MySQL 学习

SQL 的执行顺序：

- **第一步：执行 FROM**
- **第二步：WHERE 条件过滤**
- **第三步：GROUP BY 分组**
- **第四步：执行 SELECT 投影列，聚集函数**
- **第五步：HAVING 条件过滤**
- **第六步：执行 ORDER BY 排序**

概述

- 数据库：按照数据结构来组织、存储、管理数据的仓库。
- 关系型数据库，将数据放在不同的表中，而不是大仓库，提高了灵活度
- 支持大型数据库，中小型企业
- 支持多种语言，多种系统
- 开源
- 免费
- 多系统多语言：PHP、C、Python

cmd

将 mysql.exe 地址添加到环境变量 path 路径

使用 mysql -u root -p

输入密码

就可以使用 cmd 了

RDBMS

RDBMS:

Relational Database Management System 关系型数据库管理系统

特点:

1. 数据以表格形式存储
2. 每行为各种记录名称
3. 每列为记录名称所对应的数据域
4. 许多行、列组成一张表单
5. 若干表单组成 database

术语

1. 数据库：表的集合
2. 数据表：数据的矩阵
3. 列：包含相同数据格式如邮箱的数据
4. 行：一组相关的数据
5. 冗余：存储两倍数据，降低了性能、提高了安全性
6. 主键：唯一的，一个表只含有一个主键，可以使用主键来查询数据

7. 外键：用于关联两个表

8. 复合键：将多个列作为一个索引键，用于复合索引

9. 索引：相当于下标

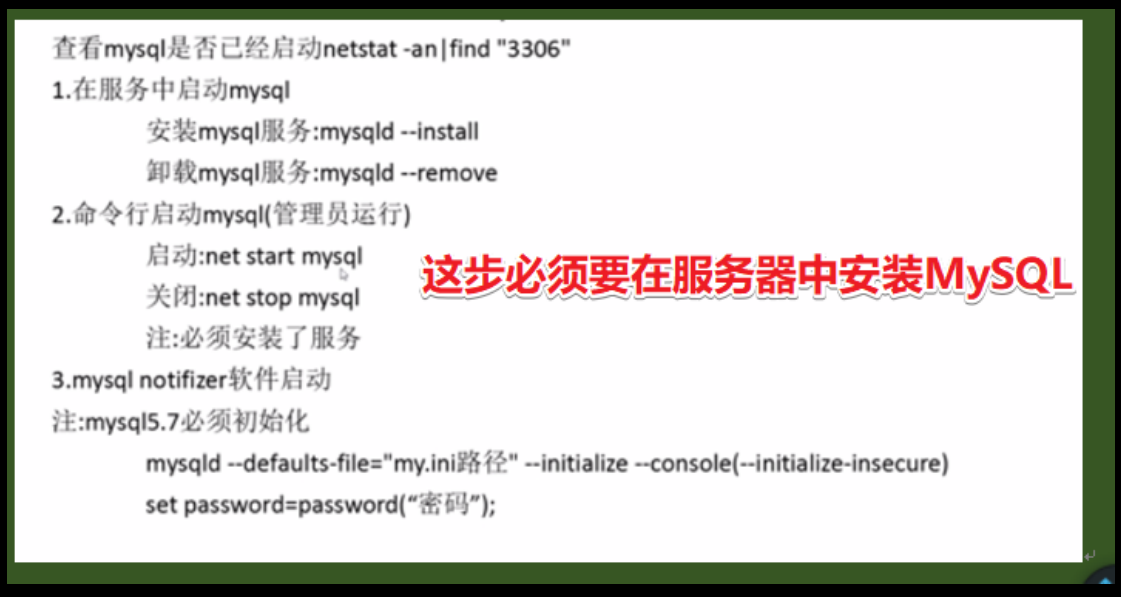
10. 参照完整性：不允许引用不存在的实体。

表格术语

1. 表头 header：每一列的名称

2. 列 col、行 row、值 value、键 key

管理



注意：命令以分号结尾，可以多行

- Show database:—列出所有数据库
- Use 数据库名—选择数据库
- Show tables—列出当前数据库中所有的表
- Show columns from 数据表名—列出表中所有的列

- Show index from 数据表名——列出表中的索引
- Show table status from 数据库名——数据库中的统计信息

启动服务器

若 ERROR 2003 (HY000): Can't connect to MySQL server on 'localhost' (10061)，则是服务器没有启动。

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqld -install

Net start mysql(或net start mysql80 或在计算机管理中启动)（管理员模式）

若再出现: Install/Remove of the Service Denied!

