**MySQL笔记**

管理员权限命令提示符：MySQL的bin目录下执行

net start mysql 启动mysql服务 net stop mysql 停止mysql服务

mysql -u root -p密码 进入MySQL数据库 exit退出mysql

Navicat下执行

CREATE DATABASE IF NOT EXISTS 数据库名 --创建数据库

CREATE TABLE IF NOT EXISTS 表名 --创建表

DROP DATABASE IF EXISTS 数据库名 --删除数据库

DROP TABLE IF EXISTS 表名 --删除表

CREATE TABLE IF NOT EXISTS `student` (```飘)(

`id` INT(4) NOT NULL AUTO\_INCREMENT COMMENT '学号',

`name` VARCHAR(20) NOT NULL DEFAULT '匿名' COMMENT '姓名',

`pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',

`birthday` DATE DEFAULT NULL COMMENT '出生日期',

`address` VARCHAR(50) DEFAULT NULL COMMENT '家庭住址',

`e-mail` VARCHAR(20) DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY(`id`) --指定`id`为主键

)ENGINE=INNODB DEFAULT CHARSET=UTF8 /\*数据库引innnoDB 默认字符集编码为utf-8 \*/

SHOW CREATE DATABASE school --查看创建数据库school的语句

SHOW CREATE TABLE student --查看创建表student的语句

DESC student --显示表的结构

修改表

ALTER TABLE teacher RENAME AS teacher1 --表格重命名

ALTER TABLE teacher1 ADD age INT(3) DEFAULT NULL – 添加字段

ALTER TABLE teacher1 MODIFY age VARCHAR(4) --MODIFY只能修改字段类型和约束，不能用于字段重命名。

ALTER TABLE teacher1 CHANGE age age1 INT(3) --CHANGE只能用于字段重命名，不能用于修改表的约束。

ALTER TABLE teacher1 DROP age1 --删除字段

**MySQL数据管理**

添加外键

CREATE TABLE `grade` (

`gradeid` INT(10) NOT NULL COMMENT '年级id',

`gradename` VARCHAR(10)NOT NULL COMMENT '年级名称',

PRIMARY KEY (`gradeid`)

)ENGINE=INNODB DEFAULT CHARSET=utf8

学生表的gradeid要去引用年级表的gradeid

CREATE TABLE IF NOT EXISTS `student` (

`id` INT(4) NOT NULL AUTO\_INCREMENT COMMENT '学号',

`gradeid` INT(10) NOT NULL COMMENT '学生年级号',

`name` VARCHAR(20) NOT NULL DEFAULT '匿名' COMMENT '姓名',

`pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',

`birthday` DATE DEFAULT NULL COMMENT '出生日期',

`address` VARCHAR(50) DEFAULT NULL COMMENT '家庭住址',

`e-mail` VARCHAR(20) DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY(`id`),

KEY `FK\_gradeid` (`gradeid`), --定义外键key

CONSTRAINT `FK\_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade` (`gradeid`) – 给这个外键添加约束（执行引用） references引用

)ENGINE=INNODB DEFAULT CHARSET=UTF8

--创建完表之后，添加外键约束

ALTER TABLE `student`

ADD CONSTRAINT `FK\_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade` (`gradeid`)

以上操作都是物理外键，数据库级别的外键，不建议使用

较好的做法：数据库就是单纯的表，只用来存数据，只有行（数据）和列（字段）

想使用多张表的数据，想使用外键（用程序来实现）

INSERT INTO `grade` VALUES('11', 'wang') --值与字段一一对应

INSERT INTO `grade` (`gradename`) VALUES('大一'),('大二') –插入多个值

INSERT INTO `student` (`id`,`gradeid`,`name`,`pwd`) VALUES ('123','333','wang','666666') --字段与值对应

INSERT INTO `student` (`id`,`gradeid`,`name`,`pwd`) VALUES('111','3','wu','6666'),('222','2','liu','6666') --同时插入多条记录

修改表

UPDATE `student` SET `name`='zhang' WHERE id=111 -- 修改id为111的记录的name

UPDATE `student` SET `name`='wang',`pwd`='666666' WHERE id BETWEEN 111 and 222 --修改id介于且包括111到222的name和pwd

UPDATE `student` SET `address`='赣州',`e-mail`='11122244@qq.com' WHERE id=111 OR `name`='wang'

UPDATE `student` SET `address`='赣州',`e-mail`='11122244@qq.com' WHERE id=111 AND `name`='wang'

删除数据

DELETE FROM `student` WHERE id=333 --从student表中删除id=333的记录

TRUNCATE `teacher1` -- 清空表teacher1,自动增量初始值归1，不会影响事务

DELETE FROM `teacher1` -- 删除表teacher1,自动增量初始值不归1

**DQL(数据查询语言)**：

#查询表格全部字段

SELECT \* FROM student

#取别名

SELECT `studentno` AS 学号,`studentname` AS 学生姓名 FROM student AS std

#与字段拼接

SELECT CONCAT('姓名：',studentname) AS newname FROM student

#查询指定字段，且去除重复数据

SELECT DISTINCT `studentno` AS 应考 FROM result

SELECT VERSION() --查询mysql版本 （函数）

SELECT 100\*21-79 AS 结果 --select用来计算 （表达式）

SELECT @@auto\_increment\_increment AS 自增步长 -- 查询自增步长（变量）

SELECT `studentno`, `studentresult`+1 FROM result -- 操作输出字段

#查询成绩位于90到100的学生

SELECT `studentno`, `studentresult` FROM result --或者用&&, ||, !=, !, or

WHERE `studentresult`<=100 AND `studentresult`>=90

SELECT `studentno`, `studentresult` FROM result

WHERE `studentresult` BETWEEN 90 AND 100

#查询学生成绩，学号为1的学生除外

SELECT `studentno`, `studentresult` FROM result

WHERE NOT studentno=1

模糊查询，运算符：is null, is not null, like, in, between

SELECT `studentno`,`studentname` FROM student

WHERE `studentname` LIKE '诸%' -- %代表0到任意个字符

SELECT `studentno`,`studentname` FROM student

WHERE `studentname` LIKE '张\_' -- \_代表一个字符

SELECT `studentno`,`studentname` FROM student

WHERE `studentname` LIKE '\_\_' -- 检索姓名为两个字的学号和姓名

SELECT `studentno`,`studentname` FROM student

WHERE `studentno` IN (7,8,9) -- 检索字段为指定值的记录

SELECT `studentno`,`studentname` FROM student

WHERE `address`='' OR `address` IS NULL -- 查询地址为空的学生

SELECT `studentno`,`studentname` FROM student

WHERE `address` IS NOT NULL -- 查询地址不为NULL的学生

**联表查询**

SELECT s.studentno,studentname,subjectno,studentresult

FROM student AS s

INNER JOIN result AS r

WHERE s.studentno = r.studentno -- 显示两个表中都有的studentno

SELECT s.studentno,studentname,subjectno,studentresult

FROM student AS s

LEFT JOIN result AS r /\*显示左表(student)中所有studentno，即使右表中没有

ON s.studentno = r.studentno --与之匹配的\*/

SELECT s.studentno,studentname,subjectno,studentresult

FROM student AS s

RIGHT JOIN result AS r /\*显示右表(student)中所有studentno，即使左表没有\*/

ON s.studentno = r.studentno --与之匹配的\*/

SELECT s.studentno,studentname,subjectno,studentresult

FROM student AS s

LEFT JOIN result AS r

ON s.studentno = r.studentno

WHERE studentresult IS NULL -- 查询缺考的同学

Join (连接的表) on (判断的条件) 连接查询

Where 等值查询

SELECT s.studentno, studentname,subjectname,studentresult

FROM student s

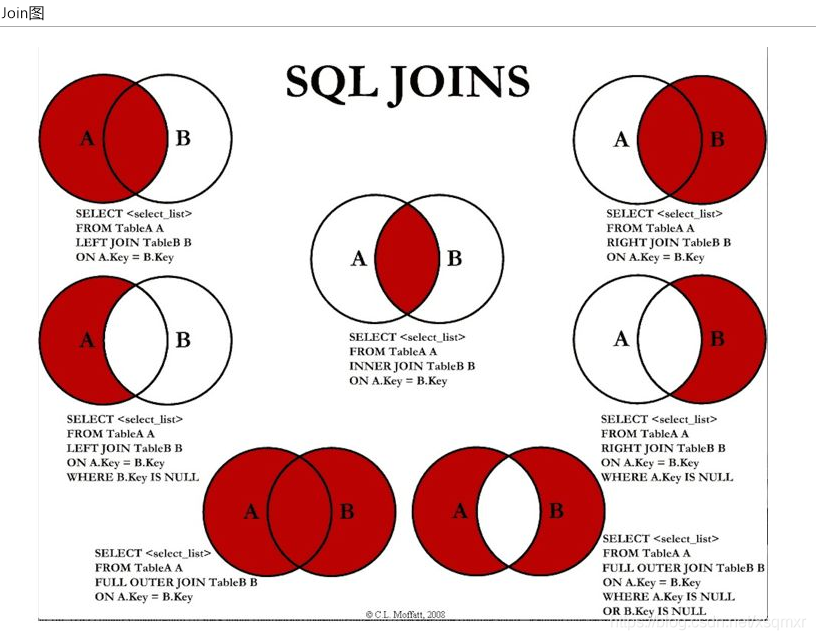
RIGHT JOIN result r

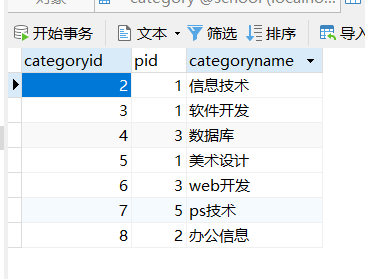
ON r.studentno=s.studentno

INNER JOIN `subject` sub

ON r.subjectno=sub.subjectno -- 查询参加考试的同学及科目名和成绩

自连接：自己的表和自己的表连接，一张表拆成两张一样的表

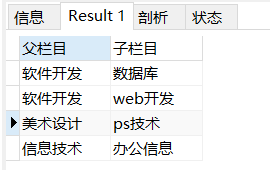




上层的categoryid对应下层的pid



SELECT a.`categoryname` '父栏目', b.`categoryname` '子栏目'

