# 1.数据库（DB）

概念：数据库就是存储数据的仓库，安装在操作系统之上。

作用：存储数据，管理数据

# **2.数据库分类**

最常用的是 关系数据库。

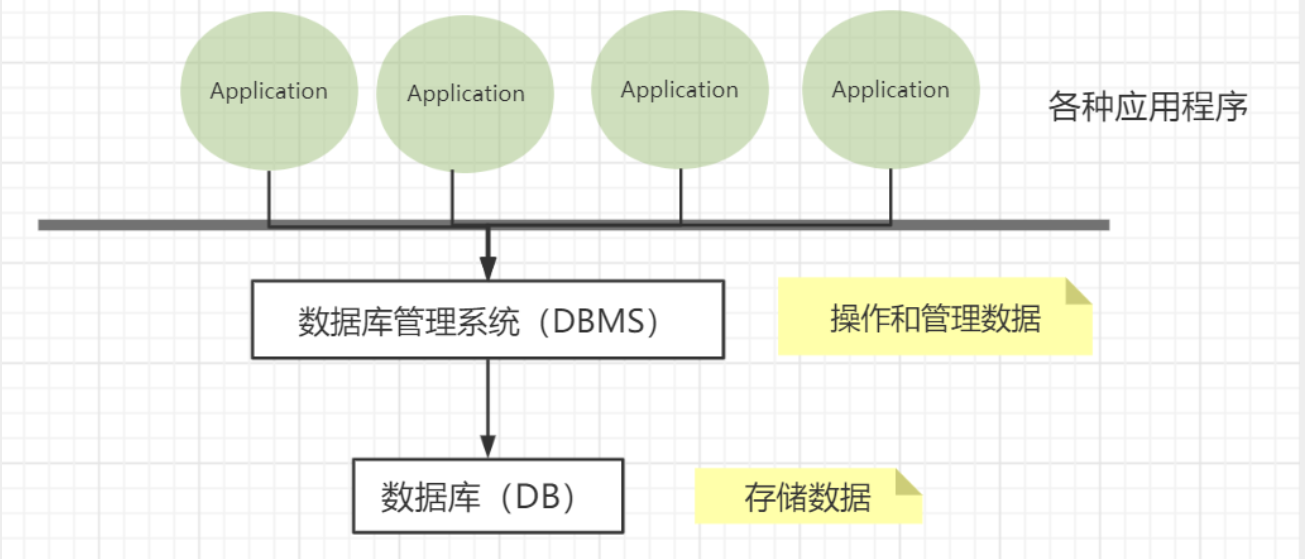
关系数据库（SQL Structured Query Language）：

* MySQL, Oracle, Sql Server, DB2，SQLite 为常见的关系数据库

关系数据库和Excel表格差不多，分为不同的表，表于表之间通过行和列之间的关系进行数据的存储，通过**外键**建立表和表之间的关系。

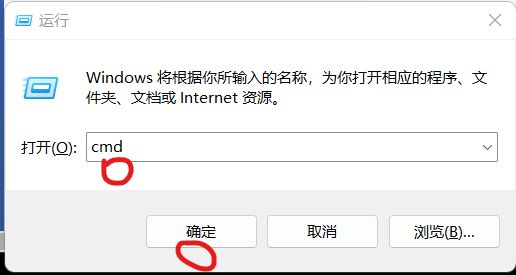
数据库管理系统（DBMS）：数据库的管理软件，管理，维护，获取我们的数据

MySQL就是数据库管理软件



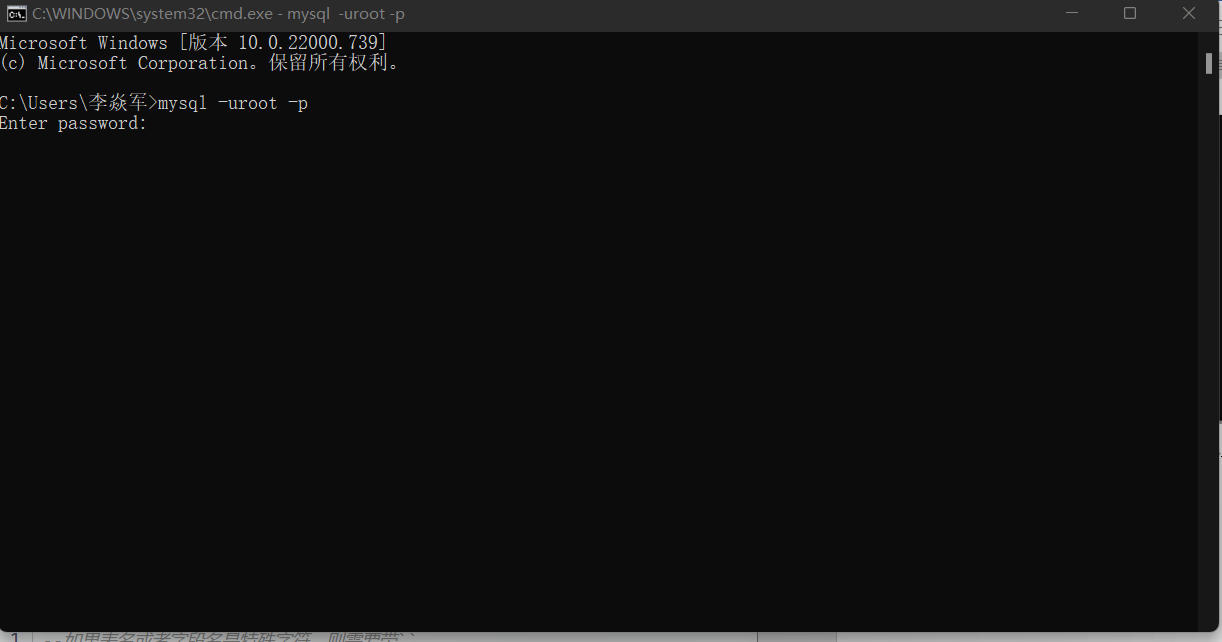
操作数据库：

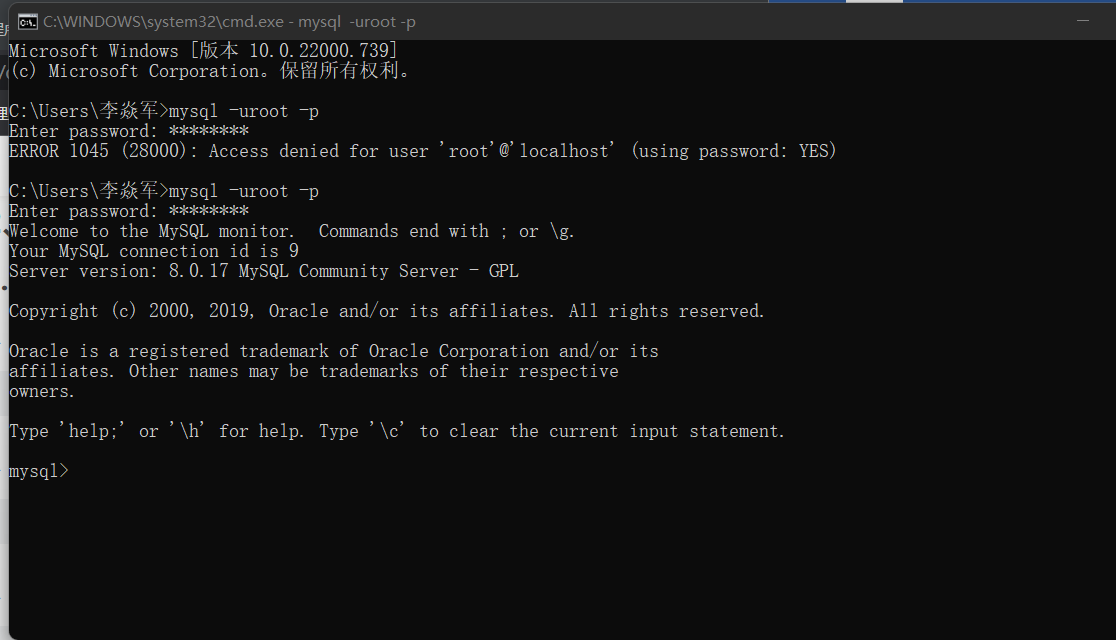
1.按住win + R 键，输入cmd，回车运行进入cmd命令窗口。





2.输入mysql –uroot –p（代码为mysql应用程序，-username，用户名为root，-password）然后回车，输入密码，进入mysql





# 3.数据库的列类型

**数值**：

| **数据类型** | **描述** | **大小** |
| --- | --- | --- |
| tinyint | 十分小的数据 | 1个字节 |
| smallint | 较小的数据 | 2个字节 |
| mediumint | 中等大小的数据 | 3个字节 |
| int | 标准的整数 | 4个字节 |
| bigint | 较大的数据 | 8个字节 |
| float | 浮点数 | 4个字节 |
| double | 浮点数 | 8个字节 |
| decimal | 字符串形式的浮点数，一般用于金融计算 |  |

**字符串：**

| **数据类型** | **描述** | **大小** |
| --- | --- | --- |
| char | 字符串固定大小 | 0~255 |
| varchar | 可变字符串 | 0~65535 |
| tinytext | 微型文本 | 2^8-1 |
| text | 文本串 | 2^16-1 |

**时间日期：**

| **数据类型** | **描述** | **格式** |
| --- | --- | --- |
| date | 日期格式 | YYYY-MM-DD |
| time | 时间格式 | HH：mm：ss |
| datetime | 最常用的时间格式 | YYYY-MM-DD HH：mm：ss |
| timestamp | 时间戳，1970.1.1到现在的毫秒数 |  |
| year | 年份表示 |  |

**Null值，空值，没有值，不要使用空值计算。**

# 4.数据库的字段属性

UnSigned：无符号，声明了该列不能为负数。

