#### 数据库基本概念

【1】数据库基本概念

（1）数据

所谓数据（Data）是指对客观事物进行描述并可以鉴别的符号，这些符号是可识别的、抽象的。它不仅仅指狭义上的数字，而是有多种表现形式：字母、文字、文本、图形、音频、视频等。现在计算机存储和处理的数据范围十分广泛，而描述这些数据的符号也变得越来越复杂了。

（2）数据库

数据库（Database，DB）指的是以一定格式存放、能够实现多个用户共享、与应用程序彼此独立的数据集合。

（3）数据库管理系统

数据库管理系统（Database Management System，DBMS）是用来定义和管理数据的软件。如何科学的组织和存储数据，如何高效的获取和维护数据，如何保证数据的安全性和完整性，这些都需要靠数据库管理系统完成。目前，比较流行的数据库管理系统有：Oracle、MySQL、SQL Server、DB2等。

（4）数据库应用程序

数据库应用程序（Database Application System，DBAS）是在数据库管理系统基础上，使用数据库管理系统的语法，开发的直接面对最终用户的应用程序，如学生管理系统、人事管理系统、图书管理系统等。

（5）数据库管理员

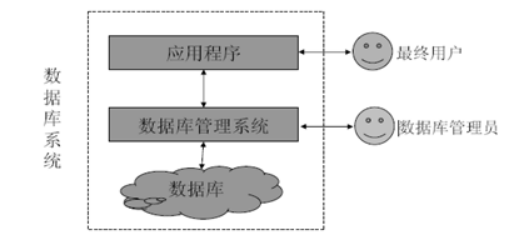
数据库管理员（Database Administrator，DBA）是指对数据库管理系统进行操作的人员，其主要负责数据库的运营和维护。

（6）最终用户

最终用户（User）指的是数据库应用程序的使用者。用户面向的是数据库应用程序（通过应用程序操作数据），并不会直接与数据库打交道。

（7） 数据库系统

数据库系统（Database System，DBS）一般是由数据库、数据库管理系统、数据库应用程序、数据库管理员和最终用户构成。其中DBMS是数据库系统的基础和核心。



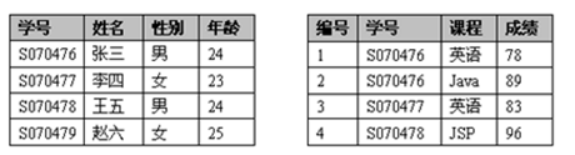
#### 数据库类型和常见的关系型数据库

【1】数据库类型

数据库经过几十年的发展，出现了多种类型。根据数据的组织结构不同，主要分为网状数据库、层次数据库、关系型数据库和非关系型数据库四种。目前最常见的数据库模型主要是：关系型数据库和非关系型数据库。

1. 关系型数据库

关系型数据库模型是将复杂的数据结构用较为简单的二元关系（二维表）来表示，如图1-4所示。在该类型数据库中，对数据的操作基本上都建立在一个或多个表格上，我们可以采用结构化查询语言（SQL）对数据库进行操作。关系型数据库是目前主流的数据库技术，其中具有代表性的数据库管理系统有：Oracle、DB2、SQL Server、MySQL等。



PS：关系=二维表

2. 非关系型数据库NOSQL

NOSQL（Not Only SQL）泛指非关系型数据库。关系型数据库在超大规模和高并发的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题。NOSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。常见的非关系型数据库管理系统有Memcached、MongoDB，redis，HBase等。

【2】常见的关系型数据库

虽然非关系型数据库的优点很多，但是由于其并不提供SQL支持、学习和使用成本较高并且无事务处理，所以本书的重点是关系型数据库。下面我们将介绍一下常用的关系型数据库管理系统。

1. Oracle

Oracle数据库是由美国的甲骨文（Oracle）公司开发的世界上第一款支持SQL语言的关系型数据库。经过多年的完善与发展，Oracle数据库已经成为世界上最流行的数据库，也是甲骨文公司的核心产品。

Oracle数据库具有很好的开放性，能在所有的主流平台上运行，并且性能高、安全性高、风险低；但是其对硬件的要求很高、管理维护和操作比较复杂而且价格昂贵，所以一般用在满足对银行、金融、保险等行业大型数据库的需求上。

2. DB2

DB2是IBM公司著名的关系型数据库产品。DB2无论稳定性，安全性，恢复性等等都无可挑剔，而且从小规模到大规模的应用都可以使用，但是用起来非常繁琐，比较适合大型的分布式应用系统。

3. SQL Server

SQL Server是由Microsoft开发和推广的关系型数据库，SQL Server的功能比较全面、效率高，可以作为中型企业或单位的数据库平台。SQL Server可以与Windows操作系统紧密继承，无论是应用程序开发速度还是系统事务处理运行速度，都能得到大幅度提升。但是，SQL Server只能在Windows系统下运行，毫无开放性可言。

4. MySQL

MySQL是一种开放源代码的轻量级关系型数据库，MySQL数据库使用最常用的结构化查询语言（SQL）对数据库进行管理。由于MySQL是开放源代码的，因此任何人都可以在General Public License的许可下下载并根据个人需要对其缺陷进行修改。

由于MySQL数据库体积小、速度快、成本低、开放源码等优点，现已被广泛应用于互联网上的中小型网站中，并且大型网站也开始使用MySQL数据库，如网易、新浪等。

#### MySQL介绍

MySQL数据库最初是由瑞典MySQL AB公司开发，2008年1月16号被Sun公司收购。2009年，SUN又被Oracle收购。MySQL是目前IT行业最流行的开放源代码的数据库管理系统，同时它也是一个支持多线程高并发多用户的关系型数据库管理系统。MySQL之所以受到业界人士的青睐，主要是因为其具有以下几方面优点：

1. 开放源代码

MySQL最强大的优势之一在于它是一个开放源代码的数据库管理系统。开源的特点是给予了用户根据自己需要修改DBMS的自由。MySQL采用了General Public License，这意味着授予用户阅读、修改和优化源代码的权利，这样即使是免费版的MySQL的功能也足够强大，这也是为什么MySQL越来越受欢迎的主要原因。

2. 跨平台

MySQL可以在不同的操作系统下运行，简单地说，MySQL可以支持Windows系统、UNIX系统、Linux系统等多种操作系统平台。这意味着在一个操作系统中实现的应用程序可以很方便地移植到其他的操作系统下。

3. 轻量级

MySQL的核心程序完全采用多线程编程，这些线程都是轻量级的进程，它在灵活地为用户提供服务的同时，又不会占用过多的系统资源。因此MySQL能够更快速、高效的处理数据。

4. 成本低

MySQL分为社区版和企业版，社区版是完全免费的，而企业版是收费的。即使在开发中需要用到一些付费的附加功能，价格相对于昂贵的Oracle、DB2等也是有很大优势的。其实免费的社区版也支持多种数据类型和正规的SQL查询语言，能够对数据进行各种查询、增加、删除、修改等操作，所以一般情况下社区版就可以满足开发需求了，而对数据库可靠性要求比较高的企业可以选择企业版。

另外，PHP中提供了一整套的MySQL函数，对MySQL进行了全方位的强力支持。

总体来说，MySQL是一款开源的、免费的、轻量级的关系型数据库，其具有体积小、速度快、成本低、开放源码等优点，其发展前景是无可限量的。

PS：社区版与企业版主要的区别是：

1. 社区版包含所有MySQL的最新功能，而企业版只包含稳定之后的功能。换句话说，社区版可以理解为是企业版的测试版。

2.MySQL官方的支持服务只是针对企业版，如果用户在使用社区版时出现了问题，MySQL官方是不负责任的。

#### MySQL8的安装

【1】MySQL的版本：

近期主要历史版本有5.0/5.1/5.5/5.6/5.7，目前最新版本是MySQL8。6.0曾经是个内部试验版本，已取消了。

MySQL8.0的版本历史

1) 2016-09-12第一个DM(development milestone)版本8.0.0发布

2) 2018-04-19第一个GA(General Availability)版本开始，8.0.11发布

3) 2018-07-27 8.0.12GA发布

4) 2018-10-22 8.0.13GA发布

5) 2019-01-21 8.0.14GA发布

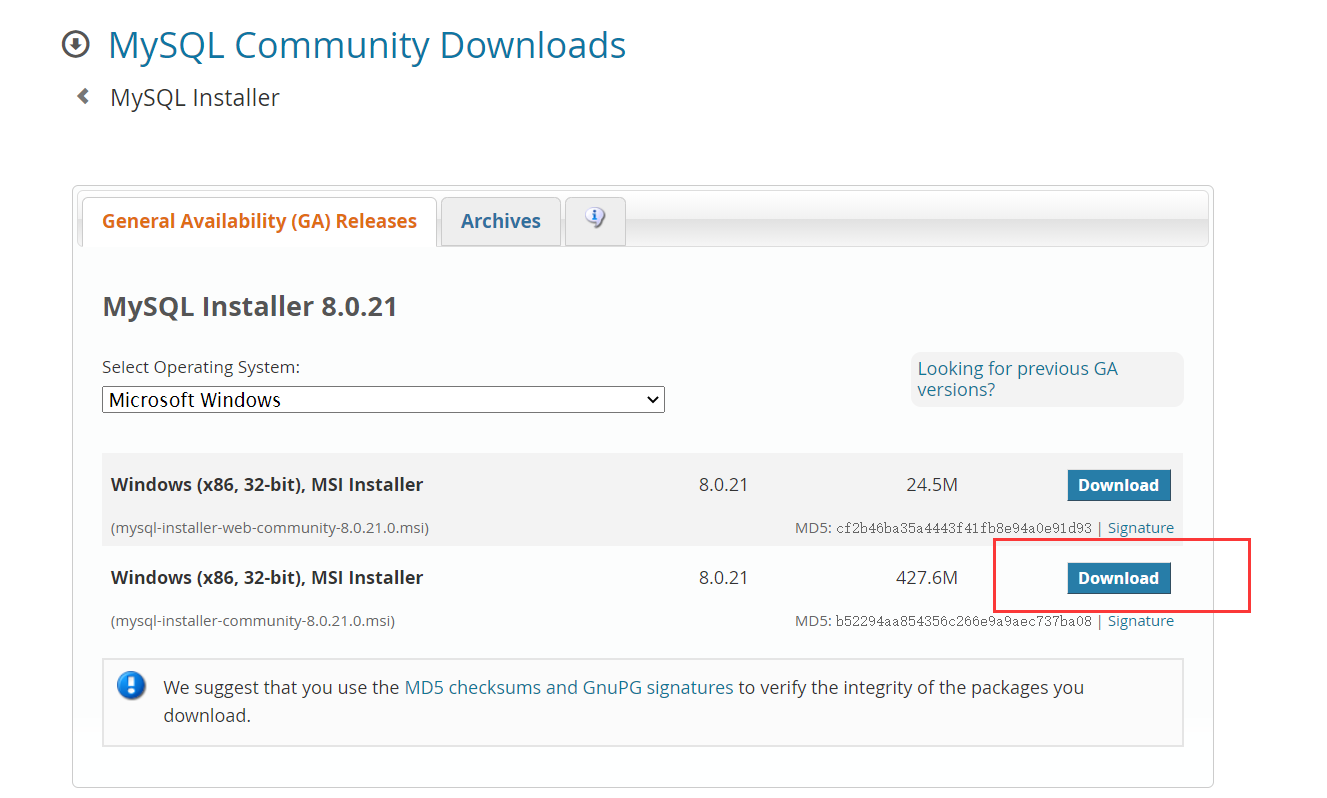
6) 2019-02-01 8.0.15GA发布

7) 最新的版本是8.0.18,2019年10月14日正式发布

8) ....后续更新

【2】官方下载地址：

<https://dev.mysql.com/downloads/windows/installer/8.0.html>



【3】安装过程：

1.双击MySQL安装文件mysql-installer-community-8.0.18.0.msi，出现安装类型选项。

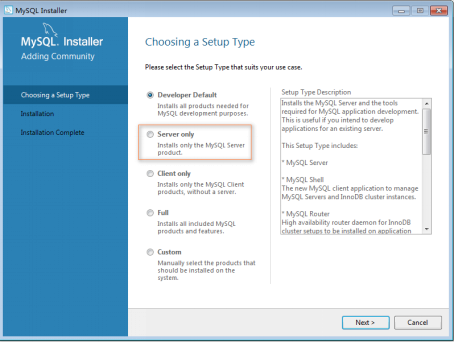
² Developer Default：开发者默认

² Server only：只安装服务器端

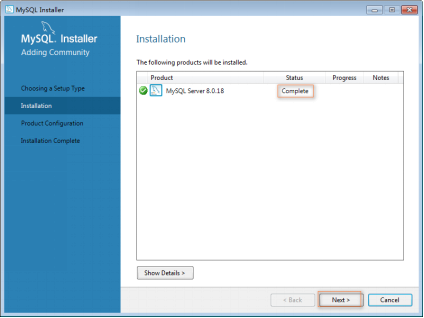
² Client only：只安装客户端

² Full：安装全部选项

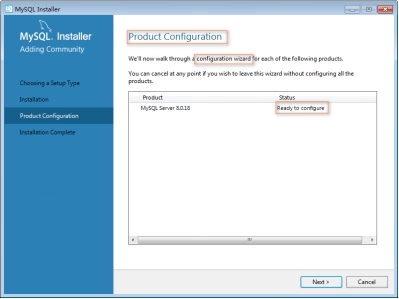
² Custom：自定义安装



2.选择，然后继续：



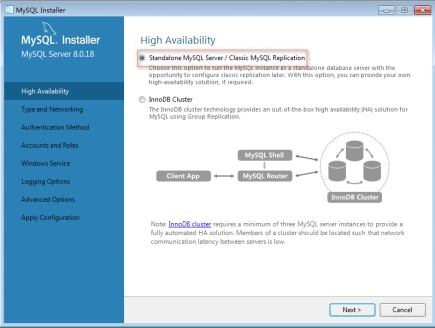
3.进入产品配置向导，配置多个安装细节，点击Next按钮即可。



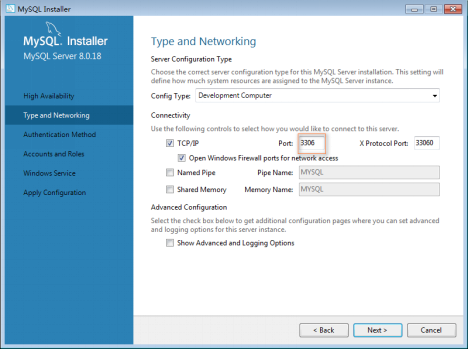
4.高可靠性High Availability，采用默认选项即可。

² Standalone MySQL Server/Classic MySQL Replication:独立MySQL服务器/经典MySQL复制

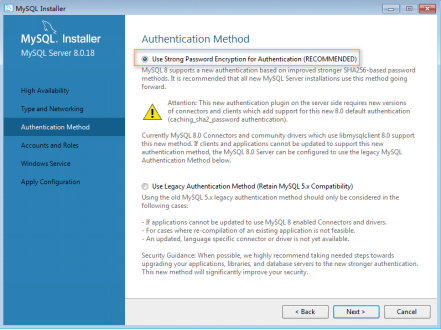
² InnoDB Cluster:InnoDB集群



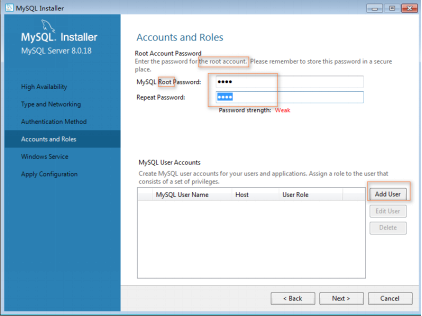
5.类型和网络 Type and Networking，采用默认选项即可。记住MySQL的监听端口默认是3306。



6.身份验证方法Authentication Method，采用默认选项即可。



7.账户和角色 Accounts and Roles。MySQL管理员账户名称是root，在此处指定root用户的密码。还可以在此处通过Add User按钮添加其他新账户，此处省略该操作。

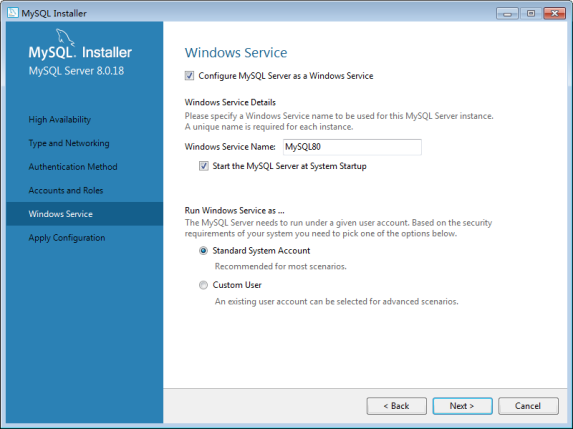


8.Windows服务：Windows Service。

² Configure MySQL Server as a Windows Service:给MySQL服务器配置一个服务项。

² Windows Service Name:服务名称，采用默认名称MySQL80即可。

² Start the MySQL at System Startup：系统启动时开启MySQL服务



9.Apply Configuration：点击Execute按钮执行开始应用这些配置项。

² Writing configuration file: 写配置文件。

² Updating Windows Firewall rules：更新Windows防火墙规则

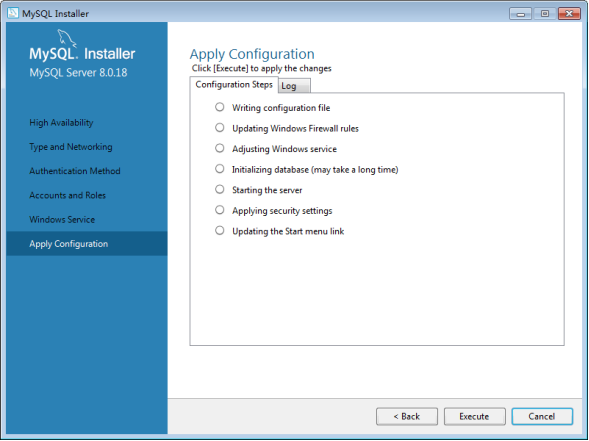
² Adjusting Windows services：调整Windows服务

² Initializing database：初始化数据库

² Starting the server： 启动服务器

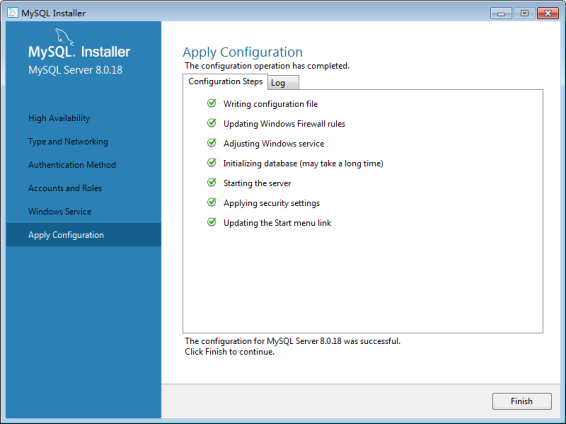
² Applying security setting：应用安全设置

² Updating the Start menu link：更新开始菜单快捷方式链接

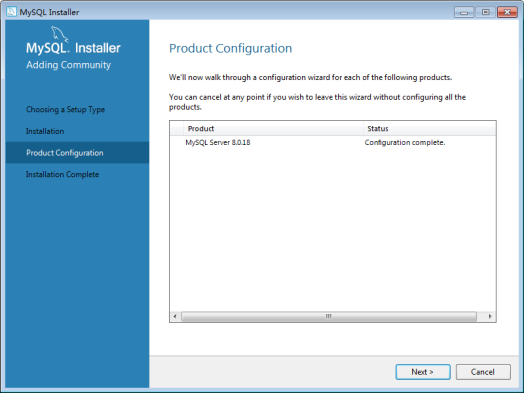


PS：如果配置出错，查看右侧的log，查看对应错误信息。

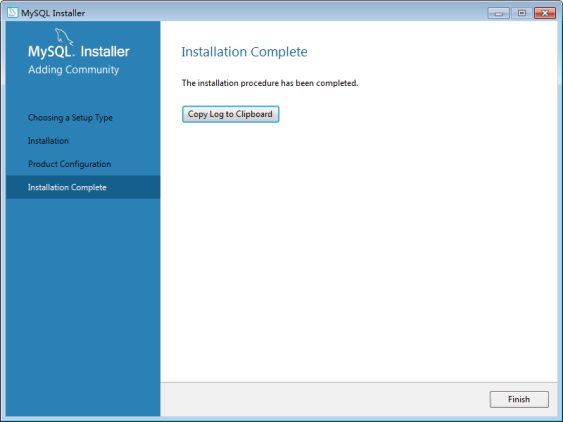
执行完成后，如下图所示。单击Finish完成安装，进入产品配置环节。



10.产品配置Product Configuration到此结束：点击Next按钮。

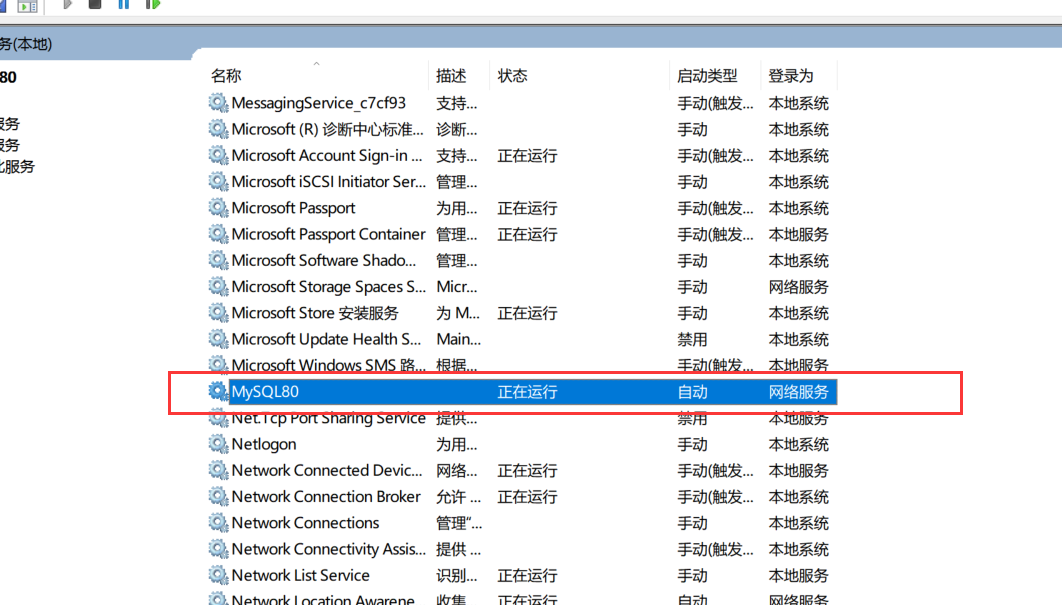


11.安装完成 Installation Complete。点击Finish按钮完成安装。

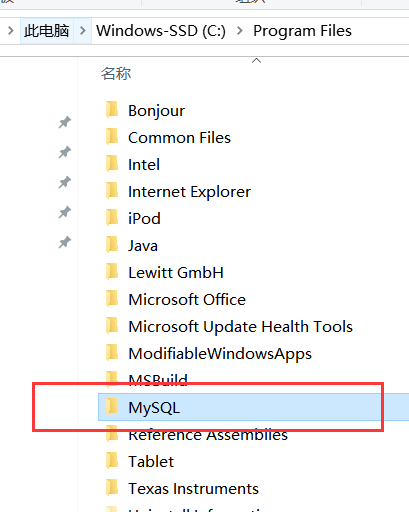


#### 查看MySQL的安装结果

1)安装了Windows Service：MySQL80，并且已经启动。

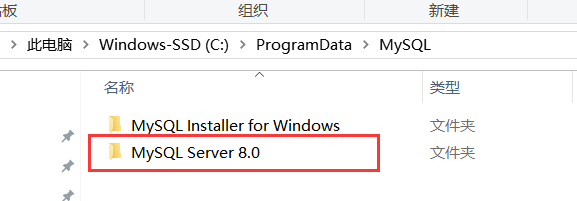


2)安装了MySQL软件。安装位置为：C:\Program Files\MySQL。



（MySQL文件下放的是软件的内容）

3)安装了MySQL数据文件夹，用来存放MySQL基础数据和以后新增的数据。安装位置为C:\ProgramData\MySQL\MySQL Server 8.0。



（ProgramData文件夹可能是隐藏的，显示出来即可）

（MySQL文件下的内容才是真正的MySQL中数据）

4)在MySQL数据文件夹中有MySQL的配置文件：my.ini。它是MySQL数据库中使用的配置文件，修改这个文件可以达到更新配置的目的。以下几个配置项需要大家特别理解。

² port=3306：监听端口是3306

² basedir="C:/Program Files/MySQL/MySQL Server 8.0/"：软件安装位置

² datadir=C:/ProgramData/MySQL/MySQL Server 8.0/Data：数据文件夹位置

² default\_authentication\_plugin=caching\_sha2\_password：默认验证插件

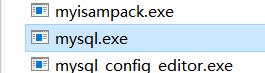
² default-storage-engine=INNODB：默认存储引擎

（这些内容在Linux下可能会手动更改）

#### MySQL登录，访问，退出操作

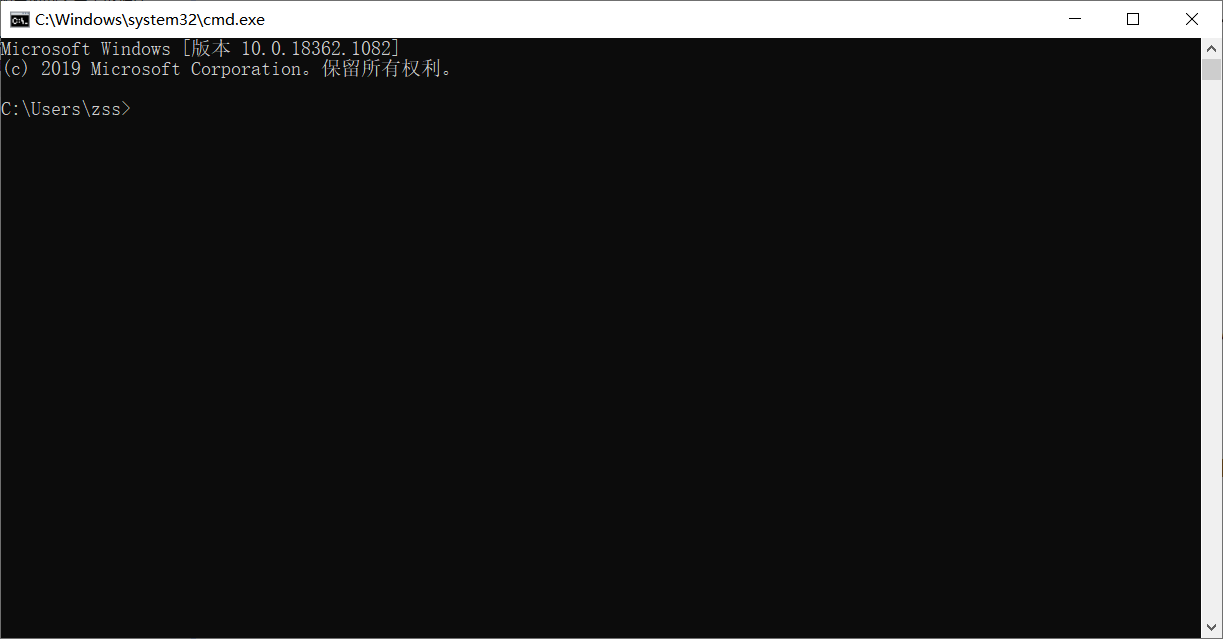
【1】登录：

访问MySQL服务器对应的命令：mysql.exe ,位置：C:\Program Files\MySQL\MySQL Server 8.0\bin

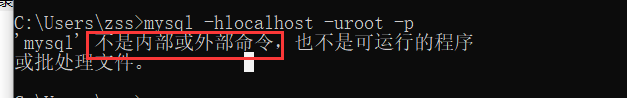


（mysql.exe需要带参数执行，所以直接在图形界面下执行该命令会自动结束）

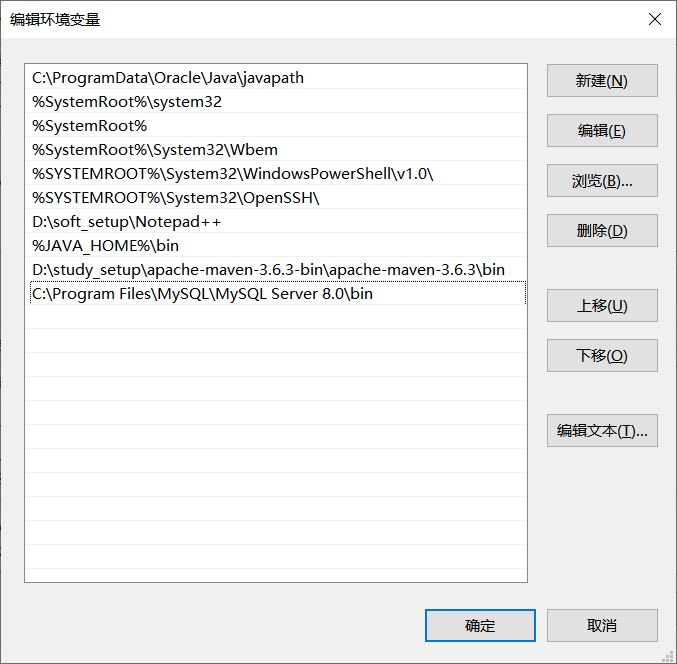
打开控制命令台：win+r:



执行mysql.exe命令的时候出现错误：



需要配置环境变量path:



注意：控制命令台必须重启才会生效：

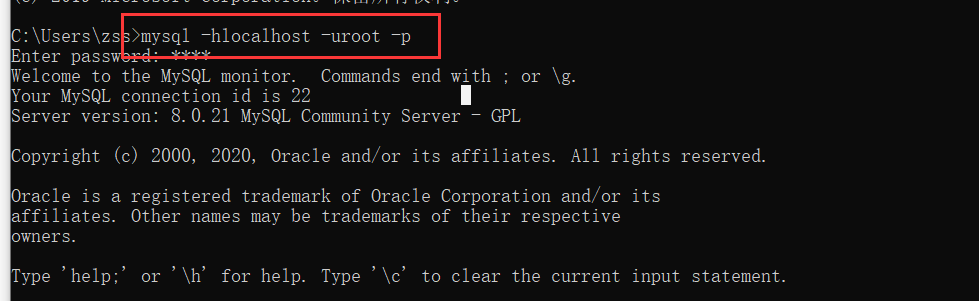
登录的命令：mysql  -hlocalhost -uroot –p

² mysql：bin目录下的文件mysql.exe。mysql是MySQL的命令行工具，是一个客户端软件，可以对任何主机的mysql服务（即后台运行的mysqld）发起连接。