FROM `category` a,`category` b

WHERE a.`categoryid`=b.`pid`

#查询学生所在年级

SELECT `studentno`,`studentname`,`gradename`

FROM `student` s

INNER JOIN `grade` g

ON s.`gradeid`=g.`gradeid`

#查询参加了“计算机网络”考试的学生信息

SELECT s.`studentno`,`studentname`,`subjectname`,`studentresult`

FROM `student` s

INNER JOIN `result` r

ON s.`studentno`=r.`studentno`

INNER JOIN `subject` sub

ON r.`subjectno`=sub.`subjectno`

WHERE `subjectname`='计算机网络'



**排序**

#将参加“计算机网络”考试的同学按成绩(DESC)降序排序或(ASC)升序排序

SELECT s.`studentno`,`studentname`,`subjectname`,`studentresult`

FROM `student` s

INNER JOIN `result` r

ON s.`studentno`=r.`studentno`

INNER JOIN `subject` sub

ON r.`subjectno`=sub.`subjectno`

WHERE `subjectname`='计算机网络'

ORDER BY `studentresult` DESC 不写DESC/ASC时，默认升序排序

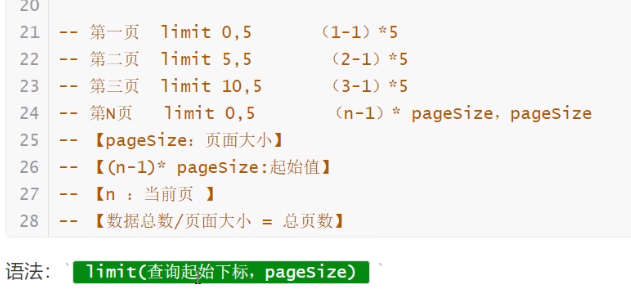
**分页（缓解数据库压力，优化体验）**

SELECT s.studentno,studentname,subjectno,studentresult

FROM student AS s

LEFT JOIN result AS r

ON s.studentno = r.studentno

LIMIT 0,8 -- 从第1条数据开始，显示8条数据

查询考了“计算机网络”且成绩大于70的前10名同学。

SELECT s.studentno, studentname,subjectname,studentresult

FROM student s

RIGHT JOIN result r

ON r.studentno=s.studentno

INNER JOIN `subject` sub

ON r.subjectno=sub.subjectno

WHERE subjectname='计算机网络' AND studentresult>70

LIMIT 0,10

**子查询 （在where里面嵌套一个子查询）**

#查询参加高等数学-1考试的学生成绩

#方式一：连接查询

SELECT `studentno`,r.`subjectno`,studentresult

FROM `result` r

INNER JOIN `subject` sub

ON r.subjectno = sub.subjectno

WHERE subjectname='高等数学-1'

ORDER BY studentresult DESC

#方式二：子查询

SELECT studentno,subjectno,studentresult

FROM `result`

WHERE `subjectno`=(SELECT subjectno FROM `subject`

WHERE subjectname='高等数学-1')

ORDER BY studentresult DESC

查询高等数学-1成绩大于80的学生学号和姓名

SELECT DISTINCT s.studentno,studentname

FROM student s

INNER JOIN result r

ON r.studentno = s.studentno

WHERE studentresult>=80 AND subjectno=(

SELECT subjectno FROM `subject` WHERE subjectname='高等数学-1'

)

**MySQL函数**

#常见函数

#数学运算

--数学运算

SELECT ABS(-9) --绝对值

SELECT CEILING(5.3) --向上取整

SELECT FLOOR(4.5) --向下取整

SELECT RAND() --生成0~1的随机数

SELECT SIGN(111) --返回参数符号，0返回0，负数返回-1，整数返回1

#字符串函数

SELECT CHAR\_LENGTH('伞兵一号卢本伟准备就绪') -- 返回字符串长度

SELECT CONCAT('伞兵一号','卢本伟','准备就绪') --拼接字符串

SELECT INSERT('江西LG大学',3,2,'理工') --从第3个位置开始，替换两个字符

SELECT LOWER('WANGXIANGTAI') -- 转小写

SELECT UPPER('wangxiangtai') -- 转大写

SELECT INSTR('WANGXIANG','g') -- 返回指定字符第一次出现的位置

SELECT REPLACE('小葵花妈妈课堂开课啦','小葵花','小菊花')—后面的替换前面

SELECT SUBSTR('小葵花妈妈课堂开课啦',1,5) -- 从第一个开始截取5个字符

SELECT REVERSE('为所欲为') -- 反转

#将姓张的替换为姓牛的

SELECT REPLACE(studentname,'张','牛') FROM student

WHERE studentname LIKE '张%'

#时间、日期函数

SELECT CURRENT\_TIME() -- 返回当前时间

SELECT CURRENT\_DATE() -- 返回当前日期

SELECT now() -- 返回当前时间和日期

SELECT LOCALTIME() -- 本地时间

SELECT SYSDATE() -- 返回系统时间

SELECT YEAR(NOW()) -- 返回当前年

SELECT MONTH(NOW()) -- 返回当前月

SELECT DAY(NOW())

SELECT HOUR(NOW())

SELECT MINUTE(NOW())

SELECT SECOND(NOW())

SELECT SYSTEM\_USER() -- 返回用户名

SELECT USER() -- 返回用户名

SELECT VERSION() -- 返回mysql版本号

**聚合函数（常用）**

SELECT COUNT(borndate) FROM student -- count(指定列)计数，字段值为null，不计数

SELECT COUNT(\*) FROM student -- 不会忽略null值，按行数计数

SELECT COUNT(1) FROM student -- 不会忽略null值，按行数计数

SELECT SUM(`studentresult`) FROM result -- 求和

SELECT AVG(`studentresult`) FROM result -- 求平均值

SELECT MAX(`studentresult`) FROM result -- 最大值

SELECT MIN(`studentresult`) FROM result -- 最小值

#查找平均分大于80的科目

SELECT subjectname,ROUND(AVG(studentresult)\*100)/100 平均分, MAX(studentresult) 最高分, MIN(studentresult) 最低分 FROM result r

INNER JOIN `subject` sub

ON r.subjectno=sub.subjectno

GROUP BY r.subjectno -- 指定字段按照哪个字段分组

HAVING 平均分>80 -- 过滤分组后的记录必须满足的次要条件

**数据库级别的MD5加密**

Md5不可逆，具体值的md5是一样的，没次加密得到不同结果，同一原数据加密相同次数的得到的MD5相同

CREATE TABLE `testmd5`(

`id` INT(4) NOT NULL,

`name` VARCHAR(20) NOT NULL,

`pwd` VARCHAR(50) NOT NULL,

PRIMARY KEY(`ID`)

)ENGINE=INNODB DEFAULT CHARSET=utf8

--明文密码

INSERT INTO `testmd5` VALUES(1,'张三','123456'),(2,'李四','111333'),(3,'王五','666666')

