# 数据库

**JDBC操作步骤**

加载数据库驱动类, 打开数据库连接, 执行sql语句,处理返回结构,关闭资源

Mysql分页

Offset 偏移量

Limit: 截取几条记录

**什么是连接池，有什么好处**

连接池：统一管理数据库连接，重复使用一个连接，避免频繁创建/删除连接

**数据源(链接池)技术有哪些，有什么好处**

c3p0数据源等

好处：维护数据库信息，只需修改配置文件(数据库地址，密码，最大链接等配置)，无需更改代码就能变更数据库信息

**模糊查询**

%(匹配多个字符)

\_(匹配单个字符)

**外键约束**

值只能是别的表的主键。维护测试数据不方便，关联性太强

**sql注入**

select \* from user\_t where username = 'test';

当输入恶意用户名 test'; drop table user\_info; #

select \* from user\_info where user\_name = 'test'; drop table user\_info; #';

#表示注释, 造成user\_info表被删除

**SELECT ... FOR UPDATE**

在事务中锁定所选行，防止其他事务进行修改。

**什么表时候使用自增id主键**

无需手动指定id主键，生成速度快，占用空间小，大多数情况下都能用自增id

如果表要合并时，避免主键冲突，不适合用自增id

**InnoDB为什么用自增列作为主键**

2.数据存在主索引(B+树)叶子节点上，同个叶子节点内(一存页/磁盘页)的数据按主键顺序存放

3.使用自增主键，会按顺序添加，一页满后开辟新页

4.使用非自增主键，会将新记录添加在中间位置(无序)，需要移动数据，分页等操作，会造成大量碎片

**为什么索引能提高效率**

索引是有序的，索引查询数据无需遍历记录，其效率为log2(n)

**什么时候不建或少建索引**

数据太少

经常增删改

存在大量重复值(例：性别)

where中用不到的字段

**什么是分表和分区**

分表：表拆分成多张不同的表(不同业务)

