法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：大数据分析挖掘
  - 新浪微博：ChinaHadoop

分布式爬虫

大纲

- 数据库基本概念
- MySQL 的数据存储结构
- MySQL 查询过程
- MySQL 的常规优化
- 深翻页过程、性能影响及优化

数据库常见概念

锁

- 表级锁：表锁是开销最小的锁策略，会锁定整张被访问到的表。写之前要获得写锁，会阻塞其它所有的读写操作；读锁属于共享锁，读互相之间不阻塞；写锁的优先级高于读锁，也就是说在排队序列中，写的操作会被插入到读之前
- 行级锁：行锁可以最大程度支持并发处理，但同时增大了锁开销。行级锁只在存储层实现（例如INNODB支持行级锁，而MyiSAM 只支持表锁）

事物

- Atomicity：原子性。一个事物被视为一个不可分割的最小单元
- Consistency：一致性。数据库总是从一个已知悉状态转换到另一个一致性状态，例如执行过程崩溃，数据库的状态并不会发生变化
- Isolation：隔离性。一个事物所做的修改在最终提交以前，对其它事物是不可见的
- Durability：持久性。一旦事物提交，则其所有的修改就会永久保存到数据库，此时即使系统崩溃，数据也不会丢失

死锁

死锁是指的多个事物在同一资源上相互占用，，并请求锁定对方占用的资源，例如以下两个事物：

```
START TRANSACTION;
UPDATE  Stock SET close = 45 WHERE id = 4;
UPDATE  Stock SET close = 39 WHERE id = 3;
COMMIT;

START TRANSACTION;
UPDATE  Stock SET high = 20 WHERE id = 3;
UPDATE  Stock SET high = 27 WHERE id = 4;
COMMIT;
```

如果正好第一行被执行完，并导致行被锁，那么彼此第二行都无法运行。InnoDB有很强的机制来检测，但是我们必须意识到这样的问题是可能出现的

AUTOCOMMIT

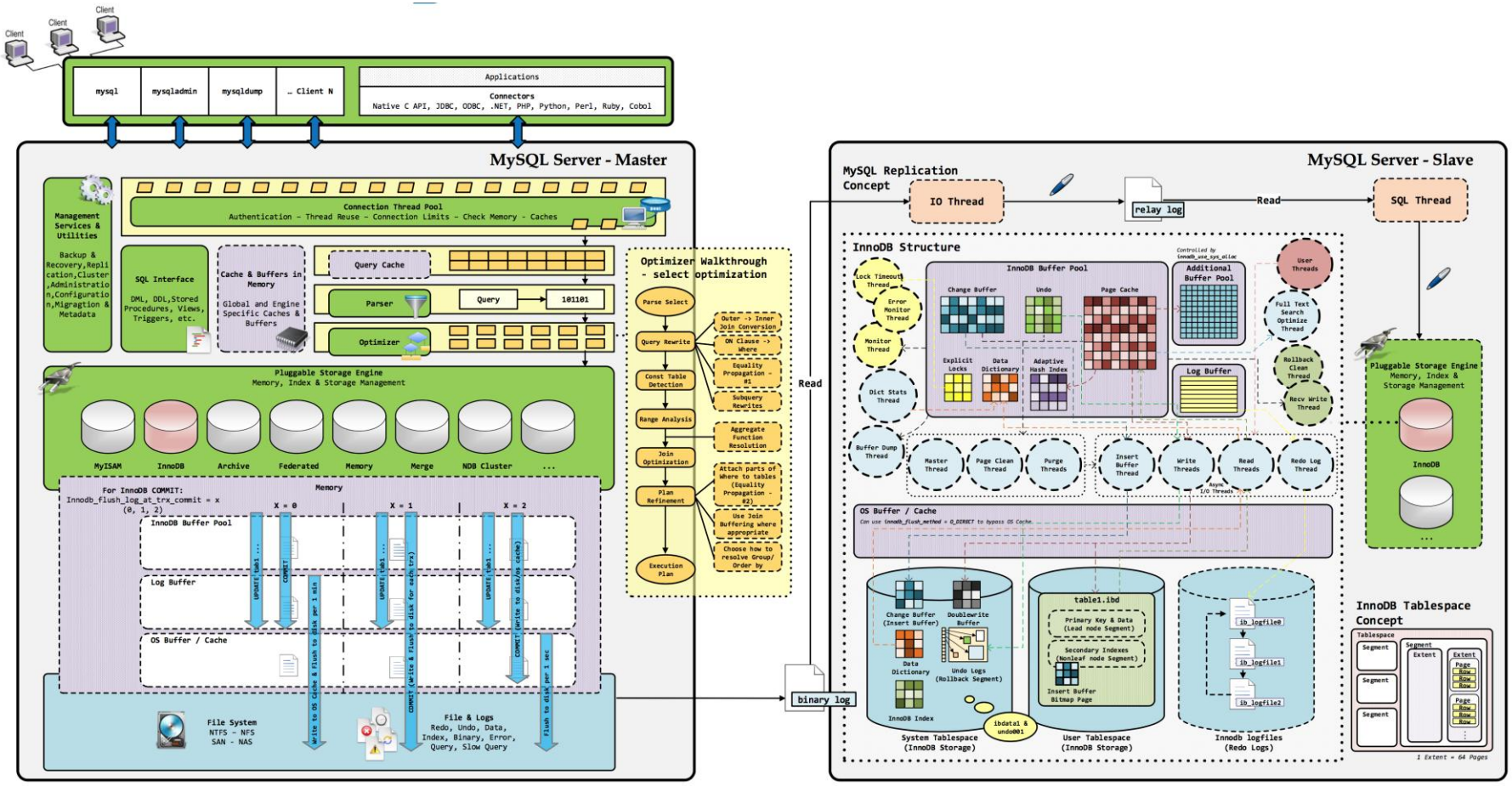
MySQL 默认采用自动提交，也就是默认把每个没有显式声明为事物的查询，当成一个一个独立的事物执行提交。

