# 通用介绍

1.数据库

数据库英文名称为DataBase，简称DB，是按照一定格式存储数据的一些文件的组合。顾名思义：存储数据的仓库，实际上就是一堆文件，这些文件中存储了具有特定格式的数据。

2.数据库管理系统

英文名称为DataBaseManagement，简称DBMS。是专门用来管理数据库中数据的一类系统，它可以对数据库中的数据进行增删改查。常见的数据库管理系统有Mysql、Oracle、MS SqlServer、DB2、sybase等。

3.SQL

Structured Query Language，译为结构化查询语言。程序员通过编写SQL语句，然后由DBMS负责执行SQL语句，最终完成对数据库中数据的增删改查。SQL是一套标准规范的语句，在Mysql、Oracle、DB2等DBMS中都可以使用。

4.Windows系统中启用和停止服务的命令格式

net start 服务名称

net stop 服务名称

# 关系型数据库通用知识

SQL(Structured Query Language)结构化查询语言

1.所有的SQL语句都以;结束，SQL语句默认在windows环境下不区分大小写，在linux环境

下区分大小写。

2.起别名

起别名可以使用AS关键字

select dept AS de from user;

-- AS可以省略

select dept de from user;

-- 如果起的别名中间有空格，可以用单引号或双引号将别名扩起来：

select dept ‘de oh’ from user;

select dept “de oh” from user;

-- 尽量用单引号，单引号是标准，双引号在oracle中无法使用。

-- 如果别名是中文，要用单引号将别名括起来。

3.使用sql语句时，可以使用数学表达式

select money\*12 AS qian from user;

4.查询表里所有的记录

select \* from 表名;

特殊：判断字段是否为null，只能使用is，而不能用=。因为null不是一个值，它表示

什么也没有，只能用是不是来判断。

select \* from student where name is null;

select \* from student where name is not null;

注意：尽量避免判断null值，因为引擎会进行全表扫描而不会使用索引。

null值的排序问题。

select \* from student order by age (默认升序asc) nulls first/last;

说明：nulls first 表示 age为null的放在最上面，nulls last(默认)则相反。

select \* from 类似的语句效率低，\*会被转化为该表所有的字段名，影响效率，且这样

写可读性差。

5.插入单条、多条记录

单条：

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

其中列名和值要一一对应。

insert into 表名values (值1,值2,...);

如果不加列名，则默认值会按从上往下的顺序给表里所有的列对应赋值。

多条：

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

insert into 表名values (值1,值2,...), (值1,值2,...),…;

补充：将查询的结果插入到另一个表中的两种方式：

insert into student\_new select \* from student;

insert into student\_new(id,name) select id,name from student;

6.修改记录

update 表名 set 列名1 = 值1, 列名2 = 值2 ... where 条件;

另一种方式：将一个表中字段的值更新到另一个表中字段的值里，例：

mysql：update tableA a left join tableB b on a.id = b.id set a.name = b.name;

oracle：update tableA a set a.name = (select b.name from tableB b where a.id = b.id);

7.删除记录

(1)delete from 表名 where 条件;

警告：如果不加where条件，则会删除表里所有的记录。

事务sql：start transaction;...(增删改)sql语句;rollback;

(2)TRUNCATE TABLE 表名

补充：删除后无法回滚；一页一页的删除；在事务日志中只记录数据页的删除；效率高



9.插入字段

ALTER TABLE 表名 ADD [COLUMN] 字段名 字段类型 [NOT NULL][DEFAULT ''];

oracle不允许带COLUMN关键字

10.SQL语言包含的4个部分

DDL:DROP、CREATE、ALTER、TRUNCATE等 --操作表结构

DML:INSERT、UPDATE、DELETE等 --操作记录

DQL:SELECT --查询记录

DCL:GRANT、REVOKE等 --授权和撤销权限

TCL: COMMIT、ROLLBACK --事务的提交与回滚

11.创建数据库

create database 库名称;

3.创建表

create table 表名(

列名 列类型 [primary key] [auto\_increment],--[主键] [自增]

列名 列类型 [not null],--[不为null]

列名 列类型 [unique],--[不重复，唯一]

列名 列类型 [default 0.0] [comment ‘字段的含义’],--[默认值] [字段注释]

...

列名 列类型,

[unique(列名1,列名2...)] --多个列联合起来唯一

--单独创建主键

constraint 主键名称 primary key(字段名)

--创建外键

constraint 外键名称 foreign key(字段名) references 表名(字段名)

);

补充：mysql中如果一个字段同时被not null和unique修饰，该字段自动变成主键字段

(oracle不会这样)。

主键：特征是not null + unique；主键只有一个，但可以设置多个字段为联合主键。

语法：constraint 主键名称 primary key(字段1,字段2...)。

主键适合用数字，而不是varchar。联合主键的每个字段都不能是null。

外键：假如在创建表的时候，执行了以下设置外键的语句

constraint fk1 foreign key(cname) references tbl\_user(name)

意为：cname作为外键，引用tbl\_user表中的name字段的值，且cname的值只能

是name字段中的值。当然外键的值可以是null。外键参照的字段不一定得是主键，

但必须得有唯一性。

补充：阿里开发手册明确不允许使用外键。原因：级联操作会增加数据库的压力；高并

发时容易死锁；有外键时，开发和维护都不方便。

12.快速创建表

--相当于表emp2是emp的复制版本

create table emp2 as select \* from emp;

--也可以将一个查询结果复制为一张表

create table emp3 as select empno,ename from emp where job = ‘MANAGER’;

12.删除数据库

drop database 库名称;

12.删除表

drop table 表名; --当表不存在时会报错,可以执行以下语句，删除表前先判断是否存在表

drop table if exists 表名;

13.修改库名称

ALTER DATABASE 原名称 RENAME TO 新名称;

13.修改表名称

alter table 表名 rename to 新表名;

14.修改列名称

alter table 表名 rename 列名 to 新列名;

15.修改列的数据类型

alter table 表名 modify 列名 数据类型;

16.删除列

alter table 表名drop column 列名;

16.增加一个列

alter table 表名 add column 列名 数据类型;

注意：oracle中是不加column关键字的

17.条件判断



between ... and ...两边都是闭区间，包含两边的值。且在使用时，必须小的值在and的左边，

大的值在and的右边，否则查不出来值。

and的优先级高于or。

如果要查询数据中带%或者\_的，需要使用\。比如：

select name from user where age like %\\_%; --查询age带\_的name

18.排序

order by 字段名 --默认升序asc

order by 字段名 desc --降序

当按多个字段排序时，用逗号隔开。例如：

order by age asc,name desc

了解：可以按照第几列排序，但不建议用，例如：

select name,sal from user order by 2; --按照第二列，也就是sal升序排。

19.查询表中某列是否有重复的数据

比如：查询表tbl\_op\_dbindex中的列organization\_id是否有重复的数据

SELECT count(distinct(organization\_id)),count(organization\_id) FROM "tbl\_op\_dbindex"

右边比左边结果大就是有重复的数据。

查询重复数据的行：

select \* from tbl\_op\_dbindex a join tbl\_op\_dbindex b on a.id != b.id and a.organization\_id = b.organization\_id;

# Mysql数据库

1.命令行登录mysql的两种方式





2.展示所有数据库

show databases;

3.进入数据库

use 库名;

4.展示进入的库的所有的表

show tables;

5.直接展示某库所有的表

show tables from 库名;

6.显示你在哪个库里

select database();

7.查看表结构

--desc是describe的缩写

desc 表名; 或者describe 表名;

8.退出命令行的数据库状态

exit

9.命令行在退出数据库状态下，可以使用

mysql --version 或者 mysql -V来显示mysql的版本。

10.mysql的注释

单行注释：# 或者 --

多行注释：/\* \*/

11.分页查询

select \* from table limit (start-1)\*limit,limit;

其中start是页码，limit是每页显示的条数。

12.mysql默认端口号是3306。

13.命令行关闭/启动Mysql服务

语法：net stop/start 服务名。如：

net stop Mysql --关闭Mysql服务

14.命令行执行sql文件：

在用命令行登录mysql之后，执行 source sql文件地址即可。例：



如果sql文件没有创建数据库而是创建表，需要使用use 库名 进入到库中再执行文件。

15.查看mysql的版本号

select version();

## 1.单行处理函数

1.lower 转换小写

select lower(name) name from user;

2.upper 转换大写

select upper(name) name from user;

3.substr 取子串

substr(被截取的字符串,起始下标,截取的长度) --起始下标从1开始

select substr(name,1,2) name from user;

4.concat 拼接字符串

select concat(name,age) nge from user;

5.length 值的长度

select length(name) length from user;

6.trim 去除最左边和最右边的空格

select \* from user where name = trim(‘ King ’);

7.round 四舍五入

round(被四舍五入的数字,保留几位小数)

select round(123.456,2) result from user; --保留两位小数，结果是123.46

select round(123.456,-1) result from user; --结果是120，原因是对十位进行了舍入

8.rand(种子数) 生成随机数，值是0-1之间的小数，包括0但不包括1。

select rand() a,rand() b from user; --a列和b列的值不同

select rand(1) a,rand(1) b from user; --a列和b列的值相同，原因是给定了相同的种子数1

select round(rand()\*100,0) from user; --生成0-100之间的数，包括0但不包括100

9.ifnull 空处理

提示：与null进行加减乘除的结果都是null。

ifnull(字段1,字段2)：当字段1为null时，返回字段2的值。

10.format 数字格式化，返回一个字符串

语法：format(n,d,locale)：n是要格式化的数字。D是要舍入的小数位数。

locale是一个可选参数，用于确定千个分隔符和分隔符之前的分组。默认是en\_US。

select format(14500.2018,2); --结果14,500.20

11.str\_to\_date 将字符串varchar类型转换成date或datetime类型

语法：str\_to\_date(‘字符串日期’,’日期格式’)