分区：表拆分成多个更小的分区表（仍是一张表，底层由多个物理分区组成）

优：分区表存在不同物理设备上，存的数据更多

where中增加分区选择条件，可以只扫描部分分区表，count()在分区上进行处理，然后汇总

分区表更易维护

缺：一个表最多1024分区

分区表无法使用外键约束

表分区和索引分区要同步

**mysql中length()和char\_length()的区别**

length:返回字节个数（utf8编码：一个汉字3个字节，gbk编码:一个汉字2个字节）

char\_length:返回字符个数（汉字，数字，字母都是一个字符）

**mysql预处理语句**

prepared 语句：提高执行速度，避免SQL 注入

set @sql = 'select name from person where id=?'; 用？占位符替代输入的条件

向mysql 发送要执行的查询模板，单独发送数据，类似mybatis 预编译

**两个字段单独建立索引，如何同时对两个字段进行条件查询**

建立两个字段的复合索引

只走一条索引，区分度大的字段放在前面

**表中数据量大时，即使有索引也很慢**

读写分离

分库：根据业务模块分库，财务/部门，不同模块有不同的库

分表：垂直分：大表分成多个表（常用列表和不常用列的表）；水平分：根据id为单的一张表，为双的一张表

**读写分离导致数据不一致**

写库同步到读库需要一定的时间，存在部分延迟

**高并发下，多个请求对同一条数据更新，导致数据多次刷新，该怎么解决**

用消息队列，将请求放队列中，先进先出，逐个完成更新。高并发下队列内存会撑爆

用悲观锁，只能一个线程更新，其他线程必须等待。高并发下，多线程等待唤醒，浪费服务器资源

用乐观锁，每条数据添加版本号字段，更新前获得记录的版本号，符合版本号才能更新，更新成功后版本号+1，否则就循环重试（CAS操作），减少线程阻塞，推荐使用

**百万级数据如何进行搜素匹配，模糊查询**

两个半模糊查询（只在右边加%）

新建字段，存字段的倒序，like reverse('%str')(相当于 like 'rst%')，模糊查询两个字段并用union联合

**订单表分库分表**

对uid取余(模)分库分表

库名称：n = uid后四位 % 32, 名称为 order\_db\_n

表名称：n = (uid后四位 / 32) % 32 名称为 order\_tb\_n

**内连接，左连接，右连接，全连接查询的区别**

inner join(内链接)： 主表，从表的交集

left join(左连接)： 主表中数据全部显示，从表没的字段为null(一般都是左连接）

right join(右连接)： 从表中数据全部显示，主表中没的显示为null

full join(全连接)： 主表和从表数据全部显示，没的显示为null

**数据库3大范式**

第一范式：表中字段不可再分（原子性）

第二范式：表中每列都依赖于主键

第三范式：表中每列不能依赖于非主属性(反例age通过birthday字段确定)

**索引类型**

主键索引：列为主键，数据唯一,不能为null，只能有一个主键索引

普通索引：只有一个字段

复合索引：有多个字段

唯一索引：数据唯一，可以为null，可以有多个唯一索引

**回表查询是什么**

第一次根据索引去找数据的主键，索引中包含select的字段就返回，没有再根据主键去找数据(回表)

**覆盖索引**

查询的字段都在索引中，无需进行回表

**聚簇索引和非聚簇索引**

聚簇索引：按索引顺序存储，一个表只有一个，范围查询效率高，插入开销大

非聚簇索引：独里于数据存储，可以有多个，范围查询效率低，插入开销小，需要额外空间

**联合索引**

多个字段组成的索引，遵循最左匹配原则，左边不能丢，中间不能少

和两个单独索引不同

**联合索引(a,b), where a>5 and b = 5 索引会失效吗**

会， a范围查询，索引失效

**联合索引（a，b，c），查询（a，c），能用到索引吗**

会，a用到了索引，b丢失，c无法使用索引

**联合索引(a, b, c) , a = 3 and b > 4 会走索引吗**

a走索引，b范围查询不走索引

**查询每个部门薪水最高的人(包括并列)**

Select t.\* from person\_t t

Left join

(select dept\_code, max(salary) from person\_t group by dept\_code) t2

On t.dept\_code = t2.dept\_code and t.salary = t2.salary

**mysql表中新增字段**

ALTER TABLE users ADD COLUMN phone VARCHAR(20);

**查看索引的命令**

SHOW INDEX FROM table\_name

**索引为什么不是越多越好**

修改数据时，都要对索引进行修改，会降低插入/修改时的速度，也会占用更大的空间

**group by的字段需要加索引**

索引就是排序好的，加了索引就不用重新排序

**mysql优化，索引优化**

能用数值就不要用字符，数值只比较一次，字符逐个比较

varchar替换char,节省空间

不要select \* ，count(\*),覆盖索引(查询的字段都在索引中，无需回表)

先where过滤，再having过滤

只查一条数据，用limit1，避免全表扫描

Where和order by用到的字段要建索引

批量插入不要单挑插入

union,distinct 去重效率低，用java去重

A数据量>B数据量，用A in(B)； 反之用 exists

去除不必要的索引

设计冗余字段

**小表驱动大表**：2次链接，每次百万查询；比百万次链接，每次2次查询要好，链接很花时间

A in B 如for B for A， B数据量要小

A exist B 如 for A for B, A数据量要小

**In的优化**

select \* from t1 where id in (select id from t2 where sex='0');

In的范围是子查询，可以改为join，可以走索引

select \* from t1 left join t2 on t1.id = t2.id and t2.sex='0';

In范围中的值很多，可以创建临时表，然后通过join的方式来优化

in中数量越多，性能越慢(小于100个)，改成多次查询，通过union/多线程进行合并

In的字段上加上索引

用exists替代in，走索引

Select \* from t1 where exists (select 1 from t2 where t2.id = t1.id and t2.sex='0')

