数据库考核

注

```
代码格式
```

字体：

1 DDL

1.1 简述 MySQL 中创建、切换、删除、查看创建数据库的语句

1. 创建数据库：create database 数据库名;

```
create database soft863;
```

2. 切换：use 数据库名;

```
use soft863;
```

3. 删除数据库：drop database 数据库名;

```
drop database soft863;
```

4. 查看数据库：show databases

1.2 简述 MySQL 中用户创建、删除语句

1. 创建用户：create user 用户名 identified by ‘密码’;

```
create user bigdata identified by ‘bigdata’;
```

2. 删除用户：drop user 用户名@localhost;

```
drop user bigdata@localhost;
```

1.3 简述 MySQL 中表创建语句的写法

创建表：

```
create table 表名 (
    字段 数据类型 （列级别约束条件）
    ……
    （表级别约束条件）
);
```

```
create table `user` (
`id` int not null,
`username` varchar (255) null,
`password` varchar (255) null,
primary key (`id`) -- 表级别约束条件
);
```

1.4 怎么为 MySQL 中某个数值自动设置为自增长

在建表时，在数值字段后面加上“AUTO_INCREMENT”，一个表只有一个字段使用自增长，还必须为主键的一部分

1.5 怎么查看 MySQL 数据表结构

1. describe 表名 / desc 表名：查看表的字段信息
2. show create table 表名：显示数据表的创建语句

1.6 简述给某个表更改名字的语句

alter table 旧表名 rename 新表名;

```
alter table user11 rename user1;
```

1.7 简述怎么修改表字段类型

alter table 表名 modify 字段 新类型;

```
alter table user11 modify flag int;
```

1.8 简述怎么修改表字段名

alter table 表名 change 旧字段名 新字段名 类型;

```
alter table user11 change remark flag varchar(255); //将字段名字 remark 改为 flag
```

1.9 简述怎么给表新增字段

alter table 表名 add '新字段' 类型 （约束条件）（after/before '已有字段'）;

```
alter table user11 add 'remark' varchar(255) after 'password';
```

1.10 简述怎么删除字段

使用 alter 删除字段：alter table 表名 drop 字段;

```
alter table user11 drop flag;
```

1.11 简述怎么给表修改存储引擎

alter table 表名 engine=新的存储引擎;

```
alter table user11 engine=InnoDB;
```

1.12 简述怎么删除表

使用 drop table 可以删除一个或多个没有被其他关联的数据表

drop table 表 1，表 2.....;

```
drop table user11;
```

2 DML

2.1 简述 SQL 怎么进行插入数据和批量插入数据

1. 插入数据：insert into 表名(字段 1，字段 2，•••) values (值 1，值 2，•••);

```
INSERT INTO dayexercise (f_sid, f_name, f_group, f_date, f_state, f_remark)
VALUES (1, '张三', '第四组', '2019-12-17', 1, NULL);
```

或者：insert into 表名 set 字段 1=值 1，字段 2=值 2 ••• ;

```
INSERT INTO dayexercise SET f_name='fjj',f_sid=4;
```

2. 批量插入数据：insert into 表名 values (值 1，值 2，•••)，(值 1，值 2，•••)，(值 1，值 2，•••) •••

```
INSERT INTO dayexercise
VALUES
(3,'里斯','第三组','2019-12-17',0,NULL),
(2,'丽丽','第四组','2019-12-17',1,NULL);
```

2.2 简述怎么进行更新数据

1. 部分更新：update 表名 set 字段 1=值 1，••• where 条件；

```
UPDATE student SET f_group='第五组',f_state=1 WHERE id=5;
```

2. 全部更新：update 表名 set 字段 1=值 1，•••

```
UPDATE student SET f_group='第五组';
```

2.3 简述数据删除的两种方式及区别

1. delete：delete from 表名 where 条件

```
delete from student where id<3;
```

删除全部数据：delete from 表名

```
delete from student;
```

2. truncate：truncate table 表名

```
truncate student;
```

区别：

(1) delete 后面可以跟 where 子句指定删除部分，truncat 只能删除整个表的所有记录；

(2) 使用 truncate 语句删除记录后，新添加的记录时会自动增长字段（如 id）会默认从 1 开始；而用 delete 删除记录后，新添加记录时，自动增加字段会从删除该字段的最大值加 1 开始计算（原本 id 最大为 5，则从 6 开始计算）。所以如果想要彻底删除一个表的记录而不会影响到重新添加记录，最好使用 truncate 来删除整个表的记录。

2.4 简述 SQL 单表查询关键字都有哪些及怎么使用

1. 关键字 * ：通配符代表全部字段

```
select * from dayexercise;
```

2. 关键字 as：表示为查询的字段取别名，可省略

```
select f_name 姓名, f_group 小组, f_score 成绩 from dayexercise;
```

根据已知的表通过一些约束条件来创建新的表，as 可不要

```
create table forthgroup as select * from dayexercise where f_group='第四组';
```

3. 关系运算符： 其中 <> 和 != 都表示不等于

```
select * from dayexercise where id >5;
select * from dayexercise where id<>3;
```

4. 关键字 in：用于判断某个字段的值是否在指定集合中

```
select * from dayexercise where id in (3,4);
```

5. 关键字 between and：用于判断摸个字段的值是否在指定范围内

```
SELECT * dayexercise WHERE f_date BETWEEN '2019-12-18' AND
DATE_ADD(''2019-12-18,INTERVAL 1 DAY);
```

6. 关键字 null：用 is null 来判断字段中的值是否为空；不为空用 is not null 表示

```
select * from dayexercise where f_remark is null;
select * from dayexercise where f_remark is not null;
```

7. 关键字 distinct：用 distinct 来过滤重复的值，只保留一个值

```
select distinct(f_group) from dayexercise;
```

8. 关键字 like：与通配符 % 和 __ 结合使用

（1） 百分号 % ：匹配任意长度的字符串，包括空字符串；

```
select * from dayexercise where f_name like '张%';
```

（2） 下划线__ ：只匹配单个字符，若要匹配多个字符，需要使用多个下划线通配符

```
select * from dayexercise where f_name like '张_';
```

9. 关键字 and：可以连接两个或者多个查询条件

```
select * from dayexercise where f_group='第三组' and f_score>60;
```

10. 关键字 or：只要满足任意一个条件就会被查询出来（**当 or 和 and 一起使用的时候，and 的优先级高于 or ，先运算 and 再运算 or**）

```
select * from dayexercise where f_group='第三组' and f_score>60;
```

2.5 简述 MySQL 和 Oracle 分页怎么实现

1. Mysql 中分页使用 limit ，与 offset（偏移量）连用

```
SELECT * FROM dayexercise LIMIT 5 OFFSET 0;
```

2. Orcale 中没有 limit 分页，只能用 rownum 来显示行号，然后通过行号的约束来进行查询

```
//先对显示所有数据的行号，然后对整理后的数据通过行号的约束来进行查询
select * from （select t.*,rownum as rowno from dayexercise t order by id）m where m.rowno >1 and m.rowno <4;
```

2.6 简述 GroupBy 和 Having 的用法

Select 字段 1，字段 2，••• from 表名 groupby 字段 1，字段 2 ••• having 条件；

1. Groupby 是按某个字段或多个字段进行分组；可以聚合函数一起使用

```
SELECT * FROM dayexercise GROUP BY f_group;
ELECT f_group, f_name, f_score FROM dayexercise GROUP BY f_group, f_name;
```

2. Having 与 where 用法相同，后接约束条件，但是 groupby 后面只能使用 having，用于对分组后的结果进行过滤；并且 having 可以跟聚合函数，但是 where 不可以

```
SELECT f_name , AVG( f_score) cs FROM dayexercise GROUP BY f_name HAVING (cs>60 AND cs<=90);
```

2.7 简述常见的聚合函数有哪些

函数名称	作用
count（）	返回某列的行数
sum（）	返回某列的值的和
avg（）	返回某列的平均数
max（）	返回某列的最大值
min（）	返回某列的最小值

```
select count(*) from dayexercise;
select avg(f_score) from dayexercise;
select max(f_score) from dayexercise;
select min(f_score) from dayexercise;
select sum(id) from dayexercise;
```

2.8 简述 MySQL 和 Oracle 字符串函数都有哪些

1. Mysql 中的字符串函数：
   （1） length（x）：字段的长度
   ```
   select f_name ,length(f_name) 长度  from dayexercise;
   ```
   （2） concat（s1，s2…）：拼接字段
   ```
   select concat(f_name,'|',f_score) from dayexercise;
   ```
   （3） trim（str）：删除字段两侧的空格
   ```
   select trim(f_group) from dayexercise;
   ```
   （4） replace（str，s1，s2）：字符串 s2 代替字符串 str 中所有的字符串 s1
   ```
   select replace (f_group,' ', '') from dayexercise;
   ```
   （5） substring（str，n，len）：截取字符，从 n 开始，长度为 len
   ```
   select substring(f_group,2,1) from dayexercise;
   ```
   （6） reverse（str）：反转字符
   ```
   select reverse('hello') from dual;
   ```
   （7） locate（s1，str）：查看指定字符在字段中所在位置
   ```
   select f_name, locate('b',f_name) location from dayexercise;
   ```
2. Orcale 中的字符串函数（与 Mysql 中有差别的字符串函数）：
   （1） 字符拼接：concat 或者 ||’’||
   ```
   // 1.用||''||  在引号里面可以添加你想用来分隔的符号
   select f_name||'_'||f_group from dayexercise;
   //2.concat 中只能加字段
   ```

```
select concat(f_name,f_score) from dayexercise;
```

（2） 截取字符：substr

```
select substr(f_group,2,1) from dayexercise;
```

（3） 查看指定字符在字段中的位置：instr

```
select instr(f_name ,'张') from dayexercise;
```

（4） 替代：replace

```
select replace(f_name,'王','') from dayexercise
```

（5） 反转字符串：reverse

```
select reverse('hello') from dual; //orcal 中必须要返回到一个表中，dual 是一个临时表
```

2.9 简述 MySQL 和 Oracle 日期函数都有哪些

1. Mysql 中的日期函数：
   （1） sysdate() / now() ：当前系统的时间
   （2） curdate() / current_date() : 当前日期（年月日）
   （3） curtime() / current_time() ：当前时间（时分秒）
   （4） time_to_sec：将日期转化为秒
   （5） adddate()：增加时间

```
select adddate(now(),interval 1 hour);
```