示例sql：

--birth是date类型的字段

insert into tbl\_user(id,birth) values(‘222’,str\_to\_date(’10-12-2020’,’%d-%m-%Y’));

--如果你的字符串日期能匹配%Y-%m-%d的话，就可以不用str\_to\_date函数了，会自动转换

insert into tbl\_user(id,birth) values(‘222’,’2020-12-10’);

--date类型的默认格式是%Y-%m-%d，而datetime是%Y-%m-%d %h:%i:%s，datetime默认格式的用法和date的默认格式用法一致。

12.date\_format 将date或datetime类型转换成具有一定格式的varchar字符串类型

语法：date\_format(日期类型的数据,’日期格式’)

示例sql：

--birth是date类型的字段

select id,date\_format(birth,’%m/%d/%Y’) as birth from tbl\_user;

--以下的sql自动进行了date类型的转换，成了varcha类型，并且采用了mysql的默认日期格式，’%Y-%m-%d’

select id, birth from tbl\_user;

--date类型的默认格式是%Y-%m-%d，而datetime是%Y-%m-%d %h:%i:%s，datetime默认格式的用法和date的默认格式用法一致。

补充：java中的日期格式：yyyy-MM-dd HH:mm:ss SSS

13.now 返回系统当前时间，带有时分秒信息

语法：now()。类型为datetime。

## 2.分组函数(多行处理函数)

分组函数在使用的时候必须先进行分组，然后才能使用。如果没有分组，默认整张表为一组。

count 计数

sum 求和

avg 平均值

max 最大值

min 最小值

注意：1.分组函数会忽略null，不会算上null。avg在除的时候也不会算上null的个数。

2.count(具体字段)和count(\*)的区别：

(1) count(具体字段)：统计分组中该字段下所有不为null的元素的总数。

(2) count(\*)：统计分组中的总行数。(在分组的所有字段的记录中，只要有一个元

素值不为null，该条记录便会让count++。如果一条记录中所有的元素值都是null，

那么这条记录便不会被count++，当然，这样的记录便不再算是记录，它不存在。)

3.分组函数不能直接使用在where子句中。

group by 具体字段：根据具体字段的值来进行分组。

原因：语句的执行顺序：from->where->group by->having->select->order by->limit

在where执行的时候，还没有group by，即还没有进行分组。

select job,sum(sal) from emp group by job; -- 根据job的不同分别求其工资总和。

4.在一条select语句中，如果有group by的话，select后面只能跟：参加分组的字段

以及分组函数，其他的一律不能跟。比如：

select name,job,sum(sal) from emp group by job; --虽然mysql不报错，但没有意义。

其他的数据库管理系统用这条语句会报错。

5.group by 后可以跟多个字段，比如：按照不同部门的不同工作分组。例：

select dept,job,max(sal) from emp group by dept,job;

6.having：可以对分完组之后的数据进一步的过滤。

having不能单独使用，不能代替where，必须和group by联合使用。

例如：查询每个部门最高薪资，要求只显示最高薪资大于3000的。

select deptno,max(sal) from emp group by deptno having max(val) > 3000;

-- having的效率不如where，所以这样改，效率更高：

select deptno,max(sal) from emp where sal > 3000 group by deptno;

-- 只有where无法实现的才用having。比如：

select deptno,avg(sal) from emp group by deptno having avg(sal) > 2500;

## 3.distinct 去除重复值

distinct只能出现在所有字段的前方。例如：

select distinct job from emp; --对工作相同的记录进行去重

select distinct job,deptno from emp; --对工作和部门编号同时一致的记录进行去重。

select count(distinct job) from emp; --统计工作岗位的数量。

## 4.连接查询

多张表联合起来查询被称为连接查询。

1.分类

根据语法的年代分类：

SQL92：1992年出现的语法。

(主用)SQL99：1999年出现的语法。

根据表连接的方式分类：

内连接：

等值连接

非等值连接

自连接

外连接：

左外连接(左连接)

右外连接(右连接)

全连接(基本不用)

2.当两张表进行连接查询且没有任何条件限制的时候，最终查询的结果条数是两张表条数的乘积，这种现象被称为：笛卡尔积现象，这是一种数学现象。例如：

select ename,dname from emp,dept;

emp表中有14条记录，dept表有4条记录，则查询结果为14\*4=56条记录。

3.如何避免笛卡尔积现象，可以加限制条件。

select ename,dname from emp,dept where emp.deptno = dept.deptno;

这样做，虽然避免了笛卡尔积现象，但是匹配的次数没有减少，效率低。

可以看出，一条语句中，连接的表越多，效率越低，应避免多个表的连接。

在多表连接查询时，最好给表加上别名，因为可以提高SQL语句的执行效率。例如：

select e.ename,d.dname from emp e,dept d where e.deptno = d.deptno;

原因是查询字段ename和dname时不用再从emp和dept表中找这俩字段属于哪个表。

4.内连接之等值连接

SQL92语法：

select e.ename,d.dname from emp e,dept d where e.deptno = d.deptno;

SQL99语法(语法更清晰，将本属于表连接的筛选条件交由on来执行而不是where)：

--inner可以忽略。

select e.ename,d.dname from emp e inner join dept d on e.deptno = d.deptno;

5.内连接之非等值连接

例如：查询每个员工的薪资等级：

--条件不是一个等量关系

select e.ename,e.sal,s.grade from emp e join salgrade s on e.sal between s.losal and s.hisal;

6.内连接之自连接(也就是表自己连接自己查询)

例如：查询员工的上级领导，要求显示员工名和对应的领导名

select a.ename as ‘员工名’,b.ename as ‘领导名’ from emp a join emp b on a.mgr = b.empno;

7.外连接

--左外连接，关键字 left outer join，outer可以省略

select e.ename,d.dname from dept d left outer join emp e on e.deptno = d.deptno;

--右外连接，关键字 right outer join，outer可以省略

select e.ename,d.dname from dept d right outer join emp e on e.deptno = d.deptno;

区别：内连接两边的表没有主次关系。而外连接有。左连接是左边的表是主表，右连接是右

边的表是主表。

外连接的查询结果条数一定大于内连接的查询结果条数。因为在内连接中，不符合on的筛选条件的记录不会查询出来，而外连接，无论符不符合on的筛选条件，首先主表的记录肯定是全部查询出来的。

在一条SQL语句中，内连接和外连接可以同时出现。

## 5.子查询(了解)

select语句中嵌套select语句，被嵌套的select语句称为子查询。

可出现的位置：

select ...(select) from ...(select) where ...(select)

例如：

1.出现在where后

select ename,sal from emp where sal > (select min(sal) from emp);

2.出现在from后，可以当作一张临时表

select t.\*,s.grade from (select job,avg(sal) as avgsal from emp group by job) t join salgrade s on

t.avgsal between s.losal and s.hisal;

3.出现在select后

select e.ename,e.deptno,(select dname from dept) as dname from emp e;

注意：这时当子查询返回的结果不是一个，而是多个时，就会报错：

ERROR 1242(21000):Subquery returns more than 1 rows

## 6.union合并结果集

union的效率要高于表的连接查询，每连接一次新表，则匹配的次数满足笛卡尔积。

union可以减少匹配的次数。例句：

select ename,job from emp where job = ‘MANAGER’

union

select ename,job from emp where job = ‘SALESMAN’

Mysql只要求两个结果集的列数相同即可。比如：

select ename,job from emp where job = ‘MANAGER’

union

select ename,sal from emp where job = ‘SALESMAN’

这条语句不会报错。

但是Oracle要求两个结果集的列数相同，且数据类型也得相同。

## 7.limit

将查询结果的一部分显示出来，通常用于分页查询。

语法：limit startIndex,length。startIndex是起始下标，length是长度。

起始下标可以省略它，默认是0。如：limit 5 是取前5条。

limit是在order by 之后执行。

取出工资排名在[3-5]名的员工：

select ename,sal from emp order by sal desc limit 2,3;

分页的实现：



## 8.Mysql的数据类型

varchar：可变长的字符串，会根据实际的数据长度分配空间

优点：节省空间

缺点：动态分配空间，速度慢

在mysql4.0之前的版本中，varchar(10)中的10指的是10个字节。在mysql5.0以后的版

本中，这个10指的是10个字符。一般在定义时，varchar(255)就够用。

char：定长字符串，根据固定的长度去存储数据

优点：速度快

缺点：使用不当会导致空间的浪费

text：特殊字符串类型，允许长度为0~65535字节，其查询速度比char和varchar慢。

int：整数型，相当于java的int

bigint：长整型，相当于java的long

float：单精度浮点型，相当于java的float

double：双精度浮点型，相当于java的double

date：短日期类型，只包括年月日

默认格式：%Y-%m-%d

time：时间类型，只包含时分秒

默认格式：%h:%i:%s

datetime：长日期类型，包括年月日，时分秒

默认格式：%Y-%m-%d %h:%i:%s

timestamp：长日期类型，包括年月日，时分秒

默认格式：%Y-%m-%d %h:%i:%s。但是只能存储”1970-01-01 00:00:01 UTC”到”2038-01-19

03:14:07 UTC”之间的时间。

clob：Character Large Object，字符大对象，最多可以存储4G的字符串

blob：Binary Large Object，二进制大对象，专门用来存储图片、声音、视频等流媒体数据

得用IO流往这种数据类型中插入数据

decimal(M,D)：定点数类型，用于表示高精度小数。其中，M被称为精度，D被称为标度。

0<=M<=65，0<=D<=30，D<M。例如，定义DECIMAL(5,2)的类型，表示该列取值范围

是-999.99~999.99。该类型适用于金额计算的场景。

## 9.存储引擎

是一个表存储/组织数据的方式。不同的引擎，表存储数据的方式不同。

show create table 表名; --展示创建表的sql语句

可以在建表的时候指定存储引擎：

create table tbl\_user(

`no` int(11) not null auto\_increment,

`name` varchar(255) default null,

`cno` int(11) default null

) engine=InnoDB default charset=utf-8;