**not exists的优化**

Select \* from t1 where not exists(select1 from t2 where t1.id = t2.id)

加索引,避免大表上用not exists

not in 替换not exists(子查询不能包含null值)

Select \* from t1 where t1.id not in(select t1.id = t2.id);

使用join和is null 替代exists

Select \* from t1 left join t2 on t1.id = t2.id where t2.id is null;

**千万数据实现分页查询优化**

根据上次查询的最大主键id, where过滤掉之前的数据(不能用order by排序，会让主键乱序)

select \* from t where id > 1000 order by id desc limit 10;

**索引失效场景及替换方式**

<,>, <>, !=，is null ,is not null 做判断，用默认值替换

in , not in 做判断，连续数值，能用between就不要用in，exists替代in

or 改为union all

字段运算/函数：num/2=100 应该改为 num=100\*2

隐式类型转换，自动转为左边类型：where 字符串类型 = 数值类型, 要给数字加引号

%like

复合索引未遵循最佳左前缀法则

**B+树一般为几层**

一般1-3层，高度为3的B+树可以存放：1170\*1170\*16=21902400行数据（千万级别）

在查找数据时一次页的查找代表一次IO，通过[主键](https://so.csdn.net/so/search?q=%E4%B8%BB%E9%94%AE&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/Soldier49Zed/article/details/_blank)索引查询只需要1-3次逻辑IO操作即可查找到数据。

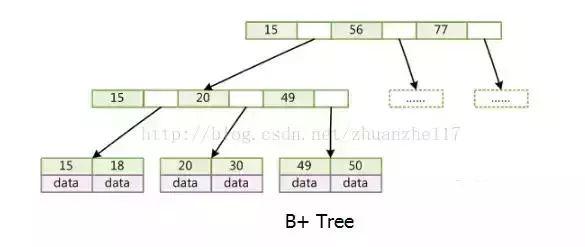
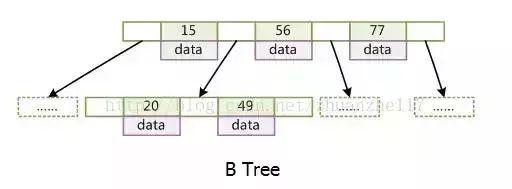
**B树和B+树的区别，为什么innodb索引用B+树而不用B树,只有叶子节点存数据的好处**

B树：每个节点存储关键字和记录指针,叶子节点指针为null

B+树：每个叶子节点存关键字和记录指针，叶子节点按顺序连接

B+树磁盘读写代价更小：内部结点并没有指向关键字具体信息的指针，其内部结点相对B 树更小,一次能读入的更多，IO读写次数就降低了

B+树的查询效率更稳定：查询关键字都要从根节点到叶子节点，路径长度相同，树比较矮，所以效率稳定



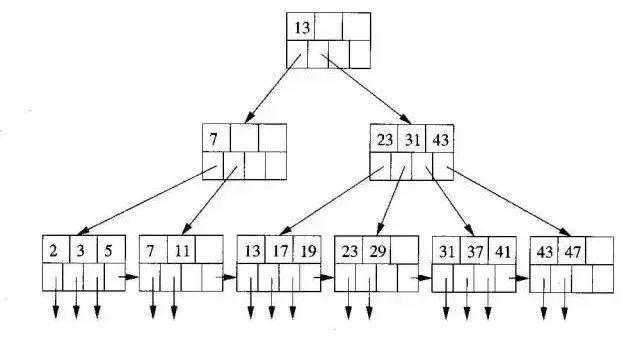
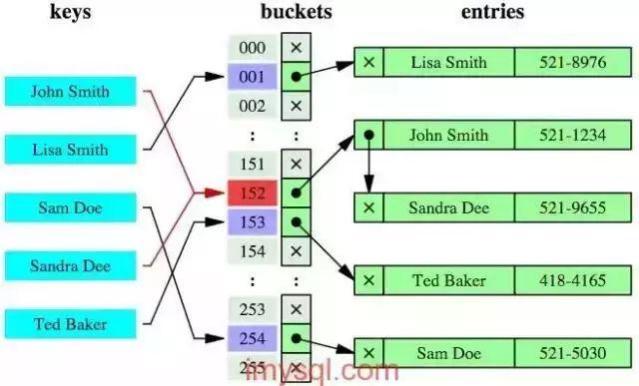
**B+树索引和哈希索引的区别**