UPDATE testmd5 SET pwd=MD5(pwd) WHERE id=1 -- 加密id=1的pwd

UPDATE testmd5 SET pwd=MD5(pwd) -- 加密全部的pwd

INSERT INTO testmd5 VALUES(4,'小明',MD5('444555')) -- 插入的时候加密

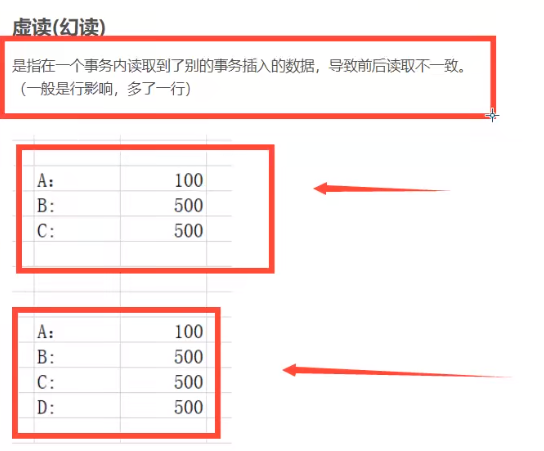
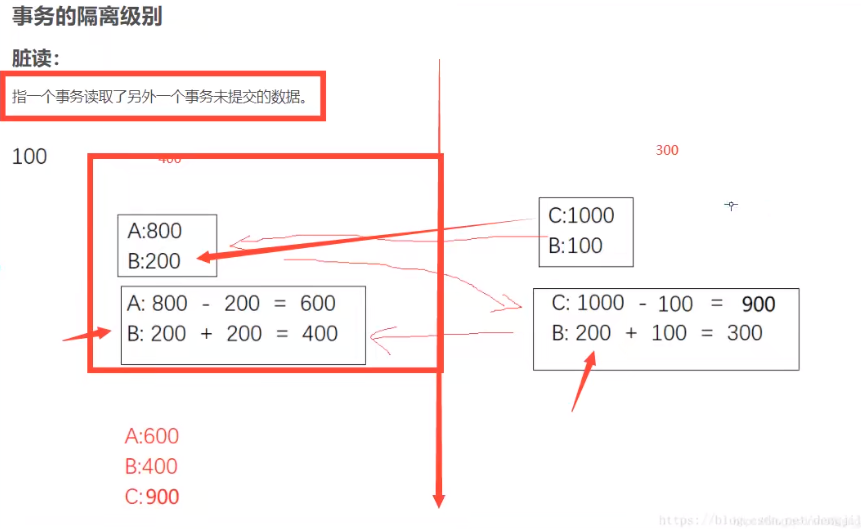
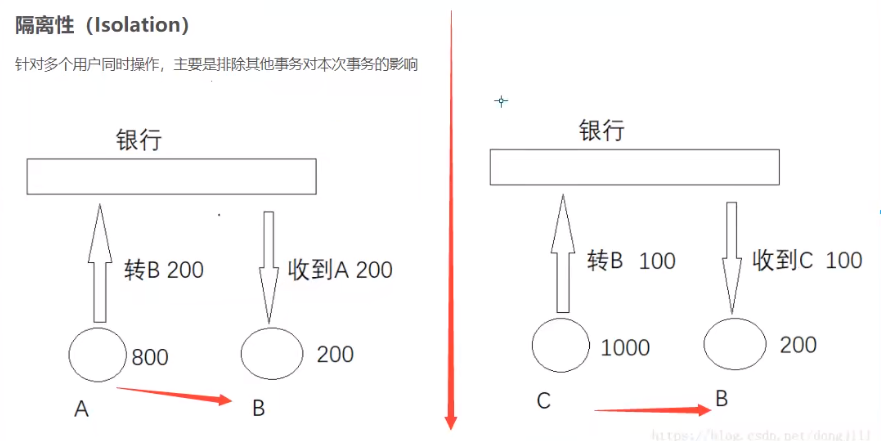


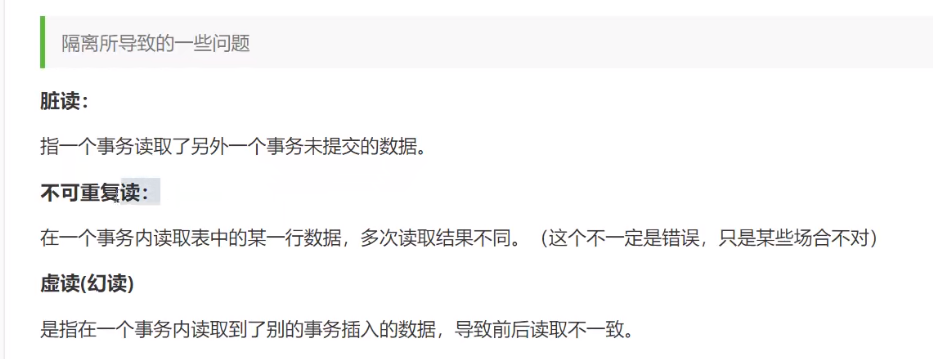
**6. 事务**

将一组SQL放在一个批次中执行，要么都成功，要么都失败









//mysql默认开启自动提交事务的

SET autocommit=0 -- 关闭自动提交事务

SET autocommit=1 -- 开启自动提交事务（默认）

-- 手动处理事务

SET autocommit=0 -- 关闭自动提交

-- 事务开启

/\*标记一个事务的开始，此处开始的sql都在同一个事务中\*/

START TRANSACTION /\*标记一个事务的开始，此处开始的sql都在同一个事务中\*/

INSERT xx

INSERT xx

COMMIT -- 提交：持久化 （成功！）

ROLLBACK -- 回滚；回到原来的样子（失败！）

-- 事务结束

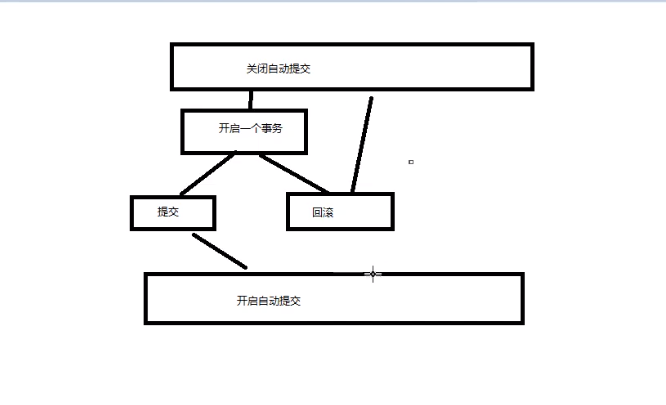
SET autocommit=1 -- 开启自动提交

-- 了解

SAVEPOINT 保存点名 -- 设置一个事务的保存点

ROLLBACK TO SAVEPOINT 保存点名 -- 回滚到保存点

RELEASE SAVEPOINT 保存点名 -- 撤销保存点



CREATE DATABASE shop

USE shop

DROP TABLE account

CREATE TABLE `account`(

`id` int(3) not null AUTO\_INCREMENT,

`name` VARCHAR(30) not NULL,

`money` DECIMAL(9,2) NOT NULL,

PRIMARY KEY (`id`)

)ENGINE=INNODB DEFAULT CHARSET=UTF8

INSERT INTO account (`name`,`money`) VALUES('张三',2000.00),('李四',10000.00)

-- 模拟转账：事务

SET autocommit=0; -- 关闭自动提交

START TRANSACTION -- 开启一个事务（一组事务）

UPDATE account SET money=money-1000 WHERE `name`='李四'

UPDATE account SET money=money+1000 WHERE `name`='张三'

COMMIT; -- 提交事务，被持久化了

ROLLBACK; -- 回滚

SET autocommit=1; -- 恢复默认值

7. 索引

索引的分类

1. 主键索引 PRIMARY KEY ：唯一标识，不可重复，只能有一列
2. 唯一索引 UNIQUE KEY：避免重复列的出现，唯一索引可以重复，可以有多个列标记为唯一索引。
3. 常规索引 KEY/INDEX：默认的，用index或Key关键字来设置
4. 全文索引 FullText：在特定的数据库引擎下才有，MYISAM，作用是快速定位数据

PRIMARY KEY (`studentno`),

UNIQUE KEY `identitycard` (`identitycard`), -- `索引名` (`字段名`)

KEY `email` (`email`)

//显示所有的索引信息

SHOW INDEX FROM student

-- 给student表中增加一个全文索引

ALTER TABLE school.student ADD FULLTEXT INDEX `studentname` (`studentname`)

-- EXPLAIN 分析sql执行的状况

EXPLAIN SELECT \* FROM student -- 非全文索引

SELECT \* FROM student WHERE MATCH(studentname) AGAINST('张三')

EXPLAIN SELECT \* FROM student WHERE MATCH(studentname) AGAINST('张三')



//创建app\_user表

CREATE TABLE `app\_user`(

`id` BIGINT(20) UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) DEFAULT '' COMMENT '用户昵称',

`email` VARCHAR(50) NOT NULL COMMENT '用户邮箱',

`phone` VARCHAR(20) DEFAULT '' COMMENT '手机号',

`gender` TINYINT(4) UNSIGNED DEFAULT '0' COMMENT '性别：0-男，1-女',

`password` VARCHAR(100) NOT NULL COMMENT '密码',

`age` TINYINT(4) DEFAULT '0' COMMENT '年龄',

`create\_time` DATETIME DEFAULT CURRENT\_TIMESTAMP,

`update\_time` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY(`id`)

)ENGINE=INNODB DEFAULT CHARSET=utf8mb4 COMMENT='app用户表'

-- 往app\_user表中插入100万条数据（了解）

DELIMITER $$ -- 写函数之前必须要写，分隔符

CREATE FUNCTION mock\_data()

RETURNS INT -- 返回值int类型

DETERMINISTIC

BEGIN

DECLARE num INT DEFAULT 1000000; -- 声明变量

DECLARE i INT DEFAULT 0;

