**Mysql数据与sql:**

数据库：存储数据的仓库

mysql:开源，免费，跨平台

登录数据库： mysql -u 账号 -p密码

mysql -h 服务器 -P 端口号 -u 用户名 -p 密码

-h:默认为127.0.0.1 -P:默认为 3306

退出命令

quit exit \q

sql规范：

sql命令不区分大小写

一条语句以;作为结束符

单行注释 --

多行注释 /\* ...... \*/

DDL:数据操作语言

DML：数据操作语言

DCL:数据控制语言

数据库命令：

显示数据库 show databases;

删除数据库 drop database 数据库名;

创建数据库 create database 数据库名；

create database if not exists 数据库名；

create database if not exists 数据库名；

create database if not exists 数据库名 character set utf8

要想修改数据库编码 alter database 数据库名 set character set gbk

显示创建信息 show create database;

显示当前操作数据库 select database();

切换数据库 use 数据库名

char(3) 表示定长为3个字节的字符串

varchar(3) 表示长为1到3个字节的字符串

在tuf8里面，一个汉字占3个字节

double(7,2) 表示一共有7位数，小数点后面有两位

查看表结构 desc 表名

show columns from 表名

查看表创建的信息 show create table表名

创建表：

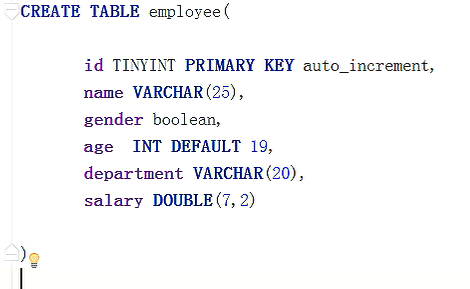
create table 表名(

列名 类型 约束，

列名 类型 约束，

列名 类型 约束，

)



修改表结构：

增加一个字段

alter table employee add is\_married tinyint(1);

删除字段

alter table employee drop A; A:表示一个字段

-- 1.创建表（类似于一个excel表）

create table tab\_name(

field1 type[完整性约束条件],

field2 type,

...

fieldn type

)[character set xxx];

-- 创建一个员工表employee

create table employee(

id int primary key auto\_increment ,

name varchar(**20**),

gender bit default **1**, -- gender char(1) default 1 ----- 或者 TINYINT(1) birthday date,

entry\_date date,

job varchar(**20**),

salary double(4,2) unsigned,

resume text -- 注意，这里作为最后一个字段不加逗号 );

/\* 约束:

primary key (非空且唯一) :能够唯一区分出当前记录的字段称为主键！

unique

not null

auto\_increment 主键字段必须是数字类型。

外键约束 foreign key \*/

-- 2.查看表信息

desc tab\_name 查看表结构

show columns from tab\_name 查看表结构

show tables 查看当前数据库中的所有的表

show create table tab\_name 查看当前数据库表建表语句

3.修改表结构

-- (1)增加列(字段)

alter table tab\_name add [column] 列名 类型［完整性约束条件］［first｜after 字段名］;

alter table user add addr varchar(**20**) not null unique first/after username;

#添加多个字段

alter table users2

add addr varchar(**20**),

add age int first,

add birth varchar(**20**) after name;

-- (2)修改一列类型

# 添加first表示修改之后放在首行，添加after需要指定放在哪一列后面

alter table tab\_name modify 列名 类型 [完整性约束条件]［first｜after 字段名］;

alter table users2 modify age tinyint default **20**;

alter table users2 modify age int after id;

-- (3)修改列名

alter table tab\_name change [column] 列名 新列名 类型 [完整性约束条件]［first｜after 字段名］;

alter table users2 change age Age int default **28** first;

-- (4)删除一列

alter table tab\_name drop [column] 列名;

-- 思考：删除多列呢？删一个填一个呢？

alter table users2

add salary float(**6**,**2**) unsigned not null after name,

drop addr;

-- (5)修改表名

rename table 表名 to 新表名;

-- (6)修该表所用的字符集

alter table student character set utf8;