SHOW VARIABLES LIKE 'AUTOCOMMIT'

如果设置为 AUTOCOMMIT = 0，那么所有查询都在同一个事物中，必须显示使用 COMMIT 提交或者ROLLBACK 回滚

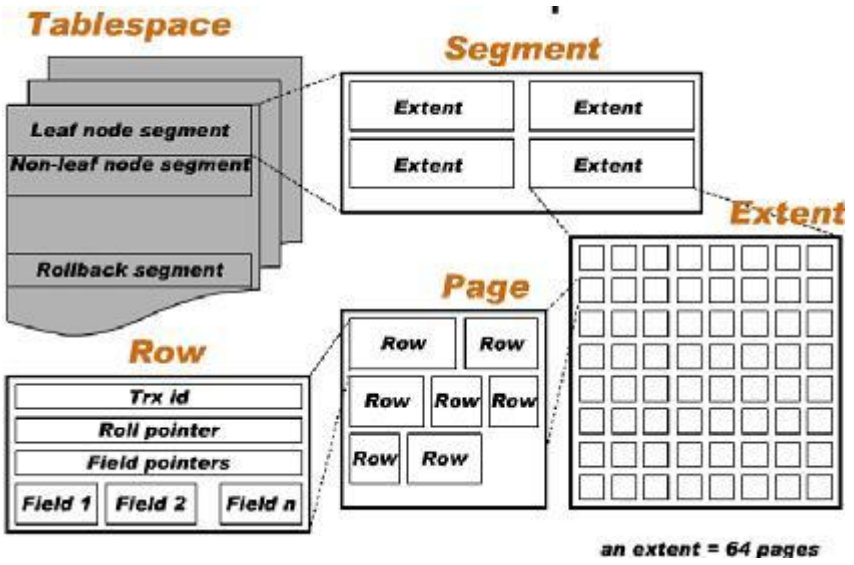
MySQL 的数据存储结构

系统框架及InnoDB存储架构



InnoDB 存储框架

- 一张表存储在一个或多个文件里
- 表包含多个 Segment，索引、数据、回滚记录等都是独立的Segment
- 每个Segment包含多个Extent
- Extent 包含 64 个 Page
- 每个Page 包含若干 Row
- 每个Row包含了数据域



