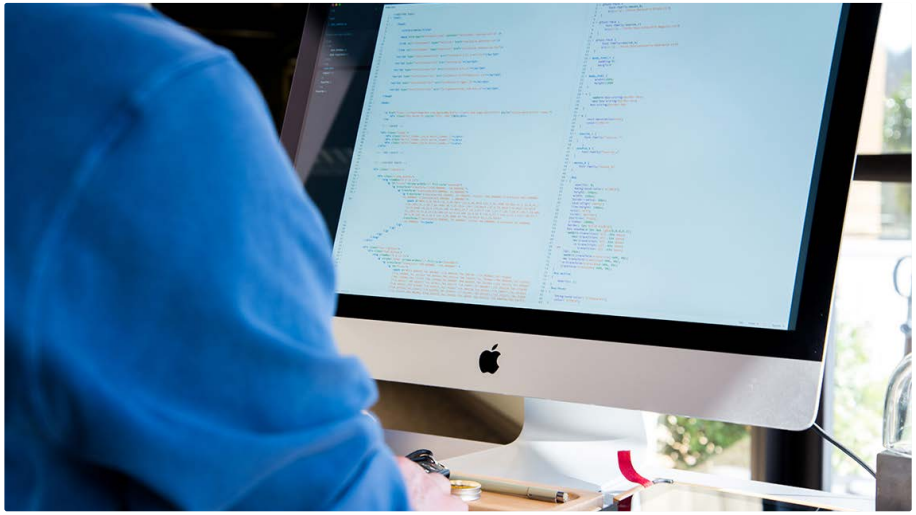
每个工程师都应该了解的：数据库知识

2017-12-08 朱赟



每个工程师都应该了解的：数据库知识
朱赟
- 00:23 / 14:18

数据库和编程语言一样，同样是软件工程师们的必争之地。今天我就和大家聊一些数据库相关的知识。

去年 Uber 发表了一篇文章，宣布他们从 PostgreSQL 转到 MySQL 了。文章的内容很好，同时还科普了一些数据库索引和复制的基本常识。当时，我转给了一个朋友，朋友看了之后说：“哦，他们 2013 年才发了一篇文章说他们从 MySQL 转到 PostgreSQL 。”

我找来朋友提到的那篇旧文，读过之后，大概理解了两篇转型文章背后的原因。

作为两大主流开源数据库，MySQL 和 PostgreSQL 的“战争”从未停止，虽然硝烟不如编程语言那么浓烈，但也是你来我往，剑影刀光。

如果去 Quora 或者 Stack Overflow 上搜索 MySQL V.S. PostgreSQL 这样的关键字，会出现一大堆帖子，大家各执一词众说纷纭，我的感觉则是：两者各有各的优势和使用场景，并不存在一种数据库对另一种压倒性优势。

对于大部分程序员来说，公司用哪个数据库，基本无需你去决定。加入一个公司的时候，除非是创业公司，或者你是 CTO、VP、总监级别的，否则大部分的技术选型早已应该尘埃落定。

尤其是数据库，一旦选择，再迁移的代价非常大。因此，除非有颠覆性的优势或者难以克服的问题，很少有公司会去费时费力做这种大的迁移。

不论是技术选型还是技术转型，其中不可忽略的因素是：你的工程师更容易驾驭哪一种技术，或者有话语权的决策者们倾向于哪一种技术。这一点其实和程序语言的选型有异曲同工之处。

类似 Uber 两次高调转型的事情，在我曾经工作过的 Square 公司也发生过。

Square 最早使用的是 MySQL，到了 2012 年，由于 PostgreSQL 的各种优势越来越突出 —— 比如对地理空间（Geospatial）数据和搜索的支持，当时几位资深工程师也开始大力倡导，很多新的服务就尝试性地使用 PostgreSQL。