WHILE i<num DO

-- 插入语句

INSERT INTO `app\_user`(`name`,`email`,`phone`,`gender`,`password`,`age`) VALUES(CONCAT('user',i),'2714661705@qq.com',CONCAT('15',FLOOR(RAND()\*((999999999-100000000)+100000000))),CEILING(RAND()\*2),UUID(),FLOOR(RAND()\*100));

SET i=i+1;

END WHILE;

RETURN i;

END $$;

SELECT @@log\_bin\_trust\_function\_creators;

SET GLOBAL log\_bin\_trust\_function\_creators=1;

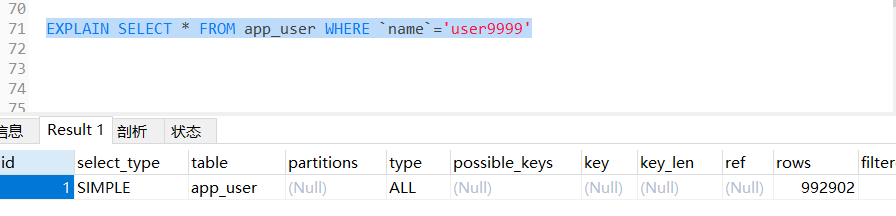
//执行函数

SELECT mock\_data();

SELECT \* FROM app\_user WHERE `name`='user9999' -- 时间: 0.538s，数据多查询时间长。

SELECT \* FROM student -- 时间: 0.001s

//分析不用索引的查询过程



//查询了992902行找到目标

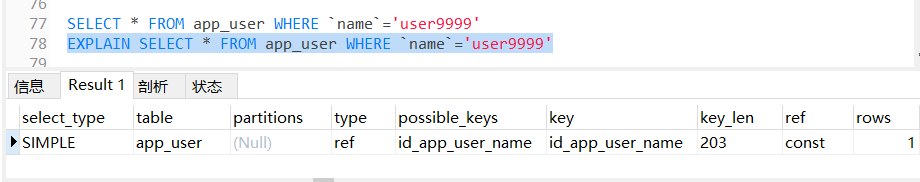
使用索引查询：

-- 添加索引

-- CREATE INDEX 索引名 ON 表名(字段名)

CREATE INDEX id\_app\_user\_name ON app\_user(`name`)

//添加索引之后再查询

SELECT \* FROM app\_user WHERE `name`='user9999' -- 时间: 0.005s

直接定位唯一索引（B-tree是InnoDB默认的索引数据结构），不用遍历

索引在数据量小时，效果不明显，在大数据时，效果突出

索引原则：

1. 索引不是越多越好
2. 不要对经常变动的数据加索引
3. 小数据量的表不需要加索引
4. 索引一般加在常用来查询的字段上

**索引的本质**

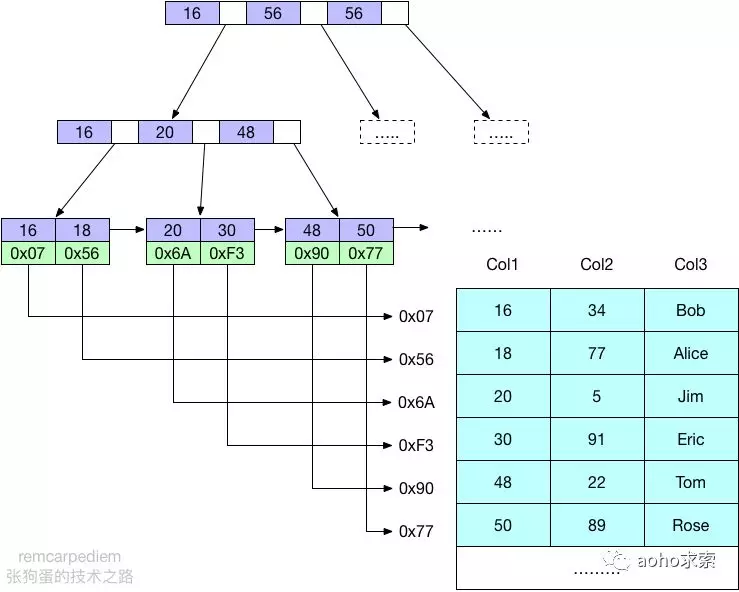
MySQL官方对索引的定义为：索引（Index）是帮助MySQL高效获取数据的数据结构。提取句子主干，就可以得到索引的本质：索引是数据结构。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

目前大部分数据库系统及文件系统都采用B-Tree或其变种B+Tree作为索引结构

B-Tree索引

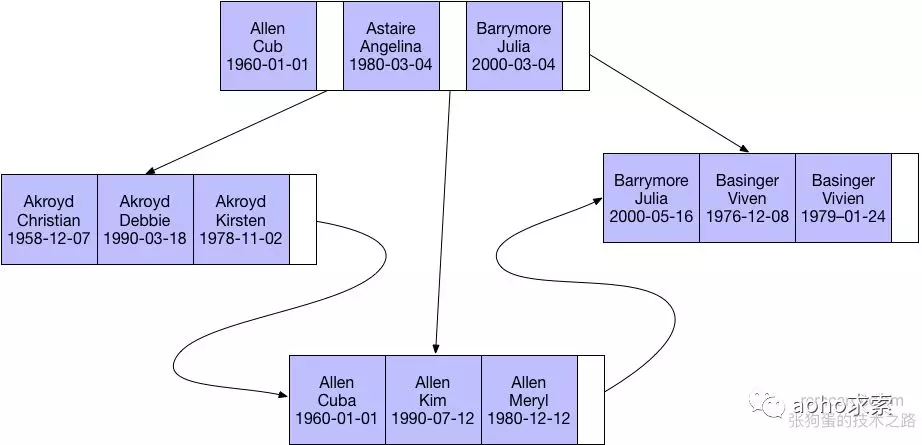
B-Tree索引使用B-Tree来存储数据，当然不同存储引擎的实现方式不同。B-Tree通常意味着所有的值都是按顺序存储的，并且每一个叶子页到根的距离相同，



MySQL可以在单独一列上添加B-Tree索引，也可以在多列数据上添加B-Tree索引，多列的数据按照添加索引声明的顺序组合起来，存储在B-Tree的页中。假设有如下数据表：

1. CREATE TABLE People (
2. last\_name    varchar(50)    not null,
3. first\_name   varchar(50)    not null,
4. birthday     date           not null,
5. gender       enum('m','f')  not null
6. key(last\_name, first\_name, birthday)
7. );

对于表中的每一行数据，索引中包含了lastname，firstname和birthday列的值，图2展示了该索引是如何组织数据的存储的。



B-Tree索引使用B-Tree作为其存储数据的数据结构，其使用的查询规则也由此决定。一般来说，B-Tree索引适用于全键值、键值范围和键前缀查找，其中键前缀查找只适用于根据最左前缀查找。

B-Tree索引支持的查询原则如下所示：

全值匹配：全值匹配指的是和索引中的所有列进行匹配，

* 匹配最左前缀：前边提到的索引可以用于查找所有姓Allen的人，即只使用索引中的第一列。
* 匹配列前缀：也可以只匹配某一列的值的开头部分。例如前面提到的索引可用于查找所有以J开头的姓的人。这里也只用到了索引的第一列。
* 匹配范围值：例如前边提到的索引可用于查找姓在Allen和Barrymore之间的人。这里也只使用了索引的第一列。
* 精确匹配某一列并范围匹配另外一列：前边提到的索引也可用于查找所有姓为Allen，并且名字是字母K开头(比如Kim,Karl等)的人。即第一列lastname全匹配，第二列firstname范围匹配。

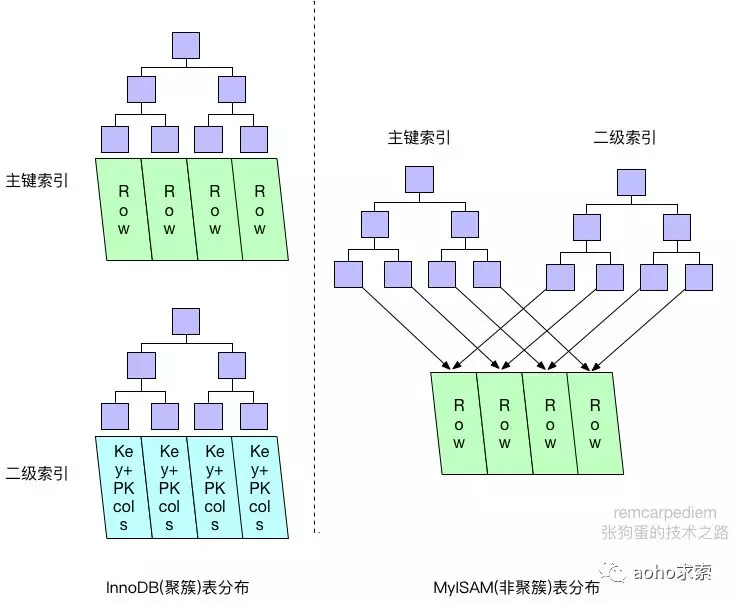
下面是一些关于B-Tree索引的限制：

* 如果不是按照索引的最左列开始查找，则无法使用索引。例如上面例子中的索引无法查找名字为Bill的人，也无法查找某个特定生日的日，因为这两列都不是最左数据列。
* 如果查询中有某个列的范围查询，则其右侧所有列都无法使用索引优化查找。