² -h：host主机名。后面跟要访问的数据库服务器的地址；**如果是登录本机，可以省略**

² -u：user 用户名。后面跟登录数据的用户名，第一次安装后以root用户来登录，是MySQL的管理员用户

² -p:   password 密码。一般不直接输入，而是回车后以保密方式输入。



【2】访问数据库

显示MySQL中的数据库列表：show databases; 默认有四个自带的数据库，每个数据库中可以有多个数据库表、视图等对象。

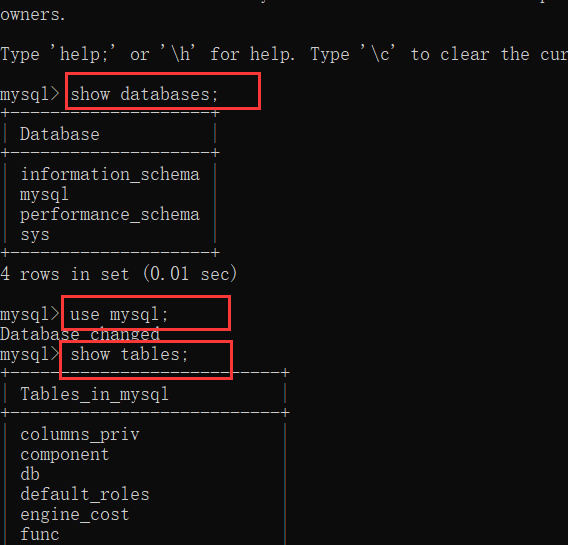
切换当前数据库的命令：use mysql;

² MySQL下可以有多个数据库，如果要访问哪个数据库，需要将其置为当前数据库。

² 该命令的作用就是将数据库mysql（默认提供的四个数据库之一的名字）置为当前数据库

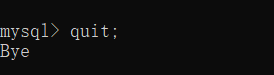
显示当前数据库的所有数据库表：show tables;

MySQL 层次：不同项目对应不同的数据库组成 - 每个数据库中有很多表  - 每个表中有很多数据



【3】**退出数据库**

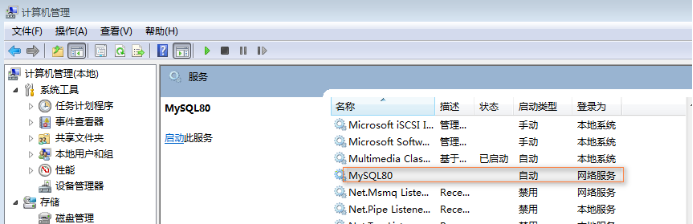
退出数据库可以使用quit或者exit命令完成，也可以用\q;  完成退出操作



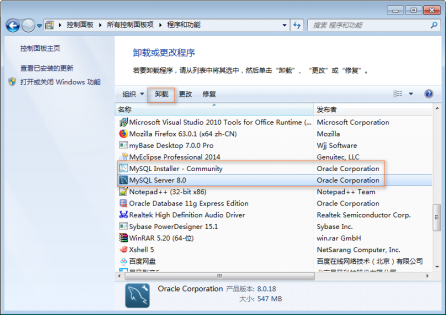
#### 数据库的卸载

【1】卸载数据库

1)停止MySQL服务：在命令行模式下执行net stop mysql或者在Windows服务窗口下停止服务



2)在控制面板中删除MySQL软件



3)删除软件文件夹：直接删除安装文件夹C:\Program Files\MySQL，其实此时该文件夹已经被删除或者剩下一个空文件夹。

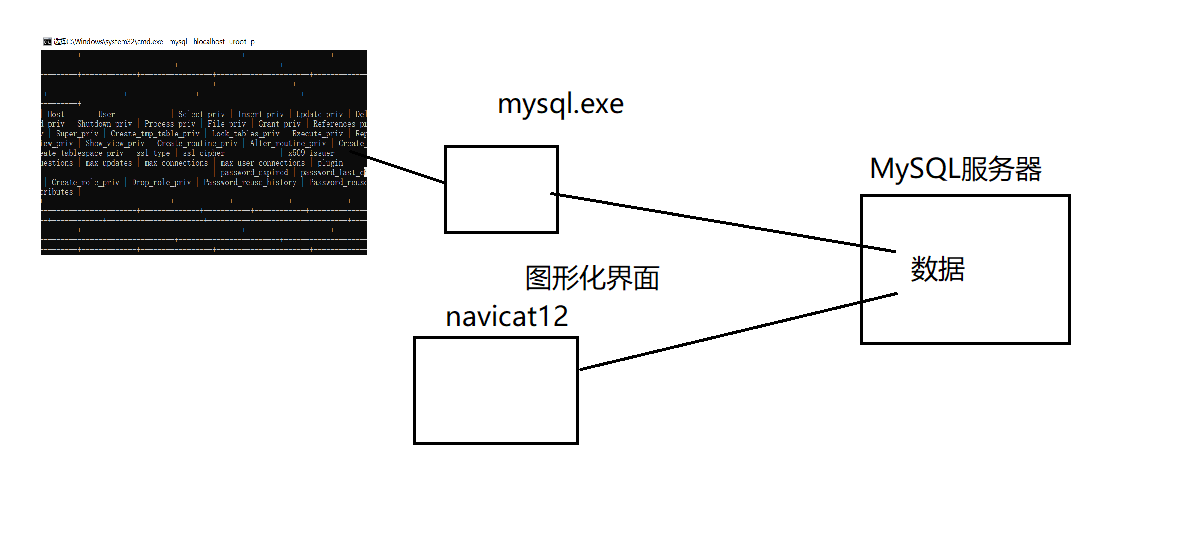
4)删除数据文件夹：直接删除文件夹C:\ProgramData\MySQL。此步不要忘记，否则会影响MySQL的再次安装。

（ProgramData文件夹可能是隐藏的，显示出来即可）

（MySQL文件下的内容才是真正的MySQL中数据）

5)删除path环境变量中关于MySQL安装路径的配置

#### 使用图形客户端navicat12连接MySQL



【1】认识Navicat

Navicat是一套快速、可靠并价格相当便宜的数据库管理工具，专为简化数据库的管理及降低系统管理成本而设。它的设计符合数据库管理员、开发人员及中小企业的需要。Navicat 是以直觉化的图形用户界面而建的，让你可以以安全并且简单的方式创建、组织、访问并共用信息。

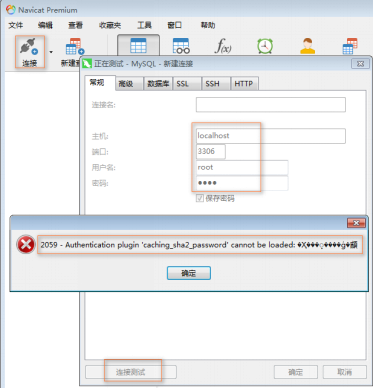


Navicat Premium 是一套数据库开发工具，让你从单一应用程序中同时连接 MySQL、MariaDB、MongoDB、SQL Server、Oracle、PostgreSQL 和 SQLite 数据库。它与 Amazon RDS、Amazon Aurora、Amazon Redshift、Microsoft Azure、Oracle Cloud、MongoDB Atlas、阿里云、腾讯云和华为云等云数据库兼容。你可以快速轻松地创建、管理和维护数据库。

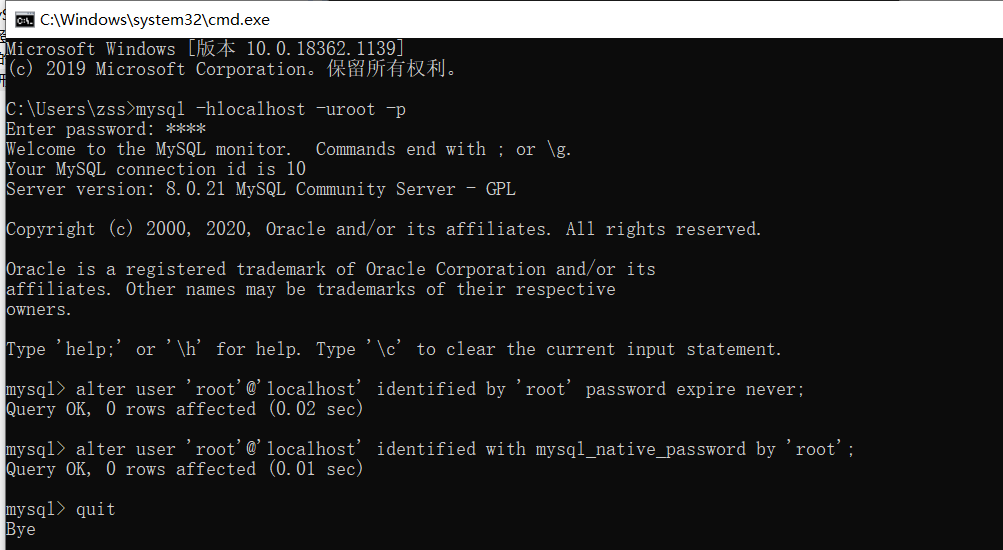


【2】安装navicat

直接解压安装包，拷贝到你定义的目录下，双击其中的navicat.exe，即可开始运行。打开后选择 连接工具按钮----连接，输入四个连接连接参数，并进行测试，结果提示连接失败，报2059异常。



该错误的原因是在MySQL8之前版本中加密规则mysql\_native\_password，而在MySQL8以后的加密规则为caching\_sha2\_password。解决此问题有两种方法，一种是更新navicat驱动来解决此问题，一种是将mysql用户登录的加密规则修改为mysql\_native\_password。此处采用第二种方式。



设置密码永不过期

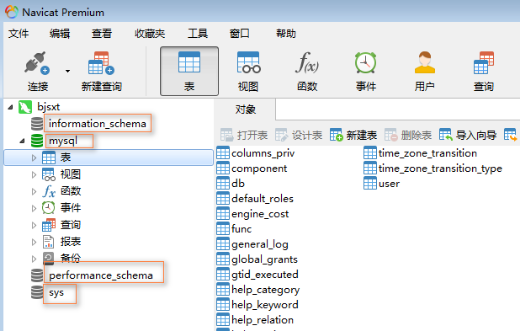
alter user ‘root’@’localhost’ identified by ‘root’ password expire never;

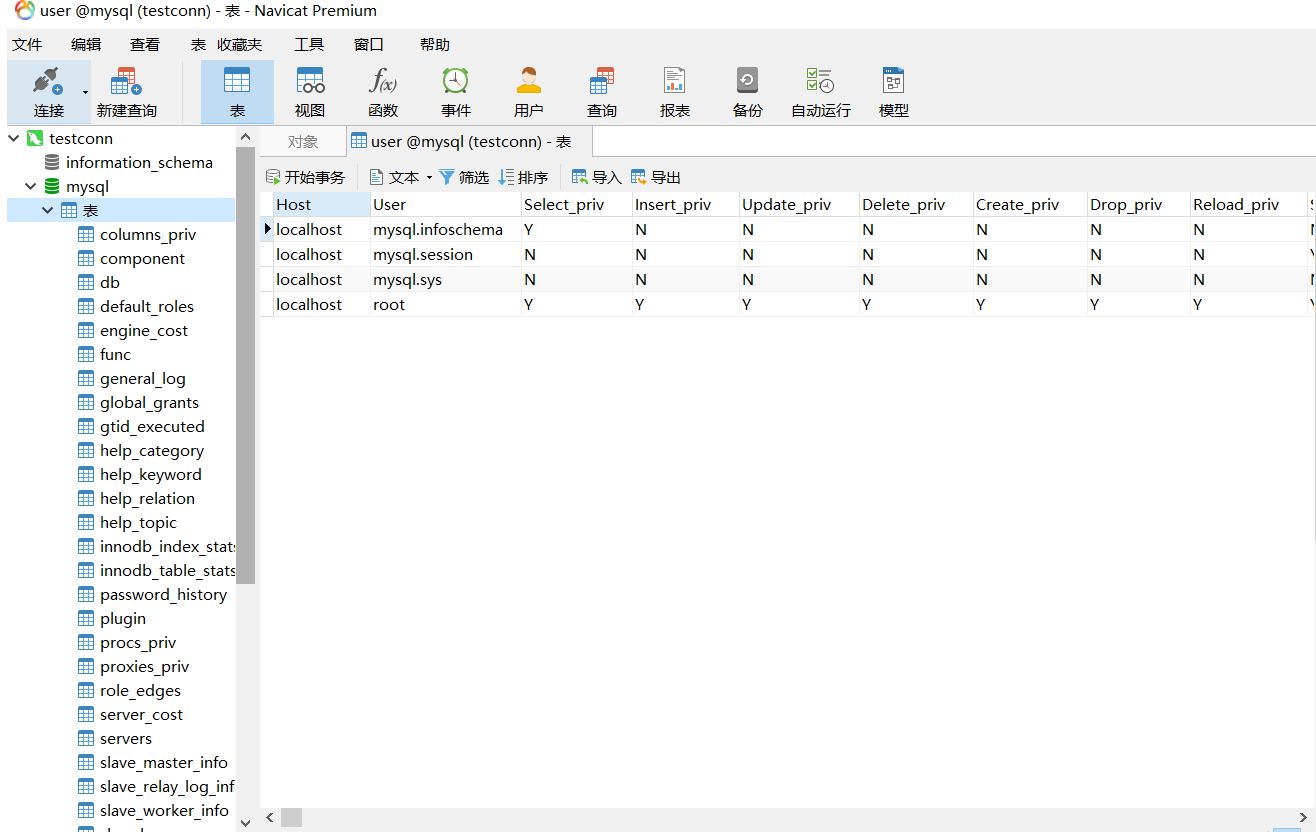
设置加密规则为mysql\_native\_password

alter user ‘root’@’localhost’ identified with mysql\_native\_password by ‘root’;

重新访问navicat，提示连接成功。

可以看到，和在cmd下执行show databases，use mysql，show tables做的任务其实是一样的，但是提供了图形化的更方便的操作页面。





#### SQL语言入门

【1】SQL语言入门

      我们都知道，数据库管理人员（DBA）通过数据库管理系统（DBMS）可以对数据库（DB）中的数据进行操作，但具体是如何操作的呢？这就涉及到我们本节要讲的SQL语言。

SQL（Structured Query Language）是结构化查询语言的简称，它是一种数据库查询和程序设计语言，同时也是目前使用最广泛的关系型数据库操作语言。在数据库管理系统中，使用SQL语言来实现数据的存取、查询、更新等功能。SQL是一种非过程化语言，只需提出“做什么”，而不需要指明“怎么做”。

      SQL是由IBM公司在1974~1979年之间根据E.J.Codd发表的关系数据库理论为基础开发的，其前身是“SEQUEL”，后更名为SQL。由于SQL语言具有集数据查询、数据操纵、数据定义和数据控制功能于一体，类似自然语言、简单易用以及非过程化等特点，得到了快速的发展，并于1986年10月，被美国国家标准协会（American National Standards Institute，ANSI）采用为关系数据库管理系统的标准语言，后为国际标准化组织（International Organization for Standardization，ISO）采纳为国际标准。



【2】SQL语言分为五个部分：

* 数据查询语言（Data Query Language，DQL）：DQL主要用于数据的查询，其基本结构是使用SELECT子句，FROM子句和WHERE子句的组合来查询一条或多条数据。
* 数据操作语言（Data Manipulation Language，DML）：DML主要用于对数据库中的数据进行增加、修改和删除的操作，其主要包括：

          1) INSERT：增加数据

          2) UPDATE：修改数据

          3) DELETE：删除数据

* 数据定义语言（Data Definition Language，DDL）：DDL主要用针对是数据库对象（数据库、表、索引、视图、触发器、存储过程、函数）进行创建、修改和删除操作。其主要包括：

          1) CREATE：创建数据库对象

          2) ALTER：修改数据库对象

          3) DROP：删除数据库对象

* 数据控制语言（Data Control Language，DCL）：DCL用来授予或回收访问 数据库的权限，其主要包括：

          1) GRANT：授予用户某种权限

          2) REVOKE：回收授予的某种权限

* 事务控制语言（Transaction Control Language，TCL）：TCL用于数据库的事务管理。其主要包括：

          1) START TRANSACTION：开启事务

          2) COMMIT：提交事务

          3) ROLLBACK：回滚事务

          4) SET TRANSACTION：设置事务的属性

#### DDL\_DML\_创建数据库表

【1】认识数据库表

表（Table）是数据库中数据存储最常见和最简单的一种形式，数据库可以将复杂的数据结构用较为简单的二维表来表示。二维表是由行和列组成的，分别都包含着数据，如表所示。



每个表都是由若干行和列组成的，在数据库中表中的行被称为记录，表中的列被称为是这些记录的字段。

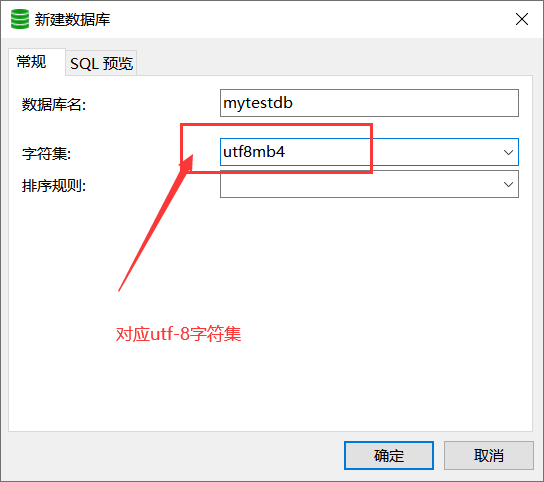
记录也被称为一行数据，是表里的一行。在关系型数据库的表里，一行数据是指一条完整的记录。

字段是表里的一列，用于保存每条记录的特定信息。如上表所示的学生信息表中的字段包括“学号”、“姓名”、“性别”和“年龄”。数据表的一列包含了某个特定字段的全部信息。

【2】创建数据库表 t\_student

* 建立一张用来存储学生信息的表
* 字段包含学号、姓名、性别，年龄、入学日期、班级，email等信息
* 学号是主键 = 不能为空 +  唯一
* 姓名不能为空
* 性别默认值是男
* Email唯一

（1）创建数据库：



（2）新建查询：



（3）创建数据库表：

1. ##这是一个单行注释
2. /\*
3. 多行注释
4. 多行注释
5. 多行注释
6. \*/
7. /\*
8. 建立一张用来存储学生信息的表
9. 字段包含学号、姓名、性别，年龄、入学日期、班级，email等信息
10. \*/
11. -- 创建数据库表：
12. create table t\_student(
13. sno int(6), -- 6显示长度
14. sname varchar(5), -- 5个字符
15. sex char(1),
16. age int(3),
17. enterdate date,
18. classname varchar(10),
19. email varchar(15)
20. );
21. -- 查看表的结构：展示表的字段详细信息
22. desc t\_student;
23. -- 查看表中数据：
24. select \* from t\_student;
25. -- 查看建表语句：
26. show create table t\_student;
27. /\*
28. CREATE TABLE `t\_student` (
29. `sno` int DEFAULT NULL,
30. `sname` varchar(5) DEFAULT NULL,
31. `sex` char(1) DEFAULT NULL,
32. `age` int DEFAULT NULL,
33. `enterdate` date DEFAULT NULL,
34. `classname` varchar(10) DEFAULT NULL,
35. `email` varchar(15) DEFAULT NULL
36. ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci
37. \*/

#### 数据库表列类型

1.整数类型



MySQL支持选择在该类型关键字后面的括号内指定整数值的显示宽度(例如，INT(4))。显示宽度并不限制可以在列内保存的值的范围，也不限制超过列的指定宽度的值的显示

主键自增：不使用序列，通过auto\_increment，要求是整数类型

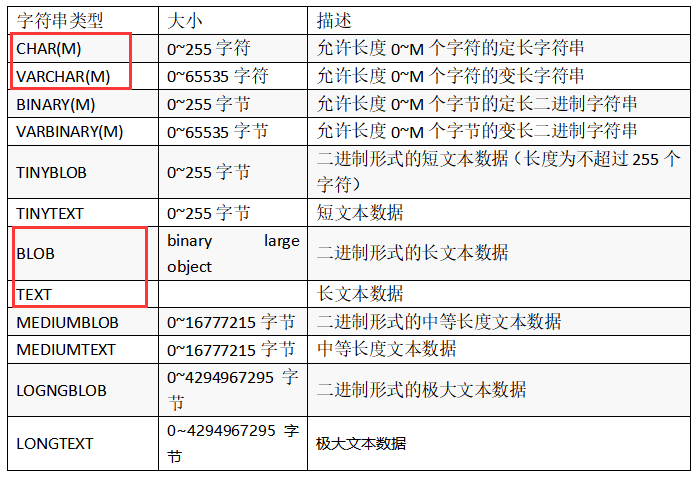
2.浮点数类型



需要注意的是与整数类型不一样的是，浮点数类型的宽度不会自动扩充。 score double(4,1)

 score double(4,1)--小数部分为1位，总宽度4位，并且不会自动扩充。

3.字符串类型

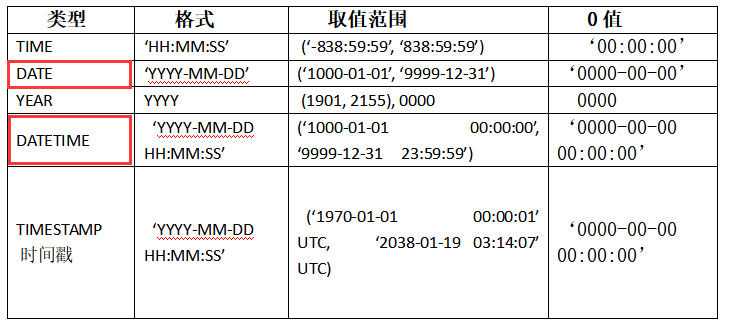


CHAR和VARCHAR类型相似，均用于存于较短的字符串，主要的不同之处在于存储方式。CHAR类型长度固定，VARCHAR类型的长度可变。

因为VARCHAR类型能够根据字符串的实际长度来动态改变所占字节的大小，所以在不能明确该字段具体需要多少字符时推荐使用VARCHAR类型，这样可以大大地节约磁盘空间、提高存储效率。

CHAR和VARCHAR表示的是字符的个数，而不是字节的个数

4.日期和时间类型



TIMESTEMP类型的数据指定方式与DATETIME基本相同，两者的不同之处在于以下几点：

(1) 数据的取值范围不同，TIMESTEMP类型的取值范围更小。

(2) 如果我们对TIMESTAMP类型的字段没有明确赋值，或是被赋与了NULL值，MySQL会自动将该字段赋值为系统当前的日期与时间。

(3) TIMESTEMP类型还可以使用CURRENT\_TIMESTAMP来获取系统当前时间。

(4) TIMESTEMP类型有一个很大的特点，那就是时间是根据时区来显示的。例如，在东八区插入的TIMESTEMP数据为2017-07-11 16:43:25，在东七区显示时，时间部分就变成了15:43:25，在东九区显示时，时间部分就变成了17:43:25。

#### DML\_添加数据

注意事项

1. int  宽度是显示宽度，如果超过，可以自动增大宽度 int底层都是4个字节
2. 时间的方式多样  &apos;1256-12-23&apos;  "1256/12/23"  "1256.12.23"
3. 字符串不区分单引号和双引号
4. 如何写入当前的时间  now() , sysdate() , CURRENT\_DATE()
5. char varchar 是字符的个数，不是字节的个数，可以使用binary，varbinary表示定长和不定长的字节个数。
6. 如果不是全字段插入数据的话，需要加入字段的名字
7. -- 查看表记录：
8. select \* from t\_student;
9. -- 在t\_student数据库表中插入数据：
10. insert into t\_student values (1,'张三','男',18,'2022-5-8','软件1班','123@126.com');
11. insert into t\_student values (10010010,'张三','男',18,'2022-5-8','软件1班','123@126.com');
12. insert into t\_student values (2,'张三','男',18,'2022.5.8','软件1班','123@126.com');
13. insert into t\_student values (2,"张三",'男',18,'2022.5.8','软件1班','123@126.com');
14. insert into t\_student values (7,"张三",'男',18,now(),'软件1班','123@126.com');
15. insert into t\_student values (9,"易烊千玺",'男',18,now(),'软件1班','123@126.com');
16. insert into t\_student (sno,sname,enterdate) values (10,'李四','2023-7-5');

#### DML\_修改，删除数据

注意事项

1.关键字，表名，字段名不区分大小写

2.默认情况下，内容不区分大小写

3.删除操作from关键字不可缺少

4.修改，删除数据别忘记加限制条件

1. -- 修改表中数据
2. update t\_student set sex = '女' ;
3. update t\_student set sex = '男' where sno = 10 ;
4. UPDATE T\_STUDENT SET AGE = 21 WHERE SNO = 10;
5. update t\_student set CLASSNAME = 'java01' where sno = 10 ;
6. update t\_student set CLASSNAME = 'JAVA01' where sno = 9 ;
7. update t\_student set age = 29 where classname = 'java01';
8. -- 删除操作：
9. delete from t\_student where sno = 2;

#### DDL\_修改，删除数据库表

1. -- 查看数据：
2. select \* from t\_student;
3. -- 修改表的结构：
4. -- 增加一列：
5. alter table t\_student add score double(5,2) ; -- 5:总位数 2：小数位数
6. update t\_student set score = 123.5678 where sno = 1 ;
7. -- 增加一列（放在最前面）
8. alter table t\_student add score double(5,2) first;
9. -- 增加一列（放在sex列的后面）
10. alter table t\_student add score double(5,2) after sex;
11. -- 删除一列：
12. alter table t\_student drop score;
13. -- 修改一列：
14. alter table t\_student modify score float(4,1); -- modify修改是列的类型的定义，但是不会改变列的名字
15. alter table t\_student change score score1 double(5,1); -- change修改列名和列的类型的定义
16. -- 删除表：
17. drop table t\_student;

#### 表的完整性约束

为防止不符合规范的数据存入数据库，在用户对数据进行插入、修改、删除等操作时，MySQL提供了一种机制来检查数据库中的数据是否满足规定的条件，以保证数据库中数据的准确性和一致性，这种机制就是完整性约束。

MySQL中主要支持以下几种种完整性约束，如表所示。 其中Check约束是MySQL8中提供的支持。

|  |  |
| --- | --- |
| **约束条件** | **约束描述** |
| PRIMARY KEY | 主键约束，约束字段的值可唯一地标识对应的记录 |
| NOT NULL | 非空约束，约束字段的值不能为空 |
| UNIQUE | 唯一约束，约束字段的值是唯一的 |
| CHECK | 检查约束，限制某个字段的取值范围 |
| DEFAULT | 默认值约束，约束字段的默认值 |
| AUTO\_INCREMENT | 自动增加约束，约束字段的值自动递增 |
| FOREIGN KEY | 外键约束，约束表与表之间的关系 |

##### 非外键约束

【1】代码演示非外键约束：

1. /\*
2. 建立一张用来存储学生信息的表
3. 字段包含学号、姓名、性别，年龄、入学日期、班级，email等信息
4. 约束：
5. 建立一张用来存储学生信息的表
6. 字段包含学号、姓名、性别，年龄、入学日期、班级，email等信息
7. 【1】学号是主键 = 不能为空 + 唯一 ，主键的作用：可以通过主键查到唯一的一条记录【2】如果主键是整数类型，那么需要自增
8. 【3】姓名不能为空
9. 【4】Email唯一
10. 【5】性别默认值是男
11. 【6】性别只能是男女
12. 【7】年龄只能在18-50之间
13. \*/
14. -- 创建数据库表：
15. create table t\_student(
16. sno int(6) primary key auto\_increment,
17. sname varchar(5) not null,
18. sex char(1) default '男' check(sex='男' || sex='女'),
19. age int(3) check(age>=18 and age<=50),
20. enterdate date,
21. classname varchar(10),
22. email varchar(15) unique
23. );
24. -- 添加数据：
25. -- 1048 - Column 'sname' cannot be null 不能为null
26. -- 3819 - Check constraint 't\_student\_chk\_1' is violated. 违反检查约束
27. insert into t\_student values (1,'张三','男',21,'2023-9-1','java01班','zs@126.com');
28. -- 1062 - Duplicate entry '1' for key 't\_student.PRIMARY' 主键重复
29. -- > 1062 - Duplicate entry 'ls@126.com' for key 't\_student.email' 违反唯一约束
30. insert into t\_student values (2,'李四','男',21,'2023-9-1','java01班','ls@126.com');
31. insert into t\_student values (3,'露露','男',21,'2023-9-1','java01班','ls@126.com');
32. -- 如果主键没有设定值，或者用null.default都可以完成主键自增的效果
33. insert into t\_student (sname,enterdate) values ('菲菲','2029-4-5');
34. insert into t\_student values (null,'小明','男',21,'2023-9-1','java01班','xm@126.com');
35. insert into t\_student values (default,'小刚','男',21,'2023-9-1','java01班','xg@126.com');
36. -- 如果sql报错，可能主键就浪费了，后续插入的主键是不连号的，我们主键也不要求连号的
37. insert into t\_student values (null,'小明','男',21,'2023-9-1','java01班','oo@126.com');
38. -- 查看数据：
39. select \* from t\_student;

【2】约束从作用上可以分为两类：

(1)   表级约束：可以约束表中任意一个或多个字段。与列定义相互独立，不包含在列定义中；与定义用‘，’分隔；必须指出要约束的列的名称；

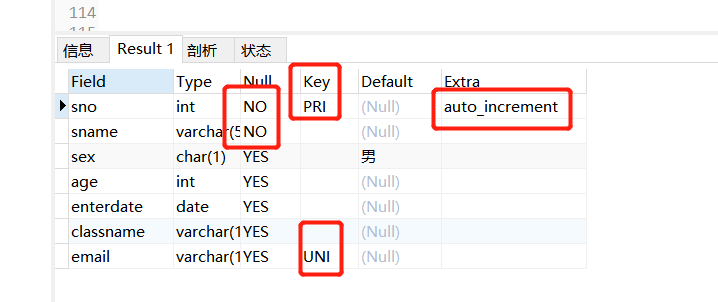
(2)   列级约束：包含在列定义中，直接跟在该列的其它定义之后 ，用空格分隔；不必指定列名；