Zerofill 0：填充的，不足位数的用0来填充，例如int（3） 5 就是 005。

Auto\_increment： 自增，自动在上一条记录的基础上默认+1。

Null和not null：默认是null,即该列可以为空值，没有值，not null为该列必须有值

Default：默认的，用于设置默认值，例如，性别字段,默认为"男" , 否则为 “女” ; 若无指定该列的值 , 则默认值为"男"的值。

**拓展**：每一个表，都必须存在以下五个字段：

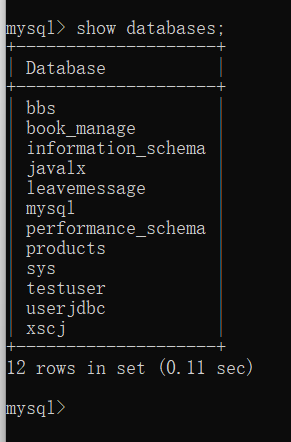
| **名称** | **描述** |
| --- | --- |
| id | 主键 |
| version | 乐观锁 |
| is\_delete | 伪删除 |
| gmt\_create | 创建时间 |
| gmt\_update | 修改时间 |

可以之后对比代码来看，现在仅作了解。

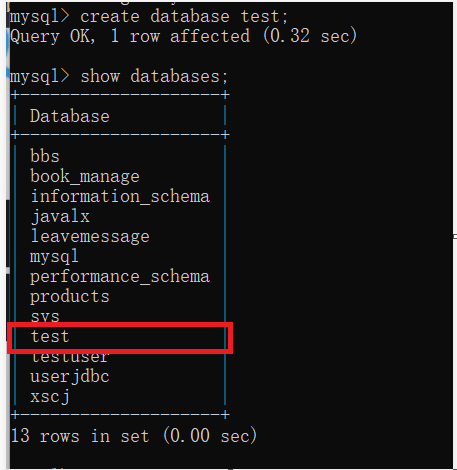
# 5.常见命令

输入命令时，命令后需要带分号 ; 且是英文的分号。

（1）show databases; 查看我们建立的数据库，其中系统自带几个。



（2）创建数据库 creat databse 数据库名;



创建数据库就相当于 创建了一个Excel表格，数据库名就是Excel的名字。

创建数据库后使用命令 use 数据库名，来切换到该数据库，就可以在该数据库中创建表了。

（3）创建数据库表 creat table 表名

CREATE TABLE IF NOT EXISTS `student`(

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

`name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',

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

`sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',

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

`address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',

`email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',

`gradeid` INT(10) NOT NULL COMMENT '学生的年级',

PRIMARY KEY (`id`),

KEY `FK\_gradeid` (`gradeid`),

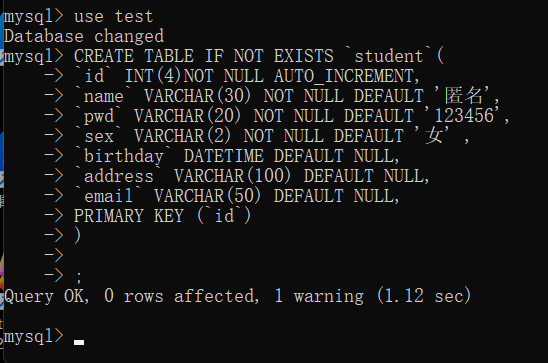
CONSTRAINT `FK\_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade`(`gradeid`)

