Java高级工程师

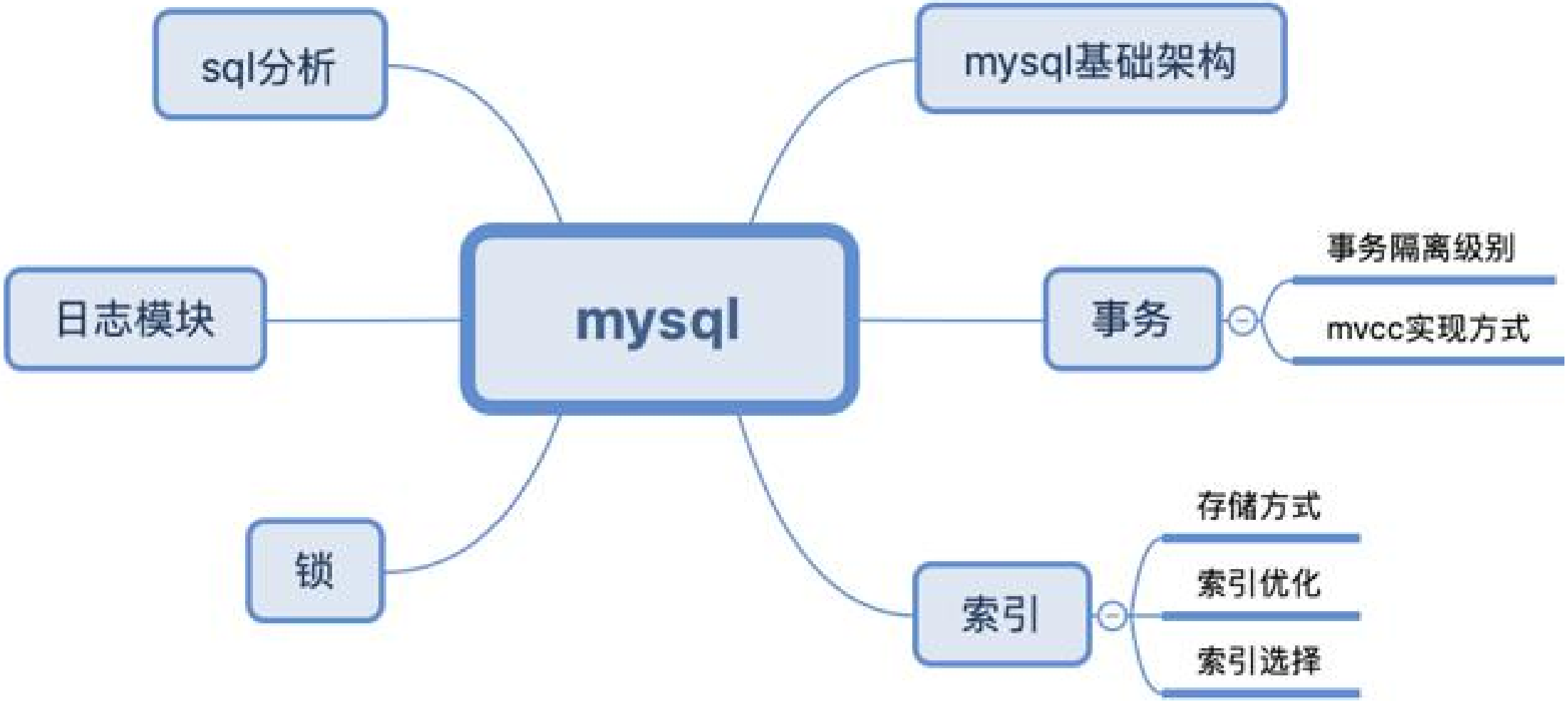
数据库基础知识

课程大纲

教学目标：应对数据库面试

1. ORM框架在项目中的运用
2. Java的数据库研发
3. 数据库基本知识

知识点思维导图



你知道在mysql中一条sql语句是怎样运行的吗?