1. -- 删除表：
2. drop table t\_student;
3. -- 创建数据库表：
4. create table t\_student(
5. sno int(6) auto\_increment,
6. sname varchar(5) not null,
7. sex char(1) default '男',
8. age int(3),
9. enterdate date,
10. classname varchar(10),
11. email varchar(15),
12. constraint pk\_stu primary key (sno), -- pk\_stu 主键约束的名字
13. constraint ck\_stu\_sex check (sex = '男' || sex = '女'),
14. constraint ck\_stu\_age check (age >= 18 and age <= 50),
15. constraint uq\_stu\_email unique (email)
16. );
17. -- 添加数据：
18. insert into t\_student values (1,'张三','男',21,'2023-9-1','java01班','zs@126.com');
19. -- > 3819 - Check constraint 'ck\_stu\_sex' is violated.
20. -- > 3819 - Check constraint 'ck\_stu\_age' is violated.
21. -- > 1062 - Duplicate entry 'zs@126.com' for key 't\_student.uq\_stu\_email'
22. insert into t\_student values (3,'李四','男',21,'2023-9-1','java01班','zs@126.com');
23. -- 查看数据：
24. select \* from t\_student;

【3】在创建表以后添加约束：

1. -- 删除表：
2. drop table t\_student;
3. -- 创建数据库表：
4. create table t\_student(
5. sno int(6),
6. sname varchar(5) not null,
7. sex char(1) default '男',
8. age int(3),
9. enterdate date,
10. classname varchar(10),
11. email varchar(15)
12. );
13. -- > 1075 - Incorrect table definition; there can be only one auto column and it must be defined as a key
14. -- 错误的解决办法：就是auto\_increment去掉
15. -- 在创建表以后添加约束：
16. alter table t\_student add constraint pk\_stu primary key (sno) ; -- 主键约束
17. alter table t\_student modify sno int(6) auto\_increment; -- 修改自增条件
18. alter table t\_student add constraint ck\_stu\_sex check (sex = '男' || sex = '女');
19. alter table t\_student add constraint ck\_stu\_age check (age >= 18 and age <= 50);
20. alter table t\_student add constraint uq\_stu\_email unique (email);
21. -- 查看表结构：
22. desc t\_student;

验证约束添加成功：查看表结构：



【4】总结：

**1.主键约束**

主键约束（PRIMARY KEY，缩写PK），是数据库中最重要的一种约束，其作用是约束表中的某个字段可以**唯一标识一条记录**。因此，使用主键约束可以快速查找表中的记录。就像人的身份证、学生的学号等等，**设置为主键的字段取值不能重复（唯一），也不能为空（非空）**，否则无法唯一标识一条记录。

主键可以是单个字段，也可以是多个字段组合。对于单字段主键的添加可使用表级约束，也可以使用列级约束；而对于多字段主键的添加只能使用表级约束。

**2.非空约束**

非空约束（NOT NULL，缩写NK）规定了一张表中指定的某个字段的值不能为空（NULL）。设置了非空约束的字段，在插入的数据为NULL时，数据库会提示错误，导致数据无法插入。

无论是单个字段还是多个字段非空约束的添加只能使用列级约束（非空约束无表级约束）

**为已存在表中的字段添加非空约束**

|  |
| --- |
| alter table student8 modify stu\_sex varchar(1) not null; |

**使用ALTER TABLE语句删除非空约束**

|  |
| --- |
| alter table student8 modify stu\_sex varchar(1) null; |

**3. 唯一约束**

唯一约束（UNIQUE，缩写UK）比较简单，它规定了一张表中指定的某个字段的值不能重复，即这一字段的每个值都是唯一的。如果想要某个字段的值不重复，那么就可以为该字段添加为唯一约束。

无论单个字段还是多个字段唯一约束的添加均可使用列级约束和表级约束

**4. 检查约束**

检查约束（CHECK）用来限制某个字段的取值范围，可以定义为列级约束，也可以定义为表级约束。MySQL8开始支持检查约束。

**5. 默认值约束**

默认值约束（DEFAULT）用来规定字段的默认值。如果某个被设置为DEFAULT约束的字段没插入具体值，那么该字段的值将会被默认值填充。

默认值约束的设置与非空约束一样，也只能使用列级约束。

**6. 字段值自动增加约束**

自增约束（AUTO\_INCREMENT）可以使表中某个字段的值自动增加。一张表中只能有一个自增长字段，并且该字段必须定义了约束（该约束可以是主键约束、唯一约束以及外键约束），如果自增字段没有定义约束，数据库则会提示“Incorrect table definition; there can be only one auto column and it must be defined as a key”错误。

由于自增约束会自动生成唯一的ID，所以自增约束通常会配合主键使用，并且只适用于整数类型。一般情况下，设置为自增约束字段的值会从1开始，每增加一条记录，该字段的值加1。

**为已存在表中的字段添加自增约束**

|  |
| --- |
| /\*创建表student11\*/  create table student11 (  stu\_id int(10) primary key,  stu\_name varchar(3),  stu\_sex varchar (1)  );  /\*为student11表中的主键字段添加自增约束\*/  alter table student11 modify stu\_id int(10) auto\_increment; |

**使用ALTER TABLE语句删除自增约束**

|  |
| --- |
| alter table studen11 modify stu\_id int(10); |

##### 外键约束

【1】什么是外键约束？

外键约束（FOREIGN KEY，缩写FK）是用来实现数据库表的参照完整性的。外键约束可以使两张表紧密的结合起来，特别是针对修改或者删除的级联操作时，会保证数据的完整性。

外键是指表中某个字段的值依赖于另一张表中某个字段的值，而被依赖的字段必须具有主键约束或者唯一约束。被依赖的表我们通常称之为父表或者主表，设置外键约束的表称为子表或者从表。

举个例子：如果想要表示学生和班级的关系，首先要有学生表和班级表两张表，然后学生表中有个字段为stu\_clazz（该字段表示学生所在的班级），而该字段的取值范围由班级表中的主键cla\_no字段（该字段表示班级编号）的取值决定。那么班级表为主表，学生表为从表，且stu\_clazz字段是学生表的外键。通过stu\_clazz字段就建立了学生表和班级表的关系。



主表（父表）：班级表  -  班级编号 - 主键

从表（子表）：学生表 - 班级编号 - 外键

【2】sql展示：

1. -- 先创建父表：班级表：
2. create table t\_class(
3. cno int(4) primary key auto\_increment,
4. cname varchar(10) not null,
5. room char(4)
6. )
7. -- 添加班级数据：
8. insert into t\_class values (null,'java001','r803');
9. insert into t\_class values (null,'java002','r416');
10. insert into t\_class values (null,'大数据001','r103');
11. -- 可以一次性添加多条记录：
12. insert into t\_class values (null,'java001','r803'),(null,'java002','r416'),(null,'大数据001','r103');
13. -- 查询班级表：
14. select \* from t\_class;
15. -- 学生表删除：
16. drop table t\_student;
17. -- 创建子表,学生表：
18. create table t\_student(
19. sno int(6) primary key auto\_increment,
20. sname varchar(5) not null,
21. classno int(4) -- 取值参考t\_class表中的cno字段，不要求字段名字完全重复，但是类型长度定义 尽量要求相同。
22. );
23. -- 添加学生信息：
24. insert into t\_student values (null,'张三',1),(null,'李四',1),(null,'王五',2);
25. -- 查看学生表：
26. select \* from t\_student;
27. -- 出现问题：
28. -- 1.添加一个学生对应的班级编码为4：
29. insert into t\_student values (null,'丽丽',4);
30. -- 2.删除班级2：
31. delete from t\_class where cno = 2;
32. -- 出现问题的原因：
33. -- 因为你现在的外键约束，没用语法添加进去，现在只是逻辑上认为班级编号是外键，没有从语法上定义
34. -- 解决办法，添加外键约束：
35. -- 注意：外键约束只有表级约束，没有列级约束：
36. create table t\_student(
37. sno int(6) primary key auto\_increment,
38. sname varchar(5) not null,
39. classno int(4),-- 取值参考t\_class表中的cno字段，不要求字段名字完全重复，但是类型长度定义 尽量要求相同。
40. constraint fk\_stu\_classno foreign key (classno) references t\_class (cno)
41. );
42. create table t\_student(
43. sno int(6) primary key auto\_increment,
44. sname varchar(5) not null,
45. classno int(4)
46. );
47. -- 在创建表以后添加外键约束：
48. alter table t\_student add constraint fk\_stu\_classno foreign key (classno) references t\_class (cno)
49. -- 上面的两个问题都解决了：
50. -- 添加学生信息：
51. -- > 1452 - Cannot add or update a child row: a foreign key constraint fails (`mytestdb`.`t\_student`, CONSTRAINT `fk\_stu\_classno` FOREIGN KEY (`classno`) REFERENCES `t\_class` (`cno`))
52. insert into t\_student values (null,'张三',1),(null,'李四',1),(null,'王五',2);
53. -- 删除班级1：
54. -- 2.删除班级2：
55. insert into t\_student values (null,'张三',3),(null,'李四',3),(null,'王五',3);
56. -- > 1451 - Cannot delete or update a parent row: a foreign key constraint fails (`mytestdb`.`t\_student`, CONSTRAINT `fk\_stu\_classno` FOREIGN KEY (`classno`) REFERENCES `t\_class` (`cno`))
57. delete from t\_class where cno = 3;

###### 外键策略

1. -- 学生表删除：
2. drop table t\_student;
3. -- 班级表删除：
4. drop table t\_class;
5. -- 注意：先删除从表，再删除主表。（视频中这个位置笔误，笔记现在已经更正）
6. -- 先创建父表：班级表：
7. create table t\_class(
8. cno int(4) primary key auto\_increment,
9. cname varchar(10) not null,
10. room char(4)
11. )
12. -- 可以一次性添加多条记录：
13. insert into t\_class values (null,'java001','r803'),(null,'java002','r416'),(null,'大数据001','r103');
14. -- 添加学生表，添加外键约束：
15. create table t\_student(
16. sno int(6) primary key auto\_increment,
17. sname varchar(5) not null,
18. classno int(4),-- 取值参考t\_class表中的cno字段，不要求字段名字完全重复，但是类型长度定义 尽量要求相同。
19. constraint fk\_stu\_classno foreign key (classno) references t\_class (cno)
20. );
21. -- 可以一次性添加多条记录：
22. insert into t\_student values (null,'张三',1),(null,'李四',1),(null,'王五',2),(null,'朱六',3);
23. -- 查看班级表和学生表：
24. select \* from t\_class;
25. select \* from t\_student;
26. -- 删除班级2：如果直接删除的话肯定不行因为有外键约束：
27. -- 加入外键策略：
28. -- 策略1：no action 不允许操作
29. -- 通过操作sql来完成：
30. -- 先把班级2的学生对应的班级 改为null
31. update t\_student set classno = null where classno = 2;
32. -- 然后再删除班级2：
33. delete from t\_class where cno = 2;
34. -- 策略2：cascade 级联操作：操作主表的时候影响从表的外键信息：
35. -- 先删除之前的外键约束：
36. alter table t\_student drop foreign key fk\_stu\_classno;
37. -- 重新添加外键约束：
38. alter table t\_student add constraint fk\_stu\_classno foreign key (classno) references t\_class (cno) on update cascade on delete cascade;
39. -- 试试更新：
40. update t\_class set cno = 5 where cno = 3;
41. -- 试试删除：
42. delete from t\_class where cno = 5;
43. -- 策略3：set null 置空操作：
44. -- 先删除之前的外键约束：
45. alter table t\_student drop foreign key fk\_stu\_classno;
46. -- 重新添加外键约束：
47. alter table t\_student add constraint fk\_stu\_classno foreign key (classno) references t\_class (cno) on update set null on delete set null;
48. -- 试试更新：
49. update t\_class set cno = 8 where cno = 1;
50. -- 注意：
51. -- 1. 策略2 级联操作 和 策略2 的 删除操作 可以混着使用：
52. alter table t\_student add constraint fk\_stu\_classno foreign key (classno) references t\_class (cno) on update cascade on delete set null ;
53. -- 2.应用场合：
54. -- （1）朋友圈删除，点赞。留言都删除 -- 级联操作
55. -- （2）解散班级，对应的学生 置为班级为null就可以了，-- set null

#### DDL和DML的补充

【1】sql展示：

1. -- 创建表：
2. create table t\_student(
3. sno int(6) primary key auto\_increment,
4. sname varchar(5) not null,
5. sex char(1) default '男' check(sex='男' || sex='女'),
6. age int(3) check(age>=18 and age<=50),
7. enterdate date,
8. classname varchar(10),
9. email varchar(15) unique
10. );
11. -- 添加数据：
12. insert into t\_student values (null,'张三','男',21,'2023-9-1','java01班','zs@126.com');
13. insert into t\_student values (null,'李四','男',21,'2023-9-1','java01班','ls@126.com');
14. insert into t\_student values (null,'露露','男',21,'2023-9-1','java01班','ll@126.com');
15. -- 查看学生表：
16. select \* from t\_student;
17. -- 添加一张表：快速添加：结构和数据跟t\_student 都是一致的
18. create table t\_student2
19. as
20. select \* from t\_student;
21. select \* from t\_student2;
22. -- 快速添加，结构跟t\_student一致，数据没有：
23. create table t\_student3
24. as
25. select \* from t\_student where 1=2;
26. select \* from t\_student3;
27. -- 快速添加：只要部分列，部分数据：
28. create table t\_student4
29. as
30. select sno,sname,age from t\_student where sno = 2;
31. select \* from t\_student4;
32. -- 删除数据操作 :清空数据
33. delete from t\_student;
34. truncate table t\_student;

【2】delete和truncate的区别：

从最终的结果来看，虽然使用TRUNCATE操作和使用DELETE操作都可以删除表中的全部记录，但是两者还是有很多区别的，其区别主要体现在以下几个方面：

(1)DELETE为数据操作语言DML；TRUNCATE为数据定义语言DDL。

(2) DELETE操作是将表中所有记录一条一条删除直到删除完；TRUNCATE操作则是保留了表的结构，重新创建了这个表，所有的状态都相当于新表。因此，TRUNCATE操作的效率更高。

(3)DELETE操作可以回滚；TRUNCATE操作会导致隐式提交，因此不能回滚（在第十章中会讲解事务的提交和回滚）。

(4)DELETE操作执行成功后会返回已删除的行数（如删除4行记录，则会显示“Affected rows：4”）；截断操作不会返回已删除的行量，结果通常是“Affected rows：0”。DELETE操作删除表中记录后，再次向表中添加新记录时，对于设置有自增约束字段的值会从删除前表中该字段的最大值加1开始自增；TRUNCATE操作则会重新从1开始自增。

#### DQL-查询操作

##### 表的准备

准备四张表：dept(部门表),emp(员工表),salgrade(薪资等级表),bonus(奖金表)

1. create table DEPT(
2. DEPTNO int(2) not null,
3. DNAME VARCHAR(14),
4. LOC VARCHAR(13)
5. );
6. alter table DEPT
7. add constraint PK\_DEPT primary key (DEPTNO);
9. create table EMP
10. (
11. EMPNO int(4) primary key,
12. ENAME VARCHAR(10),
13. JOB VARCHAR(9),
14. MGR int(4),
15. HIREDATE DATE,
16. SAL double(7,2),
17. COMM double(7,2),
18. DEPTNO int(2)
19. );
20. alter table EMP
21. add constraint FK\_DEPTNO foreign key (DEPTNO)
22. references DEPT (DEPTNO);
24. create table SALGRADE
25. (
26. GRADE int primary key,
27. LOSAL double(7,2),
28. HISAL double(7,2)
29. );
30. create table BONUS
31. (
32. ENAME VARCHAR(10),
33. JOB VARCHAR(9),
34. SAL double(7,2),
35. COMM double(7,2)
36. );
37. insert into DEPT (DEPTNO, DNAME, LOC)
38. values (10, 'ACCOUNTING', 'NEW YORK');
39. insert into DEPT (DEPTNO, DNAME, LOC)
40. values (20, 'RESEARCH', 'DALLAS');
41. insert into DEPT (DEPTNO, DNAME, LOC)
42. values (30, 'SALES', 'CHICAGO');
43. insert into DEPT (DEPTNO, DNAME, LOC)
44. values (40, 'OPERATIONS', 'BOSTON');
45. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
46. values (7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, null, 20);
47. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
48. values (7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
49. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
50. values (7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
51. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
52. values (7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, null, 20);
53. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
54. values (7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
55. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
56. values (7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, null, 30);
57. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
58. values (7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, null, 10);
59. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
60. values (7788, 'SCOTT', 'ANALYST', 7566, '1987-04-19', 3000, null, 20);
61. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
62. values (7839, 'KING', 'PRESIDENT', null, '1981-11-17', 5000, null, 10);
63. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
64. values (7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08', 1500, 0, 30);
65. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
66. values (7876, 'ADAMS', 'CLERK', 7788, '1987-05-23', 1100, null, 20);
67. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
68. values (7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, null, 30);
69. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
70. values (7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, null, 20);
71. insert into EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
72. values (7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, null, 10);
73. insert into SALGRADE (GRADE, LOSAL, HISAL)
74. values (1, 700, 1200);
75. insert into SALGRADE (GRADE, LOSAL, HISAL)
76. values (2, 1201, 1400);
77. insert into SALGRADE (GRADE, LOSAL, HISAL)
78. values (3, 1401, 2000);
79. insert into SALGRADE (GRADE, LOSAL, HISAL)
80. values (4, 2001, 3000);
81. insert into SALGRADE (GRADE, LOSAL, HISAL)
82. values (5, 3001, 9999);
83. -- 查看表：
84. select \* from dept;
85. -- 部门表：dept:department 部分 ，loc - location 位置
86. select \* from emp;
87. -- 员工表：emp:employee 员工 ,mgr :manager上级领导编号，hiredate 入职日期 firedate 解雇日期 ，common：补助
88. -- deptno 外键 参考 dept - deptno字段
89. -- mgr 外键 参考 自身表emp - empno 产生了自关联
90. select \* from salgrade;
91. -- losal - lowsal
92. -- hisal - highsal
93. select \* from bonus;

##### 单表查询

###### 最简单的SQL查询

1. -- 对emp表查询：
2. select \* from emp; -- \*代表所有数据
3. -- 显示部分列：
4. select empno,ename,sal from emp;
5. -- 显示部分行：where子句
6. select \* from emp where sal > 2000;
7. -- 显示部分列，部分行：
8. select empno,ename,job,mgr from emp where sal > 2000;
9. -- 起别名：
10. select empno 员工编号,ename 姓名,sal 工资 from emp; -- as 省略，''或者""省略了
11. -- as alias 别名
12. select empno as 员工编号,ename as 姓名,sal as 工资 from emp;
13. select empno as '员工编号',ename as "姓名",sal as 工资 from emp;
14. -- > 1064 - 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 '编号,ename as "姓 名",sal as 工资 from emp' at line 1
15. -- 错误原因：在别名中有特殊符号的时候，''或者""不可以省略不写
16. select empno as 员工 编号,ename as "姓 名",sal as 工资 from emp;
17. -- 算术运算符：
18. select empno,ename,sal,sal+1000 as '涨薪后',deptno from emp where sal < 2500;
19. select empno,ename,sal,comm,sal+comm from emp; -- ？？？后面再说
20. -- 去重操作：
21. select job from emp;
22. select distinct job from emp;
23. select job,deptno from emp;
24. select distinct job,deptno from emp; -- 对后面的所有列组合 去重 ，而不是单独的某一列去重
25. -- 排序：
26. select \* from emp order by sal; -- 默认情况下是按照升序排列的
27. select \* from emp order by sal asc; -- asc 升序，可以默认不写
28. select \* from emp order by sal desc; -- desc 降序
29. select \* from emp order by sal asc ,deptno desc; -- 在工资升序的情况下，deptno按照降序排列

###### where子句

指定查询条件使用where子句，可以查询符合条件的部分记录。

1. -- 查看emp表：
2. select \* from emp;
3. -- where子句：将过滤条件放在where子句的后面，可以筛选/过滤出我们想要的符合条件的数据：
4. -- where 子句 + 关系运算符
5. select \* from emp where deptno = 10;
6. select \* from emp where deptno > 10;
7. select \* from emp where deptno >= 10;
8. select \* from emp where deptno < 10;
9. select \* from emp where deptno <= 10;
10. select \* from emp where deptno <> 10;
11. select \* from emp where deptno != 10;
12. select \* from emp where job = 'CLERK';
13. select \* from emp where job = 'clerk'; -- 默认情况下不区分大小写
14. select \* from emp where binary job = 'clerk'; -- binary区分大小写
15. select \* from emp where hiredate < '1981-12-25';
16. -- where 子句 + 逻辑运算符：and
17. select \* from emp where sal > 1500 and sal < 3000; -- (1500,3000)
18. select \* from emp where sal > 1500 && sal < 3000;
19. select \* from emp where sal > 1500 and sal < 3000 order by sal;
20. select \* from emp where sal between 1500 and 3000; -- [1500,3000]
21. -- where 子句 + 逻辑运算符：or
22. select \* from emp where deptno = 10 or deptno = 20;
23. select \* from emp where deptno = 10 || deptno = 20;
24. select \* from emp where deptno in (10,20);
25. select \* from emp where job in ('MANAGER','CLERK','ANALYST');
26. -- where子句 + 模糊查询：
27. -- 查询名字中带A的员工 -- %代表任意多个字符 0,1,2，.....
28. select \* from emp where ename like '%A%' ;
29. -- -任意一个字符
30. select \* from emp where ename like '\_\_A%' ;
31. -- 关于null的判断：
32. select \* from emp where comm is null;
33. select \* from emp where comm is not null;
34. -- 小括号的使用 ：因为不同的运算符的优先级别不同，加括号为了可读性
35. select \* from emp where job = 'SALESMAN' or job = 'CLERK' and sal >=1500; -- 先and再or and > or
36. select \* from emp where job = 'SALESMAN' or (job = 'CLERK' and sal >=1500);
37. select \* from emp where (job = 'SALESMAN' or job = 'CLERK') and sal >=1500;

###### 使用函数

MySQL中提供了大量函数来简化用户对数据库的操作，比如字符串的处理、日期的运算、数值的运算等等。使用函数可以大大提高SELECT语句操作数据库的能力，同时也给数据的转换和处理提供了方便。 （在sql中使用函数）

函数只是对查询结果中的数据进行处理，不会改变数据库中数据表的值。MySQL中的函数主要分为单行函数和多行函数两大类，下面我们将详细讲解这两大类函数。

**单行函数**

单行函数是指对每一条记录输入值进行计算，并得到相应的计算结果，然后返回给用户，也就是说，每条记录作为一个输入参数，经过函数计算得到每条记录的计算结果。

常用的单行函数主要包括字符串函数、数值函数、日期与时间函数、流程函数以及其他函数。

**多行函数**

多行函数是指对一组数据进行运算，针对这一组数据（多行记录）只返回一个结果，也称为分组函数。

1. -- 函数举例：
2. select empno,ename,lower(ename),upper(ename),sal from emp;
3. -- 函数的功能：封装了特定的一些功能，我们直接拿过来使用，可以实现对应的功能
4. -- 函数作用：为了提高select的能力
5. -- 注意：函数没有改变数据自身的值，而是在真实数据的上面进行加工处理，展示新的结果而已。
6. select max(sal),min(sal),count(sal),sum(sal),avg(sal) from emp;
7. -- 函数的分类：
8. -- lower(ename),upper(ename) ：改变每一条结果，每一条数据对应一条结果 -- 单行函数
9. -- max(sal),min(sal),count(sal),sum(sal),avg(sal):多条数据，最终展示一个结果 -- 多行函数

PS：除了多行函数（max,min,count,sum,avg），都是单行函数

单行函数

**1.字符串函数  （String StringBuilder）**

|  |  |
| --- | --- |
| **函数** | **描述** |
| CONCAT(str1, str2, ···, strn) | 将str1、str2···strn拼接成一个新的字符串 |
| INSERT(str, index, n, newstr) | 将字符串str从第index位置开始的n个字符替换成字符串newstr |
| LENGTH(str) | 获取字符串str的长度 |
| LOWER(str) | 将字符串str中的每个字符转换为小写 |
| UPPER(str) | 将字符串str中的每个字符转换为大写 |
| LEFT(str, n) | 获取字符串str最左边的n个字符 |
| RIGHT(str, n) | 获取字符串str最右边的n个字符 |
| LPAD(str, n, pad) | 使用字符串pad在str的最左边进行填充，直到长度为n个字符为止 |
| RPAD(str, n, pad) | 使用字符串pad在str的最右边进行填充，直到长度为n个字符为止 |
| LTRIM(str) | 去除字符串str左侧的空格 |
| RTRIM(str) | 去除字符串str右侧的空格 |
| TRIM(str) | 去除字符串str左右两侧的空格 |
| REPLACE(str,oldstr,newstr) | 用字符串newstr替换字符串str中所有的子字符串oldstr |
| REVERSE(str) | 将字符串str中的字符逆序 |
| STRCMP(str1, str2) | 比较字符串str1和str2的大小 |
| SUBSTRING(str,index,n) | 获取从字符串str的index位置开始的n个字符 |

**2. 数值函数 （Math）**

|  |  |
| --- | --- |
| **函数** | **描述** |
| ABS(num) | 返回num的绝对值 |
| CEIL(num) | 返回大于num的最小整数（向上取整） |
| FLOOR(num) | 返回小于num的最大整数（向下取整） |
| MOD(num1, num2) | 返回num1/num2的余数（取模） |
| PI() | 返回圆周率的值 |
| POW(num,n)/POWER(num, n) | 返回num的n次方 |
| RAND(num) | 返回0~1之间的随机数 |
| ROUND(num, n) | 返回x四舍五入后的值，该值保留到小数点后n位 |
| TRUNCATE(num, n) | 返回num被舍去至小数点后n位的值 |

**3. 日期与时间函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| CURDATE() | 返回当前日期 |
| CURTIME() | 返回当前时间 |
| NOW() | 返回当前日期和时间 |
| SYSDATE() | 返回该函数执行时的日期和时间 |
| DAYOFYEAR(date) | 返回日期date为一年中的第几天 |
| WEEK(date)/WEEKOFYEAR(date) | 返回日期date为一年中的第几周 |
| DATE\_FORMAT(date, format) | 返回按字符串format格式化后的日期date |
| DATE\_ADD(date, INTERVAL expr unit)  /ADDDATE(date, INTERVAL expr unit) | 返回date加上一个时间间隔后的新时间值 |
| DATE\_SUB(date, INTERVAL expr unit)  /SUBDATE(date, INTERVAL expr unit) | 返回date减去一个时间间隔后的新时间值 |
| DATEDIFF(date1, date2) | 返回起始日期date1与结束日期date2之间的间隔天数 |

1. -- 单行函数包含：
2. -- 1.字符串函数
3. select ename,length(ename),substring(ename,2,3) from emp;
4. -- substring字符串截取，2:从字符下标为2开始，3：截取长度3 （下标从1开始）
5. -- 2.数值函数
6. select abs(-5),ceil(5.3),floor(5.9),round(3.14) from dual; -- dual实际就是一个伪表
7. select abs(-5) 绝对值,ceil(5.3) 向上取整,floor(5.9) 向下取整,round(3.14) 四舍五入; -- 如果没有where条件的话，from dual可以省略不写
8. select ceil(sal) from emp;
9. select 10/3,10%3,mod(10,3) ;
10. -- 3.日期与时间函数
11. select \* from emp;
12. select curdate(),curtime() ; -- curdate()年月日 curtime()时分秒
13. select now(),sysdate(),sleep(3),now(),sysdate() from dual; -- now(),sysdate() 年月日时分秒
14. insert into emp values (9999,'lili','SALASMAN',7698,now(),1000,null,30);
15. -- now()可以表示年月日时分秒，但是插入数据的时候还是要参照表的结构的
16. desc emp;

**4. 流程函数（ IF  SWITCH）**

|  |  |
| --- | --- |
| **间隔类型** | **描述** |
| IF(condition, t, f) | 如果条件condition为真，则返回t，否则返回f |
| IFNULL(value1, value2) | 如果value1不为null，则返回value1，否则返回value2 |
| NULLIF(value1, value2) | 如果value1等于value2，则返回null，否则返回value1 |
| CASE value WHEN [value1] THEN result1 [WHEN [value2] THEN result2 ...] [ELSE result] END | 如果value等于value1，则返回result1，···，否则返回result |
| CASE WHEN [condition1] THEN result1 [WHEN [condition2] THEN result2 ...] [ELSE result] END | 如果条件condition1为真，则返回result1，···，否则返回result |

**5.JSON函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| JSON\_APPEND() | 在JSON文档中追加数据 |
| JSON\_INSERT () | 在JSON文档中插入数据 |
| JSON\_REPLACE () | 替换JSON文档中的数据 |
| JSON\_REMOVE () | 从JSON文档的指定位置移除数据 |
| JSON\_CONTAINS() | 判断JSON文档中是否包含某个数据 |
| JSON\_SEARCH() | 查找JSON文档中给定字符串的路径 |

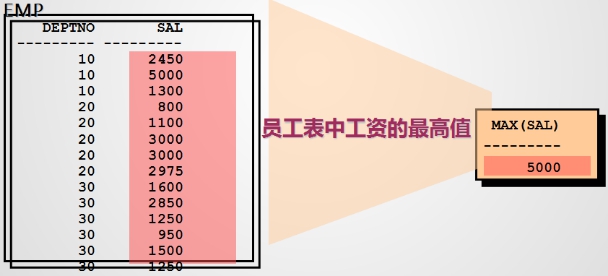
**6.其他函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| DATABASE() | 返回当前数据库名 |
| VERSION() | 返回当前MySQL的版本号 |
| USER() | 返回当前登录的用户名 |
| INET\_ATON(IP) | 返回IP地址的数字表示 |
| INET\_NTOA | 返回数字代表的IP地址 |
| PASSWORD(str) | 实现对字符串str的加密操作 |
| FORMAT(num, n) | 实现对数字num的格式化操作，保留n位小数 |
| CONVERT(data, type) | 实现将数据data转换成type类型的操作 |