)ENGINE=INNODB DEFAULT CHARSET=utf8;

注意点：COMMENT后跟注释，ENGINE=INNODB DEFAULT CHARSET=utf8是支持中文。

* 表名和字段尽量使用``括起来
* AUTO\_INCREMENT 代表自增
* 所有的语句后面加逗号，最后一个不加
* 字符串使用单引号括起来

数据库表 就是相当于 excel 中的sheet1，数据库表名就是为sheet取名



格式：

CREATE TABLE IF NOT EXISTS `student`(

'字段名' 列类型 [属性] [索引] [注释],

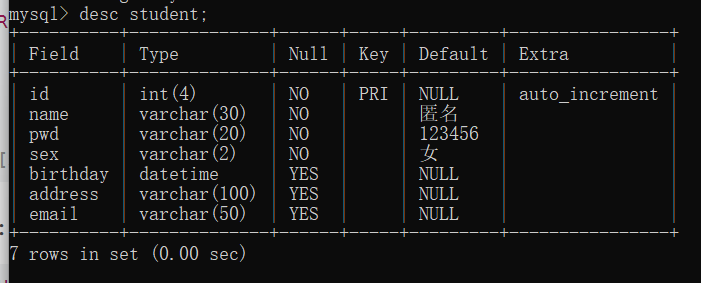
'字段名' 列类型 [属性] [索引] [注释],

......

'字段名' 列类型 [属性] [索引] [注释]

)[表的类型][字符集设置][注释]

(4)desc 表名; 显示对应的表结构



（5）修改数据库

修改表名 : ALTER TABLE 旧表名 RENAME AS 新表名

添加字段 : ALTER TABLE 表名 ADD字段名 列属性[属性]

修改字段 :

ALTER TABLE 表名 MODIFY 字段名 列类型[属性]

ALTER TABLE 表名 CHANGE 旧字段名 新字段名 列属性[属性]

删除字段 : ALTER TABLE 表名 DROP 字段名

例如：

-- 修改表名

-- ALTER TABLE 旧表名 RENAME AS 新表名

ALTER TABLE teacher RENAME AS teachers;

-- 增加表的字段

-- ALTER TABLE 表名 ADD 字段名 列属性

ALTER TABLE teachers ADD age INT(11);

-- 修改表的字段(重命名，修改约束)

-- ALTER TABLE 表名 MODIFY 字段名 [列属性];

ALTER TABLE teachers MODIFY age VARCHAR(11);-- 修改约束

-- ALTER TABLE 表名 CHANGE 旧名字 新名字 [列属性];

ALTER TABLE teachers CHANGE age age1 INT(1);-- 字段重命名

-- 删除表的字段

-- ALTER TABLE 表名 DROP 字段名

ALTER TABLE teachers DROP age1;

(6)删除表

**语法**：DROP TABLE [IF EXISTS] 表名

* IF EXISTS为可选 , 判断是否存在该数据表
* 如删除不存在的数据表会抛出错误

# 6.数据库管理

（1）外键

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。由此可见，外键表示了两个关系之间的相关联系。以另一个关系的外键作主关键字的表被称为主表，具有此外键的表被称为主表的从表。

在实际操作中，将一个表的值放入第二个表来表示关联，所使用的值是第一个表的主键值(在必要时可包括复合主键值)。此时，第二个表中保存这些值的属性称为外键(foreign key)。

外键作用：

保持数据一致性，完整性，主要目的是控制存储在外键表中的数据,约束。使两张表形成关联，外键只能引用外表中的列的值或使用空值。

**目标**：学生表（student）的gradeid字段 要去引用年级表（grade）的 gradeid字段

方法：创建外键

/\*

1. 定义外键key

2. 给外键添加约束（执行引用）references 引用

\*/

/\*

1. 定义外键key

2. 给外键添加约束（执行引用）references 引用

\*/

CREATE TABLE IF NOT EXISTS `student`(

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

`name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',

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

`sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',

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

`address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',

`email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',

`gradeid` INT(10) NOT NULL COMMENT '学生的年级',

PRIMARY KEY (`id`)

)ENGINE=INNODB DEFAULT CHARSET=utf8;

-- 创建表的时候没有外键关系

ALTER TABLE `student`

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

-- 创建年级表

CREATE TABLE `grade`(

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

`gradename` VARCHAR(50) NOT NULL COMMENT '年纪名称',

PRIMARY KEY (`gradeid`)

)ENGINE=INNODB DEFAULT CHARSET=utf8;

语句： CONSTRAINT `FK\_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade`(`gradeid`)就是绑定了外键，FK\_gradeid是外键名称，FOREIGN KEY (`gradeid`)，意思是设置student表的gradeid为外键，EFERENCES `grade`(`gradeid`) 意思是连接到grade表中的gradeid。

总体意思是，当student表中的gradeid改变时，grade表中的gradeid也需要改变。（通过触发器改变，之后讲）

以上的操作都是物理外键，数据库级别的外键，不建议使用！避免数据库过多造成困扰！

最佳实践

* 数据库就是用来单纯的表，只用来存数据，只有行（数据）和列（属性）
* 我们想使用多张表的数据，使用外键，用程序去实现

# 7.DML语言

DML语言：数据库操作语言，实现数据库的增删改查。

## 添加 insert

-- 普通用法

INSERT INTO `student`(`name`) VALUES ('zsr');

-- 插入多条数据

INSERT INTO `student`(`name`,`pwd`,`sex`) VALUES ('zsr','200024','男'),('gcc','000421','女');

-- 省略字段

INSERT INTO `student` VALUES (5,'Bareth','123456','男','2000-02-04','武汉','1412@qq.com',1);

**语法：**

**INSERT INTO 表名([字段1,字段2..])VALUES('值1','值2'..),[('值1','值2'..)..];**

**注意**：

1. 字段和字段之间使用英文逗号隔开
2. 字段是可以省略的，但是值必须完整且一一对应
3. 可以同时插入多条数据，VALUES后面的值需要使用逗号隔开

## （2）修改 update

-- 修改学员名字,指定条件

UPDATE `student` SET `name`='zsr204' WHERE id=1;

