MySQL安装与配置

本节目标

> - 数据库介绍：数据库分类、主流数据库介绍
> - MySQL安装：windows安装、Linux安装
> - SQL分类：DDL、DML、DCL

1 数据库介绍

什么是数据库

存储数据用文件就可以了，为什么还要弄个数据库?

文件保存数据有以下几个缺点：

- 文件的安全性问题
- 文件不利于数据查询和管理
- 文件不利于存储海量数据
- 文件在程序中控制不方便

数据库存储介质：

- 磁盘
- 内存

为了解决上述问题，专家们设计出更加利于管理数据的软件——数据库，它能更有效的管理数据。数据库可以提供远程服务，即通过远程连接来使用数据库，因此也称为数据库服务器。

数据库分类

数据库大体可以分为 **关系型数据库** 和 **非关系型数据库**

- 关系型数据库（RDBMS）：

  是指采用了关系模型来组织数据的数据库。 简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。

  基于标准的SQL，只是内部一些实现有区别。常用的关系型数据库如：

  1. Oracle：甲骨文产品，适合大型项目，适用于做复杂的业务逻辑，如ERP、OA等企业信息系统。收费。
  2. MySQL：属于甲骨文，不适合做复杂的业务。开源免费。
  3. SQL Server：微软的产品，安装部署在windows server上，适用于中大型项目。收费。
- 非关系型数据库：

  （了解）不规定基于SQL实现。现在更多是指NoSQL数据库，如：

  1. 基于键值对（Key-Value）：如 memcached、redis
  2. 基于文档型：如 mongodb
  3. 基于列族：如 hbase
  4. 基于图型：如 neo4j

关系型数据库与非关系型数据库的 **区别**：

	关系型数据库	非关系型数据库
使用SQL	是	不强制要求，一般不基于SQL实现
事务支持	支持	不支持
复杂操作	支持	不支持
海量读写操作	效率低	效率高
基本结构	基于表和列，结构固定	灵活性比较高
使用场景	业务方面的OLTP系统	用于数据的缓存、或基于统计分析的OLAP系统

注：OLTP（On-Line Transaction Processing）是指联机事务处理，OLAP（On-Line Analytical Processing）是指联机分析处理。

2 MySQL服务器安装

统一使用5.7.x的版本，我们这里使用5.7.27

2.1 Windows绿色安装

- 解压MySQL：如解压到D:\Tools\mysql-5.7.27-winx64（注意不要放在中文路径、有空格或特殊字符的路径中）
- 配置环境变量：
  1. 右键 **此电脑** -> **高级系统设置** -> **环境变量** -> 在下面 ***系统变量栏*** 点击 **新建** ->

     > 变量名：MYSQL_HOME
     >
     > 变量值：为MySQL解压的根目录，我这里为D:\Tools\mysql-5.7.27-winx64

  2. 在 ***系统变量栏*** 找到 **Path** 变量并双击：

     Win10中点击 **新建** 以添加新的路径：

     > %MYSQL_HOME%\bin

     Win7中直接添加到Path的最前面：

     > %MYSQL_HOME%\bin;

  > 说明：
  >
  > 需要执行某个指令，如D:\Tools\mysql-5.7.27-winx64\bin\mysqld.exe，需要在cmd中输入全路径，或切换到D:\Tools\mysql-5.7.27-winx64\bin目录下执行mysqld.exe。为方便操作，如在cmd可以直接输入mysqld执行该指令，可以按以上配置：
  >
  > 1. 以上配置 **%MYSQL_HOME%** 代表引用的 **MYSQL_HOME** 环境变量，即D:\Tools\mysql-5.7.27-winx64。
  > 2. 配置在 **Path** 中，表示设置 **%MYSQL_HOME%\bin** 路径为全局路径，全局路径下的指令可以直接执行。

- 配置MySQL初始化文件：
  1. 在MySQL根目录下创建初始化文件my.ini，即D:\Tools\mysql-5.7.27-winx64\my.ini。内容如下：

```
[mysql]
# 设置mysql客户端默认字符集
```

```
default-character-set=utf8

[mysqld]
#设置3306端口
port=3306

# 设置mysql的安装目录
basedir=D:/Tools/mysql-5.7.27-winx64

# 设置mysql数据库的数据的存放目录
datadir=D:/Tools/mysql-5.7.27-winx64/data

# 允许最大连接数
max_connections=200

# 服务端使用的字符集默认为8比特编码的latin1字符集
character-set-server=utf8

# 创建新表时将使用的默认存储引擎
default-storage-engine=innodb
```

2. 将以上 **basedir** 和 **datadir** 后的内容替换成自己的路径。
3. **注意**：需要保存为ANSI编码。方法一：使用记事本打开，保存/另存为，选择ANSI编码。方法二：使用Notpad++打开，点击编码->转为ANSI编码->保存。

- 初始化MySQL

  右键点击cmd，选择 **以管理员身份运行**，执行以下命令

  > mysqld --initialize-insecure

  如果出现找不到mysqld或是mysql命令，这是环境变量没有配置正确。检查环境变量，如果正确，则重新打开cmd，因为cmd会缓存系统变量，没有更新。

  以上初始化操作完成后，可以看到在MySQL根目录下生成了data目录。

  初始化的MySQL会生成超级管理员，账户名 **root** ，密码为空

- 安装MySQL服务：

  如果以前安装过MySQL，会默认安装MySQL的系统服务，点击开始菜单，输入services.msc，进入系统服务界面，搜索是否有mysql服务（可以在英文状态输入mysql快速定位），如果有mysql服务，需要先删除，有以下两种删除方法：

  1. cmd管理员权限打开，输入sc delete mysql
  2. 如果以上命令执行失败，还可以直接使用注册表删除，点击开始菜单，输入regedit，查找HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services中的mysql，将其删除

  在以上mysql服务删除后，可以安装新解压的mysql服务了，在cmd执行：

  > mysqld install

  此时执行`win+r`输入`services.msc`打开服务管理器，可以看到MySQL服务

- 运行/停止MySQL服务端

  cmd执行

  > net start mysql
  >
  > net stop mysql

2.2 Windows中重装MySQL

- 重装相同版本的MySQL很简单，不用真正卸载，只需要删除MySQL根目录下data文件夹，之后按照 2.1 中的内容，从 **初始化MySQL** 内容开始重新执行即可。
- 重装不同版本的MySQL，需要卸载以前安装的MySQL：

  如果以前通过解压方式安装的MySQL，需要重新安装或卸载时，需要如下操作：

  右键点击cmd，选择 **以管理员身份运行**

  ```
  net stop mysql
  mysqld -remove mysql
  ```

  之后重新执行2.1的安装步骤。

3 Mac中常见的安装问题

1. mac系统安装mysql，只需要安装dmg文件就行，安装完毕，打开系统偏好设置，可以看到mysql服务，点击可以查看服务状态。
2. 在mac中安装好mysql后，要在终端使用mysql，需要配置环境变量：
   - 切换到bash shell：

     ```
     chsh -s /bin/bash
     ```

   - 查找mysql目录：

     ```
     find / -name mysql
     ```

   - 新建用户功能设置文件：

     ```
     vim ~/.profile
     ```

     打开的文件中点击i，输入以下内容：

     ```
     export MYSQL_HOME=查找到的mysql目录（一般是/usr/local/mysql）
     export PATH=$MYSQL_HOME:$PATH
     ```

     再点击":wq"保存退出
   - 刷新环境变量：

     ```
     source ~/.profile
     ```

4 客户端连接MySQL服务器

MySQL默认只允许在服务器本机 使用 **root** 用户登录，要开启 **root** 用户的远程登录，在MySQL服务器本机执行：

> mysql -u root -p

要求输入密码，没有设置密码则直接回车

进入MySQL命令行以后，可以看到 **mysql>**

```
-- 使用mysql数据库
use mysql;

-- 更新用户表的root账户，设置为任意ip都可以访问，密码修改为123456
update user set host="%",authentication_string=password('root') where
user="root";

-- 刷新权限
flush privileges;
```

退出

```
quit;
```

5 SQL分类

- DDL数据定义语言，用来维护存储数据的**结构**

  代表指令: `create，drop，alter`

- DML数据操纵语言，用来对**数据**进行操作

  代表指令：`insert，delete，update`

  - DML中又单独分了一个DQL，数据查询语言，代表指令：`select`
- DCL数据控制语言，主要负责权限管理和事务

  代表指令：`grant，revoke，commit`

以上SQL我们主要会在后面学习DDL和DML的操作。

MySQL数据库基础

本节目标

> - 数据库的操作：创建数据库、删除数据库
> - 常用数据类型
> - 表的操作：创建表、删除表

1. 数据库的操作

1.1 显示当前的数据库

```
SHOW DATABASES;
```

1.2 创建数据库

语法：

```
CREATE DATABASE [IF NOT EXISTS] db_name [create_specification [,
create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
    [DEFAULT] COLLATE collation_name
```

说明：

- 大写的表示关键字
- [] 是可选项
- CHARACTER SET: 指定数据库采用的字符集
- COLLATE: 指定数据库字符集的校验规则

示例：

- 创建名为 `db_test1` 的数据库

  ```
  CREATE DATABASE db_test1;
  ```

> 说明：当我们创建数据库没有指定字符集和校验规则时，系统使用默认字符集：utf8，校验规则是：utf8_general_ci

- 如果系统没有 `db_test2` 的数据库，则创建一个名叫 `db_test2` 的数据库，如果有则不创建

  ```
  CREATE DATABASE IF NOT EXISTS db_test2;
  ```