1. -- 4.流程函数
2. -- if相关
3. select empno,ename,sal,if(sal>=2500,'高薪','底薪') as '薪资等级' from emp; -- if-else 双分支结构
4. select empno,ename,sal,comm,sal+ifnull(comm,0) from emp; -- 如果comm是null，那么取值为0 -- 单分支
5. select nullif(1,1),nullif(1,2) from dual; -- 如果value1等于value2，则返回null，否则返回value1
6. -- case相关：
7. -- case等值判断
8. select empno,ename,job,
9. case job
10. when 'CLERK' then '店员'
11. when 'SALESMAN' then '销售'
12. when 'MANAGER' then '经理'
13. else '其他'
14. end '岗位',
15. sal from emp;
16. -- case区间判断:
17. select empno,ename,sal,
18. case
19. when sal<=1000 then 'A'
20. when sal<=2000 then 'B'
21. when sal<=3000 then 'C'
22. else 'D'
23. end '工资等级',
24. deptno from emp;
25. from emp;
26. -- 5.JSON函数
27. -- 6.其他函数
28. select database(),user(),version() from dual;

多行函数

对一组数据进行运算，针对一组数据（多行记录）只返回一个结果，也称分组函数



多行函数包含：

|  |  |
| --- | --- |
| **函数** | **描述** |
| COUNT() | 统计表中记录的数目 |
| SUM() | 计算指定字段值的总和 |
| AVG() | 计算指定字段值的平均值 |
| MAX() | 统计指定字段值的最大值 |
| MIN() | 统计指定字段值的最小值 |

1. -- 多行函数：
2. select max(sal),min(sal),count(sal),sum(sal),sum(sal)/count(sal),avg(sal) from emp;
3. select \* from emp;
4. -- 多行函数自动忽略null值
5. select max(comm),min(comm),count(comm),sum(comm),sum(comm)/count(comm),avg(comm) from emp;
6. -- max(),min(),count()针对所有类型 sum(),avg() 只针对数值型类型有效
7. select max(ename),min(ename),count(ename),sum(ename),avg(ename) from emp;
8. -- count --计数
9. -- 统计表的记录数：方式1：
10. select \* from emp;
11. select count(ename) from emp;
12. select count(\*) from emp;
13. -- 统计表的记录数：方式2
14. select 1 from dual;
15. select 1 from emp;
16. select count(1) from emp;

###### group\_by分组

【1】group by : 用来进行分组

【2】sql展示：

1. select \* from emp;
2. -- 统计各个部门的平均工资
3. select deptno,avg(sal) from emp; -- 字段和多行函数不可以同时使用
4. select deptno,avg(sal) from emp group by deptno; -- 字段和多行函数不可以同时使用,除非这个字段属于分组
5. select deptno,avg(sal) from emp group by deptno order by deptno desc;
6. -- 统计各个岗位的平均工资
7. select job,avg(sal) from emp group by job;
8. select job,lower(job),avg(sal) from emp group by job;

###### having分组后筛选

1. -- 统计各个部门的平均工资 ,只显示平均工资2000以上的 - 分组以后进行二次筛选 having
2. select deptno,avg(sal) from emp group by deptno having avg(sal) > 2000;
3. select deptno,avg(sal) 平均工资 from emp group by deptno having 平均工资 > 2000;
4. select deptno,avg(sal) 平均工资 from emp group by deptno having 平均工资 > 2000 order by deptno desc;
5. -- 统计各个岗位的平均工资,除了MANAGER
6. -- 方法1：
7. select job,avg(sal) from emp where job != 'MANAGER' group by job;
8. -- 方法2：
9. select job,avg(sal) from emp group by job having job != 'MANAGER' ;
10. -- where在分组前进行过滤的，having在分组后进行后滤。

###### 单表查询总结

【1】select语句总结

select column, group\_function(column)

from table

[where condition]

[group by  group\_by\_expression]

[having group\_condition]

[order by column];

注意：顺序固定，不可以改变顺序

【2】select语句的执行顺序

from--where -- group  by–  select   -  having-   order  by

【3】单表查询练习：

1. -- 单表查询练习：
2. -- 列出工资最小值小于2000的职位
3. select job,min(sal)
4. from emp
5. group by job
6. having min(sal) < 2000 ;
7. -- 列出平均工资大于1200元的部门和工作搭配组合
8. select deptno,job,avg(sal)
9. from emp
10. group by deptno,job
11. having avg(sal) > 1200
12. order by deptno;
13. -- 统计[人数小于4的]部门的平均工资。
14. select deptno,count(1),avg(sal)
15. from emp
16. group by deptno
17. having count(1) < 4
18. -- 统计各部门的最高工资，排除最高工资小于3000的部门。
19. select deptno,max(sal)
20. from emp
21. group by deptno
22. having max(sal) < 3000;

##### 多表查询

###### 99语法：交叉连接，自然连接，内连接查询

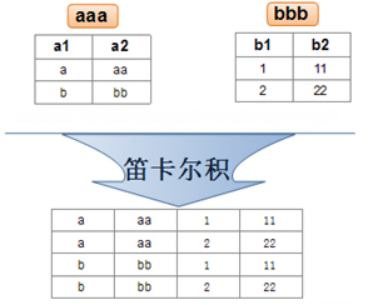
【1】多表查询引入：

实际开发中往往需要针对两张甚至更多张数据表进行操作，而这多张表之间需要使用主键和外键关联在一起，然后使用连接查询来查询多张表中满足要求的数据记录。

一条SQL语句查询多个表，得到一个结果，包含多个表的数据。效率高。在SQL99中，连接查询需要使用join关键字实现。

提供了多种连接查询的类型： cross   natural   using   on

交叉连接（CROSS JOIN）是对两个或者多个表进行笛卡儿积操作，所谓笛卡儿积就是关系代数里的一个概念，表示两个表中的每一行数据任意组合的结果。比如：有两个表，左表有m条数据记录，x个字段，右表有n条数据记录，y个字段，则执行交叉连接后将返回m\*n条数据记录，x+y个字段。笛卡儿积示意图如图所示。



**【2】sql展示：**

1. -- 查询员工的编号，姓名，部门编号：
2. select \* from emp;
3. select empno,ename,deptno from emp;
4. -- 查询员工的编号，姓名，部门编号,部门名称：
5. select \* from emp; -- 14条记录
6. select \* from dept; -- 4条记录
7. -- 多表查询 ：
8. -- 交叉连接：cross join
9. select \*
10. from emp
11. cross join dept; -- 14\*4 = 56条 笛卡尔乘积 ： 没有实际意义，有理论意义
12. select \*
13. from emp
14. join dept; -- cross 可以省略不写，mysql中可以，oracle中不可以
15. -- 自然连接：natural join
16. -- 优点：自动匹配所有的同名列 ,同名列只展示一次 ，简单
17. select \*
18. from emp
19. natural join dept;
20. select empno,ename,sal,dname,loc
21. from emp
22. natural join dept;
23. -- 缺点： 查询字段的时候，没有指定字段所属的数据库表，效率低
24. -- 解决： 指定表名：
25. select emp.empno,emp.ename,emp.sal,dept.dname,dept.loc,dept.deptno
26. from emp
27. natural join dept;
28. -- 缺点：表名太长
29. -- 解决：表起别名
30. select e.empno,e.ename,e.sal,d.dname,d.loc,d.deptno
31. from emp e
32. natural join dept d;
33. -- 自然连接 natural join 缺点：自动匹配表中所有的同名列，但是有时候我们希望只匹配部分同名列：
34. -- 解决： 内连接 - using子句：
35. select \*
36. from emp e
37. inner join dept d -- inner可以不写
38. using(deptno) -- 这里不能写natural join了 ,这里是内连接
39. -- using缺点：关联的字段，必须是同名的
40. -- 解决： 内连接 - on子句：
41. select \*
42. from emp e
43. inner join dept d
44. on (e.deptno = d.deptno);
45. -- 多表连接查询的类型： 1.交叉连接 cross join 2. 自然连接 natural join
46. -- 3. 内连接 - using子句 4.内连接 - on子句
47. -- 综合看：内连接 - on子句
48. select \*
49. from emp e
50. inner join dept d
51. on (e.deptno = d.deptno)
52. where sal > 3500;
53. -- 条件：
54. -- 1.筛选条件 where having
55. -- 2.连接条件 on,using,natural
56. -- SQL99语法 ：筛选条件和连接条件是分开的

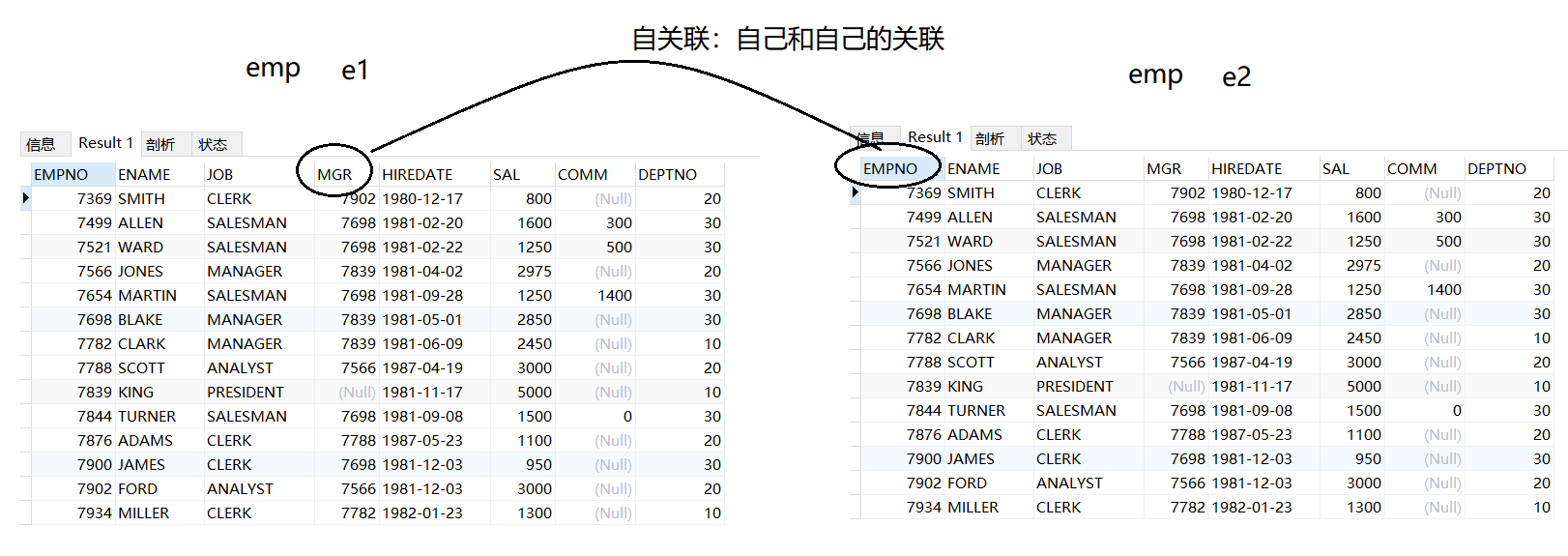
###### 99语法：外连接查询

1. -- inner join - on子句： 显示的是所有匹配的信息
2. select \*
3. from emp e
4. inner join dept d
5. on e.deptno = d.deptno;
6. select \* from emp;
7. select \* from dept;
8. -- 问题：
9. -- 1.40号部分没有员工，没有显示在查询结果中
10. -- 2.员工scott没有部门，没有显示在查询结果中
11. -- 外连接：除了显示匹配的数据之外，还可以显示不匹配的数据
12. -- 左外连接： left outer join -- 左面的那个表的信息，即使不匹配也可以查看出效果
13. select \*
14. from emp e
15. left outer join dept d
16. on e.deptno = d.deptno;
17. -- 右外连接： right outer join -- 右面的那个表的信息，即使不匹配也可以查看出效果
18. select \*
19. from emp e
20. right outer join dept d
21. on e.deptno = d.deptno;
22. -- 全外连接 full outer join -- 这个语法在mysql中不支持，在oracle中支持 -- 展示左，右表全部不匹配的数据
23. -- scott ，40号部门都可以看到
24. select \*
25. from emp e
26. full outer join dept d
27. on e.deptno = d.deptno;
28. -- 解决mysql中不支持全外连接的问题：
29. select \*
30. from emp e
31. left outer join dept d
32. on e.deptno = d.deptno
33. union -- 并集 去重 效率低
34. select \*
35. from emp e
36. right outer join dept d
37. on e.deptno = d.deptno;
38. select \*
39. from emp e
40. left outer join dept d
41. on e.deptno = d.deptno
42. union all-- 并集 不去重 效率高
43. select \*
44. from emp e
45. right outer join dept d
46. on e.deptno = d.deptno;
47. -- mysql中对集合操作支持比较弱，只支持并集操作，交集，差集不支持（oracle中支持）
48. -- outer可以省略不写

###### 99语法：三表连接查询

1. -- 查询员工的编号、姓名、薪水、部门编号、部门名称、薪水等级
2. select \* from emp;
3. select \* from dept;
4. select \* from salgrade;
5. select e.ename,e.sal,e.empno,e.deptno,d.dname,s.\*
6. from emp e
7. right outer join dept d
8. on e.deptno = d.deptno
9. inner join salgrade s
10. on e.sal between s.losal and s.hisal

###### 99语法：自连接查询



1. -- 查询员工的编号、姓名、上级编号,上级的姓名
2. select \* from emp;
3. select e1.empno 员工编号,e1.ename 员工姓名,e1.mgr 领导编号,e2.ename 员工领导姓名
4. from emp e1
5. inner join emp e2
6. on e1.mgr = e2.empno;
7. -- 左外连接：
8. select e1.empno 员工编号,e1.ename 员工姓名,e1.mgr 领导编号,e2.ename 员工领导姓名
9. from emp e1
10. left outer join emp e2
11. on e1.mgr = e2.empno;

###### 92语法：多表查询

1. -- 查询员工的编号，员工姓名，薪水，员工部门编号，部门名称：
2. select e.empno,e.ename,e.sal,e.deptno,d.dname
3. from emp e,dept d
4. -- 相当于99语法中的cross join ,出现笛卡尔积，没有意义
5. select e.empno,e.ename,e.sal,e.deptno,d.dname
6. from emp e,dept d
7. where e.deptno = d.deptno;
8. -- 相当于99语法中的natural join
9. -- 查询员工的编号，员工姓名，薪水，员工部门编号，部门名称，查询出工资大于2000的员工
10. select e.empno,e.ename,e.sal,e.deptno,d.dname
11. from emp e,dept d
12. where e.deptno = d.deptno and e.sal > 2000;
13. -- 查询员工的名字，岗位，上级编号，上级名称（自连接）：
14. select e1.ename,e1.job,e1.mgr ,e2.ename
15. from emp e1,emp e2
16. where e1.mgr = e2.empno;
17. -- 查询员工的编号、姓名、薪水、部门编号、部门名称、薪水等级
18. select e.empno,e.ename,e.sal,e.deptno,d.dname,s.grade
19. from emp e,dept d,salgrade s
20. where e.deptno = d.deptno and e.sal >= s.losal and e.sal <= s.hisal;
21. -- 总结：
22. -- 1.92语法麻烦
23. -- 2.92语法中 表的连接条件 和 筛选条件 是放在一起的没有分开
24. -- 3.99语法中提供了更多的查询连接类型：cross,natural,inner,outer

##### 子查询

###### 不相关子查询

【1】什么是子查询？

一条SQL语句含有多个select，

1. -- 引入子查询：
2. -- 查询所有比“CLARK”工资高的员工的信息
3. -- 步骤1：“CLARK”工资
4. select sal from emp where ename = 'CLARK' -- 2450
5. -- 步骤2：查询所有工资比2450高的员工的信息
6. select \* from emp where sal > 2450;
7. -- 两次命令解决问题 --》效率低 ，第二个命令依托于第一个命令，第一个命令的结果给第二个命令使用，但是
8. -- 因为第一个命令的结果可能不确定要改，所以第二个命令也会导致修改
9. -- 将步骤1和步骤2合并 --》子查询：
10. select \* from emp where sal > (select sal from emp where ename = 'CLARK');
11. -- 一个命令解决问题 --》效率高

【2】执行顺序：

先执行子查询，再执行外查询；

【3】不相关子查询：

子查询可以独立运行，称为不相关子查询。

【4】不相关子查询分类：

根据子查询的结果行数，可以分为单行子查询和多行子查询。

单行子查询

1. -- 单行子查询：
2. -- 查询工资高于平均工资的雇员名字和工资。
3. select ename,sal
4. from emp
5. where sal > (select avg(sal) from emp);
6. -- 查询和CLARK同一部门且比他工资低的雇员名字和工资。
7. select ename,sal
8. from emp
9. where deptno = (select deptno from emp where ename = 'CLARK')
10. and
11. sal < (select sal from emp where ename = 'CLARK')
12. -- 查询职务和SCOTT相同，比SCOTT雇佣时间早的雇员信息
13. select \*
14. from emp
15. where job = (select job from emp where ename = 'SCOTT')
16. and
17. hiredate < (select hiredate from emp where ename = 'SCOTT')

多行子查询

1. -- 多行子查询：
2. -- 【1】查询【部门20中职务同部门10的雇员一样的】雇员信息。
3. -- 查询雇员信息
4. select \* from emp;
5. -- 查询部门20中的雇员信息
6. select \* from emp where deptno = 20;-- CLERK,MANAGER,ANALYST
7. -- 部门10的雇员的职务：
8. select job from emp where deptno = 10; -- MANAGER,PRESIDENT,CLERK
9. -- 查询部门20中职务同部门10的雇员一样的雇员信息。
10. select \* from emp
11. where deptno = 20
12. and job in (select job from emp where deptno = 10)
13. -- > Subquery returns more than 1 row
14. select \* from emp
15. where deptno = 20
16. and job = any(select job from emp where deptno = 10)
17. -- 【2】查询工资比所有的“SALESMAN”都高的雇员的编号、名字和工资。
18. -- 查询雇员的编号、名字和工资
19. select empno,ename,sal from emp
20. -- “SALESMAN”的工资：
21. select sal from emp where job = 'SALESMAN'
22. -- 查询工资比所有的“SALESMAN”都高的雇员的编号、名字和工资。
23. -- 多行子查询：
24. select empno,ename,sal
25. from emp
26. where sal > all(select sal from emp where job = 'SALESMAN');
27. -- 单行子查询：
28. select empno,ename,sal
29. from emp
30. where sal > (select max(sal) from emp where job = 'SALESMAN');
31. -- 【3】查询工资低于任意一个“CLERK”的工资的雇员信息。
32. -- 查询雇员信息
33. select \* from emp;
34. -- 查询工资低于任意一个“CLERK”的工资的雇员信息
35. select \*
36. from emp
37. where sal < any(select sal from emp where job = 'CLERK')
38. and job != 'CLERK'
39. -- 单行子查询：
40. select \*
41. from emp
42. where sal < (select max(sal) from emp where job = 'CLERK')
43. and job != 'CLERK'

###### 相关子查询

【1】不相关的子查询引入：

不相关的子查询：子查询可以独立运行，先运行子查询，再运行外查询。

相关子查询：子查询不可以独立运行，并且先运行外查询，再运行子查询

【2】不相关的子查询优缺点：

好处：简单   功能强大（一些使用不相关子查询不能实现或者实现繁琐的子查询，可以使用相关子查询实现）

缺点：稍难理解

【3】sql展示：

1. -- 【1】查询最高工资的员工 （不相关子查询）
2. select \* from emp where sal = (select max(sal) from emp)
3. -- 【2】查询本部门最高工资的员工 （相关子查询）
4. -- 方法1：通过不相关子查询实现：
5. select \* from emp where deptno = 10 and sal = (select max(sal) from emp where deptno = 10)
6. union
7. select \* from emp where deptno = 20 and sal = (select max(sal) from emp where deptno = 20)
8. union
9. select \* from emp where deptno = 30 and sal = (select max(sal) from emp where deptno = 30)
10. -- 缺点：语句比较多，具体到底有多少个部分未知
11. -- 方法2： 相关子查询
12. select \* from emp e where sal = (select max(sal) from emp where deptno = e.deptno) order by deptno
13. -- 【3】查询工资高于其所在岗位的平均工资的那些员工 （相关子查询）
14. -- 不相关子查询：
15. select \* from emp where job = 'CLERK' and sal >= (select avg(sal) from emp where job = 'CLERK')
16. union ......
17. -- 相关子查询：
18. select \* from emp e where sal >= (select avg(sal) from emp e2 where e2.job = e.job)

#### 数据库对象

##### 事务

###### 事务及其特征

事务（Transaction）是用来维护数据库完整性的，它能够保证一系列的MySQL操作要么全部执行，要么全不执行。

例子1：

举一个例子来进行说明，例如转账操作：A账户要转账给B账户，那么A账户上减少的钱数和B账户上增加的钱数必须一致，也就是说A账户的转出操作和B账户的转入操作要么全部执行，要么全不执行；如果其中一个操作出现异常而没有执行的话，就会导致账户A和账户B的转入转出金额不一致的情况，为而事实上这种情况是不允许发生的，所以为了防止这种情况的发生，需要使用事务处理。

例子2：

在淘宝购物下订单的时候，商家库存要减少，订单增加记录，付款我的账号少100元...操作要么全部执行，要么全不执行

**【1】事务的概念**

事务（Transaction）指的是一个操作序列，该操作序列中的多个操作要么都做，要么都不做，是一个不可分割的工作单位，是数据库环境中的逻辑工作单位，由DBMS（数据库管理系统）中的事务管理子系统负责事务的处理。

目前常用的存储引擎有InnoDB（MySQL5.5以后默认的存储引擎）和MyISAM（MySQL5.5之前默认的存储引擎），其中InnoDB支持事务处理机制，而MyISAM不支持。

**【2】事务的特性**

事务处理可以确保除非事务性序列内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的序列，可以简化错误恢复并使应用程序更加可靠。

但并不是所有的操作序列都可以称为事务，这是因为一个操作序列要成为事务，必须满足事务的原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。这四个特性简称为ACID特性。



1)原子性

原子是自然界最小的颗粒，具有不可再分的特性。事务中的所有操作可以看做一个原子，事务是应用中不可再分的最小的逻辑执行体。

使用事务对数据进行修改的操作序列，要么全部执行，要么全不执行。通常，某个事务中的操作都具有共同的目标，并且是相互依赖的。如果数据库系统只执行这些操作中的一部分，则可能会破坏事务的总体目标，而原子性消除了系统只处理部分操作的可能性。

2) 一致性

一致性是指事务执行的结果必须使数据库从一个一致性状态，变到另一个一致性状态。当数据库中只包含事务成功提交的结果时，数据库处于一致性状态。一致性是通过原子性来保证的。

例如：在转账时，只有保证转出和转入的金额一致才能构成事务。也就是说事务发生前和发生后，数据的总额依然匹配。

3) 隔离性

隔离性是指各个事务的执行互不干扰，任意一个事务的内部操作对其他并发的事务，都是隔离的。也就是说：并发执行的事务之间既不能看到对方的中间状态，也不能相互影响。

例如：在转账时，只有当A账户中的转出和B账户中转入操作都执行成功后才能看到A账户中的金额减少以及B账户中的金额增多。并且其他的事务对于转账操作的事务是不能产生任何影响的。

4)持久性

持久性指事务一旦提交，对数据所做的任何改变，都要记录到永久存储器中，通常是保存进物理数据库，即使数据库出现故障，提交的数据也应该能够恢复。但如果是由于外部原因导致的数据库故障，如硬盘被损坏，那么之前提交的数据则有可能会丢失。

**【3】sql展示：使用事务保证转账安全**

1. -- 创建账户表：
2. create table account(
3. id int primary key auto\_increment,
4. uname varchar(10) not null,
5. balance double
6. );
7. -- 查看账户表：
8. select \* from account;
9. -- 在表中插入数据：
10. insert into account values (null,'丽丽',2000),(null,'小刚',2000);
11. -- 丽丽给小刚 转200元：
12. update account set balance = balance - 200 where id = 1;
13. update account set balance = balance + 200 where id = 2;
14. -- 默认一个DML语句是一个事务，所以上面的操作执行了2个事务。
15. update account set balance = balance - 200 where id = 1;
16. update account set balance = balance2 + 200 where id = 2;
17. -- 必须让上面的两个操作控制在一个事务中：
18. -- 手动开启事务：
19. start transaction;
20. update account set balance = balance - 200 where id = 1;
21. update account set balance = balance + 200 where id = 2;
22. -- 手动回滚：刚才执行的操作全部取消：
23. rollback;
24. -- 手动提交：
25. commit;
26. -- 在回滚和提交之前，数据库中的数据都是操作的缓存中的数据，而不是数据库的真实数据

###### 事务并发问题

**脏读（Dirty read）**

当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。



**不可重复读**

（Unrepeatableread）: 指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。



**幻读**

（Phantom read）: 幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。



**不可重复度和幻读区别：**

不可重复读的重点是修改，幻读的重点在于新增或者删除。

**解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表**

例1（同样的条件, 你读取过的数据, 再次读取出来发现值不一样了 ）：事务1中的A先生读取自己的工资为 1000的操作还没完成，事务2中的B先生就修改了A的工资为2000，导 致A再读自己的工资时工资变为 2000；这就是不可重复读。

例2（同样的条件, 第1次和第2次读出来的记录数不一样 ）：假某工资单表中工资大于3000的有4人，事务1读取了所有工资大于3000的人，共查到4条记录，这时事务2 又插入了一条工资大于3000的记录，事务1再次读取时查到的记录就变为了5条，这样就导致了幻读

###### 事务隔离级别

事务的隔离级别用于决定如何控制并发用户读写数据的操作。数据库是允许多用户并发访问的，如果多个用户同时开启事务并对同一数据进行读写操作的话，有可能会出现脏读、不可重复读和幻读问题，所以MySQL中提供了四种隔离级别来解决上述问题。

    事务的隔离级别从低到高依次为READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ以及SERIALIZABLE，隔离级别越低，越能支持高并发的数据库操作。

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| READ UNCOMMITTED | √ | √ | √ |
| READ COMMITTED | × | √ | √ |
| REPEATABLE READ | × | × | √ |
| SERIALIZABLE | × | × | × |

 PS：√  代表会出现问题   ， ×代表不会出现问题 = 解决问题

1. -- 查看默认的事务隔离级别 MySQL默认的是repeatable read
2. select @@transaction\_isolation;
3. -- 设置事务的隔离级别 （设置当前会话的隔离级别）
4. set session transaction isolation level read uncommitted;
5. set session transaction isolation level read committed;
6. set session transaction isolation level repeatable read;
7. set session transaction isolation level serializable;
8. start transaction ;
9. select \* from account where id = 1;

##### 视图

#### 【1】视图的概念

视图（view）是一个从单张或多张基础数据表或其他视图中构建出来的虚拟表。同基础表一样，视图中也包含了一系列带有名称的列和行数据，但是数据库中只是存放视图的定义，也就是动态检索数据的查询语句，而并不存放视图中的数据，这些数据依旧存放于构建视图的基础表中，只有当用户使用视图时才去数据库请求相对应的数据，即视图中的数据是在引用视图时动态生成的。因此视图中的数据依赖于构建视图的基础表，如果基本表中的数据发生了变化，视图中相应的数据也会跟着改变。

PS:视图本质上就是：一个查询语句，是一个虚拟的表，不存在的表，你查看视图，其实就是查看视图对应的sql语句

#### 【2】视图的好处

简化用户操作：视图可以使用户将注意力集中在所关心地数据上，而不需要关心数据表的结构、与其他表的关联条件以及查询条件等。

对机密数据提供安全保护：有了视图，就可以在设计数据库应用系统时，对不同的用户定义不同的视图，避免机密数据（如，敏感字段“salary”）出现在不应该看到这些数据的用户视图上。这样视图就自动提供了对机密数据的安全保护功能

#### 【3】SQL展示：

1. -- 创建/替换单表视图：
2. create or replace view myview01
3. as
4. select empno,ename,job,deptno
5. from emp
6. where deptno = 20
7. with check option;
8. -- 查看视图：
9. select \* from myview01;
10. -- 在视图中插入数据：
11. insert into myview01 (empno,ename,job,deptno) values (9999,'lili','CLERK',20);
12. insert into myview01 (empno,ename,job,deptno) values (8888,'nana','CLERK',30);
13. insert into myview01 (empno,ename,job,deptno) values (7777,'feifei','CLERK',30);
14. -- > 1369 - CHECK OPTION failed 'mytestdb.myview01'
15. -- 创建/替换多表视图：
16. create or replace view myview02
17. as
18. select e.empno,e.ename,e.sal,d.deptno,d.dname
19. from emp e
20. join dept d
21. on e.deptno = d.deptno
22. where sal > 2000 ;
23. select \* from myview02;
24. -- 创建统计视图：
25. create or replace view myview03
26. as
27. select e.deptno,d.dname,avg(sal),min(sal),count(1)
28. from emp e
29. join dept d
30. using(deptno)
31. group by e.deptno ;
32. select \* from myview03;
33. -- 创建基于视图的视图：
34. create or replace view myview04
35. as
36. select \* from myview03 where deptno = 20;
37. select \* from myview04;

#### 存储过程

#### 【1】什么是存储过程(Stored Procedure)

通过前面章节的学习，我们已经知道SQL是一种非常便利的语言。从数据库抽取数据，或者对特定的数据集中更新时，都能通过简洁直观的代码实现。

但是这个所谓的“简洁”也是有限制，SQL基本是一个命令实现一个处理，是所谓的非程序语言。

在不能编写流程的情况下，所有的处理只能通过一个个命令来实现。当然，通过使用连接及子查询，即使使用SQL的单一命令也能实现一些高级的处理，但是，其局限性是显而易见的。例如，在SQL中就很难实现针对不同条件进行不同的处理以及循环等功能。

这个时候就出现了存储过程这个概念，简单地说，存储过程就是数据库中保存(Stored)的一系列SQL命令（Procedure）的集合。也可以将其看作相互之间有关系的SQL命令组织在一起形成的一个小程序。

