腾讯二面：1.2 亿级大表, 如何 加索引？

原创 小北架构师团队 程序员江小北 2025年03月24日 08:30

说在前面

当面试官甩出这个问题时：

> "你们怎么给亿级用户表加索引的？线上直接执行ALTER TABLE？"

90%的程序员会犯的致命错误：

❌ 在业务高峰时段硬上弓

❌ 以为INPLACE算法绝对安全

❌ 没配置innodb_online_alter_log_max_size

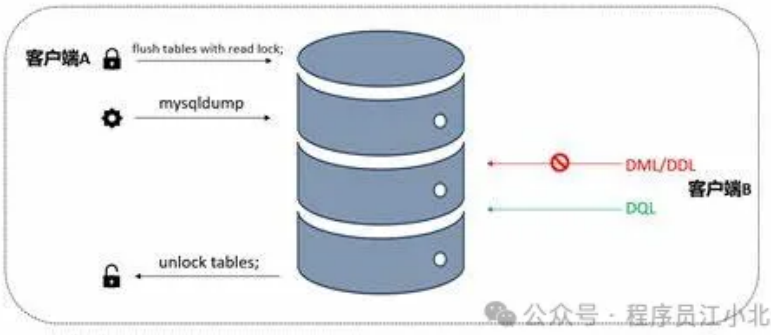
去年双十一前夕，某酒店商旅平台核心支付表突发死锁，直接导致支付链路瘫痪30分钟。

复盘发现：**运维同学在晚高峰给1.2亿行大表加索引，触发全局锁等待**。

你以为的加索引：

```
ALTER TABLE user ADD INDEX idx_phone (phone);  -- 轻轻松松3秒完成
```

实际生产环境的可能加索引：



所以，今天给大家做一下系统化、体系化的梳理，使得大家可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现"offer直提"。

如何判断什么时候可以给大表加索引？

随着业务的快速发展，像 `user` 这样的用户表数据量不断增加，可能你已经注意到查询速度变慢了，尤其是某些 SQL 变成了“慢查询”。

这时，你想通过加索引来提升查询性能。但别急，**给大表加索引可不是轻松事**，一不小心可能会让数据库“崩溃”。

因为索引的创建是DDL（数据定义语言）操作，它有可能导致锁表，阻塞其他操作，影响业务。

所以，怎样才能安全地给大表加索引？这就得好好了解一下。

两种索引构建方式

MySQL 在构建索引时，有两种方式：**在线模式**和**离线模式**。

- **在线模式（Online DDL）**：
  这种方式可以在构建索引的同时，数据库仍然可以进行读写操作，**对业务的影响较小**。但不是所有数据库版本和存储引擎都支持这个模式。比如，如果你的 MySQL 是 InnoDB 存储引擎的较新版本，可以通过一些设置启用在线模式。**不过在线模式会消耗更多的系统资源**，这点要注意。

- **离线模式（Offline DDL）**：
  这种方式在构建索引时会锁住表，**不能进行任何读写操作**，直到索引创建完成。虽然这种方式操作简单直接，但会对业务造成比较大的影响，适用于可以停机的场景。如果你的数据库能忍受一定的停机时间，或者是在业务低峰期加索引，离线模式也是一个选择。

DDL 和 DML 小知识

先给大家补充下基础知识：

- **DDL（数据定义语言）**：主要用来定义或修改数据库结构，比如创建、删除表，创建或删除索引等操作。
- **DML（数据操作语言）**：主要用来操作数据库中的数据，像插入（INSERT）、更新（UPDATE）和删除（DELETE）这些操作。

索引属于 DDL 操作，它是对数据库结构的修改，所以需要特别小心。

早期DDL操作：离线模式（Offline）

如果你有一个数据量达到千万级别的表，想给它加个索引，怎么做才不崩溃？这其实和你使用的数据库版本有关系。下面我们就来看看 MySQL 5.6.7 之前，如何通过离线模式来处理大表索引创建。

