

# 什么是SQL

Structured Query Language:结构化查询语言  
其实就是定义了操作所有关系型数据库的规则.  
每一种数据库操作的方法存在不一样的地方,称为"方言"

## 三种注释

单行注释  
-- 注释内容  
#注释内容(mysql特有)  
多行注释  
/\* 注释 \*/

# 数据表的类型

## 设置数据表的类型

```
1 CREATE TABLE 表名(  
2     -- 省略一些代码  
3     -- Mysql注释  
4     -- 1. # 单行注释  
5     -- 2. /*...*/ 多行注释  
6 )ENGINE = MyISAM (or InnoDB)  
7  
8 -- 查看mysql所支持的引擎类型 (表类型)  
9 SHOW ENGINES;
```

MySQL的数据表的类型: **MyISAM**, **InnoDB**, HEAP, BOB, CSV等...  
常见的 MyISAM 与 InnoDB 类型:

名称	MyISAM	InnoDB
事务处理	不支持	支持
数据行锁定	不支持	支持
外键约束	不支持	支持
全文索引	支持	支持
表空间大小	较小	较大,约 2 倍






经验 (适用场合):

- 适用 MyISAM: 节约空间及相应速度
- 适用 InnoDB: 安全性, 事务处理及多用户操作数据表数据表的存储位置

## 数据表的存储位置

- MySQL数据表以文件方式存放在磁盘中
  - 包括表文件, 数据文件, 以及数据库的选项文件
  - 位置: Mysql安装目录\data\下存放数据表. 目录名对应数据库名, 该目录下文件名对应数据表.
- 注意:

- \*.frm -- 表结构定义文件
  - \*.MYD -- 数据文件 ( data )
  - \*.MYI -- 索引文件 ( index )
- InnoDB类型数据表只有一个 \*.frm文件, 以及上一级目录的ibdata1文件
- MyISAM类型数据表对应三个文件:

 innodb.frm	2021/6/26 22:00	FRM 文件	9 KB
 innodb.ibd	2021/6/26 22:00	IBD 文件	96 KB
 myisam.frm	2021/6/26 21:59	FRM 文件	9 KB
 myisam.MYD	2021/6/26 21:59	MYD 文件	0 KB
 myisam.MYI	2021/6/26 21:59	MYI 文件	1 KB

## 设置数据表字符集

我们可为数据库,数据表,数据列设定不同的字符集, 设定方法:

- 创建时通过命令来设置, 如: CREATE TABLE 表名()CHARSET = utf8;
- 如无设定, 则根据MySQL数据库配置文件 my.ini 中的参数设定

## SQL分类

### DDL(Data Definition Language)

数据定义语言 操作数据库,表

用来定义数据库对象

数据库

表

列

**关键字**

create

drop

alter

### DML(Data Manipulation Language)

数据操作语言

增删改表中的数据

用来对数据库中表的数据进行增删改

**关键字**

insert

delete

update

### DQL(Data Query Language)

数据查询语言

查询表中的数据

用来查询数据库中表的数据

**关键字**

select

where

### DCL(Data Control Language)

数据控制语言

授权

用来定义数据库的访问权限和安全级别,及创建用户

**关键字**

grant  
revoke

## DDL

### 操作数据库:CRUD

#### C(Create)

创建数据库

```
create database 数据库名字  
create database if not exists 数据库名称  
create database 数据库名称 character set 字符集名
```

```
1 # 如果不存在db1数据库,那就创建db1数据库,字符集为gbk  
2 create database if not exists db1 character set gbk;
```

#### R(Retrieve)

查询所有数据库名

```
show databases;
```

查询某个数据库的字符集

```
show create database 数据库名
```

#### U(Update)

修改数据库的字符集

```
alter database 数据库名 character set 字符集名称
```

#### D(Delete)

删除数据库

```
drop database 数据库名  
drop database if exists 数据库名
```

#### 使用数据库

查询当前正在使用的数据库名

```
select database();
```

使用数据库

```
use 数据库名
```

### 操作表

#### C(Create)

```
1 create table 表名(  
2     列名1 数据类型1,  
3     列名2 数据类型2,  
4     .....  
5     列名n 数据类型n          #注意:最后一列,不需要加逗号,  
6 );
```

#### 复制表

```
create table 表名 like 被复制的表名
```

## 数据库类型

### 1.int 整数类型

age int;

### 2.double 小数类型

score double(5,2)  
整数部分5位,小数点后两位

### 3.date 日期,只包含年月日

yyyy-MM-dd

### 4.datetime日期,包含年月日时分秒

yyyy-MM-dd HH:mm:ss

### 5.timestamp:时间戳类型,包含年月日时分秒

yyyy-MM-dd HH:mm:ss  
如果将来不给这个字段赋值,或赋值为null,则默认使用当前的系统时间来自动赋值

### 6 varchar:字符串

name varchar(20):姓名最大20个字符  
汉字为一个字符

## R(Retrieve)

查询某个数据库中的所有的表

show tables

查看表的结构

desc 表名

查询某个表的字符集

show create table 表名

## U(Update)

### 1.修改表名

alter table 表名 rename to 新的表名

### 2.修改表的字符集

alter table 表名 character set 字符集名称

### 3.添加一列

alter table 表名 add 列名 数据类型

### 4.修改列名称, 类型

alter table 表名 change 列名 新列名 新数据类型

alter table 表名 modify 列名 新数据类型

### 5.删除列

alter table 表名 drop 列名

## D(Delete)

drop table 表名

drop table if exists 表名

# DML

## 1.添加数据

insert into 表名(列名1,列名2,...列名n) values (值1,值2,...值n)