建表时指定存储引擎是InnoDB以及字符集是utf-8。

建表时默认存储引擎是InnoDB，字符集是utf-8。

show engines; --展示当前数据库管理系统对所有引擎的支持情况



(1)MYISAM存储引擎：

特征：使用三个文件表示每个表

格式文件：存储表结构的定义(mytable.frm)

数据文件：存储表行的内容(mytable.MYD)

索引文件：存储表上索引(mytable.MYI)

优点：可被转换为压缩、只读表来节省空间

(2)InnoDB存储引擎：

是mysql的默认存储引擎；支持事务，支持数据库崩溃后自动恢复机制；非常安全；

特征：

每个InnoDB表在数据库目录中以.frm格式文件表示

InnoDB表空间tablespace被用于存储表的内容(数据+索引)

提供一组用来记录事务性活动的日志文件

用COMMIT(提交)、SAVEPOINT及ROLLBACK(回滚)支持事务处理

提供全ACID兼容

在mysql服务器崩溃后提供自动恢复

多版本(MVCC)和行级锁定

支持外键及引用的完整性，包括级联删除和更新

(3)MEMORY存储引擎：

使用该引擎的表，数据存在内存中，且行的长度固定，使得该引擎速度非常快。

特征：

在数据库目录内，每个表均以.frm格式的文件表示

表数据及索引被存储在内存中。目的就是快

关机后，数据消失

表级锁机制

不能包含TEXT或BLOB字段

以前被称为HEAP引擎

## 10.事务(transaction)

一个事务就是一个完整的业务逻辑，是一个最小的工作单元。

比如朝A账户减去1000，给B账户加上1000，这是一个完整的业务逻辑，要么同时成功，

要么同时失败。

事务的4个特性：

A：原子性

说明事务是最小的工作单元，不可再分。

C：一致性

所有事务要求，在同一个事务中，所有操作必须同时成功，或者同时失败。

I：隔离性

A事务和B事务之间具有一定的隔离性。

D：持久性

事务最终结束的一个保障。事务提交，相当于将没有保存到硬盘上的数据保存到

硬盘上。

只有DML语句才会有事务这一说，也就是insert、delete、update。

事务是如何实现的：InnoDB存储引擎支持事务

事务开启

insert

delete

update

...

事务结束

在事务的执行过程中，每一条DML的操作都会记录到”事务性活动的日志文件”中而不会直

接在表中修改记录，修改记录的时机是事务提交之后。

在事务的执行过程中，可以提交事务，也可以回滚事务

提交事务，即将数据全部彻底持久化到数据库表中。

标志着完成了一个完整的业务逻辑，是成功的结束。

回滚事务，将之前所有的DML操作全部撤销，并且清空”事务性活动的日志文件”

标志着事务的结束，是失败的结束。

提交事务：commit;语句

回滚事务：rollback;语句(回滚永远只能回滚到上一次的提交点)

mysql默认情况下，每执行一条DML语句，就会自动提交一次。

如何正常的执行事务呢：

start transaction; --开启事务(同时会关闭自动提交)

... DML语句

commit; -- 提交事务

-- 或者

rollback;-- 回滚事务

## 11.事务之间的隔离级别

(1)读未提交：read uncommitted(最低的隔离级别)

含义：事务A可以读取到事务B未提交的数据。

存在的问题：脏读现象！(Dirty Read)，也就是读到了脏数据

这种隔离级别一般都是理论上的，大多数的数据库隔离级别都是第二种起步。

(2)读已提交：read committed

含义：事务A可以读取到事务B提交之后的数据。

解决的问题：脏读现象

存在的问题：不可重复读

含义：在事务开启之后，第一次读到的数据是3条，当前事务还没有结束，可能第二次

再读取的时候，读到的数据是4条，3!=4，称为不可重复读。

oracle数据库的默认隔离级别就是read committed

(3)可重复读：repeatable read

含义：事务A开启之后，不管是多久，每一次在事务A中读取到的数据都是一致的，

即使事务B将数据已经修改，并且提交了，事务A读取到的数据还是没有发生改

变。这就是可重复读。

解决的问题：

不可重复读

存在的问题：  
 幻读现象

含义：事务A读取到的数据永远都是事务A刚开启事务时的数据，不真实。

mysql数据库的默认隔离级别就是repeatable read

案例：银行执行select语句，执行时间3个小时，为了保证数据完整性，会使用这种隔

离级别。

(4)序列化/串行化：serializable(最高的隔离级别)

这是最高的隔离级别，解决了所有的问题，使用了表级别的锁。

但是它不能并发，因此效率最低，类似于加了synchronized，线程同步。

mysql查询隔离级别： 8.0之前用select @@tx\_isolation，

8.0及之后用select @@transaction\_isolation

修改mysql的隔离级别为 读未提交：read uncommitted(最低的隔离级别)

set global transaction isolation level read uncommitted;

修改执行后，需要退出重进一下mysql。

修改mysql的隔离级别为 读已提交：read committed

set global transaction isolation level read committed;

修改执行后，需要退出重进一下mysql。

修改mysql的隔离级别为 可重复读：repeatable read

set global transaction isolation level repeatable read;

修改执行后，需要退出重进一下mysql。

修改mysql的隔离级别为 序列化/串行化：serializable(最高的隔离级别)

set global transaction isolation level serializable;

修改执行后，需要退出重进一下mysql。

## 12.索引

含义：索引是在数据库表的字段上添加的，是为了提高查询效率存在的一种机制。一张表的

一个字段可以添加一个索引，当然，多个字段联合起来也可以添加索引，索引相当于一

本书的目录，是为了缩小扫描范围而存在的一种机制。

原理：索引其实也就是一种排序规则的实现。比如：对于select \* from user where name = ‘ab’;

(1)没有给name字段加上索引，那么数据库会一个一个的检查name的值，直到匹配。

(2)给name字段加上了索引，索引会根据某种排序规则给name字段分区。比如：

将首字母是a的分一个区，此时，数据库会直接去a分区查name，而不用一个一

个去比对name的值，从而提高了查询效率。

在InnoDB中，索引是一个B+Tree数据结构。



注意：(1)在任何数据库上，主键都会自动添加索引。另外在mysql当中，一个字段上如果有

unique约束的话，也会自动创建索引。

(2)在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有一个硬盘的的

物理存储编号。

(3) 在mysql当中，索引是一个单独的对象，不同的存储引擎以不同的形式存在，在

MyISAM存储引擎中，索引存储在一个.MYI文件中。在InnoDB存储引擎中索引存储在

一个逻辑名称叫做tablespace的当中。在MEMORY存储引擎当中索引被存储在内存当

中。不管索引存储在哪里，索引在mysql当中都是一个树的形式存在。

什么条件下，字段上适合加索引：

(1)数据量大，感觉查询缓慢

(2)该字段经常出现在where的后面，以条件的形式存在，也就是说这个字段总是被扫

描。

(3)该字段很少的DML(insert delete update)操作。(因为DML之后，索引需要重新排序)

建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。

建议通过主键查询，建议通过unique约束的字段进行查询，效率是比较高的。

语法：

(1)创建索引

格式：create index 索引名 on 表名(字段名1,字段名2...);

(2)删除索引

格式：drop index 索引名 on 表名;

索引名可以是 表名\_字段名\_index，这样比较清晰。

(3)如何查看一个SQL语句是否使用了索引

-- 加上explain，比如：

explain SELECT \* FROM `dept` where dname = 'ACCOUNTING'



如图所示type的值是ALL，而且rows的值是4，全部记录的条数也是4，由此可见

dname字段没有加索引。

-- 给dname字段加上索引之后再查询

create index bj\_dname\_index on dept(dname);

explain SELECT \* FROM `dept` where dname = 'ACCOUNTING'



如图所示，type的值是ref，且rows的值是1，表示语句只扫描了1条记录就查询

到了结果，由此可见dname字段是加上了索引。

索引什么时候会失效：

(1)比如：select \* from emp where ename like ‘%T’;

ename即使加上了索引，查询也不会走索引，因为这是开头就模糊匹配了。

尽量避免模糊查询的时候以”%”开头。

(2) 使用or的时候会失效。如果使用or那么要求or两边的条件字段都要有索引，才会

走索引，如果其中一边有一个字段没有索引，那么另一个字段上的索引也会失效。所以这就是为什么不建议使用or的原因。or可以用union代替。

(3)使用复合索引时

两个字段，或者更多的字段联合起来添加一个索引，叫做复合索引。

使用复合索引时，没有使用左侧的列查找,索引失效。如：

-- 创建复合索引



-- 使用左侧的列查询，会走索引



-- 使用右侧的列查询，没走索引



(4)在where中索引列参加了运算



这样，查询也不会走sal索引。

(5)在where中索引列使用了函数

比如：select \* from emp where lower(ename) = ‘smith’;

索引的分类：

(1)单一索引

(2)复合索引

(3)主键索引

(4)唯一性索引

注意：唯一性比较弱的字段上添加索引用处不大，因为重复数据多，分不分区意义不大。

越唯一，索引效率越高。

## 13.视图(view)

视图是站在不同的角度去看待同一份数据。

视图是存储在硬盘上的，关机不会消失。

创建视图：

create view dept\_view as select \* from dept; -- as 后必须是DQL语句

删除视图：

drop view dept\_view;