-- 4.删除表

drop table tab\_name;

---5 添加主键，删除主键

alter table tab\_name add primary key(字段名称,...)

alter table users drop primary key;

eg:

mysql> create table test5(num int auto\_increment);

ERROR **1075** (**42000**): Incorrect table definition; there can be only one auto column and it must be defined as a key

create table test(num int primary key auto\_increment);

-- 思考，如何删除主键？

alter table test modify id int; -- auto\_increment没了，但这样写主键依然存在，所以还要加上下面这句

alter table test drop primary key;-- 仅仅用这句也无法直接删除主键

-- 唯一索引

alter table tab\_name add unique [index|key] [索引名称](字段名称,...)

alter table users add unique(name)-- 索引值默认为字段名show create table users;

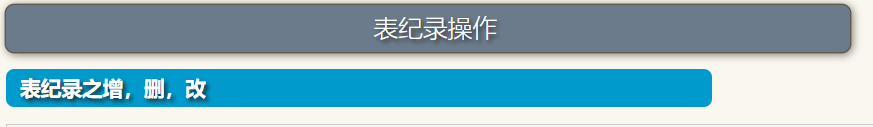
alter table users add unique key user\_name(name);-- 索引值为user\_name

-- 添加联合索引

alter table users add unique index name\_age(name,age);#show create table users;

-- 删除唯一索引

alter table tab\_name drop {index|key} index\_name



-- 1.增加一条记录insert

/\*insert ［into］ tab\_name (field1,filed2,.......) values (value1,value2,.......);\*/

create table employee\_new(

id int primary key auto\_increment,

name varchar(**20**) not null unique,

birthday varchar(**20**),  
 salary float(7,2)

);

insert into employee\_new (id,name,birthday,salary) values

(**1**,'yuan','1990-09-09',9000);

insert into employee\_new values

(**2**,'alex','1989-08-08',3000);

insert into employee\_new (name,salary) values

('xialv',1000);

-- 插入多条数据

insert into employee\_new values

(4,'alvin1','1993-04-20',3000),

(5,'alvin2','1995-05-12',5000);

-- set插入: insert ［into］ tab\_name set 字段名=值

insert into employee\_new set id=12,name="alvin3";

-- 2.修改表记录 update tab\_name set field1=value1,field2=value2,......[where 语句]

/\* UPDATE语法可以用新值更新原有表行中的各列。

SET子句指示要修改哪些列和要给予哪些值。

WHERE子句指定应更新哪些行。如没有WHERE子句，则更新所有的行。\*/

update employee\_new set birthday="1989-10-24" WHERE id=**1**;

--- 将yuan的薪水在原有基础上增加1000元。

update employee\_new set salary=salary+**4000** where name='yuan';

-- 3.删除表纪录

delete from tab\_name [where ....]

/\* 如果不跟where语句则删除整张表中的数据

delete只能用来删除一行记录

delete语句只能删除表中的内容，不能删除表本身，想要删除表，用**drop**

**TRUNCATE TABLE也可以删除表中的所有数据，词语句首先摧毁表，再新建表。此种方式删除的数据不能在事务中恢复。\*/**

-- 删除表中名称为’alex’的记录。

delete from employee\_new where name='alex';

-- 删除表中所有记录。

delete from employee\_new;-- 注意auto\_increment没有被重置:alter table employee auto\_increment=1;

-- 使用truncate删除表中记录。

truncate table emp\_new;



-- 查询表达式

SELECT \*|field1,filed2 ... FROM tab\_name

WHERE 条件

GROUP BY field

HAVING 筛选

ORDER BY field

LIMIT 限制条数

---准备表

CREATE TABLE ExamResult(

id INT PRIMARY KEY auto\_increment,

name VARCHAR (**20**),

JS DOUBLE ,

Django DOUBLE ,

OpenStack DOUBLE

);

INSERT INTO ExamResult VALUES (**1**,"yuan",**98**,**98**,**98**),

(**2**,"xialv",**35**,**98**,**67**),

(**3**,"alex",**59**,**59**,**62**),

(**4**,"wusir",**88**,**89**,**82**),

(**5**,"alvin",**88**,**98**,**67**),

(**6**,"yuan",**86**,**100**,**55**);