InnoDB

- 主键索引既存储索引值，又在叶子中存储行的数据
- 如果没有主键,，则会Unique key做主键
- 如果没有unique，则系统生成一个内部的 rowid 做主键
- 次索引需要同时存主键ID
- 像innodb中，主键的索引结构中，既存储了主键值，又存储了行数据，这种结构称为”聚簇索引”
- 对主键查询有极高的性能，但是二级索引必须包含主键列，因此如果主键列很大，其它的索引都会很大
- 支持事物，行级锁，颗粒度小，并发好，但是加锁过程更复杂
- 崩溃后可以安全恢复

Myisam

- 不支持事物，不能安全恢复
- 表级锁，读的时候所有读到的表加共享锁，写的时候加排他锁
- 加锁效率高，并发支持效率低
- 支持前文索引（基于分词），可以支持复杂字段
- 可以延迟更新索引键，如何设置了DELAY_KEY_WRITE，每次修改完成时不会立刻将修改的索引数据写入磁盘，而是会写入缓冲区，因此能提高性能，但是崩溃的情况下索引会被损坏
- 设计简单，数据紧密格式存储，某些场景下性能很好

选择合适引擎

- 默认应选择InnoDB
- 一般不应使用混合引擎存储
- 如果需要事物，InnoDB是最稳定的；如果主要是 SELECT 或者 INSERT，那么MyISAM也许更合适，例如日志类型的应用
- 需要热备份的话，应该用InnoDB
- 重要数据应该InnoDB存储，MyISAM 崩溃后损坏概率比InnoDB大得多
- 大多数情况下，InnoDB 远远比 MyISAM 要快
- 只读的表，优先考虑MyISAM，会快很多

