|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学习报告 | | | | | |
| 报告人 | 张卓毅 | 时间 | 2020年07月28日 | 备注 |  |
| 序号 | 学习题目 | | 学习内容 | | |
| 1 | SQL性能调优 | | 优化方向   1. cpu消耗 2. 内存使用 3. 对磁盘、网络或其他I/O设备的输入/输出(I/O)操作   sql调优领域  应用程序级调优   1. sql语句调优 2. 管理变化调优   示例级调优   1. 内存 2. 数据结构 3. 实例配置   操作系统交互   1. I/O 2. swap 3. Parameters   sql优化方法   1. 优化业务数据 2. 优化数据设计 3. 优化流程设计 4. 优化sql语句 5. 优化物理结构 6. 优化内存分配 7. 优化I/O 8. 优化内存竞争 9. 优化操作系统   数据库调优金字塔理论    好的sql语句特点   1. 尽量简单，模块化 2. 易读，易维护 3. 节省资源（内存，cpu，扫描数据块少，少排序） 4. 不造成死锁 | | |
| 2 | sql优化方法 | | 1. sql语句不要写得太复杂。一个sql语句尽量简单，不要嵌套太多层 2. 使用“临时表”缓存中间结果。简化sql语句的重要方法使采用临时表暂存中间结果，可以避免程序中多次扫描主表，减少阻塞，提高并发性能      1. 使用like的时候要注意是否会导致全表扫，有的时候会需要进行一些模糊查询例如：select id from table where username like ‘%hollis%’关键词%hollis%，由于hollis前面用到了“%”，因此该查询会使用全表扫描，除非必要，否则不要在关键词前加% 2. 尽量避免使用 != 或 <> 操作符。在where语句中使用 != 或 <>，引擎将放弃使用索引而进行全表扫描。“=”增加索引的使用几率 3. 尽量避免使用 or 来连接条件；在 where 子句中使用 or 来连接条件，引擎将放弃使用索引而进行全表扫描。可以使用   Select id from t where num = 10  Union all  Select id from where num = 20 替代  Select id from where num = 10 or num = 20   1. 尽量避免使用in和not in:在where子句中使用in 和not in，引擎将放弃使用索引而进行全表扫描。可以使用   Select id from t where num between 10 and 20  替代  Select id from t where num in (10,20)   1. 可以考虑强制查询使用索引   Select \* from table force index(PRI) limit 2; (强制使用主键)select \* from table force index(hollis\_index) limit 2;(强制使用索引"hollis\_index")select \* from table force index(PRI,hollis\_index) limit 2;(强制使用索引"PRI和hollis\_index")     1. 尽量避免使用表达式、函数等操作作为查询条件；尽量避免大事务操作，提高系统并发能力。尽量避免使用游标；任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段 2. 尽可能的使用 varchar/nvarchar 代替 char/nchar。尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。 3. 要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引      1. 索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率、并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引。      1. (1)在经常需要进行检索的字段上创建索引，比如要按照表字段username进行检索，那么就应该在姓名字段上创建索引，如果经常要按照员工部门和员工岗位级别进行检索，那么就应该在员工部门和员工岗位级别这两个字段上创建索引。   (2)创建索引给检索带来的性能提升往往是巨大的，因此在发现检索速度过慢的时候应该首先想到的就是创建索引。  (3)一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。   1. 避免在索引上使用计算   在where字句中，如果索引列使计算或函数的一部分，DBMS的优化器不会使用索引而使用全表查询，函数属于计算的一种，同时在in和exists中通常情况下使用EXISTS，因为in不用索引。  效率低：  select \* from user where salary\*22>11000(salary是索引列)  效率高：  select \* from user where salary>11000/22(salary是索引列)   1. 