   （6） subdate()：减少时间
   （7） date_fromat：日期转换为指定格式

```
select date_format(sysdate(),'%Y 年%m 月%日 %H 小时%i 分钟%s 秒') date;
```

   （8） to_days() / to_seconds() ：转换为天/秒

```
select * from dayexercise where to_days(f_date)-to_days(now())>=2;
```

2. Orcale 中的日期函数：
   （1） 获取当前系统时间：

```
select sysdate from dual;
```

   （2） 获取当前系统时间及时区：

```
select systimestamp from dual;
```

   （3） 把字符串变为日期格式：to_date:

```
select to_date ('2019-12-26', 'YYYY-mm-dd') from dual;
```

   （4） 把日期转换为字符串形式：to_char:

```
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss') as nowTime from dual
select to_char(sysdate,'yyyy') as nowYear from dual;    //获取时间的年
select to_char(sysdate,'mm') as nowMonth from dual;    //获取时间的月
select to_char(sysdate,'dd')  as nowDay from dual;    //获取时间的日
select to_char(sysdate,'hh24')  as nowHour from dual;    //获取时间的时
```

```
select to_char(sysdate,'mi') as nowMinute from dual;     //获取时间的分
select to_char(sysdate,'ss')  as nowSecond from dual;    //获取时间的秒
```

2.10 简述 Select 中 Case when 和 if 语句用法

1. Case when 语句：相当于 if else 的条件判断语句

（1） 把字段放在 case 后面时，when 后面接的是字段的值

```
SELECT
    f_name,
    f_state,
    CASE f_state
WHEN 0 THEN
    '未完成'
ELSE
    '已完成'
END 状态
FROM
    dayexercise;
```

（2） Case 后面不放字段，when 后面接的是判断语句

```
SELECT
    f_name,
    f_state,
    CASE
WHEN f_state = 0 THEN
    '未完成'
ELSE
    '已完成'
END 状态
FROM
    dayexercise;
```

2. If 语句：相当于三元函数，不能使用 if else 语句

```
SELECT f_name, f_state,IF(f_state=0,'未完成','已完成') 状态 FROM dayexercise;
```

2.11 简述开窗函数的用法

MySQL8.0 中新增的窗口函数，在排序方式后面加上 over（partition by 分组字段 order by 排序字段 ），有三种排序方式：rank，dense_rank，row_number：

1. rank()：跳跃排序，例如：1，2，2，4 •••

```
SELECT
 f_group,
```

```
    f_name,
    f_score,
    rank()  over (
        PARTITION BY f_group
        ORDER BY
            f_score DESC
    ) rownum
FROM
    dayexercise;
```

2. dense_rank()：连续排序；例如：1，2，2，3

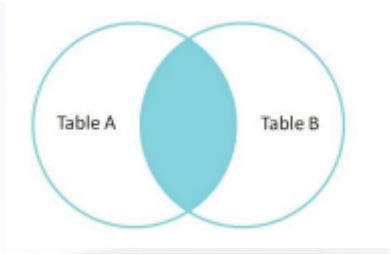
```
SELECT
    f_group,
    f_name,
    f_score,
    dense_rank()  over (
        PARTITION BY f_group
        ORDER BY
            f_score DESC
    ) rownum
FROM
    dayexercise;
```

3. row_number()：没有重复值的排序（记录相同也不重复）

```
SELECT
    f_group,
    f_name,
    f_score,
    row_number()  over (
        PARTITION BY f_group
        ORDER BY
            f_score DESC
    ) rownum
FROM
    dayexercise;
```

2.12 简述 Join 常用的用法有哪些及之间的区别

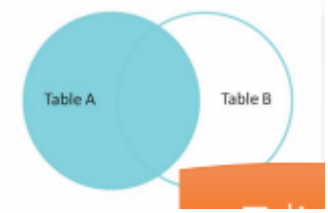
1. 内连接 inner join：也叫等值连接，inner join 产生符合 A 和 B 的一组数据，相当于 join




```
select * from stu m inner join dayexercise n on m.id=n.f_sid;
```

id	f_sid	f_name	f_group	f_date	f_state	f_remark	f_score	id(1)	sname	age	adress
1	1	byp	第四组	2019-12-19	1	(Null)	60	1	byp	19	安徽
2	2	lb	第 三 组	2019-12-20	1	(Null)	70	2	lb	19	安徽
5	4	fjj	第五组	(Null)	1	(Null)	90	4	fjj	20	浙江
6	7	mike	第一组	2019-12-28	1	(Null)	100	7	mike	21	河南
7	1	byp	第一组	2019-12-19	0	1	80	1	byp	19	安徽
11	2	lb	第四组	2019-12-17	1	(Null)	0	2	lb	19	安徽

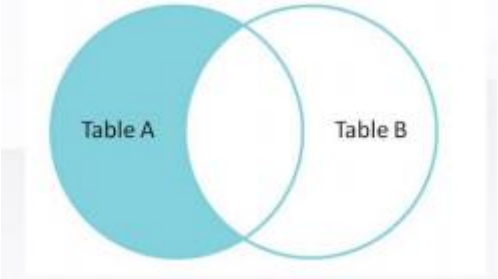
2. 左连接 left join：左连接从左表（A）产生一套完整的记录，与匹配的记录（右表（B）），如果没有匹配，右侧将包含 null




```
select * from stu s left join dayexercise d on s.sid=d.f_sid;
```

id	sname	age	adress	id(1)	f_sid	f_name	f_group	f_date	f_state	f_remark	f_score
1	byp	19	安徽	1	1	byp	第四组	2019-12-19	1	(Null)	60
1	byp	19	安徽	7	1	byp	第一组	2019-12-19	0	1	80
2	lb	19	安徽	2	2	lb	第 三 组	2019-12-20	1	(Null)	70
2	lb	19	安徽	11	2	lb	第四组	2019-12-17	1	(Null)	0
4	fjj	20	浙江	5	4	fjj	第五组	(Null)	1	(Null)	90
7	mike	21	河南	6	7	mike	第一组	2019-12-28	1	(Null)	100
5	kk	20	河南	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)
3	ll	19	北京	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)

左表独有：如果想只从左表（A）中产生一套记录，但不包含右表（B）的记录，可以通过设置 where 语句来执行




```
select * from stu s left join dayexercise d on s.sid=d.f_sid where d.f_sid is null;
```

id	sname	age	adress	id(1)	f_sid	f_name	f_group	f_date	f_state	f_remark	f_score
5	kk	20	河南	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)
3	ll	19	北京	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)

3. 右连接 right join：右连接从右表（A）产生一套完整的记录，与匹配的记录（左表（B）），如果没有匹配，左侧将包含 null

```
select * from stu s right join dayexercise d on s.sid=d.f_sid;
```

id	sname	age	adress	id(1)	f_sid	f_name	f_group	f_date	f_state	f_remark	f_score
1	byp	19	安徽	1	1	byp	第四组	2019-12-19	1	(Null)	60
2	lb	19	安徽	2	2	lb	第 三 组	2019-12-20	1	(Null)	70
4	fjj	20	浙江	5	4	fjj	第五组	(Null)	1	(Null)	90
7	mike	21	河南	6	7	mike	第一组	2019-12-28	1	(Null)	100
1	byp	19	安徽	7	1	byp	第一组	2019-12-19	0	1	80
(Null)	(Null)	(Null)	(Null)	8	9	sb	第四组	2019-12-26	0	1	88
(Null)	(Null)	(Null)	(Null)	9	10	fjj	第五组	2019-12-09	0	1	90
2	lb	19	安徽	11	2	lb	第四组	2019-12-17	1	(Null)	0

右表特有：

```
select * from stu s right join dayexercise d    on s.sid=d.f_sid where s.sid is null;
```

id	sname	age	adress	id(1)	f_sid	f_name	f_group	f_date	f_state	f_remark	f_score
(Null)	(Null)	(Null)	(Null)	8	9	sb	第四组	2019-12-26	0	1	88
(Null)	(Null)	(Null)	(Null)	9	10	fjj	第五组	2019-12-09	0	1	90

2.13 简述 Union 和 UnionAll 的区别

1. union：并集，并去除重复数据
2. unionall：罗列出表中的所有数据，可能有重复值

2.14 简述 SQL 子查询都有哪些及使用方式（6 类）

1. 带关键字 join 的子查询
2. 带关键字 all 的子查询
3. 带关键字 in 的 子查询：in 和 not in

```
//查看不足 1 人的部分名（子查询得到的所有人的部门 id）,使用 distinct 是为了效率
SELECT
 `name`
FROM
 department
WHERE
 id NOT IN ( SELECT DISTINCT ( dep_id ) FROM employee );
```

4. 带比较符的子查询：= ， != ， > ， <

```
//查询大于所有人的平均年龄的员工名与年龄
select `name`, age from employee where age > (
select avg(age) from employee);
```

5. 带关键字 exists 的子查询：在 exists 后面的括号中的值返回的是 true 或者 false；当是 true 时前面的语句就会被查询输出，如果是 false 则不会输出

```
//当岗位的 id=200 时，查询 emloyee 中数据
select * from employee where exists(
select * from department where id=200);
//当 employee 中的存在 department 中的岗位的 id 时，查询 employee 的数据
select * from employee e where exists (
select * from department d where e.dep_id=d.id);
```

6. 带关键字 any 的子查询：只要满足任意一条就输出

```
select * from employee e where e.dep_id > ANY (
select id from department);
```

2.15 简述 SQL 语句分析的过程

找关键字

1. From
   元素来自哪张表
2. Join
   表和表之间关联
3. On
   多表关联的连接条件
4. Where
   限定条件的逻辑操作符
5. Group by
   分组
6. Having
   搭配 group by 使用，相当于 where
7. Order by
   升序排序，降序使用 DESC
8. Limit
   用于分页

3 高级

3.1 简述 MySQL 中从请求到返回中间执行过程

客户端发送一条查询给服务器；服务器先检查查询缓存，如果命中了缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段。服务器段进行 SQL 解析、预处理，在优化器生成对应的执行计划；mysql 根据优化器生成的执行计划，调用存储引擎的 API 来执行查询。将结果返回给客户端。

3.2 简述 InnoDB、MyISAM、Memory 引擎之间的区别及使用场景

一：InnoDB 是一个事务型的存储引擎，支持回滚，设计目标是处理大数量数据时提供 高性能的服务，它在运行时会在内存中建立缓冲池，用于缓冲数据和索引。

优点： 1、支持事务处理、ACID 事务特性； 2、实现了 SQL 标准的四种隔离级别； 3、支持行级锁和外键约束； 4、可以利用事务日志进行数据恢复。 5、锁级别为行锁，行锁优点是适用于高并发的频繁表修改，高并发是性能优于 MyISAM。缺点是系统消耗较大。 6、索引不仅缓存自身，也缓存数据，相比 MyISAM 需要更大的内存。