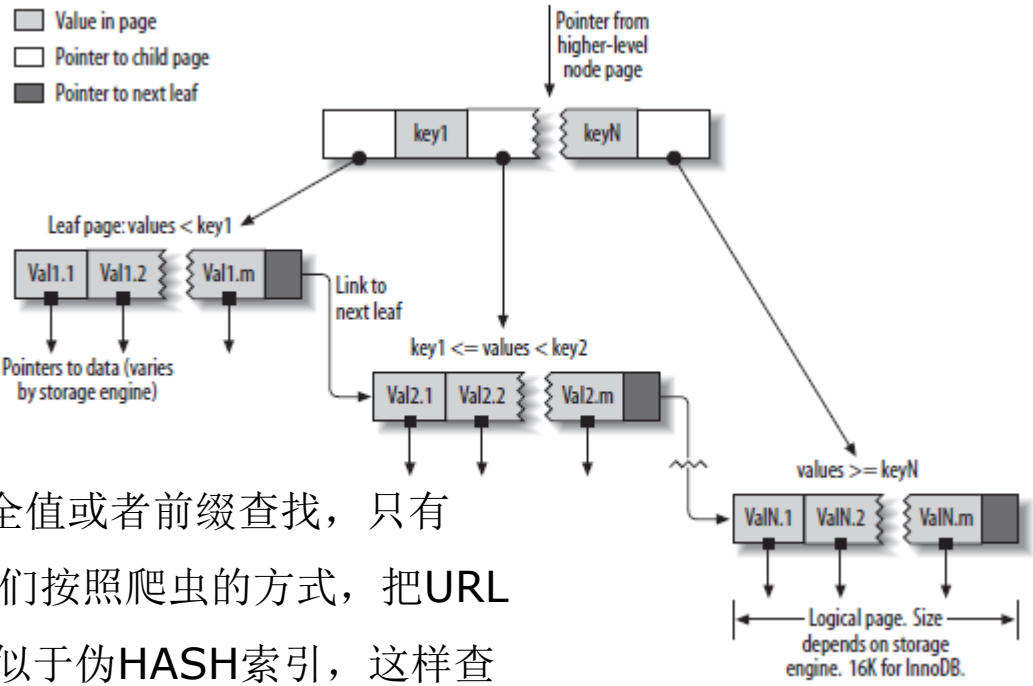
CHAR v.s. VARCHAR

- CHAR 会分配固定长度的空间，使得结构更加固定，尤其是在UPDATE的时候，没有额外的开销，对于很长的字符串，用CHAR会造成空间的浪费
- VARCHAR 可以分配变长的字符串，因为空间更加紧凑，根据实际需要来，但是当一个页满的情况下，UPDATE 更新会造成很大的负担，MyISAM 会把行拆成不同的片段存储，而InnoDB则会分裂页
- VARCHAR(5) 比 VARCHAR(200) 如果都存储 Hello 这个字符串，空间开销一样，但是更长的列会消耗更多的内存，尽量按照实际需求来设计

Schema 设计要点

- 避免太多的列：存储引擎API需要在服务器层与存储引擎层通过行缓冲格式拷贝数据，然后在服务器层讲缓冲内容解码成各个列。从行缓冲中奖编码过的列转换成行数据结构的代价是非常高的，它依赖列的数量
- 太多的关联：MYSQL限制了最多关联为61张表
- 避免使用NULL：对于索引列使用NULL，会带来存储以及大量优化问题
- 汇总表：把一些历史汇总数据，离线或定时汇总，最后总的结果是汇总结果加上当前的结果。例如个人的消费总额，可以把过去每个月的汇总，然后加上本月从1号到现在的SUM，这样的计算量会极大的减小。代价是写的速度会变慢，每个人的数据都需要定期被更新，但是读的性能得到了极大的提升。在数据库设计的时候，类似的冗余数据、统计结果的缓存往往是必要的

B+ Tree



B-Tree 索引，用来排序、范围查找、全值或者前缀查找，只有Memory引擎支持HASH索引，但是我们按照爬虫的方式，把URL进HASH计算后把HASH值存储，也类似于伪HASH索引，这样查找的就不再是 http://www.chinahadoop.cn/classroom/49/courses 这样的字符串，而是一个 11283737 这样的数值，利用 B-Tree 也能非常快速得到结果

多列索引

这是一个单列索引：

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

这是一个多列索引：

```
CREATE TABLE test (
        id INT NOT NULL,
        last_name CHAR(30) NOT NULL,
        first_name CHAR(30) NOT NULL,
        PRIMARY KEY (id),
         INDEX name (last_name,first_name)
);
```

多列索引可以看成是把多个列拼接在一起后，再排序的数组

多列索引

有用的多列索引：

```
SELECT * FROM test WHERE last_name='Widenius';
SELECT * FROM test WHERE last_name='Widenius' AND first_name='Michael';
SELECT * FROM test WHERE last_name='Widenius' AND (first_name='Michael' OR
first_name='Monty');
SELECT * FROM test WHERE last_name='Widenius' AND first_name >='M' AND first_name
< 'N';
```