MySQL 5.6.7 之前的 **早期DDL** 操作，主要有两种方式：**copy table** 和 **inplace**。这两种方式都会锁表，**禁止其他操作**，直到索引创建完成。

方式 1：copy table 方式

1. **创建临时表**：首先，创建一个和原表一样的临时表，并在临时表上添加索引。
2. **锁定原表**：原表会被锁住，**不允许修改数据**（只允许查询）。

3. **逐行复制数据**：将原表的数据逐行拷贝到临时表中，注意，复制过程中数据不会重新排序。
4. **升级锁**：复制完毕后，原表升级锁，**不允许任何读写操作**，直接暂停业务。
5. **重命名表**：最后，临时表会被重命名为原表的名字，原表则被重命名为临时名字。

举个例子，假设你有一个 `orders` 表，要给 `order_date` 字段加一个名为 `idx_order_date` 的索引。你可以这么做：

```
ALTER TABLE orders ADD INDEX idx_order_date (order_date), ALGORITHM=COPY;
```

这里， `ALGORITHM=COPY` 表示使用 `copy table` 方式来执行操作。数据库系统会创建一个临时表，添加索引，然后把原表的数据复制到新表里，最后再进行重命名。

方式 2：inplace 方式（快速索引创建，仅支持索引创建与删除）

1. **创建临时文件**：首先会创建一个 `frm` 文件，用来存储表的结构信息。
2. **锁定原表**：和 `copy table` 一样，原表会被锁住，**不允许修改数据**（只允许查询）。
3. **构建索引**：数据库根据聚集索引的顺序，直接在原表上添加新的索引项。
4. **升级锁**：当索引构建完成后，原表会被升级为锁定状态，**不允许任何读写操作**。
5. **重命名文件**：最后，原表的 `frm` 文件会被替换，完成索引创建。

如果你希望对业务影响最小，通常使用 `inplace` 方式。继续用上面的 `orders` 表，代码如下：

```
ALTER TABLE orders ADD INDEX idx_order_date (order_date), ALGORITHM=INPLACE;
```

在这个例子中， `ALGORITHM=INPLACE` 表示使用 `inplace` 方式来执行操作。数据库会直接在原表上构建索引，而不是复制数据，通常这种方式效率更高，尤其是在不需要数据复制的情况下。

Copy VS Inplace 两种方式的对比

- **copy table 方式**：通过创建一个临时表来添加索引，适用于大表，但对业务影响较大。
- **inplace 方式**：直接在原表上添加索引，避免复制数据，速度更快，适合减少业务中断的场景。

MySQL5.6.7之前，如何在线为大表添加索引？

在 MySQL 5.6.7 之前，MySQL 的 DDL 操作机制比较简陋，不能很好的支持在大表上添加索引而不影响业务。为了在不锁表的情况下添加索引，通常有两种在线模式的方案：

- **影子策略**
- **pt-online-schema-change**

这两种方法都是通过外部工具来实现的，不直接依赖 MySQL 本身。

方式一：“影子策略”

在 MySQL 5.6.7 及之前的版本，由于 DDL 操作的局限性，通常需要用“影子策略”来保证 DML 操作的正常进行。影子策略的核心思想是：在不影响原始表的情况下，创建一个和原表完全相同的副本（影子表），然后对影子表进行索引操作，最后再将数据同步回原表。

“影子策略”操作步骤：

1. **创建新表**：先创建一个和原表结构相同的新表（比如 `tb_new`）。
2. **在新表上加索引**：在新表上添加索引。
3. **重命名表**：将原表重命名为 `tb_tmp`，然后把新表 `tb_new` 重命名为原表名 `tb`。
4. **为原表加索引**：接着，为重命名后的原表（`tb_tmp`）加索引。
5. **交换表名**：最后，把新表恢复为原表名（`tb`），而原表 `tb_tmp` 改回临时表名。
6. **数据同步**：在新表的索引完成后，将新表的新增数据同步回原表。