## 注意:

- 1.列名和值要一一对应
- 2.如要表名后,不定义列名,则默认给所有列添加值  
insert into 表名 values(值1,值2,...值n)
- 3.除了数字类型,其他类型需要使用引号(单双都可以)引起来

## 2.删除数据

```
delete from 表名 [where 条件]
```

### 注意

- 1.如果不加条件,则删除表中所有记录
- 2.如果要删除所有记录  
delete from 表名  
不推荐使用,有多少条记录就会执行多少次删除操作  
**truncate** table 表名;  
先删除表,然后再创建一张一样的表

## 3.修改数据

```
update 表名 set 列名1 = 值1,列名2 = 值2,...[where 条件]
```

### 注意

如果不加任何条件,则会将表中所有记录全部修改

# DQL

```
select * from 表名;
```

## 1.语法

```
1 select
2     字段列表
3 from
4     表名列表
5 where
6     条件列表
7 group by
8     分组字段
9 having
10    分组之后的条件
11 order by
12    排序
13 limit
14    分页限定
```

## 2.基础查询

### 1.多个字段的查询

```
select 字段名1,字段名2....from 表名;
```

```
1 | SELECT NAME,sex FROM student WHERE address = '红灯区';
```

## 2.去除重复 distinct

```
1 | SELECT DISTINCT address FROM student;
```

## 3.计算列

一般可以使用四则运算计算一些列的值(一般只会进行数值型的计算)

ifnull(表达式1,表达式2):null参与的运算,计算结果都为null

表达式1:判断该字段的值是否为null

表达式2:若为null,就用表达式2替换

```
1 | SELECT DISTINCT NAME,math,english ,math+IFNULL(english,0) FROM student;
```

## 4.起别名

as:as也可以省略

```
1 | SELECT DISTINCT NAME,math AS 数学,english 英语,math+IFNULL(english,0) AS 总分 FROM student;
```

## 3.条件查询

### 1.where子句后跟条件

#### 运算符

- ①<,>,<=,>=,=,<>
- ②BETWEEN 条件1 AND 条件2  
包括条件值, 条件1<= and <= 条件2
- ③IN (集合)
- ④列名 IS NULL  
NULL值不能使用 !=判断  
列名 IS NOT NULL
- ⑤条件1 AND (&&) 条件2
- ⑥条件1 OR (||) 条件2
- ⑦NOT 条件
- ⑧LIKE模糊查询  
占位符  
\_单个任意字符

```
1 | SELECT * FROM student WHERE NAME LIKE "__老师"; #两个_,名字第三,四个字是老师的学生
```

%多个任意字符

```
1 | SELECT * FROM student WHERE NAME LIKE "%老师%"; #包含老师
```

## 查询语句

## 1.排序查询

语法:order by 子句

order by 排序列1 排序方式,排序列2 排序方式...

ASC:升序,默认

DESC:降序

SELECT \* FROM student ORDER BY math, english;

注意:

如果有多个排序条件,则当前边的条件值一样时,才会判断第二条件

## 2.聚合函数

将一列数据作为一个整体,进行纵向的计算

1.count

2.max

3.min

4.sum

5.avg

select 聚合函数(列名) from 表名;

注意

聚合函数的计算,排除null值

解决方案

1.选择不包含非空的列进行计算

2.IFNULL函数

```
1      select count(IFNULL(math,0)) from student;           #根据数学成绩统计学生人数,数学
      成绩为null的学生,math看成0
2      SELECT COUNT(studentname) FROM student;
3      SELECT COUNT(*) FROM student;
4      SELECT COUNT(1) FROM student; /*推荐*/
5
6      -- 从含义上讲, count(1) 与 count(*) 都表示对全部数据行的查询。
7      -- count(字段) 会统计该字段在表中出现的次数, 忽略字段为null 的情况。即不统计字段为null 的
      记录。
8      -- count(*) 包括了所有的列, 相当于行数, 在统计结果的时候, 包含字段为null 的记录;
9      -- count(1) 用1代表代码行, 在统计结果的时候, 包含字段为null 的记录 。
10
11     -- 1) 在表没有主键时, count(1)比count(*)快
12     -- 2) 有主键时, 主键作为计算条件, count(主键)效率最高;
13     -- 3) 若表格只有一个字段, 则count(*)效率较高
```

## 3.分组查询

GROUP BY 列名

```
1      #根据性别且数学成绩在70分以上的,查询男女的性别,数学平均成绩,该性别总人数,分组之后人数大于2
2      SELECT sex,AVG(math),COUNT(id) 人数 FROM student where math>70 GROUP BY sex HAVING
      人数>2;
```

注意

1.分组之后查询的字段

分组列,

聚合函数

## 2.where和having的区别

- ①where在**分组之前**进行限定,如果不满足条件,则不参与分组  
having在**分组之后**进行限定,如果不满足结果,则不会被查询出来
- ②where后不可以跟聚合函数  
having可以进行聚合函数的判断

## 4.分页查询

limit 开始的索引,每页查询的条数

### 公式

开始的索引 = (当前的页码 - 1) \* 每页显示的条数

limit是MySQL"方言"

```
1 SELECT * FROM student LIMIT 0,3;    -- 第一页
2
3 SELECT * FROM student LIMIT 3,3;    -- 第二页
4
5 SELECT * FROM student LIMIT 6,3;    -- 第三页
```