缺点： 因为它没有保存表的行数，当使用 COUNT 统计时会扫描全表

InnoDB 适合场景： (1)可靠性要求比较高，或者要求事务； (2)表更新和查询都相当的频繁，并且表锁定的机会比较大的情况。

二：MyISAM 是 MySQL 5.5.5 之前的默认引擎，它的设计目标是快速读取。

优点： 1.高性能读取； 2.因为它保存了表的行数，当使用 COUNT 统计时不会扫描全表，只需要直接读 取已经保存好的值即可。

缺点： 1、锁级别为表锁，表锁优点是开销小，加锁快；缺点是锁粒度大，发生锁冲 动概率较高，容纳并发能力低，这个引擎适合查询为主的业务。 2、此引擎不支持事务，也不支持外键。 3、INSERT 和 UPDATE 操作需要锁定整个表；

MyISAM 适合场景： (1)做很多 count 的计算； (2)插入不频繁，查询非常频繁； (3)没有事务。

三：MEMORY：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择 MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

3.3 简述视图创建、删除、更新语句

一：创建视图

```
CREATE VIEW viewName as SELECT stu.sname,sc.num FROM student stu, score sc where stu.sid = sc.student_id;
```

二：更新视图

```
ALTER VIEW viewName as SELECT stu.sname,sc.num FROM student stu, score sc where stu.sid = sc.student_id;
```

三：删除视图

```
DROP VIEW viewName;
```

3.4 简述 MySQL 中索引分类

一：单例索引：一个索引只包含单个列，但一个表中可以有多个单列索引

（1） 普通索引：MySQL 中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点

（2） 唯一索引：索引列中的值必须是唯一的，但是允许为空值

（3） 主键索引：是一种特殊的唯一索引，不允许有空值

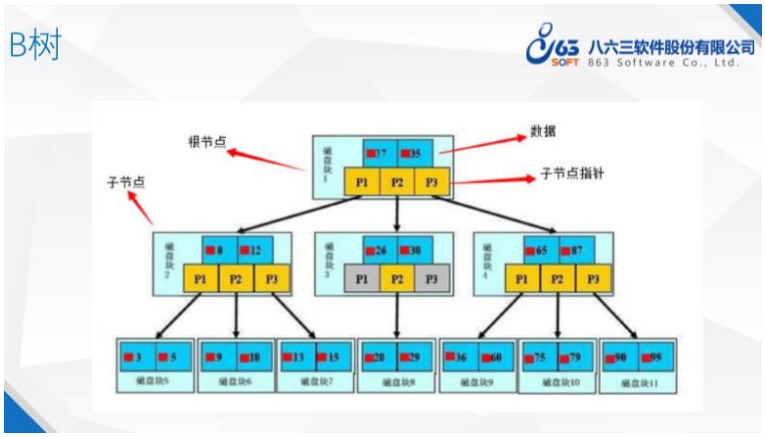
二：组合索引：在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合

三：全文索引：全文索引，只有在 MyISAM 引擎上才能使用，只能在 CHAR,VARCHAR,TEXT 类型字段上使用全文索引，介绍了要求，说说什么是全文索引，就是在一堆文字中，通过其中的某个关键字等，就能找到该字段所属的记录行

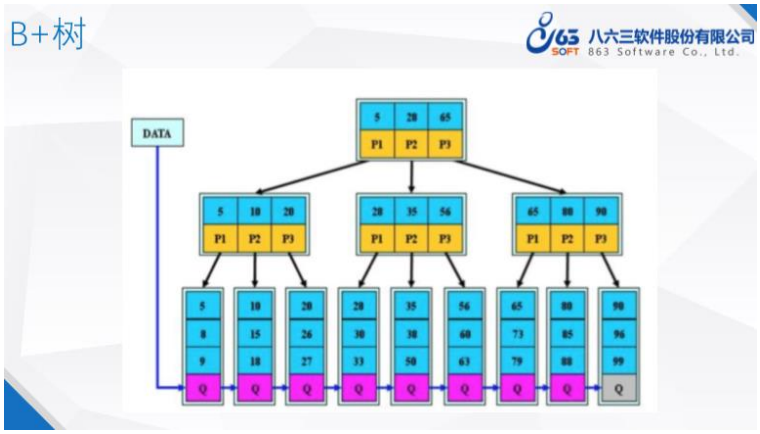
四：空间索引：空间索引是对空间数据类型的字段建立的索引，MySQL 中的空间数据类型有四种，GEOMETRY、POINT、LINESTRING、POLYGON。在创建空间索引时，使用 SPATIAL 关键字。要求，引擎为 MyISAM，创建空间索引的列，必须将其声明为 NOT NULL

3.5 简述 B 树和 B+树的特点及区别

一：B 树也称 B-树,它是一颗多路平衡查找树，一个 m 阶的 B 树满足以下条 件：1 每个结点至多拥有 m 棵子树。2 根结点至少拥有两颗子树（存在子树的情况下），根结点至少有一个关键码。3 除了根结点以外，其余每个分支结点至少拥有 m/2 棵子树。4 非叶子结点中保存的关键码个数，等于该节点子树个数-1，关键码按照递增 次序进行排列。5 所有叶子节点都位于同一层，或者说根节点到每个叶子节点的长度都相同。6 节点中每个关键字的左子树中的关键字都小于该关键字，右子树中的关键字 都大于该关键字。 7 关键码数量需要满足 ceil(m/2)-1 <= n <= m-1 8 每个节点都存有索引和数据，也就是对应的 key 和 value。



二：B+树其实和 B 树的变形树，差异在于： B+树有两种类型的节点：内部结点（也称索引结点）和叶子结点。内部节点 就是非叶子节点，内部节点不存储数据，只存储索引，所有的内部节点都可以看 成是索引部分，数据都存储在叶子节点。 2、有 n 棵子树的结点中含有 n 个关键码。 3、所有的叶子结点中包含了全部关键码的信息，及指向含有这些关键码记录的 指针，每个叶子结点都存有相邻叶子结点的指针，叶子结点本身依关键字的大小 自小而大顺序链接。 4、内部结点中含有其子树结点中最大（或最小）关键码。



三：B 树、B+树比较

B 树在查询单条数据是非常快的。但如果范围查的话，b 树每次都要从 根节点查询一遍。B+树相对于 B 树优点： 1、单一节点存储的元素更多，使得查询的 IO 次数更少，所以也就使得它更适合 做为数据库 MySQL 的底层数据结构。 2、所有的查询都要查找到叶子节点，查询性能是稳定的，而 B 树，每个节点都可 以查找到数据，所以不稳定。 3、所有的叶子节点形成了一个有序链表，更加便于范围查找。

3.6 简述什么是聚簇索引和非聚簇索引

一：聚簇索引

数据和索引存储到一起，找到索引就获取到了数据。聚簇索引是唯一的，InnoDB 一定会有一个聚簇索引来保存数据。非聚簇索引一定存储有聚簇索引的列值；InnoDB 使用的是聚簇索引

二：非聚簇索引

数据存储和索引分开，叶子节点存储对应的行，需要二次查找，通常称为[二级索引]或[辅助索引]；MyISM 使用的是非聚簇索引

三：聚集索引与非聚集索引的区别?

1、聚集索引（一般为主键）数据行的物理顺序和列值逻辑顺序相同，一个表只能拥有一个聚集索引，可以有多个非聚集索引； 2、使用非聚集索引查询，没有完整的数据还得进行二次查询，会影响查询性能； 3、聚集索引查询效率更高，但由于要移动对应数据的物理位置，写入性能并不高

四：为什么在主键上创建聚集索引比创建非聚集索引要慢?

由于主键的唯一性约束，在插入数据时要保证不能重复。聚集索引由于索引叶节点就是数据页，要检查数据唯一性就得遍历所有数据节点。而非聚集索引上已经包含了主键值，所以查找主键唯一性就只需要遍历所有索引页就可以了。

3.7 简述创建、删除索引语句

一：创建索引

```
CREATE INDEX idx_book_name ON book(name);
ALTER TABLE book ADD INDEX idx_book_name(name);
```

二：删除索引

格式一：ALTER TABLE 表名 DROP INDEX 索引名
ALTER TABLE book DROP INDEX idx_book_name;
格式二：DROP INDEX 索引名 ON 表名
DROP INDEX idx_book_name ON book;

3.8 简述查看执行计划语句

explain select * from book where studentid>2;
结果显示内容解释：
id:SELECT 识别符。
select_type:表示为简单的 SELECT，不使用 UNION 或子查询
table：数据表的名字。他们按被读取的先后顺序排列
type：可能的取值有 system、const、eq_ref、index 和 All
possible_keys：MySQL 在搜索数据记录时可以选用的各个索引
key：实际选用的索引
key_len:显示了 mysql 使用索引的长度(也就是使用的索引个数),当 key 字段的值为 null 时，索引的长度就是 null
ref:给出关联关系中另一个数据表中数据列的名字
rows：MySQL 在执行这个查询时预计会从这个数据表里读出的数据行的个数。
extra：提供了与关联操作有关的信息，没有则什么都不写

查看创建创建表语句：show create table book;

3.9 简述触发器创建语法及创建实例

语法： drop trigger if exists databaseName.tri_Name;
create trigger tri_Name -- tri_Name 代表触发器名称 tirgger_time trigger_event on tableName -- tirgger_time 为触发时机，可选值有 after/before，trigger_event 为触发事件，可选值有 insert/update/delete for each row -- 这句话在 mysql 是固定的，表示任何一条记录上的操作满足触发事件 都会触发该触发器。 begin sql 语句; end

创建插入触发器

```
CREATE TRIGGER `tri_insert_user` AFTER INSERT ON `tbsys_user` FOR EACH ROW
begin  INSERT INTO tbsys_log(userid, opertype, operdate) VALUES (new.id, 'add a user '+new.username, now());
end;
```

创建更新触发器

```
CREATE TRIGGER `tri_update_user` AFTER UPDATE ON `tbsys_user` FOR EACH ROW
begin  INSERT INTO tbsys_log(userid, opertype, operdate) VALUES (new.id, 'update
a user', now());
end;
```