影子策略的优点：

- **最小化业务影响**：在整个索引添加过程中，原表依然能进行增删改查（DML操作），业务不受影响，**可以在线进行操作**。

影子策略的缺点：

- **数据一致性问题**：在新增索引期间，原表的增删改操作可能会产生一些数据变动，导致数据不一致。比如，原表新增或删除的数据在影子表中没有反映过来，这可能需要额外的同步操作。

方案二：pt-online-schema-change 工具

Percona 提供了一些非常实用的 MySQL 工具，其中 **pt-online-schema-change**（简称 pt-osc）就可以用来相对安全地对大表进行 DDL 操作。它的工作原理和“影子策略”有点相似，但它通过触发器来保证数据同步的一致性，解决了影子策略中可能存在的数据丢失问题。

pt-online-schema-change 工作原理

1. **创建新表**：首先，创建一个结构与原表完全相同的新表。
2. **在新表上加索引**：然后，在这个新表上进行索引操作等DDL操作。
3. **添加触发器**：在原表上加上三个触发器（DELETE、UPDATE、INSERT），这样对原表的增删改操作，也会同步到新表。
4. **数据复制**：将原表的数据分成小块（chunk）复制到新表中，逐步迁移。
5. **交换表名**：当数据同步完成后，把原表重命名为 `old` ，新表重命名为原表的名字。
6. **清理操作**：删除旧表和触发器，操作完成。

pt-online-schema-change 的优点

- **最小化业务影响**：由于它是在线操作，**在加索引的过程中，原表的增删改操作可以继续进行**，业务不受影响。
- **数据一致性保障**：新表上的索引在创建期间，原表的增删改通过触发器同步到了新表，避免了数据丢失的问题，保证了数据一致性。

pt-online-schema-change 的问题

- **必须有主键**：表必须有主键，否则会报错。
- **不能有触发器**：原表不能已经有触发器。
- **资源消耗**：虽然它尽量减少对业务的影响，但在数据复制和同步阶段，还是会消耗一定的系统资源，特别是 **CPU、磁盘 I/O 和内存**。对于大型表，这个过程会比较耗时，可能会对数据库性能造成一定影响。所以，最好在负载较低的时段进行操作。

小结

`pt-online-schema-change` 本质上是"影子策略"的一种改进版，它通过触发器来保证原表和新表之间的数据一致性。

方案三：MySQL5.6.7之后的内部 Online DDL

在 MySQL 5.6.7 之前，DDL 操作受限，通常需要外部工具如 **影子策略** 或 **pt-online-schema-change** 来进行在线修改。

自 5.6.7 起，MySQL 引入了 **内部 Online DDL** 特性，**无需外部工具**，即可高效地在不中断业务的情况下修改大表，5.7 版本中该特性得到了进一步优化。

MySQL5.6.7 Online DDL 的三个阶段

MySQL 5.6.7 版本的 Online DDL 操作主要分为三个阶段：

1. **Prepare 阶段**
   MySQL 创建临时的 **frm 文件**，并获取 **MDL 写锁**，暂停所有读写操作。根据 `ALTER TABLE` 类型

选择执行方式。如果是增加索引，使用 **Online-Rebuild**，在原表上重建索引，允许 DML 操作继续。

2. **执行阶段**
   将 **MDL 写锁** 降级为 **MDL 读锁**，允许 DML 操作，禁止 DDL 操作。MySQL 记录增量 DML 操作到 **row_log**，扫描原表和新表的索引，更新新的索引树。