-- 不指定条件的情况,会改动所有表

UPDATE `student` SET `name`='zsr204';

-- 修改多个属性

UPDATE `student` SET `name`='zsr',`address`='湖北' WHERE id=1;

-- 通过多个条件定位数据

UPDATE `student` SET `name`='zsr204' WHERE `name`='zsr' AND `pwd`='200024';

**语法**：

UPDATE 表名 SET 字段1=值1,[字段2=值2...] WHERE 条件[];

**关于WHERE条件语句**：

| **操作符** | **含义** |
| --- | --- |
| = | 等于 |
| <>或!= | 不等于 |
| > | 大于 |
| < | 小于 |
| <= | 小于等于 |
| >= | 大于等于 |
| BETWEEN…AND… | 闭合区间 |
| AND | 和 |
| OR | 或 |

## 删除 delete

-- 删除数据(避免这样写,会全部删除)

DELETE FROM `student`;

-- 删除指定数据

DELETE FROM `student` WHERE id=1;

**语法**：

DELETE FROM 表名 [WHERE 条件]

关于DELETE删除的问题，重启数据库现象：

* INNODB 自增列会从1开始（存在内存当中，断电即失）
* MYISAM 继续从上一个子增量开始（存在内存当中，不会丢失）

Truncate

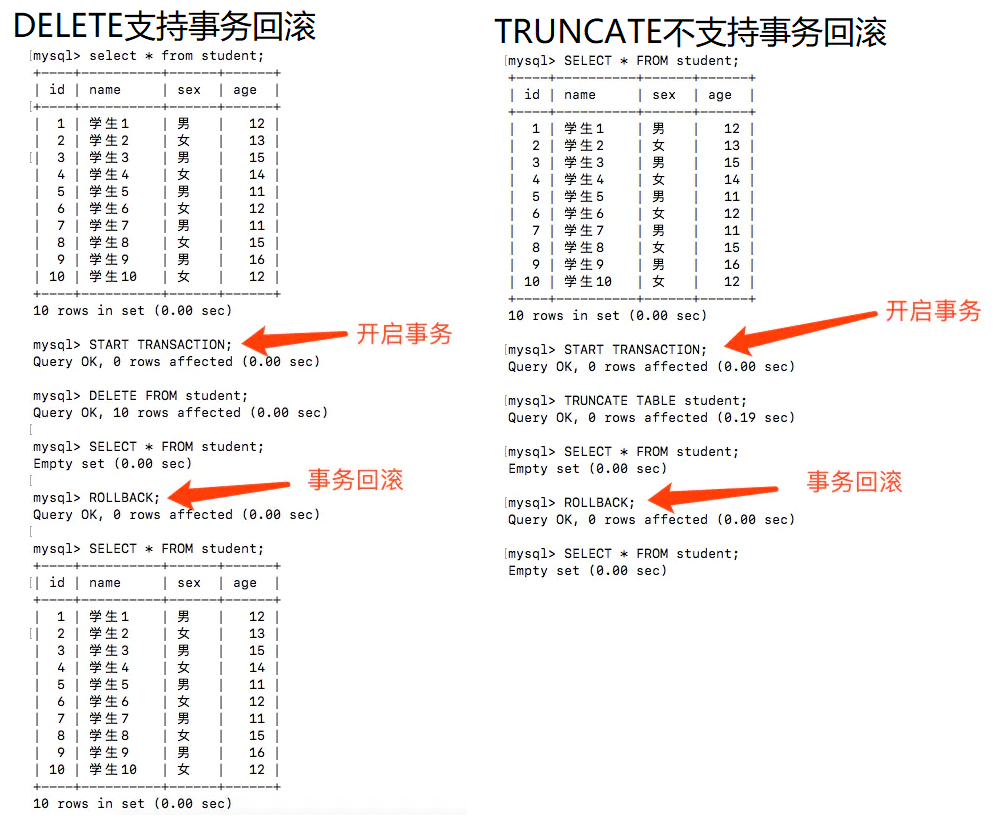
**作用**：完全清空一个数据库表，表的结构和索引约束不会变。

DELETE和TRUNCATE 的区别：

DELETE可以条件删除（where子句），而TRUNCATE只能删除整个表

TRUNCATE 重新设置自增列，计数器会归零，而DELETE不会影响自增

DELETE是数据操作语言（DML - Data Manipulation Language），操作时原数据会被放到 rollback segment中，可以被回滚；而TRUNCATE是数据定义语言（DDL - Data Definition Language)，操作时不会进行存储，不能进行回滚。



事务回滚就是撤销删除。

例如：

CREATE TABLE `test`(

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

`coll` VARCHAR(20) NOT NULL,

PRIMARY KEY (`id`)

)ENGINE=INNODB DEFAULT CHARSET=utf8;

INSERT INTO `test`(`coll`) VALUES('1'),('2'),('3');

-- 不会影响自增

DELETE FROM `test`;

-- 会影响自增

TRUNCATE TABLE `test`;

# 8.DQL查询语句

Select结构：

SELECT [ALL | DISTINCT]

{\* | table.\* | [table.field1[as alias1][,table.field2[as alias2]][,...]]}

FROM table\_name [as table\_alias]

[left | right | inner join table\_name2] -- 联合查询

[WHERE ...] -- 指定结果需满足的条件

[GROUP BY ...] -- 指定结果按照哪几个字段来分组

[HAVING] -- 过滤分组的记录必须满足的次要条件

[ORDER BY ...] -- 指定查询记录按一个或多个条件排序

[LIMIT {[offset,]row\_count | row\_countOFFSET offset}]; -- 指定查询的记录从哪条至哪条

* 查询数据库数据 , 如**SELECT**语句
* 简单的单表查询或多表的复杂查询和嵌套查询
* 是数据库语言中最核心,最重要的语句
* 使用频率最高的语句

## **前提配置**：

-- 创建学校数据库

CREATE DATABASE IF NOT EXISTS `school`;

-- 用school数据库

USE `school`;

-- 创建年级表grade表

CREATE TABLE `grade`(

`GradeID` INT(11) NOT NULL AUTO\_INCREMENT COMMENT '年级编号',

`GradeName` VARCHAR(50) NOT NULL COMMENT '年纪名称',

PRIMARY KEY (`GradeID`)