特点：可以像操作表一样操作视图，对视图中的记录进行增删改，也会影响到原表中的记录。

作用：简化SQL语句。

假如有一条很复杂的sql语句，而且还需要在不同的位置反复使用，每一次使用还需

要重新编写，很麻烦。这时可以把这条很复杂的sql语句以视图对象的形式新建，在

需要编写这条sql语句的地方直接使用视图对象，这样可以大大简化开发，并且利于

维护，因为修改的时候只需要修改一个位置就可以，只需要修改视图对象所映射的sql

语句。

## 14.DBA常见命令

(1) root账户创建用户。格式：

create user ‘username’@’host’ identified by ‘password’;

单词解析：

username指定用户名

password指定密码

host 指定该用户在哪个主机上可以登录，如果是本地用户可以使用localhost，如果想

让该用户可以从任意远程主机登录，可以使用通配符%，%表示任意数量的字符。

举例：

create user ‘lin’@’localhost’ identified by ‘123456’;

create user ‘lin’@’192.168.1.101’ identified by ‘123456’;

create user ‘lin’@’%’ identified by ‘123456’;

(2)创建用户的同时，授权。格式：

grant all privileges on \*.\* to 'yangxin'@'%' identified by 'yangxin123456' [with grant option];

注意：mysql8.0不支持创建用户的同时，授权。只能先创建，再授权。

单独授权的格式：

grant all privileges on \*.\* to 'yangxin'@'%' [with grant option];

注意：也可以只赋给字段权限。如：

grant select(id,name) on spring.user to ‘lijun’@’%’ [with grant option];

这样的话，连表都打不开，只能执行select id,name from spring.user;来查询字段。

单词解析：

all privileges：表示将所有权限授予给用户。也可指定具体的权限，如：SELECT、CREATE、

DROP等

on：表示这些权限对哪些数据库和表生效，格式：数据库名.表名，这里写“\*”表示所有

数据库，所有表。如果我要指定将权限应用到test库的user表中，可以这么写：test.user

to：将权限授予哪个用户。格式：”用户名”@”登录IP或域名”。%表示没有限制，在任

何主机都可以登录。比如：”yangxin”@”192.168.0.%”，表示yangxin这个用户只能在192.168.0IP段登录

identified by：指定用户的登录密码

with grant option：表示被创建的用户可以将自己拥有的权限授权给别人

注意：可以使用GRANT给用户添加权限，权限会自动叠加，不会覆盖之前授予的权限，比

如你先给用户添加一个SELECT权限，后来又给用户添加了一个INSERT权限，那么该用

户就同时拥有了SELECT和INSERT权限。

(3)刷新权限。格式：

flush privileges;

注意：每次修改权限后，记得刷新权限。

(4)查看用户的权限。格式：

-- CMD命令行中

select \* from mysql.user where user = ‘用户名’\G

--navicat中

select \* from mysql.user where user = ‘用户名’;

结果信息：

Host: 用户的主机

User: 用户名

authentication\_string: 加密之后的用户密码

Select\_priv：确定用户是否可以通过SELECT命令选择数据

Insert\_priv：确定用户是否可以通过INSERT命令插入数据

Update\_priv：确定用户是否可以通过UPDATE命令修改现有数据

Delete\_priv：确定用户是否可以通过DELETE命令删除现有数据

Create\_priv：确定用户是否可以创建新的数据库和表

Drop\_priv：确定用户是否可以删除现有数据库和表

Reload\_priv：确定用户是否可以执行刷新和重新加载MySQL所用各种内部缓存的特定

命令，包括日志、权限、主机、查询和表

Shutdown\_priv：确定用户是否可以关闭MySQL服务器，将此权限提供给root账户之外

的任何用户时，都应当非常谨慎

Process\_priv：确定用户是否可以通过SHOW

File\_priv：确定用户是否可以执行SELECT INTO OUTFILE和LOAD DATA INFILE命令

Grant\_priv：确定用户是否可以将已经授予给该用户自己的权限再授予其他用户，例如，

如果用户可以插入、选择和删除foo数据库中的信息，并且授予了GRANT权限，则该用户就可以将其任何或全部权限授予系统中的任何其他用户

References\_priv：目前只是某些未来功能的占位符，现在没有作用

Index\_priv：确定用户是否可以创建和删除表索引

Alter\_priv：确定用户是否可以重命名和修改表结构

Show\_db\_priv：确定用户是否可以查看服务器上所有数据库的名字，包括用户拥有足够

访问权限的数据库，可以考虑对所有用户禁用这个权限，除非有特别不可抗拒的原因

Super\_priv：确定用户是否可以执行某些强大的管理功能，例如通过KILL命令删除用户

进程，使用SET GLOBAL修改全局MySQL变量，执行关于复制和日志的各种命令

Create\_tmp\_table\_priv：确定用户是否可以创建临时表

Lock\_tables\_priv：确定用户是否可以使用LOCK

Execute\_priv：确定用户是否可以执行存储过程，此权限只在MySQL 5.0及更高版本中

有意义

Repl\_slave\_priv：确定用户是否可以读取用于维护复制数据库环境的二进制日志文件，

此用户位于主系统中，有利于主机和客户机之间的通信

Repl\_client\_priv：确定用户是否可以确定复制从服务器和主服务器的位置

Create\_view\_priv：确定用户是否可以创建视图，此权限只在MySQL 5.0及更高版本中有

意义

Show\_view\_priv：确定用户是否可以查看视图或了解视图如何执行，此权限只在MySQL

5.0及更高版本中有意义

Create\_routine\_priv：确定用户是否可以更改或放弃存储过程和函数，此权限是在MySQL

5.0中引入的

Alter\_routine\_priv：确定用户是否可以修改或删除存储函数及函数，此权限是在MySQL

5.0中引入的

Create\_user\_priv：确定用户是否可以执行CREATE

Event\_priv：确定用户能否创建、修改和删除事件，这个权限是MySQL 5.1.6新增的

Trigger\_priv：确定用户能否创建和删除触发器，这个权限是MySQL 5.1.6新增的

Create\_tablespace\_priv: 创建表的空间

(5)回收权限。格式：

revoke [权限] on \*.\* from 'yangxin’@’localhost';

举例：

revoke create on \*.\* from 'yangxin’@’localhost'; --回收create权限

(6)删除用户。格式：

select user,host from mysql.user; -- 查询所有用户名及其host

drop user ‘yangxiin’@’localhost’;

(7)用户重命名。格式：

rename user ‘test3’@’%’ to ‘test1’@’%’;

(8)修改密码。格式：  
 --mysql5.7之前

update user set password=password('123456') where user='root';

--mysql5.7之后

update user set authentication\_string=password('123456') where user='root';

(9)cmd命令导出数据库或者数据表。

注意：不需要登录mysql。

导出库的格式：

mysqldump 库名>路径名\文件名.sql -u用户名 -p密码

导出表的格式：

mysqldump 库名 表名>路径名\文件名.sql -u用户名 -p密码

如：mysqldump user>D:\usertable.sql -uroot -p123456

-- root用户将user库导出到D盘下usertable.sql文件中

补充：mysqldump其实是C:Program Files(86)\Mysql\Mysql Server8.0\bin下的一个文件，

叫做mysqldump.exe

## 15.数据库设计三范式

范式的含义：

数据库表的设计依据。教你怎么进行数据库表的设计。

第一范式：

要求任何一张表必须有主键，每一个字段原子性不可再分。

第二范式：

建立在第一范式的基础之上，要求所有非主键字段完全依赖主键;不要产生部分依赖。

第三范式：

建立在第二范式的基础之上，要求所有非主键字段直接依赖主键，不要产生传递依赖。

范式的目的：

设计数据库表的时候，按照以上的范式进行，可以避免表中数据的冗余，空间的浪费。

但有时候，为了减少表的连接次数，会合理的冗余，这样也会降低sql语句的编写难度。

## 设置字段唯一

ALTER TABLE table ADD UNIQUE (field);  
ALTER TABLE table ADD UNIQUE (field1,field2);

当设置字段唯一时，DBMS还会给唯一性的字段添加唯一性索引。

注意：在InnoDB引擎下，唯一性索引的字段大小不能超过767个字节。

若表的字符集是utf-8，则一个字符对应三个字节，255\*3=765，即varchar类型的字段设置唯一性索引时，它的大小不能超过varchar(255);

若表的字符集是utf8mb4，则一个字符对应四个字节，191\*4=764，即varchar类型的字段设置唯一性索引时，它的大小不能超过varchar(191);

## 2.字面量/字面值

假如user表中有15条记录，那么

select ‘abc’ as abc from user; --会查出15条记录，且只查出列名为abc的那一列，该列所有的值都是‘abc’。

select 2000 from user; --与上面类似。会查出15条记录，且只查出列名为2000的那一列，该列所有的值都是2000。

## 1. 随机生成字符串，以及查询插入法

INSERT INTO tp\_sys\_profile\_infoset ( id, profile\_id, info\_category, relate\_id, isenable ) SELECT

concat(

'afeccsqe01',

floor( 10000+rand ()\* 90000 ),

floor( 10000+rand ()\* 90000 )),

id,

'permission',

'aab020011',

'0'

FROM

tp\_sys\_profile;

## 2.查询某字段数据大于20字符数(LENGTH函数)

select \* from tp\_sys\_profile\_infoset where LENGTH(RELATE\_ID) > 20;

## 4.case when 函数(两种使用格式)

CASE WHEN 条件 THEN 表达式1 ELSE 表达式2 END

例：SELECT CASE WHEN a.label is not null THEN a.label ELSE a.name END AS name FROM tp\_user

a;

--下面这种格式类似于Java的swith case 语句

CASE 字段 WHEN 条件1 THEN 表达式1 WHEN条件 2 THEN 表达式2... ELSE 表达式n

END

例：SELECT CASE a.label WHEN ‘城市’ THEN a.label WHEN ‘国家’ THEN a.name ELSE a.name END

