQL的配置文件my.ini，此文件目录一般会在MySQL Data 8.0目录下

D:\MySQL Server 8.0\MySQL Data 8.0

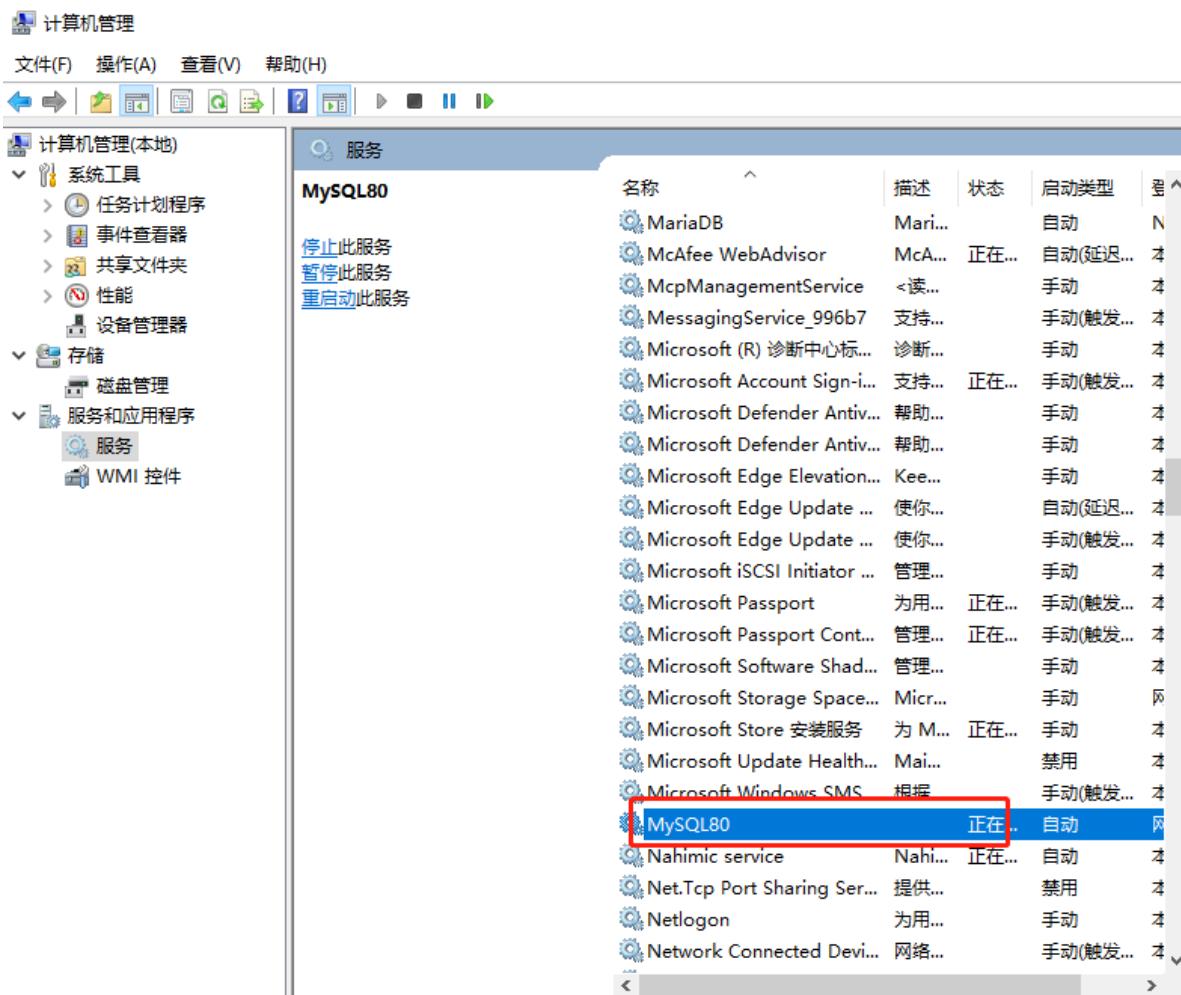
MySQL Server 8.0 > MySQL Data 8.0



然后打开文件在最后加上配置即可

```
binlog_format=STATEMENT
```

注意加上此配置以后，我们需要重启mysql服务 此电脑->管理->服务->mysql服务，重新启动配置才能生效



重启之后，重新执行一条修改SQL语句

```
update user set age = 53 where id=1;
```

此时会生成新的二进制日志，因为我们已经改变了二进制日志的格式，通过以下指令来查看二进制日志，注意此时不需要加参数-v因为STATEMENT格式是基于SQL语句的日志记录，记录的是SQL语句

```
mysqlbinlog DESKTOP-RGNNIL3-bin.000049
```

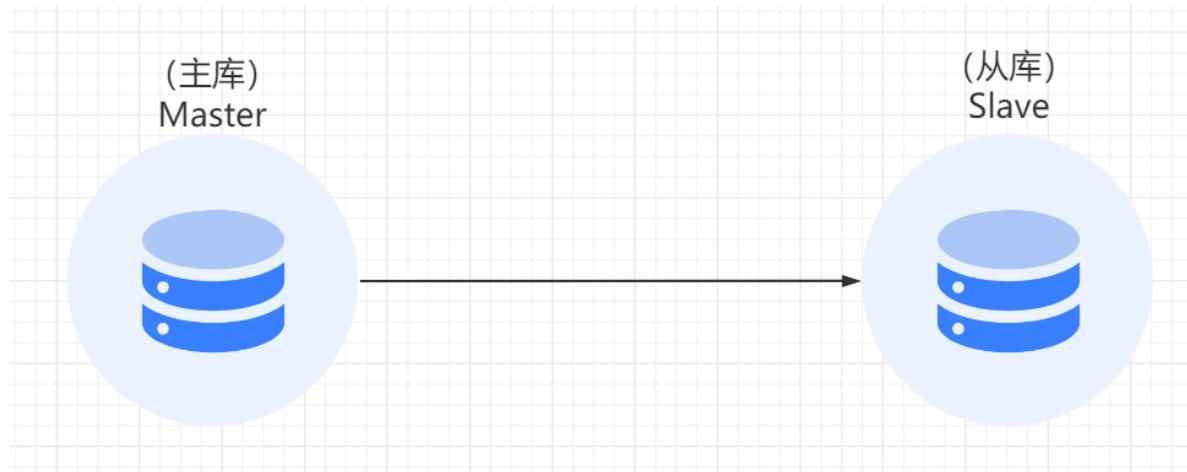
```
C:\Windows\System32\cmd.exe
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1/*!*/
SET @@session.sql_mode=107538976/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!*/\C gbk /*!*/;
SET @@session.character_set_client=28,@@session.collation_connection=28,@@session.collation_server=255/*!*/;
SET @@session_lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/*!*/180011 SET @@session.default_collation_for_utf8mb4=255/*!*/;
BEGIN
/*!*/;
# at 333
#230415 23:26:53 server id 1 end_log_pos 460 CRC32 0xb71b6d25 Query    thread_id=8      exec_time=0      error_code=0
use db_test /*!*/;
SET TIMESTAMP=1681572413/*!*/;
update user set age = 53 where id=1
/*!*/;
# at 460
#230415 23:26:53 server id 1 end_log_pos 491 CRC32 0x77da56ab  Xid = 35
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;

D:\MySQL Server 8.0\MySQL Data 8.0\Data>
```

十九、主从复制

简述

MySQL主从复制是一个异步的过程，其底层是基于MySQL数据库自带的二进制日志功能。就是一台或者多台MySQL数据库（Slave，从库），从另一台MySQL数据库（Master，主库）进行日志的复制，然后再解析日志到本身并应用，最终实现主库和从库数据库状态保持一致。