-- （1）select [distinct] \*|field1，field2，...... from tab\_name

-- 其中from指定从哪张表筛选，\*表示查找所有列，也可以指定一个列

-- 表明确指定要查找的列，distinct用来剔除重复行。

-- 查询表中所有学生的信息。

select \* from ExamResult;

-- 查询表中所有学生的姓名和对应的英语成绩。

select name,JS from ExamResult;

-- 过滤表中重复数据。

select distinct JS ,name from ExamResult;

-- （2）select 也可以使用表达式，并且可以使用: 字段 as 别名或者:字段 别名

-- 在所有学生分数上加10分特长分显示。

select name,JS+**10**,Django+**10**,OpenStack+**10** from ExamResult;

-- 统计每个学生的总分。

select name,JS+Django+OpenStack from ExamResult;

-- 使用别名表示学生总分。

select name as 姓名,JS+Django+OpenStack as 总成绩 from ExamResult;

select name,JS+Django+OpenStack 总成绩 from ExamResult;

select name JS from ExamResult; -- what will happen?---->记得加逗号

-- （3）使用where子句，进行过滤查询。

-- 查询姓名为XXX的学生成绩

select \* from ExamResult where name='yuan';

-- 查询英语成绩大于90分的同学

select id,name,JS from ExamResult where JS>**90**;

-- 查询总分大于200分的所有同学

select name,JS+Django+OpenStack as 总成绩 from

ExamResult where JS+Django+OpenStack>**200** ;

-- where字句中可以使用：

-- 比较运算符：

> < >= <= <> !=

between **80** and **100** 值在10到20之间

in(**80**,**90**,**100**) 值是10或20或30

like 'yuan%'

/\*

pattern可以是%或者\_，

如果是%则表示任意多字符，此例如唐僧,唐国强

如果是\_则表示一个字符唐\_，只有唐僧符合。两个\_则表示两个字符：\_\_

\*/

-- 逻辑运算符

在多个条件直接可以使用逻辑运算符 and or not

-- 练习

-- 查询JS分数在 70－100之间的同学。

select name ,JS from ExamResult where JS between **80** and **100**;

-- 查询Django分数为75,76,77的同学。

select name ,Django from ExamResult where Django in (**75**,**98**,**77**);

-- 查询所有姓王的学生成绩。

select \* from ExamResult where name like '王%';

-- 查询JS分>90，Django分>90的同学。

select id,name from ExamResult where JS>**90** and Django >**90**;

-- 查找缺考数学的学生的姓名

select name from ExamResult where Database is null;

-- （4）Order by 指定排序的列，排序的列即可是表中的列名，也可以是select 语句后指定的别名。

-- select \*|field1,field2... from tab\_name order by field [Asc|Desc]

-- Asc 升序、Desc 降序，其中asc为默认值 ORDER BY 子句应位于SELECT语句的结尾。

-- 练习：

-- 对JS成绩排序后输出。

select \* from ExamResult order by JS;

-- 对总分排序按从高到低的顺序输出

select name ,(ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(Database,**0**))

总成绩 from ExamResult order by 总成绩 desc;

-- 对姓李的学生成绩排序输出

select name ,(ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(OpenStack,**0**))

总成绩 from ExamResult where name like 'a%'

order by 总成绩 desc;

-- （5）group by 分组查询：

CREATE TABLE order\_menu(

id INT PRIMARY KEY auto\_increment,

product\_name VARCHAR (**20**),

price FLOAT(**6**,**2**),

born\_date DATE,

class VARCHAR (**20**)

);

INSERT INTO order\_menu (product\_name,price,born\_date,class) VALUES

("苹果",**20**,**20170612**,"水果"),

("香蕉",**80**,**20170602**,"水果"),

("水壶",**120**,**20170612**,"电器"),

("被罩",**70**,**20170612**,"床上用品"),

("音响",**420**,**20170612**,"电器"),

("床单",**55**,**20170612**,"床上用品"),

("草莓",**34**,**20170612**,"水果");