AS name FROM tp\_user a;

## 5.not in

mysql可以使用not in 来比较两个查询结果之间的差异。

## 6. 返回多条数据中的第一条

有时候查询返回的数据有多条，可以选择返回第一条。

条件加上limit 1即可

## 7.delete语句中使用别名会报错

正常：delete from Person where id = ‘2’

错误：delete from Person p where p.id = ‘2’

改正后：delete p from Person p where p.id = ‘2’

## 8. FIELD函数

在使用in的时候，想要查询的记录的顺序与in中的数据的顺序一致，可以使用FIELD函数。

方法：order by FIELD(in中的元素的字段名,in中的全部数据);如：

order by FIELD(t1.codevalue,'组织/政府','建筑材料业')

# Oracle数据库

## 1.直接将数据库中查询的Date型数据格式化为String型的字符串

TO\_CHAR(EVALUATION\_DATE,'yyyy-mm-dd hh24:mi') EVALUATION\_DATE

## 2.获得日期类型字段最新(时间最晚)的记录

SELECT NAME FROM EMPLOYEE WHERE DATE = (SELECT MAX(DATE) FROM EMPLOYEE);

## 3.修改字段长度

alter table 表名 modify 列名 数据类型；

alter table bl\_yhsz modify zcmc varchar2(120);

## 4.随机生成字符串，以及查询插入法

INSERT INTO tp\_sys\_profile\_infoset ( id, profile\_id, info\_category, relate\_id, isenable ) SELECT

'afeccsqe01' || TRUNC ( 10000+90000 \* dbms\_random.VALUE ) || TRUNC ( 10000+90000 \* dbms\_random.VALUE ),

id,

'permission',

'aab020011',

'0'

FROM

tp\_sys\_profile;

## 5.NVL(ex1,ex2)函数

与mysql的IFNULL(ex1,ex2)函数使用方法一致

## 6.返回多条数据中的第一条

有时候查询返回的数据有多条，可以选择返回第一条。

条件加上and rownum=1即可

## 7.null和’’

oracle中对于char和varchar2类型的字段，没有空字符串的概念，空字符串会被转化为null。例如：label为varchar2类型，则NVL(label,’’)，当label为null时，返回的依然是null而不是’’。

## 8. instr函数

在使用in的时候，想要查询的记录的顺序与in中的数据的顺序一致，可以使用instr函数。

方法：order by INSTR(in中的全部数据,in中的数据所属字段名);如：

order by instr('组织/政府','建筑材料业', t1.codevalue)

## 9. oracle查询sql语句执行时间

格式：explain plan set statement\_id='T\_TEST' for sql语句

例如：explain plan set statement\_id='T\_TEST' for select \* from user;

执行后，下方会显示select \* from user;的执行时间。

# SqlServer数据库

## 1.查询视图

select \* from [zy].[IN].PAT\_ALL\_INFO\_VIEW where IN\_PAT\_NAME LIKE '%曹艳%'

--zy是数据库名，IN是视图之上的模式名，PAT\_ALL\_INFO\_VIEW是视图名

## 2.查询某个字段都在哪些表里存在

select sysobjects.name from syscolumns inner join sysobjects on syscolumns.id = sysobjects.id and sysobjects.xtype = 'u' where syscolumns.name = '字段名'

## 3.将数据库查询的null转化为其他值

isnull(字段名称,转化的其他值)[AS 别名]

## 4.查询多个字段联合起来数据重复

select 字段1,字段2 from 表名  group by 字段1,字段2  having count(\*)>1

# PostgreSql数据库

## 1.时间范围查询

SELECT \* FROM "nmr\_nursing\_order" WHERE scheduled\_execution\_time >= '2020-11-03 00:00' AND scheduled\_execution\_time <= '2020-11-06 23:59'

其中，nmr\_nursing\_order是表名，scheduled\_execution\_time是它的字段且类型为Date。

此语句在mybatis中的形式请去另一个word文件 JAVA问题及轮子 的Mapper接口中方法里的形参问题 查看。

## 2.bigint和varchar类型的转换

cast(‘10086’ as bigint) --varchar转换为bigint

cast(1105698 as varchar) -- bigint转换为varchar

## 3.查询一个表中所有字段

select \* from information\_schema.columns

where table\_schema='public' and table\_name='表名称 ';

## 4.双重排序

order by cloume1 asc,cloume2 desc

## 5.查询数据库占用空间大小

SELECT pg\_size\_pretty(pg\_database\_size('数据库名称'));

## 6.查询数据库各表占用空间大小

SELECT

table\_schema || '.' || table\_name AS table\_full\_name,

pg\_size\_pretty(pg\_total\_relation\_size('"' || table\_schema || '"."' || table\_name || '"')) AS size

FROM information\_schema.tables

ORDER BY

pg\_total\_relation\_size('"' || table\_schema || '"."' || table\_name || '"') DESC

## 7.查询某个数据库中的某个表是否存在某列

SELECT  
 count(\*)  
FROM  
 information\_schema.COLUMNS  
WHERE  
 table\_schema = '数据库名'  
 AND TABLE\_NAME = '表名'  
 AND COLUMN\_NAME = '列名'

当结果小于1时，表示该表中没有该列

## 8.用navicat创建postgres数据库的角色并赋予权限

(1)以超级用户登录navicat并连接至postgres数据库



(2)新建角色



(3)设置角色权限







(4)查询和修改角色默认的访问模式

show search\_path;--查询

--修改时，需要在数据库路径下执行修改语句



SET search\_path TO nurs,nurs;--修改(要访问nurs模式下的表)

## 9.展示和修改最大连接数

(1)默认最大连接数是100



(2)修改最大连接数





改完最大连接数之后需要重启postgresql服务！

## 10.基础知识

(3)修改数据库的最大连接数为100

ALTER DATABASE 库名称 WITH CONNECTION LIMIT = 100; 或

ALTER DATABASE 库名称 CONNECTION LIMIT 100;

(4)创建数据库并指定拥有者和编码格式

create database 库名称 with owner = 用户名称 encoding = ‘utf-8’

## 11.数值类型

(1)整数类型：

SMALLINT //小范围整数，取值范围：-32768~32767

INT(INTEGER) //普通大小整数，范围-2147483648~2147483647

(2)浮点类型：

REAL //6位十进制数字精度

NUMERIC(m,n) //任意精度类型，m表示数值最大长度，n表示小数占几位。

注意：如果小数位超出了n的个数，那么数据库会自动四舍五入该小数。比如：

一个字段的类型为numeric(5,2)，而插入的数据为100.888，则数据库自动将100.888

的四舍五入为100.89，谨记。

(3)日期与时间类型

类型名称 含义 存储需求 举例

TIME 只用于一日内时间 8字节 10:05:05

DATE 只用于日期 4字节 1987-04-04

TIMESTAMP 日期和时间 8字节 1987-04-04 10:05:05

(4)字符串类型

CHAR(n)/CHARACTER(n) 固定长度字符串，不足补空白

VARCHAR(n)/CHARACTER VARYING(n) 变长字符串，有长度限制

TEXT 变长字符串，无长度限制

选择正确数据类型的好处：优化存储空间，提高数据库性能。

## 12.运算符

1.比较运算符





补充：(1)字符串与数字比较时，会将字符串转换为数字与其比较；

(2)null与任何数据比较，结果都是空。



补充：between 3 and 6 范围是[3,6]，两边都是闭区间。





2.逻辑运算符(not and or)



补充：’1’ 和 ‘y’ 一般是true，而’0’ 和 ‘n’ 一般是false





## 13.常用函数









## 14.自定义函数

(1)基本语法格式



补充：(1)$1 和 $2 分别代表传入的第一个和第二个参数

(2)create function 可以改为 create or replace function 保证当已存在相同名字的自定

义函数时，可以替换掉那个函数。

(3)select $1 + $2 中的 + 表示相加，而 || 则可以表示数据的连接。

(2)删除自定义函数

drop function 函数名(参数列表)

## 15.索引

(1)分类



创建索引(默认B-tree索引类型)

create index 索引名称 on 表名(列名)

删除索引

drop index 索引名称

(2)索引的优缺点

优点：提高数据的查询速度

加速表与表之间的连接

缺点:

创建和维护索引需要耗费时间

需要占用磁盘空间

## 16.视图

(1)创建视图

create view 视图名 as select语句;

(2)删除视图

drop view 视图名;