- 如果系统没有 `db_test` 的数据库，则创建一个使用utf8mb4字符集的 `db_test` 数据库，如果有则不创建

  ```
  CREATE DATABASE IF NOT EXISTS db_test CHARACTER SET utf8mb4;
  ```

> 说明：MySQL的utf8编码不是真正的utf8，没有包含某些复杂的中文字符。MySQL真正的utf8是使用utf8mb4，建议大家都使用utf8mb4

1.3 使用数据库

```
use 数据库名;
```

1.4 删除数据库

语法：

```
DROP DATABASE [IF EXISTS] db_name;
```

说明:

- 数据库删除以后，内部看不到对应的数据库，里边的表和数据全部被删除

```
drop database if exists db_test1;
drop database if exists db_test2;
```

2. 常用数据类型

2.1 数值类型：

分为整型和浮点型：

数据类型	大小	说明	对应java类型
BIT[(M)]	M指定位数，默认为1	二进制数，M范围从1到64，存储数值范围从0到2^M-1	常用Boolean对应BIT，此时默认是1位，即只能存0和1
TINYINT	1字节		Byte
SMALLINT	2字节		Short
INT	4字节		Integer
BIGINT	8字节		Long
FLOAT(M, D)	4字节	单精度，M指定长度，D指定小数位数。会发生精度丢失	Float
DOUBLE(M, D)	8字节		Double
DECIMAL(M, D)	M/D最大值+2	双精度，M指定长度，D表示小数点位数。精确数值	BigDecimal
NUMERIC(M, D)	M/D最大值+2	和DECIMAL一样	BigDecimal

扩展资料

> 数值类型可以指定为无符号（unsigned），表示不取负数。

> 1字节（bytes）= 8bit。
>
> 对于整型类型的范围：
>
> 1. 有符号范围：-2^（类型字节数*8-1）到2^（类型字节数*8-1）-1，如int是4字节，就是-2^31到2^31-1
> 2. 无符号范围：0到2^（类型字节数*8）-1，如int就是2^32-1
>
> 尽量不使用unsigned，对于int类型可能存放不下的数据，int unsigned同样可能存放不下，与其如此，还不如设计时，将int类型提升为bigint类型。

2.2 字符串类型

数据类型	大小	说明	对应java类型
VARCHAR (SIZE)	0-65,535字节	可变长度字符串	String
TEXT	0-65,535字节	长文本数据	String
MEDIUMTEXT	0-16 777 215字节	中等长度文本数据	String
BLOB	0-65,535字节	二进制形式的长文本数据	byte[]

2.3 日期类型

数据类型	大小	说明	对应java类型
DATETIME	8字节	范围从1000到9999年，不会进行时区的检索及转换。	java.util.Date、java.sql.Timestamp
TIMESTAMP	4字节	范围从1970到2038年，自动检索当前时区并进行转换。	java.util.Date、java.sql.Timestamp

3. 表的操作

需要操作数据库中的表时，需要先使用该数据库：

```
use db_test;
```

3.1 查看表结构

```
desc 表名;
```

示例：

3.2 创建表

语法：

```
CREATE TABLE table_name (
    field1 datatype,
    field2 datatype,
    field3 datatype
);
```

可以使用comment增加字段说明。

示例：

```
create table stu_test (
   id int,
   name varchar(20) comment '姓名',
   password varchar(50) comment '密码',
   age int,
   sex varchar(1),
   birthday timestamp,
   amout decimal(13,2),
   resume text
);
```

3.4 删除表

语法格式：

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name] ...
```

示例：

```
-- 删除 stu_test 表
drop table stu_test;
-- 如果存在 stu_test 表，则删除 stu_test 表
drop table if exists stu_test;
```

6. 内容重点总结

- 操作数据库：

```
-- 显示
show databases;
-- 创建
create database xxx;
-- 使用
use xxx;
-- 删除
drop database xxx;
```

- 常用数据类型：

  INT：整型

  DECIMAL(M, D)：浮点数类型

  VARCHAR(SIZE)：字符串类型

  TIMESTAMP：日期类型

- 操作表：

```
-- 查看
show 表;

-- 创建
create table 表名(
  字段1 类型1,
  字段2 类型2,
  ...
);

-- 删除
drop talbe 表名;
```

7. 课后作业

- 有一个商店的数据，记录客户及购物情况，有以下三个表组成：
  - 商品goods(商品编号goods_id，商品名goods_name, 单价unitprice, 商品类别category, 供应商provider)
  - 客户customer(客户号customer_id,姓名name,住址address,邮箱email,性别sex，身份证card_id)
  - 购买purchase(购买订单号order_id,客户号customer_id,商品号goods_id,购买数量nums)
- SQL:

```
-- 创建数据库
create database if not exists bit32mall
default character set utf8 ;

-- 选择数据库
use bit32mall;
```

```sql
-- 创建数据库表
-- 商品
create table if not exists goods
(
    goods_id  int comment '商品编号',
    goods_name varchar(32) comment '商品名称',
    unitprice  int comment '单价，单位分',
    category  varchar(12) comment '商品分类',
    provider  varchar(64) comment '供应商名称'
);

-- 客户
create table if not exists customer
(
    customer_id  int comment '客户编号',
    name varchar(32) comment '客户姓名',
    address  varchar(256) comment '客户地址',
    email  varchar(64) comment '电子邮箱',
    sex bit comment '性别',
    card_id varchar(18) comment '身份证'
);

-- 购买
create table if not exists purchase
(
    order_id  int comment '订单号',
    customer_id int comment '客户编号',
    goods_id  int  comment '商品编号',
    nums  int comment '购买数量'
);
```

MySQL表的增删改查（基础）

本节目标：

> - CRUD : Create, Retrieve，Update，Delete
> - 新增数据
> - 查询数据
> - 修改数据
> - 删除数据

1. CRUD

- 注释：在SQL中可以使用“--空格+描述”来表示注释说明
- `CRUD`即增加(Create)、查询(Retrieve)、更新(Update)、删除(Delete)四个单词的首字母缩写。

2. 新增（Create）

语法：

```
INSERT [INTO] table_name
    [(column [, column] ...)]
    VALUES (value_list) [, (value_list)] ...

value_list: value, [, value] ...
```

案例：

```
-- 创建一张学生表
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT,
    sn INT comment '学号',
    name VARCHAR(20) comment '姓名',
    qq_mail VARCHAR(20) comment 'QQ邮箱'
);
```

2.1 单行数据 + 全列插入

```
-- 插入两条记录，value_list 数量必须和定义表的列的数量及顺序一致
INSERT INTO student VALUES (100, 10000, '唐三藏', NULL);
INSERT INTO student VALUES (101, 10001, '孙悟空', '11111');
```

2.2 多行数据 + 指定列插入

```
-- 插入两条记录，value_list 数量必须和指定列数量及顺序一致
INSERT INTO student (id, sn, name) VALUES
    (102, 20001, '曹孟德'),
    (103, 20002, '孙仲谋');
```

3. 查询（Retrieve）

语法：

```
SELECT
    [DISTINCT] {* | {column [, column] ...}
    [FROM table_name]
    [WHERE ...]
    [ORDER BY column [ASC | DESC], ...]
    LIMIT ...
```

案例：

```
-- 创建考试成绩表
DROP TABLE IF EXISTS exam_result;
CREATE TABLE exam_result (
    id INT,
    name VARCHAR(20),
    chinese DECIMAL(3,1),
    math DECIMAL(3,1),
    english DECIMAL(3,1)
);

-- 插入测试数据
INSERT INTO exam_result (id,name, chinese, math, english) VALUES
    (1,'唐三藏', 67, 98, 56),
    (2,'孙悟空', 87.5, 78, 77),
    (3,'猪悟能', 88, 98.5, 90),
    (4,'曹孟德', 82, 84, 67),
    (5,'刘玄德', 55.5, 85, 45),
    (6,'孙权', 70, 73, 78.5),
    (7,'宋公明', 75, 65, 30);
```

3.1 全列查询

```
-- 通常情况下不建议使用 * 进行全列查询
-- 1. 查询的列越多，意味着需要传输的数据量越大；
-- 2. 可能会影响到索引的使用。（索引待后面课程讲解）
SELECT * FROM exam_result;
```

3.2 指定列查询

```
-- 指定列的顺序不需要按定义表的顺序来
SELECT id, name, english FROM exam_result;
```

3.3 查询字段为表达式

```
-- 表达式不包含字段
SELECT id, name, 10 FROM exam_result;
-- 表达式包含一个字段
SELECT id, name, english + 10 FROM exam_result;
-- 表达式包含多个字段
SELECT id, name, chinese + math + english FROM exam_result;
```

3.4 别名

为查询结果中的列指定别名，表示返回的结果集中，以别名作为该列的名称，语法：

```
SELECT column [AS] alias_name [...] FROM table_name;
```

```
-- 结果集中，表头的列名=别名
SELECT id, name, chinese + math + english 总分 FROM exam_result;
```

3.5 去重：DISTINCT

使用DISTINCT关键字对某列数据进行去重：