3. **Commit 阶段**
   完成索引操作后，重做增量 DML 操作，更新数据字典，提交事务，重命名临时文件为原表，释放 MDL 写锁，操作结束。

通过这三个阶段，MySQL 实现了在线 DDL 操作，能在不影响当前业务的情况下修改大表。

MySQL5.6.7 Online DDL 如何保证数据一致性

在进行 Online DDL 操作时，MySQL 会扫描原表的聚集索引，将每一条记录的索引项刷入新的索引树中。

然而，在扫描的过程中，原表的数据可能会发生 DML 变更（比如插入、更新或删除），这就可能导致新的索引树和原表的数据不一致。

> 那么，MySQL 是如何保证数据一致性的呢？

MySQL 通过使用 **row log**（行日志）来解决这个问题。

在 DDL 操作期间，MySQL 会记录所有 DML 操作的 **Row Log**，以确保数据的一致性和完整性。

这样，如果在执行 DDL 操作时，原表发生了变化，MySQL 会通过这些 Row Log 来更新新的索引树，确保最终的数据是准确一致的。

Row Log 核心结构：row_log_t

在线 DDL 过程中，Row Log 的核心结构是 **row_log_t**。

这个结构会在 MySQL 内部维护，存储在 **row0log.cc** 文件中。简单来说，**row_log_t** 就像一个缓存，记录下 DML 操作，在 DDL 完成后，可以根据这些日志来同步数据。

这种机制类似于 **gh-ost** 工具，只不过 **gh-ost** 是通过 binlog 进行 DML 操作回放，而 MySQL 内部则是通过 **row_log_t** 来维护和回放这些操作，确保数据的一致性。

MySQL5.6.7 Online DDL 在线添加索引的案例

在了解了 **row log** 的原理后，接下来，来看看 MySQL 在 InnoDB 引擎下如何添加索引的过程。

Prepare 阶段

1. **创建临时文件**：首先，MySQL 会创建一个新的 **frm** 文件，这是表的结构文件。
2. **加锁**：MySQL 会获取 **EXCLUSIVE-MDL 锁**，这时数据库会禁止所有读写操作。
3. **选择执行方式**：根据 `ALTER` 的操作类型，选择执行方式（比如 `copy` 、 `online-rebuild` 或 `online-norebuild` ）。对于新增二级索引，MySQL 会选择 `online-rebuild` 。
4. **更新数据字典**：MySQL 会更新数据字典，标记该表索引的状态为 `ONLINE_INDEX_CREATION` ，表示该表正在执行在线 DDL 操作。
5. **分配 Row Log**：MySQL 会分配一个 **row_log** 对象，用来记录增量 DML 操作的日志。
6. **生成副本 ibd 文件**：根据原表的 **ibd 文件**，生成新的临时副本 **ibd 文件**。

DDL 执行阶段

1. **降锁**：MySQL 会将 **EXCLUSIVE-MDL 锁** 降级为 **MDL 读锁**，此时可以进行读写操作，但 DDL 操作仍然被禁止。
2. **扫描原表记录**：MySQL 会扫描原表的聚集索引，逐条读取数据，并将新表的聚集索引和二级索引逐一处理。
3. **构建索引项**：根据读取的每条记录，构建相应的索引项，然后将这些索引项插入到 **sort_buffer** 块中，并进行排序，最终更新到新的索引树上。
4. **处理增量数据**：在构建索引的过程中，所有的 DML 操作日志会被写入 **Row Log**。这些日志会在 DDL 执行过程中被重放，确保数据同步到新的索引树。

Commit 阶段

1. **加锁**：MySQL 会重新获取 **EXCLUSIVE-MDL 锁**，暂停所有读写操作。
2. **重做增量数据**：MySQL 会重做 **Row Log** 中最后一部分增量操作。
3. **更新数据字典**：更新 InnoDB 的数据字典，确保数据结构的一致性。
4. **提交事务**：MySQL 提交事务，并刷写 **redo 日志**，确保所有数据持久化。
5. **更新统计信息**：修改索引统计信息，以确保查询优化器能正确选择索引。
6. **重命名文件**：最后，MySQL 会将临时的 **ibd 文件** 和 **frm 文件** 重命名为原表的文件名，完成索引的添加。

1000W级大表的 Online DDL 性能问题

在处理 1000W 级别的大表时，进行 **Online DDL** 操作会面临一些性能问题，主要体现在 **CPU** 和 **磁盘开销** 上：

1. CPU 开销大

当进行 **Online DDL** 时，MySQL 会循环遍历原表的聚集索引，读取每条记录。如果表的数据量很大，这个过程会消耗大量的 CPU 资源。长时间占用 CPU，可能导致 MySQL 的连接数下降，**并发处理 DML 操作的能力降低**，从而影响业务的处理速度。

2. 磁盘开销大

在进行排序操作时，MySQL 会用到磁盘临时文件来存储归并排序的数据。如果待排序的数据量非常大，就会产生大量的磁盘 I/O。**当查询的结果没有完全加载到内存中，且 `buffer pool` 已满时**，会触发磁盘刷脏操作，这时查询响应速度会变慢，甚至导致查询请求等待磁盘刷脏完成。

怎么解决这些问题？

针对上述性能问题，我们可以采取以下方法：

1. 评估数据量

首先需要评估表中的数据量。如果数据量较小，问题就不严重，影响也相对较小。如果数据量非常大，那就需要更加谨慎。

2. 监控 CPU 使用率

观察 MySQL 的 CPU 使用情况。如果 CPU 使用率过高，应该在 **低峰期** 执行 DDL 操作，尽量减少对业务的影响。**在低峰期执行 DDL，可以避免高并发操作时性能下降**。

目的	解决方法
减少业务影响	调大 `innodb_sort_buffer_size`，降低磁盘 I/O
避免 DDL 过程中写 Row Log 溢出	调大 `innodb_online_alter_log_max_size`
一定要在高峰期做 DDL	建议使用第三方工具，比如 `gh-ost`，通过 binlog 完成 DDL，避免扫描聚簇索引带来的 CPU 开销

如果一定要在高峰期做 Online DDL，怎么办？

如果一直都是高峰期，是否就不能做索引添加了呢？当然不是！这时候可以使用 **gh-ost** 工具进行在线 DDL 操作。

使用外部工具 gh-ost 做 Online DDL

gh-ost 是一个 **无阻塞** 的在线表结构迁移工具，类似于 **pt-online-schema-change**，不过它通过 **binlog** 进行数据迁移，避免了传统的锁表问题。gh-ost 的工作方式是利用 **异步迁移** 和 **分块处理** 来减少对数据库性能的影响。

gh-ost 工作原理

1. **创建幽灵表**：gh-ost 会创建一个结构与原表相同的幽灵表。你想修改表结构（比如增加新列、修改列类型等），这些修改都会反映到幽灵表。
2. **数据迁移与同步**：通过解析 MySQL 的 **binlog**，gh-ost 会把原表的插入、更新、删除操作同步到幽灵表。这是一个异步的过程，原表可以继续读写。
3. **切换表操作**：当幽灵表的数据和原表的数据基本同步后，gh-ost 会重命名原表为中间表，然后把幽灵表重命名为原表名，完成表结构的变更。

gh-ost 优势

1. **高效迁移与同步**：通过解析 **binlog** 来同步数据，这种方式 **不会阻塞原表的读写操作**，特别适用于高并发场景。
2. **灵活的分块策略**：gh-ost 会将大表分成小块逐步迁移，减少对内存和磁盘 I/O 的压力，避免一次性迁移大批量数据导致性能问题。
3. **对性能影响小**：在整个迁移过程中，原表的读写操作不会被长时间阻塞，虽然切换表时可能会有短暂影响，但影响较小，适合高负载环境。