频 度: 中
难 度: 高
通 过 率: 低

考察方法：你问我答

- 误区:

回答的是sql的语法而没意识到这道题问的是mysql的基础架构

正确回答

01 访问客户端通过连接器执行sql

02 分析器解析sql语句后分解为指令集

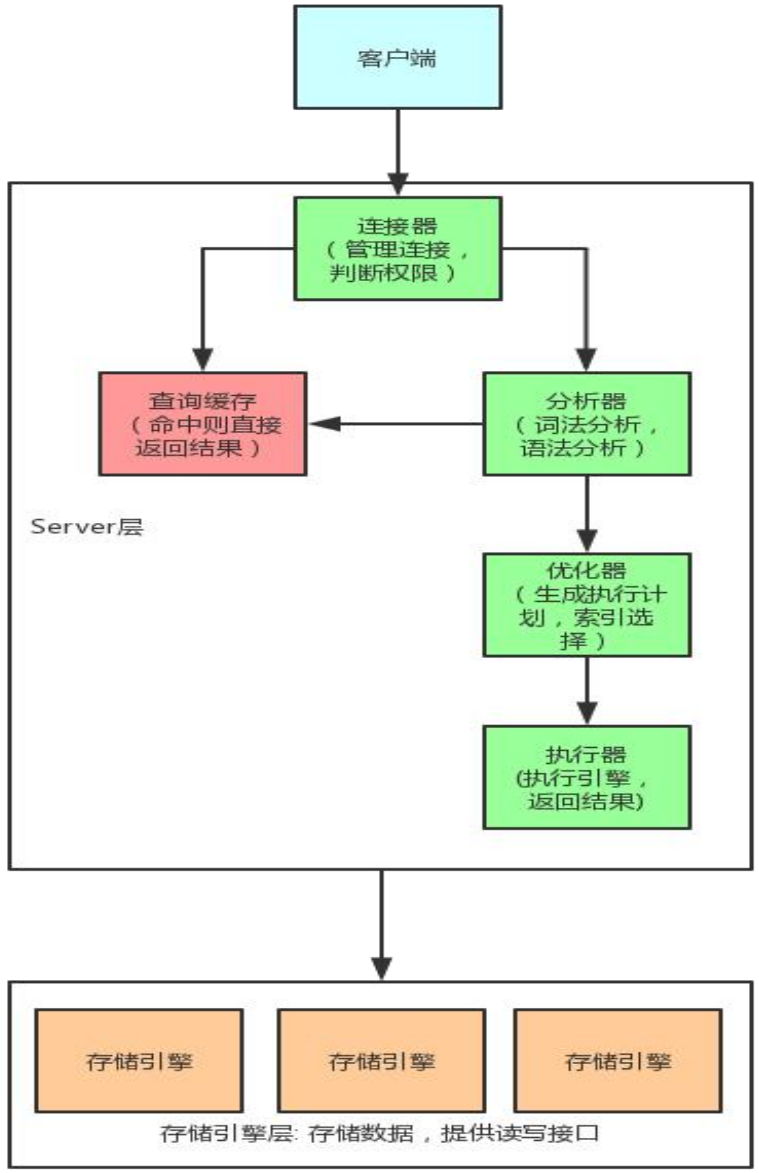
03 先查询缓存,命中就直接返回结果

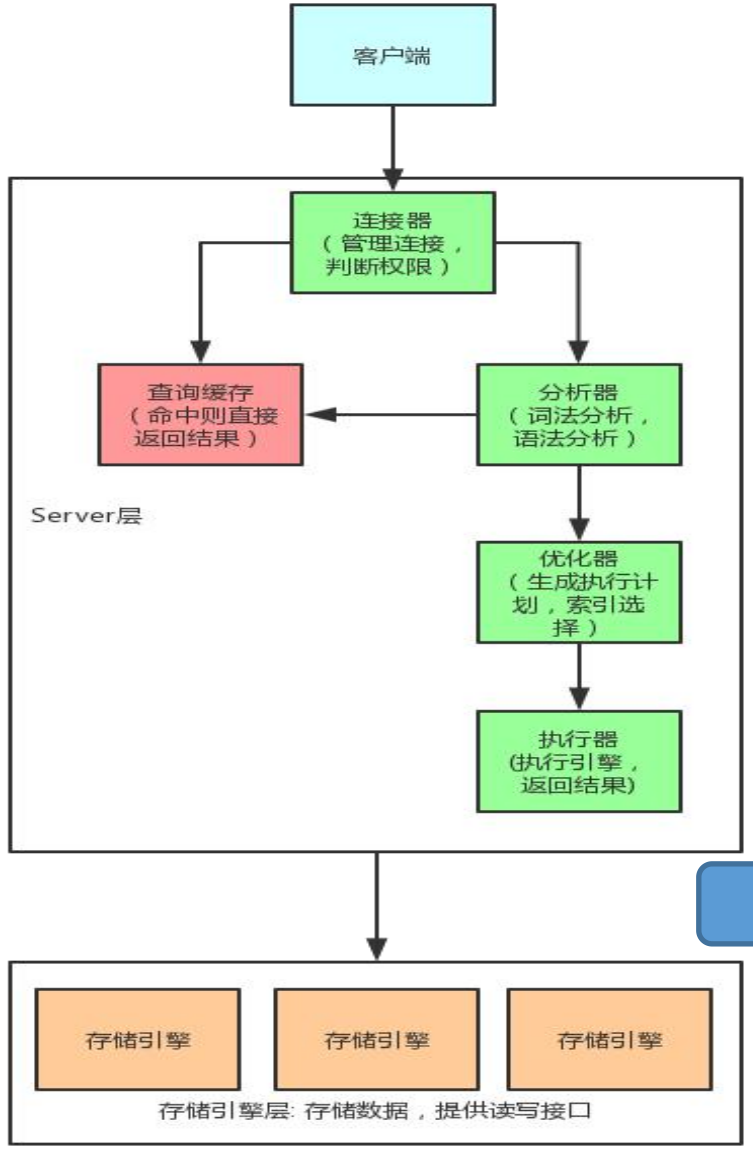
04 非查询或未命中,进入优化器产生执行计划并选择索引

05 执行器执行后返回结果

Mysql架构解读

Mysql架构上大体分为两层:Server层和存储引擎层





架构分界

连接器

用来管理连接和判断权限

一个安装好的数据库

用户名/密码:cup/cup

```
)
ys@izbp1fdo6bq9nkc18dm5r2z bin]$ ./mysql -u cup -p
nter password:
```

输入口令:


```
[ys@izbp1fdo6bq9nkc18dm5r2z bin]$ ./mysql -u cup -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1878
Server version: 5.6.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database jzsf;
Query OK, 1 row affected (0.00 sec)

mysql> use jzsf
Database changed
mysql>
```

```
[ys@izbp1fdo6bq9nkc18dm5r2z bin]$ ./mysql -u cup -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1878
Server version: 5.6.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database jzsf;
Query OK, 1 row affected (0.00 sec)

mysql> use jzsf
Database changed
mysql>
```

新建一个数据库

```
[ys@izbp1fdo6bq9nkc18dm5r2z bin]$ ./mysql -u cup -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1878
Server version: 5.6.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database jzsf;
Query OK, 1 row affected (0.00 sec)

mysql> use jzsf
Database changed
mysql>
```

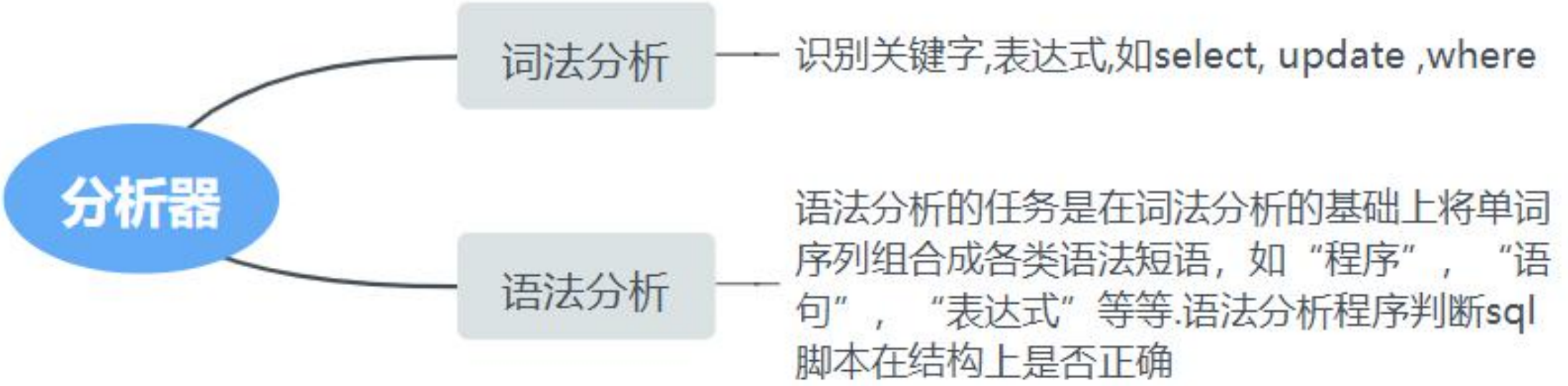
选择新建的数据库

查询缓存

1. 如果一个表的数据更新了，那么这个表上的所有查询缓存都会失效。
2. 如果表的数据更新比较频繁，那么基本上就用不上查询缓存了。
3. 在mysql8.0版本后就删除此功能了。

查询缓存以key-value的形式进行缓存，
key是查询的语句
value是查询的结果。

分析器：对sql语句进行词法分析及语法分析



优化器

生成执行计划，并进行索引的选择

执行器

调用存储层的接口，执行引擎，返回结果

存储引擎层

负责数据的存储和读取

InnoDB、MyISAM、Memory等，其中InnoDB为Mysql 5.5.5版本的默认存储引擎

提问

Mysql有哪些存储引擎,他们之间有什么区别?

- 频 度: 高
- 难 度: 低
- 通 过 率: 中

解析

MyISAM存储引擎

1. MySQL如果使用MyISAM存储引擎，数据库文件类型就包括.frm、.MYD、.MYI
2. 索引:使用B+tree索引但是和Innodb的在具体实现上有些不同
3. MyISAM 这种存储引擎不支持事务，不支持行级锁，只支持并发插入的表锁，主要用于高负载的select
4. 占用空间小，处理速度快
5. 不支持事务的完整性和并发性

innoDB存储引擎

1. 数据库文件类型就包括.frm、ibdata1、.ibd
2. 提供了事务，回滚以及系统崩溃修复能力和多版本迸发控制的事务的安全。
3. 索引使用的是B+Tree
4. 优点:提供了良好的事务处理、崩溃修复能力和并发控制。
5. 缺点:读写效率较差，占用的数据空间相对较大。

MEMORY存储引擎

1. 使用存储在内存中的数据来创建表，而且所有的数据也都存储在内存中

2. 每个基于memory存储引擎的表实际对应一个磁盘文件，该文件的文件名和表名是相同的，类型为.frm。该文件只存储表的结构，而其数据文件，都是存储在内存中

3. 使用哈希（HASH）索引，其速度比使用B-+Tree型要快 反面:开销也比较大

4. mysqld进程发生异常，重启或关闭机器这些数据都会消失。所以memory存储引擎中的表的生命周期很短，一般只使用一次。

ARCHIVE存储引擎

1. 适合存储大量独立的、作为历史记录的数据
2. 提供了压缩功能，拥有高效的插入速度
3. 不支持索引，所以查询性能较差一些。

什么是事务,什么引擎支持事务

频 度: 高

难 度: 低

通 过 率: 高

解析

- （1）数据库操作的最小工作单元，是作为单个逻辑工作单元执行的一系列操作；
  这些操作作为一个整体一起向系统提交，要么都执行、要么都不执行；
  事务是一组不可再分割的操作集合（工作逻辑单元）
- （2）InnoDB支持事务

事务有些什么特征?

频 度: 高

难 度: 低

通 过 率: 高

解析

这样就开启了一个事务


```
mysql> begin work;
Query OK, 0 rows affected (0.00 sec)
```

建一张表,插一条数据


```
mysql> begin work;
Query OK, 0 rows affected (0.00 sec)

mysql> create table userinfo (id varchar(10));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into userinfo values ('1');
Query OK, 1 row affected (0.00 sec)

mysql> select * from userinfo;
+------+
| id   |
+------+
| 1    |
+------+
1 row in set (0.00 sec)
```

在事务中插第二条数据,然后回滚它


```
mysql> begin; insert into userinfo values ('2'); rollback;
Query OK, 0 rows affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> select * from userinfo;
+------+
| id   |
+------+
| 1    |
+------+
1 row in set (0.00 sec)
```

你知道事务有哪些隔离级别?它们之间的区别是什么?

频度:高

难度:低

通过率:中

答案：

可以看出
这是一个由低到高的过程

一个事务还未提交，它的变更就能被别的事务看到

一个事务提并后，它的变更才能被其他事务看到

一个事务过程中看到的数据，总是跟这个事务启动时候看到的数据是一致的。未提交的事务对其他事务是不可见的。

写会加写锁，读会加读锁，当读写冲突时，后一个事务需等待前一个事务结束才可继续。

查询隔离级别


```
mysql> show global variables like '%isolation%';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+
1 row in set (0.00 sec)
```

设置隔离级别


```
mysql> set global tx_isolation='read-committed';
Query OK, 0 rows affected (0.00 sec)

mysql> show global variables like '%isolation%';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | READ-COMMITTED  |
+---------------+-----------------+
1 row in set (0.00 sec)
```

举例分析

- 这里有一张表T，表里有字段C。C的值为1；假设现在有两个事务，分别对C进行如下操作：

事务A	事务B
启动事务,查询得到值为1	启动事务
	查询得到值1
	将值1改为2
查询得到值V1	
	提交事务
查询得到值V2	
提交事务	
查询得到值V3	

不同的事务隔离级别会有什么样的效果

频度:高

难度:低

通过率:高

读未提交

V1=2；V2=2；V3=2

读未提交

事务未提交，其他事务便可看到修改的值，所以V1时刻能看到事务B的修改，所以V1=2；事务B提交后，V2值不变，还是2;事务A提交后，V3值还是不变，还是2;

读已提交

V1=1；V2=2；V3=2

读已提交

事务提交后其他事务才能看得到值，所以V1=1；V2时刻，事务B已提交，所以V2=2；事务A提交后，V3的值还是为2；

可重复读

V1=1；V2=1；V3=2

可重复读

事务过程中看到的数据，与事务启动时刻是一致的。所以V1与启动时查到的值1一样。V2时刻，事务B提交，但是在可重复读的隔离机制下，V2的值也与事务开始时的值一致，V2=1；事务A提交后，此时V3默认开启一个事务，这个事务读取的些为当前最新的值为2。

串行化

V1=1；V2=1；V3=2

串行化

读会加读锁，写会加写锁。所以事务A在启动时的查询动作，会加一个读写。待到事务B进行“将的值1改为2”操作时，会需要加写锁而等待事务A释放。所以事务B会一直等待，直到事务A提交。

所以V1=1；V2=1；事务A提交后，事务B继续操作，将值修改为2；之后V3的查询，需要等事务B提交后才会执行，因为此时V3的查询需要获取读锁，但是此时事务B占着写锁。待事务B提交后，执行V3查询，V3=2；

问题:多个事务并发时,用什么样的机制能保证数据的一致性(符合预期)

频度:高

难度:低

通过率:高

答案

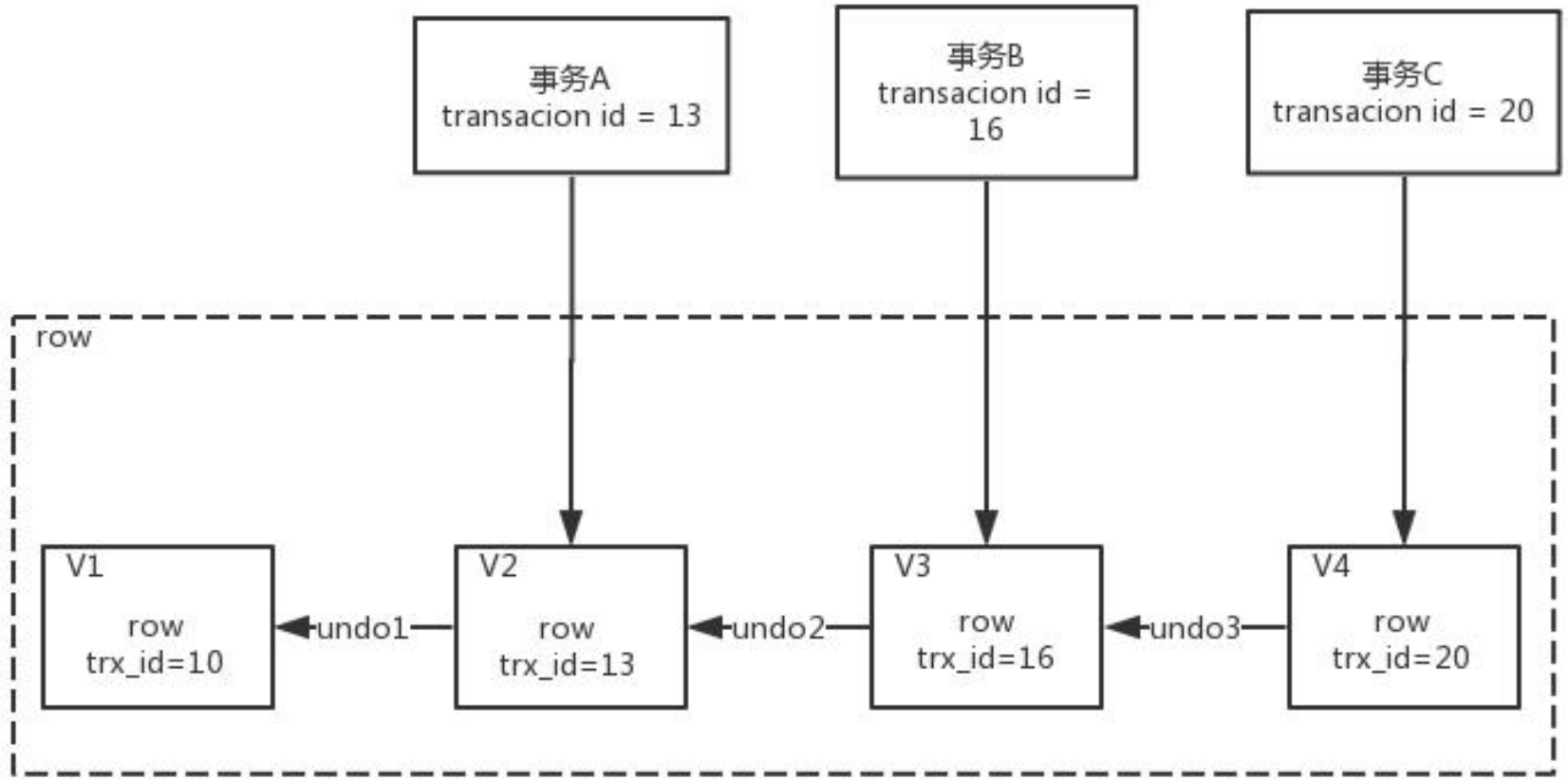
MVCC(Mutil-Version Concurrency Control**)**

多版本并发控制

在Mysql的InnoDB引擎中就是指在已提交读(READ COMMITTD)和可重复读(REPEATABLE READ)这两种隔离级别下的事务对于SELECT操作会访问版本链中的记录的过程。

MVCC机制

数据库表中的每行数据都是有多个版本的。每次事务更新时，都会产生一个新的版本，并且把transaction id的值赋给这个版本，记做row trx_id。



MVCC机制

一个事务在启动的时候，找到所有已经提交的事务ID 的最大值，记为 up_limit_id；

MVCC机制

在可重复读隔离级别下，只需要在事务开始的时候找到那个up_limit_id，之后事务里的其他查询都共用这个 up_limit_id；

MVCC机制

在读提交隔离级别下，每一个语句执行前都会重新算一次 up_limit_id 的值。

选择使用哪个版本的依据

如果一个数据版本的 row trx_id 大于 up_limit_id，那就不认，必须要找到它的上一个版本。

怎么决定?

通过最新版本和undo log日志配合，进行回滚获取前面版本的视图信息。

选择使用哪个版本的依据

如果一个事务自己更新的数据，自己还是要认的

查询语句的一致性读

在可重复读和读已提交两种隔离级别下，看到的数据视图版本叫做一致性读

查询语句的一致性读

根据 row trx_id 和 up_limit_id 的大小决定数据版本的可见性。

事务更新数据用当前读

读的是当前的值，称为当前读

事务更新数据用当前读

读的是当前的。如果当前的记录的行锁被其他事务占用的话，就需要进入锁等待，称为当前读

事务更新数据用当前读

如果查询语句加锁，也是当前读。

总结

可重复读，查询只承认在事务启动前就已经提交完成的数据

总结

读提交，查询只承认在语句启动前就已经提交完成的数据

总结

当前读，总是读取已经提交完成的最新版本。

数据库调优篇

数据中的索引有些什么类型?

频度:高

难度:低

通过率:中

什么是索引

排序的一种存储结构

相当于图书的目录

索引的分类

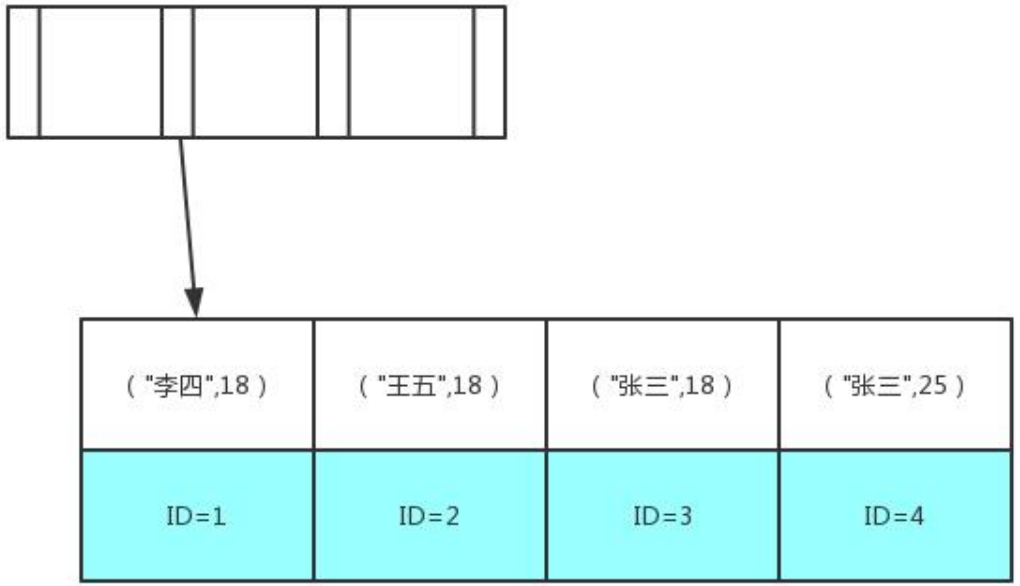
唯一索引

索引值唯一

如身份证号、手机号

前缀索引

如下图，我们创建一种（name,age）的复合索引。复合索引中，是由多个字段组成索引值。



规则

最左侧前缀原则，就是只要满足最左前缀，就可以用索引来加速检索

规则

在执行“where name like ‘张%’ ”的语句时就能用这个索引

规则

但是如果我们执行的是“where name like ‘%张’ ”这个语句，
但不能用上上述的复合索引了

规则

最左前缀可以是复合索引的最左N个字段，也可以是字符串的索引的最左M个字符。

你平时怎样做索引优化

频度:高

难度:低

通过率:中

答案

思路:尽量减少回表次数

什么是回表?

根据索引找到了指定的记录所在行后，还需要根据rowid再次到数据块里取数据的操作。

以下图索引为例，分析覆盖索引和索引下推是如何做优化的

覆盖索引的优化

将要查询的字段能直接通过非主键索引获取，不需要通过回表查询得到。

覆盖索引的优化

select name,age from t where name="王五"

索引下推的优化

非主键索引遍历时，提前判断条件是否符合
不须要再通过回表后取到字段再进行判断

索引下推的优化

select name,age,sex from t where name="张三" and age > 20

索引下推的优化

当找到（“张三”,18）索引值时，没优化的做法，是通过ID=3回表查询获取age的值，再判断age>20是否成立，来决定此记录是否是满足的。这样，就需要多次回表判断。如果在遍历非主键索引时，就直接通过（“张三”,18）获取age的值18,进行age>20的判断。成立再回表获取sex的字段值，不成立就无须回表获取值。

数据库锁

数据库锁设计的初衷是为了解决并发问题

数据库有哪些锁

全局锁

对整个库加锁

数据库有哪些锁

表级锁

表级锁分为两种

一是表锁

二是在Mysql5.5版本后，新增了元数据锁（MDL）

数据库有哪些锁

表级锁

lock tables ... read/write

unlock tables

数据库有哪些锁

行锁

行锁是一个两阶段的锁，在InnoDB事务中，行锁是需要时才加上的

待到事务结束才释放。

行锁的类型

共享锁(S)

排他锁(X)

Note

行锁是加在索引上的，不是加在记录上

问题:你知道行锁的加锁规则吗?