-- 注意,按分组条件分组后每一组只会显示第一条记录

-- group by字句，其后可以接多个列名，也可以跟having子句,对group by 的结果进行筛选。

-- 按位置字段筛选

select \* from order\_menu group by **5**;

-- 练习：对购物表按类名分组后显示每一组商品的价格总和

select class,SUM(price)from order\_menu group by class;

-- 练习：对购物表按类名分组后显示每一组商品价格总和超过150的商品

select class,SUM(price)from order\_menu group by class

HAVING SUM(price)>**150**;

/\*

having 和 where两者都可以对查询结果进行进一步的过滤，差别有：

<1>where语句只能用在分组之前的筛选，having可以用在分组之后的筛选；

<2>使用where语句的地方都可以用having进行替换

<3>having中可以用聚合函数，where中就不行。

\*/

-- GROUP\_CONCAT() 函数

SELECT id,GROUP\_CONCAT(name),GROUP\_CONCAT(JS) from ExamResult GROUP BY id;

-- （6）聚合函数： 先不要管聚合函数要干嘛，先把要求的内容查出来再包上聚合函数即可。

-- (一般和分组查询配合使用)

--<1> 统计表中所有记录

-- COUNT(列名)：统计行的个数

-- 统计一个班级共有多少学生？先查出所有的学生，再用count包上

select count(\*) from ExamResult;

-- 统计JS成绩大于70的学生有多少个？

select count(JS) from ExamResult where JS>**70**;

-- 统计总分大于280的人数有多少？

select count(name) from ExamResult

where (ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(OpenStack,**0**))>**280**;

-- 注意:count(\*)统计所有行; count(字段)不统计null值.

-- SUM(列名)：统计满足条件的行的内容和

-- 统计一个班级JS总成绩？先查出所有的JS成绩，再用sum包上

select JS as JS总成绩 from ExamResult;

select sum(JS) as JS总成绩 from ExamResult;

-- 统计一个班级各科分别的总成绩

select sum(JS) as JS总成绩,

sum(Django) as Django总成绩,

sum(OpenStack) as OpenStack from ExamResult;

-- 统计一个班级各科的成绩总和

select sum(ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(Database,**0**))

as 总成绩 from ExamResult;

-- 统计一个班级JS成绩平均分

select sum(JS)/count(\*) from ExamResult ;

-- 注意：sum仅对数值起作用，否则会报错。

-- AVG(列名)：

-- 求一个班级JS平均分？先查出所有的JS分，然后用avg包上。

select avg(ifnull(JS,**0**)) from ExamResult;

-- 求一个班级总分平均分

select avg((ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(Database,**0**)))

from ExamResult ;

-- Max、Min

-- 求班级最高分和最低分（数值范围在统计中特别有用）

select Max((ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(OpenStack,**0**)))

最高分 from ExamResult;

select Min((ifnull(JS,**0**)+ifnull(Django,**0**)+ifnull(OpenStack,**0**)))

最低分 from ExamResult;

-- 求购物表中单价最高的商品名称及价格

---SELECT id, MAX(price) FROM order\_menu;--id和最高价商品是一个商品吗?

SELECT MAX(price) FROM order\_menu;

-- 注意：null 和所有的数计算都是null，所以需要用ifnull将null转换为0！

-- -----ifnull(JS,0)

-- with rollup的使用

--<2> 统计分组后的组记录

-- （7） 重点：Select from where group by having order by

-- Mysql在执行sql语句时的执行顺序：

-- from where select group by having order by

-- 分析:

select JS as JS成绩 from ExamResult where JS成绩 >**70**; ---- 不成功

select JS as JS成绩 from ExamResult having JS成绩 >**90**; --- 成功

-- (8) limit

SELECT \* from ExamResult limit **1**;

SELECT \* from ExamResult limit **2**,**5**;--跳过前两条显示接下来的五条纪录

SELECT \* from ExamResult limit **2**,**2**;

--- (9) 使用正则表达式查询

SELECT \* FROM employee WHERE emp\_name REGEXP '^yu';