```
-- 98 分重复了
SELECT math FROM exam_result;
+--------+
| math |
+--------+
|     98 |
|     78 |
|     98 |
|     84 |
|     85 |
|     73 |
|     65 |
+--------+
7 rows in set (0.00 sec)
```

```
-- 去重结果
SELECT DISTINCT math FROM exam_result;
+--------+
| math |
+--------+
|     98 |
|     78 |
|     84 |
|     85 |
|     73 |
|     65 |
+--------+
6 rows in set (0.00 sec)
```

3.6 排序：ORDER BY

语法：

```
-- ASC 为升序（从小到大）
-- DESC 为降序（从大到小）
-- 默认为 ASC
SELECT ... FROM table_name [WHERE ...]
    ORDER BY column [ASC|DESC], [...];
```

1. 没有 ORDER BY 子句的查询，返回的顺序是未定义的，永远不要依赖这个顺序
2. NULL 数据排序，视为比任何值都小，升序出现在最上面，降序出现在最下面

```
-- 查询同学姓名和 qq_mail，按 qq_mail 排序显示
SELECT name, qq_mail FROM student ORDER BY qq_mail;
SELECT name, qq_mail FROM student  ORDER BY qq_mail DESC;
```

3. 使用**表达式**及**别名**排序

```
-- 查询同学及总分，由高到低
SELECT name, chinese + english + math FROM exam_result
    ORDER BY chinese + english + math DESC;

SELECT name, chinese + english + math total FROM exam_result
    ORDER BY total DESC;
```

4. 可以对多个字段进行排序，排序优先级随书写顺序

```
-- 查询同学各门成绩，依次按 数学降序，英语升序，语文升序的方式显示
SELECT name, math, english, chinese FROM exam_result
    ORDER BY math DESC, english, chinese;
```

3.7 条件查询：WHERE

比较运算符：

运算符	说明
>, >=, <, <=	大于，大于等于，小于，小于等于
=	等于，NULL 不安全，例如 NULL = NULL 的结果是 NULL
<=>	等于，NULL 安全，例如 NULL <=> NULL 的结果是 TRUE(1)
!=, <>	不等于
BETWEEN a0 AND a1	范围匹配，[a0, a1]，如果 a0 <= value <= a1，返回 TRUE(1)
IN (option, ...)	如果是 option 中的任意一个，返回 TRUE(1)
IS NULL	是 NULL
IS NOT NULL	不是 NULL
LIKE	模糊匹配。% 表示任意多个（包括 0 个）任意字符；_ 表示任意一个字符

逻辑运算符：

运算符	说明
AND	多个条件必须都为 TRUE(1)，结果才是 TRUE(1)
OR	任意一个条件为 TRUE(1), 结果为 TRUE(1)
NOT	条件为 TRUE(1)，结果为 FALSE(0)

注:

1. WHERE条件可以使用表达式，但不能使用别名。
2. AND的优先级高于OR，在同时使用时，需要使用小括号()包裹优先执行的部分

案例:

- 基本查询:

```
-- 查询英语不及格的同学及英语成绩 ( < 60 )
SELECT name, english FROM exam_result WHERE english < 60;

-- 查询语文成绩好于英语成绩的同学
SELECT name, chinese, english FROM exam_result WHERE chinese > english;

-- 查询总分在 200 分以下的同学
SELECT name, chinese + math + english 总分 FROM exam_result
    WHERE chinese + math + english < 200;
```

- AND与OR:

```
-- 查询语文成绩大于80分，且英语成绩大于80分的同学
SELECT * FROM exam_result WHERE chinese > 80 and english > 80;

-- 查询语文成绩大于80分，或英语成绩大于80分的同学
SELECT * FROM exam_result WHERE chinese > 80 or english > 80;

-- 观察AND 和 OR 的优先级:
SELECT * FROM exam_result WHERE chinese > 80 or math>70 and english > 70;
SELECT * FROM exam_result WHERE (chinese > 80 or math>70) and english > 70;
```

- 范围查询:

1. BETWEEN ... AND ...

```
-- 查询语文成绩在 [80, 90] 分的同学及语文成绩
SELECT name, chinese FROM exam_result WHERE chinese BETWEEN 80 AND 90;

-- 使用 AND 也可以实现
SELECT name, chinese FROM exam_result WHERE chinese >= 80 AND chinese
<= 90;
```

2. IN

```sql
-- 查询数学成绩是 58 或者 59 或者 98 或者 99 分的同学及数学成绩
SELECT name, math FROM exam_result WHERE math IN (58, 59, 98, 99);

-- 使用 OR 也可以实现
SELECT name, math FROM exam_result WHERE math = 58 OR math = 59 OR math
= 98 OR math = 99;
```

- 模糊查询：LIKE

```sql
-- % 匹配任意多个（包括 0 个）字符
SELECT name FROM exam_result WHERE name LIKE '孙%';-- 匹配到孙悟空、孙权
-- _ 匹配严格的一个任意字符
SELECT name FROM exam_result WHERE name LIKE '孙_';-- 匹配到孙权
```

- NULL 的查询：IS [NOT] NULL

```sql
-- 查询 qq_mail 已知的同学姓名
SELECT name, qq_mail FROM student WHERE qq_mail IS NOT NULL;

-- 查询 qq_mail 未知的同学姓名
SELECT name, qq_mail FROM student WHERE qq_mail IS NULL;
```

3.8 分页查询：LIMIT

语法：

```sql
-- 起始下标为 0

-- 从 0 开始，筛选 n 条结果
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT n;
-- 从 s 开始，筛选 n 条结果
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT s, n;
-- 从 s 开始，筛选 n 条结果，比第二种用法更明确，建议使用
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT n OFFSET s;
```

案例：按 id 进行分页，每页 3 条记录，分别显示 第 1、2、3 页

```sql
-- 第 1 页
SELECT id, name, math, english, chinese FROM exam_result ORDER BY id LIMIT 3
OFFSET 0;
-- 第 2 页
SELECT id, name, math, english, chinese FROM exam_result ORDER BY id LIMIT 3
OFFSET 3;
-- 第 3 页，如果结果不足 3 个，不会有影响
SELECT id, name, math, english, chinese FROM exam_result ORDER BY id LIMIT 3
OFFSET 6;
```

4. 修改（Update）

语法：

```sql
UPDATE table_name SET column = expr [, column = expr ...]
    [WHERE ...] [ORDER BY ...] [LIMIT ...]
```

案例:

```
-- 将孙悟空同学的数学成绩变更为 80 分
UPDATE exam_result SET math = 80 WHERE name = '孙悟空';
-- 将曹孟德同学的数学成绩变更为 60 分，语文成绩变更为 70 分
UPDATE exam_result SET math = 60, chinese = 70 WHERE name = '曹孟德';
-- 将总成绩倒数前三的 3 位同学的数学成绩加上 30 分
UPDATE exam_result SET math = math + 30 ORDER BY chinese + math + english LIMIT
3;
-- 将所有同学的语文成绩更新为原来的 2 倍
UPDATE exam_result SET chinese = chinese * 2;
```

5. 删除（Delete）

语法:

```
DELETE FROM  table_name [WHERE ...] [ORDER BY ...] [LIMIT ...]
```

案例:

```
-- 删除孙悟空同学的考试成绩
DELETE FROM exam_result WHERE name = '孙悟空';

-- 删除整张表数据
-- 准备测试表
DROP TABLE IF EXISTS for_delete;
CREATE TABLE for_delete (
    id INT,
    name VARCHAR(20)
);
-- 插入测试数据
INSERT INTO for_delete (name) VALUES ('A'), ('B'), ('C');
-- 删除整表数据
DELETE FROM for_delete;
```

6. 内容重点总结

- 新增:

  ```
  -- 单行插入
  insert into 表(字段1, ..., 字段N) values (value1, ..., value N);

  -- 多行插入
  insert into 表(字段1, ..., 字段N) values
  (value1, ...),
  (value2, ...),
  (value3, ...);
  ```

- 查询

  ```
  -- 全列查询
  select * from 表
  -- 指定列查询
  ```

```
select 字段1,字段2... from 表
-- 查询表达式字段
select 字段1+100,字段2+字段3 from 表
-- 别名
select 字段1 别名1, 字段2 别名2 from 表
-- 去重DISTINCT
select distinct 字段 from 表
-- 排序ORDER BY
select * from 表 order by 排序字段
-- 条件查询WHERE:
-- (1)比较运算符 (2)BETWEEN ... AND ... (3)IN (4)IS NULL (5)LIKE (6)AND (7)OR
(8)NOT
select * from 表 where 条件
```

- 修改

```
update 表 set 字段1=value1, 字段2=value2... where 条件
```

- 删除

```
delete from 表 where 条件
```

7. 课后作业

-

MySQL表的增删改查（进阶）

本节目标：

> - 数据库约束
> - 表的关系
> - 新增：
> - 删除
> - 修改
> - 查询

1. 数据库约束

1.1 约束类型

- **NOT NULL** - 指示某列不能存储 NULL 值。
- **UNIQUE** - 保证某列的每行必须有唯一的值。
- **DEFAULT** - 规定没有给列赋值时的默认值。
- **PRIMARY KEY** - NOT NULL 和 UNIQUE 的结合。确保某列（或两个列多个列的结合）有唯一标识，有助于更容易更快速地找到表中的一个特定的记录。
- **FOREIGN KEY** - 保证一个表中的数据匹配另一个表中的值的参照完整性。
- **CHECK** - 保证列中的值符合指定的条件。对于MySQL数据库，对CHECK子句进行分析，但是忽略CHECK子句。

1.2 NULL约束

创建表时，可以指定某列不为空：