频度:中

难度:低

通过率:中

自动加锁

对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及的数据集索引加排他锁

自动加锁

对于访问到的数据，InnoDB都会加锁。

这里有个细节，就是InnoDB在二级索引上使用的是共享锁，而在主键索引需要排他锁；如果没走索引，走的是全表扫描，会锁住所有的行。

显式加锁

SELECT * FROM table_name WHERE ... FOR UPDATE

锁的释放时机

mysql 5.1 版本前，需要在事务提交后才释放锁

mysql 5.1 版本后，InnoDB在服务层过滤掉行后，就释放对应行的行锁

你听说过“幻读”吗?为什么会产生幻读的现象?

频度:中

难度:低

通过率:中

幻读

就是同一个事务里，两次相同查询语句获取当前读时，第二次当前读获取的记录数比第一次的多。多的这部分记录就是幻读了。

这里有一张表T，表里有字段c,和主键id。id=1,c=1

事务A	**事务B**
select * from t for update;	
	insert into t (2,3)
select * from t for update;	

幻读

事务A第一次执行查询时，获取的写锁，并且锁住了所有的行。

幻读

事务B执行插入，由于新插入的行不在事务A行锁的范围内，所以事务B插入成功。

幻读

事务A再次执行查询所有记录，跟第一次执行查询就会多出事务B新插入的（2,3）这一行

原因

是事务B的插入不受事务A行锁的限制

解决幻读问题

间隙锁

间隙锁

在记录的间隙加锁

间隙锁

InnoDB在可重复读的事务隔离级别下，会有间隙锁；
也就是说，在可重复读级别下，是不会出现幻读现象的。

面试题:自增长列在并发时是怎么保证不会产生重复记录的?

频度:中

难度:低

通过率:中

答案

自增锁

自增锁

在插入语句获取自增主键值的时候，独占获取自增锁。

自增锁

获取自增主建值后，到这个语句执行结束释放自增锁；其他语句才能再获取这个表的自增锁，从而获取自增主键值。

你知道mysql有哪些重要的日志吗?它们的用途是什么?

频度:中

难度:低

通过率:中

binlog日志

binlog数据保存的是逻辑日志，记录的是语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ;”。

主要用于主从复制

Mysql的读写分离实践

主从库方式实现

主库写

从库读

与事务有关的日志

redo log

undo log

redo log 物理日志,在某个数据页上做了什么修改

undo log 逻辑日志事务回滚时，只是将数据库逻辑地恢复到原来的样子

事务的记录过程

Undo log是如何保证事务的原子性和持久化的?

- 频度:中
- 难度:高
- 通过率:低
- .

A 更新数据前记录undo log。

B 为了保证持久性，必须将数据在事务提交前写到磁盘。只要事务成功提交，数据必然已经持久化。

C undo log必须先于数据持久化到磁盘。如果在7,8之间系统崩溃，undo log是完整的，可以用来回滚事务。

D 如果在1-6之间系统崩溃,因为数据没有持久化到磁盘。所以磁盘上的数据还是保持在事务开始前的状态。

Update和delete在事务中是如何记录的?

- 频 度: 中
- 难 度: 高
- 通 过 率: 低

Delete

delete操作实际上不会直接删除，而是将delete对象打上delete flag，标记为删除，最终的删除操作是purge线程完成的。

update

分为两种情况：update的列是否是主键列。

不是主键列：

undo log中直接反向记录是如何update的

是主键列：

update分两部执行：先删除该行，再插入一行目标行

课程总结

本讲对数据库基础知识进行了回顾

面试考察中“说”的部分主要通过对数据库的细节知识考察来验证候选人对数据库的理解

课后作业

1、尝试在linux上跟随网络教程安装好mysql,并启动成功\

2、在安装好的数据库上通过调整事务隔离级别重现今天课程上提到的例子