SELECT \* FROM employee WHERE emp\_name REGEXP 'yun$';

SELECT \* FROM employee WHERE emp\_name REGEXP 'm{2}';



--- 每一个班主任会对应多个学生 , 而每个学生只能对应一个班主任

----主表

CREATE TABLE ClassCharger(

id TINYINT PRIMARY KEY auto\_increment,

name VARCHAR (**20**),

age INT ,

is\_marriged boolean -- show create table ClassCharger: tinyint(1)

);

INSERT INTO ClassCharger (name,age,is\_marriged) VALUES ("冰冰",**12**,**0**),

("丹丹",**14**,**0**),

("歪歪",**22**,**0**),

("姗姗",**20**,**0**),

("小雨",**21**,**0**);

----子表

CREATE TABLE Student(

id INT PRIMARY KEY auto\_increment,

name VARCHAR (**20**),

charger\_id TINYINT --切记:作为外键一定要和关联主键的数据类型保持一致

-- [ADD CONSTRAINT charger\_fk\_stu]FOREIGN KEY (charger\_id) REFERENCES ClassCharger(id)

) ENGINE=INNODB;

INSERT INTO Student(name,charger\_id) VALUES ("alvin1",**2**),

("alvin2",**4**),

("alvin3",5),

("alvin4",**3**),

("alvin5",5),

("alvin6",**3**),

("alvin7",**2**);

DELETE FROM ClassCharger WHERE name="冰冰";INSERT student (name,charger\_id) VALUES ("yuan",**1**);-- 删除居然成功,可是 alvin3显示还是有班主任id=1的冰冰的;

-----------增加外键和删除外键---------

ALTER TABLE student ADD CONSTRAINT abc

FOREIGN KEY(charger\_id)

REFERENCES classcharger(id);

ALTER TABLE student DROP FOREIGN KEY abc;



--外键约束对子表的含义: 如果在父表中找不到候选键,则不允许在子表上进行insert/update

--外键约束对父表的含义: 在父表上进行update/delete以更新或删除在子表中有一条或多条对

-- 应匹配行的候选键时,父表的行为取决于：在定义子表的外键时指定的

-- on update/on delete子句

-----------------innodb支持的四种方式---------------------------------------

-----cascade方式 在父表上update/delete记录时，同步update/delete掉子表的匹配记录

-----外键的级联删除：如果父表中的记录被删除，则子表中对应的记录自动被删除--------

FOREIGN KEY (charger\_id) REFERENCES ClassCharger(id)

ON DELETE CASCADE

------set null方式 在父表上update/delete记录时，将子表上匹配记录的列设为null

-- 要注意子表的外键列不能为not null

FOREIGN KEY (charger\_id) REFERENCES ClassCharger(id)

ON DELETE SET NULL

------Restrict方式 :拒绝对父表进行删除更新操作(了解)

------No action方式 在mysql中同Restrict,如果子表中有匹配的记录,则不允许对父表对应候选键

-- 进行update/delete操作（了解）



-- 准备两张表

-- company.employee

-- company.department

create table employee(

emp\_id int auto\_increment primary key not null,

emp\_name varchar(**50**),

age int,

dept\_id int

);

insert into employee(emp\_name,age,dept\_id) values

('A',**19**,**200**),

('B',**26**,**201**),

('C',**30**,**201**),

('D',**24**,**202**),

('E',**20**,**200**),

('F',**38**,**204**);

create table department(

dept\_id int,

dept\_name varchar(**100**)