```
-- 重新设置学生表结构
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT NOT NULL,
    sn INT,
    name VARCHAR(20),
    qq_mail VARCHAR(20)
);
```

1.3 UNIQUE：唯一约束

指定sn列为唯一的、不重复的：

```
-- 重新设置学生表结构
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT NOT NULL,
    sn INT UNIQUE,
    name VARCHAR(20),
    qq_mail VARCHAR(20)
);
```

1.4 DEFAULT：默认值约束

指定插入数据时，name列为空，默认值unkown：

```
-- 重新设置学生表结构
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT NOT NULL,
    sn INT UNIQUE,
    name VARCHAR(20) DEFAULT 'unkown',
    qq_mail VARCHAR(20)
);
```

1.5 PRIMARY KEY：主键约束

指定id列为主键：

```
-- 重新设置学生表结构
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT NOT NULL PRIMARY KEY,
    sn INT UNIQUE,
    name VARCHAR(20) DEFAULT 'unkown',
    qq_mail VARCHAR(20)
);
```

对于整数类型的主键，常配搭自增长auto_increment来使用。插入数据对应字段不给值时，使用最大值+1。

```
-- 主键是 NOT NULL 和 UNIQUE 的结合，可以不用 NOT NULL
id INT PRIMARY KEY auto_increment,
```

1.6 FOREIGN KEY：外键约束

外键用于关联其他表的**主键**或**唯一键**，语法：

```
foreign key (字段名) references 主表(列)
```

案例：

- 创建班级表classes，id为主键：

  ```
  -- 创建班级表，有使用MySQL关键字作为字段时，需要使用``来标识
  DROP TABLE IF EXISTS classes;
  CREATE TABLE classes (
      id INT PRIMARY KEY auto_increment,
      name VARCHAR(20),
      `desc` VARCHAR(100)
  );
  ```

- 创建学生表student，一个学生对应一个班级，一个班级对应多个学生。使用id为主键，classes_id为外键，关联班级表id

```sql
-- 重新设置学生表结构
DROP TABLE IF EXISTS student;
CREATE TABLE student (
    id INT PRIMARY KEY auto_increment,
    sn INT UNIQUE,
    name VARCHAR(20) DEFAULT 'unkown',
    qq_mail VARCHAR(20),
    classes_id int,
    FOREIGN KEY (classes_id) REFERENCES classes(id)
);
```

1.7 CHECK约束（了解）

MySQL使用时不报错，但忽略该约束：

```sql
drop table if exists test_user;
create table test_user (
   id int,
   name varchar(20),
   sex varchar(1),
   check (sex ='男' or sex='女')
);
```

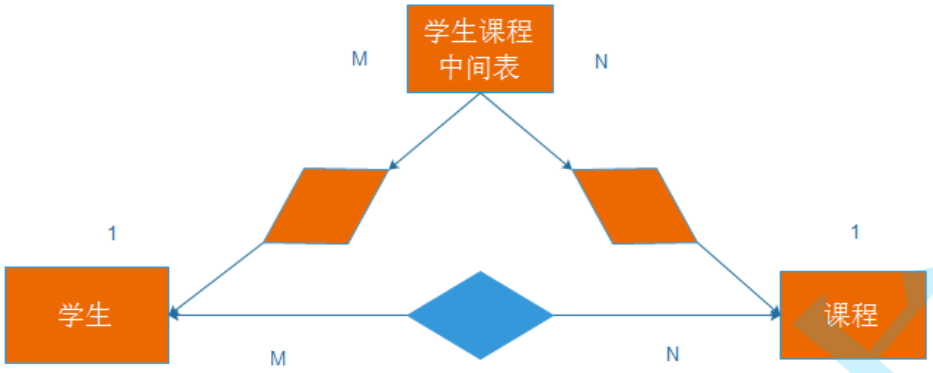
2. 表的设计

三大范式：

2.1 一对一

2.2 一对多

2.3 多对多



- 创建课程表

```
-- 创建课程表
DROP TABLE IF EXISTS course;
CREATE TABLE course (
    id INT PRIMARY KEY auto_increment,
    name VARCHAR(20)
);
```

- 创建学生课程中间表，考试成绩表

```
-- 创建课程学生中间表：考试成绩表
DROP TABLE IF EXISTS score;
CREATE TABLE score (
    id INT PRIMARY KEY auto_increment,
    score DECIMAL(3, 1),
    student_id int,
    course_id int,
    FOREIGN KEY (student_id) REFERENCES student(id),
    FOREIGN KEY (course_id) REFERENCES course(id)
);
```

3. 新增

插入查询结果

语法：

```
INSERT INTO table_name [(column [, column ...])] SELECT ...
```

案例：创建一张用户表，设计有name姓名、email邮箱、sex性别、mobile手机号字段。需要把已有的学生数据复制进来，可以复制的字段为name、qq_mail

```sql
-- 创建用户表
DROP TABLE IF EXISTS test_user;
CREATE TABLE test_user (
    id INT primary key auto_increment,
    name VARCHAR(20) comment '姓名',
    age INT comment '年龄',
    email VARCHAR(20) comment '邮箱',
    sex varchar(1) comment '性别',
    mobile varchar(20) comment '手机号'
);

-- 将学生表中的所有数据复制到用户表
insert into test_user(name, email) select name, qq_mail from student;
```

4. 查询

4.1 聚合查询

4.1.1 聚合函数

常见的统计总数、计算平局值等操作，可以使用聚合函数来实现，常见的聚合函数有：

函数	说明
COUNT([DISTINCT] expr)	返回查询到的数据的 数量
SUM([DISTINCT] expr)	返回查询到的数据的 总和，不是数字没有意义
AVG([DISTINCT] expr)	返回查询到的数据的 平均值，不是数字没有意义
MAX([DISTINCT] expr)	返回查询到的数据的 最大值，不是数字没有意义
MIN([DISTINCT] expr)	返回查询到的数据的 最小值，不是数字没有意义

案例：

- COUNT

```sql
-- 统计班级共有多少同学
SELECT COUNT(*) FROM student;
SELECT COUNT(0) FROM student;

-- 统计班级收集的 qq_mail 有多少个，qq_mail 为 NULL 的数据不会计入结果
SELECT COUNT(qq_mail) FROM student;
```

- SUM

```sql
-- 统计数学成绩总分
SELECT SUM(math) FROM exam_result;

-- 不及格 < 60 的总分，没有结果，返回 NULL
SELECT SUM(math) FROM exam_result WHERE math < 60;
```

- AVG

```
-- 统计平均总分
SELECT AVG(chinese + math + english) 平均总分 FROM exam_result;
```

- MAX

```
-- 返回英语最高分
SELECT MAX(english) FROM exam_result;
```

- MIN

```
-- 返回 > 70 分以上的数学最低分
SELECT MIN(math) FROM exam_result WHERE math > 70;
```

4.1.2 GROUP BY子句

SELECT 中使用 GROUP BY 子句可以对指定列进行分组查询。需要满足：使用 GROUP BY 进行分组查询时，SELECT 指定的字段必须是“分组依据字段”，其他字段若想出现在SELECT 中则必须包含在聚合函数中。

```
select column1, sum(column2), .. from table group by column1,column3;
```

案例：

- 准备测试表及数据：职员表，有id（主键）、name（姓名）、role（角色）、salary（薪水）

```
create table emp(
    id int primary key auto_increment,
    name varchar(20) not null,
    role varchar(20) not null,
    salary numeric(11,2)
);

insert into emp(name, role, salary) values
('马云','服务员', 1000.20),
('马化腾','游戏陪玩', 2000.99),
('孙悟空','游戏角色', 999.11),
('猪无能','游戏角色', 333.5),
('沙和尚','游戏角色', 700.33),
('隔壁老王','董事长', 12000.66);
```

- 查询每个角色的最高工资、最低工资和平均工资

```
select role,max(salary),min(salary),avg(salary) from emp group by role;
```

4.1.3 HAVING

GROUP BY 子句进行分组以后，需要对分组结果再进行条件过滤时，不能使用 WHERE 语句，而需要用 HAVING

- 显示平均工资低于1500的角色和它的平均工资

```
select role,max(salary),min(salary),avg(salary) from emp group by role
having avg(salary)<1500;
```

4.2 联合查询

实际开发中往往数据来自不同的表，所以需要多表联合查询。多表查询是对多张表的数据取笛卡尔积：

aaa

a1	a2
a	aa
b	bb

bbb

b1	b2
1	11
2	22

a	aa	1	11
a	aa	2	22
b	bb	1	11
b	bb	2	22

注意：关联查询可以对关联表使用别名。

初始化测试数据：