B+树：平衡的多叉树，根节点到叶子节点的高度差不超过1，同层级的点间有指针相互链接(有序)

优：范围查询，用索引完成排序，最左前缀匹配原则

**哈希索引：用哈希算法，把键值换算成新的哈希值(无序)**

优：等值查询前提(没有大量重复键值,存在哈希碰撞,效率低)

**mysql索引为什么用B+树而不用跳表**

1.B+树的节点可以存储很多关键字，而且单个节点大小可以与磁盘页对其（默认16KB），一次IO就能传输一整个节点（跳表可能会出现跨页IO）,大幅减少磁盘IO次数

2.跳表的索引层建立具有随机性，磁盘不能对链表进行预读，会产生大量随机IO（磁盘预读：从本次被预读的位置开始，顺序向后读取一定长度的数据放入内存）

3.B+树检索效率比跳表高

**B+树最后一层的链表是单向/双向**

双向：既能向右遍历，也能向左遍历，再查询范围时比较好用

**cahr和varchar的区别**

Char:定长，最长255个字符，查询效率更高(数据)，适合存放短的，定长的数据

Varchar:动态长度，最长65535个字节，更加节省空间，结构(长度+数据)

**为什么Char比varchar查找更快**

varchar存长度+结构，mysql先读len，再往后读len个字符,进行查询，比较消耗性能

**DDL和DML**

DDL(Data Definition Language)数据定义语言:create,alter,drop, truncate,comment,grant

DML(Data Manipulation Language)数据操作语言：insert,delete,merge,select,update

**having和where的区别**

where分组前过滤

having分组后过滤，可以用聚合函数

**union和union all区别**

Union:去重合并查询结果

Union all:合并查询结果，不去重，效率高

**mysql的日志**

二进制日志（bin log）:记录更改数据的语句（重要）

重做日志（redo log）:记录事务里对数据的修改（重要）

回滚日志（undo log）:事务提交前，记录更新前的数据，当事务回滚时可用来回滚（重要）

慢查询日志（show log）：记录所有执行时间超过long\_query\_time秒的所有查询或不使用索引的查询（重要）

错误日志（error log）：记录mysql启动，运行，停止中遇到的问题

通用查询日志（general log）：记录建立的客户端连接和执行的语句

中继日志(relay log)：记录从复制主服务器接受的数据更改

DDL日志：记录元数据操作有DDL语句执行

**如何查询慢sql**

Slow\_query\_log:是否开启慢查询日志(ON表示开启)

Log\_output:日志输出位置

Slow\_query\_log\_file: 指定慢查询日志文件的路径和名字

Long\_query\_time：慢查询时间阈值

show variables where variable\_name in ('slow\_query\_log\_file', 'long\_query\_time', 'slow\_query\_log', 'log\_output');

set global slow\_query\_log = on

select \* from mysql.slow\_log

**不用%XXX怎么进行左模糊匹配(如手机号后四位)**

增加冗余列，存手机号的倒叙字符串 将%7590 改为0957%的右模糊

**建立索引的原则**

where和 order by会用到的字段上建索引

多字段联合索引要遵循最左匹配原则，中间字段不能少，频繁查询的字段往前靠

索引多会降低insert和update的效率，一般索超过6个

大量重复数据的字段（例：sex字段)不建索引

需要频繁更新的字段不建索引

**为什么联合索引中频繁查询的字段要往前靠**

要遵循最左匹配原则，左边字段会导致索引失效，频繁查询的字段靠前能提高索引的命中率

**TRUNCATE和DELETE的区别**

Drop：删除表结构和数据（DDL），不能回滚，会释放空间

truncate删除表中全部数据(DDL)，不能回滚，会释放空间，不会删除索引

Delete 根据where条件删除数据(DML),可以回滚，不会释放空间

**sql执行过程**

1 from 左边的表们s

2 on 连接条件

3 连接类型(left|inner) join 右边的表们

4 where 筛选条件

5 group by 分组列表(按什么字段分组)