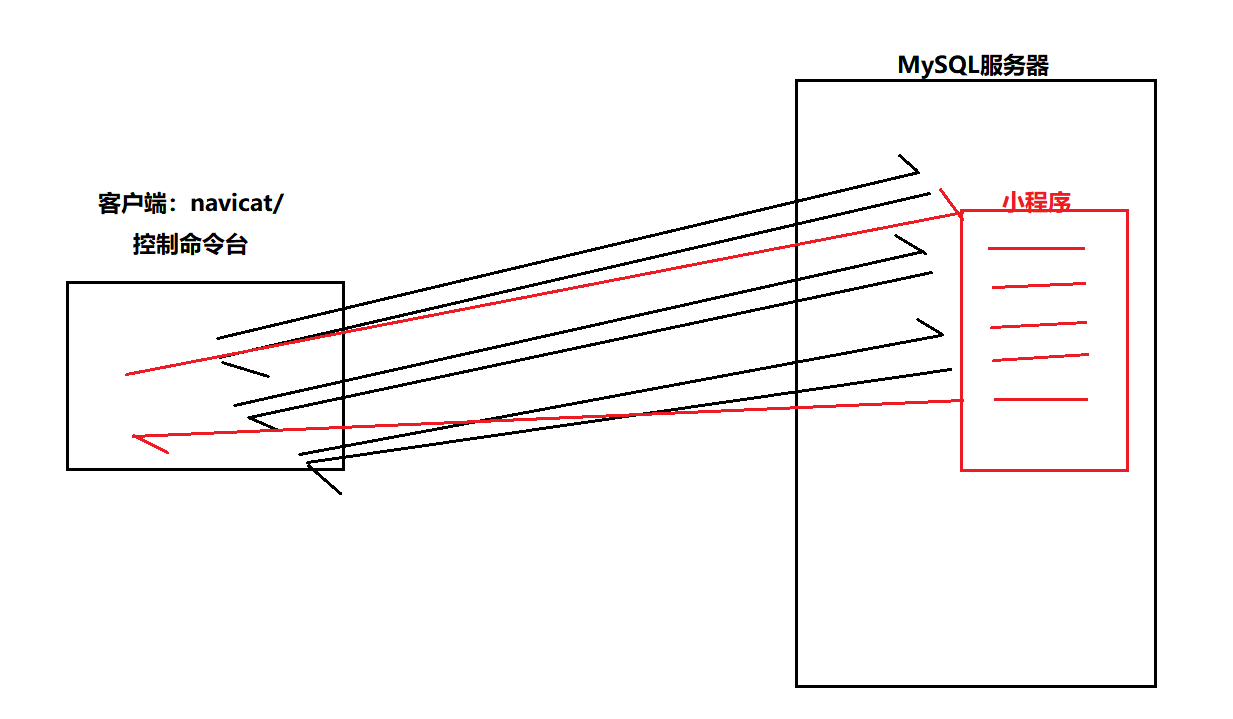
#### 【2】存储过程的优点

1) 提高执行性能。存储过程执行效率之所高，在于普通的SQL语句，每次都会对语法分析，编译，执行，而存储过程只是在第一次执行语法分析，编译，执行，以后都是对结果进行调用。

2) 可减轻网络负担。使用存储过程，复杂的数据库操作也可以在数据库服务器中完成。只需要从客户端(或应用程序)传递给数据库必要的参数就行，比起需要多次传递SQL命令本身，这大大减轻了网络负担。

3) 可将数据库的处理黑匣子化。应用程序中完全不用考虑存储过程的内部详细处理，只需要知道调用哪个存储过程就可以了

【3】图解：



【4】展示存储过程：

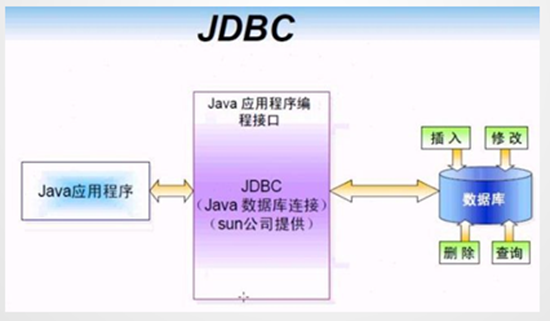
1. -- 定义一个没有返回值 存储过程
2. -- 实现：模糊查询操作：
3. select \* from emp where ename like '%A%';
4. create procedure mypro01(name varchar(10))
5. begin
6. if name is null or name = "" then
7. select \* from emp;
8. else
9. select \* from emp where ename like concat('%',name,'%');
10. end if;
11. end;
12. -- 删除存储过程：
13. drop procedure mypro01;
14. -- 调用存储过程：
15. call mypro01(null);
16. call mypro01('R');
17. -- 定义一个 有返回值的存储过程：
18. -- 实现：模糊查询操作：
19. -- in 参数前面的in可以省略不写
20. -- found\_rows()mysql中定义的一个函数，作用返回查询结果的条数
21. create procedure mypro02(in name varchar(10),out num int(3))
22. begin
23. if name is null or name = "" then
24. select \* from emp;
25. else
26. select \* from emp where ename like concat('%',name,'%');
27. end if;
28. select found\_rows() into num;
29. end;
30. -- -- 调用存储过程：
31. call mypro02(null,@num);
32. select @num;
33. call mypro02('R',@aaa);
34. select @aaa;

# 第18章\_JDBC

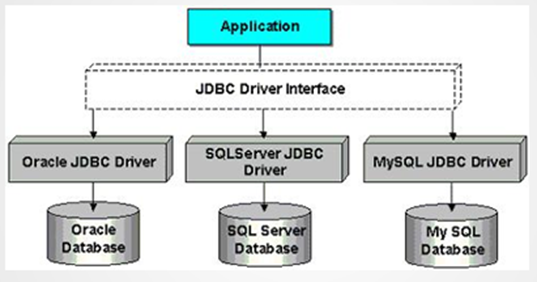
## 1\_JDBC概述

**什么是JDBC**

JDBC（Java DataBase Connectivity,  Java数据库连接） ,是一种用于执行SQL语句的Java API，为多种关系数据库提供统一访问,它由一组用Java语言编写的类和接口组成



有了JDBC，程序员只需用JDBC API写一个程序，就可访问所有数据库。



Sun公司、数据库厂商、程序员三方关系

SUN公司是规范制定者，制定了规范JDBC（连接数据库规范）

DriverManager类   作用：管理各种不同的JDBC驱动

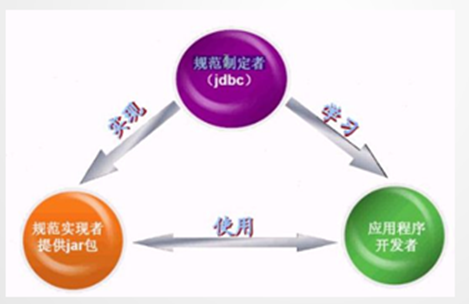
Connection接口

Statement接口和PreparedStatement接口

ResultSet接口

数据库厂商微软、甲骨文等分别提供实现JDBC接口的驱动jar包

程序员学习JDBC规范来应用这些jar包里的类。



JDBC访问数据库编码步骤

1：加载一个Driver驱动

2：创建数据库连接（Connection）

3：创建SQL命令发送器Statement

4：通过Statement发送SQL命令并得到结果

5：处理结果（select语句）

6：关闭数据库资源ResultSet Statement Connection

创建模块/项目 导入jar包

### JDBD历史版本及特征

#### JDBC 1.0

JDBC 1.0 随JDK1.1一起发布,JDBC操作相关的接口和类位于java.sql包中。

#### JDBC 2.0

JDBC 2.0 API被划分为两部分：核心API和扩展API,有两个包,分别是java.sql包和javax.sql包。

**java.sql核心API包**

在支持新功能方面：包括结果集可以向后滚动，批量的更新数据。另外，还提供了UNICODE字符集的字符流操作。

在支持SQL的数据类型方面：新增加的BLOB, CLOB,和数组接口能够是应用程序操作大块的数据类型

**javax.sql扩展API包**

DataSource数据源接口：

JDBC1.0是原来是用DriverManager类来产生一个对数据源的连接。JDBC2.0用一种替代的方法，使用DataSource的实现，代码变的更小巧精致，也更容易控制。

Connection pooling

如果DataSource对象实现与一个支持连接池的中间层的服务器一起工作，DataSource对象就会自动的返回连接池中的连接，这个连接也是可以重复利用的。

Distrubute transaction：

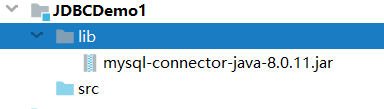
在一个事务中涉及到了多个数据库服务器。获得一个用来支持分布式事务的连接与获得连接池中的连接是很相似的。同样，不同之处在于DataSource的实现上的不同，而不是在应用程序中获得连接的方式上有什么不同。

Rowsets：

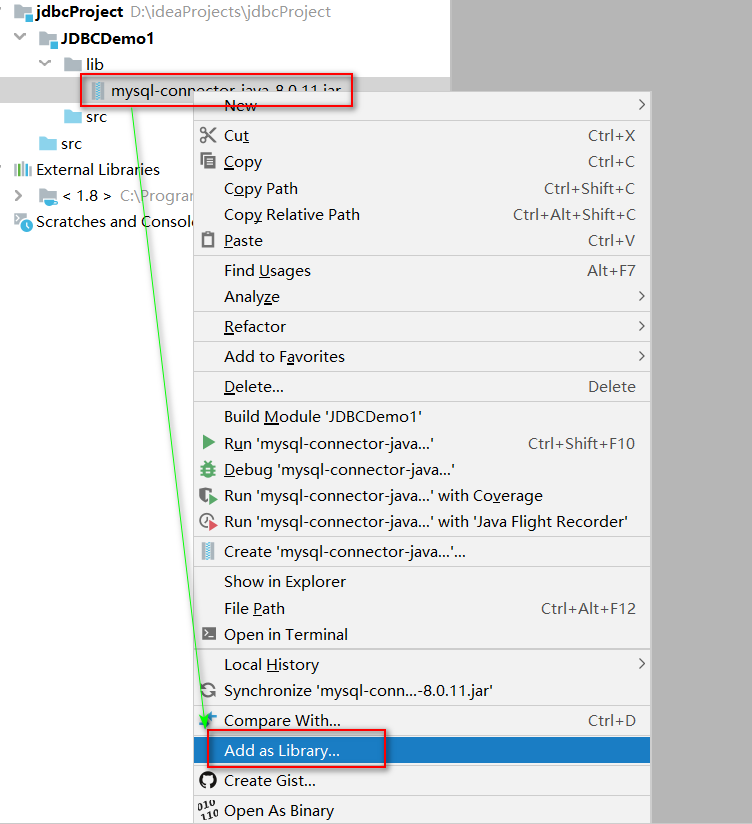
RowSet接口扩展了ResultSet接口。这样RowSet对象就有了ResultSet对象所有的功能。不可以滚动的ResultSet变成了可以滚动的RowSet。

#### 2\_JDBC初识

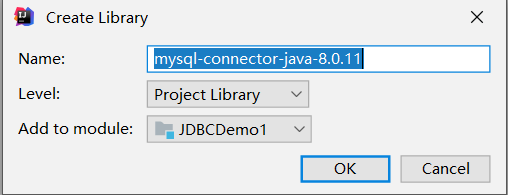
1创建项目和模块.将jar文件放入项目的lib目录中



2给当前项目添加依赖(告诉当前项目/模块可以依赖jar文件中的代码)



点击OK



向部门表中添加一条数据

1. package com.msb.test1;
2. import java.sql.Connection;
3. import java.sql.Driver;
4. import java.sql.DriverManager;
5. import java.sql.Statement;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestJDBC {
11. public static void main(String[] args) throws Exception {
12. /\*
13. \* 向Dept表增加一条数据
14. \*
15. \* \*/
16. //1加载驱动 Driver
17. Driver driver =new com.mysql.cj.jdbc.Driver();
18. //2注册驱动 DriverManager
19. DriverManager.registerDriver(driver);
20. //3获得链接 Connection
21. /\*
22. \*
23. \* user:用户名
24. \* password:密码
25. \* url:统一资源定位符 定位我们要连接的数据库的
26. \* 1协议 jdbc:mysql
27. \* 2IP 127.0.0.1/localhost
28. \* 3端口号 3306
29. \* 4数据库名字 mydb
30. \* 5参数
31. \* 协议://ip:端口/资源路径?参数名=参数值&参数名=参数值&....
32. \* jdbc:mysql://127.0.0.1:3306/mydb
33. \* \*/
34. String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai";
35. String user="root";
36. String password="root";
37. Connection connection =DriverManager.getConnection(url, user,password );
38. //4获得语句对象 Statment
39. Statement statement = connection.createStatement();
40. //5执行SQL语句,返回结果
41. /\*
42. \* insert delete update 操作都是调用statement.executeUpdate
43. \* executeUpdate返回一个int值,代表数据库多少行数据发生了变化
44. \* \*/
45. String sql="insert into dept values(50,'教学部','北京');";
46. int rows = statement.executeUpdate(sql);
47. System.out.println("影响数据行数为:"+rows);
48. //6释放资源
49. /\*
50. \* 注意顺序
51. \* 后获得的先关闭,先获得的后关闭
52. \* \*/
53. statement.close();
54. connection.close();
55. }
56. }

**总结**

MySQL8中数据库连接的四个参数有两个发生了变化

String driver = "com.mysql.cj.jdbc.Driver";

String url = "jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Shanghai";

或者String url = ".......serverTimezone=GMT%2B8";

错误1：Exception in thread "main" java.lang.ClassNotFoundException: com.mysql.jdbc2.Driver

原因：没有添加jar包或者com.mysql.jdbc2.Driver路径错误

错误2：Exception in thread "main" java.sql.SQLException:

No suitable driver found for jbdc:mysql://127.0.0.1:3306/stumgr

原因：url错误

错误3：Exception in thread "main" java.sql.SQLException:

Access denied for user &apos;root&apos;@&apos;localhost&apos; (using password: YES)

原因：用户名或者密码错误

  错误4：Exception in thread "main" com.mysql.jdbc.exceptions

.jdbc4.MySQLIntegrityConstraintViolationException:Duplicate entry &apos;90&apos; for key &apos;PRIMARY&apos;

原因：主键冲突

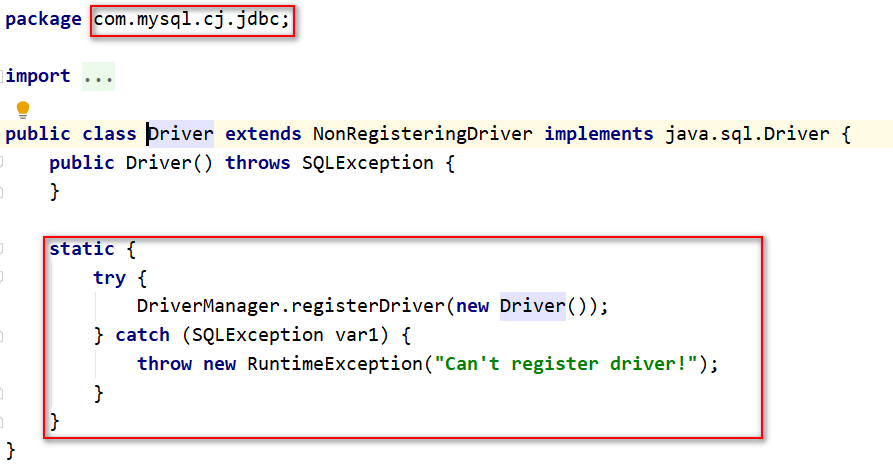
# 错误5：Public Key Retrieval is not allowed

如果用户使用 sha256\_password 认证，密码在传输过程中必须使用 TLS 协议保护，但是如果 RSA 公钥不可用，可以使用服务器提供的公钥；可以在连接中通过 ServerRSAPublicKeyFile 指定服务器的 RSA 公钥，或者AllowPublicKeyRetrieval=True参数以允许客户端从服务器获取公钥；但是需要注意的是 AllowPublicKeyRetrieval=True可能会导致恶意的代理通过中间人攻击(MITM)获取到明文密码，所以默认是关闭的，必须显式开启

在jdbc连接添加上参数allowPublicKeyRetrieval=true即可，注意参数间用&

### 驱动的加载

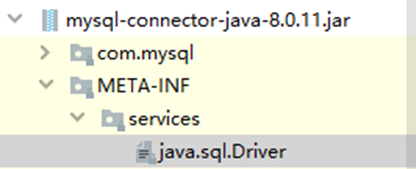
加载数据库驱动时,我们可以通过自己创建一个实例的方式,然后去注册驱动



在查看Driver的源代码时我们发现,该类内部有一个静态代码块,在代码块中就是在实例化一个驱动并在驱动中心注册.静态代码块会在类进入内存时执行,也就是说,我们只要让该类字节码进入内存,就会自动完成注册,不需要我们手动去new

所以我们在代码中直接使用反射,通过Class.forName("com.mysql.jdbc.Driver"),加载该类进入内存即可

我们继续查看jar包发现,jar包中已经默认配置了驱动类的加载



jar--META-INF--services--java.sql.Driver--com.mysql.jdbc.Driver,在加载jar包时,会自动读取该内容并加载驱动,所以我们不去编写Class.forName("com.mysql.jdbc.Driver"),程序也是可以自动完成加载驱动的

**结合异常处理代码**

1. package com.msb.test1;
2. import java.sql.Connection;
3. import java.sql.DriverManager;
4. import java.sql.SQLException;
5. import java.sql.Statement;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestJDBC3 {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
13. private static String user="root";
14. private static String password="root";
15. public static void main(String[] args) {
16. Connection connection=null;
17. Statement statement=null;
18. try{
19. Class.forName(driver);
20. connection =DriverManager.getConnection(url, user,password);
21. statement = connection.createStatement();
22. String sql="insert into dept values(DEFAULT ,'助教部门','北京');";
23. int rows = statement.executeUpdate(sql);
24. System.out.println("影响数据行数为:"+rows);
25. }catch (Exception e){
26. e.printStackTrace();
27. }finally {
28. if(null != statement){
29. try {
30. statement.close();
31. } catch (SQLException e) {
32. e.printStackTrace();
33. }
34. }
35. if(null != connection){
36. try {
37. connection.close();
38. } catch (SQLException e) {
39. e.printStackTrace();
40. }
41. }
42. }
43. }
44. }

### 3\_JDBC完成CURD

### 删除和修改部门信息

1. package com.msb.test1;
2. import java.sql.Connection;
3. import java.sql.DriverManager;
4. import java.sql.SQLException;
5. import java.sql.Statement;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestJDBC4 {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
13. private static String user="root";
14. private static String password="root";
15. public static void main(String[] args) {
16. //testDelete();
17. testUpdate();
18. }
19. public static void testUpdate(){
20. Connection connection=null;
21. Statement statement=null;
22. try{
23. Class.forName(driver);
24. connection =DriverManager.getConnection(url, user,password);
25. statement = connection.createStatement();
26. String sql="update dept set dname='总部',loc='北京' where deptno= 30 ";
27. int rows = statement.executeUpdate(sql);
28. System.out.println("影响数据行数为:"+rows);
29. }catch (Exception e){
30. e.printStackTrace();
31. }finally {
32. if(null != statement){
33. try {
34. statement.close();
35. } catch (SQLException e) {
36. e.printStackTrace();
37. }
38. }
39. if(null != connection){
40. try {
41. connection.close();
42. } catch (SQLException e) {
43. e.printStackTrace();
44. }
45. }
46. }
47. }
48. public static void testDelete(){
49. Connection connection=null;
50. Statement statement=null;
51. try{
52. Class.forName(driver);
53. connection =DriverManager.getConnection(url, user,password);
54. statement = connection.createStatement();
55. String sql="delete from dept where deptno =40";
56. int rows = statement.executeUpdate(sql);
57. System.out.println("影响数据行数为:"+rows);
58. }catch (Exception e){
59. e.printStackTrace();
60. }finally {
61. if(null != statement){
62. try {
63. statement.close();
64. } catch (SQLException e) {
65. e.printStackTrace();
66. }
67. }
68. if(null != connection){
69. try {
70. connection.close();
71. } catch (SQLException e) {
72. e.printStackTrace();
73. }
74. }
75. }
76. }
77. }

### 需求:查询全部 员工信息

1. package com.msb.test1;
2. import java.sql.\*;
3. /\*\*
4. \* @Author: Ma HaiYang
5. \* @Description: MircoMessage:Mark\_7001
6. \*/
7. public class TestJDBC5 {
8. private static String driver ="com.mysql.cj.jdbc.Driver";
9. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
10. private static String user="root";
11. private static String password="root";
12. public static void main(String[] args) {
13. testQuery();
14. }
15. public static void testQuery(){
16. Connection connection = null;
17. Statement statement=null;
18. ResultSet resultSet=null;
19. try{
20. Class.forName(driver);
21. connection = DriverManager.getConnection(url, user,password);
22. statement = connection.createStatement();
23. String sql="select \* from emp";
24. resultSet = statement.executeQuery(sql);
25. while(resultSet.next()){
26. int empno = resultSet.getInt("empno");
27. String ename = resultSet.getString("ename");
28. String job = resultSet.getString("job");
29. int mgr = resultSet.getInt("mgr");
30. Date hiredate = resultSet.getDate("hiredate");
31. double sal= resultSet.getDouble("sal");
32. double comm= resultSet.getDouble("comm");
33. int deptno= resultSet.getInt("deptno");
34. System.out.println(""+empno+" "+ename+" "+job+" "+mgr+" "+hiredate+" "+sal+" "+comm+" "+deptno);
35. }
36. }catch (Exception e){
37. e.printStackTrace();
38. }finally {
39. if(null != resultSet){
40. try {
41. resultSet.close();
42. } catch (SQLException e) {e.printStackTrace();
43. }
44. }
45. if(null != statement){
46. try {
47. statement.close();
48. } catch (SQLException e) {
49. e.printStackTrace();
50. }
51. }
52. if(null != connection){
53. try {
54. connection.close();
55. } catch (SQLException e) {
56. e.printStackTrace();
57. }
58. }
59. }
60. }
61. }

•       ResultSet里的数据一行一行排列，每行有多个字段，且有一个记录指针，指针所指的数据行叫做当前数据行，我们只能来操作当前的数据行。我们如果想要取得某一条记录，就要使用ResultSet的next()方法 ,如果我们想要得到ResultSet里的所有记录，就应该使用while循环。

•       ResultSet对象自动维护指向当前数据行的游标。每调用一次next()方法，游标向下移动一行。

•       初始状态下记录指针指向第一条记录的前面，通过next()方法指向第一条记录。循环完毕后指向最后一条记录的后面。

|  |  |  |
| --- | --- | --- |
| 方法名 | 说 明 | |
| boolean   next() | 将光标从当前位置向下移动一行 | |
| boolean   previous() | 游标从当前位置向上移动一行 | |
| void   close() | 关闭ResultSet 对象 | |
| int   getInt(int colIndex) | | 以int形式获取结果集当前行指定列号值 |
| int   getInt(String colLabel) | | 以int形式获取结果集当前行指定列名值 |
| float   getFloat(int colIndex) | | 以float形式获取结果集当前行指定列号值 |
| Float   getFloat(String colLabel) | | 以float形式获取结果集当前行指定列名值 |
| String   getString(int colIndex) | | 以String 形式获取结果集当前行指定列号值 |
| StringgetString(String   colLabel) | | 以String形式获取结果集当前行指定列名值 |

作为一种好的编程风格，应在不需要Statement对象和Connection对象时显式地关闭它们。关闭Statement对象和Connection对象的语法形式为：用户不必关闭ResultSet。当它的 Statement 关闭、重新执行或用于从多结果序列中获取下一个结果时，该ResultSet将被自动关闭。

**为什么将结果封装成对象或者对象集合?**

1java是面向对象的编程语言,java中所有的数据处理都是基于面向对象的编码风格实现的,让数据以符合java风格的形式存在,便于对数据的后续处理

2ResultSet 集合虽然可以存放数据,但是它是JDBC中查询数据的一种手段,是一种数据的临时存储方案,使用完毕是要进行释放和关闭

#### 封装后台查询数据并在前台显示



如何将结果集中的数据在java中进行存储和传递?

准备和数据库表格相对应的一个实体类,用于封装结果集中的每一条数据,数据库表格中的每一个字段就是实体类的一个属性,实体类的一个对象就可以用于存储数据库表中的一条记录.

**准备实体类**

1. package com.msb.entity;
2. import java.io.Serializable;
3. import java.util.Date;
4. /\*\*
5. \* @Author: Ma HaiYang
6. \* @Description: MircoMessage:Mark\_7001
7. \*/
8. /\*
9. \* 实体类:
10. \* 和数据库表格名称和字段是一一对应的类
11. \* 该类的对象主要用处是存储从数据库中查询出来的数据
12. \* 除此之外,该类没有任何的其他功能
13. \* 要求
14. \* 1类名和表名保持一致 (见名知意)
15. \* 2属性个数和数据库的表的列数保持一致
16. \* 3属性的数据类型和列的数据类型保持一致
17. \* 4属性名和数据库表格的列名要保持一致
18. \* 5所有的属性必须都是私有的 (出于安全考虑)
19. \* 6实体类的属性推荐写成包装类
20. \* 7日期类型推荐写成java.util.Date
21. \* 8所有的属性都要有get和set方法
22. \* 9必须具备空参构造方法
23. \* 10实体类应当实现序列化接口 (mybatis缓存 分布式需要 )
24. \* 11实体类中其他构造方法可选
25. \* \*/
26. public class Emp implements Serializable {
27. private Integer empno;
28. private String ename;
29. private String job;
30. private Integer mgr;
31. private Date hiredate;
32. private Double sal;
33. private Double comm;
34. private Integer deptno;
35. @Override
36. public String toString() {
37. return "Emp{" +
38. "empno=" + empno +
39. ", ename='" + ename + '\'' +
40. ", job='" + job + '\'' +
41. ", mgr=" + mgr +
42. ", hiredate=" + hiredate +
43. ", sal=" + sal +
44. ", comm=" + comm +
45. ", deptno=" + deptno +
46. '}';
47. }
48. public Emp(Integer empno, String ename, String job, Integer mgr, Date hiredate, Double sal, Double comm, Integer deptno) {
49. this.empno = empno;
50. this.ename = ename;
51. this.job = job;
52. this.mgr = mgr;
53. this.hiredate = hiredate;
54. this.sal = sal;
55. this.comm = comm;
56. this.deptno = deptno;
57. }
58. public Emp(){
59. }
60. public Integer getEmpno() {
61. return empno;
62. }
63. public void setEmpno(Integer empno) {
64. this.empno = empno;
65. }
66. public String getEname() {
67. return ename;
68. }
69. public void setEname(String ename) {
70. this.ename = ename;
71. }
72. public String getJob() {
73. return job;
74. }
75. public void setJob(String job) {
76. this.job = job;
77. }
78. public Integer getMgr() {
79. return mgr;
80. }
81. public void setMgr(Integer mgr) {
82. this.mgr = mgr;
83. }
84. public Date getHiredate() {
85. return hiredate;
86. }
87. public void setHiredate(Date hiredate) {
88. this.hiredate = hiredate;
89. }
90. public Double getSal() {
91. return sal;
92. }
93. public void setSal(Double sal) {
94. this.sal = sal;
95. }
96. public Double getComm() {
97. return comm;
98. }
99. public void setComm(Double comm) {
100. this.comm = comm;
101. }
102. public Integer getDeptno() {
103. return deptno;
104. }
105. public void setDeptno(Integer deptno) {
106. this.deptno = deptno;
107. }
108. }

使用实体类封装结果集

1. package com.msb.test1;
2. import com.msb.entity.Emp;
3. import java.sql.\*;
4. import java.util.ArrayList;
5. import java.util.List;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestJDBC5 {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
13. private static String user="root";
14. private static String password="root";
15. public static void main(String[] args) {
16. List<Emp> emps = testQuery();
17. // 遍历集合
18. for (Emp emp : emps) {
19. System.out.println(emp);
20. }
21. }
22. public static List<Emp> testQuery(){
23. Connection connection = null;
24. Statement statement=null;
25. ResultSet resultSet=null;
26. List<Emp> list =null;
27. try{
28. Class.forName(driver);
29. connection = DriverManager.getConnection(url, user,password);
30. statement = connection.createStatement();
31. String sql="select \* from emp";
32. resultSet = statement.executeQuery(sql);
33. list=new ArrayList<>();
34. while(resultSet.next()){
35. int empno = resultSet.getInt("empno");
36. String ename = resultSet.getString("ename");
37. String job = resultSet.getString("job");
38. int mgr = resultSet.getInt("mgr");
39. Date hiredate = resultSet.getDate("hiredate");
40. double sal= resultSet.getDouble("sal");
41. double comm= resultSet.getDouble("comm");
42. int deptno= resultSet.getInt("deptno");
43. Emp emp =new Emp(empno, ename, job, mgr, hiredate, sal, comm, deptno);
44. list.add(emp);
45. }
46. }catch (Exception e){
47. e.printStackTrace();
48. }finally {
49. if(null != resultSet){
50. try {
51. resultSet.close();
52. } catch (SQLException e) {e.printStackTrace();
53. }
54. }
55. if(null != statement){
56. try {
57. statement.close();
58. } catch (SQLException e) {
59. e.printStackTrace();
60. }
61. }
62. if(null != connection){
63. try {
64. connection.close();
65. } catch (SQLException e) {
66. e.printStackTrace();
67. }
68. }
69. }
70. return list;
71. }
72. }

#### 4\_SQL注入攻击

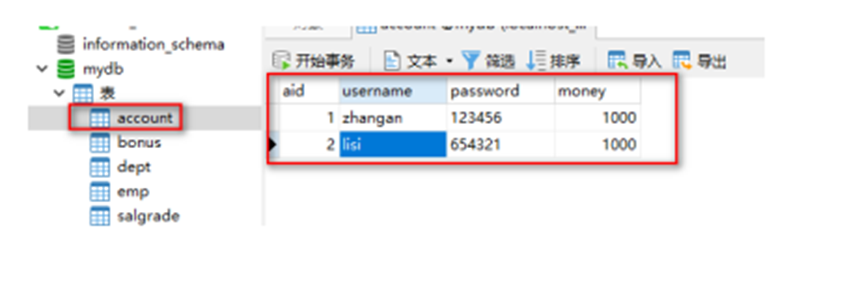
#### Sql注入

SQL注入攻击指的是通过构建特殊的输入作为参数传入Web应用程序，而这些输入大都是SQL语法里的一些组合，通过执行SQL语句进而执行攻击者所要的操作，其主要原因是程序没有细致地过滤用户输入的数据，致使非法数据侵入系统。

以模拟登录为例:在前台输入用户名和密码，后台判断信息是否正确，并给出前台反馈信息，前台输出反馈信息。

具体实现步骤为

 创建数据库表



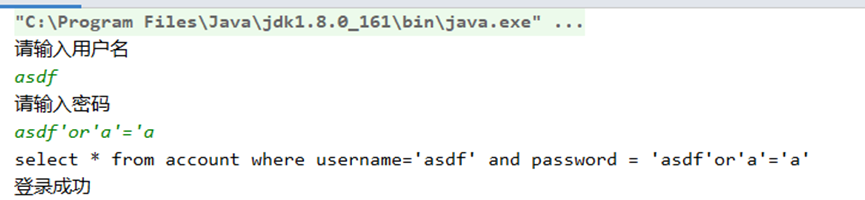
**创建实体类**

**public class** Account **implements** Serializable {  
 **private int aid**;  
 **private** String **username**;  
 **private** String **password**;  
 **private int money**;

