**MySQL篇**

1. **MySQL中，如何定位慢查询？**

**候选人：**

嗯，我们当时在做压力测试时发现有些接口响应时间非常慢，超过了2秒。因为我们的系统部署了运维监控系统Skywalking，在它的报表展示中可以看到哪个接口慢，并且能分析出接口中哪部分耗时较多，包括具体的SQL执行时间，这样就能定位到出现问题的SQL。

如果没有这种监控系统，MySQL本身也提供了慢查询日志功能。可以在MySQL的系统配置文件中开启慢查询日志，并设置SQL执行时间超过多少就记录到日志文件，比如我们之前项目设置的是2秒，超过这个时间的SQL就会记录在日志文件中，我们就可以在那里找到执行慢的SQL。

2. **那这个SQL语句执行很慢，如何分析呢？**

**候选人**：如果一条SQL执行很慢，我们通常会使用MySQL的EXPLAIN命令来分析这条SQL的执行情况。通过key和key\_len可以检查是否命中了索引，如果已经添加了索引，也可以判断索引是否有效。通过type字段可以查看SQL是否有优化空间，比如是否存在全索引扫描或全表扫描。通过extra建议可以判断是否出现回表情况，如果出现，可以尝试添加索引或修改返回字段来优化。

3. **了解过索引吗？（什么是索引）**

**候选人**：嗯，索引在项目中非常常见，它是一种帮助MySQL高效获取数据的数据结构，主要用来提高数据检索效率，降低数据库的I/O成本。同时，索引列可以对数据进行排序，降低数据排序的成本，也能减少CPU的消耗。

4. **索引的底层数据结构了解过吗？**

**候选人**：MySQL的默认存储引擎InnoDB使用的是B+树作为索引的存储结构。选择B+树的原因包括：相比于红黑树，节点可以有更多子节点，路径更短；相比于b树，只有在叶子节点才存储数据，磁盘读写代价更低；而且B+树适合范围查询和扫描，因为叶子节点形成了一个双向链表。

红黑树和AVL树之间的区别在于，红黑树允许局部不平衡，这使得红黑树的查找效率要略低于AVL，但是插入和删除的效率高于AVL，但是基本也都是一个数量级。

5. **B树和B+树的区别是什么呢？**

**候选人**：B树和B+树的主要区别在于：

1. B树的非叶子节点和叶子节点都存放数据，而B+树的所有数据只出现在叶子节点，这使得B+树在查询时效率更稳定。
2. B+树在进行范围查询时效率更高，因为所有数据都在叶子节点，并且叶子节点之间形成了双向链表。

6. **什么是聚簇索引什么是非聚簇索引？**

**候选人**：聚簇索引是指数据与索引放在一起，B+树的叶子节点保存了整行数据，通常只有一个聚簇索引，一般是由主键构成。

非聚簇索引则是数据与索引分开存储，B+树的叶子节点保存的是主键值，可以有多个非聚簇索引，通常我们自定义的索引都是非聚簇索引。

7. **知道什么是回表查询吗？**

**候选人**：回表查询是指通过非聚簇索引找到对应的主键值，然后再通过主键值查询聚簇索引中对应的整行数据的过程。

8. **知道什么叫覆盖索引吗？**

**候选人**：覆盖索引是指在SELECT查询中，返回的列全部能在索引中找到，避免了回表查询，提高了性能。使用覆盖索引可以减少对主键索引的查询次数，提高查询效率。

9. **MySQL超大分页怎么处理？**

**候选人**：超大分页通常发生在数据量大的情况下，使用LIMIT分页查询且需要排序时效率较低。可以通过覆盖索引和子查询来解决。首先查询数据的ID字段进行分页，然后根据ID列表用子查询来过滤只查询这些ID的数据，因为查询ID时使用的是覆盖索引，所以效率可以提升。

10. **索引创建原则有哪些？**

**候选人**：创建索引的原则包括：

索引的话其实b+树的叶子节点存储的是索引值+值（主键或者整行数据）

* 表中的数据量超过10万以上时考虑创建索引。
* 选择查询频繁的字段作为索引，如查询条件、排序字段或分组字段。
* 字段经常作为 WHERE 条件
* 字段经常用于排序（ORDER BY）
* 字段经常参与 JOIN 或 GROUP BY
* 尽量使用复合索引，覆盖SQL的返回值。
* 如果字段区分度不高，可以将其放在组合索引的后面。
* 对于内容较长的字段，考虑使用前缀索引。
* 控制索引数量，因为索引虽然可以提高查询速度，但也会影响插入、更新的速度。