6 having 筛选条件

7 select

8 distinct 查询列表(要查询的字段)

9 order by 排序的字段等

10 limit

**Sql语句很慢怎么排查**

1.explain看执行计划，是否命中了索引

2.看数据量，是否因为数据多导致了瓶颈

3.看网络/机器负载

4.热点数据导致单点负载不均衡

**如何使用explain和慢日志来分析sql语句**

explain：查看索引使用情况, 用法：explain sql语句

id字段：每个select的id

select\_type：查询类型

simple:简单select查询，不包含子查询/union

primary: 复杂子查询的最外层

subquery：select/where中包含子查询

deriued：from列表中包含的子查询

union: 第二个select出现union后，表记为union,若Union包含在from子句中的子查询中，外层select表记为derived

union result：从union表中获取结果的select

type：索引级别

从好到差： system>const>eq\_ref>ref>range>index>all

一般查询要达到range级别，最好达到ref

system:表中只有一行记录，系统表

const:通过索引一次就能找到，用于primary key/unique索引，只匹配一行数据

eq\_ref：唯一性索引扫描等值查询

ref：非唯一索引扫描等值查询

range：值检索给定范围的行， where语句中出现between,<,>,in等，比全表扫描好一点

index：从索引遍历全表，比all块

all：从硬盘遍历全表

possible\_keys:查询可能用到的索引

key: 查询实际用到的索引，查询中使用了覆盖索引，则指挥出现在key列表中

key\_len: 索引中使用的字节数

ref: 显示索引的哪一列被使用了，哪些列/常量被用于查找索引列上的值

row:有多少行被优化器查询到， 越少越好

extra：额外的信息

using filesort(使用文件内排序):不按索引排好的顺序读取,重新排序(九死一生)

using temporary(使用临时表):order by /group by中使用临时表保存中间结果，(拖慢sql的元凶，要少使用)

using index(使用索引):select使用了覆盖索引，效率不错，

如果同时using where，表明索引被用来执行索引键值的查找

如果没有，则表明索引读取数据而非执行查找

using where:使用where过滤

using join buffer:使用join缓存

impossible where: where子句总是false,不能用来获取任何数据

selelct tables optimized away:没有group子句的情况下，在min/max/count(\*)在查询执行计划生成时完成，不在执行计划阶段再进行计算

distince: 在找到第一匹配的数据(元组)后，停止找同样是数据

**数据库并发操作的问题**

脏读：B修改数据后但未提交事务，被A读取到，B回滚，A读取有误，(读到事务未提交的数据)

不可重复读: 事务A多次读取数据，事务B对数据修改，导致数据不一致（重点在数据）

幻读：事务A多次读取，事务B进行新增/删除，导致读行数不一致（重点数量修改）

**mysql中的锁**

表锁有：表级读/写锁，读意向锁，写意向锁，自增锁；

行锁有：记录锁，[间隙锁](https://so.csdn.net/so/search?q=%E9%97%B4%E9%9A%99%E9%94%81&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/FINYIE/article/details/_blank)，Next-key 锁，插入意向锁。

记录锁(Record Locks): update user set age = 1 where id = 1; 只锁id=1的记录

间隙锁(Gap Locks):update user set age = 1 where id > 1 and id < 10 ,锁住(1,10)区间

临键锁(next-key locks): update user set age = 1 where id>1 and id <= 10, 锁住(1,10]区间，左开右闭