## 17.limit的使用

select \* from student limit 5;--表示查询前5条数据

select \* from student limit 5 offset 5;--表示查询第6-10条数据

limit m offset n 表示忽略前n条数据，从n+1条往下查m条数据。

## 18.连接查询

(1)内连接查询



说明：上面第一条为隐式的内连接查询，第二条为显式的内连接查询，但结果都是一样的。

(2)左外连接查询



(3)右外连接查询





## 19.union 和 union all





二者都可以将多条查询的结果联合起来。

补充：如果第一条语句查询的字段有e.hireDate，而第二个语句没有，则第二个语句可以用null代替它。





## 20.exists的使用

sql语句性能调优

# MongoDB数据库

## 1.MongoDB相关概念

**(1)业务应用场景**

传统的关系型数据库(如MySQL)，在数据操作的三高需求以及应对Web2.0的网站需求面前，

显得力不从心，而MongoDB可应对三高需求。

三高需求：

• High performance - 对数据库高并发读写的需求。

• Huge Storage - 对海量数据的高效率存储和访问的需求。

• High Scalability && High Availability - 对数据库的高可扩展性和高可用性的需求。

具体的应用场景：

1）社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理

位置索引实现附近的人、地点等功能。

2）游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的

形式存储，方便查询、高效率存储和访问。

3）物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以

MongoDB 内嵌数组的形式来存储，一次查询就能将 订单所有的变更读取出来。

4）物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，

并对这些信息进行多维度的分析。

5）视频直播，使用 MongoDB 存储用户信息、点赞互动信息等。

这些应用场景中，数据操作方面的共同特点是：

1）数据量大。

2）写入操作频繁（读写都很频繁）。

3）价值较低的数据，对事务性要求不高。

什么时候选择MongoDB？

a.应用不需要事务及复杂 join 支持

b.新应用，需求会变，数据模型无法确定，想快速迭代开发

c.应用需要2000-3000以上的读写QPS（更高也可以）

d.应用需要TB甚至 PB 级别数据存储

e.应用发展迅速，需要能快速水平扩展

f.应用要求存储的数据不丢失

g.应用需要99.999%高可用

h.应用需要大量的地理位置查询、文本查询

如果上述有1个符合，可以考虑 MongoDB，2个及以上的符合，MongoDB会非常合适。

**(2)MongoDB简介**

MongoDB是一个开源、高性能、无模式的文档型数据库，当初的设计就是用于简化开发和方便扩展，是NoSQL数据库产品中的一种。是最像关系型数据库（MySQL）的非关系型数据库。它支持的数据结构非常松散，是一种类似于JSON的格式叫BSON，所以它既可以存储比较复杂的数据类型，又相当的灵活。 MongoDB中的记录是一个文档，它是一个由字段和值对（field:value）组成的数据结构。MongoDB文档类似于JSON对象，即一个文档认为就是一个对象。字段的数据类型是字符型，它的值除了使用基本的一些类型外，还可以包括其他文档、普通数组和文档数组。

**(3)体系结构**

|  |  |  |
| --- | --- | --- |
| **SQL术语/概念** | **MongoDB术语/概念** | **解释/说明** |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins |  | 表连接,MongoDB不支持 |
|  | 嵌入文档 | MongoDB通过嵌入式文档来替代多表连接 |
| primary key | primary key | 主键,MongoDB自动将\_id字段设置为主键 |

**(4)数据模型**

MongoDB的最小存储单位就是文档(document)对象。文档(document)对象对应于关系型数据库的行。数据在MongoDB中以 BSON（Binary-JSON）文档的格式存储在磁盘上。

BSON（Binary Serialized Document Format）是一种类json的一种二进制形式的存储格式，简称Binary JSON。BSON和JSON一样，支持内嵌的文档对象和数组对象，但是BSON有JSON没有的一些数据类型，如Date和BinData类型。

BSON采用了类似于 C 语言结构体的名称、对表示方法，支持内嵌的文档对象和数组对象，具有轻量性、可遍历性、高效性的三个特点，可以有效描述非结构化数据和结构化数据。这种格式的优点是灵活性高，但它的缺点是空间利用率不是很理想。

Bson中，除了基本的JSON类型：string,integer,boolean,double,null,array和object，mongo还使用了特殊的数据类型。这些类型包括 date,object id,binary data,regular expression 和code。每一个驱动都以特定语言的方式实现了这些类型，查看你的驱动的文档来获取详细信息。

BSON数据类型参考列表：

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **举例** |
| 字符串 | UTF-8字符串都可表示为字符串类型的数据 | {"x" : "foobar"} |
| 对象id | 对象id是文档的12字节的唯一 ID | {"X" : ObjectId() } |
| 布尔值 | 真或者假：true或者false | {"x" : true} |
| 数组 | 值的集合或者列表可以表示成数组 | {"x" : ["a", "b", "c"]} |
| 32位整数 | 类型不可用。JavaScript仅支持64位浮点数，所以32位整数会被自动转换。 | shell是不支持该类型的，shell中默认会转换成64 位浮点数 |
| 64位整数 | 不支持这个类型。shell会使用一个特殊的内嵌文档来显示64位整数 | shell是不支持该类型的，shell中默认会转换成64 位浮点数 |
| 64位浮点数 | shell中的数字就是这一种类型 | {"x" : 3.14159，"y" : 3} |
| null | 表示空值或者未定义的对象 | {"x" : null} |
| undefined | 文档中也可以使用未定义类型 | {"x" : undefined} |
| 符号 | shell不支持，shell会将数据库中的符号类型的数据自动转换成字符串 |  |
| 正则表达式 | 文档中可以包含正则表达式，采用JavaScript的正则表达式语法 | {"x" : /foobar/i} |
| 代码 | 文档中还可以包含JavaScript代码 | {"x" : function() { /\* …… \*/ }} |
| 二进制数据 | 二进制数据可以由任意字节的串组成，不过shell中无法使用 |  |
| 最大值/最 小值 | BSON包括一个特殊类型，表示可能的最大值。shell中没有这个类型。 |  |

## 2.Windows版本单机部署

**(1)下载安装包**

下载地址：https://www.mongodb.com/download-center/community



MongoDB的版本命名规范如：x.y.z；

y为奇数时表示当前版本为开发版，如：1.5.2、4.1.13；

y为偶数时表示当前版本为稳定版，如：1.6.3、4.0.10；

z是修正版本号，数字越大越好。

**(2)解压安装**

将压缩包解压到一个目录中。

在解压目录中，手动建立一个目录用于存放数据库文件，如 data/db。



启动方式1：

在 bin 目录中打开命令行提示符，输入如下命令：mongod --dbpath=..\data\db





我们在启动信息中可以看到，mongoDB的默认端口是27017，如果我们想改变默认的启动端

口，可以通过--port来指定端口。 为了方便我们每次启动，可以将安装目录的bin目录

设置到环境变量的path中， bin 目录下是一些常用命令，比如 mongod 启动服务用的，

mongo 客户端连接服务用的。

启动方式2：

在解压目录中新建 config 文件夹，该文件夹中新建配置文件 mongod.conf，内容参考如下：



注意：缩进不要用Tab键，要用空格。

详细配置项内容可以参考官方文档：

<https://docs.mongodb.com/manual/reference/configuration-options/>

启动方式：mongod -f ../config/mongod.conf 或者 mongod --config ../config/mongod.conf

**(3) Shell连接(mongo命令)**

a. 在命令提示符输入以下shell命令即可完成登陆(前提是启动了MongoDB)

mongo 或 mongo --host=127.0.0.1 --port=27017

b. 查看已经有的数据库

show dbs 或者show databases

c. 退出mongodb

exit 或者 Ctrl+C

d. 更多参数可以通过帮助查看

mongo –help

## 3.Linux版本单机部署

## 4.基本常用命令

**(1)选择和创建数据库，如果数据库不存在则自动创建。语法格式：use 数据库名称。**

注意：创建数据库之后，该库只会存在内存中，只有往库里创建了集合后，该库才会持

久化到磁盘上。而且一个集合只有在内容插入后才会被创建。

**(2)查看有权限查看的所有数据库，命令：show dbs 或者 show databases。**

**(3)查看当前所处的数据库，命令：db。**

MongoDB 中默认的数据库为 test，如果你没有选择数据库，集合将存放在 test 数据

库中。

**(4)删除数据库，语法格式：db.dropDatabase()。**

提示：主要用来删除已经持久化的数据库。

补充：介绍一些MongoDB自带的数据库。

admin：从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据

库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能

从这个数据库运行，比如列出所有的数据库或者关闭服务器。

local：该库数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合。

config: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相

关信息。

**(5)集合的显示创建，语法格式：db.createCollection(集合名称)。**

**(6)集合的隐式创建，即向一个集合插入文档的时候，如果集合不存在，则会自动创建集合。**

**(7)查看当前库中的集合，命令：show collections 或 show tables。**

**(8)集合的删除，语法格式：db.collection.drop() 或 db.集合名称.drop()。**

如果成功删除，则drop()返回true，反之返回false。

**(9)单个文档的插入，使用insert() 或 save()方法，语法如下：**

db.集合名称.insert(

<document or array of documents>,

{

writeConcern: <document>,

ordered: <boolean>

}

)

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| document | document or array | 要插入到集合中的文档或文档数组。(json格式) |
| writeConcern | document | 插入文档时的性能级别。 |
| ordered | boolean | 可选。如果为真，则按顺序插入数组中的文档，如果其中一个文档出现错误，MongoDB将返回而不处理数组中的其余文档。如果为假，则执行无序插入，如果其中一个文档出现错误，则继续处理数组中的主文档。在版本2.6+中默认为true。 |

单个文档的插入，语句举例：

db.comment.insert({"articleid":"100000","content":"今天天气真好，阳光明媚","userid":"1001","nickname":"Rose","createdatetime":new Date(),"likenum":NumberInt(10),"state":null})

提示：

1）comment集合如果不存在，则会隐式创建。

2）mongo中的数字，默认情况下是double类型，如果要存整型，必须使用函数

NumberInt(整型数字)，否则取出来就有问题了。

3）插入当前日期使用 new Date()。

4）插入的数据没有指定 \_id ，会自动生成主键值，类型为ObjectID。如果手动指

定，其类型可以是ObjectID，也可以是MongoDB支持的其他类型。

5）如果某字段没值，可以赋值为null，或不写该字段。

注意：

1. 文档中的键/值对是有序的。

2. 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型(甚

至可以是整个嵌入的文档)。

3. MongoDB区分类型和大小写。

4. MongoDB的文档不能有重复的键。

5. 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。

**(10)多个文档的插入，语法如下：**

db.集合名称.insertMany(

[ <document 1>,<document 2>,<document 3>...],

{

writeConcern: <document>,

ordered: <boolean>

}

)

多个文档的插入，语句举例：

db.comment.insertMany([

{"\_id":"1","articleid":"100001","content":"我们不应该把清晨浪费在手机上，健康很重要，一杯温水幸福你我他。","userid":"1002","nickname":"相忘于江湖","createdatetime":new Date("2019-08-05T22:08:15.522Z"),"likenum":NumberInt(1000),"state":"1"},