使用预编译查询   程序中通常根据用户输入来动态执行SQL，这时应该尽量使用参数化SQL。数据库会对这些参数化SQL进行预编译，这样第一次执行的时候DBMS会这个SQL语句进行查询优化并且以后再执行这个SQL的时候就直接使用预编译结果   1. 调整Where字句中的连接顺序   DBMS一般采用自下而上的顺序解析where字句，根据这个原理表连接最好写在其他where条件前   1. 尽量将多条SQL语句压缩到一句SQL中。每次执行SQL的时候都有建立网络连接，进行权限校验，进行SQL语句的查询优化，发送执行结果的过程，非常耗时。 2. 用where字句替换having字句。避免使用having字句，因为having只会再检索出所有记录后才会对结果集进行过滤而where再聚合前 3. 使用表的别名。当SQL语句中连接多个表时，使用表的别名并把别名前缀于每个列名上。可以减少解析的时间和列名歧义引起语法错误 4. 用union all替换union。当SQL语句需要union两个查询结果集合时，即使检索结果中不会有重复的记录，如果使用union这两个结果集同样会尝试合并，然后在输出最总结果前进行排序，因此如果可以判断检索结果中不会重复的记录时候，应用union all，这样效率就会因此提高 5. 尽量避免使用游标。尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。 6. 用varchar/nvarchar代替char/nchar 7. 查询select语句优化   任何地方都不要使用select \* from t;用具体字段列表代替“\*”   1. 更新Update语句优化   如果只更改一两个字段，不要update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志   1. 删除delete语句优化   最高效的删除重复记录方法例子  DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);   1. 插入Insert语句优化   新建临时表时，如果一次性插入数据量很大，那么可以使用select into代替create table，避免造成大量log，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，后insert | | |
| 3 | 数据库表优化设计 | | 1. 表中应该避免可空的列，虽然表中允许空列，但是，空字段是一种比较特殊的数据类型 2. 表不应该有重复的值或列 3. 表中记录应该有一个唯一的标识符，在数据库表设计的时候，应用一个ID号来唯一的标识行记录，而不要通过名字、编号等字段来对记录进行区分 4. 数据库对象要有一个统一的前缀名 5. 尽量只存储单一实体类型的数据，这里的实体类型和数据类型不一样。这里的实体类型是所需要描述对象的本身 6. 字段顺序 7. 靠近记录开始的地方字段定位速度会明显快于记录尾的字段 8. 常用的字段放在前面 | | |
| 4 | 数据库性能监控 | | 业务关注MySQL的性能监控分为两种：  第一种是通过业务性能指标，或者说超时机制来判定，MySQL是否出现了瓶颈。  第二种是直接对监控项进行关注，应该注意以下几点：   1. Liunx操作系统层面的监控指标 2. 1.CPU核心指标 3. （1）%user(用户态的CPU使用)保持在20%以下，若太高可能是索引使用不 4. 当 5. (2) 各CPU负载是否均衡 6. (3) 是否有swap产生(关闭NUMA) 7. [Swap分区在系统的物理内存不够用的时候，把硬盘内存中的一部分空间释放出来，以供当前运行的程序使用。那些被释放的空间可能来自一些很长时间没有什么操作的程序，这些被释放的空间被临时保存到Swap分区中，等到那些程序要运行时，再从Swap分区中恢复保存的数据到内存中。] 8. [NUMA通过提供分离的存储器给各个处理器，避免当多个处理器访问同一个存储器产生的性能损失来试图解决这个问题。对于涉及到分散的数据的应用（在服务器和类似于服务器的应用中很常见），NUMA可以通过一个共享的存储器提高性能至n倍,而n大约是处理器（或者分离的存储器）的个数。] 9. (4) 可使用命令或工具vmstat、sar、dstat等 10. [vmstat 命令报告关于内核线程、虚拟内存、[磁盘](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98/2842227)、陷阱和 CPU 活动的统计信息。由 vmstat 命令生成的报告可以用于平衡系统负载活动。系统范围内的这些统计信息（所有的处理器中）都计算出以百分比表示的平均值，或者计算其总和。 ] 11. [sar是系统维护的重要工具，主要帮助我们掌握系统资源的使用情况，特别是内存和 CPU 的使用情况] 12. [dstat 是一个可以取代vmstat，iostat，netstat和ifstat这些命令的多功能产品。dstat扬长避短，即克服了这些命令的局限又增加了一些额外的功能，不但拥有更多的监控项，也更灵活。dstat在性能测试、基准测试和排除故障过程中可以很方便监控系统运行状况。   dstat可以查看所有的实时系统资源，如：通过统计IDE控制器当前状态来比较磁盘利用率，或者直接通过网络带宽数值来比较磁盘的吞吐率（在相同的时间间隔内）。   