下图展示了MyISAM和InnoDB的不同索引和数据存储方式：

* MyISAM的数据分布非常简单，按照数据插入的顺序存储在磁盘上，主键索引和二级索引的叶节点存储着指针，指向对应的数据行。
* InnoDB中，聚簇索引“就是”表，所以不会像MyISAM那样需要独立的行存储。聚簇索引的每个叶节点都包含了主键值和所有的剩余列(在此例中是col2)。
* InnoDB的二级索引和聚簇索引很不同。InnoDB二级索引的叶节点中存储的不是“行指针”，而是主键值，并以此作为指向行的“指针”。

InnoDB和MyISAM的索引区别



松散索引扫描

MySQL并不支持松散索引扫描，也就是无法按照不连续的方式扫描一个索引。通常，MySQL的索引扫描需要先定义一个起点和终点，即使需要的数据只是这段索引中很少数的几个，MySQL仍然需要扫描这段索引中的每个条目。

MySQL 5.0之后的版本，在某些特殊的场景下是可以使用松散索引扫描的，例如，在一个分组查询中需要找到分组的最大值和最小值：

覆盖索引

索引除了是一种查找数据的高效方式之外，也是一种列数据的直接获取方式。MySQL可以使用索引来直接获取列的数据，这样就不需要读取数据行。如果一个索引包含所有需要查询的字段的值，我们就称之为“覆盖索引”。

* 覆盖索引是非常有用的工具，能够极大地提高性能。SQL查询只需要扫描索引而无需回表，会带来很多好处：
* 索引条目数量和大小通常远小于数据行的条目和大小，所以如果只需要读取索引，那么MySQL就会极大地减少数据访问量。
* 因为索引是按照列顺序存储的，所以对于I/O密集型的范围查找会比随机从磁盘读取每一行数据的I/O要少的多。

由于InnoDB的聚簇索引，覆盖索引对InnoDB表特别有用。InnoDB的二级索引在叶子节点中保存了行的主键，索引如果二级主键能够覆盖查询，则避免对主键索引的第二次查询。

索引查询的数据主要受限于硬盘的I/O速度，查询I/O次数越少，速度越快，所以B树的结构才应需求而生；B树的每个节点的元素可以视为一次I/O读取，树的高度表示最多的I/O次数，在相同数量的总元素个数下，每个节点的元素个数越多，高度越低，查询所需的I/O次数越少；假设，一次硬盘一次I/O数据为8K，索引用int(4字节)类型数据建立，理论上一个节点最多可以为2000个元素，2000\*2000\*2000=8000000000，80亿条的数据只需3次I/O（理论值），可想而知，B树做为索引的查询效率有多高；另外也可以看出同样的总元素个数，查询效率和树的高度密切相关。

B+树的特征：

有m个子树的中间节点包含有m个元素（B树中是k-1个元素），每个元素不保存数据，只用来索引；

所有的叶子结点中包含了全部关键字的信息，及指向含有这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大的顺序链接。 (而B 树的叶子节点并没有包括全部需要查找的信息)；

所有的非终端结点可以看成是索引部分，结点中仅含有其子树根结点中最大（或最小）关键字。 (而B 树的非终节点也包含需要查找的有效信息)；

为什么说B+树比B树更适合数据库索引？

1）B+树的磁盘读写代价更低

　　B+树的内部结点并没有指向关键字具体信息的指针。因此其内部结点相对B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多。相对来说IO读写次数也就降低了；

2）B+树查询效率更加稳定

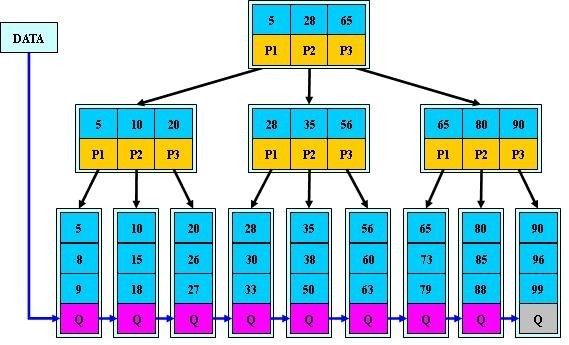
　　由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当；

3）B+树便于范围查询（最重要的原因，范围查找是数据库的常态）

　　B树在提高了IO性能的同时并没有解决元素遍历效率低下的问题，正是为了解决这个问题，B+树应用而生。B+树只需要去遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而B树不支持这样的操作或者说效率太低。

补充：B树的范围查找用的是中序遍历，而B+树用的是在链表上遍历；

B+树如下：



**8.权限管理和备份**

SQL命令操作

用户表: mysql.user

-- 创建用户（CREATE USER 用户名 IDENTIFIED BY '密码'）

CREATE USER Xiangtai IDENTIFIED BY '123456'

-- 修改当前用户密码

SET PASSWORD = PASSWORD('111111')

-- 修改指定用户密码

SET PASSWORD FOR Xiangtai=PASSWORD('111222')

-- 用户重命名

RENAME USER Xiangtai TO Xiangtai1

-- [ALL PRIVILEGES]授予用户(Xiangtai1)对所有(库.表)的(所有权限),

除了GRANT权限

GRANT ALL PRIVILEGES ON \*.\* TO Xiangtai1

-- 查询用户权限

SHOW GRANTS FOR Xiangtai1

SHOW GRANTS FOR root@localhost

Revoke vt.撤销，取消; 废除;

-- 撤销权限：REVOKE 权限 ON 库.表 FROM 用户

REVOKE ALL PRIVILEGES ON \*.\* FROM Xiangtai1

--删除用户

DROP USER Xiangtai1

**Mysql备份**

为什么要备份：

* 保证重要的数据不丢失
* 数据转移

MySQL数据库备份方式：

* 直接拷贝data物理文件
* 在Navicat这种可视化工具中手动导出：

数据库右击->转储SQL文件->结构和数据

* 使用命令行导出 mysqldump 命令行使用

Mysqldump -h 主机 -u 用户名 -p 密码 数据库名 [表名] >磁盘/文件名.sql

命令行导入：切到mysql>use database -> source sql文件地址



**9.规范数据库设计**

为什么需要数据规范化？

* 信息重复
* 更新异常
* 插入异常：无法正常显示信息
* 删除异常：丢失有效的信息

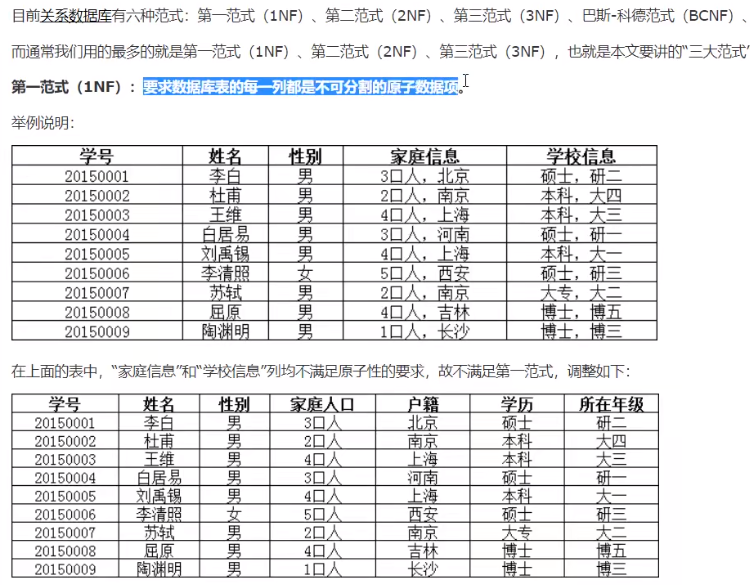
软件开发中，关于数据库的设计：

* 分析需求：分析业务和需要处理的数据库的需求
* 概要设计：设计关系图（E-R图）

三大范式：

第一范式（1NF）：要求数据库表的每一列都是不可分割的原子数据项。

原子性：保证每一列不可再分。



第二范式（2NF）

前提：满足第一范式，

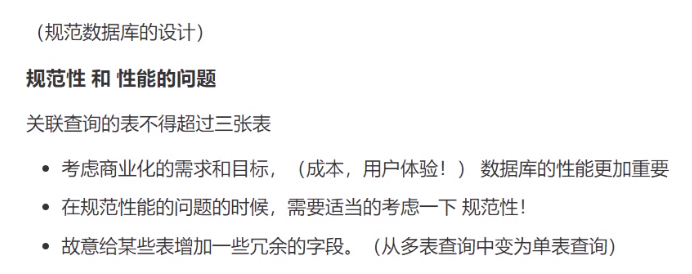
每张表只描述一件事情



第三范式（3NF）

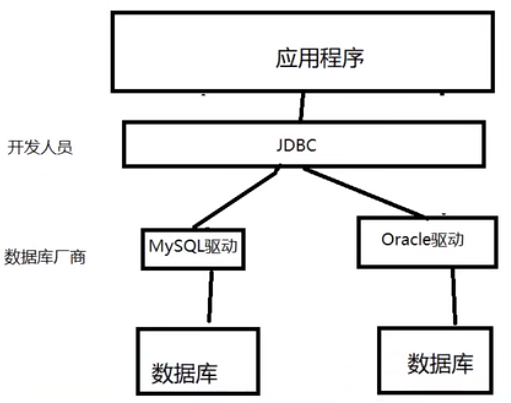
前提：满足第一范式和第二范式





**10.JDBC**

数据库驱动



Java.sql

Javax.sql

还需要导入一个数据库驱动包 mysql-connector-java-版本.jar

1. idea创建一个普通项目
2. 导入数据库驱动（在项目下新建lib目录，将jar包复制到lib目录）
3. 右键lib目录选择 add as library