## 约束

对表中的数据进行限定,保证数据的正确性,有效性和完整性

### 分类

#### 1.主键约束(primary key)

##### 注意

- 1.含义  
非空且唯一
- 2.一张表只能有一个字段为主键
- 3.主键就是表中记录的唯一标识

##### 1.创建表时添加约束

```
1 列名 类型 PRIMARY KEY;
```

##### 2.创建表后,添加非空约束

```
1 ALTER TABLE 表名 MODIFY 列名 数据类型 PRIMARY KEY;
```

##### 3.删除非空约束

```
1 ALTER TABLE 表名 DROP PRIMARY KEY ;
```

### 自动增长

如果某一列是数值类型的,使用AUTO\_INCREMENT可以来完成值的自动增长

##### 1.创建表时添加约束

```
1 列名 类型 PRIMARY KEY AUTO_INCREMENT;
```



## 2.创建表后,添加非空约束

```
1 ALTER TABLE 表名 MODIFY 列名 数据类型  
2 AUTO_INCREMENT;
```

## 3.删除非空约束

```
1 ALTER TABLE 表名 MODIFY 列名 数据类型 ;
```

## 2.非空约束(not null)

### 1.创建表时添加约束

```
1 列名 类型 NOT NULL;
```

### 2.创建表后,添加非空约束

```
1 ALTER TABLE 表名MODIFY 列名 数据类型 NOT NULL
```

### 3.删除非空约束

```
1 ALTER TABLE 表名 MODIFY 列名 数据类型 ;
```

## 3.唯一约束(unique)

### 1.创建表时添加约束

```
1 列名 类型 UNIQUE;
```

### 2.创建表后,添加非空约束

```
1 ALTER TABLE 表名 MODIFY 列名 数据类型 UNIQUE;
```

### 3.删除非空约束

```
1 ALTER TABLE 表名 DROP INDEX 列名 ;
```

## 4.外键约束(foreign key)

让表与表产生关系,从而保证数据的正确性

一般加外键是多表连接,一对一谁加都可以,一对多,多加外键,多对多,派生表加

### 1.创建表时添加约束

```
1 CONSTRAINT 外键名 FOREIGN KEY (本表列名) REFERENCES 外键表(外键表列名)
```

### 2.创建表后,添加非空约束

```
1 ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表列名) REFERENCES 外键表(外键表列名)
```

### 3.删除非空约束

```
1 | ALTER TABLE 表名 DROP FOREIGN KEY 外键名 ;
```

#### 4.级联操作

1.添加级联操作      外键连接的表都随之更改

```
1 | ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表列名) REFERENCES 外键表(外键表列名) ON UPDATE CASCADE ON DELETE CASCADE;
```

#### 2.分类

1.级联更新

```
1 | ON UPDATE CASCADE
```

2.级联删除

```
1 | ON DELETE CASCADE
```

## 多表之间的关系

### 1.分类:

#### 1.一对一(了解):

\*如:人和身份证

\*分析:一个人只有一个身份证,一个身份证只能对应一个人

#### 2.一对多(多对一):

\*如:部门和员工

\*分析:一个部门有多个员工,一个员工只能对应一个部门

#### 3.多对多:

\*如:学生和课程

\*分析:一个学生可以选择很多门课程,一个课程也可以被很多学生选择

### 2.实现关系:

#### 1.-对多(多对一):

\*如:部门和员工

\*实现方式:在**多的一方建立外键**,指向一的一方的主键。

#### 2.多对多:

\*如:学生和课程

\*实现方式:多对多关系实现需要**借助第三张中间表**。中间表至少包含两个字段,这两个字段作为第三张表的外键,分别指向两张表的主键

## MD5 加密

### 一、MD5简介

MD5即Message-Digest Algorithm 5 (信息-摘要算法5), 用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一(又译摘要算法、哈希算法), 主流编程语言普遍已有MD5实现。将数据(如汉字)运算为另一固定长度值, 是杂凑算法的基础原理, MD5的前身有MD2、MD3和MD4。

## 二、实现数据加密

新建一个表 testmd5

```
1 CREATE TABLE `testmd5` (  
2   `id` INT(4) NOT NULL,  
3   `name` VARCHAR(20) NOT NULL,  
4   `pwd` VARCHAR(50) NOT NULL,  
5   PRIMARY KEY (`id`)  
6 ) ENGINE=INNODB DEFAULT CHARSET=utf8
```

插入一些数据

```
1 INSERT INTO testmd5 VALUES(1,'kuangshen','123456'),(2,'qinjiang','456789')
```

如果我们要对pwd这一列数据进行加密，语法是：

```
1 update testmd5 set pwd = md5(pwd);
```

如果单独对某个用户(如kuangshen)的密码加密：

```
1 INSERT INTO testmd5 VALUES(3,'kuangshen2','123456')  
2 update testmd5 set pwd = md5(pwd) where name = 'kuangshen2';
```

插入新的数据自动加密

```
1 INSERT INTO testmd5 VALUES(4,'kuangshen3',md5('123456'));
```

查询登录用户信息（md5对比使用，查看用户输入加密后的密码进行比对）

```
1 SELECT * FROM testmd5 WHERE `name`='kuangshen' AND pwd=MD5('123456');
```

## 范式

### 概念

设计数据库时，需要遵循的一些**规范**。要遵循后边的范式要求，必须先遵循前边的所有范式要求

设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库,这些不同的规范要求被称为不同的范式，各种范式呈递次规范，越高的范式数据库冗余越小。.

目前关系数据库有六种范式:第一范式(1NF)、第二范式(2NF)、第三范式(3NF)、巴斯-科德范式(BCNF)、第四范式(4NF)和第五范式(5NF,又称完美范式)。

### 第一范式(1NF)

每一列都是不可分割的原子数据项

### 第二范式(2NF)

在1NF的基础上,非码属性必须**完全依赖于候选码**(在1NF基础上消除非主属性对主码的**部分函数依赖**)

## 1.函数依赖

A-->B,如果通过A属性(属性组)的值,可以**确定唯一**B属性的值.则称B依赖于A

\*学号-->姓名

一个学号确定一个姓名

\*(学号,课程名称) --> 分数

一个学号和一个课程名称,确定一个分数

## 2.完全函数依赖

A --> B,如果A是一个**属性组**,则B属性值的确定需要依赖于A属性组中**所有的属性值**

\*(学号,课程名称) --> 分数

一个学号和一个课程名称,确定一个分数

## 3.部分函数依赖

A --> B,如果A是一个**属性组**,则B属性值的确定只需要依赖于A属性组中的**某一些值**即可

\*(学号,课程名称) --> 姓名

该属性组中的一个学号就能确定一个姓名

## 4.传递函数依赖

A --> B, B --> C.如果通过A属性(属性组)的值,可以**确定唯一**B属性的值,在通过B属性(属性组)的值可以**确定唯一**C属性的值,则称C传递函数依赖于A

学号 --> 系名,系名 --> 系主任

## 5.码

如果一张表中,一个属性或属性组,被其他**所有属性所完全依赖**,则称这个属性(属性组)为该表的**码**

(学号,课程名称)

一个学号能确定一部分属性值,一个课程名称能确定剩下一部分属性值,一个学号又不能确定一个课程名称

### 主属性:

码属性组中的所有属性

### 非主属性

除了码属性组的属性

## 第三范式(3 NF)

在2NF基础上,任何非主属性不依赖于其他非主属性(在2NF基础上**消除传递依赖**)

## 案例

### 1.1NF

该表存在三个问题

其中,学号和课程名称为码,分数完全依赖码,姓名、系名、系主任部分依赖码。

学号	姓名	系名	系主任	课程名称	分数
10010	张无忌	经济系	张三丰	高等数学	95
10010	张无忌	经济系	张三丰	大学英语	87
10010	张无忌	经济系	张三丰	计算机基础	65
10011	令狐冲	法律系	任我行	法理学	77
10011	令狐冲	法律系	任我行	大学英语	87
10011	令狐冲	法律系	任我行	法律社会学	65
10012	杨过	法律系	任我行	法律社会学	95
10012	杨过	法律系	任我行	法理学	97
10012	杨过	法律系	任我行	大学英语	99
		计算机系	殷天正		
存在的问题： 1. 存在非常严重的数据冗余(重复)：姓名、系名、系主任 2. 数据添加存在问题：添加新开设的系和系主任时，数据不合法 3. 数据删除存在问题：张无忌同学毕业了，删除数据，会将系的数据一起删除。					

## 2.2NF

2NF解决了第一个问题,还有两个问题.

其中,系主任传递依赖学号.

选课表			学生表			
学号	课程名称	分数	学号	姓名	系名	系主任
10010	高等数学	95	10010	张无忌	经济系	张三丰
10010	大学英语	87	10011	令狐冲	法律系	任我行
10010	计算机基础	65	10012	杨过	法律系	任我行
10011	法理学	77				
10011	大学英语	87				
10011	法律社会学	65				
10012	法律社会学	95				
10012	法理学	97				
10012	大学英语	99				
存在的问题： 1. 存在非常严重的数据冗余(重复)：姓名、系名、系主任 2. 数据添加存在问题：添加新开设的系和系主任时，数据不合法 3. 数据删除存在问题：张无忌同学毕业了，删除数据，会将系的数据一起删除。						

## 3.3NF

3NF解决了所有问题

选课表			学生表		
学号	课程名称	分数	学号	姓名	系名
10010	高等数学	95	10010	张无忌	经济系
10010	大学英语	87	10011	令狐冲	法律系
10010	计算机基础	65	10012	杨过	法律系
10011	法理学	77			
10011	大学英语	87			
10011	法律社会学	65		系表	
10012	法律社会学	95		系名	系主任
10012	法理学	97		经济系	张三丰
10012	大学英语	99		法律系	任我行

## 数据库的备份和还原

### 备份

mysqldump -u 用户名 -p密码 数据库名称 > 保存的路径

```
1 | mysqldump -u root -root db1 > d://a.sql
```

### 还原

登录数据库

```
mysql -u root -p root
```

创建新的数据库

```
create database db1;
```

使用该数据库

```
use db1;
```

执行文件

```
source d://a.sql
```

source 文件路径

## 多表查询

笛卡尔积

有两个集合A,B,取这两个集合的所有组成情况

会产生多余无用的数据

要完成多表查询,需要消除无用的数据

### 分类

#### 1.内连接查询

##### 1.隐式内连接

使用where条件消除无用数据

```

1 SELECT
2     别名1.列名1,
3     别名2.列名1
4 FROM
5     表名1 别名1,
6     表名2 别名2
7 WHERE
8     别名1.共有列名 = 别名2.共有列名(条件)

```

## 2.显式内连接

```

1 SELECT
2     别名1.列名1,
3     别名2.列名1
4 FROM
5     表名1 别名1
6 [INNER] JOIN
7     表名2 别名2
8 ON
9     别名1.共有列名 = 别名2.共有列名(条件)

```

## 3.思路

- 1.从哪些表中查询数据
- 2.条件是什么
- 3.查询哪些字段

## 2.外连接查询

### 1.左外连接

平常代码

```

1 SELECT
2     e.*,
3     j.jname
4 FROM
5     emp e,
6     job j
7 WHERE
8     e.job_id = j.id;

```

如果job\_id没有的不会显示

id	ename	job_id	mgr	hiredate	salary	bonus	dept_id	jname
1	飞飞	1	1004	2021-03-02 14:47:19	10000.00	(NULL)	20	董事长
3	虎	2	1006	(NULL)	(NULL)	(NULL)	(NULL)	经理

```

1 SELECT
2     别名1.列名1,
3     别名2.列名1
4 FROM
5     表名1 别名1
6 LEFT [OUTER] JOIN
7     表名2 别名2
8 ON
9     别名1.共有列名 = 别名2.共有列名(条件)

```

查询的是左表(表1)所有数据以及其交集部分

```

1  SELECT
2      e.ename,
3      e.salary,
4      j.jname
5  FROM
6      emp e
7  LEFT JOIN
8      job j
9  ON
10     e.job_id = j.id

```

job\_id没有的也会显示

<input type="checkbox"/>	id	ename	job_id	mgr	joindate	salary	bonus	dept_id	jname
<input type="checkbox"/>	1	飞飞	1	1004	2021-03-02 14:47:19	10000.00	5.00	20	董事长
<input type="checkbox"/>	3	虎	2	1006	2021-03-10 14:58:22	1.00	3.00	(NULL)	经理
<input type="checkbox"/>	2	hiao	(NULL)	1005	2021-04-01 14:58:26	29.00	2.00	10	(NULL)

## 2.右外连接

同左外连接,只是判断哪个表的所有数据

```

1  SELECT
2      别名1.列名1
3      别名2.列名1
4  FROM
5      表名1 别名1
6  RIGHT [OUTER] JOIN
7      表名2 别名2
8  ON
9      别名1.共有列名 = 别名2.共有列名(条件)

```

查询的是右表(表2)所有数据以及其交集部分

```

1  SELECT
2      e.*,
3      j.jname
4  FROM
5      empp e
6  RIGHT JOIN
7      job j
8  ON
9      e.job_id = j.id;

```

显示右表job查询列的所有数据,和交集部分,没有值的显示null

<input type="checkbox"/>	id	ename	job_id	mgr	joindate	salary	bonus	dept_id	jname
<input type="checkbox"/>	1	飞飞	1	1004	2021-03-02 14:47:19	10000.00	(NULL)	20	董事长
<input type="checkbox"/>	3	虎	2	1006	(NULL)	(NULL)	(NULL)	(NULL)	经理
<input type="checkbox"/>	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	销售员
<input type="checkbox"/>	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	文员

## 3.子查询

查询中嵌套查询,称嵌套查询为子查询



## 1.子查询的结果是单行单列

子查询可以作为条件,使用运算符去判断.> ,>= ,< ,<= , =

```
1  ①SELECT
2      列名
3  FROM
4      表名
5  WHERE
6      列名 运算符
7      ②(
8          SELECT
9              列名
10             FROM
11                 表名
12         )
```

一般②的结果是一行一列,每一行都可以当做①的查询条件

```
1  SELECT
2      *
3  FROM
4      emp
5  WHERE emp.salary = (
6      SELECT MAX(salary) FROM emp
7  )
```

## 2.子查询的结果是多行单列

子查询可以作为条件,使用运算符 IN 来判断

```
1  ① SELECT
2      列名
3  FROM
4      表名1
5  WHERE
6      列名 IN
7      ②(
8          SELECT
9              列名
10             FROM
11                 表名2
12             WHERE
13                 条件
14         )
```

一般②的结果是多行一列,每一行都可以当做①的查询条件,所有条件在()里当集合,用IN运算符连接

```
1  SELECT
2      *
3  FROM
4      emp
5  WHERE emp.dept_id IN (
6      SELECT
7          id
```

```

8      FROM
9      dept
10     WHERE
11     dept.dname='财务部'
12     OR
13     dept.dname='销售部'
14 )

```

id	ename	job_id	mgr	hiredate	salary	bonus	dept_id
1002	卢俊义	3	1006	2001-02-20 14:35:48	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22 14:35:48	12500.00	5000.00	30
1005	李逵	4	1006	2001-09-28 14:35:48	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01 14:35:48	28500.00	(NULL)	30
1010	吴用	3	1006	2001-09-08 14:35:48	15000.00	0.00	30
1012	李逵	4	1006	2001-12-03 14:35:48	9500.00	(NULL)	30

### 3.子查询的结果是多行多列的

子查询可以作为一张虚拟表参与查询

```

1  ① SELECT
2      列名
3  FROM
4      表名1 别名1,
5      ②(
6          SELECT 多个列名
7          FROM 表名2
8          WHERE 条件
9      ) 别名2
10 WHERE
11     别名1.共有列名 = 别名2.共有列名(条件)

```

一般②是多行多列,可以看成是一个表格

```

1  SELECT
2      *
3  FROM
4      dept t1,
5      (
6          SELECT
7              *
8          FROM
9              emp
10         WHERE
11             emp.hiredate > '2002-05-01'
12     ) t2
13
14 WHERE t1.id = t2.dept_id

```

id	dname	loc	id	ename	job_id	mgr	hiredate	salary	bonus	dept_id
20	学工部	上海	1008	猪八戒	4	1004	2007-04-19 14:35:48	30000.00	(NULL)	20
20	学工部	上海	1011	沙僧	4	1004	2007-05-23 14:35:48	11000.00	(NULL)	20

内连接查询的只有满足条件的

左连接是显示表1中所有数据,即使外键右表数据为空

## 事务

如果一个包含多个步骤的业务操作,被事务管理,那么这些操作要么同时成功,要么同时失败

## 操作

### 1.开启事务

start transaction

### 2.回滚

rollback

### 3.提交

commit

1	1. START TRANSACTION;	#1. 开启事务
2	2. SQL 语句;	#2. SQL操作语句
3	3. COMMIT;	#3. 发现执行没有问题,提交事务
4	4. ROLLBACK;	#4. 发现出问题了,回滚事务