那时候，公司的架构是 MySQL 和 PostgreSQL 并存的。对于我而言，这就有机会学习、掌握和比较两种不同的技术。

在我来看，两者各有特点，有些技术实现在 MySQL 里更方便一点，另一些则反之。无论哪种数据存储方式，总有方案可以解决问题，并没有觉得非要哪一种才行。

一个公司如果数据库从来不出问题，那一定是因为没有业务量或者流量。所有技术的选型和设计，都有它的应用场景，除去那些让人开心的案例，剩下的毫无疑问就是坑。

如何尽可能地避开这些坑，如何在出现问题的时候可以用最快的速度去修复，这些都是至关重要的因素。

大部分工程师并不是数据库专家，在 Square 公司两种数据库并存的期间，PostgreSQL 的牛人寥寥无几，但是 MySQL 却有几个专家是极为靠谱的。对我们而言 ，PostgreSQL 和 MySQL 的相对优势，都比不过出问题的时候有人救火和解惑来的重要。

另外，一个公司维系两套同类的数据库系统本身就是个负担，因为这些原因，那些使用 PostgreSQL 的服务慢慢就转成了 MySQL 。

我们支付类的技术工作需要强事务和一致性的支持，所以 NoSQL 类型的数据库用得比较少，主要使用的是 MySQL 或者 PostgreSQL。由于在工作中常常与数据库打交道，我也逐渐了解到不少相关的知识和技术，但是线上真的出了问题，我还是没有把握自己去搞定。

好在，每个公司都会有一些专攻数据库的大牛，这种专门的职位便是 DBA，有的中小创业公司没有专职 DBA，数据库便由做运维的人维护。我和几位这样的牛人私交甚好，加之平时自己处理起系统中出现的相关问题，也会常常请教他们，一来二去就知道了很多有趣的数据库知识。

对数据库专家我一向是敬佩的态度，他们的价值不可小觑。公司只要稍具规模，如果数据库这块做不好，基本也就没什么可玩的了。数据库可以说是互联网公司最宝贵的资产，这块不出问题也罢，一出问题，即是见血封喉，服务直接宕机。

关于数据库，最常见的问题都有哪些呢？

第一个是选型

因为每个公司的业务不同，数据库系统的应用场景不一样，选型也会不尽相同，但可以肯定的是，没有哪个系统一定是最好的。

比如做支付业务一定要强事务性、一致性的支持，很多社交平台更多时候需要的是高可用；有的业务写操作特别重，有的业务更重要的是读操作；有的业务可能只关心最近几天的数据，于是可以容忍老数据读写的低效，有的却要频频访问历史数据，需要读写的高效；有的业务可以通过加索引解决查询效率，有的却只能通过加缓存等等。

这就是为什么很多公司会选择多个数据库系统并存，通过不同的技术和架构给予相关的业务场景最优支持。如果一旦选型失误，便不会有频频踩坑一说，因为这基本就算直接掉进了坑里。

第二个是数据库相关的架构

什么意思呢？这里的架构包括数据库上层的缓存系统设计，程序语言对数据库连接的处理，代理层（Proxy Layer）的功能，以及和二进制日志（Binlog）等相关的数据管道（Data Pipeline）的搭建。当然，其中也包括了数据库系统的分区、备份等的具体设计。

很多公司早期所有的表都在一个数据库里面，因为各种连接池（Connection Pool）和吞吐量（Throughput）的限制，基本没法做扩展。能够合理地设计数据库表的分离，把数据相关的放在一起，不那么相关或甚至不相关的放在另一个数据库里。这些看起来很简单的做法，很多时候却可以很大程度上缓解可扩展的问题。

第三个，也是我们平时遇到最多问题的：人为错误

再好的系统，使用姿势不对也是枉然，更何况并不是所有的工程师都是数据库专家，所以人为的错误是最常出现的问题。

人为错误分成两种。 一种是操作数据库时犯的错误。另一种是程序员在程序里或者脚本里犯的错误。

操作数据库时犯的错误的概率比较低，但危害却最大。几乎所有的公司都会有类似的传奇时间，我听过的就有三种误操作的版本。

第一种是工程师无意或有意，“不小心”删掉了数据库核心表中所有的数据。这不是段子，在 Facebook 就曾经发生过类似的事情，当事人还是我的一个朋友，好在后来恢复了，这事儿便也成了他的工程师历史上光辉的一笔。

第二种工程师在线修改表结构（ online schema change ）的时候，不小心一步误操作，结果数据库被锁（ Lock ）长达几个小时，该公司网站也就挂了好几个小时。