{"\_id":"2","articleid":"100001","content":"我夏天空腹喝凉开水，冬天喝温开水","userid":"1005","nickname":"伊人憔悴","createdatetime":new Date("2019-08-05T23:58:51.485Z"),"likenum":NumberInt(888),"state":"1"},

{"\_id":"3","articleid":"100001","content":"我一直喝凉开水，冬天夏天都喝。","userid":"1004","nickname":"杰克船长","createdatetime":new Date("2019-08-06T01:05:06.321Z"),"likenum":NumberInt(666),"state":"1"},

{"\_id":"4","articleid":"100001","content":"专家说不能空腹吃饭，影响健康。","userid":"1003","nickname":"凯撒","createdatetime":new Date("2019-08-06T08:18:35.288Z"),"likenum":NumberInt(2000),"state":"1"},

{"\_id":"5","articleid":"100001","content":"研究表明，刚烧开的水千万不能喝，因为烫嘴。","userid":"1003","nickname":"凯撒","createdatetime":new Date("2019-08-06T11:01:02.521Z"),"likenum":NumberInt(3000),"state":"1"}]);

提示：

1)插入时指定了 \_id ，则主键就是该值。

2)如果某条数据插入失败，将会终止插入，但已经插入成功的数据不会回滚掉。

3)因为批量插入由于数据较多容易出现失败，因此，可以使用try catch进行异常捕

捉处理，测试的时候可以不处理。如（了解）：



**(11)文档的基本查询，语法格式：**

db.集合名称.find(<query>, [projection])。

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| query | document | 可选。使用查询运算符指定选择筛选器。若要返回集合中的所有文档，请省略此参数或传递空文档 ( {} )。 |
| projection | document | 可选。指定要在与查询筛选器匹配的文档中返回的字段（投影）。若要返回匹配文档中的所有字段，请省略此参数。 |

a.查询一个集合中的所有文档，命令：db.集合名称.find() 或 db.集合名称.find({})。

b.查询userid为1003的文档，命令：db.集合名称.find({userid:'1003'})。

c.查询第一条userid为1003的文档，命令：db.集合名称.findOne({userid:'1003'})。

d.查询userid为1003的文档，且只想查询结果显示userid和nickname这俩域，命令：

db.集合名称.find({userid:'1003'},{userid:1,nickname:1})。

\_id这个域是默认显示的，如果想去掉，可以这么写：

db.集合名称.find({userid:'1003'},{userid:1,nickname:1,\_id:0})。

这种查询方式称为投影查询（Projection Query）。

**(12)文档的更新，语法如下：**

db.集合名称.update(query, update, options) 或

db.集合名称.update(

<query>,

<update>,

{

upsert: <boolean>,

multi: <boolean>,

writeConcern: <document>,

collation: <document>,

arrayFilters: [ <filterdocument1>, ... ],

hint: <document|string> // Available starting in MongoDB 4.2

}

)

a. 覆盖修改

如果想将\_id为1的记录的likenum修改为1001，可以这么写：

db.集合名称.update({\_id:"1"},{likenum:NumberInt(1001)})

执行后我们会发现，这条文档除了likenum和\_id字段其它字段都不见了。

b. 局部修改

如果想将\_id为1的记录的likenum修改为1001，这条记录的其他字段值不变，可以这

么写：

db.集合名称.update({\_id:"1"},{$set:{likenum:NumberInt(1001)}})

c. 批量修改

更新所有userid为 1003 的用户的nickname为张三

//默认只修改第一条数据

db.集合名称.update({userid:"1003"},{$set:{nickname:"张三"}})

//修改所有符合条件的数据

db.集合名称.update({userid:"1003"},{$set:{nickname:"张三"}},{multi:true})

d. 列值增长的修改

如果我们想实现对某列值在原有值的基础上进行增加或减少，可以使用 $inc 运算符来

实现。

将\_id为3的记录的likenum在原有基础上加1

db.集合名称.update({\_id:"3"},{$inc:{likenum:NumberInt(1)}})

**(13)文档的删除，语法如下：**

db.集合名称.remove(条件)

a. 删除一个集合里的全部文档数据

db.集合名称.remove({})

b. 删除\_id=1的文档

db.集合名称.remove({\_id:"1"})

**(14)文档的分页查询**

a. 统计查询，语法如下：

db.集合名称.count(query, options)

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| query | document | 查询选择条件。 |
| options | document | 可选。用于修改计数的额外选项。 |

<1>统计所有记录数

db.集合名称.count()

<2>按条件统计记录数

//统计userid为1003的记录条数

db.集合名称.count({userid:"1003"})

b. 分页列表查询

可以使用limit()方法来读取指定数量的数据，使用skip()方法来跳过指定数量的数据。语法

如下：

db.集合名称.find().limit(NUMBER).skip(NUMBER)

<1>如果你想返回指定条数的记录，可以在find方法后调用limit来返回结果(TopN)，TopN默

认值为20。例如，查询集合中前三条文档：

db.集合名称.find().limit(3)

<2> skip方法同样接受一个数字参数作为跳过的记录条数，该参数默认值是0。

db.集合名称.find().skip(3)

<3>分页查询，每页查询两条记录

//第一页

db.comment.find().skip(0).limit(2)

//第二页

db.comment.find().skip(2).limit(2)

//第三页

db.comment.find().skip(4).limit(2)

c. 排序查询

sort() 方法可以通过参数指定排序的字段，并使用1 和 -1来指定排序的方式，其中1为升

序排列，而-1是用于降序排列。语法如下：

db.集合名称.find().sort({KEY:1}) 或 db.集合名称.find().sort(排序方式)

补充：MongoDB默认以\_id为升序排列，且自动生成的\_id，其所在的文档优先排在上面。

<1>对userid降序排列，并对likenum进行升序排列

db.comment.find().sort({userid:-1,likenum:1})

提示：

skip()、limilt()、 sort()三个放在一起执行的时候，执行的顺序是先 sort(), 然后是 skip()，

最后是显示的 limit()，和命令编写顺序无关。

**(15)文档的更多查询**

a. 正则的复杂条件查询

MongoDB的模糊查询是通过正则表达式的方式实现的。格式为：

db.集合名称.find({field:/正则表达式/}) 或 db.集合名称.find({字段:/正则表达式/})

提示：正则表达式是js的语法，直接量的写法。

<1>查询content包含“开水”的所有文档

db.集合名称.find({content:/开水/})

<2>查询content中以“专家”开头

db.集合名称.find({content:/^专家/})

b. 比较查询

<,<=, >, >= 这个操作符也是很常用的，格式如下：

db.集合名称.find({ "field" : { $gt: value }}) // 大于: field > value

db.集合名称.find({ "field" : { $lt: value }}) // 小于: field < value

db.集合名称.find({ "field" : { $gte: value }}) // 大于等于: field >= value

db.集合名称.find({ "field" : { $lte: value }}) // 小于等于: field <= value

db.集合名称.find({ "field" : { $ne: value }}) // 不等于: field != value

<1>查询评论likenum大于700的记录

db.comment.find({likenum:{$gt:NumberInt(700)}})

c. 包含查询

包含查询需要使用$in操作符。

<1>查询userid字段包含1003或1004的文档

db.comment.find({userid:{$in:["1003","1004"]}})

<2>查询userid字段不包含1003和1004的文档

db.comment.find({userid:{$nin:["1003","1004"]}})

d. 条件连接查询

如果查询需要同时满足两个以上条件，需要使用$and操作符将条件进行关联。语法如下：

$and:[ { },{ },{ } ]

如果两个以上条件之间是或的关系，需要使用$or操作符。语法如下：

$or:[ { },{ },{ } ]

<1>查询likenum大于等于700 并且小于2000的文档

db.comment.find({$and:[{likenum:{$gte:NumberInt(700)}},{likenum:{$lt:NumberInt(2000)}}]})

<2>查询userid为1003或者likenum小于1000的文档

db.comment.find({$or:[ {userid:"1003"} ,{likenum:{$lt:1000}}]})

## 5.索引(index)

**(1)概述**

索引是特殊的数据结构，它以易于遍历的形式存储集合数据集的一小部分。索引存储特定字段或一组字段的值，按字段值排序。索引项的排序支持有效的相等匹配和基于范围的查询操作。此外，MongoDB还可以使用索引中的排序返回排序结果。

官网文档：<https://docs.mongodb.com/manual/indexes/>

了解：MongoDB索引使用B树数据结构（确切的说是B-Tree，MySQL是B+Tree）。

**(2)索引的类型**

a. 单字段索引

MongoDB支持在文档的单个字段上创建用户定义的升序/降序索引，称为单字段索引（Single Field Index）。

对于单个字段索引和排序操作，索引键的排序顺序（即升序或降序）并不重要，因为MongoDB可以在任何方向上遍历索引。



b. 复合索引

MongoDB还支持多个字段的用户定义索引，即复合索引（Compound Index）。

复合索引中列出的字段顺序具有重要意义。例如，如果复合索引由 { userid: 1, score: -1 } 组成，则索引首先按userid正序排序，然后在每个userid的值内，再在按score倒序排序。



c. 其他索引

地理空间索引（Geospatial Index）、文本索引（Text Indexes）、哈希索引（Hashed Indexes）。

<1>地理空间索引（Geospatial Index）

为了支持对地理空间坐标数据的有效查询，MongoDB提供了两种特殊的索引：返回结果时使用平面几何的二维索引和返回结果时使用球面几何的二维球面索引。

<2>文本索引（Text Indexes）

MongoDB提供了一种文本索引类型，支持在集合中搜索字符串内容。这些文本索引不存储特定于语言的停止词（例如“the”、“a”、“or”），而将集合中的词作为词干，只存储根词。