)ENGINE=INNODB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

-- 给grade表插入数据

INSERT INTO `grade`(`GradeID`,`GradeName`)

VALUES (1,'大一'),(2,'大二'),(3,'大三'),(4,'大四');

-- 创建成绩result表

CREATE TABLE `result`(

`StudentNo` INT(4) NOT NULL COMMENT '学号',

`SubjectNo` INT(4) NOT NULL COMMENT '考试编号',

`ExamDate` DATETIME NOT NULL COMMENT '考试日期',

`StudentResult` INT(4) NOT NULL COMMENT '考试成绩',

KEY `SubjectNo` (`SubjectNo`)

)ENGINE=INNODB DEFAULT CHARSET=utf8;

-- 给result表插入数据

INSERT INTO `result`(`StudentNo`,`SubjectNo`,`ExamDate`,`StudentResult`)

VALUES (1000,1,'2019-10-21 16:00:00',97),(1001,1,'2019-10-21 16:00:00',96),

(1000,2,'2019-10-21 16:00:00',87),(1001,3,'2019-10-21 16:00:00',98);

-- 创建学生表student

CREATE TABLE `student`(

`StudentNo` INT(4) NOT NULL COMMENT '学号',

`LoginPwd` VARCHAR(20) DEFAULT NULL,

`StudentName` VARCHAR(20) DEFAULT NULL COMMENT '学生姓名',

`Sex` TINYINT(1) DEFAULT NULL COMMENT '性别,取值0或1',

`GradeID` INT(11) DEFAULT NULL COMMENT '年级编号',

`Phone` VARCHAR(50) NOT NULL COMMENT '联系电话,允许为空,即可选输入',

`Adress` VARCHAR(255) NOT NULL COMMENT '地址,允许为空,即可选输入',

`BornDate` DATETIME DEFAULT NULL COMMENT '出生时间',

`Email` VARCHAR(50) NOT NULL COMMENT '邮箱账号,允许为空,即可选输入',

`IdentityCard` VARCHAR(18) DEFAULT NULL COMMENT '身份证号',

PRIMARY KEY (`StudentNo`),

UNIQUE KEY `IdentityCard` (`IdentityCard`),

KEY `Email` (`Email`)

)ENGINE=MYISAM DEFAULT CHARSET=utf8;

-- 给学生表插入数据

INSERT INTO `student`(`StudentNo`,`LoginPwd`,`StudentName`,`Sex`,`GradeID`,`Phone`,`Adress`,`BornDate`,`Email`,`IdentityCard`)

VALUES (1000,'1241','dsaf',1,2,'24357','unknow','2000-09-16 00:00:00','1231@qq.com','809809'),

(1001,'1321','dfdj',0,2,'89900','unknow','2000-10-16 00:00:00','5971@qq.com','908697');

-- 创建科目表

CREATE TABLE `subject`(

`SubjectNo` INT(11) NOT NULL AUTO\_INCREMENT COMMENT '课程编号',

`SubjectName` VARCHAR(50) DEFAULT NULL COMMENT '课程名称',

`ClassHour` INT(4) DEFAULT NULL COMMENT '学时',

`GradeID` INT(4) DEFAULT NULL COMMENT '年级编号',

PRIMARY KEY (`SubjectNo`)

)ENGINE=INNODB AUTO\_INCREMENT=18 DEFAULT CHARSET=utf8;

-- 给科目表subject插入数据

INSERT INTO `subject`(`SubjectNo`,`SubjectName`,`ClassHour`,`GradeID`)

VALUES(1,'高数','96',2),(2,'大物','112',2),(3,'程序设计',64,3);

## 基础查询

**语法**：

SELECT 查询列表 FROM 表名;

* 查询列表可以是：表中的（一个或多个）字段，常量，变量，表达式，函数
* 查询结果是一个虚拟的表格

-- 查询全部学生

SELECT \* FROM student;

-- 查询指定的字段

SELECT `LoginPwd`,`StudentName` FROM student;

-- 别名 AS(可以给字段起别名,也可以给表起别名)

SELECT `StudentNo` AS 学号,`StudentName` AS 学生姓名 FROM student AS 学生表;

-- 函数 CONCAT(str1,str2,...)

SELECT CONCAT('姓名',`StudentName`) AS 新名字 FROM student;

-- 查询系统版本(函数)

SELECT VERSION();

-- 用来计算(计算表达式)

SELECT 100\*53-90 AS 计算结果;

-- 查询自增步长(变量)

SELECT @@auto\_increment\_increment;

-- 查询有哪写同学参加了考试,重复数据要去重

SELECT DISTINCT `StudentNo` FROM result;

## 条件查询

**语法**：

select 查询列表 from 表名 where 筛选条件;

-- 查询考试成绩在95~100之间的

SELECT `StudentNo`,`StudentResult` FROM result

WHERE `StudentResult`>=95 AND `StudentResult`<=100;

-- &&

SELECT `StudentNo`,`StudentResult` FROM result

WHERE `StudentResult`>=95 && `StudentResult`<=100;

-- BETWEEN AND

SELECT `StudentNo`,`StudentResult` FROM result

WHERE `StudentResult`BETWEEN 95 AND 100;

-- 查询除了1000号以外的学生

SELECT `StudentNo`,`StudentResult` FROM result

WHERE `StudentNo`!=1000;

-- NOT

SELECT `StudentNo`,`StudentResult` FROM result

WHERE NOT `StudentNo`=1000;

-- 查询名字含d的同学

SELECT `StudentNo`,`StudentName` FROM student

WHERE `StudentName` LIKE '%d%';

-- 查询名字倒数第二个为d的同学

SELECT `StudentNo`,`StudentName` FROM student

WHERE `StudentName` LIKE '%d\_';

-- 查询1000,1001学员

SELECT `StudentNo`,`StudentName` FROM student

WHERE `StudentNo` IN (1000,1001);

## 分组查询

**语法**：

select 分组函数，分组后的字段

from 表

【where 筛选条件】

group by 分组的字段

【having 分组后的筛选】

【order by 排序列表】

**区别**：

| **使用关键字** | **筛选的表** | **位置** |
| --- | --- | --- |
| 分组前筛选 | where | 原始表 | group by的前面 |
| 分组后筛选 | having | 分组后的结果 | group by 的后面 |