## MySQL数据中心事务默认自动提交

### 事务提交的两种方式

#### 自动提交

mysql就是自动提交的

一条DML(数据表增删改)语句会自动提交一次事务

#### 手动提交

Oracle数据库默认是手动提交事务

需要先开启事务(START TRANSACTION),再提交(COMMIT)

### 修改事务的默认提交方式

#### 查看事务的默认提交方式

SELECT @@autocommit;

1 代表自动提交

0 代表手动提交

#### 修改默认提交方式

set @@autocommit = 1/0;

修改后,必须重启才能生效

## 事务四大特征(ACID)

### 1.原子性 (atomicity)

是不可分割的最小操作单位,要么同时成功,要么同时失败

### 2.持久性 (durability)

当事务提交或回滚后,数据库会持久化的保存数据

### 3.隔离性 (isolation)

多个事物之间,相互独立,一个事务的修改在最终提交前,对其他事务是不可见的。

### 4.一致性 (consistency)

事务操作前后,数据总量不变,通过前三种保证一致性

## 事务的隔离级别

多个事务之间隔离的,相互独立的,但是如果多个事务操作同一批数据,则会引发一些问题,设置不同的隔离级别就可以解决这些问题

## 存在问题

### 1.脏读

一个事务,读取到另一个事务中没有提交的数据

两个事务①,②都开启

①事务SQL语句执行完,还没有提交,②事务就能查询到①事务操作后的数据

### 2.不可重复读(虚读)

在同一个事务中,两次读取到的数据不一样

两个事务①,②都开启

①事务SQL语句执行完,还没有提交,②事务查询的数据是没有更改的,当①事务提交后,②事务没有重启,②事务再

查询数据,就变成了①事务修改后的数据.

②事务,两次查询结果不一样,此时②事务就发生了虚读

### 3.幻读

一个事务操作(DML)数据表中所有记录,另一个事务添加了一条数据,则第一个事务查询不到自己的修改

一个事务读取了一个表的一部分数据,另一个事务向表中插入新的行,第一个事务再读的时候多了一部分

## 隔离级别

### 1.read uncommitted

读未提交

产生的问题

脏读,不可重复读,幻读

### 2.read committed

读已提交(Oracle)

产生的问题

不可重复读,幻读

解决了脏读问题,不能读取①事务未提交的数据

### 3.repeatable read

可重复读(MySQL)

产生的问题

幻读

解决了不可重复读问题,②事务不结束,读取到的数据都一样

### 4.serializable

串行化

可以解决所有的问题

①事务不提交结束,②事务不能查询

## 注意

隔离级别从小到大安全性越来越高,但是效率越来越低

## 数据库查询隔离级别

```
SELECT @@tx_isolation;
```

## 数据库设置隔离级别

SET GLOBAL TRANSACTION ISOLATION LEVEL 级别字符串;

# DCL

## 管理用户,授权

1.管理用户

1.添加用户

CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';

主机名一般是'localhost'

2.删除用户

DROP USER '用户名'@'主机名';

3.修改用户密码

UPDATE USER SET PASSWORD = PASSWORD('新密码') WHERE USER = '用户名'

DCL特有的方法

SET PASSWORD FOR '用户名'@'主机号' = PASSWORD('新密码');

4.查询用户

切换到mysql数据库

use mysql

查询user表

SELECT \* FROM USER;

通配符

%表示可以在任意主机使用用户登录数据库

CREATE USER '用户名'@'%' IDENTIFIED BY '密码';

忘记密码

1.cmd --> net stop mysql 停止MySQL服务器

需要管理员运行cmd

2.使用无验证方式启动mysql服务器

mysql --skip-grant-tables

3.打开新的cmd窗口,直接输入mysql,回车,就可以登录成功

4.use mysql

5.UPDATE USER SET PASSWORD = PASSWORD('新密码') WHERE USER = '用户名'

6.关闭两个窗口

7.打开任务管理器,手动结束mysqld.exe的进程

8.启动mysql服务

9.使用新密码登录

授权

1.查询权限

SHOW GRANTS FOR '用户名'@'主机名'

2.授予权限

GRANT 权限列表 ON 数据库名.表名 TO '用户名'@'主机名'

GRANT ALL ON . TO '用户名'@'主机名'

授予所有权限,在任意数据库任意表上

权限列表

SELECT

DELETE

UPDATE

3.撤销权限

REVOKE 权限列表 ON 数据库名.表名 FROM '用户名'@'主机名'

# 索引

# 索引的基本原理

索引用来快速地寻找那些具有特定值的记录。如果没有索引，一般来说执行查询时遍历整张表。

索引的原理:就是把无序的数据变成有序的查询

- 1.把创建了索引 | 的列的内容进行排序
- 2.对排序结果生成倒排表
- 3.在倒排表内容上拼上数据地址链
- 4.在查询的时候，先拿到倒排表内容，再取出数据地址链,从而拿到具体数据

## 索引的作用

- 提高查询速度
- 确保数据的唯一性
- 可以加速表和表之间的连接，实现表与表之间的参照完整性
- 使用分组和排序子句进行数据检索时，可以显著减少分组和排序的时间
- 全文检索字段进行搜索优化.