**数据库隔离级别**

Read uncommitted(读未提交)： 查询不加锁，并发性最好，一致性最差

Read committed(读已提交)： 避免脏读

Repeatable read (可重复读)： 避免脏读，不可重复读；

Serializable(串行化)： 避免脏读，不可重复读，幻读；有未提交的事务正在修改某些行，所有读取这行的select都会被阻塞；一致性好，并发性差，高并发不适用

**mysql索引优化器**

Mysql执行会根据环境分析，筛选出最适合的索引

**Mysql,innodb为什么用可重复读（RR）作为默认隔离级别**

RR在保证数据一致性的同时，允许并发操作，性能比串行化高

**MVCC(多版本并发控制)**

事务读取数据时，会生成对应的快照版本，每次读的都是该版本，而不是最新的版本呢

**MVCC(多版本并发控制)，可重复读下生成ReadView时机，如果生成两个ReadView带来什么问题**

会造成前后读取到的数据不一致

**事务的四大特性（acid）**

原子性：要么全部执行成功，要么回滚全不执行

一致性:事务前后完整性约束没被破坏，字段只有失效/生效，如果改成待定，则需要回滚

隔离性:在事务执行完前，访问的数据不能被其他事务影响

持久性:事务提交后，对数据库的更改会保存

**mysql事务的acid特性是怎么什么保证/实现的**

原子性：undo log（执行失败后进行回滚）。

持久性：redo log（宕机后，重新执行修改语句）。

隔离性：MVCC机制（快照读、当前读）来保证隔离性。

一致性：两阶段提交事务。

**冗余字段的优缺点**

优：减少查询时的表关联，提高sql速度，尽量用在不太修改的字段上

缺：违反三范式，修改时要同步修改冗余字段，漏修改会导致数据不一致

**冗余表的优点**

用来恢复数据；分主从数据库，读写分离，提高读写性能

**Mysql存储ip，用什么类型**

用字符串类型也可以

IPV4最好用数字类型，比字符串更加节省空间，查询起来更快

**mysql存储货币用什么类型**

Decimal和Numric，不会有精度误差

**存储引擎innodb和myisam对比及使用情况**

InnoDB(适合需要事务，适合频繁更新，一般以innodb作为存储引擎):

支持事务，外键，支持行锁

不支持全文索引

MyISAM(适合无事务，适合读为主):

不支持事务，外键

支持全文索引

读取快，空间占用少

**Mysql常用的数据类型**

1. 整数类型：TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT

2. 浮点数类型：FLOAT、DOUBLE、DECIMAL

3. 字符类型：CHAR、VARCHAR、TEXT、TINYTEXT、MEDIUMTEXT、LONGTEXT、BLOB、TINYBLOB、MEDIUMBLOB、LONGBLOB

4. 日期和时间类型：DATE、TIME、DATETIME、TIMESTAMP、YEAR

5. 布尔类型：BOOLEAN

6. 枚举类型：ENUM

7. 集合类型：SET

8. JSON类型：JSON

9. 空间数据类型：GEOMETRY、POINT、LINESTRING、POLYGON

**行锁和表锁**（增删改查匹配到索引时，Innodb 会使用行锁。如果没有匹配不到索引，那么就会直接使用表级锁。）

行锁：只锁某几行，开销大，加锁慢；会出现死锁；锁定粒度小，锁冲突概率低，并发度高

表锁：锁整个表。开销小，加锁快；不会出现死锁；锁定粒度大，锁冲突概率高，并发度低

页锁：锁一页数据(16kb)，会出现死锁；介于行锁和表锁之间

**主键索引**

如果存在id = 5，临键锁则退化成记录锁，只锁住id = 5的这行数据

如果不存在id = 5，则锁住(1,10]这个区间

**Delete时加锁**

delete from test where a = '1' ，a 既不是主键也不是普通索引 ，会加行锁

**插入一亿条数据**

更新1亿条：使用update case where语法

插入1亿条，使用 insert into values()语法

分库分表，读写分离，分批次，分线程，分机器，将大问题分而治之

**主从复制（主从同步）的过程**

减缓数据库读写压力，读写操作分离，读操作分散到几台不同的从服务器中，主服务器的写操作会通过bin log(二进制日志，记录了写的sql)发送给从服务器进行执行，做到数据同步