-- 查询不同科目的平均分、最高分、最低分且平均分大于90

-- 核心：根据不同的课程进行分组

SELECT SubjectName,AVG(StudentResult),MAX(`StudentResult`),MIN(`StudentResult`)

FROM result r

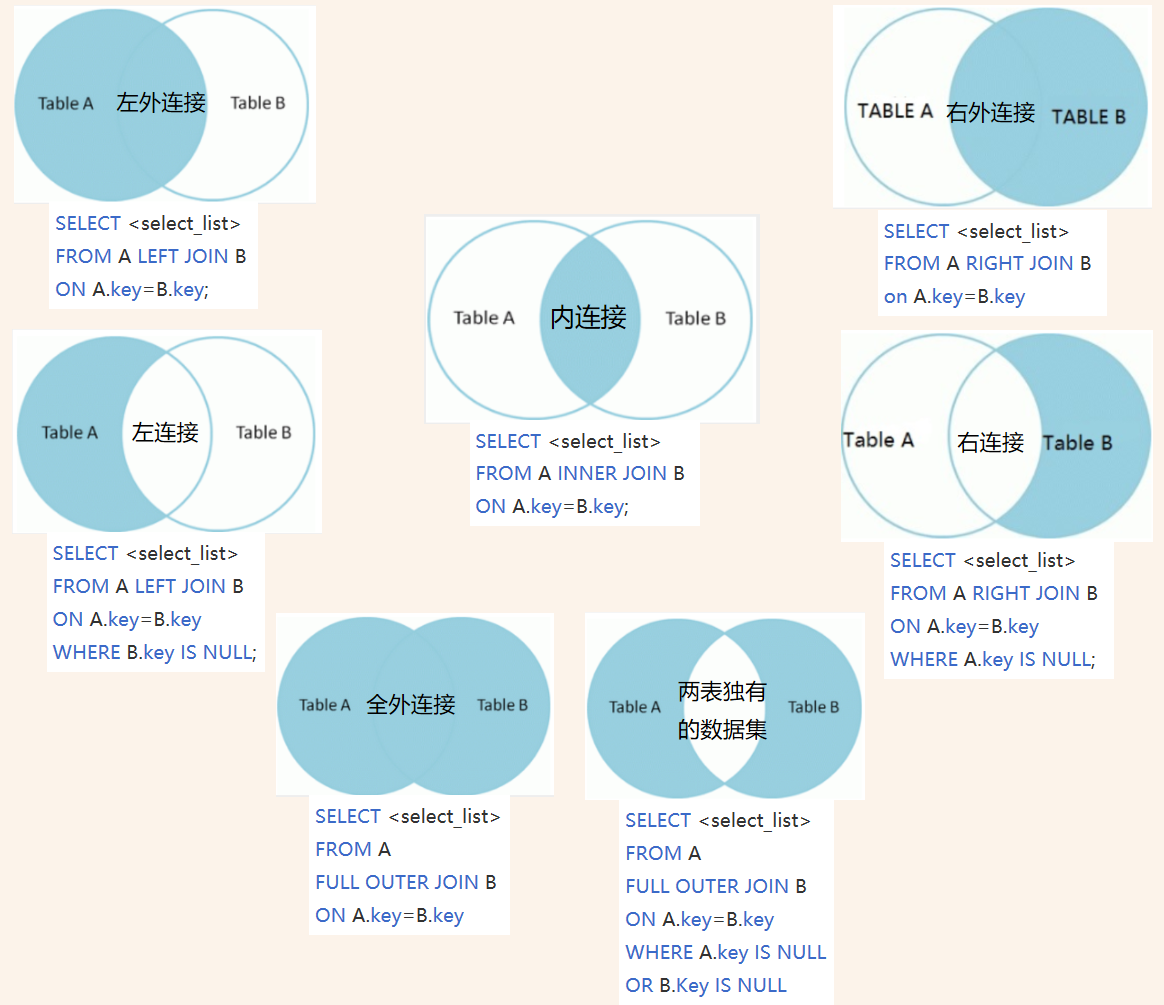
INNER JOIN `subject` s

on r.SubjectNo=s.SubjectNo

GROUP BY r.SubjectNo

HAVING AVG(StudentResult)>90;

## 连接查询



-- 查询学员所属的年级（学号，学生姓名，年级名称）

SELECT `StudentNo`,`StudentName`,`GradeName`

FROM student s

INNER JOIN grade g

ON s.GradeID=g.GradeID;

-- 查询科目所属的年级

SELECT `SubjectName`,`GradeName`

FROM `subject` s

INNER JOIN `grade` g

ON s.GradeID=g.GradeID;

-- 查询列参加程序设计考试的同学信息（学号，姓名，科目名，分数）

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='课程设计';

自连接：

自己的表和自己的表连接，核心：**一张表拆为两张一样的表即可**

-- 创建一个表

CREATE TABLE `course` (

`courseid` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT '课程id',

`pid` INT(10) NOT NULL COMMENT '父课程id',

`courseName` VARCHAR(50) NOT NULL COMMENT '课程名',

PRIMARY KEY (`courseid`)

) ENGINE=INNODB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8

-- 插入数据

INSERT INTO `course` (`courseid`, `pid`, `courseName`)

VALUES('2','1','信息技术'),

('3','1','软件开发'),

('4','3','数据库'),

('5','1','美术设计'),

('6','3','web开发'),

('7','5','ps技术'),

('8','2','办公信息');

将该表进行拆分：

| **pid（父课程id）** | **courseid（课程id）** | **courseName（课程名）** |
| --- | --- | --- |
| 1 | 2 | 信息技术 |
| 1 | 3 | 软件开发 |
| 1 | 5 | 美术设计 |

| **pid（父课程id）** | **courseid（课程id）** | **courseName（课程名）** |
| --- | --- | --- |
| 2 | 8 | 办公信息 |
| 3 | 4 | 数据库 |
| 3 | 6 | web开发 |
| 5 | 7 | ps技术 |

**操作**：查询父类对应的子类关系

| **父类** | **子类** |
| --- | --- |
| 信息技术 2 | 办公信息 4 |
| 软件开发 3 | 数据库 4、web开发 6 |
| 美术设计 5 | ps技术 7 |

SELECT a.`courseid` AS '父课程',b.`courseid` AS '子课程'

FROM course AS a,course AS b

WHERE a.`courseid`=b.`pid`;