第三种是听国内一个大公司的朋友说的，细节还已经无法还原。只记得听说他们的两台服务器，在做主从切换的时候，拔错了一个电源插头，然后……就没有下文了。

程序员在程序里或者脚本里犯的错误就很常见了。

举个简单的例子，我们知道 Ruby on Rails 对数据库的操作基本是通过 Active Record 来完成的。Active Record 可以通过一个数据库连接池来限制每个应用到数据库的连接。

如果某一个程序或者脚本查询没有索引的数据，导致全表扫描，再加上一些网页服务器（Web Server）的并行访问，经常会有整个数据库的所有链接被占用的情况，连终止查询（Kill Query ）都没法执行，只能人肉去做一些类似重启的物理操作。还有更常见的，就是程序里的 N+1 查询问题。面对这类问题，可以使用数据库的连接查询功能，比如 left outer join 来避免 N+1 的问题。

最后一个是数据库访问瓶颈

只要是数据库，就有吞吐量的限制，而数据库访问瓶颈便是自然流量增长或者流量突增造成的。只要你的业务在增长，总有一天数据库访问就会达到一个上限。在这个预警到来前，你需要做各种垂直或水平扩展来提升这个上限，或者，你可以通过缓存和其他机制来对访问量进行分流，这里面可以做的工作就多了。

流量的突增一般是类似分布式拒绝服务（DDoS）或市场活动带来的，也可能是因为某个黑天鹅事件造成的，这些原因都很难预料。

如果是有计划的市场活动，就需要提前做好各种战斗准备。如果是恶意攻击，那就只能靠各种防御工程，如 IP 阻塞（IP Blocking）或者第三方的高防系统挡掉这些流量来保证数据库的正常工作。

研发过程中，有哪些与数据库相关的问题呢?

我们以 MySQL 为例，讲讲日常开发中应该注意的，与数据库相关的问题。

索引

创建索引通常是为了提高常用查询语句的性能，将某些列以特定的数据结构（常见的如 B-Tree ）有序存储起来。维持这样的一个数据结构在写数据的时候会有一些系统开销（Overhead）。但如果查询确实是高频的，那么这样的系统开销就很划算。在建表时需要考虑所有可能的高频查询，另一方面，忌讳过度地“为未来设计 "(Design for the Future)，也就是加一堆可能根本不常用的索引，反而增加了写数据时候的成本和负担。

索引另一个常见的用途就是保证某一列或者某几列的组合是唯一的（ Unique ），这也称为唯一性索引（Unique Indexing），在写业务逻辑代码的时候会常常用到。

比如你有一个用户表（ User Table ），你想让所有用户（ User ）的电子邮件都是唯一的，这个时候用唯一索引（ Unique Indexing ）就很方便。不过唯一索引（ Unique Indexing ）和可选列（ Optional Column ）组合在一起的时候，也有很多需要注意的地方。

比如，你想对列X做唯一索引（Unique Index Column X），过了一段时间，也许有些情况下列X（Column X）并不唯一，我们便把索引改成了对列X和列Y作唯一索引（ Unique Index Column X + Column Y），但是列Y（Column Y）是 Nullable 的。

这个时候会出现什么情况呢?

你会有多条记录，有着一样的 X 值，以及 Null 的 Y 值。很意外对吧，原因就是 Null 在数据库里常常解释为 “不确定” 而不是空。

事务支持

还有一个比较重要的问题就是数据库的事务支持（Transactional Support）。简单说来，就是利用数据库本身提供的事务性，来封装一系列需要同时完成的动作。

比如在一段事务里面，先执行X，再执行Y (Transaction do X；Y；end)，如果 X 和 Y 都是数据库写操作，那要么两个写操作都成功，要么都失败。也就是说，对数据库的改动会统一把事务所做的修改提交到数据库（ Commit），而提交（ Commit ）前的任何错误都会触发所有更新回滚到开始的状态（ Rollback ）或引发不正常进程的终止 （ Abort ）。