```
insert into classes(name, `desc`) values
('计算机系2019级1班', '学习了计算机原理、C和Java语言、数据结构和算法'),
('中文系2019级3班','学习了中国传统文学'),
('自动化2019级5班','学习了机械自动化');

insert into student(sn, name, qq_mail, classes_id) values
('09982','黑旋风李逵','xuanfeng@qq.com',1),
('00835','菩提老祖',null,1),
('00391','白素贞',null,1),
('00031','许仙','xuxian@qq.com',1),
('00054','不想毕业',null,1),
('51234','好好说话','say@qq.com',2),
('83223','tellme',null,2),
('09527','老外学中文','foreigner@qq.com',2);

insert into course(name) values
('Java'),('中国传统文化'),('计算机原理'),('语文'),('高阶数学'),('英文');

insert into score(score, student_id, course_id) values
-- 黑旋风李逵
(70.5, 1, 1),(98.5, 1, 3),(33, 1, 5),(98, 1, 6),
-- 菩提老祖
(60, 2, 1),(59.5, 2, 5),
-- 白素贞
(33, 3, 1),(68, 3, 3),(99, 3, 5),
-- 许仙
(67, 4, 1),(23, 4, 3),(56, 4, 5),(72, 4, 6),
-- 不想毕业
(81, 5, 1),(37, 5, 5),
```

```
-- 好好说话
(56, 6, 2),(43, 6, 4),(79, 6, 6),
-- tellme
(80, 7, 2),(92, 7, 6);
```

4.2.1 内连接

语法：

```
select 字段 from 表1 别名1 [inner] join 表2 别名2 on 连接条件 and 其他条件;
select 字段 from 表1 别名1,表2 别名2 where 连接条件 and 其他条件;
```

案例：

（1）查询“许仙”同学的 成绩

```
select sco.score from student stu inner join score sco on stu.id=sco.student_id
and stu.name='许仙';
-- 或者
select sco.score from student stu, score sco where stu.id=sco.student_id and
stu.name='许仙';
```

（2）查询所有同学的总成绩，及同学的个人信息：

```
-- 成绩表对学生表是多对1关系，查询总成绩是根据成绩表的同学id来进行分组的
SELECT
    stu.sn,
    stu.NAME,
    stu.qq_mail,
    sum( sco.score )
FROM
    student stu
    JOIN score sco ON stu.id = sco.student_id
GROUP BY
    sco.student_id;
```

（3）查询所有同学的成绩，及同学的个人信息：

```
-- 查询出来的都是有成绩的同学，“老外学中文”同学 没有显示
select * from student stu join score sco on stu.id=sco.student_id;

-- 学生表、成绩表、课程表3张表关联查询
SELECT
    stu.id,
    stu.sn,
    stu.NAME,
    stu.qq_mail,
    sco.score,
    sco.course_id,
    cou.NAME
FROM
    student stu
    JOIN score sco ON stu.id = sco.student_id
    JOIN course cou ON sco.course_id = cou.id
ORDER BY
```

```
    stu.id;
```

4.2.2 外连接

外连接分为左外连接和右外连接。如果联合查询，左侧的表完全显示我们就说是左外连接；右侧的表完全显示我们就说是右外连接。

语法：

```
-- 左外连接，表1完全显示
select 字段名  from 表名1 left join 表名2 on 连接条件;

-- 右外连接，表2完全显示
select 字段 from 表名1 right join 表名2 on  连接条件;
```

案例：查询所有同学的成绩，及同学的个人信息，如果该同学没有成绩，也需要显示

```
-- “老外学中文”同学 没有考试成绩，也显示出来了
select * from student stu left join score sco on stu.id=sco.student_id;
-- 对应的右外连接为：
select * from  score sco right join student stu on stu.id=sco.student_id;

-- 学生表、成绩表、课程表3张表关联查询
SELECT
    stu.id,
    stu.sn,
    stu.NAME,
    stu.qq_mail,
    sco.score,
    sco.course_id,
    cou.NAME
FROM
    student stu
    LEFT JOIN score sco ON stu.id = sco.student_id
    LEFT JOIN course cou ON sco.course_id = cou.id
ORDER BY
    stu.id;
```

4.2.3 自连接

自连接是指在同一张表连接自身进行查询。

案例：

显示所有“计算机原理”成绩比“Java”成绩高的成绩信息

```
-- 先查询“计算机原理”和“Java”课程的id
select id,name from course where name='Java' or name='计算机原理';

-- 再查询成绩表中，“计算机原理”成绩比“Java”成绩 好的信息
SELECT
    s1.*
FROM
    score s1,
    score s2
WHERE
    s1.student_id = s2.student_id
```

```
    AND s1.score < s2.score
    AND s1.course_id = 1
    AND s2.course_id = 3;

-- 也可以使用join on 语句来进行自连接查询
SELECT
    s1.*
FROM
    score s1
    JOIN score s2 ON s1.student_id = s2.student_id
    AND s1.score < s2.score
    AND s1.course_id = 1
    AND s2.course_id = 3;
```

以上查询只显示了成绩信息，并且是分布执行的。要显示学生及成绩信息，并在一条语句显示：

```
SELECT
    stu.*,
    s1.score Java,
    s2.score 计算机原理
FROM
    score s1
    JOIN score s2 ON s1.student_id = s2.student_id
    JOIN student stu ON s1.student_id = stu.id
    JOIN course c1 ON s1.course_id = c1.id
    JOIN course c2 ON s2.course_id = c2.id
    AND s1.score < s2.score
    AND c1.NAME = 'Java'
    AND c2.NAME = '计算机原理';
```

4.2.4 子查询

子查询是指嵌入在其他sql语句中的select语句，也叫嵌套查询

- 单行子查询：返回一行记录的子查询

  查询与“不想毕业” 同学的同班同学：

  ```
  select * from student where classes_id=(select classes_id from student where
  name='不想毕业');
  ```

- 多行子查询：返回多行记录的子查询

  案例：查询“语文”或“英文”课程的成绩信息

  1. **[NOT] IN**关键字：

     ```
     -- 使用IN
     select * from score where course_id in (select id from course where
     name='语文' or name='英文');

     -- 使用 NOT IN
     select * from score where course_id not in (select id from course where
     name!='语文' and name!='英文');
     ```

     可以使用多列包含：

```
-- 插入重复的分数：score，student_id，course_id列重复
insert into score(score, student_id, course_id) values
-- 黑旋风李逵
(70.5, 1, 1),(98.5, 1, 3),
-- 菩提老祖
(60, 2, 1);

-- 查询重复的分数
SELECT
    *
FROM
    score
WHERE
    ( score, student_id, course_id ) IN ( SELECT score, student_id,
course_id FROM score GROUP BY score, student_id, course_id HAVING
count( 0 ) > 1 );
```

2. **[NOT] EXISTS**关键字：

```
 -- 使用 EXISTS
select * from score sco where exists (select sco.id from course cou
where (name='语文' or name='英文') and cou.id = sco.course_id);

-- 使用 NOT EXISTS
select * from score sco where not exists (select sco.id from course cou
where (name!='语文' and name!='英文') and cou.id = sco.course_id);
```

- 在from子句中使用子查询：子查询语句出现在from子句中。这里要用到数据查询的技巧，把一个子查询当做一个临时表使用。

  查询所有比“中文系2019级3班”平均分高的成绩信息：

```
-- 获取“中文系2019级3班”的平均分，将其看作临时表
SELECT
    avg( sco.score ) score
FROM
    score sco
    JOIN student stu ON sco.student_id = stu.id
    JOIN classes cls ON stu.classes_id = cls.id
WHERE
    cls.NAME = '中文系2019级3班';
```

  查询成绩表中，比以上临时表平均分高的成绩：

```
SELECT
    *
FROM
    score sco,
    (
    SELECT
        avg( sco.score ) score
    FROM
        score sco
        JOIN student stu ON sco.student_id = stu.id
        JOIN classes cls ON stu.classes_id = cls.id
    WHERE
```

```
        cls.NAME = '中文系2019级3班'
    ) tmp
WHERE
    sco.score > tmp.score;
```

4.2.5 合并查询

在实际应用中，为了合并多个select的执行结果，可以使用集合操作符 union，union all。使用UNION和UNION ALL时，前后查询的结果集中，字段需要一致。

- union

该操作符用于取得两个结果集的并集。当使用该操作符时，会自动去掉结果集中的重复行。

案例：查询id小于3，或者名字为“英文”的课程：

```
select * from course where id<3
union
select * from course where name='英文';

-- 或者使用or来实现
select * from course where id<3 or name='英文';
```

- union all

该操作符用于取得两个结果集的并集。当使用该操作符时，不会去掉结果集中的重复行。

案例：查询id小于3，或者名字为“Java”的课程

```
-- 可以看到结果集中出现重复数据Java
select * from course where id<3
union all
select * from course where name='英文';
```

5. 内容重点总结

- 数据库约束

约束类型	说明	示例

约束类型	说明	示例
NULL约束	使用NOT NULL指定列不为空	name varchar(20) not null,
UNIQUE唯一约束	指定列为唯一的、不重复的	name varchar(20) unique,
DEFAULT默认值约束	指定列为空时的默认值	age int default 20,
主键约束	NOT NULL 和 UNIQUE 的结合	id int primary key,
外键约束	关联其他表的**主键**或**唯一键**	foreign key (字段名) references 主表(列)
CHECK约束（了解）	保证列中的值符合指定的条件	check (sex ='男' or sex='女')

- 表的关系
  1. 一对一：
  2. 一对多：
  3. 多对多：需要创建中间表来映射两张表的关系
- 新增：

```
INSERT INTO table_name [(column [, column ...])] SELECT ...
```

- 查询
  1. 聚合函数：MAX、MIN、AVG、COUNT、SUM
  2. 分组查询：GROUP BY... HAVING ...
  3. 内连接：