dstat以列表的形式提供选项信息，并清晰地告知以何种幅度和单位显示输出。输出信息整洁，降低发生错误的概率。最重要的是，整洁的数据更容易编写插件用来收集分析关注的数据信息。   dstat默认输出是专门为实时查看而设计的，然而也可以将详细信息通过cvs输出到一个文件，然后将cvs文件导入到Gnumeric或者Excel中生成图表。] 13. MySQL状态层面的监控 14. tps、qps、并发连接数(Thread\_connected)、并发活跃线程数(Thread\_running)、临时表(tmp\_disk\_tables)、锁等指标，有问题应排查sql语句或业务逻辑 15. 是否有不良线程状态，如copy to tmp table、Creating sort index、Sorting result、Creating tmp table、长时间Sending data等，如有问题应排查sql语句是否有索引使用不当，大的扫描 16. Slow query sql慢日志的增长情况，如有问题应根据慢日志排查sql语句问题或随数据量增大产生的问题   Mysql监控平台  prometheus+Grafana  prometheus 是由 SoundCloud 开发的开源监控报警系统和时序列数据库(TSDB)，prometheus是一个监控采集与数据存储框架（监控server端），具体采集什么数据依赖于具体的exporter（监控client端）  grafana是一个高颜值的监控绘图程序，也是一个可视化面板（Dashboard），grafana的厉害之处除了高颜值，还支持多种数据源（支持Graphite、zabbix、InfluxDB、Prometheus和OpenTSDB作为数据源）、支持灵活丰富的dashboard配置选项(例如：可以把多个实例的相同采集项配置在一个展示框里)，使得相较于其他开源监控系统来说更易用性，学习成本更低。    SolarWinds Database Performance Analyzer for SQL Server | | |
| 5 | mysql慢查询日志 | | 1. 开启慢查询日志   在 MySQL 中，慢查询日志默认为OFF状态，通过如下命令进行查看： mysql> show variables like "slow\_query\_log";  通过如下命令进行设置为 ON 状态： set global slow\_query\_log = "ON";   1. 其中slow\_query\_log\_file属性，表示慢查询日志存储位置，其日志默认名称为 host 名称      1. 慢查询 查询时间，当SQL执行时间超过该值时，则会记录在slow\_query\_log\_file 文件中，其默认为 10 ，最小值为 0，(单位：秒)     可通过以下命令修改当设置值小于0时，默认为 0  mysql> set global long\_query\_time = 5;  通过上述设置后，退出当前会话或者开启一个新的会话，执行如下命令： select sleep(11); **备注**: 这里的 11 并不是固定值，仅仅为了展示，其值只需要符合以下条件即可： **该值大于等于long\_query\_time 值即可**。  该 SQL 则会进入慢查询日志中。通过cat 命令查看后如下所示：     1. 慢查询日志以#作为起始符。 2. User@Host：表示用户 和 慢查询查询的ip地址。 3. 如上所述，表示 root用户 localhost地址。 4. Query\_time: 表示SQL查询持续时间， 单位 (秒)。 5. Lock\_time: 表示获取锁的时间， 单位(秒)。 6. Rows\_sent: 表示发送给客户端的行数。 7. Rows\_examined: 表示：服务器层检查的行数。 8. set timestamp ：表示 慢SQL 记录时的时间戳。 9. 其中 select sleep(6) 则表示慢SQL语句。   **注意事项**   1. 在 MySQL 中，慢查询日志中默认不记录管理语句，如： alter table, analyze table，check table等。 不过可通过以下属性进行设置： mysql> set global log\_slow\_admin\_statements = "ON"; Query OK, 0 rows affected (0.00 sec) 2. 在 MySQL 中，还可以设置将未走索引的SQL语句记录在慢日志查询文件中(默认为关闭状态)。通过下述属性即可进行设置： mysql> set global log\_queries\_not\_using\_indexes = "ON"; Query OK, 0 rows affected (0.00 sec) 3. 在MySQL中，日志输出格式有支持：FILE(默认)，TABLE 两种，可进行组合使用。如下所示: set global log\_output = "FILE,TABLE"; 这样设置会同时在 FILE, mysql库中的slow\_log表中同时写入。 4. 设置该属性后，只要SQL未走索引，即使查询时间小于long\_query\_time值，也会记录在慢SQL日志文件中 5. 该设置会导致慢日志快速增长，开启前应检查慢查询日志文件所在磁盘空间是否充足 6. 生产环境中不建议开启 | | |