则使用 cmd 的时候必须用管理员登录

进入管理员 cmd: win+R cmd Ctrl+shift+Enter

C:\WINDOWS\system32>

Help / \h :获取帮助

连接与断开服务器

为了连接服务器，调用 MySQL 时通常需要提供一个 MySQL 用户名、密码，如果服务器运行在登录服务器之外的其他机器上，还需要指定主机名。

---

mysql -h host -u user -p

mysql -u root -p

输入查询

Select version(), current_date

Select now() ——当前时间: 2019-03-24 12:40:31

\c ——正在输入一个语句的过程中取消。

说明:

1. 一个 SQL 命令以分号结尾（个别不需要）
2. 使用表格和列的方式显示数据
3. 会显示查询用时（时钟时间，不准确）
4. 对大小写不敏感
5. 可以使用算术表达式: PI()*4, (1+2)*3/5
6. 一行可以输入多条语句，但是需要使用;隔开（每条语句都需要完整），会分别显示。
7. 多行可以输入一条语句
8. 提示符:

mysql>	准备好接受新命令。
->	等待多行命令的下一行。
'>	等待下一行，等待以单引号(“'”)开始的字符串的结束。
">	等待下一行，等待以双引号(“"”)开始的字符串的结束。
`>	等待下一行，等待以反斜点(“`”)开始的识别符的结束。
/*>	等待下一行，等待以/*开始的注释的结束。

9. 跨行可以输入多个字符串

10.

创建、使用数据库

- Create database DBName——创建自己的数据库
- Use dbname(也可以在启动时 use: mysql -h host -u user -p dbname)
- 创建表：create table tableName (para1 type, para2 type……)

```
create table pet (name varchar(20),species varchar(20),gender char(1),birth date,death date);
```

  - Varchar(n): 表示变长的字符串，但是不超过 n
  - Char（n）：表示定长的字符串

  使用 default 来默认赋值

- Describe tableName: 显示表的名称、类型

将数据装入表

方式 1: 文件导入

```
load data infile "pet.txt" into table pet
 fields terminated by ":"
 lines terminated by "\r\n"   (我这里一直没有成功)
```

方式 2: 逐条增加

```
Insert into tableName vlues('','','',……)
```

索引

索引可以大大提高 MySQL 的检索速度

实际上，索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录。

过多的使用索引将会造成滥用。因此索引也会有它的缺点：虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行 INSERT、UPDATE 和 DELETE。因为更新表时，MySQL 不仅要保存数据，还要保存一下索引文件。

1. 普通索引

   创建：

   CREATE INDEX indexName ON mytable(username(length));

   如果是 CHAR，VARCHAR 类型，length 可以小于字段实际长度；

   如果是 BLOB 和 TEXT 类型，必须指定 length。

   修改表结构(添加索引)：

   ALTER table tableName ADD INDEX indexName(columnName)

   创建表的时候直接指定：

```
CREATE TABLE mytable(
ID INT NOT NULL,
username VARCHAR(16) NOT NULL,
INDEX [indexName] (username(length))
```

```
);
```

删除：

```
DROP INDEX [indexName] ON mytable;
```

2. 唯一索引

索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

创建索引

```
CREATE UNIQUE INDEX indexName ON mytable(username(length))
```

修改表结构

```
ALTER table mytable ADD UNIQUE [indexName] (username(length))
```

创建表的时候直接指定：

```
CREATE TABLE mytable(
ID INT NOT NULL,
username VARCHAR(16) NOT NULL,
UNIQUE [indexName] (username(length))
);
```

检索信息

SELECT 语句

Select what_to_select from which_table where condition

Eg:

*Select * from pet ——全选*

*select * from pet where gender="f"; ——性别为女的所有*

select name="HelloKitty" from pet;——使用名字限定

注意：

- 对字符串的大小写不敏感。
- 判定相等使用=，而不是==
- 可以使用逻辑操作符：and，or

```
select * from pet where gender="f" or name="HelloKitty";
```

and 优先级要高于 or，所以混用时最好使用括号限定。

- 支持 is，not 等判断：IS NULL，IS NOT NULL
- 0 或 null 意味着 false，二其他意味着 true

选择特殊列

```
Select para1,para2… from table_name

Eg:  select name,gender,birth from pet;
```

选择的某列可能有许多的值是重复的，可以使用 distinct 唯一检索

```
Select distinct paraName from tableName

Eg: select distinct gender from pet;
```

分类行

排序：order by

```
Select para… from order by aPara

Eg: select name,birth,death from pet order by birth;
```

强制按大小写： binary para

```
select name,birth,death from pet order by binary name;
```

默认是升序，如果要使用降序——列名+desc

```
select name,birth,death from pet order by binary name desc;
```

分组：group by：

```
SELECT owner, COUNT(*) FROM pet GROUP BY owner;
```

Group by 必须结合**聚合函数**使用：

常用聚合函数

- count() *计数*
- sum() *求和*
- avg() *平均数*
- max() *最大值*
- min() *最小值*

按字段分组，然后可以使用函数计算所需值

	student_id	student_name	student_sex	student_age	student_class
☐	1	张三	男	25	T01
☐	2	李四	女	20	T01
☐	3	王五	男	22	T03
☐	4	赵六	男	18	T02
☐	5	孙七	女	21	T02
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

```
SELECT student_class,COUNT(ALL student_name) AS 总人数 FROM t_student GROUP BY (student_class);
```

Having

对 group 进一步筛选：having

聚合之后执行过滤条件

例，我们想查询平均年龄在 20 岁以上的班级

使用

```
SELECT student_class, AVG(student_age) FROM t_student WHERE AVG(student_age)>20 GROUP BY student_class;
```

会出错，因为聚合函数在 WHERE 之后执行，所以这里在 WHERE 判断条件里加入聚合函数是做不到的。

使用 having：

```
SELECT student_class, AVG(student_age) AS 平均年龄 FROM t_student GROUP BY (student_class) HAVING AVG(student_age)>20;
```

即 having 之后可以使用聚合函数作为筛选条件

```
select species,count(*)  from pet group by species having count(*)>=2
```

日期

- Curdate()——当前日期

- Year(date)——获取一个时间的年，eg: year(“2202-02-23”)

```
select name,birth,death,(year(curdate())-year(birth)) as age
from pet;
```

- Month(): 截取月份
- Dayofmonth(): 一月的第几天
- Day_add(date_1, date_2)

```
select date_add(curdate(),interval 3 month);
```

- Mod(number, base) : 取模

模式匹配

SQL 的模式匹配是忽略大小写的

- _ : 任意单个字符
- %: 任意数目字符
- 使用 like, not like 判断匹配

```
select * from pet where name like 'b%'; 以b开头

select * from pet where name like '%hh%'; 包含hh

select * from pet where name like '____'; 长度为4的
```

扩展正则表达式

使用 REGEXP、NOT REGEXP 表示匹配

- . : 单个字符
- […]: 其中任意字符: [0-9]
- * : 0 个或多个: .*
- ^x : 以 x 开始
- x$: 以 x 结尾

- 使用 binary 可以强制区分大小写:

```
select * from pet where name regexp binary '^p';
```

计数行

Count()

```
select count(*) from pet ; 计算总数

select gender, count(*) from pet group by gender;
```

可以使用点运算符: 在多个表中时, 若有相同的列名, 则需要使用.

```
select pet.name from pet;

select std.name, pet.sex from std,pet;
```