11. **什么情况下索引会失效？**

**候选人**：索引可能在以下情况下失效：

* 没有遵循最左匹配原则。
* 使用了模糊查询且%号在前面。
* 在索引字段上进行了运算或类型转换。
* 使用了复合索引但在中间使用了范围查询，导致右边的条件索引失效。（这样的话中间和左边都算索引命中，但是右边的的失效了，只能使用回表查询）

12. **SQL的优化经验有哪些？**

**候选人**：SQL优化可以从以下几个方面考虑：

* 建表时选择合适的字段类型。
* 使用索引，遵循创建索引的原则。
* 编写高效的SQL语句，比如避免使用SELECT \*，尽量使用UNION ALL代替UNION，以及在表关联时使用INNER JOIN。
* 采用主从复制和读写分离提高性能。
* 在数据量大时考虑分库分表。

13. **创建表的时候，你们是如何优化的呢？**

**候选人**：创建表时，我们主要参考《嵩山版》开发手册，选择字段类型时结合字段内容，比如数值类型选择TINYINT、INT、BIGINT等，字符串类型选择CHAR、VARCHAR或TEXT。

14. **在使用索引的时候，是如何优化呢？**

**候选人**：在使用索引时，我们遵循索引创建原则，确保索引字段是查询频繁的，使用复合索引覆盖SQL返回值，避免在索引字段上进行运算或类型转换，以及控制索引数量。

15. **你平时对SQL语句做了哪些优化呢？**

**候选人**：

我对SQL语句的优化包括指明字段名称而不是使用SELECT \*

避免造成索引失效的写法

聚合查询时使用UNION ALL代替UNION

表关联时优先使用INNER JOIN以及在必须使用LEFT JOIN或RIGHT JOIN时，确保小表作为驱动表。

16. **事务的特性是什么？可以详细说一下吗？**

**候选人**：事务的特性是ACID，即原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。例如，A向B转账500元，这个操作要么都成功，要么都失败，体现了原子性。转账过程中数据要保持一致，A扣除了500元，B必须增加500元。隔离性体现在A向B转账时，不受其他事务干扰。持久性体现在事务提交后，数据要被持久化存储。

17. **并发事务带来哪些问题？**

**候选人**：并发事务可能导致脏读、不可重复读和幻读。脏读是指一个事务读到了另一个事务未提交的“脏数据”。不可重复读是指在一个事务内多次读取同一数据，由于其他事务的修改导致数据不一致。幻读是指一个事务读取到了其他事务插入的“幻行”。

18. **怎么解决这些问题呢？MySQL的默认隔离级别是？**

**候选人**：解决这些问题的方法是使用事务隔离。MySQL支持四种隔离级别：

1. 未提交读（READ UNCOMMITTED）：解决不了所有问题。
2. 读已提交（READ COMMITTED）：能解决脏读，但不能解决不可重复读和幻读。
3. 可重复读（REPEATABLE READ）：能解决脏读和不可重复读，但不能解决幻读，这也是MySQL的默认隔离级别。
4. 串行化（SERIALIZABLE）：可以解决所有问题，但性能较低。

19. **undo log和redo log的区别是什么？**

**候选人：**redo log记录的是数据页的物理变化（执行语句），用于服务宕机后的恢复，保证事务的持久性。而undo log记录的是逻辑日志（数据快照，只不过执行的时候根据上一个数据快照进行反向操作），用于事务回滚时恢复原始数据，保证事务的原子性和一致性。

20. **事务中的隔离性是如何保证的呢？（你解释一下MVCC）**

**候选人**：事务的隔离性通过锁和多版本并发控制（MVCC）来保证。MVCC通过维护数据的多个版本来避免读写冲突。底层实现包括隐藏字段、undo log和read view。隐藏字段包括trx\_id（事务id）和roll\_pointer（回滚指针）。undo log记录了不同版本的数据，通过roll\_pointer形成版本链。read view定义了不同隔离级别下的快照读，决定了事务访问哪个版本的数据。

21. **MySQL主从同步原理是什么？**

**候选人**：MySQL主从复制的核心是二进制日志（Binlog）。步骤如下：

1. 主库在事务提交时记录数据变更到Binlog。
2. 从库读取主库的Binlog并写入中继日志（Relay Log）。
3. 从库重做中继日志中的事件，反映到自己的数据中。

22. **你们项目用过MySQL的分库分表吗？**

**候选人**：我们采用微服务架构，每个微服务对应一个数据库，是根据业务进行拆分的，这个其实就是垂直拆分。

23. **那你之前使用过水平分库吗？**

**候选人**：使用过。当时业务发展迅速，某个表数据量超过1000万，单库优化后性能仍然很慢，因此采用了水平分库。我们首先部署了3台服务器和3个数据库，使用mycat进行数据分片。旧数据也按照ID取模规则迁移到了各个数据库中，这样各个数据库可以分摊存储和读取压力，解决了性能问题。