**SQL入门二**

# mysqld进程

mysqld进程由连接层（网络），SQL层（内存），存储引擎层（磁盘）组成

## sql执行过程

### SQL层

1、接收上层（连接层）转发过来的SQL。

2、语法检查

3、语义检查（检查是什么类型的SQL），以Select为例，检查到时一个DQL，交给专用的解析器

注：SQL主要包含：DCL、DDL、DML、DQL

3.1、权限验证

4、解析器将SQL解析成SQL接口能够识别方式（执行计划explain），解析完成后，交给执优化器进行优化

5、优化器，做判断，选择一个他认为成本最低执行计划，交给执行器

---优化器本身就是一个程序，几乎做到了极致，可能性几乎都考虑到了。

---剩下的优化项主要是使用者的优化（执行的语句不一样）

---所以，最大化适应优化器程序特点的优化方案，大概率是最优的。。。。

6、执行器执行explain，生成执行结果（去将A个数据文件的第N个数据页把我需要的数据给我取出来）

7、把执行结果交给下层（存储引擎层）继续处理

8、查询缓存（query\_cache）

---把经常查询的数据放在内存中，提升速率

### 存储引擎层

1、接收上层的执行结果

2、取出磁盘文件的相应数据

3、返回给SQL层，结构化之后（生成表格），由专用线程 A thread，返回给客户端

# SQL操作规范

## 创建表规范—避免出错

1.库名不能出现大写

2.库名不能以数字开头

3.库名要和业务功能香瓜

4.建库要加字符集

## 查询规范—性能优化

1.使用select查询时，尽量不要使用\*

2.使用select时，使用where限定条件，明确上下限

3.避免使用逻辑‘非’---因为会影响到索引的使用

# DDL语句

mysql> help create table

Name: 'CREATE TABLE'

Description:

Syntax:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

(create\_definition,...)

[table\_options]

[partition\_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

[(create\_definition,...)]

[table\_options]

[partition\_options]

[IGNORE | REPLACE]

[AS] query\_expression

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name

{ LIKE old\_tbl\_name | (LIKE old\_tbl\_name) }

create\_definition:

col\_name column\_definition

| [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)

[index\_option] ...

| {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...)

[index\_option] ...

| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]

[index\_name] [index\_type] (index\_col\_name,...)

[index\_option] ...

| {FULLTEXT|SPATIAL} [INDEX|KEY] [index\_name] (index\_col\_name,...)

[index\_option] ...

| [CONSTRAINT [symbol]] FOREIGN KEY

[index\_name] (index\_col\_name,...) reference\_definition

| CHECK (expr)

column\_definition:

data\_type [NOT NULL | NULL] [DEFAULT default\_value]

[AUTO\_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]

[COMMENT 'string']

[COLUMN\_FORMAT {FIXED|DYNAMIC|DEFAULT}]

[STORAGE {DISK|MEMORY|DEFAULT}]

[reference\_definition]

data\_type:

BIT[(length)]

| TINYINT[(length)] [UNSIGNED] [ZEROFILL]

| SMALLINT[(length)] [UNSIGNED] [ZEROFILL]

| MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]

| INT[(length)] [UNSIGNED] [ZEROFILL]

| INTEGER[(length)] [UNSIGNED] [ZEROFILL]

| BIGINT[(length)] [UNSIGNED] [ZEROFILL]

| REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]

| DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]

| FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]

| DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]

| NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]

| DATE

| TIME[(fsp)]

| TIMESTAMP[(fsp)]

| DATETIME[(fsp)]

| YEAR

| CHAR[(length)] [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| VARCHAR(length) [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| BINARY[(length)]

| VARBINARY(length)

| TINYBLOB

| BLOB

| MEDIUMBLOB

| LONGBLOB

| TINYTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| TEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| MEDIUMTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| LONGTEXT [BINARY]

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| ENUM(value1,value2,value3,...)

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| SET(value1,value2,value3,...)

[CHARACTER SET charset\_name] [COLLATE collation\_name]

| spatial\_type

index\_col\_name:

col\_name [(length)] [ASC | DESC]

index\_type:

USING {BTREE | HASH}

index\_option:

KEY\_BLOCK\_SIZE [=] value

| index\_type

| WITH PARSER parser\_name

| COMMENT 'string'

reference\_definition:

REFERENCES tbl\_name (index\_col\_name,...)