MySQL支持一台主库同时向多台从库进行复制，从库同时也可作为其他从库的主库，实现链状复制。

使用主从复制的好处

1. 主库出现问题，可以快速切换到从库提供服务，因为主库和从库的数据一致。
2. 可以实现读写分离，降低主库的访问压力，可以设置主库完成增删改的操作，从库完成查询。
3. 实现数据备份，可以在从库中执行备份，之前我们提到过，一旦要实现数据备份需要锁定这个数据库，如果有主从存在，可以锁定从库完成数据备份，从库在锁定期间还可以查询，主库增删改也不会受到影响，但是这期间可能会出现数据同步延迟的问题。

主从复制原理

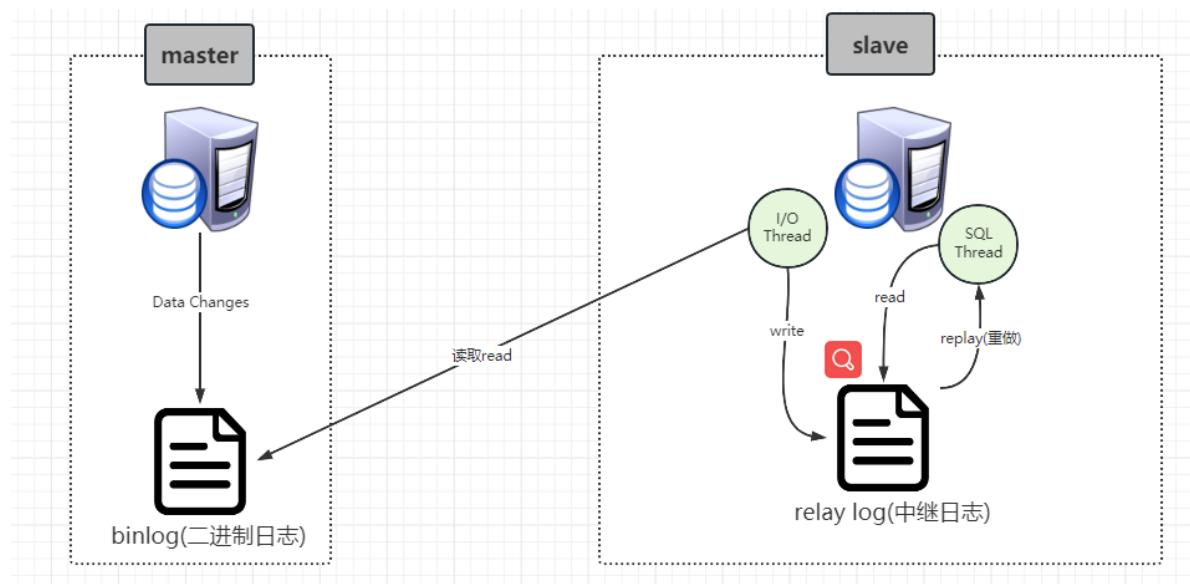
在linux系统中binlog为二进制日志

具体原理如下：

1. Master主库在事务提交时，会把数据变更记录在二进制日志Binlog中
2. slave读取主库的二进制日志，写入到slave的中继日志Relay Log中
3. slave重做中继日志中事件，将Master变更反映为自己的数据

复制的原则：

1. 每个slave只有一个master
2. 每个slave只能有一个唯一的服务器ID
3. 每个master可以有多个slave
4. master与slave数据库服务版本要保持一致



具体搭建

环境准备

1. 准备好两台Linux服务器，当然我们可以用两台虚拟机分别安装好MySQL，我这里的两台机器分比为：
 1. 192.168.189.130作为Master
 2. 192.168.189.131作为Slave
 3. 要注意两台服务器需要把防火墙全部关闭

```
systemctl stop firewalld //关闭防火墙
systemctl disable firewalld //防止防火墙开启自启动
```

3. 启动两台服务器的mysql服务

```
systemctl start mysql //启动MySQL服务命令
systemctl restart mysql //重启MySQL服务命令
systemctl stop mysql //关闭MySQL服务命令
```

查看linux版本MySQL的二进制文件位置和状态

linux版本的MySQL二进制日志文件的名称前缀为binlog (binglog.000001...)