虽然正确使用事务支持（ Transactional Support ）会很方便，但是也常常见到过度使用让代码变得很脆弱甚至是出现 Bug 的情况。

常见的几种情况如下。

1. 事务（Transaction ）中封装的代码逻辑太长太复杂，甚至调用了别的函数。很多时候，很难去推理当执行中抛出异常的话，到底哪些会回滚，哪些会产生遗留影响。
2. 事务中封装了与数据库改动无关的逻辑。
3. 事务中有不可逆的操作，例如发送电子邮件给用户，发布（Publish）到一个Job队列（Queue）等。这种情况会导致系统的不一致。比如，一个写操作被回滚了，但这条数据相关的 Job 还是被加入到队列了，就会引发错误。
4. 事务中包括了在不同数据库里面的事务，也就是分布式事务，这需要单独处理。
5. 事务中嵌套了事务，不同情况可能会有不同的结果，如果没搞清楚，就可能出现意外的行为。

更多情况就不一一列举了，但过度使用事务支持往往会让逻辑变得不必要的复杂。

数据库锁

数据库会出现 Race Condition，我们常常把 Race Condition 叫做竞争条件，是指多个进程或者线程并发访问和操作同一数据，且执行结果与访问发生的特定顺序有关的现象。

如何解决竞争条件（ Race Condition ）呢？常见的方法是使用各种锁机制来确保行为的可预测性和正确性。根据实际情况的不同，加锁的方式会不一样。

常见的有乐观锁定（ Optimistic Locking ）和悲观锁定（ Pessimistic Locking）。总的说来，前者在对性能要求比较高的系统里更为常见。在实际应用中，很多系统都会自己实现锁定（ Locking ）机制。

缓存和主从机制

为了提高性能，我们会为数据库增加缓存（ Caching ）和主从（ Master - Slave ）等机制，这有时候会引起数据的不一致性。常见的情况是，如果系统默认是在从节点（ Slave ）读数据，那么一些刚刚更新到主节点（ Master ）的数据在读的时候就有可能读不到。这个情况在使用一些数据关联 （Association）的时候更容易读不到。Rails 的 Active Record 数据关联 （Association），就很容易出现这一类的问题。

今天跟大家介绍了不少数据库相关的基础知识，如果你是个软件工程师，想必这些内容都已经耳熟能详，我们来总结一下。

1. 本文从 MySQL 和 PostgreSQL 的迁移和选型入手，介绍了数据库的技术特点和选型问题。在我眼里，没有更好的技术，只有更适合的技术。
2. 数据库领域会碰到哪些问题呢？我为大家介绍了数据库选择、数据库相关的架构问题、人为失误的问题，还有数据库遭遇流量瓶颈以及相关的应对方式。
3. 几乎每个工程师在编程的时候都会和数据库打交道，研发过程中我们应该注意什么问题呢？在这个章节我们讲了索引、事务支持、数据库锁、缓存和主从机制。

文中每个点都可以深入展开，独自梳理，最终形成系统的知识储备。因为篇幅所限，不能涉及所有的技术细节，文中提到的内容都是我在工作中遇到过的问题和实战经验，希望对你有帮助。

如果你有不同的想法，更好的观点，请在留言中告诉我，互通有无，一起成长。

参考链接：https://eng.uber.com/mysql-migration/

一路向北

2017-12-09

用过数据库，但是没有怎么去理解过。读过之后，醍醐灌顶

shibo

2017-12-08

非常干货！！！！安姐以后是不是可以在billing搞个tech talk呀 lol

Silence

2017-12-08

安姐的这篇文章很实用，Mark了！

Geek_c8eb71

2018-08-03

数据库的初级原理

RAY

2018-04-11

适合初级科普

Dylan

2017-12-29

前两天就刚刚在生产的数据库上直接进行操作～由两个资深的数据库工程师一起再三核对脚本，才敢最后运行～真的是后怕

kimi

2017-12-16

安姐什么时候说一下分布式事务，以及你曾工作过的两个公司在支付环节如何实现分布式事务

公剑 | Gong Jian

2017-12-14

MySQL　lock

安姐，在线修改大表的表结构，　　会　　住，有什么好的解决方案吗？

AlphaGo

2017-12-08

嗯，就在刚刚上线一个核心存储过程的变更。此存储过程所以请求都会涉及到。然后性能与原先的有所变差，导致API请求积压并超时��。Rollback后，写了个Incident Report。现在在BART等车回家...

---

simaopig

2017-12-08

刚参加工作时，公司就流传一个大神误操作删库的传说。