[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

[ON DELETE reference\_option]

[ON UPDATE reference\_option]

reference\_option:

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table\_options:

table\_option [[,] table\_option] ...

table\_option:

AUTO\_INCREMENT [=] value

| AVG\_ROW\_LENGTH [=] value

| [DEFAULT] CHARACTER SET [=] charset\_name

| CHECKSUM [=] {0 | 1}

| [DEFAULT] COLLATE [=] collation\_name

| COMMENT [=] 'string'

| CONNECTION [=] 'connect\_string'

| {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'

| DELAY\_KEY\_WRITE [=] {0 | 1}

| ENGINE [=] engine\_name

| INSERT\_METHOD [=] { NO | FIRST | LAST }

| KEY\_BLOCK\_SIZE [=] value

| MAX\_ROWS [=] value

| MIN\_ROWS [=] value

| PACK\_KEYS [=] {0 | 1 | DEFAULT}

| PASSWORD [=] 'string'

| ROW\_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}

| STATS\_AUTO\_RECALC [=] {DEFAULT|0|1}

| STATS\_PERSISTENT [=] {DEFAULT|0|1}

| STATS\_SAMPLE\_PAGES [=] value

| TABLESPACE tablespace\_name [STORAGE {DISK|MEMORY|DEFAULT}]

| UNION [=] (tbl\_name[,tbl\_name]...)

partition\_options:

PARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY [ALGORITHM={1|2}] (column\_list)

| RANGE{(expr) | COLUMNS(column\_list)}

| LIST{(expr) | COLUMNS(column\_list)} }

[PARTITIONS num]

[SUBPARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY [ALGORITHM={1|2}] (column\_list) }

[SUBPARTITIONS num]

]

[(partition\_definition [, partition\_definition] ...)]

partition\_definition:

PARTITION partition\_name

[VALUES

{LESS THAN {(expr | value\_list) | MAXVALUE}

|

IN (value\_list)}]

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

[(subpartition\_definition [, subpartition\_definition] ...)]

subpartition\_definition:

SUBPARTITION logical\_name

[[STORAGE] ENGINE [=] engine\_name]

[COMMENT [=] 'comment\_text' ]

[DATA DIRECTORY [=] 'data\_dir']

[INDEX DIRECTORY [=] 'index\_dir']

[MAX\_ROWS [=] max\_number\_of\_rows]

[MIN\_ROWS [=] min\_number\_of\_rows]

[TABLESPACE [=] tablespace\_name]

[NODEGROUP [=] node\_group\_id]

query\_expression:

SELECT ... (Some valid select or union statement)

CREATE TABLE creates a table with the given name. You must have the

CREATE privilege for the table.

By default, tables are created in the default database, using the

InnoDB storage engine. An error occurs if the table exists, if there is

no default database, or if the database does not exist.

For information about the physical representation of a table, see

http://dev.mysql.com/doc/refman/5.6/en/create-table-files.html.

URL: http://dev.mysql.com/doc/refman/5.6/en/create-table.html

## 列的属性

### basic desc

not null –非空

primary key :非空且唯一

unique key ：唯一

### extent

primary key和unique key的区别：

1.primary key 一个表中只能有一个primary key，会根据主键生成聚集索引阻止表

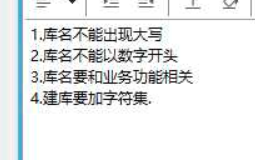
在存数据时，会按照主键的顺序（比如1 2 3 …）组织磁盘的存储顺序

---》未来按照主键条件查询时，会非常高效---因为它是一个有序的I/O

2.unique key 单纯的去重

## CREAT-创建表

### 创建表规范



### 创建student表

USE school;

CREATE TABLE student(

sno INT NOT NULL PRIMARY KEY AUTO\_INCREMENT COMMENT '学号' ,

sname VARCHAR(20) NOT NULL COMMENT '学生姓名' ,

sage TINYINT UNSIGNED NOT NULL COMMENT '年龄' ,

ssex ENUM('m','f','n') NOT NULL DEFAULT 'm' COMMENT '性别'

)ENGINE=INNODB CHARSET utf8;

DESC student;

SHOW CREATE TABLE student;

### 创建course表

CREATE TABLE course(

cno INT NOT NULL PRIMARY KEY COMMENT '课程编号',

cname VARCHAR(20) NOT NULL COMMENT '课程名称',

tno INT NOT NULL COMMENT '教师编号'

)ENGINE=INNODB CHARSET utf8;

SHOW TABLES

### 创建sc表

CREATE TABLE sc (

sno INT NOT NULL COMMENT '学号',

cno INT NOT NULL COMMENT '课程编号',

score INT NOT NULL DEFAULT '0'

)ENGINE=INNODB CHARSET utf8;

CREATE TABLE teacher (

tno INT NOT NULL PRIMARY KEY COMMENT '教师编号',

tname VARCHAR(20) NOT NULL COMMENT '教师名字'

)ENGINE=INNODB CHARSET utf8;

SHOW TABLES;

---测试的时候会使用以下复制表的操作

DESC student1;

SHOW CREATE TABLE student1;

CREATE TABLE stu1 LIKE student1;

DESC student1;

DESC stu1;

---使用绝对路径复制表结构一模一样的表

CREATE TABLE school.stu LIKE oldboy.stu1;

---创建表结构相同的并且数据也相同的表

crate TABLE t1 SELECT \* FROM stu;

---属于元数据的操作，显示是每一行，其实是一列的属性定义

---1.向表中添加列(默认最后一行)

ALTER TABLE stu ADD stel CHAR(11) NOT NULL UNIQUE COMMENT '手机号';

DESC stu;

## ALTER-操作列及列定义

### 向指定位置添加sid列(身份证号)

ALTER TABLE stu ADD sid CHAR(18) NOT NULL UNIQUE COMMENT '身份证号' AFTER age;

DESC stu;

### 向指定位置添加多列

ALTER TABLE student ADD gender VARCHAR(20),ADD qq INT;

### 在首列之前添加

ALTER TABLE stu ADD suser VARCHAR(10) NOT NULL UNIQUE COMMENT '用户名' FIRST;

DESC stu;

### 删除类及列定义

ALTER TABLE stu DROP suser;

DESC stu;

### ALTER--modify修改，列定义为时间类型

ALTER TABLE stu MODIFY stime DATETIME;

DESC stu;

### ALTER---change修改，列名称及列定义(未测试)

ALTER TABLE stu CHANGE stime st VARCHAR(30);

DESC stu;

### ALTER—RENAME-改表名

alter table sc1 rename sc;

### ALTER—修改某库下某表的存储引擎

alter table test.t1 engine=myisam

# DCL语句

## grant ---授权

grant select,delete,update on db\_name.\* to [user@’10.0.0.0%’identified](mailto:user@'10.0.0.0%25'identified) by ‘123’;

## revoke ---回收权限

revoke select on db\_name.\* from [user@’10.0.0.0%’identified](mailto:user@'10.0.0.0%25'identified);

# DML语句

## desc

name：数据操作语言

作用：操作的是表中的数据行，---------增删改查操作

## insert

### syntax

insert into 表名（列1，列2，列3） values(v1，v2….), (v1，v2….) #<==多行值

or

insert into 表明 value（v1,v2 …）#<==插入的值必须严格与列定义一致

--有多少列就有多少个value

### desc

一般有auto\_increment或default值的数据列，可以不需要指定插入值

## insert语句练习

### 1.向student表中添加value

USE oldboy;

DESC student;

INSERT INTO student(sname,sage,ssex)

VALUES

('zhang3',18,'m'),

('li4',19,'m'),

('xiaohong',18,'f');

DESC student;

SELECT \* FROM student;

### 2.先teacher中添加信息

DESC teacher;

INSERT INTO teacher(tno,tname) VALUES

(101,'oldboy'),

(102,'hesw'),

(103,'oldguo');

SELECT \* FROM teacher;

### 3.在course中插入数据信息

DESC course;

INSERT INTO course(cno,cname,tno)

VALUES

(1001,'linux','101'),

(1002,'python','102'),

(1003,'mysql','103');

SELECT \* FROM course;

### 4.成绩表录入信息

DESC sc;

SELECT \* FROM sc;

INSERT INTO sc(sno,cno,score)

VALUES

(1,1001,80),

(1,1002,59),

(2,1002,90),

(2,1003,100),

(3,1001,99),

(3,1003,40);

SELECT \* FROM sc;

## update语句练习---对数据库有一定的危险，但是可以挽回？？？；

USE oldboy;

### 1.给li4改名为‘lisi’

SELECT \* FROM student;

UPDATE student SET sname='lisi' WHERE sno=2;

### 2. sno=1学员的1001课程的分数改成89分

SELECT \* FROM sc;

UPDATE sc SET score=89 WHERE sno=1 AND cno=1001;

## delete

### desc

delete（逻辑层面删除，DDL？？？）

#逻辑删除，对文件内容读一行，删除一行，所以删除速度慢，适用于小表的删除；如果误删，可以恢复。

DELETE FROM student； #删除较小的表，不标准写法

DELETE FROM student WHERE sid=3 #标准的写法

### eg01

#### 1.先做备份

CREATE TABLE sc\_bak SELECT \* FROM sc;

DESC sc\_bak;

DESC sc;

SELECT \* FROM sc\_bak;

#### -2.delete测试删除1号学员的成绩信息

DELETE FROM sc\_bak WHERE sno=1;

---delete危险程度非常高---生产环境中直接屏蔽掉delete

---伪删除:使用状态列，实现伪删除，将表多加delete

---1)添加一个状态列state ENUM(1,0) NOT NULL DEFAULT 1

ALTER TABLE sc\_bak ADD state ENUM('1','0') NOT NULL DEFAULT '1';

---2）使用update替代delete，执行删除sno=2的信息

UPDATE sc\_bak SET state='0' WHERE sno=2;

---3)使用select屏蔽state='0'行信息==伪删除

SELECT \* FROM sc\_bak WHERE state='1';

# delete、truncate和drop删除类命令区别

## delete（逻辑层面删除，DDL？？？）

#逻辑删除，对文件内容读一行，删除一行，所以删除速度慢，适用于小表的删除；如果误删，可以恢复。

DELETE FROM student； #删除较小的表，不标准写法

DELETE FROM student WHERE sid=3 #标准的写法

## truncate table stu；（与DROP一样，从物理底层删除）

#从物理层面删除，将表删除，删除速度快，适用于大表删除；如果误删，只能找专业的数据公司恢复，且可能恢复得不完整

TRUNCATE TABLE student;

## drop {database|table}

#同truncate一样，从物理磁盘的数据页删除数据，速度很快

#---本质上是调用---rm -rf 命令==rm -rf frm ibd 元数据

# 如果使用drop误删除了文件—未解决问题---

不能重启、不能进行大量的文件录入

如何基于linux文件句柄恢复rm的文件

# DQL语句（SELECT SHOW）

## SELECT

### syntax

1）简化语法

SELECT 列 FROM 表 #<==select子表

2)标准语法

SELECT table.c1,table.c2 FROM 表;

3）查看当前登录的用户

ymq [world]>select current\_user();

+----------------+

| current\_user() |

+----------------+

| root@localhost |

+----------------+

1 row in set (0.01 sec)

或者

ymq [world]>select user();

## class

### 【环境】

USE world;

SHOW TABLES;

DESC city;

### 查询表中所有数据

SELECT \* FROM city; #<==尽量不要使用此语句--特别是数据量特别大的情形

### 只查询id和name列的信息--select语句#《==类似于awk命令

SELECT id,NAME FROM city;

### 查询参数默认值@@

#查看系统默认的autocommit的设定值，1为开启状态

ymq [oldboy]>select @@innodb\_file\_per\_table

-> ;

+-------------------------+

| @@innodb\_file\_per\_table |

+-------------------------+

| 1 |

+-------------------------+

1 row in set (0.00 sec)

ymq [oldboy]>select @@autocommit;

+--------------+

| @@autocommit |

+--------------+

| 1 |

+--------------+

1 row in set (0.00 sec)

ymq [oldboy]>

## SELECT结合where

### 1）等值查询:

##只查看CHN的数据-where语句#《==类似于grep命令

eg01：【只看中国的 城市】---adj一般作为限定条件

SELECT NAME FROM city WHERE countrycode='chn';

### 2)不等值查询

【世界上超过500w人口的城市信息】

SELECT \* FROM city WHERE population < 5000000;

### 3)不等于尽量不要使用---数据比较大，影响索引

### 4）模糊查询--国家代号为CH开头的城市

SELECT countrycode,NAME FROM city WHERE countrycode LIKE 'ch%';

PS:未解决问题（怎么理解）：尽量不要在like语句中出现'%hn' '%hn%'---因为%号在前，会影响sql语句的性能，金额索引的使用有关

## 逻辑条件and ,OR 的结合---分批查询，有利于sql的性能

### ---会进行索引的范围扫描—未解决问题（待深入）

### AND:

SELECT \* FROM city WHERE population>5000000 AND population<10000000;

### 使用union all对表进行合并

#### A表

|  |  |
| --- | --- |
| id | name |
| 101 |  |
| 102 |  |
| 103 |  |

#### B表

|  |  |
| --- | --- |
| id | hobby |
| 101 |  |
| 102 |  |
| 103 |  |

#### 如果使用union all

select \* from A where id=101 or id=103

|  |  |
| --- | --- |
| id | name |
| 101 |  |
| 102 |  |
| 103 |  |
| id | hobby |
| 101 |  |
| 103 |  |

#### 如果使用join

|  |  |  |
| --- | --- | --- |
| id | name | hobby |
| 101 |  |  |
| 102 |  |  |
| 103 |  |  |

### OR

SELECT \* FROM city WHERE countrycode='USA' OR countrycode='chn';

### 与in ,between的结合

SELECT \* FROM city WHERE countrycode IN ('usa','chn'); #<==性能同上

SELECT \* FROM city WHERE population BETWEEN 5000000 AND 10000000;

## group BY +（分组功能和 聚合函数（count(),MAX(),AVG()）sum(),MIN()）

### desc

#《==聚合函数---重复数据的分组策略-有去重的功能

#<==mysql中有很多函数，诸如时间格式、文档处理等函数

5.1 COUNT()---统计函数（内置函数）#《==mysql自带加速模块，所以处理速度比较快

### 1）【统计city表中城市的个数】

SELECT COUNT(NAME) FROM city;

#<==不要使用具体的列值==比如SELECT COUNT(shanghai) FROM city;

### 2）统计一下各个国家的城市个数

#《==使用group by做多个数组==以国家类别为数组的特性，再进行统计

SELECT countrycode,COUNT(NAME) FROM city GROUP BY countrycode;

### 3）统计中国各个省的总人数总和

### --地理|空间层次：中国---》省份---》属性之一：总人口

SHOW TABLES;

#《==先执行select \* from city where countrycode='chn';

#《==然后在进行以district为特性的数组的统计

SELECT district,SUM(population) FROM city WHERE countrycode='chn' GROUP BY district;

### 4)统计各大洲的面积

SELECT \* FROM country;

SELECT continent,SUM(surfacearea) FROM country GROUP BY continent;

### 5)统计每个国家的总人口数

SELECT \* FROM city;

SELECT countrycode,SUM(population) FROM city GROUP BY countrycode;

### 6)统计中国，人口数量最小的城市名及所在省

SELECT district,NAME,MIN(population) FROM city WHERE countrycode='chn' GROUP BY NAME ORDER BY population;

SELECT NAME,district,MIN(population) FROM city WHERE countrycode='chn';

#<==未解决问题，如何取第一行 #

末尾加limit 1即可

## order BY #<==排序

### desc

#DESC #<==从大到小

#ASC #<==默认从小到大

#字符串也可以排序，且不区分大小写

### 6.1查询中国所有的城市信息，并以人口数量从大到小排序输出

SELECT \* FROM city WHERE countrycode='chn' ORDER BY population DESC;

### 6.2统计一下各个国家的城市个数,并根据统计后的城市个数进行排序

#<== AS 可以设置别名

SELECT countrycode,COUNT(NAME) AS alias\_name

FROM city

GROUP BY countrycode

ORDER BY alias\_name DESC;

## limit

### def

#<==分页取值

#<==一般放在order by 语句之后，对输出行的数量限制

### eg01统计一下各个国家的城市个数,并取统计后的城市个数前五名

SELECT countrycode,COUNT(NAME) AS alias\_name

FROM city

GROUP BY countrycode

ORDER BY alias\_name DESC

LIMIT 5;

### eg02取区间

ymq [world]>select countrycode,sum(population) from city

-> group by countrycode

-> order by sum(population)

-> limit 2,5 desc;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'desc' at line 4

ymq [world]>select countrycode,sum(population) from city group by countrycode order by sum(population) desc limit 2,5;

+-------------+-----------------+ #limit 2,5 #<==跳过2行，取后面5行

| countrycode | sum(population) |

+-------------+-----------------+

| BRA | 85876862 |

| USA | 78625774 |

| JPN | 77965107 |

| RUS | 69150700 |

| MEX | 59752521 |

+-------------+-----------------+

5 rows in set (0.01 sec)

ymq [world]>select countrycode,sum(population) from city group by countrycode order by sum(population) desc limit 5;

+-------------+-----------------+

| countrycode | sum(population) |

+-------------+-----------------+

| CHN | 175953614 |

| IND | 123298526 |

| BRA | 85876862 |

| USA | 78625774 |

| JPN | 77965107 |

+-------------+-----------------+

5 rows in set (0.01 sec)

ymq [world]>select countrycode,sum(population) from city group by countrycode order by sum(population) desc limit 5 offset 2;

+-------------+-----------------+ #limit 5 offset 2 #<==跳过2行，取后面5行

| countrycode | sum(population) |

+-------------+-----------------+

| BRA | 85876862 |

| USA | 78625774 |

| JPN | 77965107 |

| RUS | 69150700 |

| MEX | 59752521 |

+-------------+-----------------+

5 rows in set (0.00 sec)

ymq [world]>

=============================================

## 多表集合查询

=============================================

多表连接查询

#《使用场景==所需数据，无法从单个表中获取到的时候，需要结合多表进行查询

### 1.传统连接#<WHERE

1.1查询世界人口数量少于100人的城市（国家名，城市名，人口数量）

SELECT \* FROM city;

SELECT countrycode,NAME,COUNT(population)

FROM city

WHERE population < 100;

1.1.1将上例中查询的关键词PCN全称在country表中查出

SELECT \* FROM country;

SELECT NAME FROM country WHERE CODE='pcn';

1.2将以上两步骤合并#《==多表查询的关联列查询

SELECT ci.name,co.name,ci.population

FROM city AS ci,country AS co

WHERE ci.population<100

AND ci.countrycode=co.code;

### 2.标准的表连接方式#<join on

#### ---优先于where执行

1.人口数小于100人的城市

SELECT ci.name,co.name,ci.population

FROM city AS ci

JOIN country AS co

ON ci.countrycode=co.code

WHERE ci.population<100;

2.世界上人口大于500w的城市，所在国家名及国土面积

SHOW TABLES;

SELECT \* FROM country;

SELECT \* FROM city;

SELECT country.name,country.SurfaceArea,city.population

FROM country

JOIN city

ON country.code=city.countrycode

WHERE city.population>10000000;

3.查询亚洲所有的城市信息，并统计各个省的人口总数 #adj ==限定条件==group by

SELECT co.continent,co.name,ci.district,SUM(ci.population) AS sum\_p

FROM city AS ci

JOIN country co

ON co.code=ci.countrycode

WHERE co.continent='asia'

GROUP BY ci.district

ORDER BY sum\_p DESC

LIMIT 10

#<==老师的sql语句

SELECT co.continent ,co.name,ci.district,SUM(ci.population) AS sp FROM city AS ci

JOIN country co

ON ci.countrycode=co.code

WHERE co.continent='Asia'

GROUP BY ci.district

ORDER BY sp DESC

LIMIT 10;

4.查询语句

USE oldboy;

SHOW TABLES;

DESC student;

DESC sc;

SELECT \* FROM student;

SELECT \* FROM sc;

SELECT \* FROM teacher;

SELECT \* FROM course;

5.查询oldguo老师教的学生名

SELECT t.tname,s.sname

FROM student AS s

JOIN sc

ON sc.sno=s.sno

JOIN course AS c

ON sc.cno=c.cno

JOIN teacher AS t

ON c.tno=t.tno

WHERE t.tname='oldguo';

6.查询oldguo所教课程的平均分数

SELECT t.tname,c.cname,AVG(sc.score) FROM teacher AS t

JOIN course AS c

ON c.tno=t.tno

JOIN sc

ON sc.cno=c.cno

WHERE t.tname='oldguo'

GROUP BY sc.`cno`; #<==为什么要以组的形式

7.查询oldguo所教的不及格的学生姓名

SELECT t.tname,c.cname,s.sname,sc.score FROM student AS s

JOIN sc

ON sc.sno=s.sno

JOIN course AS c

ON c.cno=sc.`cno`

JOIN teacher AS t

ON t.tno=c.tno

WHERE sc.score<60 AND t.tname='oldguo';

## select 去重查询

#对单列进行去重

select distinct id from user;

#对多列进行去重查询

select distinct id,name from user;

distinct name,id 这样的mysql 会认为要过滤掉name和id两个字段都重复的记录，如果sql这样写：select id,distinct name from user，这样mysql会报错，因为distinct必须放在要查询字段的开头。

所以一般distinct用来查询不重复记录的条数。

PS:

#使用group by也同样可以去重

select id,name from user group by name;

## SHOW

### def

实际上是将information\_schema中的查询语句封装、简化为show语句

### desc

1.实质查询的是元数据

2.以下语句可以处理公司业务中的70~80%的业务，更复杂的语句可以通过information\_schema查询

### 查询库、表

show databases； #查看mysql中已经创建的所有库

show tables； #显示当前库中的所有表

show tables from world； #显示指定库下的表

或者

SHOW TABLE STATUS FROM world like 'city'\G #查看指定库db\_name下的city表的详细信息

### 查询索引名称

show keys from table\_name; #查看表table\_name的索引名字

show index from table\_name; #查看表table\_name的索引名称

### 查询创建库表语句

show create database db\_name; #查看创建db\_name库所用的语句

show create table world.city; #查看创建库下的city表所用的语句

### 查看用户权限

show grants for root@'localhost';

### 查看列信息、字符集及校对规则等

desc city #查看city表的列信息==表结果

==show columns from city; #与上述等价

show charset； #查看所有的字符集

show collation； #查看所有支持的校对规则

### 查看存储引擎

show engines; #查看所有存储引擎

show engine innodb status\G #查看存储引擎情况

### 查看数据库状态、参数

show processlist； #查看所有的连接情况

show status ; #查看所有数据库状态情况

show variables; #查看系统所有的参数信息

show variables like 'innodb\_data\_file\_path'; #查看所有的innodb参数

ymq [(none)]>show variables like 'innodb\_data\_file\_path';

+-----------------------+------------------------+

| Variable\_name | Value |

+-----------------------+------------------------+

| innodb\_data\_file\_path | ibdata1:12M:autoextend |

+-----------------------+------------------------+

1 row in set (0.00 sec)

## 内连接与外连接查询区别

### inner join、left join 和right join

1）INNER JOIN（内连接,或等值连接）：获取两个表中字段匹配关系的记录。



2）LEFT JOIN（左连接）：获取左表所有记录，即使右表没有对应匹配的记录。



3）RIGHT JOIN（右连接）： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。



### 内连接查询

ymq [school]>select t.tname,c.cname

-> from teacher as t

-> join course as c

-> on t.tno=c.tno

-> where t.tname='oldguo';

+--------+-------+

| tname | cname |

+--------+-------+

| oldguo | mysql |

+--------+-------+

1 row in set (0.01 sec)

### 外连接查询—left

ymq [school]>select t.tname,c.cname from teacher as t left join course as c on t.tno=c.tno and t.tname='oldguo';

+--------+-------+

| tname | cname |

+--------+-------+

| oldguo | mysql |

| oldboy | NULL |

| hesw | NULL |

+--------+-------+

3 rows in set (0.00 sec)

### 外连接查询—right

ymq [school]>select t.name,c.name from teacher as t right join course as c on t.tno=c.tno and t.tname='oldguo';

ERROR 1054 (42S22): Unknown column 't.name' in 'field list'

ymq [school]>select t.tname,c.cname from teacher as t right join course as c on t.tno=c.tno and t.tname='oldguo';

+--------+--------+

| tname | cname |

+--------+--------+

| oldguo | mysql |

| NULL | linux |

| NULL | python |

+--------+--------+

3 rows in set (0.00 sec)

# 子查询

## desc

先运行子查询，将结果传递给父查询，再继续运行SQL语句

尽量不要使用子查询

## eg01--select语句

select 列名字

from 表

join 表

on 列[属性]=列[属性]

where 列条件

group by 列名

order by 列名

limit 范围

## eg02选出人口数小于100人的国家-- where子句

### 性能

---》多表连接查询比子查询语句性能高

### 子查询

SELECT CODE,NAME, population FROM country WHERE population <100;

#以子查询的方式

SELECT NAME,CODE,population FROM country WHERE CODE IN (SELECT CODE FROM country WHERE population <100);

### 多表连接查询

select co.name,ci.population

from country as co

join city as ci

on co.code=ci.countrycode

where ci.population<100;

# 练习

8.1将student表添加8个新学员

8.2将teacher表添加2位老师

8.3.将course表添加6门课程

8.4每位老师保证教不同的2门以上课程

8.5保证每位老师的每门课程都有至少一人学习

【查询命令】

SELECT \* FROM student;

SELECT \* FROM sc;

SELECT \* FROM teacher;

SELECT \* FROM course;

8.1将student表添加8个新学员

INSERT INTO student(sname,sage,ssex)

VALUES

('xiaoming',12,'m'),

('wang5',16,'m'),

('shiyun',18,'f'),

('xuner',16,'f'),

('mazi',19,'m'),

('qiubai',16,'f'),

('feixue',19,'f'),

('little',16,'m');

8.2将teacher表添加2位老师

INSERT INTO teacher(tno,tname)

VALUES

(104,'alex'),

(105,'egon');

#<==修改104的值

UPDATE teacher SET tname='alex' WHERE tno=104;

SELECT tno,tname FROM teacher;

8.3.将course表添加6门课程

INSERT INTO course(cno,cname,tno)

VALUES

(1004,'java',104),

(1005,'go',105)

8.4每位老师保证教不同的2门以上课程

INSERT INTO course(cno,cname,tno)

VALUES

(1001,'linux',102),

(1002,'python',104),

(1003,'mysql',105),

(1004,'java',103),

(1005,'go',101);

#出现如下报错

#Error Code: 1062

#Duplicate entry '1001' for key 'PRIMARY'

【解决办法】

ALTER TABLE course DROP cno;

ALTER TABLE course ADD cno INT NOT NULL COMMENT '课程编号' FIRST;

UPDATE course SET cno=1001 WHERE cname='linux';

UPDATE course SET cno=1002 WHERE cname='python';

UPDATE course SET cno=1003 WHERE cname='mysql';

UPDATE course SET cno=1004 WHERE cname='java';

UPDATE course SET cno=1005 WHERE cname='go';

【表的排序优化】

1.删除表

DROP TABLE course;

2.建立新表

CREATE TABLE course(

cno INT NOT NULL COMMENT '课程编号',

cname VARCHAR(20) NOT NULL COMMENT '课程名称',

tno INT NOT NULL COMMENT '教师编号'

)ENGINE=INNODB CHARSET utf8;

3.插入新的数据

INSERT INTO course(cno,cname,tno)

VALUES

(1001,'linux',101),

(1002,'linux',102),

(1003,'python',102),

(1004,'python',104),

(1005,'mysql',103),

(1006,'mysql',105),

(1007,'java',103),

(1008,'java',104),

(1009,'go',101),

(1010,'go',105);

8.5保证每位老师的每门课程都有至少一人学习

1.删除表

DROP TABLE sc

2.新建表

CREATE TABLE sc(

sno INT NOT NULL COMMENT '学号',

cno INT NOT NULL COMMENT '课程编号',

score INT NOT NULL DEFAULT '0' COMMENT '分数'

)ENGINE=INNODB CHARSET utf8;

3.插入数据

INSERT INTO sc(sno,cno,score)

VALUES

(1,1001,80),

(1,1002,59),

(2,1002,90),

(2,1003,100),

(3,1001,99),

(3,1003,40),

(4,1004,80),

(4,1005,59),

(5,1006,90),

(6,1007,100),

(7,1008,99),

(8,1009,40),

(9,1010,80),

(10,1002,59),

(11,1008,90),

(8,1009,100),

(9,1007,99),

(7,1006,40);

UPDATE sc SET cno=1010 WHERE sno=9 AND score=80;

【常用查询】

DESC course;

SHOW CREATE TABLE sc;

SELECT \* FROM sc;

表的优化