```
select ... from 表1，表2 where 条件
-- inner可以缺省
select ... from 表1 join 表2 on 条件 where 其他条件
```

  4. 外连接：

```
 select ... from 表1 left/right join 表2 on 条件 where 其他条件
```

  5. 自连接：

```
 select ... from 表1，表1 where 条件
select ... from 表1 join 表1 on 条件
```

```
6. 子查询：

   ```sql
 -- 单行子查询
   select ... from 表1 where 字段1 = (select ... from ...);
```
```

```
  -- [NOT] IN
select ... from 表1 where 字段1 in (select ... from ...);

  -- [NOT] EXISTS
  select ... from 表1 where exists (select ... from ... where 条件);

  -- 临时表：form子句中的子查询
  select ... from 表1,  (select ... from ...) as tmp where 条件
```

7. 合并查询：

```
-- UNION：去除重复数据
select ... from ... where 条件
 union
select ... from ... where 条件

-- UNION ALL：不去重
 select ... from ... where 条件
union all
select ... from ... where 条件

-- 使用UNION和UNION ALL时，前后查询的结果集中，字段需要一致
```

SQL查询中各个关键字的执行先后顺序： from > on> join > where > group by > with > having > select > distinct > order by > limit

6. 课堂作业

- 设计图书管理系统，包含学生和图书信息，且图书可以进行分类，学生可以在一个时间范围内借阅图书，并在这个时间范围内归还图书。

  要求：

  1. 涉及以上场景的数据库表，并建立表关系。
  2. 查询某个分类下的图书借阅信息。
  3. 查询在某个时间之后的图书借阅信息。
  4. 查询图书借阅周期在某个时间范围内的图书借阅信息（图书借阅周期与查询时间范围有交集）。

- 设计网上商城的商品模块：包含商品SPU、SKU、动态属性。

  说明：SPU是指产品的某个型号，不涉及具体规格，如iPhone 11，包含一些共有属性，如CPU、屏幕材质等。而SKU是具体到规格的具体某个商品，如128G，黑色，无需合约版的iPhone 11。

  详情可参见以下商品：

  Apple iPhone11

  Apple iPhone11 Pro

  华为P30 Pro

  华为Mate20 Pro

  要求：

  1. 涉及以上场景的数据库表，并建立表关系。
  2. 查询以iPhone开头的黑色手机。

MySQL索引事务

本节目标

> - 索引
> - 事务

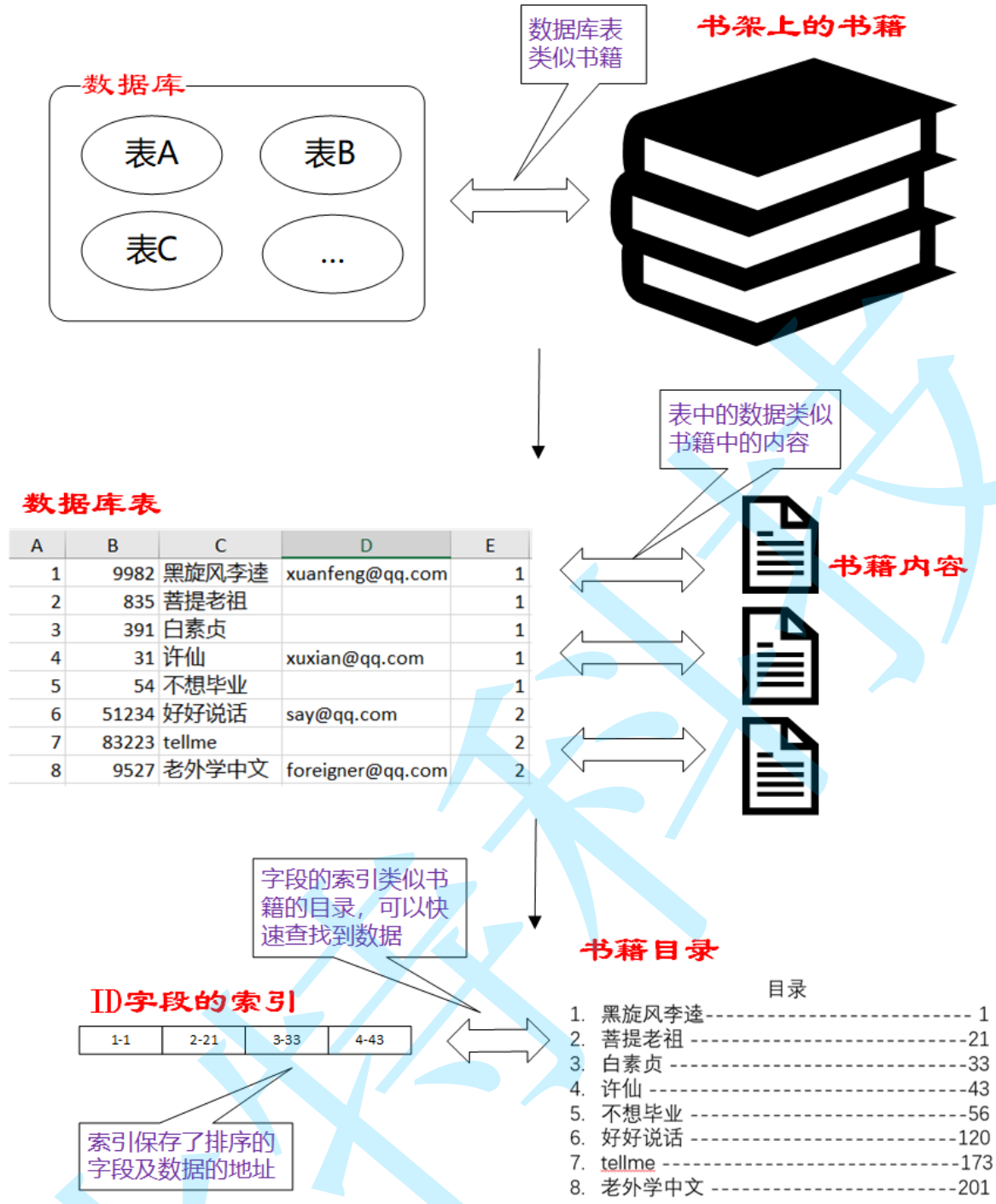
1. 索引

1.1 概念

索引是一种特殊的文件，包含着对数据表里所有记录的引用指针。可以对表中的一列或多列创建索引，并指定索引的类型，各类索引有各自的数据结构实现。（具体细节在后续的数据库原理课程讲解）

1.2 作用

- 数据库中的表、数据、索引之间的关系，类似于书架上的图书、书籍内容和书籍目录的关系。
- 索引所起的作用类似书籍目录，可用于快速定位、检索数据。
- 索引对于提高数据库的性能有很大的帮助。



A	B	C	D	E
1	9982	黑旋风李逵	xuanfeng@qq.com	1
2	835	菩提老祖		1
3	391	白素贞		1
4	31	许仙	xuxian@qq.com	1
5	54	不想毕业		1
6	51234	好好说话	say@qq.com	2
7	83223	tellme		2
8	9527	老外学中文	foreigner@qq.com	2

1.3 使用场景

要考虑对数据库表的某列或某几列创建索引，需要考虑以下几点：

- 数据量较大，且经常对这些列进行条件查询。
- 该数据库表的插入操作，及对这些列的修改操作频率较低。
- 索引会占用额外的磁盘空间。

满足以上条件时，考虑对表中的这些字段创建索引，以提高查询效率。

反之，如果非条件查询列，或经常做插入、修改操作，或磁盘空间不足时，不考虑创建索引。

1.4 使用

创建主键约束（PRIMARY KEY）、唯一约束（UNIQUE）、外键约束（FOREIGN KEY）时，会自动创建对应列的索引。

- 查看索引

```
show index from 表名;
```

案例：查看学生表已有的索引

```
show index from student;
```

- 创建索引

  对于非主键、非唯一约束、非外键的字段，可以创建普通索引

  ```
  create index 索引名 on 表名(字段名);
  ```

  案例：创建班级表中，name字段的索引

  ```
  create index idx_classes_name on classes(name);
  ```

- 删除索引

  ```
  drop index 索引名 on 表名;
  ```

  案例：删除班级表中name字段的索引

  ```
  drop index idx_classes_name on classes;
  ```

1.5 案例

准备测试表：

```
-- 创建用户表
DROP TABLE IF EXISTS test_user;
CREATE TABLE test_user (
        id_number INT,
        name VARCHAR(20) comment '姓名',
        age INT comment '年龄',
        create_time timestamp comment '创建日期'
);
```

准备测试数据，批量插入用户数据（操作耗时较长，约在1小时+）：

```
-- 构建一个8000000条记录的数据
-- 构建的海量表数据需要有差异性，所以使用存储过程来创建， 拷贝下面代码就可以了，暂时不用理解

-- 产生名字
drop function if exists rand_name;
delimiter $$
create function rand_name(n INT, l INT)
returns varchar(255)
begin
 declare return_str varchar(255) default '';
 declare i int default 0;
 while i < n do
     if i=0 then
            set return_str = rand_string(l);
```

```
      else
          set return_str =concat(return_str,concat(' ', rand_string(l)));
      end if;
      set i = i + 1;
      end while;
     return return_str;
     end $$
delimiter ;