);

insert into department values

(**200**,'人事部'),

(**201**,'技术部'),

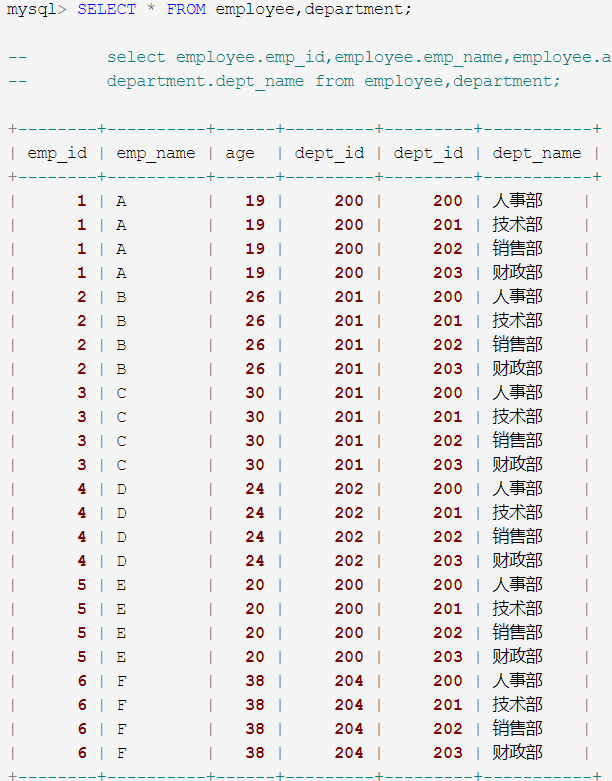
(**202**,'销售部'),

(**203**,'财政部');

mysql> select \* from employee;

mysql> select \* from department;





内连接：



外连接：







-- 2. 带比较运算符的子查询

-- =、!=、>、>=、<、<=、<>

-- 查询员工年龄大于等于25岁的部门

select dept\_id,dept\_name from department

where dept\_id IN

(select DISTINCT dept\_id from employee where age>=**25**);

-- 3. 带EXISTS关键字的子查询

-- EXISTS关字键字表示存在。在使用EXISTS关键字时，内层查询语句不返回查询的记录。

-- 而是返回一个真假值。Ture或False

-- 当返回Ture时，外层查询语句将进行查询；当返回值为False时，外层查询语句不进行查询

select \* from employee

WHERE EXISTS

(SELECT dept\_name from department where dept\_id=**203**);

--department表中存在dept\_id=203，Ture

select \* from employee

WHERE EXISTS

(SELECT dept\_name from department where dept\_id=**205**);

-- Empty set (0.00 sec)  
  
  
 ps: create table t1(select \* from t2);

感谢：Yuan先生的提供原链接：

（<https://www.cnblogs.com/yuanchenqi/articles/6357507.html>）

注意：



group by 先分组，再利用聚合函数sum(),再通过 having 过滤

where 放在group 前面，having 对分组前的数据过滤

having 放在group 后面，having 对分组后的数据过滤

# 如果js的成绩为null,那么计算时就默认为0

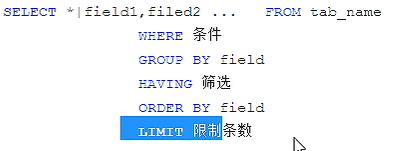
ifnull(JS,0)

limit n

显示前n条数据，

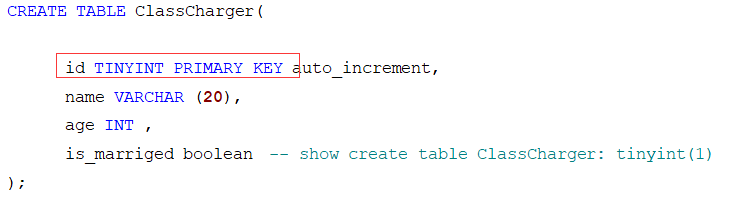
limit n,m

跳过前n条，显示m条数据（limit 2,2）

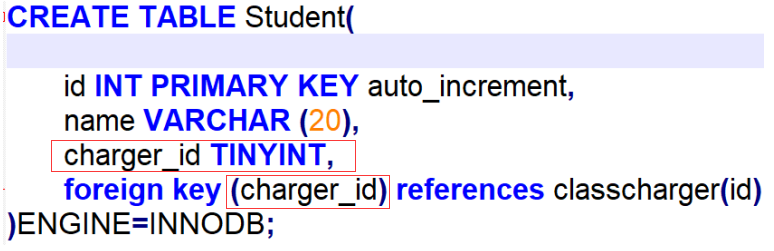


多表连接与查询：

**老师表：**



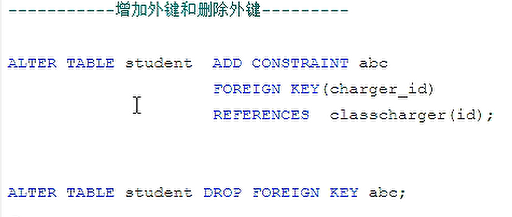
**学生表：**



创建外键： foreign key (charger\_id) references ClassCharger(id)