**测试代码**

1. package com.msb.test2;
2. import com.msb.entity.Account;
3. import com.msb.entity.Emp;
4. import java.sql.\*;
5. import java.util.ArrayList;
6. import java.util.List;
7. import java.util.Scanner;
8. /\*\*
9. \* @Author: Ma HaiYang
10. \* @Description: MircoMessage:Mark\_7001
11. \*/
12. public class TestInjection {
13. private static String driver ="com.mysql.cj.jdbc.Driver";
14. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
15. private static String user="root";
16. private static String password="root";
17. public static void main(String[] args) {
18. Scanner sc =new Scanner(System.in);
19. System.out.println("请输入用户名");
20. String username=sc.next();
21. System.out.println("请输入密码");
22. String pwd =sc.next();
23. Account account = getAccount(username, pwd);
24. System.out.println(null!= account?"登录成功":"登录失败");
25. sc.close();
26. }
27. public static Account getAccount(String username,String pwd){
28. Connection connection = null;
29. Statement statement=null;
30. ResultSet resultSet=null;
31. Account account =null;
32. try{
33. Class.forName(driver);
34. connection = DriverManager.getConnection(url, user,password);
35. statement = connection.createStatement();
36. String sql="select \* from account where username ='"+username+"' and password ='"+pwd+"'";
37. System.out.println(sql);
38. resultSet = statement.executeQuery(sql);
39. while(resultSet.next()){
40. int aid = resultSet.getInt("aid");
41. String usernamea = resultSet.getString("username");
42. String pwda = resultSet.getString("password");
43. double money = resultSet.getDouble("money");
44. account=new Account(aid,usernamea,pwda,money);
45. System.out.println(account);
46. }
47. }catch (Exception e){
48. e.printStackTrace();
49. }finally {
50. if(null != resultSet){
51. try {
52. resultSet.close();
53. } catch (SQLException e) {e.printStackTrace();
54. }
55. }
56. if(null != statement){
57. try {
58. statement.close();
59. } catch (SQLException e) {
60. e.printStackTrace();
61. }
62. }
63. if(null != connection){
64. try {
65. connection.close();
66. } catch (SQLException e) {
67. e.printStackTrace();
68. }
69. }
70. }
71. return account;
72. }
73. }

测试结果为:



当输入了精心设计的用户名密码后，即使是错误的，也能登录成功。让登录功能形同虚设。这是为什么呢，这就是SQL注入风险，原因在于SQL语句是字符串拼接的。SQL语句中拼接的内容破坏了SQL语句原有的判断逻辑

如何解决呢?使用PreparedStatement预编译语句对象就可以解决掉。

#### 5\_预编译语句对象

     使用预编译语句对象防止注入攻击

1. package com.msb.test2;
2. import com.msb.entity.Account;
3. import java.sql.\*;
4. import java.util.Scanner;
5. /\*\*
6. \* @Author: Ma HaiYang
7. \* @Description: MircoMessage:Mark\_7001
8. \*/
9. public class TestInjection2 {
10. private static String driver ="com.mysql.cj.jdbc.Driver";
11. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
12. private static String user="root";
13. private static String password="root";
14. public static void main(String[] args) {
15. Scanner sc =new Scanner(System.in);
16. System.out.println("请输入用户名");
17. String username=sc.next();
18. System.out.println("请输入密码");
19. String pwd =sc.next();
20. Account account = getAccount(username, pwd);
21. System.out.println(null!= account?"登录成功":"登录失败");
22. sc.close();
23. }
24. public static Account getAccount(String username,String pwd){
25. Connection connection = null;
26. PreparedStatement preparedStatement=null;
27. ResultSet resultSet=null;
28. Account account =null;
29. try{
30. Class.forName(driver);
31. connection = DriverManager.getConnection(url, user,password);
32. /\*
33. \* 1使用PreparedStatement语句对象防止注入攻击
34. \* 2PreparedStatement 可以使用 ? 作为参数的占位符
35. \* 3使用?作为占位符,即使是字符串和日期类型,也不使用单独再添加 ''
36. \* 4connection.createStatement();获得的是普通语句对象 Statement
37. \* 5connection.prepareStatement(sql);可以获得一个预编译语句对象PreparedStatement
38. \* 6如果SQL语句中有?作为参数占位符号,那么要在执行CURD之前先设置参数
39. \* 7通过set\*\*\*(问号的编号,数据) 方法设置参数
40. \* \*/
41. String sql="select \* from account where username = ? and password = ?";
42. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
43. //设置参数
44. preparedStatement.setString(1,username );
45. preparedStatement.setString(2,pwd );
46. //执行CURD
47. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
48. while(resultSet.next()){
49. int aid = resultSet.getInt("aid");
50. String usernamea = resultSet.getString("username");
51. String pwda = resultSet.getString("password");
52. double money = resultSet.getDouble("money");
53. account=new Account(aid,usernamea,pwda,money);
54. System.out.println(account);
55. }
56. }catch (Exception e){
57. e.printStackTrace();
58. }finally {
59. if(null != resultSet){
60. try {
61. resultSet.close();
62. } catch (SQLException e) {e.printStackTrace();
63. }
64. }
65. if(null != preparedStatement){
66. try {
67. preparedStatement.close();
68. } catch (SQLException e) {
69. e.printStackTrace();
70. }
71. }
72. if(null != connection){
73. try {
74. connection.close();
75. } catch (SQLException e) {
76. e.printStackTrace();
77. }
78. }
79. }
80. return account;
81. }
82. }

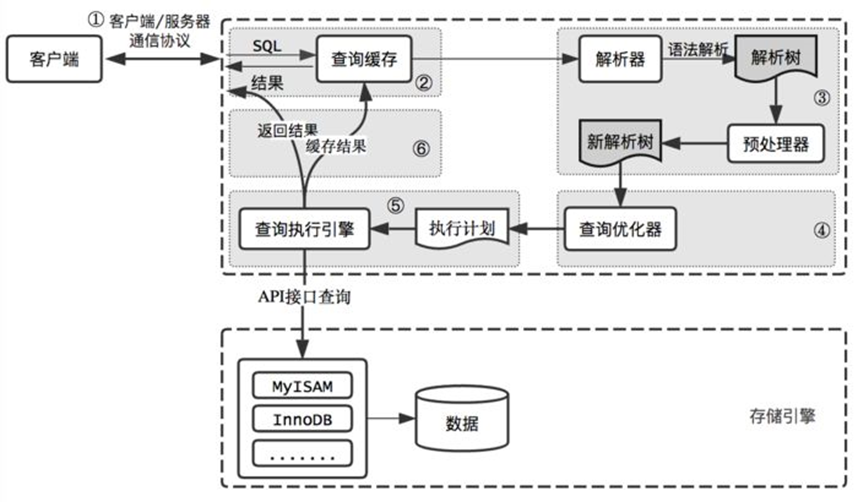
prepareStatment对象在set\*\*\*方法上,会对单引号进行转译处理,也就是说,?中的数据的单引号 ‘ 会被转义成 \’,这样就单引号就不会破坏sql语句的结构,

|  |
| --- |
| SELECT \*   FROM users WHERE userName = ? AND password = ?    preparedStatement.setString(1,"xiaoming");    preparedStatement.setString(2,&apos;anything&apos;   OR &apos;x&apos;=&apos;x&apos;);    会被转义为    SELECT \*   FROM users WHERE userName = &apos;xiaoming&apos;   AND password = &apos;anything\&apos;   OR\&apos;x\&apos;=\&apos;x\&apos;&apos;    而不是    SELECT \*   FROM users WHERE userName = &apos;xiaoming&apos; AND password = &apos;anything&apos; OR &apos;x&apos;=&apos;x&apos; |

说白了就是把值当中的所有单引号给转义了!这就达到了防止sql注入的目的，说白了mysql驱动的PreparedStatement实现类的setString();方法内部做了单引号的转义，而Statement不能防止sql注入，就是因为它没有把单引号做转义，而是简单粗暴的直接拼接字符串，所以达不到防止sql注入的目的。

#### 预编译

当客户端发送一条sql语句给DBMS时,MySQL的执行流程如下图



**sql命令的执行流程如下**

1. 客户端向服务器端发送SQL命令

2. 服务器端连接模块连接并验证

3. 缓存模块解析SQL为Hash并与缓存中Hash表对应。如果有结果直接返回结果，如果没有对应继续向下执行

4. 解析器解析SQL为解析树，如果出现错误，报SQL解析错误。如果正确，向下传递

5. 预处理器对解析树继续处理，处理成新的解析树。

6. 优化器根据开销自动选择最优执行计划，生成执行计划

7. 执行器执行执行计划，访问存储引擎接口

8. 存储引擎访问物理文件并返回结果

9. 如果开启缓存，缓存管理器把结果放入到查询缓存中。

10. 返回结果给客户端

当客户发送一条SQL语句给DBMS后，DBMS总是需要校验SQL语句的语法格式是否正确，然后把SQL语句编译成可执行的函数，最后才是执行SQL语句。其中校验语法，和编译所花的时间可能比执行SQL语句花的时间还要多。

预编译语句PreparedStatement 是java.sql中的一个接口，它是Statement的子接口。通过Statement对象执行SQL语句时，需要将SQL语句发送给DBMS，由DBMS首先进行编译后再执行。预编译语句和Statement不同，在创建PreparedStatement 对象时就指定了SQL语句，该语句立即发送给DBMS进行编译。当该编译语句被执行时，DBMS直接运行编译后的SQL语句，而不需要像其他SQL语句那样首先将其编译。预编译的SQL语句处理性能**稍微**高于普通的传递变量的办法。

例如:我们需要执行多次insert语句，但只是每次插入的值不同，MySQL服务器也是需要每次都去校验SQL语句的语法格式，以及编译，这就浪费了太多的时间。如果使用预编译功能，那么只对SQL语句进行一次语法校验和编译，所以效率要高。

**预编译如何开启?**

我们可以通过设置URL中的参数来控制预编译是否开启

useServerPrepStmts是否开启预编译

cachePrepStmts 是否启用预编译缓存

|  |
| --- |
| "jdbc:mysql://localhost:3306/mydb?\*\*\*\*\*&**useServerPrepStmts=true&cachePrepStmts=true";** |

**值得注意的是,我们的Connector/J 5.0.5及之后useServerPrepStmts默认false,就是默认没有开启预编译,之前默认为true, cachePrepStmts 一直默认为false,需要我们手动设置才可以启用预编译,在开启预编译的同时要同时开启预编译缓存才能带来些许的性能提升**

**Statement和PreparedStatment的关系和区别**

关系：public interface PreparedStatement extends Statement

区别

PreparedStatment安全性高,可以避免SQL注入

PreparedStatment简单不繁琐,不用进行字符串拼接

PreparedStatment性能高，用在执行多个相同数据库DML操作时,可以减少sql语句的编译次数

#### 6\_PrepareStatement完成CURD

1. package com.msb.test3;
2. import com.msb.entity.Emp;
3. import java.sql.\*;
4. import java.util.ArrayList;
5. import java.util.List;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestPreparedSstatement {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
13. private static String user="root";
14. private static String password="root";
15. public static void main(String[] args) {
16. //testAdd();
17. //testUpdate();
18. //testDelete();
19. testQuery();
20. }
21. public static void testAdd(){
22. // 向 Emp表中增加一条数据
23. Connection connection = null;
24. PreparedStatement preparedStatement=null;
25. try{
26. Class.forName(driver);
27. connection = DriverManager.getConnection(url, user,password);
28. String sql="insert into emp values(DEFAULT ,?,?,?,?,?,?,?)";
29. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
30. //设置参数
31. preparedStatement.setString(1,"Mark");
32. preparedStatement.setString(2,"MANAGER" );
33. preparedStatement.setInt(3,7839);
34. preparedStatement.setDate(4,new Date(System.currentTimeMillis()));
35. preparedStatement.setDouble(5,3000.12);
36. preparedStatement.setDouble(6,0.0);
37. preparedStatement.setDouble(7,30);
38. //执行CURD
39. int rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
40. System.out.println(rows);
41. }catch (Exception e){
42. e.printStackTrace();
43. }finally {
44. if(null != preparedStatement){
45. try {
46. preparedStatement.close();
47. } catch (SQLException e) {
48. e.printStackTrace();
49. }
50. }
51. if(null != connection){
52. try {
53. connection.close();
54. } catch (SQLException e) {
55. e.printStackTrace();
56. }
57. }
58. }
59. }
60. public static void testUpdate(){
61. // 根据工号修改员工表中的数据
62. Connection connection = null;
63. PreparedStatement preparedStatement=null;
64. try{
65. Class.forName(driver);
66. connection = DriverManager.getConnection(url, user,password);
67. String sql="update emp set ename =? ,job=? where empno =?";
68. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
69. //设置参数
70. preparedStatement.setString(1,"Jhon");
71. preparedStatement.setString(2,"ANALYST" );
72. preparedStatement.setInt(3,7935);
73. //执行CURD
74. int rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
75. System.out.println(rows);
76. }catch (Exception e){
77. e.printStackTrace();
78. }finally {
79. if(null != preparedStatement){
80. try {
81. preparedStatement.close();
82. } catch (SQLException e) {
83. e.printStackTrace();
84. }
85. }
86. if(null != connection){
87. try {
88. connection.close();
89. } catch (SQLException e) {
90. e.printStackTrace();
91. }
92. }
93. }
94. }
95. public static void testDelete(){
96. // 根据工号删除员工表中的数据
97. Connection connection = null;
98. PreparedStatement preparedStatement=null;
99. try{
100. Class.forName(driver);
101. connection = DriverManager.getConnection(url, user,password);
102. String sql="delete from emp where empno =?";
103. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
104. //设置参数
105. preparedStatement.setInt(1,7935);
106. //执行CURD
107. int rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
108. System.out.println(rows);
109. }catch (Exception e){
110. e.printStackTrace();
111. }finally {
112. if(null != preparedStatement){
113. try {
114. preparedStatement.close();
115. } catch (SQLException e) {
116. e.printStackTrace();
117. }
118. }
119. if(null != connection){
120. try {
121. connection.close();
122. } catch (SQLException e) {
123. e.printStackTrace();
124. }
125. }
126. }
127. }
128. public static void testQuery(){
129. // 查询名字中包含字母A的员工信息
130. Connection connection = null;
131. PreparedStatement preparedStatement=null;
132. ResultSet resultSet=null;
133. List<Emp> list =null;
134. try{
135. Class.forName(driver);
136. connection = DriverManager.getConnection(url, user,password);
137. /\*
138. \* 1使用PreparedStatement语句对象防止注入攻击
139. \* 2PreparedStatement 可以使用 ? 作为参数的占位符
140. \* 3使用?作为占位符,即使是字符串和日期类型,也不使用单独再添加 ''
141. \* 4connection.createStatement();获得的是普通语句对象 Statement
142. \* 5connection.prepareStatement(sql);可以获得一个预编译语句对象PreparedStatement
143. \* 6如果SQL语句中有?作为参数占位符号,那么要在执行CURD之前先设置参数
144. \* 7通过set\*\*\*(问号的编号,数据) 方法设置参数
145. \* \*/
146. String sql="select \* from emp where ename like ? ";
147. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
148. //设置参数
149. preparedStatement.setString(1,"%A%");
150. //执行CURD
151. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
152. list=new ArrayList<Emp>() ;
153. while(resultSet.next()){
154. int empno = resultSet.getInt("empno");
155. String ename = resultSet.getString("ename");
156. String job = resultSet.getString("job");
157. int mgr = resultSet.getInt("mgr");
158. Date hiredate = resultSet.getDate("hiredate");
159. double sal= resultSet.getDouble("sal");
160. double comm= resultSet.getDouble("comm");
161. int deptno= resultSet.getInt("deptno");
162. Emp emp =new Emp(empno, ename, job, mgr, hiredate, sal, comm, deptno);
163. list.add(emp);
164. }
165. }catch (Exception e){
166. e.printStackTrace();
167. }finally {
168. if(null != resultSet){
169. try {
170. resultSet.close();
171. } catch (SQLException e) {
172. e.printStackTrace();
173. }
174. }
175. if(null != preparedStatement){
176. try {
177. preparedStatement.close();
178. } catch (SQLException e) {
179. e.printStackTrace();
180. }
181. }
182. if(null != connection){
183. try {
184. connection.close();
185. } catch (SQLException e) {
186. e.printStackTrace();
187. }
188. }
189. }
190. // 遍历集合
191. for (Emp emp : list) {
192. System.out.println(emp);
193. }
194. }
195. }

#### 7\_批处理

### PreparedStatement批处理

**什么是批处理?**

当我们有多条sql语句需要发送到数据库执行的时候，有两种发送方式，一种是执行一条发送一条sql语句给数据库,另一个种是发送一个sql集合给数据库，也就是发送一个批sql到数据库。普通的执行过程是：每处理一条数据，就访问一次数据库；而批处理是：累积到一定数量，再一次性提交到数据库，减少了与数据库的交互次数，所以效率会大大提高,很显然两者的数据库执行效率是不同的，我们发送批处理sql的时候数据库执行效率要高

**statement语句对象实现批处理有如下问题**  
缺点：采用硬编码效率低，安全性较差。  
原理：硬编码，每次执行时相似SQL都会进行编译

**PreparedStatement+批处理**优点：语句只编译一次，减少编译次数。提高了安全性（阻止了SQL注入）

原理：相似SQL只编译一次，减少编译次数

注意: 需要设置批处理开启&rewriteBatchedStatements=true

1. package com.msb.test4;
2. import java.sql.\*;
3. /\*\*
4. \* @Author: Ma HaiYang
5. \* @Description: MircoMessage:Mark\_7001
6. \*/
7. public class TestBatch {
8. private static String driver ="com.mysql.cj.jdbc.Driver";
9. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true&useServerPrepStmts=true&cachePrepStmts=true&&rewriteBatchedStatements=true";
10. private static String user="root";
11. private static String password="root";
12. public static void main(String[] args) {
13. testAddBatch();
14. }
15. // 定义一个方法,向部门表增加1000条数据
16. public static void testAddBatch(){
17. Connection connection = null;
18. PreparedStatement preparedStatement=null;
19. try{
20. Class.forName(driver);
21. connection = DriverManager.getConnection(url, user,password);
22. String sql="insert into dept values (DEFAULT ,?,?)";
23. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
24. //设置参数
25. for (int i = 1; i <= 10663; i++) {
26. preparedStatement.setString(1, "name");
27. preparedStatement.setString(2, "loc");
28. preparedStatement.addBatch();// 将修改放入一个批次中
29. if(i%1000==0){
30. preparedStatement.executeBatch();
31. preparedStatement.clearBatch();// 清除批处理中的数据
32. }
33. }
34. /\*
35. \* 整数数组中的元素代表执行的结果代号
36. \* SUCCESS\_NO\_INFO -2
37. \* EXECUTE\_FAILED -3
38. \* \*/
39. /\*int[] ints = \*/
40. preparedStatement.executeBatch();
41. preparedStatement.clearBatch();
42. }catch (Exception e){
43. e.printStackTrace();
44. }finally {
45. if(null != preparedStatement){
46. try {
47. preparedStatement.close();
48. } catch (SQLException e) {
49. e.printStackTrace();
50. }
51. }
52. if(null != connection){
53. try {
54. connection.close();
55. } catch (SQLException e) {
56. e.printStackTrace();
57. }
58. }
59. }
60. }
61. }

### 8\_事务及回滚点

### JDBC中使用事务

**事务回顾:**

事务概念:在逻辑上一组不可分割的操作,由多个sql语句组成,多个sql语句要么全都执行成功,要么都不执行. 原子性 一致性 隔离性 持久性

JDBC控制事物主要就是在学习如何让多个数据库操作成为一个整体,实现要么全都执行成功,要么全都不执行

在JDBC中，事务操作是自动提交。一条对数据库的DML(insert、update、delete)代表一项事务操作,操作成功后，系统将自动调用commit()提交，否则自动调用rollback()回滚,在JDBC中，事务操作方法都位于接口java.sql.Connection中,可以通过调用setAutoCommit(false)来禁止自动提交。之后就可以把多个数据库操作的表达式作为一个事务，在操作完成后调用commit()来进行整体提交，倘若其中一个表达式操作失败，都不会执行到commit()，并且将产生响应的异常；此时就可以在异常捕获时调用rollback()进行回滚,回复至数据初始状态.事务开始的边界则不是那么明显了，它会开始于组成当前事务的所有statement中的第一个被执行的时候。事务结束的边界是commit或者rollback方法的调用

**使用事务保证转账安全性**

1. package com.msb.test5;
2. import java.sql.Connection;
3. import java.sql.DriverManager;
4. import java.sql.PreparedStatement;
5. import java.sql.SQLException;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class TestTransaction {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true&useServerPrepStmts=true&cachePrepStmts=true&&rewriteBatchedStatements=true";
13. private static String user="root";
14. private static String password="root";
15. public static void main(String[] args) {
16. testTransaction();
17. }
18. // 定义一个方法,向部门表增加1000条数据
19. public static void testTransaction(){
20. Connection connection = null;
21. PreparedStatement preparedStatement=null;
22. /\*
23. \* JDBC 默认是自动提交事务
24. \* 每条DML都是默认提交事务的,多个preparedStatement.executeUpdate();都会提交一次事务
25. \* 如果想手动控制事务,那么就不能让事务自动提交
26. \* 通过Connection对象控制connection.setAutoCommit(false);
27. \* 如果不设置 默认值为true,自动提交,设置为false之后就是手动提交了
28. \* 无论是否发生回滚,事务最终会一定要提交的 提交我们建议放在finally之中进行提交
29. \* 如果是转账的过程中出现异常了,那么我们就要执行回滚,回滚操作应该方法catch语句块中
30. \*
31. \* \*/
32. try{
33. Class.forName(driver);
34. connection = DriverManager.getConnection(url, user,password);
35. // 设置事务手动提交
36. connection.setAutoCommit(false);
37. String sql="update account set money =money- ? where aid = ?";
38. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
39. // 转出
40. preparedStatement.setDouble(1, 100);
41. preparedStatement.setInt(2, 1);
42. preparedStatement.executeUpdate();
43. // 产生异常
44. //int i =1/0;
45. // 转入
46. preparedStatement.setDouble(1, -100);
47. preparedStatement.setInt(2, 2);
48. preparedStatement.executeUpdate();
49. }catch (Exception e){
50. if(null != connection){
51. try {
52. connection.rollback();// 回滚事务
53. } catch (SQLException ex) {
54. ex.printStackTrace();
55. }
56. }
57. e.printStackTrace();
58. }finally {
59. // 提交事务
60. if(null != connection){
61. try {
62. connection.commit();
63. } catch (SQLException e) {
64. e.printStackTrace();
65. }
66. }
67. if(null != preparedStatement){
68. try {
69. preparedStatement.close();
70. } catch (SQLException e) {
71. e.printStackTrace();
72. }
73. }
74. if(null != connection){
75. try {
76. connection.close();
77. } catch (SQLException e) {
78. e.printStackTrace();
79. }
80. }
81. }
82. }
83. }

**设置回滚点**

1. package com.msb.test5;
2. import java.sql.\*;
3. import java.util.LinkedList;
4. /\*\*
5. \* @Author: Ma HaiYang
6. \* @Description: MircoMessage:Mark\_7001
7. \*/
8. public class TestTransaction2 {
9. private static String driver ="com.mysql.cj.jdbc.Driver";
10. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true&useServerPrepStmts=true&cachePrepStmts=true&&rewriteBatchedStatements=true";
11. private static String user="root";
12. private static String password="root";
13. public static void main(String[] args) {
14. testAddBatch();
15. }
16. // 定义一个方法,向部门表增加1000条数据
17. public static void testAddBatch(){
18. Connection connection = null;
19. PreparedStatement preparedStatement=null;
20. LinkedList<Savepoint> savepoints =new LinkedList<Savepoint>();
21. try{
22. Class.forName(driver);
23. connection = DriverManager.getConnection(url, user,password);
24. connection.setAutoCommit(false);
25. String sql="insert into dept values (DEFAULT ,?,?)";
26. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
27. //设置参数
28. for (int i = 1; i <= 10663; i++) {
29. preparedStatement.setString(1, "name");
30. preparedStatement.setString(2, "loc");
31. preparedStatement.addBatch();// 将修改放入一个批次中
32. if(i%1000==0){
33. preparedStatement.executeBatch();
34. preparedStatement.clearBatch();// 清除批处理中的数据
35. // 设置回滚点
36. Savepoint savepoint = connection.setSavepoint();
37. savepoints.addLast(savepoint);
38. }
39. // 数据在 100001条插入的时候出现异常
40. if(i ==10001){
41. int x =1/0;
42. }
43. }
44. /\*
45. \* 整数数组中的元素代表执行的结果代号
46. \* SUCCESS\_NO\_INFO -2
47. \* EXECUTE\_FAILED -3
48. \* \*/
49. /\*int[] ints = \*/
50. preparedStatement.executeBatch();
51. preparedStatement.clearBatch();
52. }catch (Exception e){
53. if(null != connection){
54. try {
55. //Savepoint sp = savepoints.getLast();
56. Savepoint sp = savepoints.get(4);
57. if(null != sp){
58. // 选择回滚点
59. connection.rollback(sp);// 回滚
60. }
61. } catch (SQLException e2) {
62. e2.printStackTrace();
63. }
64. }
65. e.printStackTrace();
66. }finally {
67. if(null != connection){
68. try {
69. connection.commit();// 提交
70. } catch (SQLException e) {
71. e.printStackTrace();
72. }
73. }
74. if(null != preparedStatement){
75. try {
76. preparedStatement.close();
77. } catch (SQLException e) {
78. e.printStackTrace();
79. }
80. }
81. if(null != connection){
82. try {
83. connection.close();
84. } catch (SQLException e) {
85. e.printStackTrace();
86. }
87. }
88. }
89. }
90. }

#### 9\_JDBCAPI总结\_阅读

### JDBC API总结

**Connection接口**

•       作用：代表数据库连接

|  |  |
| --- | --- |
| **方法摘要** | |
| void | [**close**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#close()) ()      立即释放此   Connection  对象的数据库和 JDBC 资源，而不是等待它们被自动释放。 |
| void | [**commit**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#commit()) ()               使所有上一次提交/回滚后进行的更改成为持久更改，并释放此 Connection  对象当前持有的所有数据库锁。 |
| [Statement](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Statement.html) | [**createStatement**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#createStatement()) ()               创建一个   Statement  对象来将 SQL 语句发送到数据库。 |
| [CallableStatement](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/CallableStatement.html) | [**prepareCall**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#prepareCall(java.lang.String)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  sql)               创建一个   CallableStatement  对象来调用数据库存储过程。 |
| [PreparedStatement](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/PreparedStatement.html) | [**prepareStatement**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#prepareStatement(java.lang.String)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  sql)               创建一个   PreparedStatement  对象来将参数化的 SQL 语句发送到数据库。 |
| [PreparedStatement](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/PreparedStatement.html) | [**prepareStatement**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#prepareStatement(java.lang.String, int)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  sql, int autoGeneratedKeys)              创建一个默认   PreparedStatement  对象，该对象能获取自动生成的键。 |
| void | [**rollback**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#rollback()) ()               取消在当前事务中进行的所有更改，并释放此   Connection  对象当前持有的所有数据库锁。 |
| void | [**setAutoCommit**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html#setAutoCommit(boolean)) (boolean autoCommit)               将此连接的自动提交模式设置为给定状态。 |

**DriverManager类**

作用：管理一组 JDBC 驱动程序的基本服务

应用程序不再需要使用 Class.forName() 显式地加载 JDBC 驱动程序。在调用 getConnection 方法时，DriverManager 会试着从初始化时加载的那些驱动程序以及使用与当前 applet 或应用程序相同的类加载器显式加载的那些驱动程序中查找合适的驱动程序。

|  |  |
| --- | --- |
| **方法摘要** | |
| static [Connection](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html) | [**getConnection**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/DriverManager.html#getConnection(java.lang.String)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  url)               试图建立到给定数据库 URL 的连接。 |
| static [Connection](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html) | [**getConnection**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/DriverManager.html#getConnection(java.lang.String, java.util.Properties)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  url, [Properties](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/util/Properties.html)  info)               试图建立到给定数据库 URL 的连接。 |
| static [Connection](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/Connection.html) | [**getConnection**](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/sql/DriverManager.html#getConnection(java.lang.String, java.lang.String, java.lang.String)) ([String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  url, [String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  user, [String](mk:@MSITStore:C:\Users\Administrator\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html)  password)              试图建立到给定数据库 URL 的连接。 |

**Statement接口**

作用：用于将 SQL 语句发送到数据库中，或理解为执行sql语句

有三种 Statement对象：

Statement：用于执行不带参数的简单SQL语句；

PreparedStatement（从 Statement 继承）：用于执行带或不带参数的预编译SQL语句；

CallableStatement（从PreparedStatement 继承）：用于执行数据库存储过程的调用。

|  |  |
| --- | --- |
| **方法** | **作用** |
| ResultSet   executeQuery(String   sql) | 执行SQL查询并获取到ResultSet对象 |
| int   executeUpdate(String   sql) | 可以执行插入、删除、更新等操作，返回值是执行该操作所影响的行数 |

**PreparedStatement接口**

关系：public interface PreparedStatement extends Statement

区别

PreparedStatment安全性高，可以避免SQL注入

PreparedStatment简单不繁琐，不用进行字符串拼接

PreparedStatment性能高，用在执行多个相同数据库DML操作时

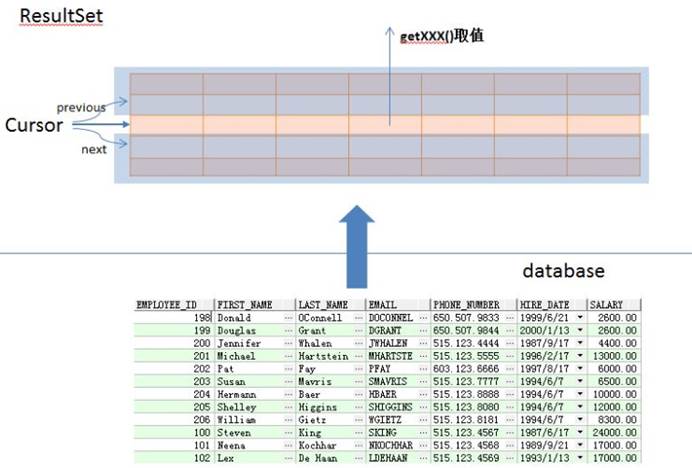
**ResultSet接口**

ResultSet对象是executeQuery()方法的返回值，它被称为结果集，它代表符合SQL语句条件的所有行，并且它通过一套getXXX方法（这些get方法可以访问当前行中的不同列）提供了对这些行中数据的访问。

ResultSet里的数据一行一行排列，每行有多个字段，且有一个记录指针，指针所指的数据行叫做当前数据行，我们只能来操作当前的数据行。我们如果想要取得某一条记录，就要使用ResultSet的next()方法 ,如果我们想要得到ResultSet里的所有记录，就应该使用while循环。

ResultSet对象自动维护指向当前数据行的游标。每调用一次next()方法，游标向下移动一行。

初始状态下记录指针指向第一条记录的前面，通过next()方法指向第一条记录。循环完毕后指向最后一条记录的后面。



|  |  |
| --- | --- |
| 方法名 | 说 明 |
| boolean   next() | 将光标从当前位置向下移动一行 |
| boolean   previous() | 游标从当前位置向上移动一行 |
| void   close() | 关闭ResultSet 对象 |
| int   getInt(int colIndex) | 以int形式获取结果集当前行指定列号值 |
| int   getInt(String colLabel) | 以int形式获取结果集当前行指定列名值 |
| float   getFloat(int colIndex) | 以float形式获取结果集当前行指定列号值 |
| Float   getFloat(String colLabel) | 以float形式获取结果集当前行指定列名值 |
| String   getString(int colIndex) | 以String 形式获取结果集当前行指定列号值 |
| StringgetString(String   colLabel) | 以String形式获取结果集当前行指定列名值 |

### 10\_DAO模式

DAO(Data Access Object)是一个数据访问接口，数据访问：顾名思义就是与数据库打交道。夹在[业务逻辑](https://baike.baidu.com/item/%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91)与数据库资源中间。

在核心[J2EE](https://baike.baidu.com/item/J2EE)模式中是这样介绍DAO模式的：为了建立一个健壮的J2EE应用，应该将所有对[数据源](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%BA%90)的访问操作抽象封装在一个公共[API](https://baike.baidu.com/item/API/10154)中。用程序设计的语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有[事务](https://baike.baidu.com/item/%E4%BA%8B%E5%8A%A1)方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口在逻辑上对应这个特定的数据存储.