## 分类

- 主键索引 (Primary Key)
- 唯一索引 (Unique)
- 常规索引 (Index)
- 全文索引 (FullText)

### 主键索引

主键：某一个属性组能唯一标识一条记录

特点：

- 最常见的索引类型
- 确保数据记录的唯一性
- 确定特定数据记录在数据库中的位置

### 唯一索引

作用：避免同一个表中某数据列中的值重复

与主键索引的区别

- 主键索引只能有一个
- 唯一索引可能有多个

```
1 CREATE TABLE `Grade` (  
2   `GradeID` INT(11) AUTO_INCREMENT PRIMARYKEY,  
3   `GradeName` VARCHAR(32) NOT NULL UNIQUE  
4   -- 或 UNIQUE KEY `GradeID` (`GradeID`)  
5 )
```

### 常规索引

作用：快速定位特定数据

注意：

- index 和 key 关键字都可以设置常规索引
- 应加在查询找条件的字段
- 不宜添加太多常规索引,影响数据的插入,删除和修改操作

```
1 CREATE TABLE `result` (  
2   -- 省略一些代码  
3   INDEX/KEY `ind` (`studentNo`, `subjectNo`) -- 创建表时添加  
4 )  
5 -- 创建后添加  
6 ALTER TABLE `result` ADD INDEX `ind` (`studentNo`, `subjectNo`);
```

# 全文索引

百度搜索：全文索引

作用：快速定位特定数据

注意：

- 只能用于MyISAM类型的数据表
- 只能用于CHAR, VARCHAR, TEXT数据列类型
- 适合大型数据集

```
1  /*
2  #方法一：创建表时
3      CREATE TABLE 表名 (
4          字段名1 数据类型 [完整性约束条件...],
5          字段名2 数据类型 [完整性约束条件...],
6          [UNIQUE | FULLTEXT | SPATIAL ] INDEX | KEY
7          [索引名] (字段名[(长度)] [ASC |DESC])
8      );
9
10
11 #方法二：CREATE在已存在的表上创建索引
12     CREATE [UNIQUE | FULLTEXT | SPATIAL ] INDEX 索引名
13         ON 表名 (字段名[(长度)] [ASC |DESC]) ;
14
15
16 #方法三：ALTER TABLE在已存在的表上创建索引
17     ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL ] INDEX
18         索引名 (字段名[(长度)] [ASC |DESC]) ;
19
20
21 #删除索引：DROP INDEX 索引名 ON 表名字；
22 #删除主键索引：ALTER TABLE 表名 DROP PRIMARY KEY；
23
24
25 #显示索引信息：SHOW INDEX FROM student；
26 */
27
28 /*增加全文索引*/
29 ALTER TABLE `school`.`student` ADD FULLTEXT INDEX `studentname` (`StudentName`);
30
31 /*EXPLAIN ：分析SQL语句执行性能*/
32 EXPLAIN SELECT * FROM student WHERE studentno='1000';
33
34 /*使用全文索引*/
35 -- 全文搜索通过 MATCH() 函数完成。
36 -- 搜索字符串作为 against() 的参数被给定。搜索以忽略字母大小写的方式执行。对于表中的每个记录行，
37 MATCH() 返回一个相关性值。即，在搜索字符串与记录行在 MATCH() 列表中指定的列的文本之间的相似性尺度。
38
39 EXPLAIN SELECT *FROM student WHERE MATCH(studentname) AGAINST('love');
40
41
42 /*
43 开始之前，先说一下全文索引的版本、存储引擎、数据类型的支持情况
44
45 MySQL 5.6 以前的版本，只有 MyISAM 存储引擎支持全文索引；
46 MySQL 5.6 及以后的版本，MyISAM 和 InnoDB 存储引擎均支持全文索引；
47 只有字段的数据类型为 char、varchar、text 及其系列才可以建全文索引。
48 测试或使用全文索引时，要先看一下自己的 MySQL 版本、存储引擎和数据类型是否支持全文索引。
49 */
```

## 索引准则

- 索引不是越多越好
- 不要对经常变动的数据加索引
- 小数据量的表建议不要加索引
- 索引一般应加在查找条件的字段

# 锁

## 锁的类型有哪些

基于锁的属性分类:共享锁、排他锁。

基于锁的粒度分类:行级锁(INNODB)、表级锁(INNODB、MYISAM)、页级锁(BDB引擎)、记录锁、间隙锁、临键锁。

基于锁的状态分类:意向共享锁、意向排它锁。

### ●共享锁(Share Lock)

共享锁又称**读锁**，简称S锁;当一个事务为数据加上读锁之后，其他事务**只能对该数据加读锁**，而不能对数据加写锁，直到所有的读锁释放之后其他事务才能对其进行加写锁。共享锁的特性主要是为了支持并发的读取数据，读取数据的时候**不支持修改**，避免出现**重复读**的问题。

### ●排他锁(eXclusive Lock)

排他锁又称**写锁**，简称X锁;当一个事务为数据加上写锁时，其他请求将**不能再为数据加任何锁**，直到该锁释放之后，其他事务才能对数据进行加锁。排他锁的目的是在数据修改时候，**不允许其他人同时修改，也不允许其他人读取**。避免了出现**脏数据和脏读**的问题。

### ●表锁

表锁是指上锁的时候锁住的是**整个表**，当下一个事务访问该表的时候，必须等前一个事务释放了锁才能进行对表进行访问;

特点:粒度大，加锁简单，容易冲突;

### ●行锁(INNODB)

行锁是指上锁的时候锁住的是表的**某一行或多行记录**，其他事务访问同一张表时，只有被锁住的记录不能访问，其他的记录可正常访问;