需要注意： 作为外键一定要和关联主键的类型一致

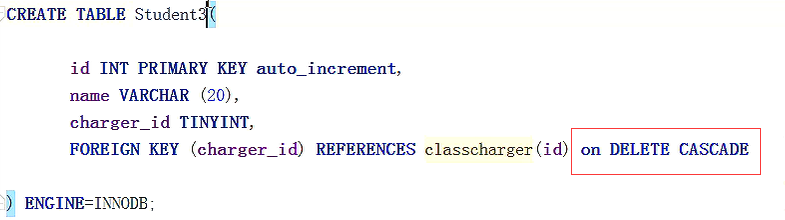
通过sql语句增加/删除外键



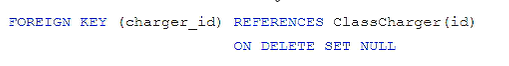
INNODB支持的on语句

——级联删除与set null:

级联删除：删除父表，与之关联的字表的值将会被删除



Set null: 删除父表，与之关联的字表的值将会设置为NULL



多表查询之连接查询：

连接查询：

全连接：（笛卡儿积）：SELECT \* FROM employee,department;

内连接：-- 查询两张表中都有的关联数据,相当于利用条件从笛卡尔积结果中筛选出了正确的结果。

select \* from employee,department where employee.dept\_id = department.dept\_id;

或者:select \* from employee inner join department on employee.dept\_id = department.dept\_id;

外连接：

左外连接--（1）左外连接：在内连接的基础上增加左边有右边没有的结果

select \* from employee left join department on employee.dept\_id = department.dept\_id;

右外连接--（2）右外连接：在内连接的基础上增加右边有左边没有的结果

select \* from employee right join department on employee.dept\_id = department.dept\_id;

全连接--（3)在内连接的基础上增加左边有右边没有的和右边有左边没有的结果

select \* from employee full join department on employee.dept\_id = department.dept\_id;(mysql不支持)

但是可以通过把左外连接和右外连接拼接起来

select \* from employee left join department on employee.dept\_id = department.dept\_id union

select \* from employee full join department on employee.dept\_id = department.dept\_id;

多表查询之子查询：

1. 带IN关键字的子查询

---查询employee表，但dept\_id必须在department表中出现过

select \* from employee

where dept\_id IN

(select dept\_id from department);

1. 带比较运算符的子查询

-- 查询员工年龄大于等于25岁的部门

select dept\_id,dept\_name from department

where dept\_id IN

(select DISTINCT dept\_id from employee where age>=**25**);

1. 带EXISTS关键字的子查询

select \* from employee

WHERE EXISTS

(SELECT dept\_name from department where dept\_id=**203**);

--department表中存在dept\_id=203，Ture

select \* from employee

WHERE EXISTS

(SELECT dept\_name from department where dept\_id=**205**);

-- EXISTS关字键字表示存在。在使用EXISTS关键字时，内层查询语句不返回查询的记录。

**-- 而是返回一个真假值。Ture或False**

**-- 当返回Ture时，外层查询语句将进行查询；当返回值为False时，外层查询语句不进行查询**

索引：

索引特点：创建和维护会消耗很多时间与磁盘空间，但查询速度大大提高

根据下面两种算法根据索引内容

B+树:

Hash:

**普通索引：**

create table test(

id int,

name varchar(20),

resume varchar(50),

**index/key 索引名 (字段名)**

);

表结构里面就有

KEY `index\_name` (`name`)