**主从复制延迟的原因**

1.从服务器太多，需要复制的节点太多

2.从库的服务器性能比主库差，造成写的sql语句执行的时间长，

**主从同步延迟，读写分离，导致数据不一致如何处理**

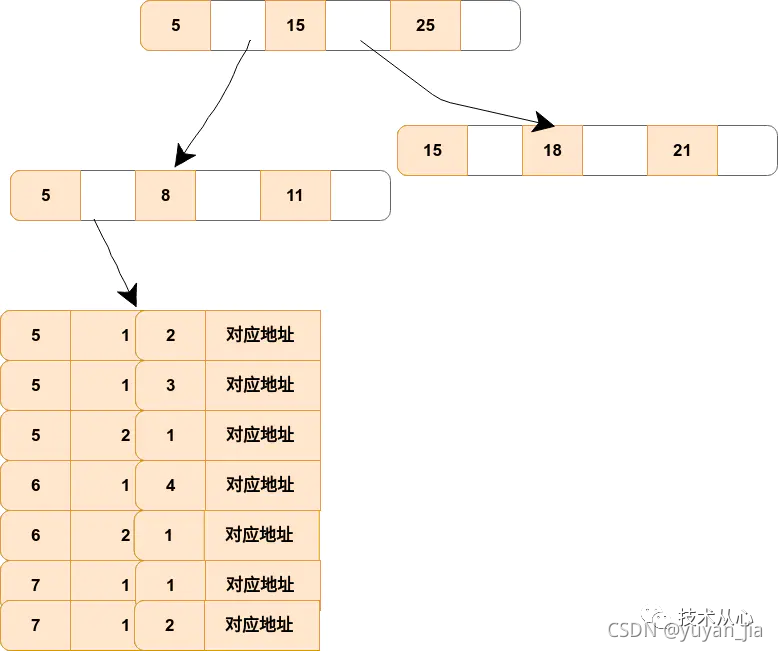
修改数据后，将key放入redis,如果redis存在key则去主库，不在则去从库查询

强制读主库

业务要求不高则最终一致性即可，无需处理

**最左匹配原则(索引结构)**

假如创建一个（a,b,c)的联合索引



**mysql数据库中cpu飙升要如何处理**

Top命令查看是否是mysqld导致

使用 show processlist 查看正在运行的会话，是否有慢sql在运行

找出消耗高的sql,查看执行计划，是不是少索引，数据量是不是过大

Kill掉有问题的线程

调整sql/索引后再执行

**百万级数据删除如何优化**

先删除索引，再删数据，再重建索引

如果需要保留的数据量较少,可以把保留的数据复制到新建的表里，删除原表，再将新建的表换表名

**JDBC连接数据库步骤**

Class.forName()加载JDBC驱动

建立连接(数据库地址，用户名，密码)

创建Statement ,执行sql

返回结果集

关闭连接

查看当前运行的SQL语句：

SHOW FULL PROCESSLIST;

SELECT pid, now() - pg\_stat\_activity.query\_start AS duration, query, state FROM pg\_stat\_activity WHERE state = 'active';

使用pg\_cancel\_backend函数终止慢查询：

SELECT pg\_cancel\_backend([进程ID]);

KILL [进程ID];

慢sql情况：关联查询很多，索引失效，数据多，条件复杂

**存储过程的好处**

将多个sql存储起来，减少建立连接的次数

进行预编译，将执行计划缓存，无需重新解析sql,执行快

**为什么不建议使用存储过程**

调试麻烦，不好切换，不能用git管理版本

# Oracle

**存储过程和数据库函数的区别**

函数必须返回值，能在sql中使用，不能单独执行;