-- 产生随机字符串
drop function if exists rand_string;
delimiter $$
create function rand_string(n INT)
returns varchar(255)
begin
 declare lower_str varchar(100) default
     'abcdefghijklmnopqrstuvwxyz';
 declare upper_str varchar(100) default
     'ABCDEFJHIJKLMNOPQRSTUVWXYZ';
 declare return_str varchar(255) default '';
 declare i int default 0;
 declare tmp int default 5+rand_num(n);
 while i < tmp do
     if i=0 then
            set return_str
=concat(return_str,substring(upper_str,floor(1+rand()*26),1));
     else
          set return_str
=concat(return_str,substring(lower_str,floor(1+rand()*26),1));
     end if;

     set i = i + 1;
     end while;
    return return_str;
    end $$
delimiter ;

-- 产生随机数字
drop function if exists rand_num;
delimiter $$
create function rand_num(n int)
returns int(5)
begin
 declare i int default 0;
 set i = floor(rand()*n);
return i;
end $$
delimiter ;

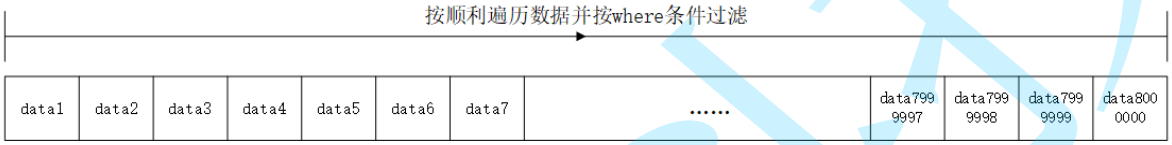
-- 向用户表批量添加数据
drop procedure if exists insert_user;
delimiter $$
create procedure insert_user(in start int(10),in max_num int(10))
begin
declare i int default 0;
 set autocommit = 0;
 repeat
    set i = i + 1;
```

```
    insert into test_user values ((start+i) ,rand_name(2,
5),rand_num(120),CURRENT_TIMESTAMP);
    until i = max_num
 end repeat;
 commit;
end $$
delimiter ;

-- 执行存储过程，添加8000000条用户记录
call insert_user(1, 8000000);
```

查询 id_number 为778899的用户信息：

```
-- 可以看到耗时4.93秒，这还是在本机一个人来操作，在实际项目中，如果放在公网中，假如同时有1000
个人并发查询，那很可能就死机。
select * from test_user where id_number=556677;
```



可以使用explain来进行查看SQL的执行：

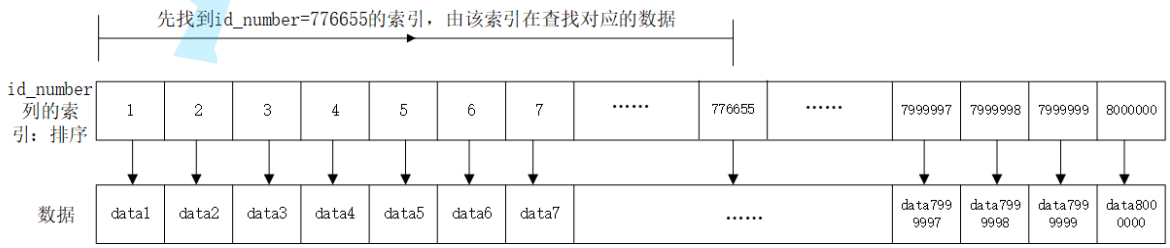
```
explain select * from test_user where id_number=556677;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_user
         type: ALL
possible_keys: NULL
          key: NULL  <== key为null表示没有用到索引
      key_len: NULL
          ref: NULL
         rows: 6
        Extra: Using where
1 row in set (0.00 sec)
```

为提供查询速度，创建 id_number 字段的索引：

```
create index idx_test_user_id_number on test_user(id_number);
```

换一个身份证号查询，并比较执行时间：

```
select * from test_user where id_number=776655;
```



可以使用explain来进行查看SQL的执行：

```
explain select * from test_user where id_number=776655;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_user
         type: ref
possible_keys: idx_test_user_id_number
          key: idx_test_user_id_number <= key用到了idx_test_user_id_number
      key_len: NULL
          ref: const
         rows: 1
        Extra: Using where
1 row in set (0.00 sec)
```

索引保存的数据结构主要为B+树，及hash的方式，实现原理会在以后数据库原理的部分讲解。

2. 事务

2.1 为什么使用事务

准备测试表：

```
drop table if exists accout;
create table accout(
    id int primary key auto_increment,
    name varchar(20) comment '账户名称',
    money decimal(11,2) comment '金额'
);

insert into accout(name, money) values
('阿里巴巴', 5000),
('四十大盗', 1000);
```

比如说，四十大盗把从阿里巴巴的账户上偷盗了2000元

```
-- 阿里巴巴账户减少2000
update accout set money=money-2000 where name = '阿里巴巴';
-- 四十大盗账户增加2000
update accout set money=money+2000 where name = '四十大盗';
```

假如在执行以上第一句SQL时，出现网络错误，或是数据库挂掉了，阿里巴巴的账户会减少2000，但是四十大盗的账户上就没有了增加的金额。

解决方案：使用事务来控制，保证以上两句SQL要么全部执行成功，要么全部执行失败。

2.2 事务的概念

事务指逻辑上的一组操作，组成这组操作的各个单元，要么全部成功，要么全部失败。

在不同的环境中，都可以有事务。对应在数据库中，就是数据库事务。

2.3 使用

（1）开启事务：start transaction;

（2）执行多条SQL语句

（3）回滚或提交：rollback/commit;

说明：rollback即是全部失败，commit即是全部成功。

```
start transaction;
-- 阿里巴巴账户减少2000
update accout set money=money-2000 where name = '阿里巴巴';
-- 四十大盗账户增加2000
update accout set money=money+2000 where name = '四十大盗';
commit;
```

事务的特性及设置，会在后续 数据库原理 部分进一步讲解。

3. 内容重点总结

- 索引：
  （1）对于插入、删除数据频率高的表，不适用索引
  （2）对于某列修改频率高的，该列不适用索引
  （3）通过某列或某几列的条件查询频率高的，可以对这些列创建索引
- 事务

  ```
  start transaction;
  ...
  rollback/commit;
  ```

4. 课后作业

Java的JDBC编程

本节目标

> - 数据库驱动
> - JDBC的概念及作用
> - 掌握JDBC的工作原理
> - 掌握JDBC中几个常用接口和类
> - 掌握基于数据库的应用程序开发流程

1. 数据库编程的必备条件

- 编程语言，如Java，C、C++、Python等
- 数据库，如Oracle，MySQL，SQL Server等
- 数据库驱动包：不同的数据库，对应不同的编程语言提供了不同的数据库驱动包，如：MySQL提供了Java的驱动包mysql-connector-java，需要基于Java操作MySQL即需要该驱动包。同样的，要基于Java操作Oracle数据库则需要Oracle的数据库驱动包ojdbc。

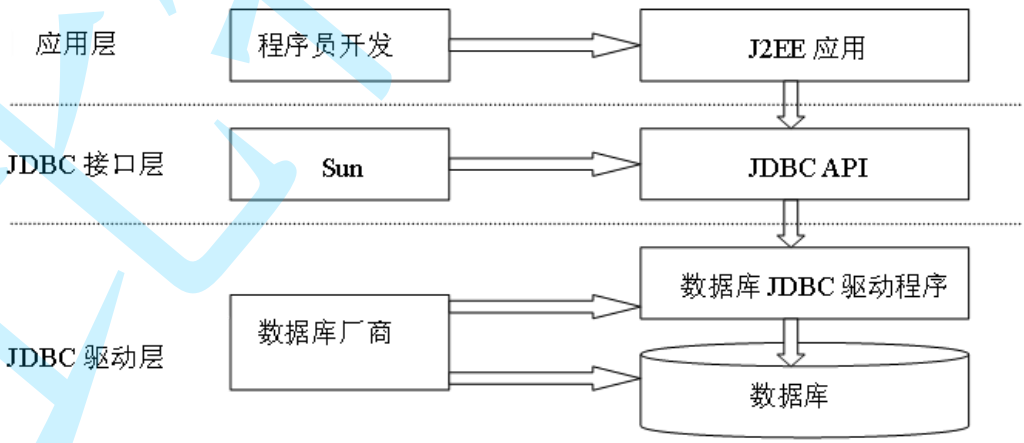
2. Java的数据库编程：JDBC

JDBC，即Java Database Connectivity，java数据库连接。是一种用于执行SQL语句的Java API，它是Java中的数据库连接规范。这个API由`java.sql.*,javax.sql.*`包中的一些类和接口组成，它为Java开发人员操作数据库提供了一个标准的API，可以为多种关系数据库提供统一访问。

3. JDBC工作原理

JDBC 为多种关系数据库提供了统一访问方式，作为特定厂商数据库访问API的一种高级抽象，它主要包含一些通用的接口类。

JDBC访问数据库层次结构:



JDBC优势:

- Java语言访问数据库操作完全面向抽象接口编程
- 开发数据库应用不用限定在特定数据库厂商的API
- 程序的可移植性大大增强

4. JDBC使用

4.1 JDBC开发案例

- 准备数据库驱动包，并添加到项目的依赖中：

  在项目中创建文件夹lib，并将依赖包mysql-connector-java-5.1.47.jar复制到lib中。再配置该jar包到本项目的依赖中：右键点击项目Open Module Settings，在Modules中，点击项目，配置Dependencies，点击+，JARS or Directories，将该lib文件夹配置进依赖中，表示该文件夹下的jar包都引入作为依赖。

- 建立数据库连接