编写测试代码：

|  |
| --- |
| package com.xiangtai.study1; import java.sql.\*; *//第一个JDBC程序* public class JdbcFirstDemo {  public static void main(String[] args) throws ClassNotFoundException, SQLException {  *//1. 加载驱动*  *//DriverManager.registerDriver(new com.mysql.jdbc.Driver()); 建议使用下面这种* Class.*forName*("com.mysql.jdbc.Driver"); *//固定写法，加载驱动  //2. 用户信息和url* String url = "jdbc:mysql://localhost:3306/jdbcstudy?useUnicode=true&characterEncoding=utf8&useSSL=true";  String username = "root";  String password = "123456";  *//3. 连接成功，返回数据库对象Connection，Connection代表数据库* Connection connection = DriverManager.*getConnection*(url, username, password);  *//4. 执行SQL的对象（Statement） 执行sql的对象* Statement statement = connection.createStatement();  *//5. 用SQL对象去执行SQL，可能存在结果，查看返回结果* String sql = "SELECT \* FROM users";  ResultSet resultSet = statement.executeQuery(sql); *//返回的结果集，结果集中封装了执行sql语句的结果* while (resultSet.next()){  System.*out*.println("id=" + resultSet.getObject("id"));  System.*out*.println("name=" + resultSet.getObject("name"));  System.*out*.println("pwd=" + resultSet.getObject("password"));  System.*out*.println("email=" + resultSet.getObject("email"));  System.*out*.println("birthday=" + resultSet.getObject("birthday"));  System.*out*.println("==============================");  }  *//6. 释放连接* resultSet.close();  statement.close();  connection.close();  } } |

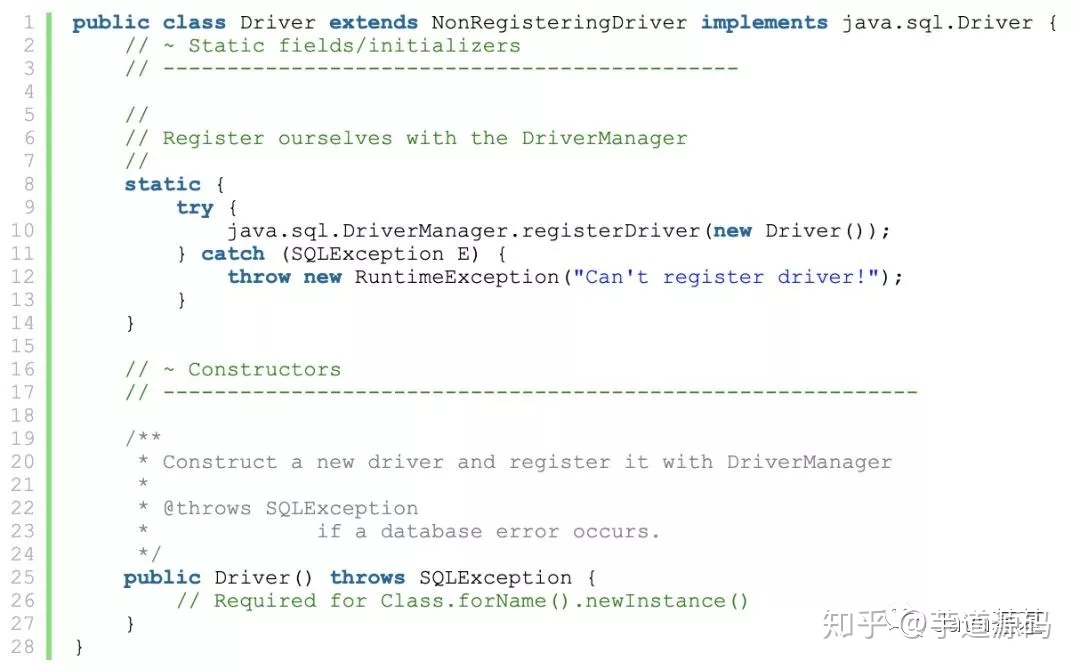
步骤：

1. 加载驱动
2. 连接数据库DriverManager
3. 获得执行sql的对象Statement（不安全）
4. 获得返回的结果集
5. 释放连接 .close()

DriverManage->DriverManage.getConnection(url,user,pwd)->connection.createStatement->statement.execute(sql)-> .close()

使用JDBC时通常是使用Class.forName()方法来加载数据库连接驱动。这是因为在JDBC规范中明确要求Driver(数据库驱动)类必须向DriverManager注册自己。

以MySQL的驱动为例解释：



我们看到Driver注册到DriverManager中的操作写在了静态代码块中，这就是为什么在写JDBC时使用Class.forName()的原因了。

Java静态代码块的作用：Java静态代码块中的代码会在类加载JVM时运行，且只被执行一次，也就是说这些代码不需要实例化类就能够被调用。一般情况下,如果有些代码必须在[项目启动](https://wenwen.sogou.com/s/?w=%E9%A1%B9%E7%9B%AE%E5%90%AF%E5%8A%A8&ch=ww.xqy.chain" \t "_blank)的时候就执行的时候,就需要使用静态代码块。

关键字:

DriverManger

Class.*forName*("com.mysql.jdbc.Driver"); *//固定写法，加载驱动*

Connection connection = DriverManager.*getConnection*(url, username, password);

//connection代表数据库，可以像数据库那样设置自动提交、事务提交、事务回滚(connection.rollback(),connection.commit(),connection.setAutoCommit)

URL

//mysql默认端口号3306

String url = "jdbc:mysql://localhost:3306/jdbcstudy?useUnicode=true&

characterEncoding=utf8&useSSL=true";

// 协议: ://主机地址:端口号/数据库名？参数1&参数2&参数3

//Oracle 默认端口号1521

//jdbc:oracle:thin:@localhost:1521:sid

String sql = “SELECT \* FROM users”;

Statement执行SQL的对象 PrepareStatement执行SQL的对象

Statement.excuteQuery(sql); //执行查询操作，返回结果集ResultSet

Statement.execute(); //可以执行任何SQL语句

Statement.executeUpdate(); //更新、插入、删除都使用这个，返回受影响的行数

Statement.executeBatch(); //执行多条SQL语句

String sql = “update user set name = ‘’ where name=’’ “;

Int num = statement.executeUpdate(sql);

If(num>0){

System.out.println(“修改成功！”);

}

ResultSet查询的结果集，封装了所有的查询结果，指针遍历

ResultSet.get+”列类型”();

不知道类型时使用resultSet.getObject();

resultSet.beforeFirst(); //移动到最前面

resultSet.afterLast(); //移动到最后面

resultSet.next(); //移动到下一个

resultSet.previous(); // 移动到前一行

resultSet.absolute(int row); //移动到指定行

释放连接（资源）

resultSet.close();

statement.close();

connection.close();

**Statement对象**

Jdbc中的statement对象用于向数据库发送SQL语句，要完成对数据库的增删改查，只需要通过这个对象向数据库发送增删改查语句即可。

获取Class实例的三种方式：

1.调用对象的getClass()方法获取该对象的Class实例；

2.使用Class类的静态方法forName("类名")获取该类的Class实例；

3.使用类的属性.class来获取Class实例，对于基本数据类型的封装类，还可以采用.TYPE来获取对应的基本数据类型的Class实例。

代码实现：

1.提取工具类

2.编写增删改操作(使用statement.executeUpdate)

public static void main(String[] args){  
  
 Connection cnn = null;  
 Statement stt = null;  
 ResultSet rss = null;  
  
 try {  
 cnn = jdbcUtils.*getConnection*(); *//获取数据库连接* stt = cnn.createStatement(); *//获得SQL的执行对象* String sql = "INSERT INTO `users`(id,`name`,`password`,`email`,`birthday`)" +  
 "VALUES(5,'Li Ye','555666','223446622@qq.com','2021-08-16')";  
 int i = stt.executeUpdate(sql);  
 if(i > 0){  
 System.*out*.println("执行插入成功！！");  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(cnn,stt,rss);  
 }  
}