简单来说,就是定义一个接口,规定一些增删改查的方法,然后交给实现类去实现, 它介于数据库和业务逻辑代码之间,这样当我们需要操作数据库是,根据接口定义的API去操作数据库就可以了,每个方法都是一个原子性的操作,例如：增加、修改、删除等

Dao模式要求项目必须具备这样几个结构

1实体类:和数据库表格一一对应的类,单独放入一个包中,包名往往是 pojo/entity/bean,要操作的每个表格都应该有对应的实体类

emp > class Emp

dept > class Dept

account > class Account

2DAO 层:定义了对数据要执行那些操作的接口和实现类,包名往往是 dao/mapper,要操作的每个表格都应该有对应的接口和实现类

emp > interface EmpDao >EmpDaoImpl

dept > interface DeptDao> DeptDaoImpl

3Mybatis/Spring JDBCTemplate 中,对DAO层代码进行了封装,代码编写方式会有其他变化

### 项目的搭建

1.创建项目

2.添加jar包

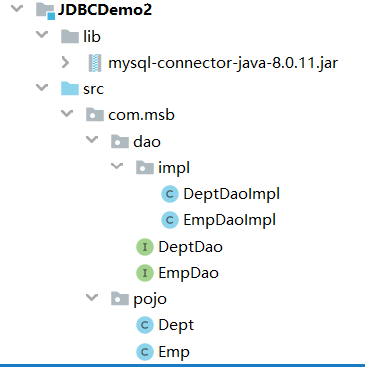
3.创建包

4.创建实体类Emp

5.创建后台的接口EmpDao和实现类EmpDaoImpl

导入各个层级的接口和页面之后的项目

**项目结构截图如下**



**实体类代码**

1. public class Emp implements Serializable {
2. private Integer empno;
3. private String ename;
4. private String job;
5. private Integer mgr;
6. private Date hiredate;
7. private Double sal;
8. private Double comm;
9. private Integer deptno;
10. public class Dept implements Serializable {
11. private Integer deptno;
12. private String dname;
13. private String loc;

**DAO接口代码**

1. package com.msb.dao;
2. import com.msb.pojo.Emp;
3. /\*\*
4. \* @Author: Ma HaiYang
5. \* @Description: MircoMessage:Mark\_7001
6. \*/
7. public interface EmpDao {
8. /\*\*
9. \* 向数据库Emp表中增加一条数据的方法
10. \* @param emp 要增加的数据封装成的Emp类的对象
11. \* @return 增加成功返回大于0 的整数,增加失败返回0
12. \*/
13. int addEmp(Emp emp);
14. /\*\*
15. \* 根据员工编号删除员工信息的方法
16. \* @param empno 要删除的员工编号
17. \* @return 删除成功返回大于0的整数,失败返回0
18. \*/
19. int deleteByEmpno(int empno);
20. }

DAO实现类代码

1. package com.msb.dao.impl;
2. import com.msb.dao.EmpDao;
3. import com.msb.pojo.Emp;
4. import java.sql.\*;
5. /\*\*
6. \* @Author: Ma HaiYang
7. \* @Description: MircoMessage:Mark\_7001
8. \*/
9. public class EmpDaoImpl implements EmpDao {
10. private static String driver ="com.mysql.cj.jdbc.Driver";
11. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
12. private static String user="root";
13. private static String password="root";
14. @Override
15. public int addEmp(Emp emp) {
16. // 向 Emp表中增加一条数据
17. Connection connection = null;
18. PreparedStatement preparedStatement=null;
19. int rows=0;
20. try{
21. Class.forName(driver);
22. connection = DriverManager.getConnection(url, user,password);
23. String sql="insert into emp values(DEFAULT ,?,?,?,?,?,?,?)";
24. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
25. //设置参数
26. preparedStatement.setObject(1,emp.getEname());
27. preparedStatement.setObject(2,emp.getJob() );
28. preparedStatement.setObject(3,emp.getMgr());
29. preparedStatement.setObject(4,emp.getHiredate());
30. preparedStatement.setObject(5,emp.getSal());
31. preparedStatement.setObject(6,emp.getComm());
32. preparedStatement.setObject(7,emp.getDeptno());
33. //执行CURD
34. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
35. }catch (Exception e){
36. e.printStackTrace();
37. }finally {
38. if(null != preparedStatement){
39. try {
40. preparedStatement.close();
41. } catch (SQLException e) {
42. e.printStackTrace();
43. }
44. }
45. if(null != connection){
46. try {
47. connection.close();
48. } catch (SQLException e) {
49. e.printStackTrace();
50. }
51. }
52. }
53. return rows;
54. }
55. @Override
56. public int deleteByEmpno(int empno) {
57. // 向 Emp表中增加一条数据
58. Connection connection = null;
59. PreparedStatement preparedStatement=null;
60. int rows=0;
61. try{
62. Class.forName(driver);
63. connection = DriverManager.getConnection(url, user,password);
64. String sql="delete from emp where empno =?";
65. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
66. //设置参数
67. preparedStatement.setObject(1,empno);
68. //执行CURD
69. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
70. }catch (Exception e){
71. e.printStackTrace();
72. }finally {
73. if(null != preparedStatement){
74. try {
75. preparedStatement.close();
76. } catch (SQLException e) {
77. e.printStackTrace();
78. }
79. }
80. if(null != connection){
81. try {
82. connection.close();
83. } catch (SQLException e) {
84. e.printStackTrace();
85. }
86. }
87. }
88. return rows;
89. }
90. }

### 11\_员工管理系统开发

DAO接口

1. package com.msb.dao;
2. import com.msb.pojo.Emp;
3. import java.util.List;
4. /\*\*
5. \* @Author: Ma HaiYang
6. \* @Description: MircoMessage:Mark\_7001
7. \*/
8. public interface EmpDao {
9. /\*\*
10. \* 向数据库Emp表中增加一条数据的方法
11. \* @param emp 要增加的数据封装成的Emp类的对象
12. \* @return 增加成功返回大于0 的整数,增加失败返回0
13. \*/
14. int addEmp(Emp emp);
15. /\*\*
16. \* 根据员工编号删除员工信息的方法
17. \* @param empno 要删除的员工编号
18. \* @return 删除成功返回大于0的整数,失败返回0
19. \*/
20. int deleteByEmpno(int empno);
21. /\*\*
22. \* 查看数据库表格中所有的员工信息
23. \* @return 所有员工信息封装的一个List<Emp>集合
24. \*/
25. List<Emp> findAll();
26. /\*\*
27. \* 根据员工编号修改员工其他所有字段的方法
28. \* @param emp 员工编号和其他7个字段封装的一个Emp类对象
29. \* @return 修改成功返回大于0的整数,失败返回0
30. \*/
31. int updateEmp(Emp emp);
32. }
33. package com.msb.dao;
34. import com.msb.pojo.Dept;
35. import java.util.List;
36. /\*\*
37. \* @Author: Ma HaiYang
38. \* @Description: MircoMessage:Mark\_7001
39. \*/
40. public interface DeptDao {
41. /\*\*
42. \* 查询全部门的方法
43. \* @return Dept对象封装的List集合
44. \*/
45. List<Dept> findAll();
46. int addDept(Dept dept);
47. }

DAO实现类

1. package com.msb.dao.impl;
2. import com.msb.dao.EmpDao;
3. import com.msb.pojo.Emp;
4. import java.sql.\*;
5. import java.util.ArrayList;
6. import java.util.List;
7. /\*\*
8. \* @Author: Ma HaiYang
9. \* @Description: MircoMessage:Mark\_7001
10. \*/
11. public class EmpDaoImpl implements EmpDao {
12. private static String driver ="com.mysql.cj.jdbc.Driver";
13. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
14. private static String user="root";
15. private static String password="root";
16. @Override
17. public int addEmp(Emp emp) {
18. // 向 Emp表中增加一条数据
19. Connection connection = null;
20. PreparedStatement preparedStatement=null;
21. int rows=0;
22. try{
23. Class.forName(driver);
24. connection = DriverManager.getConnection(url, user,password);
25. String sql="insert into emp values(DEFAULT ,?,?,?,?,?,?,?)";
26. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
27. //设置参数
28. preparedStatement.setObject(1,emp.getEname());
29. preparedStatement.setObject(2,emp.getJob() );
30. preparedStatement.setObject(3,emp.getMgr());
31. preparedStatement.setObject(4,emp.getHiredate());
32. preparedStatement.setObject(5,emp.getSal());
33. preparedStatement.setObject(6,emp.getComm());
34. preparedStatement.setObject(7,emp.getDeptno());
35. //执行CURD
36. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
37. }catch (Exception e){
38. e.printStackTrace();
39. }finally {
40. if(null != preparedStatement){
41. try {
42. preparedStatement.close();
43. } catch (SQLException e) {
44. e.printStackTrace();
45. }
46. }
47. if(null != connection){
48. try {
49. connection.close();
50. } catch (SQLException e) {
51. e.printStackTrace();
52. }
53. }
54. }
55. return rows;
56. }
57. @Override
58. public int deleteByEmpno(int empno) {
59. // 向 Emp表中增加一条数据
60. Connection connection = null;
61. PreparedStatement preparedStatement=null;
62. int rows=0;
63. try{
64. Class.forName(driver);
65. connection = DriverManager.getConnection(url, user,password);
66. String sql="delete from emp where empno =?";
67. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
68. //设置参数
69. preparedStatement.setObject(1,empno);
70. //执行CURD
71. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
72. }catch (Exception e){
73. e.printStackTrace();
74. }finally {
75. if(null != preparedStatement){
76. try {
77. preparedStatement.close();
78. } catch (SQLException e) {
79. e.printStackTrace();
80. }
81. }
82. if(null != connection){
83. try {
84. connection.close();
85. } catch (SQLException e) {
86. e.printStackTrace();
87. }
88. }
89. }
90. return rows;
91. }
92. @Override
93. public List<Emp> findAll() {
94. // 查询名字中包含字母A的员工信息
95. Connection connection = null;
96. PreparedStatement preparedStatement=null;
97. ResultSet resultSet=null;
98. List<Emp> list =null;
99. try{
100. Class.forName(driver);
101. connection = DriverManager.getConnection(url, user,password);
102. String sql="select \* from emp";
103. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
104. //执行CURD
105. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
106. list=new ArrayList<Emp>() ;
107. while(resultSet.next()){
108. int empno = resultSet.getInt("empno");
109. String ename = resultSet.getString("ename");
110. String job = resultSet.getString("job");
111. int mgr = resultSet.getInt("mgr");
112. Date hiredate = resultSet.getDate("hiredate");
113. double sal= resultSet.getDouble("sal");
114. double comm= resultSet.getDouble("comm");
115. int deptno= resultSet.getInt("deptno");
116. Emp emp =new Emp(empno, ename, job, mgr, hiredate, sal, comm, deptno);
117. list.add(emp);
118. }
119. }catch (Exception e){
120. e.printStackTrace();
121. }finally {
122. if(null != resultSet){
123. try {
124. resultSet.close();
125. } catch (SQLException e) {
126. e.printStackTrace();
127. }
128. }
129. if(null != preparedStatement){
130. try {
131. preparedStatement.close();
132. } catch (SQLException e) {
133. e.printStackTrace();
134. }
135. }
136. if(null != connection){
137. try {
138. connection.close();
139. } catch (SQLException e) {
140. e.printStackTrace();
141. }
142. }
143. }
144. return list;
145. }
146. @Override
147. public int updateEmp(Emp emp) {
148. // 向 Emp表中增加一条数据
149. Connection connection = null;
150. PreparedStatement preparedStatement=null;
151. int rows=0;
152. try{
153. Class.forName(driver);
154. connection = DriverManager.getConnection(url, user,password);
155. String sql="update emp set ename =? ,job=?, mgr =?,hiredate =?,sal=?,comm=?,deptno=? where empno =?";
156. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
157. //设置参数
158. preparedStatement.setObject(1,emp.getEname());
159. preparedStatement.setObject(2,emp.getJob() );
160. preparedStatement.setObject(3,emp.getMgr());
161. preparedStatement.setObject(4,emp.getHiredate());
162. preparedStatement.setObject(5,emp.getSal());
163. preparedStatement.setObject(6,emp.getComm());
164. preparedStatement.setObject(7,emp.getDeptno());
165. preparedStatement.setObject(8,emp.getEmpno());
166. //执行CURD
167. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
168. }catch (Exception e){
169. e.printStackTrace();
170. }finally {
171. if(null != preparedStatement){
172. try {
173. preparedStatement.close();
174. } catch (SQLException e) {
175. e.printStackTrace();
176. }
177. }
178. if(null != connection){
179. try {
180. connection.close();
181. } catch (SQLException e) {
182. e.printStackTrace();
183. }
184. }
185. }
186. return rows;
187. }
188. }
189. package com.msb.dao.impl;
190. import com.msb.dao.DeptDao;
191. import com.msb.pojo.Dept;
192. import com.msb.pojo.Emp;
193. import java.sql.\*;
194. import java.util.ArrayList;
195. import java.util.List;
196. /\*\*
197. \* @Author: Ma HaiYang
198. \* @Description: MircoMessage:Mark\_7001
199. \*/
200. public class DeptDaoImpl implements DeptDao {
201. private static String driver ="com.mysql.cj.jdbc.Driver";
202. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
203. private static String user="root";
204. private static String password="root";
205. @Override
206. public List<Dept> findAll() {
207. // 查询名字中包含字母A的员工信息
208. Connection connection = null;
209. PreparedStatement preparedStatement=null;
210. ResultSet resultSet=null;
211. List<Dept> list =null;
212. try{
213. Class.forName(driver);
214. connection = DriverManager.getConnection(url, user,password);
215. String sql="select \* from dept";
216. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
217. //执行CURD
218. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
219. list=new ArrayList<Dept>() ;
220. while(resultSet.next()){
221. int deptno = resultSet.getInt("deptno");
222. String dname = resultSet.getString("dname");
223. String loc = resultSet.getString("loc");
224. Dept dept =new Dept(deptno,dname,loc);
225. list.add(dept);
226. }
227. }catch (Exception e){
228. e.printStackTrace();
229. }finally {
230. if(null != resultSet){
231. try {
232. resultSet.close();
233. } catch (SQLException e) {
234. e.printStackTrace();
235. }
236. }
237. if(null != preparedStatement){
238. try {
239. preparedStatement.close();
240. } catch (SQLException e) {
241. e.printStackTrace();
242. }
243. }
244. if(null != connection){
245. try {
246. connection.close();
247. } catch (SQLException e) {
248. e.printStackTrace();
249. }
250. }
251. }
252. return list;
253. }
254. @Override
255. public int addDept(Dept dept) {
256. // 向 Emp表中增加一条数据
257. Connection connection = null;
258. PreparedStatement preparedStatement=null;
259. int rows=0;
260. try{
261. Class.forName(driver);
262. connection = DriverManager.getConnection(url, user,password);
263. String sql="insert into dept values(?,?,?)";
264. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
265. //设置参数
266. preparedStatement.setObject(1,dept.getDeptno());
267. preparedStatement.setObject(2,dept.getDname());
268. preparedStatement.setObject(3,dept.getLoc() );
269. //执行CURD
270. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
271. }catch (Exception e){
272. e.printStackTrace();
273. }finally {
274. if(null != preparedStatement){
275. try {
276. preparedStatement.close();
277. } catch (SQLException e) {
278. e.printStackTrace();
279. }
280. }
281. if(null != connection){
282. try {
283. connection.close();
284. } catch (SQLException e) {
285. e.printStackTrace();
286. }
287. }
288. }
289. return rows;
290. }
291. }

EmpManageSystem类

1. package com.msb.view;
2. import com.msb.dao.DeptDao;
3. import com.msb.dao.EmpDao;
4. import com.msb.dao.impl.DeptDaoImpl;
5. import com.msb.dao.impl.EmpDaoImpl;
6. import com.msb.pojo.Dept;
7. import com.msb.pojo.Emp;
8. import java.text.ParseException;
9. import java.text.SimpleDateFormat;
10. import java.util.Date;
11. import java.util.List;
12. import java.util.Scanner;
13. /\*\*
14. \* @Author: Ma HaiYang
15. \* @Description: MircoMessage:Mark\_7001
16. \*/
17. public class EmpManageSystem {
18. private static Scanner sc =new Scanner(System.in);
19. private static EmpDao empDao =new EmpDaoImpl();
20. private static DeptDao deptDao=new DeptDaoImpl();
21. private static SimpleDateFormat simpleDateFormat=new SimpleDateFormat("yyyy-MM-dd");;
22. public static void main(String[] args) {
23. while(true){
24. showMenu();
25. System.out.println("请录入选项");
26. int option =sc.nextInt();
27. switch (option){
28. case 1:
29. case1();
30. break;
31. case 2:
32. case2();
33. break;
34. case 3:
35. case3();
36. break;
37. case 4:
38. case4();
39. break;
40. case 5:
41. case5();
42. break;
43. case 6:
44. case6();
45. break;
46. case 7:
47. sc.close();
48. System.exit(0);
49. break;
50. default:
51. System.out.println("请正确输入选项");
52. }
53. }
54. }
55. private static void case1(){
56. List<Emp> emps = empDao.findAll();
57. emps.forEach(System.out::println);
58. }
59. private static void case2(){
60. List<Dept> depts = deptDao.findAll();
61. depts.forEach(System.out::println);
62. }
63. private static void case3(){
64. System.out.println("请输入要删除的员工编号");
65. int empno=sc.nextInt();
66. empDao.deleteByEmpno(empno);
67. }
68. private static void case4(){
69. System.out.println("请输入员工编号");
70. int empno =sc.nextInt();
71. System.out.println("请输入员工姓名");
72. String ename =sc.next();
73. System.out.println("请输入员工职位");
74. String job =sc.next();
75. System.out.println("请输入员工上级");
76. int mgr =sc.nextInt();
77. System.out.println("请输入员工入职日期,格式为yyyy-MM-dd");
78. Date hiredate =null;
79. try {
80. hiredate = simpleDateFormat.parse(sc.next());
81. } catch (ParseException e) {
82. e.printStackTrace();
83. }
84. System.out.println("请输入员工工资");
85. double sal =sc.nextDouble();
86. System.out.println("请输入员工补助");
87. double comm=sc.nextDouble();
88. System.out.println("请输入员工部门号");
89. int deptno =sc.nextInt();
90. Emp emp=new Emp(empno, ename, job, mgr, hiredate, sal, comm,deptno);
91. empDao.updateEmp(emp);
92. }
93. private static void case5(){
94. System.out.println("请输入员工姓名");
95. String ename =sc.next();
96. System.out.println("请输入员工职位");
97. String job =sc.next();
98. System.out.println("请输入员工上级");
99. int mgr =sc.nextInt();
100. System.out.println("请输入员工入职日期,格式为yyyy-MM-dd");
101. Date hiredate =null;
102. try {
103. hiredate = simpleDateFormat.parse(sc.next());
104. } catch (ParseException e) {
105. e.printStackTrace();
106. }
107. System.out.println("请输入员工工资");
108. double sal =sc.nextDouble();
109. System.out.println("请输入员工补助");
110. double comm=sc.nextDouble();
111. System.out.println("请输入员工部门号");
112. int deptno =sc.nextInt();
113. Emp emp=new Emp(null, ename, job, mgr, hiredate, sal, comm,deptno);
114. empDao.addEmp(emp);
115. }
116. private static void case6(){
117. System.out.println("请录入部门号");
118. int deptno =sc.nextInt();
119. System.out.println("请录入部门名称");
120. String dname =sc.next();
121. System.out.println("请录入部门位置");
122. String loc =sc.next();
123. Dept dept =new Dept(deptno,dname,loc);
124. deptDao.addDept(dept);
125. }
126. public static void showMenu(){
127. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
128. System.out.println("\* 1 查看所有员工信息");
129. System.out.println("\* 2 查看所有部门信息");
130. System.out.println("\* 3 根据工号删除员工信息");
131. System.out.println("\* 4 根据工号修改员工信息");
132. System.out.println("\* 5 增加员工信息");
133. System.out.println("\* 6 增加部门信息");
134. System.out.println("\* 7 退出");
135. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
136. }
137. }

### 12\_BaseDao抽取

**BaseDAO代码**

1. package com.msb.dao;
2. import com.msb.pojo.Emp;
3. import java.lang.reflect.Field;
4. import java.sql.\*;
5. import java.util.ArrayList;
6. import java.util.List;
7. /\*\*
8. \* @Author: Ma HaiYang
9. \* @Description: MircoMessage:Mark\_7001
10. \*/
11. public abstract class BaseDao {
12. private static String driver ="com.mysql.cj.jdbc.Driver";
13. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
14. private static String user="root";
15. private static String password="root";
16. public int baseUpdate(String sql,Object ... args){
17. // 向 Emp表中增加一条数据
18. Connection connection = null;
19. PreparedStatement preparedStatement=null;
20. int rows=0;
21. try{
22. Class.forName(driver);
23. connection = DriverManager.getConnection(url, user,password);
24. preparedStatement = connection.prepareStatement(sql);
25. //设置参数
26. for (int i = 0; i <args.length ; i++) {
27. preparedStatement.setObject(i+1, args[i]);
28. }
29. //执行CURD
30. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
31. }catch (Exception e){
32. e.printStackTrace();
33. }finally {
34. if(null != preparedStatement){
35. try {
36. preparedStatement.close();
37. } catch (SQLException e) {
38. e.printStackTrace();
39. }
40. }
41. if(null != connection){
42. try {
43. connection.close();
44. } catch (SQLException e) {
45. e.printStackTrace();
46. }
47. }
48. }
49. return rows;
50. }
51. public List baseQuery(Class clazz,String sql,Object ... args) {
52. // 查询名字中包含字母A的员工信息
53. Connection connection = null;
54. PreparedStatement preparedStatement=null;
55. ResultSet resultSet=null;
56. List list =null;
57. try{
58. Class.forName(driver);
59. connection = DriverManager.getConnection(url, user,password);
60. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
61. //设置参数
62. for (int i = 0; i <args.length ; i++) {
63. preparedStatement.setObject(i+1, args[i]);
64. }
65. //执行CURD
66. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
67. list=new ArrayList() ;
68. // 根据字节码获取所有 的属性
69. Field[] fields = clazz.getDeclaredFields();
70. for (Field field : fields) {
71. field.setAccessible(true);// 设置属性可以 访问
72. }
73. while(resultSet.next()){
74. // 通过反射创建对象
75. Object obj = clazz.newInstance();//默认在通过反射调用对象的空参构造方法
76. for (Field field : fields) {// 临时用Field设置属性
77. String fieldName = field.getName();// empno ename job .... ...
78. Object data = resultSet.getObject(fieldName);
79. field.set(obj,data);
80. }
81. list.add(obj);
82. }
83. }catch (Exception e){
84. e.printStackTrace();
85. }finally {
86. if(null != resultSet){
87. try {
88. resultSet.close();
89. } catch (SQLException e) {
90. e.printStackTrace();
91. }
92. }
93. if(null != preparedStatement){
94. try {
95. preparedStatement.close();
96. } catch (SQLException e) {
97. e.printStackTrace();
98. }
99. }
100. if(null != connection){
101. try {
102. connection.close();
103. } catch (SQLException e) {
104. e.printStackTrace();
105. }
106. }
107. }
108. return list;
109. }
110. }

到实现类代码

1. package com.msb.dao.impl;
2. import com.msb.dao.BaseDao;
3. import com.msb.dao.EmpDao;
4. import com.msb.pojo.Emp;
5. import java.sql.\*;
6. import java.util.ArrayList;
7. import java.util.List;
8. /\*\*
9. \* @Author: Ma HaiYang
10. \* @Description: MircoMessage:Mark\_7001
11. \*/
12. public class EmpDaoImpl extends BaseDao implements EmpDao {
13. @Override
14. public int addEmp(Emp emp) {
15. String sql="insert into emp values(DEFAULT ,?,?,?,?,?,?,?)";
16. return baseUpdate(sql, emp.getEname(),emp.getJob(),emp.getMgr(),emp.getHiredate(),emp.getSal(),emp.getComm(),emp.getDeptno());
17. }
18. @Override
19. public int deleteByEmpno(int empno) {
20. String sql="delete from emp where empno =?";
21. return baseUpdate(sql, empno);
22. }
23. @Override
24. public List<Emp> findAll() {
25. String sql ="select \* from emp";
26. return baseQuery(Emp.class, sql );
27. }
28. @Override
29. public int updateEmp(Emp emp) {
30. String sql="update emp set ename =? ,job=?, mgr =?,hiredate =?,sal=?,comm=?,deptno=? where empno =?";
31. return baseUpdate(sql, emp.getEname(),emp.getJob(),emp.getMgr(),emp.getHiredate(),emp.getSal(),emp.getComm(),emp.getDeptno(),emp.getEmpno());
32. }
33. }
34. package com.msb.dao.impl;
35. import com.msb.dao.BaseDao;
36. import com.msb.dao.DeptDao;
37. import com.msb.pojo.Dept;
38. import com.msb.pojo.Emp;
39. import java.sql.\*;
40. import java.util.ArrayList;
41. import java.util.List;
42. /\*\*
43. \* @Author: Ma HaiYang
44. \* @Description: MircoMessage:Mark\_7001
45. \*/
46. public class DeptDaoImpl extends BaseDao implements DeptDao {
47. @Override
48. public List<Dept> findAll() {
49. String sql="select \* from dept";
50. return baseQuery(Dept.class, sql);
51. }
52. @Override
53. public int addDept(Dept dept) {
54. String sql="insert into dept values(?,?,?)";
55. return baseUpdate(sql, dept.getDeptno(),dept.getDname(),dept.getLoc());
56. }
57. }

### 13\_连接池的使用

**建立数据库连接的两种方式:**

**传统连接方式：**

首先调用Class.forName()方法加载数据库驱动，然后调用DriverManager.getConnection()方法建立连接.

**连接池方式:**

连接池解决方案是在应用程序启动时就预先建立多个数据库连接对象,然后将连接对象保存到连接池中。当客户请求到来时,从池中取出一个连接对象为客户服务。当请求完成时,客户程序调用close()方法,将连接对象放回池中.对于多于连接池中连接数的请求，排队等待。应用程序还可根据连接池中连接的使用率，动态增加或减少池中的连接数。

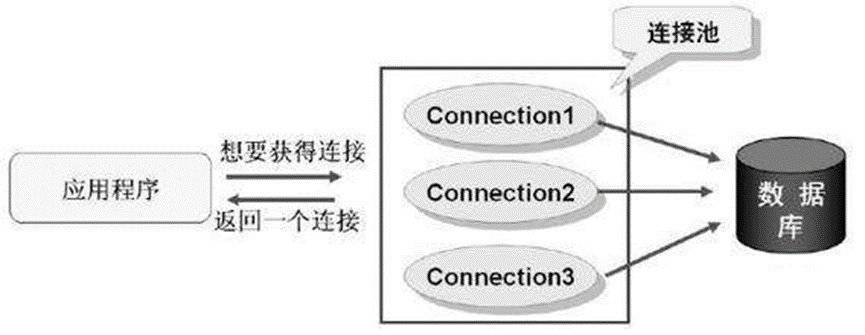
**传统方式存在问题**

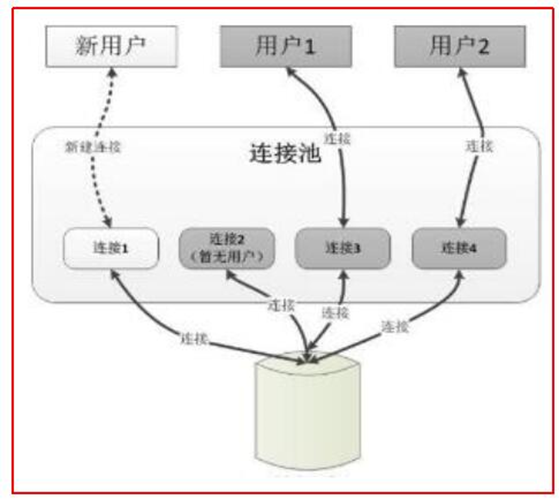
Connection对象在每次执行DML和DQL的过程中都要创建一次,DML和DQL执行完毕后,connection对象都会被销毁. connection对象是可以反复使用的,没有必要每次都创建新的.该对象的创建和销毁都是比较消耗系统资源的,如何实现connection对象的反复使用呢?使用连接池技术实现.