```
// 加载JDBC驱动程序：反射，这样调用初始化com.mysql.jdbc.Driver类，即将该类加载到JVM方法区，并执行该类的静态方法块、静态属性。
Class.forName("com.mysql.jdbc.Driver");

// 创建数据库连接
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test?
user=root&password=root&useUnicode=true&characterEncoding=UTF-8");
```

```
//MySQL数据连接的URL参数格式如下：
jdbc:mysql://服务器地址:端口/数据库名?参数名=参数值
```

- 创建操作命令（Statement）

```
Statement statement = connection.createStatement();
```

- 执行SQL语句

```
ResultSet resultSet= statement.executeQuery(
                "select id, sn, name, qq_mail, classes_id from student");
```

- 处理结果集

```
while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String sn = resultSet.getString("sn");
            String name = resultSet.getString("name");
            int classesId = resultSet.getInt("classes_id");
            System.out.println(String.format("Student: id=%d, sn=%s, name=%s,
classesId=%s", id, sn, name, classesId));
        }
```

- 释放资源（关闭结果集，命令，连接）

```
//关闭结果集
if (resultSet != null) {
    try {
        resultSet.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
}
//关闭命令
if (statement != null) {
    try {
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
//关闭连接命令
if (connection != null) {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

4.2 JDBC使用步骤总结

1. 创建数据库连接Connection
2. 创建操作命令Statement
3. 使用操作命令来执行SQL
4. 处理结果集ResultSet
5. 释放资源

5. JDBC常用接口和类

5.1 JDBC API

在Java JDBC编程中对数据库的操作均使用JDK自带的API统一处理，通常与特定数据库的驱动类是完全解耦的。所以掌握Java JDBC API （位于`java.sql`包下） 即可掌握Java数据库编程。

5.2 数据库连接Connection

Connection接口实现类由数据库提供，获取Connection对象通常有两种方式：

- 一种是通过DriverManager（驱动管理类）的静态方法获取：

```
// 加载JDBC驱动程序
Class.forName("com.mysql.jdbc.Driver");

// 创建数据库连接
Connection connection = DriverManager.getConnection(url);
```

- 一种是通过DataSource（数据源）对象获取。**实际应用中会使用DataSource对象。**

```
DataSource ds = new MysqlDataSource();
((MysqlDataSource) ds).setUrl("jdbc:mysql://localhost:3306/test");
((MysqlDataSource) ds).setUser("root");
((MysqlDataSource) ds).setPassword("root");
Connection connection = ds.getConnection();
```

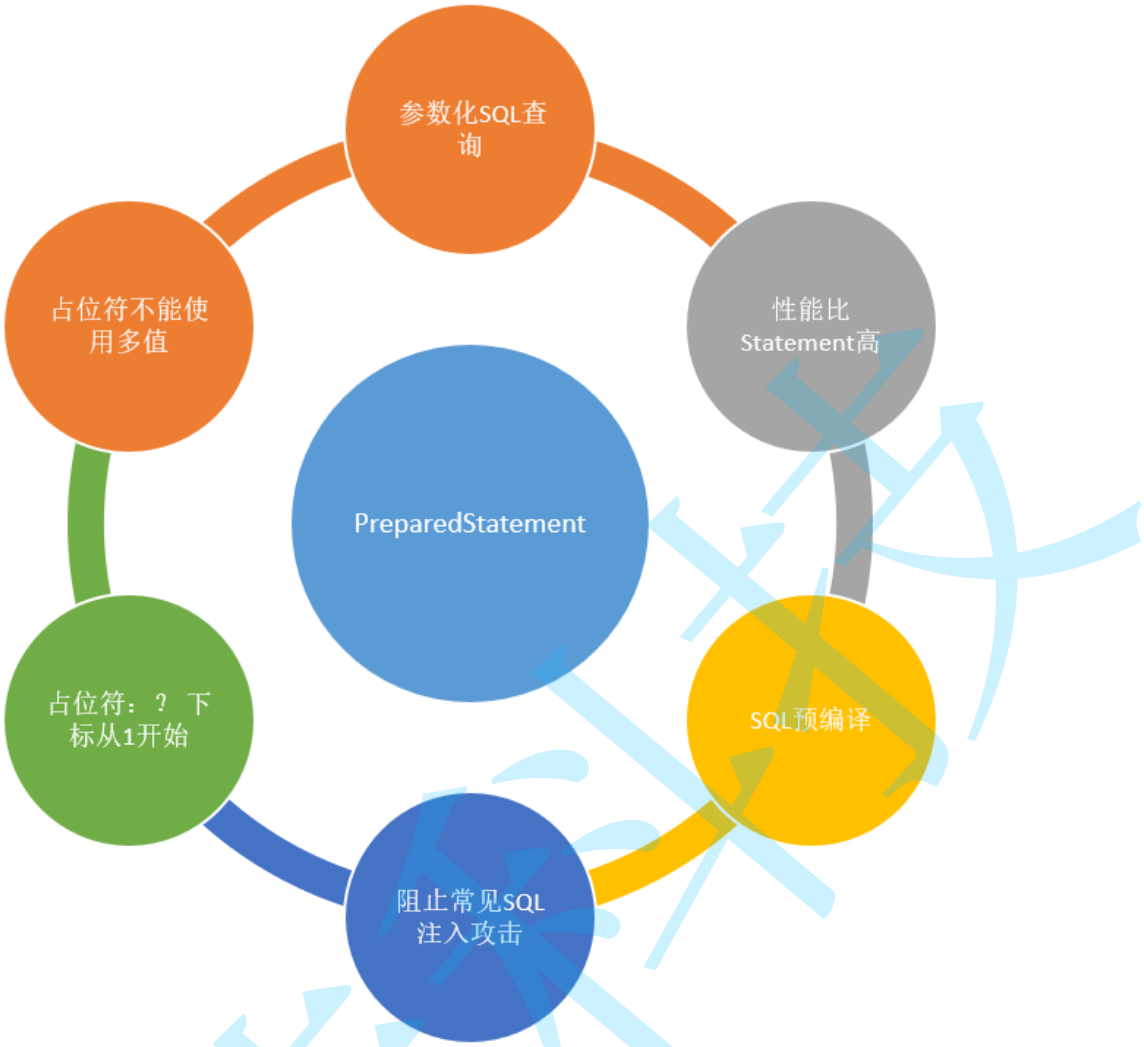
- 以上两种方式的区别是：

1. DriverManager类来获取的Connection连接，是无法重复利用的，每次使用完以后释放资源时，通过connection.close()都是关闭物理连接。
2. DataSource提供连接池的支持。连接池在初始化时将创建一定数量的数据库连接，这些连接是可以复用的，每次使用完数据库连接，释放资源调用connection.close()都是将Conncetion连接对象回收。

5.3 Statement对象

Statement对象主要是将SQL语句发送到数据库中。JDBC API中主要提供了三种Statement对象。

实际开发中最常用的是PreparedStatement对象，以下对其的总结：



主要掌握两种执行SQL的方法：

- executeQuery() 方法执行后返回单个结果集的，通常用于select语句
- executeUpdate()方法返回值是一个整数，指示受影响的行数，通常用于update、insert、delete语句

5.4 ResultSet对象

ResultSet对象它被称为结果集，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，并且有一个记录指针，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。

6. 应用案例

技术知识点：

- JDBC API的CRUD
- JDBC API的事务控制

功能要求：

- 学生表
  - 自动化2019级5班新增一名同学程咬金
  - 修改该同学的班级为中文系2019级3班

    - 查询所有中文系2019级3班的同学
    - 删除名叫程咬金的同学
- 成绩表
    - 新增许仙同学的成绩：英文80分，Java65分，计算机原理76分，语文59分
    - 修改许仙同学的成绩：英文81分
    - 删除许仙同学的语文成绩
    - 查询中文系2019级3班同学的成绩

7. 内容重点总结

JDBC使用步骤：

1. 创建数据库连接Connection
    - DriverManager创建
    - DataSource获取
2. 创建操作命令Statement
    - PreparedStatement
3. 使用操作命令来执行SQL

```
// 查询操作
preparedStatement.executeQuery();

// 新增、修改、删除操作
preparedStatement.executeUpdate();
```

4. 处理结果集ResultSet

```
while (resultSet.next()) {
    int xxx = resultSet.getInt("xxx");
    String yyy= resultSet.getString("yyy");
    ...
}
```

5. 释放资源

```
try {
    if(resultSet != null){
        resultSet.close();
    }
    if(preparedStatement != null){
        preparedStatement.close();
    }
    if(connection != null){
        connection.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
    throw new RuntimeException("数据库错误");
}
```

面试问答：

1. 数据库连接有哪些方式？分别有什么区别

2. 数据库Statement和PreparedStatement有什么区别?

8. 课后作业

- 图书管理系统
  - 新增貂蝉同学的借阅记录：诗经，从2019年9月25日17:50到2019年10月25日17:50
  - 查询计算机分类下的图书借阅信息
  - 修改图书《深入理解Java虚拟机》的价格为61.20
  - 删除id最大的一条借阅记录