## 排序和分页

排序：

**语法**：

select 查询列表

from 表

where 筛选条件

order by 排序列表 asc/desc

* order by的位置一般放在查询语句的最后（除limit语句之外）

| **asc ：** | **升序，如果不写默认升序** |
| --- | --- |
| **desc：** | **降序** |

SELECT `StudentNo`,`StudentName`,`GradeName`

FROM student s

INNER JOIN grade g

ON s.GradeID=g.GradeID

ORDER BY `StudentNo` DESC;

分页:

**语法**：

select 查询列表

from 表

limit offset,pagesize;

* offset代表的是起始的条目索引，默认从0开始
* size代表的是显示的条目数
* offset=(n-1)\*pagesize

-- 第一页 limit 0 5

-- 第二页 limit 5,5

-- 第三页 limit 10,5

-- 第n页 limit (n-1)\*pagesize,pagesize

-- pagesize:当前页面大小

-- (n-1)\*pagesize:起始值

-- n:当前页面

-- 数据总数/页面大小=总页面数

-- limit n 表示从0到n的页面

## 子查询

**本质**：在 where子句中嵌套一个子查询语句

-- 查询‘课程设计’的所有考试结果（学号，科目编号，成绩）降序排列

-- 方式一:使用连接查询

SELECT `StudentNo`,r.`SubjectNo`,`StudentResult`

FROM result r

INNER JOIN `subject` s

on r.StudentNo=s.SubjectNo

WHERE SubjectName='课程设计'

ORDER BY StudentResult DESC;

-- 方式二:使用子查询（由里到外）

SELECT StudentNo,SubjectNo,StudentResult

from result

WHERE SubjectNo=(

SELECT SubjectNo FROM `subject`

WHERE SubjectName='课程设计'

)

# 9. MySQL函数

## （1）常用函数

-- 数学运算

SELECT ABS(-8); -- 绝对值

SELECT CEIL(5.1); -- 向上取整

SELECT CEILING(5.1); -- 向上取整

SELECT RAND(); -- 返回0~1之间的一个随机数

SELECT SIGN(-10); -- 返回一个数的符号;0返回0;正数返回1;负数返回-1

-- 字符串函数

SELECT CHAR\_LENGTH('我喜欢你'); -- 字符串长度

SELECT CONCAT('我','喜欢','你'); -- 拼接字符串

SELECT INSERT('我喜欢',1,1,'超级') -- INSERT(str,pos,len,newstr) 从str的pos位置开始替换为长度为len的newstr

SELECT UPPER('zsr'); -- 转大写

SELECT LOWER('ZSR'); -- 转小写

SELECT INSTR('zsrs','s'); -- 返回第一次出现字串索引的位置

SELECT REPLACE('加油就能胜利','加油','坚持'); -- 替换出现的指定字符串

SELECT SUBSTR('坚持就是胜利',3,6); -- 返回指定的字符串(源字符串,截取位置,截取长度)

SELECT REVERSE('rsz'); -- 反转字符串

-- 时间日期函数

SELECT CURRENT\_DATE(); -- 获取当前日期

SELECT CURDATE(); -- 获取当前日期

SELECT now(); -- 获取当前时间

SELECT LOCALTIME(); -- 本地时间

SELECT SYSDATE(); -- 系统时间

SELECT YEAR(NOW());

SELECT MONTH(NOW());

SELECT DAY(NOW());

SELECT HOUR(NOW());

SELECT MINUTE(NOW());

SELECT SECOND(NOW());

-- 系统信息

SELECT SYSTEM\_USER();

SELECT USER();

SELECT VERSION();

## （2） 聚合函数

| **函数** | **描述** |
| --- | --- |
| max | 最大值 |
| min | 最小值 |
| sum | 和 |
| avg | 平均值 |
| count | 计算个数 |

SELECT COUNT(StudentName) FROM student;

SELECT COUNT(\*) FROM student;

SELECT COUNT(1) FROM student;

SELECT SUM(`StudentResult`) FROM result;

SELECT AVG(`StudentResult`) FROM result;

SELECT MAX(`StudentResult`) FROM result;

SELECT MIN(`StudentResult`) FROM result;

# 10.事务

要么都成功，要么都失败

SQL执行：A转账给B

SQL执行：B收到A的钱

将一组SQL放在一个批次中去执行

* 例如银行转账：只有A转账成功且B成功到账，该事件才算结束，如果一方不成功，则该事务不成功

## 事务原则：ACID

原子性（Atomicity） 原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性（Consistency） 事务前后数据的完整性必须保持一致。

隔离性（Isolation） 事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

持久性（Durability） 事务一旦被提交则不可逆，被持久化到数据库中，接下来即使数据库发生故障也不应该对其有任何影响

## 事务并发导致的问题

隔离所导致的一些问题：

| **名称** | **描述** |
| --- | --- |
| **脏读** | 指一个事务读取了另外一个事务未提交的数据。 |
| **不可重复读** | 在一个事务内读取表中的某一行数据，多次读取结果不同。 |
| **虚读(幻读)** | 是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致 |

## 隔离级别

在数据库操作中，为了有效保证并发读取数据的正确性，提出的**事务隔离级别**

读未提交：一个事务读取到其他事务未提交的数据；这种隔离级别下，查询不会加锁，一致性最差，会产生脏读、不可重复读、幻读的问题

读已提交：一个事务只能读取到其他事务已经提交的数据；该隔离级别避免了脏读问题的产生，但是不可重复读和幻读的问题仍然存在；

读提交事务隔离级别是大多数流行数据库的默认事务隔离级别，比如 Oracle，但是不是 MySQL 的默认隔离界别

可重复读：事务在执行过程中可以读取到其他事务已提交的新插入的数据，但是不能读取其他事务对数据的修改，也就是说多次读取同一记录的结果相同；该个里级别避免了脏读、不可重复度的问题，但是仍然无法避免幻读的问题

可重复读是MySQL默认的隔离级别

串行化：事务串行化执行，事务只能一个接着一个地执行,、，并且在执行过程中完全看不到其他事务对数据所做的更新；缺点是并发能力差，最严格的事务隔离，完全符合ACID原则，但是对性能影响比较大

| **事务隔离级别** | **脏读** | **不可重复读** | **幻读** |
| --- | --- | --- | --- |
| 读未提交（read-uncommitted） | 是 | 是 | 是 |
| 读已提交（read-committed） | 否 | 是 | 是 |
| 可重复读（repeatable-read） | 否 | 否 | 是 |
| 串行化（serializable） | 否 | 否 | 否 |