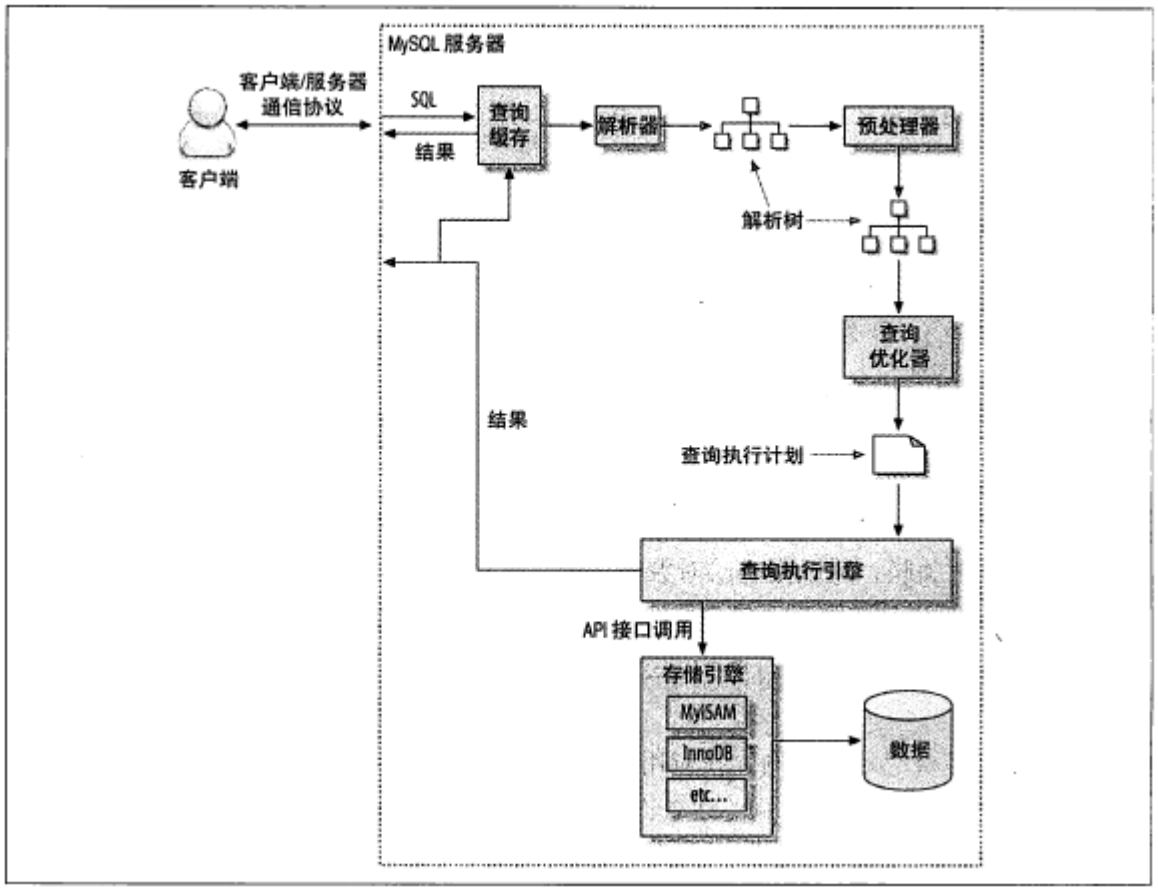
无用的多列索引：

```
SELECT * FROM test WHERE first_name='Michael';
SELECT * FROM test WHERE last_name='Widenius' OR first_name='Michael';
```

- 多列索引可以看成是把多个列拼接在一起后，再排序的数组，所以对于AND 过滤是有用的，而对OR过滤是无用的
- 多列索引的顺序对最终的索引有影响，索引首先是按照最左列进行排序

MySQL 的查询过程

MySQL 查询流程


MySQL 服务器
客户端/服务器
通信协议
客户端
SQL
结果
查询
缓存
解析器
预处理器
解析树
查询
优化器
结果
查询执行计划
查询执行引擎
API 接口调用
存储引擎
MyISAM
InnoDB
etc...
数据

MySQL 查询流程

1. 客户端发送一条查询给服务器
2. 服务器检查缓存，如果命中则直接返回
3. 服务器进行SQL解析，预处理
4. 交给优化器生成执行计划
5. 根据优化器生成的执行计划，调用存储引擎API执行查询
6. 结果返回

MySQL 查询流程 – 通信协议

- 半双工的方式通信，意味着任意时刻，要么服务端发送数据，要么服务端接受请求
- 服务端必须接收完整请求，客户端也必须接收完整结构
- 很多时候，客户端驱动会先读完数据，缓存到自己的缓存，然后应用层代码从本地驱动缓存读取，这样可能会导致性能降低，可以使用 unbuffered_read 来直接从服务端读取，这样能减少本地驱动层的内存缓存使用，针对特别大的数据结果集会有用