创建删除触发器

```
CREATE TRIGGER `tri_delete_user` AFTER DELETE ON `tbsys_user` FOR EACH ROW
begin  INSERT INTO tbsys_log(userid, opertype, operdate) VALUES (old.id, 'delete
a user', now());
end;
```

查看触发器

```
show triggers from bigdata274;
```

3.10 简述数据库优化原则

1、硬件优化
2、数据库设计(分库、数据库选型)
    分库，不同的库可用不同的服务器，比如一个系统涵盖了订单\物流\仓储等
    数据库选型：如是否用关系型数据库，是否用空间数据库等
3、分表
4、分区
    把一个大的数据文件拆分为多个小文件，存到磁盘的不同目录下
5、空间换时间设计模型（字段重复设计）
6、物化视图（临时表）
    物化视图主要用于远程数据访问，需要占用磁盘空间
7、索引
8、字段设计（表层次）
    例如：
    被索引列必须定义为 not null,并设置 default 值
    禁止使用 float、double 类型，建议使用 decimal 替代
9、SQL 语句优化
10、算法设计
11、内存表的优化（内存数据库）

3.11 简述 SQL 优化原则

1、字段类型转换导致不用索引，
    如字符串类型的不用引号,数字类型的用引号等,这有可能会用不到索引导致全表扫描;
（EXPLAIN select * from book where name =1）
2、like % 在前面用不到索引;

（EXPLAIN select * from book where name like '%西'）
3、不要使用 select *， 输写 SQL 带字段，以防止后面表变更带来的问题，性能也是比较优的 ；
4、排序请尽量使用升序 ；
5、in 的子查询尽量用 union 代替；
6、复合索引高选择性的字段排在前面,并且使用索引的第一列；
7、where 条件中，过滤量最大的条件放在 where 子句最后；
8、删除表所有记录请用 truncate，不要用 delete；
9、尽量把字段设置为 NOT NULL
10、读取适当的记录 LIMIT M,N
11、尽量批量 Insert Into 代替单个 Insert
12、在索引列上使用计算、改变索引列的类型、在索引列上使用!=将放弃索引（select id from t where num/2=100）

3.12 简述什么是慢查询，怎么进行设置 MySQL 的慢查询

慢查询：将超过指定时间的 SQL 语句查询称为“慢查询”
设置 MySQL 的慢查询：
slow_query_log 慢查询开启状态
slow_query_log_file 慢查询日志存放的位置（这个目录需要 MySQL 的运行帐号的可写权限，一般设置为 MySQL 的数据存放目录）
slow_launch_time 超过多少秒算是慢查询
long_query_time 超过多少秒的查询就写入日志

```
命令演示：
show variables like 'slow%';
set global slow_query_log='ON';
set global slow_query_log_file='C:\\mysql.log';
set global long_query_time =2;
验证：关闭当前设置窗口，重新打开一个 sql 界面，查询设置
select sleep(3);
```

3.13 简述生成 OracleAWR 性能分析报告的过程

1. oracle 的快照生成完成之后，进行生成报告
执行以下命令（注意地址）：

```
@c:\oracle\product\10.2.0\db_1\RDBMS\ADMIN\awrrpt.sql
```

2. 进入到了窗口之后，默认回车为 html 的操作，进行输入 html
3. 进入到了输入 num_days 的值中，输入 1 为 1 天的数据，为空为所有的数据。
4. 在输入 begin_snap 的值的，为开始的是 snap id 的值
5. 然后进行输入 end_snap 值的快照结束的 snap_id 的值。
6. 进入到了下一步之后，进行输入 report_name 值的中，输入报告的名称即可。

3.14 简述什么是存储过程及优缺点

存储过程（ Stored Procedure）是一种在数据库中存储复杂程序，以便外部程序调用的一种数据库对象。思想上很简单，就是数据库 SQL 语言层面的代码封装与重用。
优点：存储过程可以封装，并隐藏复杂的业务逻辑；可以回传值，并可以接受参数。
缺点：存储过程，往往定制化于特定的数据库上，因为支持的编程语言不同。当切换到其他厂商的数据库系统时，需要重写原有的存储过程。

3.15 简述存储过程创建语法

```
Create procedure 存储过程名字([in|out|inout] 参数名 参数类型)
Begin
  执行的代码
End;
```

例如 无参创建:

```
CREATE PROCEDURE avg_age()
Begin
SELECT AVG(age) 平均年龄 FROM employee;
End
```

有参创建：

```
CREATE PROCEDURE avgageBysex(in strSex VARCHAR(10),OUT avgage DECIMAL(8,2))
Begin
SELECT avg(age) into avgage from employee WHERE sex = strSex;
End;
```

3.16 简述存储过程使用的关键字及使用实例

（1）set @变量名=值

此处的变量不需要声明， mysql 会自动根据值类型来确定类型，这种变量要在变量名称前面加上“@” 符号，叫做会话变量，代表整个会话过程他都是有作用的，有点类似于全局变量一样。

例如：

```
Set @aveage=200;
Select @avgage
```

（2）declare 变量名 变量类型 default 默认值;
declare 定义的变量 相当于一个局部变量 在 end 之后失效，而且 declare 只能在 begin，end 中定义。
例如：

```
CREATE PROCEDURE patchInsert()
Begin
  Declare i int;
  Set i = 1;
  While i <10 do
     INSERT into tb_user VALUES(i,CONCAT('user',i),'pw123');
     Set   i = i + 1;
  End while;
End;
```

（3）select 使用

Select @变量名

例如：

```
Set @aveage=200;
Select @avgage;
```

3.17 简述怎么调用存储过程

执行存储过程: call 存储过程名字

例如：

```
create PROCEDURE avgageBysex(in strSex VARCHAR(10),OUT avgage
DECIMAL(8,2))
BEGIN
select avg(age) INTO avgage from employee where sex = strSex;
END
call avgageBysex('male',@avgSex);
select @avgSex;
```

3.18 简述存储过程中 if\while\casewhen 使用方法

If 语句：

```
create procedure calcBySex(in sexType varchar(20),out result double)
begin
if sexType='male'
then
select max(age) into result from employee ;
elseif sexType='female'
then
select min(age) into result from employee;
else
select sum(age) into result from employee;
```

```
end if;
end;

CALL calcBySex('male',@res);
SELECT @res;
```

While 语句：

```
create procedure patchInsert()
begin
declare i int;
set i=1;
while i<10
do
insert into tb_user values (i,concat('user',i),'pw');
set i=i+1;
end while;
end;

call patchInsert();
```

casewhen 语句：

```
CREATE PROCEDURE GetCustomerShipping(in p_customerNumber int(11), out p_shiping varchar(50))
BEGIN
 DECLARE customerCountry varchar(50);
SELECT country INTO customerCountry FROM customers
WHERE customerNumber = p_customerNumber;
 CASE customerCountry
 WHEN  'USA' THEN
     SET p_shiping = '2-day Shipping';
 WHEN 'Canada' THEN
     SET p_shiping = '3-day Shipping';
 ELSE
     SET p_shiping = '5-day Shipping';
 END CASE;
End;
```

3.19 简述 MySQL 中怎么查看已有的存储过程

查看库里面的存储过程：

```
Show procedure status;
```

查看存储过程创建语句：

```
Show create procedure 存储过程名字
```

例如：

```
SHOW CREATE PROCEDURE patchInsert;
```

3.20 简述怎么删除存储过程

删除存储过程：

```
Drop procedure  存储过程名字
```

例如：

```
DROP PROCEDURE patchInsert;
```

3.21 简述存储过程和函数的区别

（1） 函数和存储过程统称为存储例程,函数的限制比较多,而存储过程的限制较少,存储过程的实现功能要复杂些,而函数的实现功能针对性比较强。

（2）返回值不同：

函数必须有返回值,且仅返回一个结果值；

存储过程可以没有返回值,也能返回结果集(out,inout)

（3）调用时的不同：

函数嵌入在 SQL 中使用,用 select 调用： select 函数名(变量值)；

存储过程通过 call 语句调用： call 存储过程名；

（4）参数的不同：

函数的参数类型类似于 IN 参数；

存储过程的参数类型有三种：

1）in 数据只是从外部传入内部使用(值传递),可以是数值也可以是变量；

2）out 只允许过程内部使用(不用外部数据)，并将结果返回给外部,只能是变量；

3） inout 外部和内部都可以使用,内部修改的也可以给外部使用,典型的引 传递,只能传递变量。

3.22 简述创建函数语法

语法：

```
Create function  函数名([参数列表]) returns  数据类型
Begin
 Sql 语句;
 Return  值;
End;
参数列表的格式是：变量名  数据类型
```

注意：Mysql8 默认开启 binlog，需要设置成如下参数才可以创建函数：

```
set global log_bin_trust_function_creators=TRUE;
```

无参创建：

```
create function fun2() returns int
begin
declare res int;
SELECT id from employee where name = 'egon' into res;
return res;
end;

select fun2();
```

有参创建：

```
create function fun3(ename varchar(15)) returns int
begin
declare res int;
select id from employee e where e.`name`=ename into res;
return res;
end;

select fun3('egon');
```

3.23 简述怎么调用函数

用 select 调用：select 函数名

```
select fun2();

select fun3('egon');
```

3.24 简述怎么查看所有函数

查看所有函数:

```
Show function status;
```

查看函数创建过程：

```
Show create function  函数名;
```

例如：

```
show create FUNCTION fun2;
```

3.25 简述怎么删除函数

```
Drop function  函数名;
```

例如：

```
DROP FUNCTION fun2;
```

3.26 简述 Oracle 创建序列的语句

创建序列需要 CREATE SEQUENCE 系统权限。序列的创建语法如下：

```
CREATE SEQUENCE  序列名
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE/ MINVALUE n| NOMAXVALUE}]
[{CYCLE|NOCYCLE}]
[{CACHE n| NOCACHE}];
```

1) INCREMENT BY 用于定义序列的步长，如果省略，则默认为 1，如果出现负值，则代表 Oracle 序列的值是按照此步长递减的。

2) START WITH 定义序列的初始值(即产生的第一个值)，默认为 1。

3) MAXVALUE 定义序列生成器能产生的最大值。选项 NOMAXVALUE 是默认选项，代表没有最大值定义，这时对于递增 Oracle 序列，系统能够产生的最大值是 10 的 27 次方;对于递减序列，最大值是-1。

4) MINVALUE 定义序列生成器能产生的最小值。选项 NOMAXVALUE 是默认选项，代表没有最小值定义，这时对于递减序列，系统能够产生的最小值是?10 的 26 次方;对于递增序列，最小值是 1。

5) CYCLE 和 NOCYCLE 表示当序列生成器的值达到限制值后是否循环。 CYCLE 代表循环，NOCYCLE 代表不循环。如果循环，则当递增序列达到最大值时，循环到最小值;对于递减序列达到最小值时，循环到最大值。如果不循环，达到限制值后，继续产生新值就会发生错误。

6) CACHE(缓冲)定义存放序列的大小，默认为 20。 NOCACHE 表示不对序列进行内存缓冲。对序列进行内存缓冲，可以改善序列的性能。

7) NEXTVAL 返回序列中下一个有效的值，任何用户都可以引用。

8) CURRVAL 中存放序列的当前值。

创建：

```
create sequence seq0;
create sequence seq1 increment by 1 start with 1;
create sequence seq2 increment by 2 start with 1 maxvalue 10 cycle
cache 5;
```

修改：

```
ALTER SEQUENCE seq1 INCREMENT BY 10;
```

删除：

```
drop sequence seq1;
```

查询：

```
select seq2.nextval from dual;
select seq1.currval from dual;
```

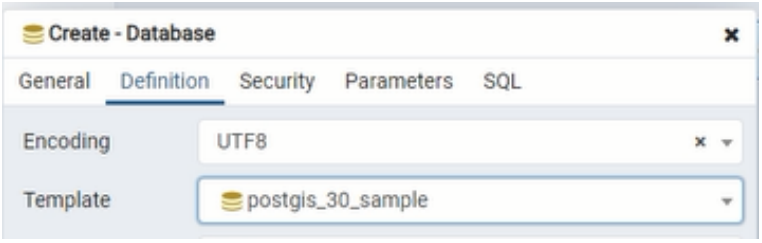
3.27 简述 MySQL 中实现序列器的方式

一般使用序列(Sequence)来处理主键字段，在 **MySQL 中是没有序列**的，但是 MySQL 有提供了自增长(increment)来实现类似的目的，但也只是自增，而不能设置步长、开始索引、是否循环等，最重要的是**一张表只能由一个字段使用自增**，但有的时候我们需要两个或两个以上的字段实现自增（单表多字段自增），MySQL 本身是实现不了的，但我们**可以用创建一个序列表，使用函数来获取序列的值**

```
1、常见序列维护表
CREATE TABLE seq (
name varchar(20) NOT NULL,
val int(10) UNSIGNED NOT NULL,
PRIMARY KEY (name)
)
2、创建函数
CREATE FUNCTION seq(seq_name char (20)) returns int
begin
UPDATE seq SET val=last_insert_id(val+1) WHERE name=seq_name;
RETURN last_insert_id();
end
3、插入数据
INSERT INTO seq VALUES('tb_user',10);
INSERT INTO seq VALUES('tb_right',100);
INSERT INTO seq VALUES('tb_sysinfo',1000);
4、使用序列
SELECT seq('tb_user'),seq('tb_right'),seq('tb_user'),seq('tb_right');
```

3.28 简述 Postgre 中导入空间数据的过程

1. 连接数据库。用 PostGIS 连接 Postgres，并选择要导入的空间数据库（创建空间数据库时，注意选择 Template）



2. 选择要导入的数据。操作 PostGIS，Database->DB Manager

在弹出的窗口中选择 PostGIS，再点击 Import Layer
在新窗口中，从当前 Layer 中选择或者从外部导入文件
3. 设置表名。输入 Table 名，其他选项默认，点击 ok。

3.29 简述 PostGIS 中空间相交查询语句

```
判断两个 geometry 是否相交
ST_Intersects(geometry, geometry)
例如：
Select * from person as zz
where st_intersects (zz.geom,
ST_GeomFromText('point(113.611576 34.748286)',4326))
```

3.30 简述 PostGIS 中空间几何转文本的函数

```
ST_AsText(geometry)
例如：
(0,0 0,1 1,1 1,0 0,0)
转换后应该是这样的结果 POLYGON(0 0,0 1,1 1,1 0,0 0)
```

3.31 简述 PostGIS 中空间几何缓冲函数

```
ST_Buffer(geometry, double, [integer])
查询在该 city 中，位于某区域内的数据
select *  from city as zz
where st_intersects(zz.geom, ST_Buffer(ST_GeomFromText('POINT(113.611576
34.748286)', 4326), 10))
```

3.32 简述事务的四个特征

原子性（Atomicity）：一个事务（ transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（ Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
一致性 （ Consistency）：事务执行后，数据库状态与其他业务规则保持一致。如转账业务，无论事务执行成功否，参与转账的两个账号余额之和应该是不变的。在事务开始之前和事务结束以后，数据库的完整性没有被破坏。
隔离性（ Isolation）：事务独立运行。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（ Read uncommitted）、读提交（ read committed）、可重复读（ repeatable

read）和串行化（ Serializable）
持久性（ Durability）：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

3.33 简述事务中的控制语句

BEGIN 或 START TRANSACTION 显式地开启一个事务；
COMMIT:提交事务，并使已对数据库进行的所有修改成为永久性的；
ROLLBACK:回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；
SET TRANSACTION 用来设置事务的隔离级别。 InnoDB 存储引擎提供事务的隔离级别有 READ、UNCOMMITTED、 READ COMMITTED、 REPEATABLE READ 和 SERIALIZABLE。
SET AUTOCOMMIT 用来设置是否自动提交， 0 禁止自动提交， 1 开启自动提交

3.34 简述 MySQL 中事务处理的两种方式

1、用 BEGIN, ROLLBACK, COMMIT 来实现
BEGIN 开始一个事务
ROLLBACK 事务回滚
COMMIT 事务确认
2、直接用 SET 来改变 MySQL 的自动提交模式:
SET AUTOCOMMIT=0 禁止自动提交
SET AUTOCOMMIT=1 开启自动提交
提示：在 MySQL 命令行的默认设置下，事务都是自动提交的，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT=0，用来禁止使用当前会话的自动提交。
实例：

```
begin; #开启事务
insert tb_user (username,password) values ('hadoop','hdp'); #插入数据，未
提交到库中
commit; #提交到数据库
start transaction; #开启事务
insert tb_user (username,password) values ('hadoop','hdp'); #插入数据，未
提交到库中
update tb_user set password = ‘root’ where id = 1;#更新密码
rollback; #回滚数据
insert tb_user (username,password) values ('hadoop','hdp'); #插入数据，默
认自动提交
set autocommit = 0; #禁止自动提交
insert tb_user (username,password) values ('hadoop','hdp'); #插入数据，未
提交到库中
set autocommit = 1; #开启自动提交
```

3.35 简述并发事务引起的五类问题

Lost Update **更新丢失**

a. 第一类更新丢失，**回滚覆盖**：撤消一个事务时， 在该事务内的写操作要回滚，把其它已提交的事务写入的数据覆盖了。

b. 第二类更新丢失，**提交覆盖**：提交一个事务时， 写操作依赖于事务内读到的数据，读发生在其他事务提交前，写发生在其他事务提交后，把其他已提交的事务写入的数据覆盖了。这是不可重复读的特例。

Dirty Read **脏读**：一个事务读到了另一个未提交的事务写的数据。

Non-Repeatable Read **不可重复读**：一个事务中两次读同一行数据，可是这两次读到的数据不一样。

Phantom Read **幻读**：一个事务中两次查询，但第二次查询比第一次查询多了或少了几行或几列数据。

3.36 简述事务的隔离级别

Read Uncommitted **读未提交**：不允许第一类更新丢失。允许脏读，不隔离事务。

Read Committed **读已提交**：不允许脏读，允许不可重复读。

Repeatable Read **可重复读**：不允许不可重复读。但可能出现幻读。

Serializable **串行化**：所有的增删改查串行执行。

3.37 简述 JDBC 中事务处理的方式

Connection 的三个方法与事务有关：

setAutoCommit（ boolean） :设置是否为自动提交事务，如果 true（默认值为 true）表示自动提交，也就是每条执行的 SQL 语句都是一个单独的事务，如果设置为 false，那么相当于开启了事务了；

con.setAutoCommit(false) 表示开启事务。

commit（）：提交结束事务。

rollback（）：回滚结束事务。

3.38 简述 JDBC 中常用的接口和类

DriverManager ：依据数据库的不同，管理 JDBC 驱动

Connection ：负责连接数据库并担任传送数据的任务

Statement ：由 Connection 产生、负责执行 SQL 语句

ResultSet：负责保存 Statement 执行后所产生的查询结果

3.39 简述 JDBC 进行增删改查的过程

JDBC 执行增删改

1、获取连接
2、获取 Statement 对象
Connection.createStatement()
3、整理插入的 sql 语句字符串
String sql="sql 语句"
4、发送并执行 sql 语句
Statement.executeUpdate(sql 语句);
5、关闭连接
JDBC 执行查询操作
1、获取连接
2、获取 Statement 对象
Connection.createStatement()
3、整理查询的 sql 语句字符串
String sql="sql 语句"
4、发送并执行 sql 语句
ResultSet rs = Statement.executeQuery(sql 语句);
5、处理结果集
6、关闭连接

3.40 简述 MySQL 和 Oracle 数据库备份恢复的命令

MySQL
备份：cmd 下执行
语法：mysqldump -u root -p 数据库 [表名 1 表名 2..] > 文件路径

```
mysqldump -u root -p bigdata274 >E:\\dbbackup.sql
```

恢复：

```
mysql -u root -p bigdata2741 < E:\\dbbackup.sql
```

Oracle
备份：cmd 窗口中执行

```
exp root/root@192.168.13.51:1521/db11g file=c:\\dbbackup.dmp
```

恢复：cmd 下执行

```
imp root/root@192.168.13.51:1521/db11g file=c:\\dbbackup.dmp full=y ignore=y
```

4 MongoDB

4.1 简述 Mongodb 库、集合查看、创建、删除命令

创建命令 use 数据库名
- 删除命令 db.dropDatabase()

- 查看所有数据库：show dbs
- 查看当前数据库：db
- 查看库所有的集合：show collections 或 show tables
- 创建集合有两种方式，显示创建和隐式创建

显示创建可以使用命令 db.createCollection("soft863")

隐式创建可以使用命令 db.集合名称.insert({}),指创建集合并同时向集合中插 入数据,例如：db.student.insert({name:"lisi"})

4.2 简述文档插入、更新、删除命令

- 插入：db.student.insert({name:"hadoop",age:18})
- 删除：语法 db.集合名称.remove({删除条件})， 不加删除条件为删除集合中的所有文档，例如，db.student.remove() 为删除 student 集合中的所有文档 db.student.remove({name:”user1”})为删除 student 集合中 name 为 lisi 的文档
- 更改：更新集合中的文档,将集合中 name 为 zhangsan 的文档改成 name 为 lisi db.soft.update({age:3},{$set:{age:7}})

4.3 简述 MongoDB 查询语法、关键字都有哪些及使用方法

- 查询语法 db.集合名称.find({条件})，或者 db.集合名称.findOne() 查询第一 个文档
- 返回除了 age 之外的所有字段：db.student.find({},{age:0})
- 返回除了条件是 name=“hadoop”并且 age 之外的所有字段：db.student.find({name:”hadoop”},{age:0})
- 返回除了条件是 name=“hadoop”的 age 字段： db.student.find({name:”hadoop”},{age:1})
- 注意：_id 字段都会返回

MongoDB 文档操作-模糊查询

- 查询 title 包含"教"字的文档： db.col.find({title:/教/})
- 查询 title 字段以"教"字开头的文档： db.col.find({title:/^教/})
- 查询 titl e 字段以"教"字结尾的文档： db.col.find({title:/教$/})

MongoDB 查询（条件表达式）

- $gt -------- greater than >
- $gte --------- gt equal >=
- $lt -------- less than <
- $lte --------- lt equal <=
- $ne ---------- not equal !=
- $eq -------- equal =

MongoDB 查询（条件表达式）

```
• //大于： field > value db.collection.find({field:{$gt:value}});
• //小于： field < value db.collection.find({field:{$lt:value}});
• //大于等于： field >= value db.collection.find({field:{$gte:value}});
• //小于等于： field <= value db.collection.find({field:{$lte:value}});
• //不等于： field != value db.collection.find({field:{$ne:value}});
```

```
• //大于 1，小于 100 db. collection.find({age : {$lt :100, $gt : 1}})
```

MongoDB 查询（统计、排序、分页）

• 查询集合中的文档 ，统计(count)、排序(sort)、分页(skip、limit)

```
• db.customer.count();
• db.customer.find().count();
• db.customer.find({age:{$lt:5}}).count();
• db.customer.find().sort({age:1}); 降序-1
• db.customer.find().skip(2).limit(3);
• db.customer.find().sort({age:-1}).skip(2).limit(3);
• db.customer.find().sort({age:-1}).skip(2).limit(3).count();
• db.customer.find().sort({age:-1}).skip(2).limit(3).count(0);
• db.customer.find().sort({age:-1}).skip(2).limit(3).count(1);
```

MongoDB 查询（包含$all）

• 查询集合中的文档 ,$all 主要用来查询数组中的包含关系，查询条件中只要 有一个不包含就不返回

```
• db.student.insert({array:[1,2,3,4]});
• db.student.insert({array:[1]});
• db.student.find({array:{$all:[1,2]}})
```

MongoDB 查询（包含$in 和$nin）

• 查询集合中的文档 ,$in，类似于关系型数据库中的 IN • 查询集合中的文档 ,$nin，与$in 相反

```
• db.student.find({age:{$in:[10,20]}})
• db.student.find({age:{$nin:[20]}})
```

MongoDB 查询（$or 和$nor）

• 查询集合中的文档 ,$or，相当于关系型数据库中的 OR，表示或者的关系， 例如查询 name 为 hadoop 或者 age 为 18 的文档，命令为： db.student.find({$or:[{name:”hadoop”,{age:18}}]})

• 查询集合中的文档 ,$nor，表示根据条件过滤掉某些数据，例如查询 name 不是 hadoop，age 不是 18 的文档，命令为： db.student.find({$nor:[{name:”hadoop”,{age:18}}]})

MongoDB 查询（$exist）

• 查询集合中的文档 ,$exists，用于查询集合中存在某个键的文档或不存在某个键的文档，例如查询 student 集合中存在 name 键的所有文档，可以使用 db.student.find({name:{$exists:1}})，

• $exists:1 表示真，指存在

• $exists:0 表示假，指不存在

4.4 简述 MongoDB 数据备份恢复命令

MongoDB 备份（ mongodump ）

• MongoDB 提供了备份和恢复的功能，分别是 MongoDB 下载目录下的 mongodump.exe 和 mongorestore.exe 文件

• 备份数据使用下面的命令： >mongodump -h dbhost -d dbname -o dbdirectory

-h：MongDB 所在服务器地址，例如：127.0.0.1，当然也可以指定端口号： 127.0.0.1:27017

-d： 需要备份的数据库实例， 例如：test

-o：备份的数据存放位置，例如：e:\datas，当然该目录需要提前建立，在 备份完成后，系统自动在 dump 目录下建立一个 test 目录，这个目录里面存 放该数据库实例的备份数据。

MongoDB 恢复（ mongorestore ）

• 恢复数据使用下面的命令： >mongorestore -h dbhost -d dbname dbdirectory

-h：MongoDB 所在服务器地址

-d：需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候 的不一样，比如 test2 directory：备份数据所在位置，例如：e:\datas\soft86

实例：mongorestore -h 127.0.0.1 -d testsoft E:\\Datas\\soft863

4.5 简述 MongoDB 集合导入导出命令

MongoDB 导出

• 导出数据可以使用命令： mongoexport -h dbhost -d dbname -c collectionName -o output

• 参数说明:

-h 数据库地址

-d 指明使用的库

-c 指明要导出的集合

-o 指明要导出的文件名

实例：mongoexport -h 127.0.0.1 -d soft863 -c soft -o E:\\Datas\\soft863.bson

MongoDB 导入

• 导入数据可以使用命令： mongoimport -h dbhost -d dbname -c collectionname 文件的地址...

• 参数说明:

-h 数据库地址

-d 指明使用的库

-c 指明要导入的集合 本地的文件地址...

实例：mongoimport -h 127.0.0.1 -d testsoft -c test E:\\Datas\\soft863.bson

4.6 简述 MongDB 索引创建方式

- 创建普通索引，使用命令 db.soft.ensureIndex({age:1})
- 创建唯一索引，使用命令 db.soft.ensureIndex({id:1},{unique:true})
- 查看关于索引的相关信息，使用命令 db.soft.stats()
- 查看查询使用索引的情况，使用命令 db.soft.find({age:10}).explain()
- 删除索引，使用命令 db.soft.dropIndex({age:1})
- 删除集合，也会将集合中的索引全部删除

4.7 简述 MongoDB 编程中连接数据库的过程

数据库连接：

```
public class MongoDBUtil {
        public static MongoDatabase getConnect(){              // 连接到 mongodb 服务
        MongoClient mongoClient = new MongoClient("localhost", 27017);              //
连接到数据库
        MongoDatabase mongoDatabase = mongoClient.getDatabase("soft863");
// 返回连接数据库对象
        return mongoDatabase;
         }
        }
```

4.8 简述 MongDB 编程中数据增删改查过程

插入

```
// 获取集合
MongoCollection<Document> collection = MongoDBUtil. getConnect
().getCollection("soft");
// 创建文档
Document document = new Document() .append("name", "lisi")
.append("sex", "男")
.append("age", 18);
// 插入
collection.insertOne(document);
// 批量插入
ArrayList<Document> docs = new ArrayList<>();
for (int i = 0; i < 10; i++) {
Document subDocument = new Document() .append("name", "lisi" + i)
.append("sex", "男")
.append("age", 1 + i);
docs.add(subDocument);
}
collection.insertMany(docs)
```

删除

```
// 删除文档
Bson filter = Filters. eq ("age", 2);
// 删除与筛选器匹配的单个文档
collection.deleteOne(filter);
// 删除所有
collection.deleteMany(filter);
```

修改

```
// 指定修改的更新文档
document = new Document("$set", new Document("name", "hadoop"));
collection.updateOne(filter, document);
collection.updateMany(filter, document);
```

查询

```
// 查询所有
FindIterable findIterable0 = collection.find();
MongoCursor cursor0 = findIterable0.iterator();
while (cursor0.hasNext()) {
Document doc = (Document) cursor0.next();
System. out .println(doc.get("name"));
}
// 通过条过滤
FindIterable findIterable = collection.find(filter);
MongoCursor cursor = findIterable.iterator();
Document doc1 = (Document) findIterable.first();
// 获取第一个
while (cursor.hasNext()) {
Document doc = (Document) cursor.next();
System. out .println(doc); }
```

5 Elastic

5.1 简述什么是倒序索引

全文检索是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。

5.2 简述什么是中文分词

分词就是将 连续的字序列 按照一定的规范 重新组合成词序列 的过程。

5.3 简述 ES 特点

1、规模不限，可以作为一个大型分布式集群技术，处理 PB 级数据；也可以运行在单机上。

2、没有什么新技术，主要是将全文检索、数据分析以及分布式技术，合并在了一起。

3、对用户而言操作简单。

4、功能太少。不能同义词搜索，复杂的数据分析，海量数据的实时处理。

5.4 简述 ES 中物理架构都有哪几部分构成

1.1 Cluster（集群） 集群包含多个节点，每个节点属于哪个集群是通过一个配置（集群名称，默认是 elasticsearch）来决定的，对于中小型应用来说，刚开始一个集群就一个节点很正常

2.1 Node（节点） 集群中的一个节点，节点也有一个名称（默认是随机分配的），节点名称很重要(在执行运维管理操作的时候)，默认节点会去加入一个名称为“elasticsearch”的集群，如果直接启动一堆节点，那么它们会自动组成一个 elasticsearch 集群，当然一个节点也可以组成一个 elasticsearch 集群。

3.1 分片 当我们的文档量很大时，由于内存和硬盘的限制，同时也为了提高 ES 的处理能力、容错能力及高可用能力，我们将索引分成若干分片，每个分片可以放在不同的服务器，这样就实现了多个服务器共同对外提供索引及搜索服务。一个搜索请求过来，会分别从各各分片去查询，最后将查询到的数据合并返回给用户

4.1 副本 为了提高 ES 的高可用同时也为了提高搜索的吞吐量，我们将分片复制一份或多份存储在其它的服务器，这样即使当前的服务器挂掉了，拥有副本的服务器照常可以提供服务。