**唯一索引：**

create table test(

id int,

name varchar(20),

resume varchar(50),

**unique index/key 索引名 (字段名)**

);

主键唯一索引如增加name字段为唯一索引

alter table 表 modify name varchar(25) unique;

**全文索引：**

create table test(

id int,

name varchar(20),

resume varchar(50),

**fulltext index/key 索引名 (字段名)**

);

**多列索引（即多个字段）：**

create table test(

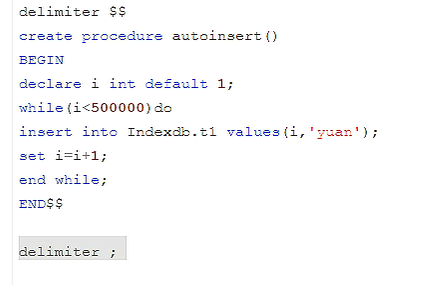
id int,

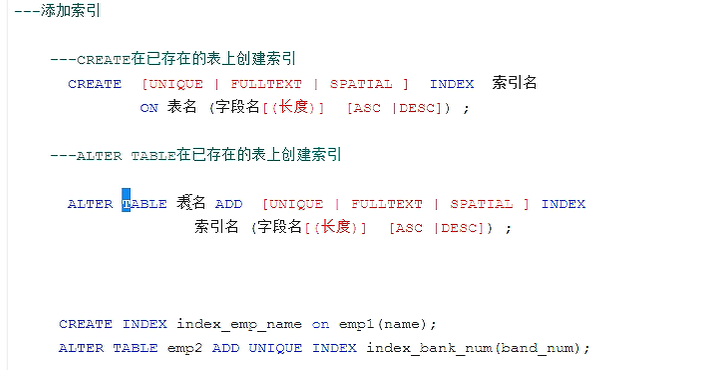
name varchar(20),

resume varchar(50),

**fulltext index/key 索引名 (name,resume)**

);



测试：

delimiter 定义分隔符

事物：

**<1> 原子性**（Atomicity）：事务中的操作要么都发生，要么都不发生。

**<2> 一致性**（Consistency）：事务前后数据的完整性必须保持一致。在事务执行之前数据库是符合数据完整性约束的，无论事务是否执行成功，事务结束后的数据库中的数据也应该是符合完整性约束的。

**<3> 隔离性（**Isolation）：事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离。

**脏读：**-一个事务读取到了另一个事务未提交的数据,这是特别危险的，要尽力防止

**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。（一个事务读取到了另一个事务已经提交的数据--增加记录、删除记录、修改记录），在某写情况下并不是问题，在另一些情况下就是问题。

**虚读：**虚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。（一个事务读取到了另一个事务已经提交的数据---增加记录、删除记录），在某写情况下并不是问题，在另一些情况下就是问题。

**<4> 持久性（Durability）**：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的

开启两个事物：（同一个mysql中操作同一表）系统将自动提交第一个事物进行的操作，

（在不同mysql中操作同一表）将不会受影响(隔离性)。

事务指逻辑上的一组操作，组成这组操作的各个单元，要不全部成功，要不全部不成功。

--  **start transaction** 开启事务

-- **Rollback** 回滚事务,即撤销指定的sql语句(只能回退insert delete update语句)，回滚到上一次commit的位置

-- **Commit** 提交事务，提交未存储的事务

--

-- **savepoint** 保留点 ，事务处理中设置的临时占位符 你可以对它发布回退(与整个事务回退不同)

事物一般流程：

start transaction; 开启事务

执行操作(insert delete update语句)

发现操作失败，需要取消操作则需要回滚事物 ，默认直接回滚到开启事物的点上，通过设置保留点（savepoint）回滚到保留点的位置

回滚到rollback;

然后执行正确的操作并提交

commit;

注意：开始事物之后，如果**没有提交事务（commit）,数据就没有真正写入数据库**

**(rollback返回到顶时，再次rollback将不会起作用)**

详情请看：<https://www.cnblogs.com/yuanchenqi/articles/6437362.html>



**利用pymysql操作数据库以及如何开启事务**