gh-ost 局限性和注意事项

1. **依赖二进制日志**：gh-ost 强烈依赖 MySQL 的二进制日志，如果二进制日志配置不正确或损坏，gh-ost 将无法正常工作。
2. **复杂环境的风险**：在有大量存储过程、视图或外键约束的复杂数据库环境中，gh-ost 可能会遇到问题，尤其是涉及外键关系的表结构变更。
3. **资源消耗**：尽管 gh-ost 对性能的影响较小，但它在迁移过程中仍然会消耗 CPU、磁盘 I/O 和内存，尤其是对于大表，需优化迁移速度和分块大小。

gh-ost 和 pt-online-schema-change 的区别

gh-ost 和 **pt-online-schema-change** 都是常用的在线 DDL 工具，它们主要的区别如下：

1. **触发器使用**：
   - **gh-ost**：不使用触发器，而是通过解析 **binlog** 捕获数据变更，这样可以更灵活地控制迁移过程，避免性能瓶颈。
   - **pt-online-schema-change**：使用触发器来捕获数据变更，这在高并发场景下可能会影响性能。
2. **控制迁移过程**：
   - **gh-ost**：通过 **binlog** 迁移，可以暂停迁移，控制写负载，减轻数据库主负载。
   - **pt-online-schema-change**：触发器的方式让迁移过程更紧密地和原表的操作绑定，可能导致性能问题，特别是大表或高负载环境下。

简而言之，gh-ost 通过使用 **binlog** 实现高效的异步迁移，并通过分块处理减少了系统资源的冲击，适合高并发和大表操作的场景。

而 pt-online-schema-change 则是通过触发器同步原表数据，可能在高并发时带来性能问题。

二者对比：gh-ost vs pt-online-schema-change

在进行在线 DDL 操作时，**gh-ost** 和 **pt-online-schema-change** 都是常用工具，它们各有优劣，选择哪个工具取决于你的具体需求。

下面我们来简明扼要地对比一下这两者。

1. 同步性：

- **gh-ost**：采用 **异步** 方式，通过解析 **binary log** 将变更应用到幽灵表。这种方式虽然效率高，但可能会增加网络流量，且需要 **行复制（row-based replication）** 来确保数据一致性。
- **pt-online-schema-change**：采用 **同步** 方式，通过触发器将变更同步到新表。这可能会在高负载时影响数据库性能。

2. 对复制的影响：

- **gh-ost**：不使用触发器，因此对复制的影响较小，迁移过程可以暂停和恢复，控制性更强。
- **pt-online-schema-change**：使用触发器，可能会增加主从延迟，尤其在高负载时更为明显。

3. 负载控制：

- **gh-ost**：支持动态控制，允许根据 **MySQL 的实时指标** 调整迁移行为，比如设置线程运行阈值，以控制迁移过程中的负载。
- **pt-online-schema-change**：同样支持负载控制，但需要在迁移过程中停止并重新配置，这增加了操作的复杂度。

4. 支持的场景：

- **gh-ost**：不支持外键、触发器和 **Galera Cluster**（因为它使用了 **LOCK TABLE** 进行表切换）。
- **pt-online-schema-change**：支持外键、触发器等复杂场景，且可以在 **Galera Cluster** 上使用。

5. 性能影响：

- 在一些测试中，**gh-ost** 的性能开销几乎可以忽略不计，而 **pt-online-schema-change** 可能会导致 **12%** 的性能下降。不过，在其他场景下，**pt-online-schema-change** 可能表现得更好。因此，选择工具要根据具体的数据库环境和业务需求来决定。

如果觉得有帮助，欢迎点击“在看”和“赞”！

第3版：互联网大厂面试题

包括 Java 集合、JVM、多线程、并发编程、设计模式、算法调优、Spring全家桶、Java、MyBatis、ZooKeeper、Dubbo、Elasticsearch、Memcached、MongoDB、Redis、MySQL、RabbitMQ、Kafka、Linux、Netty、Tomcat、Python、HTML、CSS、Vue、React、JavaScript、Android 大数据、阿里巴巴等大厂面试题等、等技术栈！

阅读原文： 高清 7701页大厂面试题 PDF

阅读原文