5.1 主结点 一个集群中会有一个或多个主结点，主结点的作用是集群管理，比如增加节点，移除节点等，主结点挂掉后 ES 会重新选一个主结点。

6.1 结点转发 每个结点都知道其它结点的信息，我们可以对任意一个结点发起请求，接收请求的结点会转发给其它结点查询数据。

5.5 简述 ES 和关系型数据库模型对比

关系型数据库（ 比如 Mysql）	非关系型数据库（ Elasticsearch）
数据库 Database	索引 Index
表 Table	类型 Type
数据行 Row	文档 Document
数据列 Column	字段 Field
约束 Schema	映射 Mapping

5.6 简述 ES 安装过程

1、 解压
2、 创建用户(创建除了 root 权限以外的用户)
3、 关闭防火墙
4、 创建日志文件夹

5、 修改软件解压路径下的 config/elasticsearch.yml 文件。主要是 network 以及 http.port
6、 修改配置文件 jvm.options。主要是-Xms、-Xmx
7、 将文件授权给创建的用户
8、 修改 limits.conf 文件，添加

```
root soft nofile 100001
root hard nofile 100002
hadoop soft nofile 65536
hadoop hard nofile 65536
hadoop soft nproc 4096
hadoop hard nproc 4096
```

9、 编辑 /etc/sysctl.conf 在末尾加上 vm.max_map_count = 655360

5.7 简述 ES 中使用 CURL 常用的监控语句有哪些

1、 查看集群健康状态：curl -XGET http://localhost:9200/_indices/health?pretty
2、 查看所有分片状态：curl -XGET http://localhost:9200/_cat
3、 查询所有索引库信息 curl http://192.168.9.163:9200/_search?pretty

5.8 简述 ES 中使用 CURL 怎么创建和删除索引

1、 创建索引：
curl -XPUT http://192.168.9.163:9200/bigdata
2、 删除索引：
curl -XDELETE http://192.168.9.163:9200/customer

5.9 简述 ES 中使用 CURL 怎么进行增删改查数据

```
增加、更新：
curl -XPOST http://192.168.9.163:9200/bigdata/product/
     -H "Content-Type: application/json" -d'{"author" : "Doug Cutting"}'
curl -XPUT http://192.168.9.163:9200/bigdata/product/4
     -H "Content-Type: application/json" -d'{"author" : "Doug Cutting"}'
查询：
curl http://192.168.9.163:9200/bigdata/product/_search
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?q=id:"2“
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?size=2&from=0
删除：
curl -XDELETE http://master:9200/bigdata/product/4/
```

5.10 简述 Kibana 中怎么对数据进行增删改查

1、 创建索引 `PUT /bigdata`
2、 删除索引，及其数据 `DELETE /bigdata  DELETE /bigdata/_doc/1`
3、 插入更新数据

```
PUT /bigdata/product/2
{ "name":"lisi", "sex":"man"}

POST /bigdata1/product/2/_update
{ "doc":{ "name":"wangwu" }}
```

4、 查询

```
GET /_cat/indices
GET /_cat/health?v
GET /bigdata/product/4
GET /bigdata/_doc/4?pretty
```

5.11 简述 Kibana 中你熟悉的模块

1、 Index Management 索引模块
2、 Dev Tools 执行命令界面
3、 Discover 数据搜索界面
4、 Visualize 视图界面
5、 Dashboard 仪表盘
6、 Canvas 汇报模块

6 Neo4j

6.1 简述 Neo4j 中节点创建方式

创建一个节点

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

- CREATE 创建数据的子句
- () 圆括号表示一个节点
- ee:Person 变量 'ee' 和标签 'Person' 代表新的节点
- {} 花括号添加属性到节点

创建很多节点

```
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" }),
       (ir:Person { name: "Ian", from: "England", title: "author" }),
```

(animal:Animal { name: "小白",age:2 }),

6.2 简述 Neo4j 数据增删改查的语句

查找代表 email 的节点：MATCH (ee:Person) WHERE ee.name = "Emil" RETURN ee;

- MATCH 指定节点和关系的模式的子句
- (ee:Person) 带有标签 “Person” 的单节点模式，将匹配项赋给变量 “ee”
- WHERE 约束结果的子句
- ee.name = "Emil" 比较 name 属性与 "Emil" 值
- RETURN 用于请求特定结果的子句

删除所有节点：MATCH (n:Person) remove n:Person

清空数据库：match(n) detach delete n

添加关系：

不同标签：match(p:Person{name:”lan”}),(a:Animal{name:”小白”})create(p)-[:饲养{since:2020}]->(a)

同一标签：match(s:student{age:12}),(m:student{age:22})create(s)-[:认识{since:2010}]->(m)

更新节点（如果不存在，则添加）：MATCH (s:student) where s.id=1 set s.age=1

删除节点：MATCH (s:student) where s.age=12 detach delete s 或

MATCH (s:student{age:12}) detach delete s

删除单个关系：match(s:student)-[re:认识{since:2010}]->(m:student) where s.age=12 and m.age=22 delete re

模式匹配（描述在图中要查找的内容）：

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = "Emil" RETURN ee, friends
```

- MATCH 用于描述从已知节点到找到的节点的模式的子句
- (ee)模式以 Person 开始（由 WHERE 限定）
- -[:KNOWS]-匹配 "KNOWS" 关系（任意方向）
- (friends)将绑定为 Emil 的朋友

推荐（可用来做推荐）：

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

- ()空的圆括号表示忽略这些节点
- DISTINCT 因为不止一条路径将与模式匹配
- surfer 将包含 Allison，一个朋友的朋友，且冲浪

分析（了解查询工作原理，可在查询前加 explain 或 profile）：

```
PROFILE MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

创建更多节点和关系：

```
MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" }),
```

(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby: "surfing" }),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]->(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally), (rvb)-[:KNOWS]->(ally)

7 NoSQL

7.1 简述关系型数据库和非关系型数据库区别

关系型数据库：具有完善的事务机制和高效的查询机制
二维表格 1、Mysql 和 Oracle 数据库，互联网运维最常用的是 MySQL
2、通过 SQL 结构化查询语句存储数据
3、保持数据一致性方面很强二维表格
非关系型数据库（NoSql）：web2.0 时代出现
1、NoSQL 不是否定关系数据库，而是作为关系数据库的 一个重要补充
2、NOSQL 为了高性能、高并发而生，忽略影响高性能， 高并发的功能
3、NOSQL 典型产品 memcached （纯内存）， redis（持久化缓存）， mongodb（文档的数据库）

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	RDBMS有关系代数理论作为基础 NoSQL没有统一的理论基础
数据规模	大	超大	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限，性能会随着数据规模的增大而降低 NoSQL可以很容易通过添加更多设备来支持更大规模的数据
数据库模式	固定	灵活	RDBMS需要定义数据库模式，严格遵守数据定义和相关的束条件 NoSQL不存在数据库模式，可以自由灵活定义并存储各种不同类型的数据
查询效率	快	可以实现高效的简单查询，但是不具备高度结构化查询等特性，复杂查询的性能不尽人意	RDBMS借助于索引机制可以实现快速查询（包括记录查询和范围查询） NoSQL没有索引，虽然NoSQL可以使用MapReduce来加速查询，但是，在复杂查询方面的性能仍然不如RDBMS

一致性	强一致性	弱一致性	RDBMS严格遵守事务ACID模型，可以保证事务强一致性 NoSQL放松对事务ACID四性的要求，而是遵守BASE模型，只能保证最终一致性
数据完整性	容易实现	很难实现	任何一个RDBMS都可以很容易实现数据完整性，比如通过主键或者非空约束来实现实体完整性，通过主键、外键来实现参照完整性，通过约束或者触发器来实现用户自定义完整性 但是，在NoSQL数据库却无法实现
扩展性	一般	好	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限 NoSQL在设计之初就充分考虑了横向扩展的需求，可以很容易通过添加廉价设备实现扩展
可用性	好	很好	RDBMS在任何时候都以保证数据一致性为优先目标，其次才是优化系统性能，随着数据规模的增大，RDBMS为了保证严格的一致性，只能提供相对较弱的可用性 NoSQL任何时候都能提供较高的可用性
标准化	是	否	RDBMS已经标准化（SQL） NoSQL还没有行业标准，不同的NoSQL数据库都有自己的查询语言，很难规范应用程序接口
技术支持	高	低	RDBMS经过几十年的发展，已经非常成熟，Oracle等大型厂商都可以提供很好的技术支持 NoSQL在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持
可维护性	复杂	复杂	RDBMS需要专门的数据库管理员(DBA)维护 NoSQL数据库虽然没有DBMS复杂，也难以维护

7.2 简述 NoSQL 数据库类型有哪些

键值数据库　代表：redis，SimpleDB
列族数据库　代表：BigDate，hadoopDB
文档数据库　代表：MongoDB，CouchDB
图形数据库　代表：Neo4j，GraphDB

7.3 简述什么是 CAP

C（consistency 一致性）：所有节点在同一时间具有相同的数据
A（availability 可用性）：是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应
P（tolerance of network partition 分区容忍性）：是指出现网络分区的情况时（即系统中一部分节点无法和其它节点进行通信），分离的系统也能正常运行，即系统中任意信息的丢失或失败不会影响系统的继续运作

7.4 简述什么是 Base

ACID	BASE
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

基本可用：是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用， 也就是允许分区失败的情形出现
软状态：与""硬状态“相对应的一种提法。数据库保存的数据是硬状态时，可以保证数据一致性，即保证数据一直是正确的。软状态指状态可以有一段时间不同步，具有一定的滞后性
最终一致性（弱一致性的一种特例）：允许后续的访问操作可以暂时读不到更新后的数据，但经过一段时间后，必须最终读到更新后的数据
一致性强一致性和弱一致性，二者主要区别高并发的数据访问操作下，后续操作是否能获取最新的数据。对于强一致性而言，当执行完一次更新操作后，后续的其他读操作可以保证读到更新后的最新数据；反之，如果不能保证后续访问读到的数据都是更新后的最新数据就是弱一致性。

8 redis

8.1 简述 Redis 有什么特点和优势

1、异常快速：Redis 的速度非常快，每秒能执行约 11 万集合，每秒约 81000+条记录。
2、支持丰富的数据类型：String（字符串）、List（列表）、set（集合）、Sort Set（有序集合）、Hash（散列数据）。
3、操作都是原子性：所有 Redis 操作是原子的，这保证了如果两个客户端同时访问的 Redis 服务器将获得更新后的值。
4、多功能实用工具：Redis 是一个多实用的工具，例如缓存，消息，队列使用(Redis 原生支持发布/订阅)，任何短暂的数据，应用程序，如 Web 应用程序会话，网页命中计数等。