存储过程可以没有返回值，可以独立执行，适合复杂业务逻辑判断

**存储过程参数模式**

In:默认作为入参

Out:只能作为出参，不接收入参

In out ：可以接收入参，也能作为出参

**存储过程中的 is和declare**

Is/as后面声明的变量，整个包都能用

Declare声明的变量只有begin和end直接使用(局部)

**创建存储过程**

Create or replace procedure 存储过程名(param1 in type, param2 out type)

Is

v\_variable1 NUMBER; #变量声明

Begin

v\_variable1 := #赋值

Insert语句

Commit； //提交事务

Exception

When DUP\_VAL\_ON\_INDEX then //DUP\_VAL\_ON\_INDEX:插入的数据主键已存在

update语句

When others then //other 其他情况

Rollback; //事务失败执行回滚

End；

**存储过程中调用另一个存储过程**

Begin

存储过程名(); #方式1

call 存储过程名();*-*

End

**简单的循环(先执行后判断)**

Loop #循环

Fetch 游标名 into var1, var2; #获取下一行游标的内容

EXIT WHEN 游标名%NOTFOUND; #当游标的下一行找不到时退出

End loop;

**While循环 :多了进入循环的条件**

While count < 5 loop

count:= count+1;

Exit when ....

End loop;

**For 循环**

For item in (select \* from table) loop

If (item.sex = 0) then 逻辑

End if

End loop

**存储过程的游标(cursor)**

集合类型，遍历查询结果的集合

Declare

游标名 CURSOR FOR SELECT column1, column2 FROM table1

var1 table1.column1%TYPE;

var2 table1.column2%TYPE;

Begin

Open 游标名; #开启游标

Loop #循环

Fetch 游标名 into var1, var2; #获取下一行游标的内容

EXIT WHEN 游标名%NOTFOUND; #当游标的下一行找不到时退出

End loop;

Close 游标名 #关闭游标

End

**oracle定义属性类型**

表名.字段名%type

表名%rowtype : 将所有字段定义成一个结构体类型

TYPE t\_my\_table IS TABLE OF my\_table%ROWTYPE;

my\_table\_data t\_my\_table;

**Forall用法 遍历集合进行批量插入**

SELECT id, name BULK COLLECT INTO my\_table\_data FROM another\_table;

FORALL i IN my\_table\_data.FIRST .. my\_table\_data.LAST

Insert语句 my\_table\_data(i)

Bulk collect:将查询结果一次性地加载到集合中

IS TABLE OF:指定是一个集合的表的数组类型

**存储过程定义定时任务**

DBMS\_SCHEDULER.create\_job (

job\_name => 'my\_job',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN 存储过程名; END;',

start\_date => SYSTIMESTAMP,

repeat\_interval => 'FREQ=MINUTELY; INTERVAL=5', -- 每5分钟执行一次

enabled => TRUE

);

**Oracle中字符串怎么拼接**

||

**Oracle中如何分页查询**

select \* from

 ( select rownum ,\* from 表名where rownum <= 20 )

where rownum > 10

**Oracle触发器写法**

CREATE OR REPLACE TRIGGER 触发器名称

BEFORE/AFTER/INSTEAD OF 触发事件ON 触发对象

[FOR EACH ROW]

[WHEN (condition)]

DECLARE

Begin

End

**oracle中的null**

条件中尽量避免null判断，会造成索引失效，建议给字段设置默认值

**字符串转为日期类型**

TO\_DATE('10-FEB-2025', 'dd-mon-yyyy')

Oracle中不能再group by后进行子查询

Continue：跳出当层循环

Exit跳出整个循环

Oracle中空字符串就是null

**Oracle查看执行计划**

Explain plan for sql语句

**创建临时表**

Create table 表名 as (select语句) ： 创建的是物理表，需要手动删除

With 表名 as(select语句)： 创建的逻辑表，存放临时结果