内置函数

- Max()

```
select name,max(age) as max from std;
```

- Min()
-

数据类型

- enum('T','F')
- char()
- varchar()
- float()
-

exists

drop table if exists tableName

source 与.sql

可以建立一个.sql 文件，把 mysql 语句写入其中

然后在命令行 mysql 中输入

Source sql 文件全地址。

这样就可以运行 SQL 文件中的所有语句了。

自增参数

使用 auto_increment 表示自增，注意：自增的列必须是主键，

在创建表时，在全部声明列后，使用：

```
Primary key (‘autoIncreColName as ID’)
```

SQL 语法再学习

```
create table course (courseId int not null auto_increment ,courseName varchar(45) not null,stuNum int not null ,fullScore int notnull default 100,startTime timestamp not null default  current_timestamp, primary key(courseId))
```

id 如果没有添加自增，需要添加：先移除，重新添加。

修改主键，需要先删除主键在添加

```
ALTER TABLE tt DROP PRIMARY KEY;

ALTER TABLE tt CHANGE COLUMN id id int(10) NOT NULL AUTO_INCREMENT PRIMARY KEY;
```

插入数据时，自增主键处用 0 或 null 代替

也可以指定插入的列，排除该自增主键

对于默认值也是，指定非默认值，然后给非默认值赋值：

```
insert into course(courseName,stuNum)values('SE5',220)//其他是主键或默认值
```

MYSQL 时间戳：

timestamp

当前时间戳:CURRENT_TIMESTAMP

使用<,>,=可以比较时间前后

```
select `courseName`,count(`courseName`) from course group by `courseNam`
```

limit 语句:

select name from country limit 100

(其他 DB 不一样，如 SQL server 是 top 语句)

like: _:单个字符

%:多个任意字符

[charlist] 字符列表中任意字符

[^charlist] or [!charlist]: 非列表中任意字符

In:

in 允许在 where 子句中规定多个值

select name from country where Name in ('China','Peru'

Between ••• and •••:

介于两个值之间的数据

select name from country where `Population` between 1 and 10000

不在之间，可以加not: not between - and -

alial别名:

select as Name (Name 可以用来显示输出的名字)

join:

用于根据表之间的联系来查询数据

```
select * from table1
XX_join table2
on table1.col = table2 = col
```

连接方式:

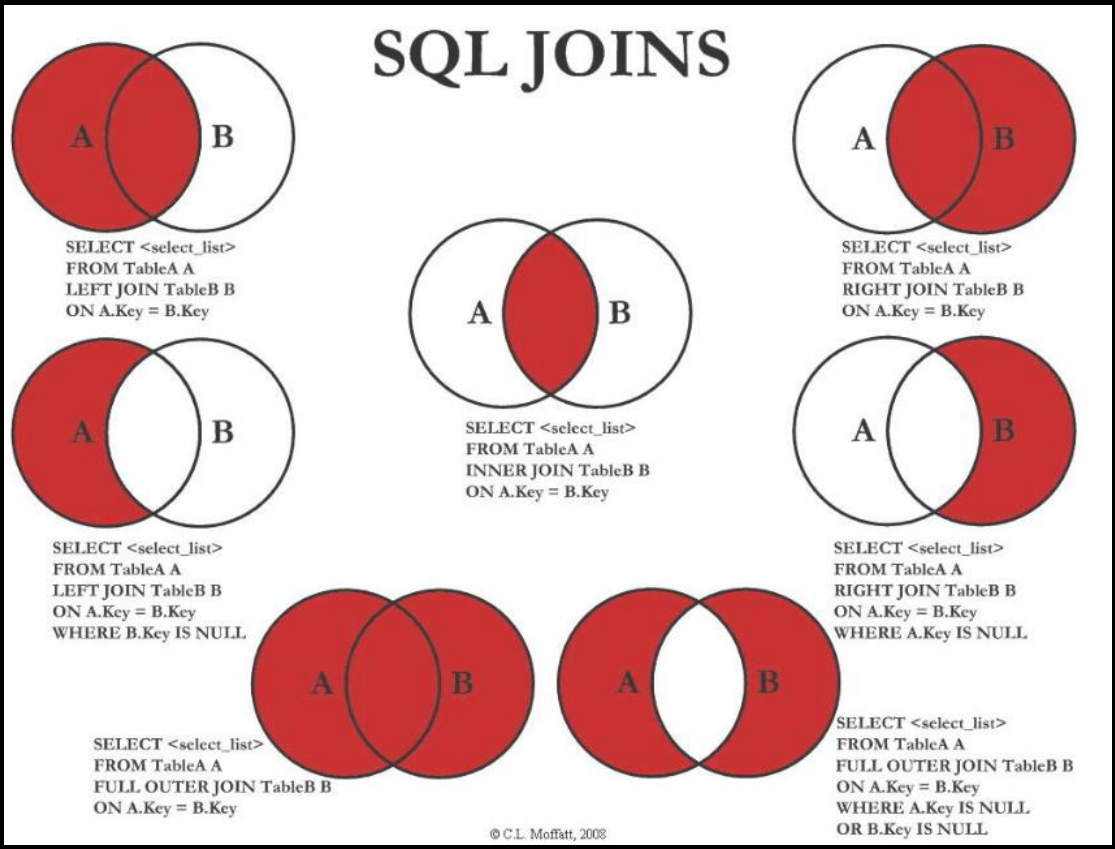
JOIN:如果表中至少有一个匹配返回行

INNER JOIN:

LEFT_JOIN:即使右表中没有匹配，也要返回坐表中所有行

RIGHT_JOIN:即使左表中没有匹配，也要返回右表中所有行

FULL-JOIN:只要其中一个表中存在匹配，就返回行。



改表名:

```
alter table po_oeder rename  to po_order
```

Timestamp

```
+---------------+----------------+
| 列类型        | 显示格式       |
| TIMESTAMP(14) | YYYYMMDDHHMMSS |
| TIMESTAMP(12) | YYMMDDHHMMSS   |
| TIMESTAMP(10) | YYMMDDHHMM     |
```

```
| TIMESTAMP(8)   | YYYYMMDD       |
| TIMESTAMP(6)   | YYMMDD         |
| TIMESTAMP(4)   | YYMM           |
| TIMESTAMP(2)   | YY             |
+----------------+----------------+
```

Load csv

load data infile 'D:\\links.csv' into table movies fields terminated by ',' optionally enclosed by '"' escaped by '"' lines terminated by '\r\n';

mysql vs. mongodv

https://www.guru99.com/mongodb-vs-mysql.html

Item\DB	MySQL	MongoDB
类型	关系型	非关系型 document-oriented NoSQL
编写语言	C & C++	C++
诞生日期	1997	2007
数据形式	表 行 列	Collection JSON Document Field
存储方式	不同的引擎上有不同的存储方式	虚拟内存+持久化(JSON 格式)
查询方式	通用的 sql 语句	独特的的查询方式(JavaScript)
存储占用	节省空间	耗费空间
灵活性	较差。需要严格定义每一个 schema	极好，限制宽松，不用定义 schema
分片	mysql 5.1 以后支持分区。按照一定的规则，将一个数据库表分解成很多细小的表，存储在不同的位置	如果负载的增加，它可以分布在计算机网络中的其他节点上
MapReduce	不支持	支持

replication	master-slave replication and master replication.	built-in replication, and auto-elections.
适合数据形式	结构化	(非)结构化
安全性	很好	稍差
安全性问题	SQL 注入攻击	多方面安全隐患
Join	支持	不支持
大量数据下效率	高	低
事务	支持	不支持/实现复杂

适用场景：

Mysql：

1. 结构化数据，传统关系型；
2. ‘高价值数据’
3. 采用事务，重视安全性；
4. 如 web 网站系统、日志记录系统、数据仓库系统、嵌入式系统等

MongoDB：

1. 结构化、非结构化数据，如基于位置的数据查询
2. ‘低价值数据’，如内容型数据
3. 高性能：高写入负载，快速读取、更新，可以分片
4. 高可用性：快速响应的处理单节点故障，自动、安全的完成故障转移
5. 数据量很大或者未来会变得很大
6.

错误整理

No query specified——sql 不合法

时区错误

```
show variables like '%time_zone%';

set global time_zone='+8:00';
```