8.2 简述 Redis 持久化的两种方式的区别及配置方式

RDB 持久化（原理是将 Reids 在内存中的数据库记录定时 dump 到磁盘上的 RDB 持久化）
参数配置：
save 900 1:
在 900 秒(15 分钟)之后，如果至少有 1 个 key 发生变化，则 dump 内存快照。
save 300 10:

在 300 秒(5 分钟)之后，如果至少有 10 个 key 发生变化，则 dump 内存快照。

save 60 10000;

在 60 秒(1 分钟)之后，如果至少有 10000 个 key 发生变化，则 dump 内存快照。

AOF（append only file）持久化（原理是将 Reids 的操作日志以追加的方式写入文件）

参数配置：

appendfsync always

每次有数据修改发生时都会写入 AOF 文件。

appendfsync everysec

每秒钟同步一次，该策略为 AOF 的缺省策略。

appendfsync no

从不同步。高效但是数据不会被持久化。

8.3 简述 Redis 支持常见的数据类型有哪些

KEY:字符串的形式

关于 key 的使用注意：

1.key 不要太长，尽量不要超过 1024 字节，这不仅消耗内存，而且会降低查找的效率；

2.key 也不要太短，太短的话，key 的可读性会降低；

3.在一个项目中，key 最好使用统一的命名模式，例如 user:10000:passwd

常见的 value 支持五种数据类型:

1.字符串（strings）

2.列表（lists）

3.集合（sets）

4.有序集合（sorted sets）

5.哈希（hash）

8.4 简述 Redis 中常见的数据类型中常用的命令有哪些

1、字符串（strings）

set(key, value)：给数据库中名称为 key 的 string 赋予值 value（会覆盖原来 key 相同的）

eg: set k0 "jack"

mset(key N, value N)：批量设置多个 string 的值（会覆盖原来 key 相同的）

eg: mset k0 "jack" k1 "mike"

msetnx(key N, value N)：如果所有名称为 key i 的 string 都不存在（只要有 key 存在，返回 0）

eg: msetnx k0 "jack1"

msetnx k2 "merry"

get(key)：返回数据库中名称为 key 的 string 的 value

eg: get k0

mget(key1, key2,..., key N)：返回库中多个 string 的 value

eg: mget k0 k1 k2

incr(key)：名称为 key 的 string 增 1 操作(对数值有效)

eg: set k3 3
incr k3

incrby(key, integer)：名称为 key 的 string 增加 integer

eg: incrby k3 2

decr(key)：名称为 key 的 string 减 1 操作

eg: decr k3 1

decrby(key, integer)：名称为 key 的 string 减少 integer

eg: decrby k3 2

append(key, value)：名称为 key 的 string 的值附加 value(追加)

eg: append k0 1

substr(key, start, end)：返回名称为 key 的 string 的 value 的子串(下标索引从 0 开始)

eg: substr k0 1 3

2、列表（lists）

1、rpush(key, value)：在名称为 key 的 list 尾添加一个值为 value 的元素(列表右边：rpush)

2、lpush(key, value)：在名称为 key 的 list 头添加一个值为 value 的元素(列表左边：lpush)

3、llen(key)：返回名称为 key 的 list 的长度

4、lrange(key, start, end)：返回名称为 key 的 list 中 start 至 end 之间的元素(下标从 0 开始，包含 start、end 的元素)

5、lindex(key, index)：返回名称为 key 的 list 中 index 位置的元素

6、lset(key, index, value)：给名称为 key 的 list 中 index 位置的元素赋值

7、lrem(key, count, value)：删除 count 个 key 的 list 中值为 value 的元素

8、lpop(key)：返回并删除名称为 key 的 list 中的首元素

9、rpop(key)：返回并删除名称为 key 的 list 中的尾元素

3、集合（sets）

1、sadd(key, member)：向名称为 key 的 set 中添加元素 member

2、smembers(key) ：返回名称为 key 的 set 的所有元素

3、scard(key) ：返回名称为 key 的 set 的个数

4、sismember(key, member) ：member 是否是名称为 key 的 set 的元素

5、srandmember(key) ：随机返回名称为 key 的 set 的一个元素

6、srem(key, member) ：删除名称为 key 的 set 中的元素 member

7、spop(key) ：随机返回并删除名称为 key 的 set 中一个元素

8、sinter(key1, key2,...key N) ：求交集

9、sinterstore(dstkey, (keys)) ：求交集并将交集保存到 dstkey 的集合

10、sunion(key1, (keys)) ：求并集

11、sunionstore(dstkey, (keys)) ：求并集并将并集保存到 dstkey 的集合

12、sdiff(key1, (keys)) ：求差集

13、sdiffstore(dstkey, (keys)) ：求差集并将差集保存到 dstkey 的集合

4、有序集合（sorted sets）

1、zadd key score member //zadd 有序集合顺序编号元素值

2、zcardkey //计算成员个数

3、zscorekey member //计算某个成员的分数

4、zrankkey member //计算成员排名，从低到高

5、zrevrankkey member //计算成员排名，从高到低

6、zremkey member [member...] //删除成员

7、zincrbykey increnmentmember //增加成员的分数

8、zrangekey start end [withscores] //从低到高返回指定排名的分数

9、zrevrangekey start end [withscores] //从高到低返回

10、zrangebyscorekey min max [withscores] [limit offset count] //按照分数从低到高返回

11、withscore 代表返回的时候带上成员的分数

5、哈希（hash）

1、hset(key, field, value)：向名称为 key 的 hash 中添加元素 field

2、hget(key, field)：返回名称为 key 的 hash 中 field 对应的 value

3、hmget(key, (fields))：返回名称为 key 的 hash 中 field i 对应的 value

4、hmset(key, (fields))：向名称为 key 的 hash 中添加元素 field

5、hlen(key)：返回名称为 key 的 hash 中元素个数

6、hkeys(key)：返回名称为 key 的 hash 中所有键

7、hvals(key)：返回名称为 key 的 hash 中所有键对应的 value

8、hgetall(key)：返回名称为 key 的 hash 中所有的键（field）及其对应的 value

9、hincrby(key, field, integer)：将名称为 key 的 hash 中 field 的 value 增加 integer

10、hexists(key, field)：名称为 key 的 hash 中是否存在键为 field 的域

11、hdel(key, field)：删除名称为 key 的 hash 中键为 field 的域

8.5 简述 Redis 设置选区某个库的命令

1、 在命令行选取
Select key index
2、 在 Java 中：
Jedis jedis = new Jedis("192.168.13.66", 6378);
Jedis.select(int index);

8.6 简述 Redis 中针对于键的操作命令有哪些

1、 exists(key)：确认一个 key 是否存在

2、 del(key)：删除一个 key

3、 type(key)：返回值的类型

4、 keys(pattern)：返回满足给定 pattern(k*)的所有 key

5、randomkey：随机返回 key 空间的一个

6、rename(oldname, newname)：重命名 key

7、dbsize：返回当前数据库中 key 的数目

8、expire：设定一个 key 的活动时间（s）

9、ttl：获得一个 key 的活动时间

8.7 简述 Redis 清库的命令

1、flushdb：删除当前选择数据库中的所有 key

2、flushall：删除所有数据库中的所有 key

8.8 简述什么是发布订阅

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

8.9 简述 Redis 中发布和订阅的命令有哪些

1、PSUBSCRIBE pattern [pattern ...] ：订阅一个或多个符合给定模式的频道。

2、PUBSUB subcommand [argument [argument ...]] ：查看订阅与发布系统状态。

3、PUBLISH channel message ：将信息发送到指定的频道。

4、PUNSUBSCRIBE [pattern [pattern ...]] ：退订所有给定模式的频道。

1、 SUBSCRIBE channel [channel ...] ：订阅给定的一个或多个频道的信息。

2、 UNSUBSCRIBE [channel [channel ...]] ：指退订给定的频道。

Eg：
订阅频道：subscribe soft863_1
订阅所有 soft863 的频道：psubscribesoft863*
发布消息：publish soft863_1 begin learn
查看活跃状态：PUBSUB CHANNELS

8.10 简述 Redis 怎么进行主从架构部署

Redis 的单机部署:

1、 将 Redis 拷贝 3 份，一个主节点，端口为 6378。两个从节点端口为 6380 和 6381
2、 修改主节点的 redis.windows.conf 文件，找到 port 端口号，将其修改为 6378
3、 修改两份从节点的 redis.windows.conf 文件，找到端口号，分别修改成 6380、6381
4、 修改两份从节点 slaveof，将其前面#号去掉，并且去除前面空格，
slaveof 127.0.0.16378（ip 为主节点的 ip 地址）

8.11 简述 Redis 哨兵作用及部署方式

1、 哨兵的作用

1、监控（Monitoring）：Sentinel 会不断地检查你的主服务器和从服务器是否允许正常。

2、提醒（Notification）：当被监控的某个 Redis 服务器出现问题时，Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

3、自动故障迁移（Automatic failover）：

（1）当一个主服务器不能正常工作时，Sentinel 会开始一次自动故障迁移操作，他会将失效主服务器的其中一个从服务器升级为新的主服务器，并让失效主服务器的其他从服务器改为复制新的主服务器；

（2）客户端试图连接失败的主服务器时，集群也会向客服端返回新主服务器的地址，是的集群可以使用新主服务器代替失效服务器。

2、部署方式

1、在每个 redis 文件目录下创建 sentinel.conf 配置文件，内容如下（端口号修改）：

```
port 26478
sentinel monitor master 127.0.0.1 6378 1
sentinel down-after-milliseconds master 5000
sentinel config-epoch master 17
sentinel leader-epoch master 17
sentinel current-epoch 17
protected-mode yes
bind 192.168.13.66
```

2、当前 Sentinel 服务运行的端口（port 26378）

3、设置哨兵监听的主服务器，这个主服务器判断为失效至少需要 2 个 Sentinel 同意（只要同意 Sentinel 的数量不达标，自动故障迁移就不会执行）

sentinel monitor master 127.0.0.1 6378 2

4、设置 3s 内 master 无响应，则认为 master 宕机了

sentinel down-after-milliseconds master 3000

3、 运行

1、 启动：在每个 redis 目录下创建 startRedisSentinel.bat 文件，内容如下：

redis-server.exe sentinel.conf--sentinel

2、启动哨兵与 redis 服务器（运行.bat 文件）

4、检测

关闭 master 节点，查看从节点的信息（在客户端执行 info replication）