MySQL 查询流程 – 查询缓存

- 缓存是通过一个对大小敏感的HASH查找实现的，必须完整匹配
- 如果命中，仍然要检查权限，权限本身也是缓存的
- 如果都通过，会直接返回结果

MySQL 查询流程 – 优化处理

- 对关键字做SQL解析，生成一颗“解析树”
- 优化器会尝试多种查询执行方式，来预测执行计划的成本，因此语句要尽量帮助正确预测，否则可能会使用最糟糕的查询方案来执行，有很多的因素，比如并发、统计信息错误、索引预判错误等，导致优化预判不准确
- 优化器是非常复杂的组件，一般会对关联表的顺序、排序方法、MIN()、MAX()、COUNT() 等表达式做优化，比如 MyISAM 维护了一个变量来存放表的行数，因此 COUNT(*) 就能直接返回

MySQL 查询流程 – 错误的优化

```
SELECT * FROM film WHERE file_id IN
    (SELECT film_id FROM film_actor WHERE actor_id = 1)
```

我们认为的是会先执行子查询，得到结果后 film_id 集后，再执行主查询
但实际MYSQL是反的，会扫描表，然后每次调用子查询，再用当前的 film_id 与 子查询结果集的 film_id 过滤，考虑使用 EXISTS 或者关联查询

MySQL 查询流程 – 错误的优化

SELECT MIN（actor_id) FROM actors WHERE first_name = 'BECKHAM’

这样的查询，会全盘扫描，因为 first_name 并没有建立索引，然后再取回最小的 actor_id

SELECT actor_id FROM actors USE INDEX(PRIMARY) WHERE first_name = 'BECKHAM' LIMIT 1

因为 actor_id 是主键，默认是按照从小到大排序的，因此只要按照主键优先查询，LIMIT 1 就可以在几乎 O(1) 的时间内得到返回结果

MySQL 查询流程 – 优化处理

- 对关键字做SQL解析，生成一颗“解析树”
- 优化器会尝试多种查询执行方式，来预测执行计划的成本，因此语句要尽量帮助正确预测，否则可能会使用最糟糕的查询方案来执行，有很多的因素，比如并发、统计信息错误、索引预判错误等，导致优化预判不准确
- 优化器是非常复杂的组件，一般会对关联表的顺序、排序方法、MIN()、MAX()、COUNT() 等表达式做优化，比如 MyISAM 维护了一个变量来存放表的行数，因此 COUNT(*) 就能直接返回

MySQL 查询流程 – 执行及返回

- 执行阶段只是简单根据执行计划给出的指令逐步执行
- 如果查询结果可以被缓存，那么在执行完成后结果会放到查询缓存
- 执行过程中，临时数据集会放到临时结果表，例如关联查询得到的结果，会缓存到一个文件里，然后等到其它关联结果出来后，开始进一步合并、过滤
- 结果返回给客户端是一个增量的、逐步返回的过程，比如 SELECT 被命中的结果，如果不需要排序，那么从命中的第一条结果就开始返回给客户端

深翻页及优化

深翻页查询过程

数据库查询，会优先通过WHERE 条件过滤，如果没有设置 LIMIT 参数，会全部扫描全部数据；一般情况，如果不是 GROUP BY、SUM、COUNT 这一类的聚合操作，当 LIMIT 限制达到后就立即返回不再查询

如果没有LIMIT这样的限制，查询结果会被缓存下来，然后再写入到一个缓存文件，然后会对这样的缓存文件再进行归并

深翻页查询过程

对于神翻页的情况，例如下面的语句：

```
SELECT * FROM orders ORDER BY time DESC LIMIT 100000,20
```

查询的时候，会查询到前100020个结果，然后进行排序，再从第一个结果开始往后扫描，直到找到第 100000 个结果，此时向后取回20条结果

因为缓存的结果是不能随机访问的，必须得顺序扫描（类似于链表结构），因此整个耗时会非常长

深翻页优化

- 在查询的时候，要尽量避免这样的深翻页，一般人类的行为不会需要进行深翻页
- 如果不得不做深翻页，比如skip前800个结果，尽量考虑增加冗余列，例如用冗余列来标记当前结果的ID号，然后通过WHERE来帮助过滤，或者用时间来做偏移量的分割等，总之，使用参数来帮助过滤

```
SELECT * FROM orders where time > 418203977 ORDER BY time DESC LIMIT 20
```

疑问

- 问题答疑： http://www.xxwenda.com/
  - 可邀请老师或者其他人回答问题

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop