|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | |  | | |
| 5 | mysql慢查询日志 | |  | | |