**//删除**

public static void main(String[] args){  
  
 Connection cnn = null;  
 Statement stt = null;  
 ResultSet rss = null;  
  
 try {  
 cnn = jdbcUtils.*getConnection*(); *//获取数据库连接* stt = cnn.createStatement(); *//获得SQL的执行对象* String sql = "DELETE FROM `users` WHERE id=5";  
 int i = stt.executeUpdate(sql);  
 if(i > 0){  
 System.*out*.println("执行删除成功！！");  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(cnn,stt,rss);  
 }  
}

**//修改**

public static void main(String[] args){  
  
 Connection cnn = null;  
 Statement stt = null;  
 ResultSet rss = null;  
  
 try {  
 cnn = jdbcUtils.*getConnection*(); *//获取数据库连接* stt = cnn.createStatement(); *//获得SQL的执行对象* String sql = "UPDATE `users` SET `name`='wangxiangtai',`email`='2714661705@qq.com' WHERE id=4";  
 int i = stt.executeUpdate(sql);  
 if(i > 0){ *//执行更新操作返回受影响的行数* System.*out*.println("执行更新成功！！");  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(cnn,stt,rss);  
 }  
}

**//查询**

public static void main(String[] args){  
 Connection conn = null;  
 Statement st = null;  
 ResultSet rs = null;  
 try {  
 conn = jdbcUtils.*getConnection*();  
 st = conn.createStatement();  
 String sql = "SELECT \* FROM `users` WHERE id=4";  
 rs = st.executeQuery(sql);  
 while (rs.next()){  
 System.*out*.println("id=" + rs.getObject("id"));  
 System.*out*.println("name=" + rs.getObject("name"));  
 System.*out*.println("pwd=" + rs.getObject("password"));  
 System.*out*.println("email=" + rs.getObject("email"));  
 System.*out*.println("birthday=" + rs.getObject("birthday"));  
 System.*out*.println("==============================");  
  
 }  
  
  
 }catch (Exception e){  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, st, rs);  
 }

**SQL注入的问题：**

Sql存在漏洞，会被攻击导致数据泄露。SQL会被拼接

package com.xiangtai.study1;  
  
import com.xiangtai.stydy2.utils.jdbcUtils;  
  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.Statement;  
  
public class SQL注入 {  
 public static void main(String[] args){  
*// login("wangxiangtai","555555");  
 login*("' or '1=1","' or '1=1"); *//SQL注入* }  
 public static void login(String username,String password){  
 Connection conn = null;  
 Statement st = null;  
 ResultSet rs = null;  
 try {  
 conn = jdbcUtils.*getConnection*();  
 st = conn.createStatement();  
 *//SELECT \* FROM `users` WHERE `name` = 'wangxiangtai' AND `password` = '555555';  
 //SELECT \* FROM `users` WHERE `name` = '' or '1=1' AND `password` = '' or '1=1'; SQL注入* String sql = "SELECT \* FROM `users` WHERE `name`='"+username+"' AND `password`='"+password+"'";  
 rs = st.executeQuery(sql); *//查询完毕返回一个结果集* while (rs.next()){  
 System.*out*.println("id=" + rs.getObject("id"));  
 System.*out*.println("name=" + rs.getObject("name"));  
 System.*out*.println("pwd=" + rs.getObject("password"));  
 System.*out*.println("email=" + rs.getObject("email"));  
 System.*out*.println("birthday=" + rs.getObject("birthday"));  
 System.*out*.println("======================");  
 }  
 }catch (Exception e){  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, st, rs);  
 }  
 }  
  
}

**更安全的PreparedStatement执行对象:**

PreparedStatement对象可以防止SQL注入，且效率更高

**//修改**

public class TestSQLUpdate {  
 public static void main(String[] args) {  
 Connection conn = null;  
 PreparedStatement pst = null;  
 try {  
 conn = jdbcUtils.*getConnection*();  
 *//区别  
 //使用？ 占位符代替参数* String sql = "update `users` set `name`=? where id=?";  
 *//预编译SQL，先写sql，不执行* pst = conn.prepareStatement(sql);  
  
 *//手动给参数赋值* pst.setString(1,"老六");  
 pst.setInt(2,1);  
 *//执行* int i = pst.executeUpdate();  
 if(i>0){  
 System.*out*.println("执行更新成功！！！");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, pst,null);  
 }  
 }  
}

**//增加**

public class TestSQLInsert {  
 public static void main(String[] args){  
  
 Connection cnn = null;  
 Statement stt = null;  
 ResultSet rss = null;  
  
 try {  
 cnn = jdbcUtils.*getConnection*(); *//获取数据库连接* stt = cnn.createStatement(); *//获得SQL的执行对象* String sql = "INSERT INTO `users`(id,`name`,`password`,`email`,`birthday`)" +  
 "VALUES(5,'Li Ye','555666','223446622@qq.com','2021-08-16')";  
 int i = stt.executeUpdate(sql);  
 if(i > 0){  
 System.*out*.println("执行插入成功！！");  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(cnn,stt,rss);  
 }  
 }  
  
}

**//删除**

public class TestSQLDelete {  
 public static void main(String[] args) {  
 Connection conn = null;  
 PreparedStatement pst = null;  
 try {  
 conn = jdbcUtils.*getConnection*();  
 *//区别  
 //使用？ 占位符代替参数* String sql = "delete from `users` where id=?";  
 *//预编译SQL，先写sql，不执行* pst = conn.prepareStatement(sql);  
  
 *//手动给参数赋值* pst.setInt(1,5);  
  
 *//执行* int i = pst.executeUpdate();  
 if(i>0){  
 System.*out*.println("执行删除成功！！！");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, pst,null);  
 }  
 }  
  
}

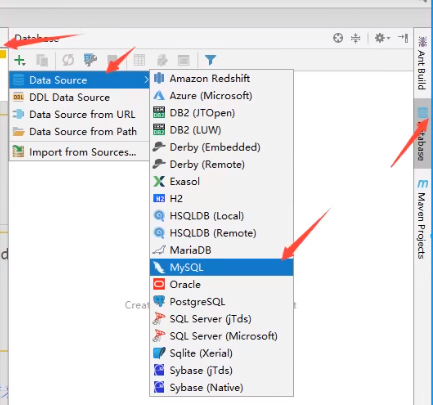
**//查找**

public class TestSQLSelect {  
 public static void main(String[] args) {  
 Connection conn = null;  
 PreparedStatement pst = null;  
 ResultSet rs = null;  
  
 try {  
 conn = jdbcUtils.*getConnection*();  
 String sql = "select \* from users where id = ?";  
 pst = conn.prepareStatement(sql);  
 pst.setInt(1,1);  
 rs = pst.executeQuery();  
 if(rs.next()){  
 System.*out*.println("name=" + rs.getString("name"));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, pst, rs);  
 }  
 }  
  
}

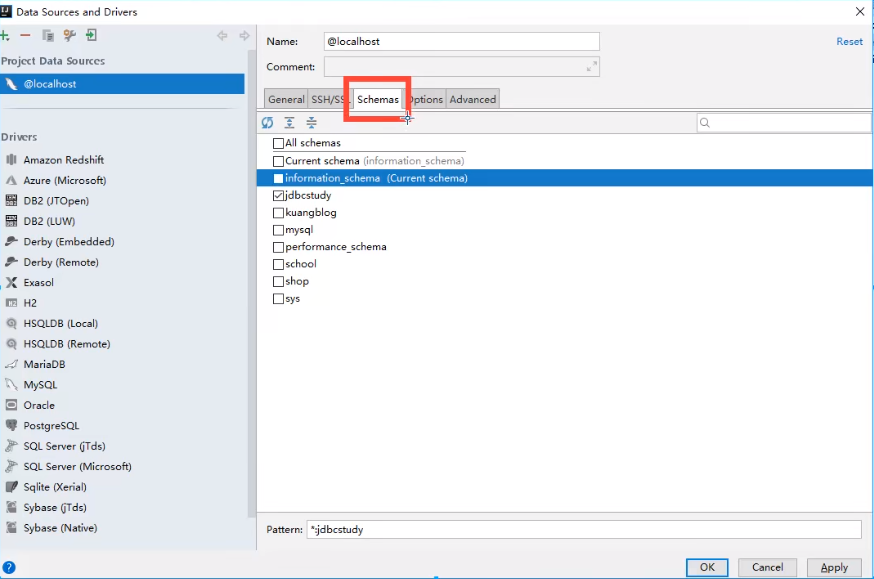
**//防止SQL注入**

public class SQL注入 {  
 public static void main(String[] args){  
 *//login("wangxiangtai","555555");  
 login*("'' or 1=1","123456"); *//SQL注入* }  
  
 public static void login(String username,String password){  
 Connection conn = null;  
 PreparedStatement st = null;  
 ResultSet rs = null;  
 try {  
 conn = jdbcUtils.*getConnection*();  
 *//PreparedStatement防止SQL注入的本质：把传递进来的参数当作字符  
 //若其中存在转义字符，如’（引号）会被直接转义* String sql = "select \* from users where `name`=? AND `password`=?";  
 st = conn.prepareStatement(sql);  
 st.setString(1,username);  
 st.setString(2,password);  
 rs = st.executeQuery(); *//查询完毕返回一个结果集* while (rs.next()){  
 System.*out*.println("id=" + rs.getObject("id"));  
 System.*out*.println("name=" + rs.getObject("name"));  
 System.*out*.println("pwd=" + rs.getObject("password"));  
 System.*out*.println("email=" + rs.getObject("email"));  
 System.*out*.println("birthday=" + rs.getObject("birthday"));  
 System.*out*.println("======================");  
 }  
  