<3>哈希索引（Hashed Indexes）

为了支持基于散列的分片，MongoDB提供了散列索引类型，它对字段值的散列进行索引。这些索引在其范围内的值分布更加随机，但只支持相等匹配，不支持基于范围的查询。

**(3)索引的管理操作**

a. 索引的查看

<1>返回一个集合中的所有索引的数组。语法：

db.集合名称.getIndexes()

提示：该语法命令运行要求是MongoDB 3.0+。

示例：

db.comment.getIndexes()

返回结果：

[

{

v: NumberInt("2"),

key: {

\_id: NumberInt("1")

},

name: "\_id\_",

ns: "articledb.comment"

}

]

结果说明：从返回结果可以看出，索引数组只有一个对象，即comment集合只有一个索引。

其中，v是MongoDB索引引擎的版本号；

key里面是当作索引的字段，可以看到只有一个字段，就是\_id，而NumberInt("1")表示

索引按1(升序)排列；

name是索引的名称，如果创建索引时不手动指定，MongoDB会自动生成一个索引名称；

而ns，全称是namespace(名称空间)，表示索引所在的名称空间，其值是库名.集合名。

b. 索引的创建

语法：db.集合名称.createIndex(keys, options)

参数：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| keys | document | 包含字段和值对的文档，其中字段是索引键，值描述该字段的索引类型。对于字段上的升序索引，请指定值1；对于降序索引，请指定值-1。比如： {字段:1或-1} ，其中1为指定按升序创建索引，如果你想按降序来创建索引指定为 -1 即可。另外，MongoDB支持几种不同的索引类型，包括文本、地理空间和哈希索引。 |
| options | document | 可选。包含一组控制索引创建的选项的文档。有关详细信息，请参见选项详情列表。 |

options（更多选项）列表：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| background | Boolean | 建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为false。 |
| unique | Boolean | 建立的索引是否唯一。指定为true创建唯一索引。默认值为false。 |
| name | string | 索引的名称。如果未指定，MongoDB通过连接索引的字段名和排序顺序生成一个索引名称。 |
| dropDups | Boolean | 3.0+版本已废弃。在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 false。 |
| sparse | Boolean | 对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档。默认值为 false。 |
| expireAfterSeconds | integer | 指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。 |
| v | index version | 索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。 |
| weights | document | 索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。 |
| default\_language | string | 对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语。 |
| language\_override | string | 对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language。 |

提示：

注意在 3.0.0 版本前创建索引方法为 db.集合名称.ensureIndex()，之后的版本使用了

db.集合名称.createIndex()方法，ensureIndex() 还能用，但只是 createIndex() 的别名。

<1>对userid字段建立索引

db.comment.createIndex({userid:1})

<2>对 userid 和 nickname 同时建立复合（Compound）索引

db.comment.createIndex({userid:1,nickname:-1})

c. 索引的移除

可以移除指定的索引，或移除所有索引。语法：

db.collection.dropIndex(index)

参数：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| index | string or document | 指定要删除的索引。可以通过索引名称或索引规范文档指定索引。若要删除文本索引，请指定 索引名称。 |

<1>删除 comment 集合中 userid 字段上的升序索引

db.comment.dropIndex({userid:1})

<2>删除 comment 集合中所有索引

db. comment.dropIndexes()

提示：\_id 字段的索引是无法删除的，只能删除非 \_id 字段的索引。

**(4)索引的使用**

a. 执行计划

分析查询性能（Analyze Query Performance）通常使用执行计划（解释计划、Explain Plan）来查看查询的情况，如查询耗费的时间、是 否基于索引查询等。

那么，通常，我们想知道，建立的索引是否有效，效果如何，都需要通过执行计划查看。

语法：db.collection.find(query,options).explain(options)

<1>查看根据userid查询数据的情况

db.comment.find({userid:"1003"}).explain()

返回结果：

{

plannerVersion: NumberInt("1"),

namespace: "articledb.comment",

indexFilterSet: false,

parsedQuery: {

userid: {

$eq: "1003"

}

},

winningPlan: {

stage: "FETCH",

inputStage: {

stage: "IXSCAN",

keyPattern: {

userid: 1,

nickname: -1

},

indexName: "userid\_1\_nickname\_-1",

isMultiKey: false,

multiKeyPaths: {

userid: [ ],

nickname: [ ]

},

isUnique: false,

isSparse: false,

isPartial: false,

indexVersion: NumberInt("2"),

direction: "forward",

indexBounds: {

userid: [

"[\"1003\", \"1003\"]"

],

nickname: [

"[MaxKey, MinKey]"

]

}

}

},

rejectedPlans: [

{

stage: "FETCH",

inputStage: {

stage: "IXSCAN",

keyPattern: {

userid: 1

},

indexName: "userid\_1",

isMultiKey: false,

multiKeyPaths: {

userid: [ ]

},

isUnique: false,

isSparse: false,

isPartial: false,

indexVersion: NumberInt("2"),

direction: "forward",

indexBounds: {

userid: [

"[\"1003\", \"1003\"]"

]

}

}

}

]

}

结果说明：plannerVersion是计划的内部版本；namespace是语句执行的名称空间(位置)；

parsedQuery是查询的条件；winningPlan里是成功查询的一些信息：stage是查询的方

式，值为COLLSCAN表示是全局查询，值为FETCH表示根据索引去检索指定文档。

inputStage里stage的值为IXSCAN，表示为基于索引的扫描。rejectedPlans是没有执行的查询，因为没有winningPlan中的查询方式高。

b. 涵盖的查询(覆盖索引)

含义：当查询条件和查询的投影仅包含索引字段时，MongoDB直接从索引返回结果，而不

扫描任何文档或将文档带入内存。 这些覆盖的查询非常高效。

<1>在userid上创建索引，并只查询userid

db.comment.createIndex({userid:1})

db.comment.find({userid:"1003"},{userid:1,\_id:0})

这样查询时会使用覆盖索引的查询，效率很高。

## 6. MongoDB集群和安全

**(1)副本集-Replica Sets**

a. 简介

MongoDB中的副本集（Replica Set）是一组维护相同数据集的mongod服务。 副本集可提供冗余和高 可用性，是所有生产部署的基础。

也可以说，副本集类似于有自动故障恢复功能的主从集群。通俗的讲就是用多台机器进行同一数据的异 步同步，从而使多台机器拥有同一数据的多个副本，并且当主库当掉时在不需要用户干预的情况下自动 切换其他备份服务器做主库。而且还可以利用副本服务器做只读服务器，实现读写分离，提高负载。

<1>冗余和数据可用性

复制提供冗余并提高数据可用性。 通过在不同数据库服务器上提供多个数据副本，复制可提供一定级别 的容错功能，以防止丢失单个数据库服务器。 在某些情况下，复制可以提供增加的读取性能，因为客户端可以将读取操作发送到不同的服务上， 在不 同数据中心维护数据副本可以增加分布式应用程序的数据位置和可用性。 您还可以为专用目的维护其他 副本，例如灾难恢复，报告或备份。

<2>MongoDB中的复制

副本集是一组维护相同数据集的mongod实例。 副本集包含多个数据承载节点和可选的一个仲裁节点。 在承载数据的节点中，一个且仅一个成员被视为主节点，而其他节点被视为次要（从）节点。 主节点接收所有写操作。 副本集只能有一个主要能够确认具有{w：“most”}写入关注的写入; 虽然在某 些情况下，另一个mongod实例可能暂时认为自己也是主要的。主要记录其操作日志中的数据集的所有 更改，即oplog。辅助(副本)节点复制主节点的oplog并将操作应用于其数据集，以使辅助节点的数据集反映主节点的数据 集。 如果主要人员不在，则符合条件的中学将举行选举以选出新的主要人员。

<3>主从复制和副本集区别

主从集群和副本集最大的区别就是副本集没有固定的“主节点”；整个集群会选出一个“主节点”，当其挂 掉后，又在剩下的从节点中选中其他节点为“主节点”，副本集总有一个活跃点(主、primary)和一个或多 个备份节点(从、secondary)。

b. 副本集的三个角色

副本集有两种类型，三种角色

两种类型：

主节点（Primary）类型：数据操作的主要连接点，可读写。

次要（辅助、从）节点（Secondaries）类型：数据冗余备份节点，可以读或选举。

三种角色：

主要成员（Primary）：主要接收所有写操作。就是主节点。

副本成员（Replicate）：从主节点通过复制操作以维护相同的数据集，即备份数据，不

可写操作，但可 以读操作（但需要配置）。是默认的一种从节点类型。

仲裁者（Arbiter）：不保留任何数据的副本，只具有投票选举作用。当然也可以将仲裁

服务器维护为副 本集的一部分，即副本成员同时也可以是仲裁者。也是一种从节

点类型。

关于仲裁者的额外说明：

您可以将额外的mongod实例添加到副本集作为仲裁者。 仲裁者不维护数据集。 仲裁者的目的是通过 响应其他副本集成员的心跳和选举请求来维护副本集中的仲裁。 因为它们不存储数据集，所以仲裁器可 以是提供副本集仲裁功能的好方法，其资源成本比具有数据集的全功能副本集成员更便宜。 如果您的副本集具有偶数个成员，请添加仲裁者以获得主要选举中的“大多数”投票。 仲裁者不需要专用 硬件。 仲裁者将永远是仲裁者，而主要人员可能会退出并成为次要人员，而次要人员可能成为选举期间的主要 人员。 如果你的副本+主节点的个数是偶数，建议加一个仲裁者，形成奇数，容易满足大多数的投票。 如果你的副本+主节点的个数是奇数，可以不加仲裁者。

c. 副本集的创建

<1>第一步：创建主节点