特点:粒度小，加锁比表锁麻烦，不容易冲突，相比表锁支持的并发要高;会出现死锁

### ●记录锁(Record Lock)

记录锁也属于**行锁中的一种**，只不过记录锁的范围只是表中的**某一条记录**，记录锁是说事务在加锁后锁住的只是表的某一条记录。

精准条件命中，并且命中的条件字段是**唯一索引**

加了记录锁之后数据可以避免数据在查询的时候被修改的**重复读问题**，也避免了在修改的事务未提交前被其他事务读取的**脏读**问题。

### ●页锁

页级锁是MySQL中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，一次锁定相邻的一组记录。

特点:开销和加锁时间界于表锁和行锁之间;会出现死锁;锁定粒度界于表锁和行锁之间，并发度一般

### ●间隙锁(Gap Lock)

属于**行锁中的一种**，间隙锁是在事务加锁后其锁住的是表记录的某一个区间，当表的相邻ID之间出现空隙则会形成**一个区间，遵循左开右闭原则**。

范围查询并且查询未命中记录，查询条件必须命中索引、间隙锁只会出现在REPEATABLE\_READ (重复读)的事务级别中。

触发条件:防止**幻读**问题，事务并发的时候，如果没有间隙锁，就会发生如下图的问题，在同一个事务里，A事务的两次查询出的结果会不一样。



比如表里面的数据ID为1,4,5,7,10, 那么会形成以下几个间隙区间, -n-1区间, 1-4区间, 7-10区间, 10-n区间(-n代表负无穷大, n代表正无穷大)

## ●临键锁(Next-Key Lock)

也属于**行锁的一种**, 并且它是INNODB的**行锁默认算法**, 总结来说它就是**记录锁和间隙锁**的组合, 临键锁会把查询出来的记录锁住, 同时也会把该范围查询内的所有间隙空间也会锁住, 再之它会把相邻的下一个区间也会锁住

触发条件:范围查询并命中, 查询命中了索引。

结合记录锁和间隙锁的特性, 临键锁避免了在范围查询时出现脏读、重复读、幻读问题。加了临键锁之后, 在范围区间内数据不允许被修改和插

如果当事务A加锁成功之后就设置一个状态告诉后面的人, 已经有人对表里的行加了一个排他锁了, 你们不能对整个表加共享锁或排它锁了, 那么后面需要对整个表加锁的人只需要获取这个状态就知道自己是不是可以对表加锁, 避免了对整个索引树的每个节点扫描是否加锁, 而这个状态就是意向锁。

## ●意向共享锁

当一个事务试图对整个表进行加共享锁之前, 首先需要**获得**这个表的意向共享锁。

## ●意向排他锁

当一个事务试图对整个表进行加排它锁之前, 首先需要**获得**这个表的意向排它锁。

# 数据库优化

## 慢查询

### 查看慢查询

```
1 | show VARIABLES like 'slow_query_log';
```

## 配置文件

打开E:\mysql\配置文件.ini文件  
添加或者修改配置文件

```
1 | #--是否开启慢查询日志,0->off,1->on
2 | slow_query_log=on
3 |
4 | # --指定保存路径及文件名,默认为数据文件目录
5 | slow_query_log_file="bxg_mysql_slow.log"
6 |
7 | #--指定多少秒返回查询的结果为慢查询
8 | long_query_time=1
9 |
10 | #--记录所有没有使用到索引的查询语句
11 | log_queries_not_using_indexes=1
12 |
13 | #--记录那些由于查找了多于1000次而引发的慢查询
14 | min_examined_row_limit=1000
15 |
16 | # --记录那些慢的optimize table, analyze table和alter table语句
17 | log_slow_admin_statements=1
18 |
19 | #--记录由Slave所产生的慢查询
20 | log_slow_slave_statements=1
```

然后重启mysql服务即可

## 命令行

```
1 set global slow_query_log=1;
2 set global slow_query_log_file='bxg_mysql_slow.log';
3 set long_query_time=1;
4 set global log_queries_not_using_indexes=1;
5 set global min_examined_row_limit=1000;
6 set global log_slow_admin_statements=1;
7 set global log_slow_slave_statements= 1;
```

## 配置文件修改和命令行修改的区别

- 1.配置文件,配置之后需要重启MySQL服务,永久生效
- 2.命令行修改配置方式不需要重启即可生效,但重启之后会自动失效

## 清除缓存

```
1 # 设置缓存大小
2 set global query_cache_size=0;
3 # 设置缓存功能,0->off,1->on
4 set global query_cache_type=0;
```

整个数据库重启之后第一 执行sql语句, 考虑到第一次磁盘读写和数据库启动时间.

为了提高速度,可以暂停事务.测试添加100万随机数据,大概600s左右时间.

```
1 -- 停掉事务
2 set autocommit = 0;
3 call generate(1000000)
4 -- CALL genDepAdd(1000000);
5 -- 重启事务
6 set autocommit = 1;
```

字段	说明
Id	查询中执行select子句或操作表的顺序 数字越高执行快
Select_type	所使用的SELECT查询类型，包括以下常见类型:SIMPLE、PRIMARY、SUBQUERY、UNION、DERIVED、UNION RESULT、DEPENDENT、UNCACHEABLE
table	所使用的的数据表的名字
type	表示MySQL在表中找到所需行的方式，又称“访问类型”。取值按优劣排序为NULL>system > const > eg_ref ()> ref (找到首字拼音,比range更好)> fulltext >ref_or_null > index_merge > unique_subquery > index_subquery > range (根据首字的首字母查询) > index(扫描所有的索引,找拼音或者难查字) > ALL(效率特别低,需要对表中所有的数据全部查询才能找到)
Possible_keys	可能使用哪个索引在表中找到记录
key	实际使用的索引(如果为空就要考虑优化,没有用到索引)
Key_len	索引中使用的字节数
ref	显示索引的哪一列被使用了
rows	估算的找到所需的记录所需要读取的行数
filtered	通过条件过滤出的行数的百分比估计值
extra	包含不适合在其他列中显示但十分重要的额外信息

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	student	ALL	(Null)	(Null)	(Null)	(Null)	7	Using where