我们需要在mysql中输入

```
show variables like '%log_bin%'; #查看二进制日志是否打开与文件位置
```

```
mysql> show variables like '%log_bin%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_bin | ON |
| log_bin_basename | /var/lib/mysql/binlog |
| log_bin_index | /var/lib/mysql/binlog.index |
| log_bin_trust_function_creators | OFF |
| log_bin_use_v1_row_events | OFF |
| sql_log_bin | ON |
+-----+-----+
6 rows in set (0.01 sec)
```

退出mysql，在系统中输入指令查看mysql二进制文件

```
cd /var/lib/mysql/binlog
```

主库配置

1. 修改主库的配置文件 /ect/my.cnf中

```
#主服务器唯一Id[必填]
server-id=1

#主机，读写都可以 0表示读写 1表示只读
read-only=0

#设置不要复制的数据库[可选]
#binlog-ignore-db=mysql

#设置需要复制的数据库[可选]（输数据库名字）
#binlog-do-db=test
```

打开主库进行配置192.168.3.119

```
vim /etc/my.cnf #打开配置文件并且编写配置
server-id=1
read-only=0
```

```

1 mysql-Server-1 x 2 mysql-Servlet-2 x +
# http://dev.mysql.com/doc/refman/8.0/en/server-configuration-defaults.html

[mysqld]
#
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove the leading "#" to disable binary logging
# Binary logging captures changes between backups and is enabled by
# default. It's default setting is log_bin=binlog
# disable_log_bin
#
# Remove leading # to set options mainly useful for reporting servers,
# The server defaults are faster for transactions and fast SELECTs,
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
#
# Remove leading # to revert to previous value for default_authentication_plugin,
# this will increase compatibility with older clients. For background, see:
# https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_default_authentication_plugin
# default-authentication-plugin=mysql_native_password

datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

server-id=1
read-only=0

```

更改完配置之后，需要重启数据库

注意：一旦重启出现报错，就证明配置文件有问题需要重新检查

```
systemctl restart mysqld
```

2. 建立账户并授权，此账号就是后续从库salve来连接主库的账号和密码

注意：设置的账号密码要求最低8位，并且包含大小写字母和特殊字符与数字组合

```
#创建用户 '@' '%' 代表用户可以再任何主机上访问当前MySQL服务器
CREATE USER 'qsf'@'%' IDENTIFIED BY 'Qf@123456';
#授权 'qf'@'%' 用户分配主从复制权限
GRANT REPLICATION SLAVE ON *.* TO 'qsf'@'%';
```

3. 执行指令，查看二进制日志坐标，这里的意思就是我们需要知道当前使用的是那个二进制日志，并且从那个位置开始推送数据

```
show master status;
```

这里要关注file与position

file表示具体文件为binlog.000003 二进制文件

position表示从此文件的什么位置开始推送数据

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File | 具体文件 | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+-----+
| binlog.000003 | 685 | 从哪个位置开始推送日志 |          |          |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

从库配置

1. 修改配置文件 /etc/my.cnf

```
#编写配置文件
vim /etc/my.cnf

#从服务器唯一ID[必填]
server-id=2

#从库，改为1只读 0表示读写 1表示只读
read-only=1

#重启服务
systemctl restart mysqld
```

注意：此时的只读只是针对普通用户的权限设置，但是当前从库超级管理员依旧可以进行读写，如果想让超级管理员也不能写，需要设置此属性

```
#设置Slave超级管理员读写权限（一般不需要）
super-read-only=1
```

3. 配置从库连接主库

到此位置，从库和主库的基本配置完成，但是此时master与slave并没有关联，所以我们需要设置从库去连接主库，也就是从库如何登陆主库和通过那个二进制文件来进行数据同步

这里我们需要执行一条sql语句（进入到MySQL中执行）

```
CHANGE REPLICATION SOURCE TO SOURCE_HOST='192.168.189.130',
SOURCE_USER='qf',
SOURCE_PASSWORD='Qf@123456', SOURCE_LOG_FILE='binlog.000003',
SOURCE_LOG_POS=685;
```