## 执行事务的过程

**关闭自动提交**

SET autocommit=0;

**事务开启**

START TRANSACTION -- 标记一个事务的开始，从这个之后的sql都在同一个事务内

**成功则提交，失败则回滚**

-- 提交：持久化（成功）

COMMIT

-- 回滚：回到原来的样子（失败）

ROLLBACK

**其他操作**

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

# 11.索引

索引（Index）是帮助MySQL高效获取数据的**数据结构**。

* 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

## 索引的分类

例如student表

-- 创建学生表student

CREATE TABLE `student`(

`StudentNo` INT(4) NOT NULL COMMENT '学号',

`LoginPwd` VARCHAR(20) DEFAULT NULL,

`StudentName` VARCHAR(20) DEFAULT NULL COMMENT '学生姓名',

`Sex` TINYINT(1) DEFAULT NULL COMMENT '性别,取值0或1',

`GradeID` INT(11) DEFAULT NULL COMMENT '年级编号',

`Phone` VARCHAR(50) NOT NULL COMMENT '联系电话,允许为空,即可选输入',

`Adress` VARCHAR(255) NOT NULL COMMENT '地址,允许为空,即可选输入',

`BornDate` DATETIME DEFAULT NULL COMMENT '出生时间',

`Email` VARCHAR(50) NOT NULL COMMENT '邮箱账号,允许为空,即可选输入',

`IdentityCard` VARCHAR(18) DEFAULT NULL COMMENT '身份证号',

PRIMARY KEY (`StudentNo`),

UNIQUE KEY `IdentityCard` (`IdentityCard`),

KEY `Email` (`Email`)

)ENGINE=MYISAM DEFAULT CHARSET=utf8;

### 主键索引（PRIMARY KEY）

唯一的标识，主键不可重复，只有一个列作为主键

* 最常见的索引类型，不允许为空值
* 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

-- 创建表的时候指定主键索引

CREATE TABLE tableName(

......

PRIMARY INDEX (columeName)

)

-- 修改表结构添加主键索引

ALTER TABLE tableName ADD PRIMARY INDEX (columnName)

### 普通索引（KEY / INDEX）

默认的，快速定位特定数据

* 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

-- 直接创建普通索引

CREATE INDEX indexName ON tableName (columnName)

-- 创建表的时候指定普通索引

CREATE TABLE tableName(

......

INDEX [indexName] (columeName)

)

-- 修改表结构添加普通索引

ALTER TABLE tableName ADD INDEX indexName(columnName)

### 唯一索引（UNIQUE KEY）

它与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值

与主键索引的区别：主键索引只能有一个、唯一索引可以有多个

-- 直接创建唯一索引

CREATE UNIQUE INDEX indexName ON tableName(columnName)

-- 创建表的时候指定唯一索引

CREATE TABLE tableName(

......

UNIQUE INDEX [indexName] (columeName)

);

-- 修改表结构添加唯一索引

ALTER TABLE tableName ADD UNIQUE INDEX [indexName] (columnName)

### 全文索引（FULLText）

快速定位特定数据（百度搜索就是全文索引）

* 在特定的数据库引擎下才有：MyISAM
* 只能用于CHAR , VARCHAR , TEXT数据列类型
* 适合大型数据集

-- 增加一个全文索引

ALTER TABLE `student` ADD FULLTEXT INDEX `StudentName`(`StudentName`);

-- EXPLAIN 分析sql执行的情况

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

EXPLAIN SELECT \* FROM student WHERE MATCH(StudentName) AGAINST('d'); -- 全文索引

## 索引的使用

### 索引的创建

* 在创建表的时候给字段增加索引

CREATE TABLE 表名 (

字段名1 数据类型 [完整性约束条件…],

字段名2 数据类型 [完整性约束条件…],

[UNIQUE|FULLTEXT|SPATIAL] INDEX|KEY [索引名] (字段名[(长度)] [ASC |DESC])

);

* 创建完毕后，增加索引

-- 方法一：CREATE在已存在的表上创建索引

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX 索引名

ON 表名 (字段名[(长度)] [ASC |DESC]) ;

-- 方法二：ALTER TABLE在已存在的表上创建索引

ALTER TABLE 表名 ADD [UNIQUE|FULLTEXT|SPATIAL]

INDEX 索引名 (字段名[(长度)] [ASC |DESC]) ;

### 索引的删除

-- 删除索引

DROP INDEX 索引名 ON 表名;

-- 删除主键索引

ALTER TABLE 表名 DROP PRIMARY KEY;

### 显示索引信息

SHOW INDEX FROM 表名;

### explain分析sql执行的情况

-- 增加一个全文索引

ALTER TABLE `student` ADD FULLTEXT INDEX `StudentName`(`StudentName`);

-- EXPLAIN 分析sql执行的情况

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

EXPLAIN SELECT \* FROM student WHERE MATCH(StudentName) AGAINST('d'); -- 全文索引

## 测试索引（可以不看）

**建表app\_user：**

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 UPDATECURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

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

**批量插入数据：100w**

-- 1418错解决方案(创建函数前执行此语句)

set global log\_bin\_trust\_function\_creators=true;

-- 插入100万条数据

DELIMITER $$ -- 写函数之前要写的标志

CREATE FUNCTION mock\_data() -- 创建mock\_data()函数

RETURNS INT

BEGIN

DECLARE num INT DEFAULT 1000000;

DECLARE i INT DEFAULT 0;

WHILE i < num DO

INSERT INTO app\_user(`name`, `email`, `phone`, `gender`, `password`, `age`)

VALUES(CONCAT('用户', i), '24736743@qq.com', CONCAT('18', FLOOR(RAND()\*(999999999-100000000)+100000000)),FLOOR(RAND()\*2),UUID(), FLOOR(RAND()\*100));

SET i = i + 1;

END WHILE;

RETURN i;

END;

-- 执行函数

SELECT mock\_data();

**测试查询速度**

-- 查询用户名为'用户9999'性能分析

EXPLAIN SELECT \* FROM app\_user where name='用户99999'

**增加索引后测试**

-- 给name列创建常规索引

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

-- 再测试

EXPLAIN SELECT \* FROM app\_user where name='用户99999'

## 索引原则

* 索引不是越多越好，小数据量的表不需要加索引
* 不要对经常变动的数据增加索引
* 索引一般加在经常要查询的列上

# 12. 数据库驱动和JDBC

之后补上