**连接池的优势**

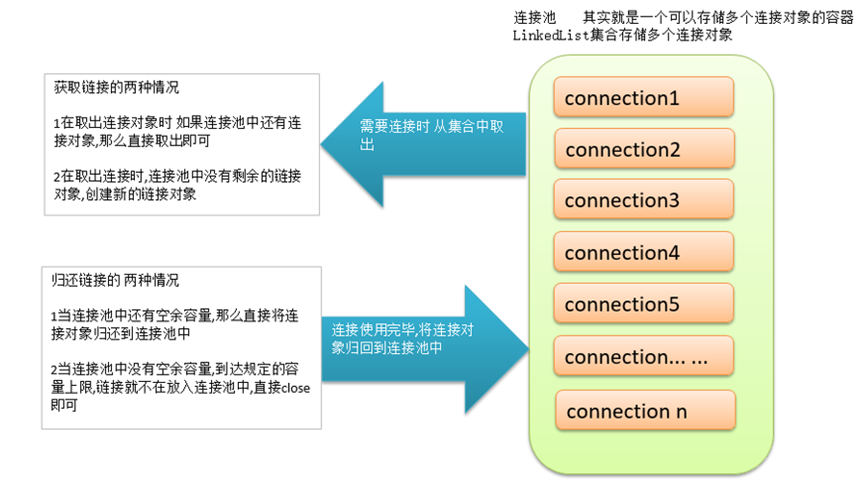
1预先准备一些链接对象,放入连接池中,当多个线程并发执行时,可以避免短时间内一次性大量创建链接对象,减少计算机单位时间内的运算压力,提高程序的响应速度

2实现链接对象的反复使用,可以大大减少链接对象的创建次数,减少资源的消耗





具体实现如下



1定义连接池

1. package com.msb.dao;
2. import java.sql.Connection;
3. import java.sql.DriverManager;
4. import java.sql.SQLException;
5. import java.util.LinkedList;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public class MyConnectionPool {
11. private static String driver ="com.mysql.cj.jdbc.Driver";
12. private static String url="jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true";
13. private static String user="root";
14. private static String password="root";
15. private static int initSize=1;
16. private static int maxSize=1;
17. private static LinkedList<Connection> pool;
18. static{
19. // 加载驱动
20. try {
21. Class.forName(driver);
22. } catch (ClassNotFoundException e) {
23. e.printStackTrace();
24. }
25. // 初始化pool
26. pool=new LinkedList<Connection>();
27. // 创建5个链接对象
28. for (int i = 0; i <initSize ; i++) {
29. Connection connection = initConnection();
30. if(null != connection){
31. pool.add(connection);
32. System.out.println("初始化连接"+connection.hashCode()+"放入连接池");
33. }
34. }
35. }
36. // 私有的初始化一个链接对象的方法
37. private static Connection initConnection(){
38. try {
39. return DriverManager.getConnection(url,user,password);
40. } catch (SQLException e) {
41. e.printStackTrace();
42. }
43. return null;
44. }
45. // 共有的向外界提供链接对象的
46. public static Connection getConnection(){
47. Connection connection =null;
48. if(pool.size()>0){
49. connection= pool.removeFirst();// 移除集合中的第一个元素
50. System.out.println("连接池中还有连接:"+connection.hashCode());
51. }else{
52. connection = initConnection();
53. System.out.println("连接池空,创建新连接:"+connection.hashCode());
54. }
55. return connection;
56. }
57. // 共有的向连接池归还连接对象的方法
58. public static void returnConnection(Connection connection){
59. if(null != connection){
60. try {
61. if(!connection.isClosed()){
62. if(pool.size()<maxSize){
63. try {
64. connection.setAutoCommit(true);// 调整事务状态
65. System.out.println("设置连接:"+connection.hashCode()+"自动提交为true");
66. } catch (SQLException e) {
67. e.printStackTrace();
68. }
69. pool.addLast(connection);
70. System.out.println("连接池未满,归还连接:"+connection.hashCode());
71. }else{
72. try {
73. connection.close();
74. System.out.println("连接池满了,关闭连接:"+connection.hashCode());
75. } catch (SQLException e) {
76. e.printStackTrace();
77. }
78. }
79. }else{
80. System.out.println("连接:"+connection.hashCode()+"已经关闭,无需归还");
81. }
82. } catch (SQLException e) {
83. e.printStackTrace();
84. }
85. }else{
86. System.out.println("传入的连接为null,不可归还");
87. }
88. }
89. }

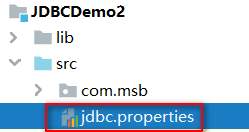
2修改BaseDao

1. package com.msb.dao;
2. import java.lang.reflect.Field;
3. import java.sql.\*;
4. import java.util.ArrayList;
5. import java.util.List;
6. /\*\*
7. \* @Author: Ma HaiYang
8. \* @Description: MircoMessage:Mark\_7001
9. \*/
10. public abstract class BaseDao {
11. public int baseUpdate(String sql,Object ... args){
12. // 向 Emp表中增加一条数据
13. Connection connection = null;
14. PreparedStatement preparedStatement=null;
15. int rows=0;
16. try{
17. connection = MyConnectionPool.getConnection();
18. preparedStatement = connection.prepareStatement(sql);
19. //设置参数
20. for (int i = 0; i <args.length ; i++) {
21. preparedStatement.setObject(i+1, args[i]);
22. }
23. //执行CURD
24. rows =preparedStatement.executeUpdate();// 这里不需要再传入SQL语句
25. }catch (Exception e){
26. e.printStackTrace();
27. }finally {
28. if(null != preparedStatement){
29. try {
30. preparedStatement.close();
31. } catch (SQLException e) {
32. e.printStackTrace();
33. }
34. }
35. MyConnectionPool.returnConnection(connection);
36. }
37. return rows;
38. }
39. public List baseQuery(Class clazz,String sql,Object ... args) {
40. // 查询名字中包含字母A的员工信息
41. Connection connection = null;
42. PreparedStatement preparedStatement=null;
43. ResultSet resultSet=null;
44. List list =null;
45. try{
46. connection = MyConnectionPool.getConnection();
47. preparedStatement = connection.prepareStatement(sql);//这里已经传入SQL语句
48. //设置参数
49. for (int i = 0; i <args.length ; i++) {
50. preparedStatement.setObject(i+1, args[i]);
51. }
52. //执行CURD
53. resultSet = preparedStatement.executeQuery();// 这里不需要再传入SQL语句
54. list=new ArrayList() ;
55. // 根据字节码获取所有 的属性
56. Field[] fields = clazz.getDeclaredFields();
57. for (Field field : fields) {
58. field.setAccessible(true);// 设置属性可以 访问
59. }
60. while(resultSet.next()){
61. // 通过反射创建对象
62. Object obj = clazz.newInstance();//默认在通过反射调用对象的空参构造方法
63. for (Field field : fields) {// 临时用Field设置属性
64. String fieldName = field.getName();// empno ename job .... ...
65. Object data = resultSet.getObject(fieldName);
66. field.set(obj,data);
67. }
68. list.add(obj);
69. }
70. }catch (Exception e){
71. e.printStackTrace();
72. }finally {
73. if(null != resultSet){
74. try {
75. resultSet.close();
76. } catch (SQLException e) {
77. e.printStackTrace();
78. }
79. }
80. if(null != preparedStatement){
81. try {
82. preparedStatement.close();
83. } catch (SQLException e) {
84. e.printStackTrace();
85. }
86. }
87. MyConnectionPool.returnConnection(connection);
88. }
89. return list;
90. }
91. }

配置文件优化参数存储

准备jdbc.properties配置文件,放在src下

1. ## key=value
2. driver=com.mysql.cj.jdbc.Driver
3. url=jdbc:mysql://127.0.0.1:3306/mydb?useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true
4. user=root
5. password=root
6. initSize=1
7. maxSize=1



准备PropertiesUtil工具类

1. package com.msb.util;
2. import java.io.IOException;
3. import java.io.InputStream;
4. import java.util.Properties;
5. /\*\*
6. \* @Author: Ma HaiYang
7. \* @Description: MircoMessage:Mark\_7001
8. \*/
9. public class PropertiesUtil {
10. private Properties properties;
11. public PropertiesUtil(String path){
12. properties=new Properties();
13. InputStream inputStream = this.getClass().getResourceAsStream(path);
14. try {
15. properties.load(inputStream);
16. } catch (IOException e) {
17. e.printStackTrace();
18. }
19. }
20. public String getProperties(String key){
21. return properties.getProperty(key);
22. }
23. }

连接池中代码修改

1. package com.msb.dao;
2. import com.msb.util.PropertiesUtil;
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.SQLException;
6. import java.util.LinkedList;
7. /\*\*
8. \* @Author: Ma HaiYang
9. \* @Description: MircoMessage:Mark\_7001
10. \*/
11. public class MyConnectionPool {
12. private static String driver;
13. private static String url;
14. private static String user;
15. private static String password;
16. private static int initSize;
17. private static int maxSize;
18. private static LinkedList<Connection> pool;
19. static{
20. // 初始化参数
21. PropertiesUtil propertiesUtil=new PropertiesUtil("/jdbc.properties");
22. driver=propertiesUtil.getProperties("driver");
23. url=propertiesUtil.getProperties("url");
24. user=propertiesUtil.getProperties("user");
25. password=propertiesUtil.getProperties("password");
26. initSize=Integer.parseInt(propertiesUtil.getProperties("initSize"));
27. maxSize=Integer.parseInt(propertiesUtil.getProperties("maxSize"));
28. // 加载驱动
29. try {
30. Class.forName(driver);
31. } catch (ClassNotFoundException e) {
32. e.printStackTrace();
33. }
34. // 初始化pool
35. pool=new LinkedList<Connection>();
36. // 创建5个链接对象
37. for (int i = 0; i <initSize ; i++) {
38. Connection connection = initConnection();
39. if(null != connection){
40. pool.add(connection);
41. System.out.println("初始化连接"+connection.hashCode()+"放入连接池");
42. }
43. }
44. }
45. // 私有的初始化一个链接对象的方法
46. private static Connection initConnection(){
47. try {
48. return DriverManager.getConnection(url,user,password);
49. } catch (SQLException e) {
50. e.printStackTrace();
51. }
52. return null;
53. }
54. // 共有的向外界提供链接对象的
55. public static Connection getConnection(){
56. Connection connection =null;
57. if(pool.size()>0){
58. connection= pool.removeFirst();// 移除集合中的第一个元素
59. System.out.println("连接池中还有连接:"+connection.hashCode());
60. }else{
61. connection = initConnection();
62. System.out.println("连接池空,创建新连接:"+connection.hashCode());
63. }
64. return connection;
65. }
66. // 共有的向连接池归还连接对象的方法
67. public static void returnConnection(Connection connection){
68. if(null != connection){
69. try {
70. if(!connection.isClosed()){
71. if(pool.size()<maxSize){
72. try {
73. connection.setAutoCommit(true);// 调整事务状态
74. System.out.println("设置连接:"+connection.hashCode()+"自动提交为true");
75. } catch (SQLException e) {
76. e.printStackTrace();
77. }
78. pool.addLast(connection);
79. System.out.println("连接池未满,归还连接:"+connection.hashCode());
80. }else{
81. try {
82. connection.close();
83. System.out.println("连接池满了,关闭连接:"+connection.hashCode());
84. } catch (SQLException e) {
85. e.printStackTrace();
86. }
87. }
88. }else{
89. System.out.println("连接:"+connection.hashCode()+"已经关闭,无需归还");
90. }
91. } catch (SQLException e) {
92. e.printStackTrace();
93. }
94. }else{
95. System.out.println("传入的连接为null,不可归还");
96. }
97. }
98. }

### 14\_log4j日志框架\_了解

### log4j日志处理

**1)     什么是日志log**

异常信息 登录成功失败的信息 其他重要操作的信息

日志可以记录程序的运行状态,运行信息,用户的一些常用操作.日志可以帮助我们分析程序的运行状态,帮我们分析用户的操作习惯,进而对程序进行改进

**2)     如何记录日志**

方式1：System.out.println(.....) e.printStackTrace();

缺点：不是保存到文件，不能长久存储

方式2：IO流 将System.out.println(.....) e.printStackTrace();写入文件

缺点：操作繁琐,IO流操作容易阻塞线程,日志没有等级,日志的格式不能很好的定制,要想实行编程复杂

方式3：使用现成的日志框架，比如log4j

优点：1长久保存 2有等级3格式可以很好的定制 4代码编写简单

**3)     log4j日志的级别**

FATAL： 指出现非常严重的错误事件，这些错误可能导致应用程序异常中止

ERROR： 指虽有错误，但仍允许应用程序继续运行

WARN： 指运行环境潜藏着危害

INFO： 指报告信息，这些信息在粗粒度级别上突出显示应用程序的进程

DEBUG： 指细粒度信息事件，对于应用程序的调试是最有用的

**4)     使用log4j记录日志**

1.加入jar包 log4j-1.2.8.jar

2.加入属性文件 src 下 log4j.properties

1. log4j.rootLogger=error,logfile
2. log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3. log4j.appender.stdout.Target=System.err
4. log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
5. log4j.appender.logfile=org.apache.log4j.FileAppender
6. log4j.appender.logfile.File=d:/msb.log
7. log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
8. log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %l %F %p %m%n

通过属性文件理解log4j的主要API

Appender 日志目的地 :ConsoleAppender FileAppender

Layout 日志格式化器 ：SimpleLayout PatternLayout

3.代码中记录日志

|  |
| --- |
| //创建一个日志记录器   **private static final** Logger **logger** =   Logger.getLogger(DBUtil.**class**.getName());        //在合适的地方添加日志    **logger**.info(**"正确的读取了属性文件："**+prop);    **logger**.debug(**"正确的关闭了结果集"**);    **logger**.error(**"DML操作错误："**+e); |

**5)     理解日志格式化字符的含义**

%p：输出日志信息的优先级，即DEBUG，INFO，WARN，ERROR，FATAL。

%d：输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，如：%d{yyyy/MM/dd HH:mm:ss,SSS}。

%r：输出自应用程序启动到输出该log信息耗费的毫秒数。

%t：输出产生该日志事件的线程名。

%l：输出日志事件的发生位置，相当于%c.%M(%F:%L)的组合，包括类全名、方法、文件名以及在代码中的行数。例如

test.TestLog4j.main(TestLog4j.java:10)。  
 %c：输出日志信息所属的类目，通常就是所在类的全名。  
 %M：输出产生日志信息的方法名。  
 %F：输出日志消息产生时所在的文件名称。  
 %L:：输出代码中的行号。  
 %m:：输出代码中指定的具体日志信息。  
 %n：输出一个回车换行符，Windows平台为"rn"，Unix平台为"n"。  
 %x：输出和当前线程相关联的NDC(嵌套诊断环境)，尤其用到像java servlets这样的多客户多线程的应用中。  
 %%：输出一个"%"字符。

**6)     使用log4j记录日志 连接池中通过log4j记录日志**

1. package com.msb.dao;
2. import com.msb.util.PropertiesUtil;
3. import org.apache.log4j.Logger;
4. import java.sql.Connection;
5. import java.sql.DriverManager;
6. import java.sql.SQLException;
7. import java.util.LinkedList;
8. /\*\*
9. \* @Author: Ma HaiYang
10. \* @Description: MircoMessage:Mark\_7001
11. \*/
12. public class MyConnectionPool {
13. private static String driver;
14. private static String url;
15. private static String user;
16. private static String password;
17. private static int initSize;
18. private static int maxSize;
19. private static Logger logger;
20. private static LinkedList<Connection> pool;
21. static{
22. logger=Logger.getLogger(MyConnectionPool.class);
23. // 初始化参数
24. PropertiesUtil propertiesUtil=new PropertiesUtil("/jdbc.properties");
25. driver=propertiesUtil.getProperties("driver");
26. url=propertiesUtil.getProperties("url");
27. user=propertiesUtil.getProperties("user");
28. password=propertiesUtil.getProperties("password");
29. initSize=Integer.parseInt(propertiesUtil.getProperties("initSize"));
30. maxSize=Integer.parseInt(propertiesUtil.getProperties("maxSize"));
31. // 加载驱动
32. try {
33. Class.forName(driver);
34. } catch (ClassNotFoundException e) {
35. logger.fatal("找不到数据库驱动类"+driver,e);
36. }
37. // 初始化pool
38. pool=new LinkedList<Connection>();
39. // 创建5个链接对象
40. for (int i = 0; i <initSize ; i++) {
41. Connection connection = initConnection();
42. if(null != connection){
43. pool.add(connection);
44. logger.info("初始化连接"+connection.hashCode()+"放入连接池");
45. }
46. }
47. }
48. // 私有的初始化一个链接对象的方法
49. private static Connection initConnection(){
50. try {
51. return DriverManager.getConnection(url,user,password);
52. } catch (SQLException e) {
53. logger.fatal("初始化连接异常",e);
54. }
55. return null;
56. }
57. // 共有的向外界提供链接对象的
58. public static Connection getConnection(){
59. Connection connection =null;
60. if(pool.size()>0){
61. connection= pool.removeFirst();// 移除集合中的第一个元素
62. logger.info("连接池中还有连接:"+connection.hashCode());
63. }else{
64. connection = initConnection();
65. logger.info("连接池空,创建新连接:"+connection.hashCode());
66. }
67. return connection;
68. }
69. // 共有的向连接池归还连接对象的方法
70. public static void returnConnection(Connection connection){
71. if(null != connection){
72. try {
73. if(!connection.isClosed()){
74. if(pool.size()<maxSize){
75. try {
76. connection.setAutoCommit(true);// 调整事务状态
77. logger.debug("设置连接:"+connection.hashCode()+"自动提交为true");
78. } catch (SQLException e) {
79. e.printStackTrace();
80. }
81. pool.addLast(connection);
82. logger.info("连接池未满,归还连接:"+connection.hashCode());
83. }else{
84. try {
85. connection.close();
86. logger.info("连接池满了,关闭连接:"+connection.hashCode());
87. } catch (SQLException e) {
88. e.printStackTrace();
89. }
90. }
91. }else{
92. logger.info("连接:"+connection.hashCode()+"已经关闭,无需归还");
93. }
94. } catch (SQLException e) {
95. e.printStackTrace();
96. }
97. }else{
98. logger.warn("传入的连接为null,不可归还");
99. }
100. }
101. }

### 15\_三大范式

### ****什么是范式****

必须保证数据库设计的合理性,对数据库设计总结的一些经验性的规范,称之为范式

1.数据库设计关系整个系统的架构，关系到后续开发效率和运行效率

2.数据库的设计主要包含了设计表结构和表之间的联系

如何是合理数据库

1.结构合理

2.冗余较小

3.尽量避免插入删除修改异常

如何才能保证数据库设计水平

1.遵循一定的规则

2.在关系型数据库中这种规则就称为范式

什么是范式（NF= NormalForm）

1.范式是符合某一种设计要求的总结。

2.要想设计一个结构合理的关系型数据库，必须满足一定的范式。

**范式分类**

第一范式:列原子性

第二范式:数据和联合主键完全相关性

第三范式:数据和主键直接相关性

1.Boyce Codd范式=BCNF

2.由Boyce和Codd提出的,

3.比3NF又进了一步

4.通常认为是修正的第三范式.

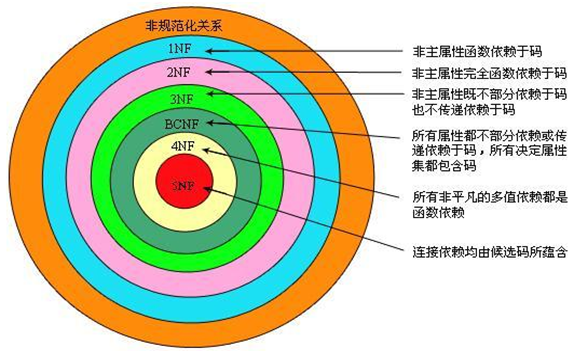
第四范式

第五范式

各个范式是依次嵌套包含的

范式越高，设计质量越高，在现实设计中也越难实现

### 一般数据库设计，只要达到第三范式，即可避免异常的出现



### 第一范式

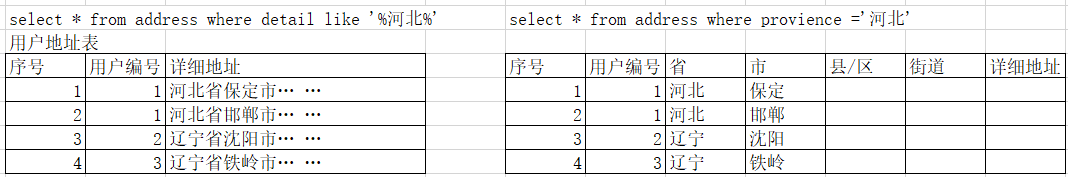
要求

最基本的范式

数据库表每一列都是不可分割基本数据项，同一列中不能有多个值

简单说就是要确保每列保持原子性

第一范式的合理遵循需要根据系统的实际需求来定



示例

用户表（用户名，家庭地址）

用户表（用户名，省，城市，详细地址）

系（系名称，系主任，系高级职称人数）

系（系名称，系主任，系教授人数，系副教授人数）

### 第二范式

要求

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（言）。即在一个数据库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。



示例

学号和课程编号作为联合主键

课程名称只依赖于课程编号，而和学号没有关系

解决

提取出学生表

提取成课程表

提取选课表，存放选课记录

学生表 选课表 课程表



### 第三范式

要求

确保数据表中的每一列数据都和主键直接相关，而不能间接相关

属性不依赖于其他非主属性。

示例1：学生班级表



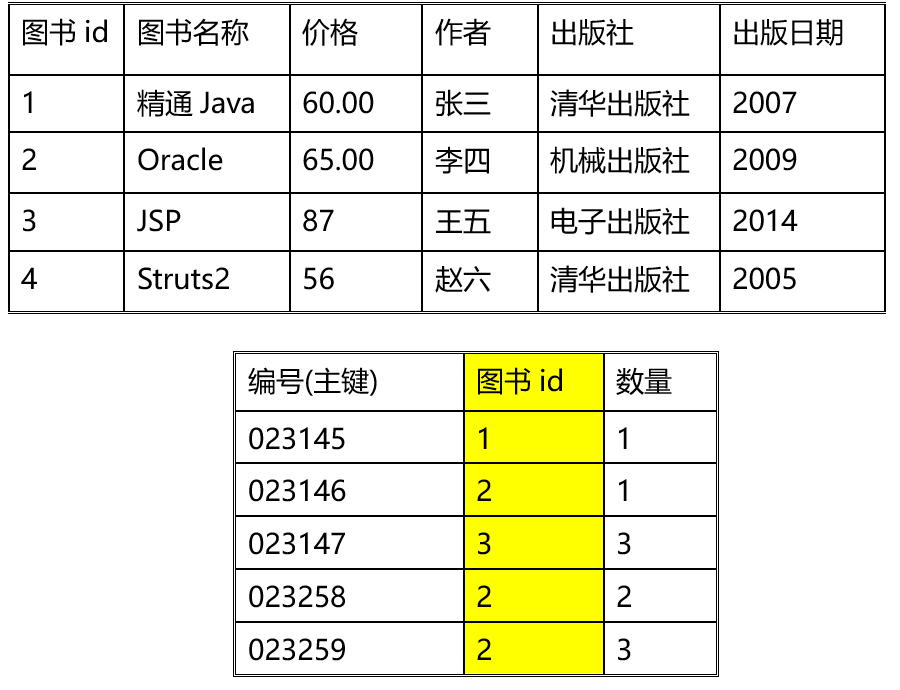
完善之后的方案：



示例2:订单明细表



完善之后的方案：分割成图书表和订单表两种表



### 范式的总结

•       优点

•       结构合理

•       冗余较小

•       尽量避免插入删除修改异常

•       缺点

•       性能降低

•       多表查询比单表查询速度慢

•       数据库的设计应该根据当前情况和需求做出灵活的处理。

•       在实际设计中，要整体遵循范式理论。

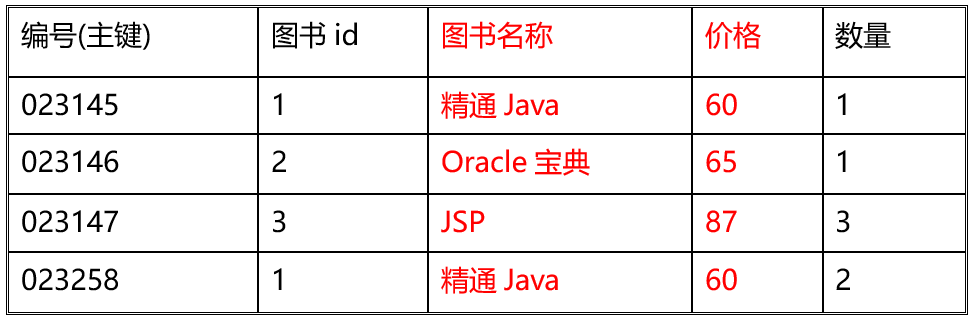
•       如果在某些特定的情况下还死死遵循范式也是不可取的，因为可能降低数据库的效率，此时可以**适当增加冗余而提高性能**。

•       示例：

•       比如经常购物车条目的中除了条目编号，商品编号，商品数量外，可以增加经常使用的商品名称，商品价格等

图书表

订单表中增加冗余列图书名称、价格，以空间换时间。



•       范式是指导数据设计的规范化理论，可以保证数据库设计质量

•       第一范式：字段不能再分

•       第二范式：不存在局部依赖

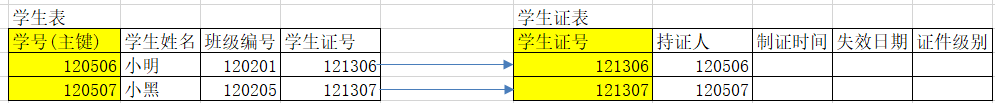
•       第三范式：不含传递依赖（间接依赖）

•       使用范式可以减少冗余，但是会降低性能

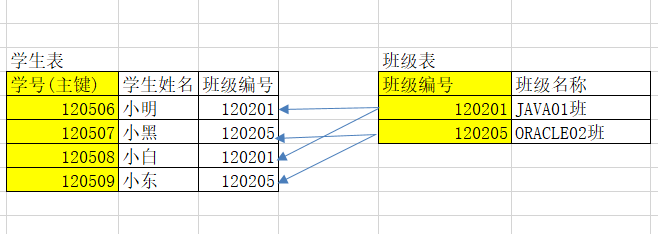
•       特定表的的设计可以违反第三范式，增加冗余提高性能

### 16\_数据之间的三大关系

一对一 A表中的一条数据对应B表中的一条数据



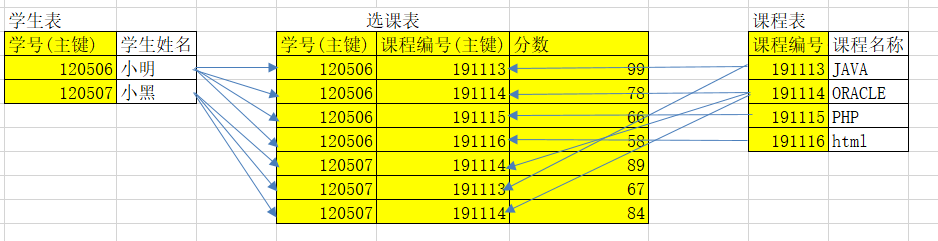
一对多 A表中的一条数据对应B表中的多条数据



多对多 A表中对应B表中多条数据,同样B表中对应A表中多条数据

              多对多需要通过中间表体现关系

              中间表讲多对多的关系转变成两个一对多



**1.主键约束**

主键约束（PRIMARY KEY，缩写PK），是数据库中最重要的一种约束，其作用是约束表中的某个字段可以**唯一标识一条记录**。因此，使用主键约束可以快速查找表中的记录。就像人的身份证、学生的学号等等，**设置为主键的字段取值不能重复（唯一），也不能为空（非空）**，否则无法唯一标识一条记录。

主键可以是单个字段，也可以是多个字段组合。对于单字段主键的添加可使用表级约束，也可以使用列级约束；而对于多字段主键的添加只能使用表级约束。

**2.非空约束**

非空约束（NOT NULL，缩写NK）规定了一张表中指定的某个字段的值不能为空（NULL）。设置了非空约束的字段，在插入的数据为NULL时，数据库会提示错误，导致数据无法插入。

无论是单个字段还是多个字段非空约束的添加只能使用列级约束（非空约束无表级约束）

**为已存在表中的字段添加非空约束**

|  |
| --- |
| alter   table student8 modify stu\_sex varchar(1) not null; |

**使用ALTER TABLE语句删除非空约束**

|  |
| --- |
| alter  table student8 modify stu\_sex varchar(1) null; |

**3. 唯一约束**

唯一约束（UNIQUE，缩写UK）比较简单，它规定了一张表中指定的某个字段的值不能重复，即这一字段的每个值都是唯一的。如果想要某个字段的值不重复，那么就可以为该字段添加为唯一约束。

无论单个字段还是多个字段唯一约束的添加均可使用列级约束和表级约束

**4. 检查约束**

检查约束（CHECK）用来限制某个字段的取值范围，可以定义为列级约束，也可以定义为表级约束。MySQL8开始支持检查约束。

**5. 默认值约束**

默认值约束（DEFAULT）用来规定字段的默认值。如果某个被设置为DEFAULT约束的字段没插入具体值，那么该字段的值将会被默认值填充。

     默认值约束的设置与非空约束一样，也只能使用列级约束。

**6. 字段值自动增加约束**

自增约束（AUTO\_INCREMENT）可以使表中某个字段的值自动增加。一张表中只能有一个自增长字段，并且该字段必须定义了约束（该约束可以是主键约束、唯一约束以及外键约束），如果自增字段没有定义约束，数据库则会提示“Incorrect table definition; there can be only one auto column and it must be defined as a key”错误。

由于自增约束会自动生成唯一的ID，所以自增约束通常会配合主键使用，并且只适用于整数类型。一般情况下，设置为自增约束字段的值会从1开始，每增加一条记录，该字段的值加1。

**为已存在表中的字段添加自增约束**

|  |
| --- |
| /\*创建表student11\*/   create   table student11 (         stu\_id int(10) primary key,         stu\_name varchar(3),         stu\_sex varchar (1)  );  /\*为student11表中的主键字段添加自增约束\*/  alter   table student11 modify stu\_id int(10) auto\_increment; |

**使用ALTER TABLE语句删除自增约束**

|  |
| --- |
| alter   table studen11 modify stu\_id int(10); |