  
 }catch (Exception e){  
 e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn, st, rs);  
 }  
 }  
}

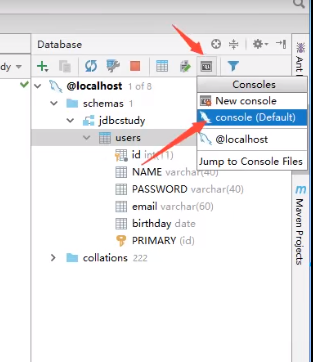
使用IDEA连接数据库



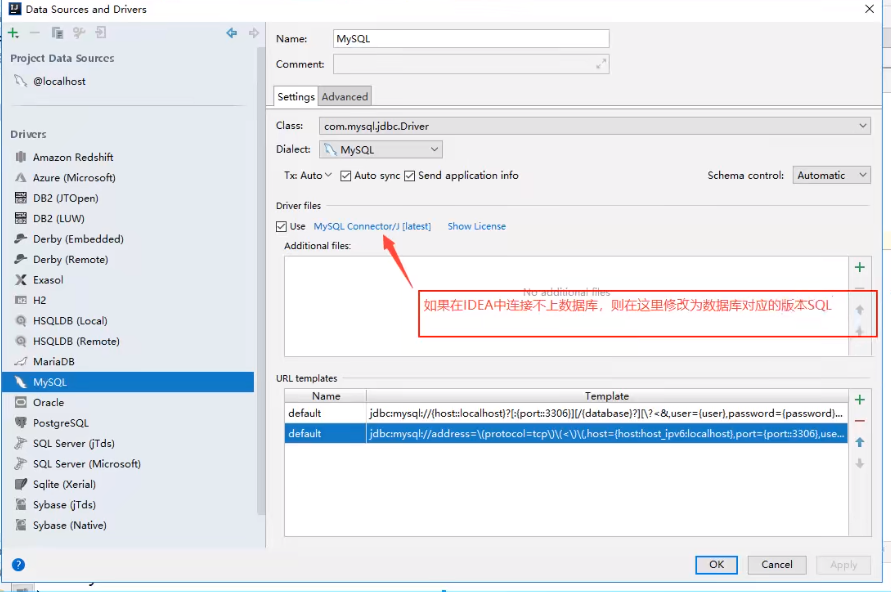
连接成功之后，可以选择数据库



打开sql语言编辑界面



版本问题



**事务**

ACID原则

public class transaction1 {  
 public static void main(String[] args) {  
 Connection conn = null;  
 PreparedStatement pst = null;  
 ResultSet rs = null;  
  
 try {  
 conn = jdbcUtils.*getConnection*();  
 *//关闭数据库自动提交事务，并开启事务* conn.setAutoCommit(false);  
 String sql1 = "update account set money=money-100 where name='A'";  
 pst = conn.prepareStatement(sql1);  
 pst.executeUpdate();  
  
 String sql2 = "update account set money=money+100 where name='B'";  
 pst = conn.prepareStatement(sql2);  
 pst.executeUpdate();  
  
 *//业务完毕，提交事务* conn.commit();  
 System.*out*.println("事务执行成功！！！");  
 } catch (SQLException e) {  
*// try {//失败会默认回滚，不写也行  
// conn.rollback(); //失败，则回滚事务  
// } catch (SQLException throwables) {  
// throwables.printStackTrace();  
// }* e.printStackTrace();  
 }finally {  
 jdbcUtils.*release*(conn,pst,rs);  
 }  
 }  
}

**数据库连接池**

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由最小数据库连接数制约。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的最大数据库连接数量限定了这个连接池能占有的[最大连接数](https://baike.baidu.com/item/%E6%9C%80%E5%A4%A7%E8%BF%9E%E6%8E%A5%E6%95%B0" \t "_blank)，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。

连接池基本的思想是在[系统初始化](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E5%88%9D%E5%A7%8B%E5%8C%96" \t "_blank)的时候，将数据库连接作为对象存储在内存中，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接对象。使用完毕后，用户也并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。而连接的建立、断开都由连接池自身来管理。同时，还可以通过设置连接池的参数来控制连接池中的初始连接数、连接的上下限数以及每个连接的最大使用次数、最大空闲时间等等。也可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

数据库连接—执行完毕—释放

连接—释放 十分浪费系统资源

池化技术：预先准备一些连接对象，需要新的连接时，可以直接分配连接对象，不用重新请求连接。

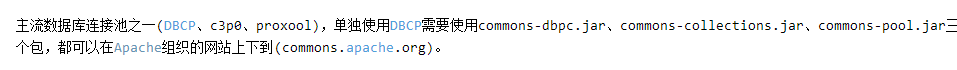
编写连接池，实现一个接口DataSource

数据库连接池开源数据源实现；

DBCP

C3P0

Druid（德鲁伊）：阿里巴巴的



使用了这些数据库连接池之后，在项目开发中就不需要编写连接数据库的代码了

DBCP:

配置文件jdbcconfig.properties,连接类：jdbcUtils\_DBCP.java

Private static DataSource dataSource = null;

InputStream in jdbcUtils\_DBCP.class.getClassLoader().getResourceAsStream(“dbcpconfig.properties”);

Properties properties = new Properties();

Properties.load(in);

//创建数据源 工厂模式—>创建

dataSource = BasicDataSourceFactory.createDataSource(properties);

数据源定义的是连接到实际数据库的一条路径而已，数据源中并无真正的数据，它仅仅记录的是你连接到哪个数据库，以及如何连接的，如odbc数据源。也就是说数据源仅仅是数据库的连接名称，一个数据库可以有多个数据源连接

C3P0:

dataSource = new ComboPooleDataSource(configName:”配置名”);//不填参数有默认参数，也可以指定自己定义的配置。

或者不用配置文件的写法：

Private static ComboPooledDataSource dataSource = null;

dataSource = new ComboPooledDataSource();

dataSource.setDriverClass();

dataSource.setUser();

dataSource.setPassword();

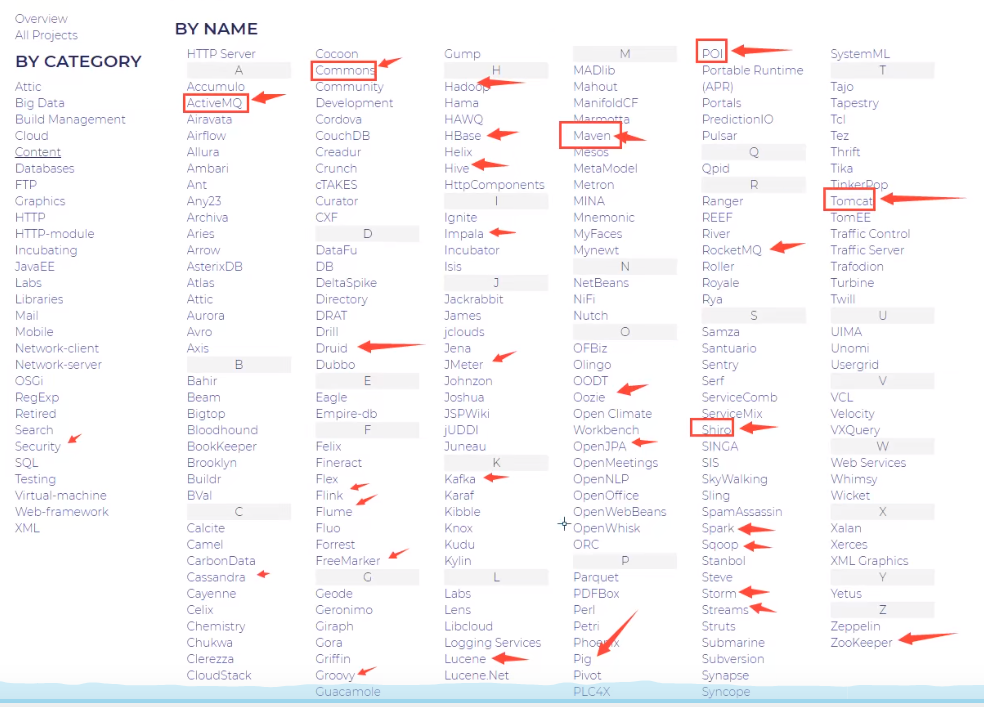
dataSource.setJdbcUrl();

dataSource.setMaxPoolSize();

dataSource.setMinPoolSize();

**结论**

无论使用什么数据源，本质还是一样的，DataSource接口不会变，方法就不会变



红色框是要学的

DAO：Data Access Object 数据访问对象

ORM：Object Relationship Database Mapping 对象和关系数据库的映射（一个对象对应数据库中的一条记录）