SOURCE_HOST 主库ip地址

SOURCE_USER 连接主库用户名

SOURCE_PASSWORD 连接主库密码

SOURCE_LOG_FILE 二进制binlog日志文件名

SOURCE_LOG_POS 二进制binlog日志文件位置

注意：如果你的数据库是8.0.23之前的版本需要执行以下SQL语句，当然新版本兼容老版本语句

```
CHANGE MASTER TO MASTER_HOST='主库ip地址', MASTER_USER='连接用户名',
MASTER_PASSWORD='连接密码', MASTER_LOG_FILE='二进制文件名',
MASTER_LOG_POS=具体文件位置;
```

4. 以上操作完成之后，我们需要开启同步操作

```
start replica; #8.0.22之后
start slave; #8.0.22之前
```

5. 查看主从同步状态

```
show replica status\G; #8.0.22之后
show slave status\G; #8.0.22之前
```

```
mysql> show replica status\G;
***** 1. row *****
    Replica_IO_State: Waiting for source to send event
        Source_Host: 192.168.189.130
        Source_User: qf
        Source_Port: 3306
        Connect_Retry: 60
        Source_Log_File: binlog.000006
        Read_Source_Log_Pos: 156
            Relay_Log_File: 192-relay-bin.000004
            Relay_Log_Pos: 321
        Relay_Source_Log_File: binlog.000006
            Replica_IO_Running: Yes
            Replica_SQL_Running: Yes
                Replicate_Do_DB:
                Replicate_Ignore_DB:
                Replicate_Do_Table:
                Replicate_Ignore_Table:
                Replicate_Wild_Do_Table:
            Replicate_Wild_Ignore_Table:
                Last_Error:
                Skip_Counter: 0
                Last_Erno: 0
                Last_Error:
                Skip_Counter: 0
            Exec_Source_Log_Pos: 156
            Relay_Log_Space: 528
            Until_Condition: None
            Until_Log_File:
            Until_Log_Pos: 0
        Source_SSL_Allowed: No
        Source_SSL_CA_File:
```

这里两个都为yes就是成功

测试

此时主从复制已经完成，现在我们就可以来测试一下，当我们在主库中执行增删改操作，从库会对应的同步数据，我们现在主库中执行一下脚本：

```
create database db_user;

use db_user;

CREATE TABLE user (
    id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name varchar(255) DEFAULT NULL,
    age int NOT NULL
);

INSERT INTO user VALUES (1, 'tom', 18);
INSERT INTO user VALUES (3, 'jerry', 20);
INSERT INTO user VALUES (8, '张三', 21);
INSERT INTO user VALUES (11, '李四', 11);
INSERT INTO user VALUES (19, '王五', 19);
INSERT INTO user VALUES (25, '赵六', 25);
```

```
mysql>
mysql> use db_user;
Database changed
mysql>
mysql> CREATE TABLE user (
    -> id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> name varchar(255) DEFAULT NULL,
    -> age int NOT NULL
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> INSERT INTO user VALUES (1, 'tom', 18);
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO user VALUES (3, 'jerry', 20);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO user VALUES (8, '张三', 21);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO user VALUES (11, '李四', 11);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO user VALUES (19, '王五', 19);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO user VALUES (25, '赵六', 25);
Query OK, 1 row affected (0.00 sec)

mysql> |
```

从库效果

```

mysql> show databases;
+-----+
| Database      |
+-----+
| db_user       |
| information_schema |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.00 sec)

mysql> use db_user;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_db_user |
+-----+
| user             |
+-----+
1 row in set (0.00 sec)

mysql> select * from user;
+---+---+---+
| id | name   | age  |
+---+---+---+
| 1  | tom    | 18   |
| 3  | jerry  | 20   |
| 8  | 张三    | 21   |
| 11 | 李四    | 11   |
| 19 | 王五    | 19   |
| 25 | 赵六    | 25   |
+---+---+---+

```

成功完成主从复制。

注意：我们现在只是对二进制日志指定位置进行的复制，如果想要主库全部数据，需要将主库导出sql脚本，到从库进行执行，这样能保证主库从库初始化数据一致。