SQL优化

- 一、优化器
  - （一）什么是优化器：Oracle的优化器是SQL分析和执行优化工具，它负责生成和制定SQL的执行计划
  - （二）优化器种类及运行模式
    - 1、RBO(Rule-Based-Optimization)基于规则的优化器
      - RBO有严格的使用规则，只有按照这套规则去写SQL语句，无论数据表中的内容怎样，也不会影响你的执行计划
      - RBO对数据"不敏感"，它要求SQL编写人员必须要了解各项细则
      - RBO一直沿用到ORACLE 9I，从ORACLE 10g开始彻底被抛弃
    - 2、CBO(Cost-Based-Optimization)基于代价的优化器
      - CBO是比RBO更加合理、可靠的优化器，在ORACLR 10g中完全取代了RBO
      - CBO通过计算各种可能的执行计划的“代价”，即COST，从中选用COST最低的执行方案作为实际运行方案
      - CBO依赖数据库对象的统计信息，统计信息的准确与否会影响CBO做出最优的选择，对数据“敏感”
    - 3、ORACLE 11g中的运行模式
      - （1）跟RBO相关的运行模式
        - RULE是基于规则的运行模式
        - CHOOSE表示如果查询的表存在搜集的统计信息则基于代价来执行（choose模式下oracle采用的是first_rows），否则基于规则来执行
      - （2）跟CBO相关的运行模式
        - First_rows(n)：oracle在执行时，优先考虑将结果集中的前N条记录以最快的速度反馈，而其他的结果并不需要同时返回
        - All_rows --11g中的默认值：oracle会用最快的速度将sql执行完毕，将结果集全部返回
        - first_rows(n)和all_rows的区别：all_rows强调以最快的速度将sql执行完毕，并将结果集反馈回来，而first_rows(n)则侧重于返回前n条记录的执行时间
    - 4、修改CBO模式的三种方法
      - （1）SQL语句
        - Sessions级别：
          SQL> alter session set optimizer_mode=all_rows;
      - （2）修改pfile参数
        - OPTIMIZER_MODE=RULE/CHOOSE/FIRST_ROWS/ALL_ROWS

- （3）语句级别用HINT(/* + ...*/)来设定
  - Select /*+ first_rows(10) */ name from table;

    Select /*+ all_rows */ name from table;
-

- 二、执行计划
  - （一）什么是ORACLE执行计划
    - 执行计划是一条查询语句在ORACLE中的执行过程或访问路径，就是对一个查询任务，做出一份怎样去完成任务的详细方案
    - 通常SQL中没有提示信息，是由数据库来决定的
  - （二）怎么查看ORACLE执行计划
    - 1、在SQL窗口执行完一条select语句后，按F5即可查看刚刚执行的这条查询语句的执行计划
    - 2、执行计划的常用列子段解释
      - 基数（Rows）：Oracle估计的当前操作的返回结果集行数
      - 字节（Bytes）：执行该步骤后返回的字节数
      - 耗费（COST）、CPU耗费：Oracle估计的该步骤的执行成本，用于说明SQL执行的代价，理论上越小越好（该值可能与实际有出入）
      - 时间（Time）：Oracle估计的当前操作所需的时间
      - 🔴可以自行添加自己想要看的信息
  - （三）看懂ORACLR 执行计划
    - 1、执行顺序
      - 根据Operation缩进来判断，缩进最多的最先执行（缩进相同时，最上面的最先执行）
      - 同一级的动作执行时，遵循"最上最右"执行原则
      - 注：PLSQL提供了查看执行顺序的功能按钮
    - 2、表的扫描方式
      - （1）TBALE ACCESS BY ...即描述是该动作执行时表访问（或者说orcale访问数据）的方式
      - （2）表的扫描方式
        - 1） TABLE ACCESS FULL (全表扫描)
          - Oracle会读取表中所有的行，并检查每一行是否满足SQL中的where限定条件
          - 全表扫描时可以使用多块读（即一次I/O读取多块数据块）操作，提升吞吐量

- 使用建议：数据量太大的表不建议使用全表扫描，除非本身需要取出的数据较多，占到表数据量的10%以上
- 对应的HINTS:

  /*+ FULL(表名) */ 来指定全表扫描

- 2）TABLE ACCESS BY ROWID（通过ROWID的表存取）
  - 什么是ROWID：是oracle自动加在表中每一行的一列伪列，不会占用物理存储，可以像其他列一样使用，但不能增删改操作，一旦一行数据插入后，就会生成唯一的ROWID，即使发生迁移该行的ROWID也不会改变
  - ROWID指出该行所在的数据文件、数据块以及行在该块中的位置
  - ROWID扫描的逻辑：通过ROWID可以快速定位到目标数据上，这也是oracle中存取单行数据最快的方法
- 3）TABLE ACCESS BY INDEX SCAN（索引扫描）
  - 在索引块中，既存储每个索引的键值，也存储具有该键值的行的ROWID。
  - 索引扫描其实分为两步：
    - Ⅰ：扫描索引得到对应的ROWID
    - Ⅱ：通过ROWID定位到具体的行读取数据
  - 索引扫描的细分
    - INDEX UNIQUE SCAN(索引唯一扫描)
    - INDEX RANGE SCAN(索引范围扫描)
    - INDEX FULL SCAN（索引全扫描)
    - INDEX FAST FULL（索引快速扫描)
    - INDEX SKIP SCAN(索引跳跃扫描)
- 4） 表访问方式总结
  - 表的访问方式主要有两种：全表扫描（TABLE ACCESS FULL ）和索引扫描（INDEX SCAN）
  - 扫描方式可能存在的问题
    - 如果表上存在选择性很好的索引，却走了全表扫描，而且是大表的全表扫描，就说明表的访问方式可能存在问题
    - 如果大表上没有合适的索引二走了全表扫描，就需要分析能否建立索引，或者选择更合适的表连接方式和连接顺序以提高效率
    - 对应的HINTS:

      /*+ INDEX(表名 索引名) */ 来指定索引扫描

- 3、表连接方式
  - 注：这里将首先存取的表称作 row source 1，将之后参与连接的表称作 row source 2；

- 1） 表连接对查询效率的影响：
  - 表（row source）之间的连接顺序对于查询效率有很大的影响，对首先存取的表（驱动表）先应用某些限制条件（Where过滤条件）以得到一个较小的row source，可以使得连接效率提高。
  - 驱动表与匹配表
    - 驱动表（Driving Table）
      - 表连接时首先存取的表，又称外层表（Outer Table），这个概念用于NESTED LOOPS（嵌套循环） 与 HASH JOIN（哈希连接）中；
      - 如果驱动表返回较多的行数据，则对所有的后续操作有负面影响，故一般选择小表（应用Where限制条件后返回较少行数的表）作为驱动表。
    - 匹配表（Probed Table）
      - 又称为内层表（Inner Table），从驱动表获取一行具体数据后，会到该表中寻找符合连接条件的行。故该表一般为大表（应用Where限制条件后返回较多行数的表）。
- 2） 表连接方式种类
  - a） NESTED LOOPS(嵌套循环)
    - 内部连接过程
      - a) 取出 row source 1 的 row 1（第一行数据），遍历 row source 2 的所有行并检查是否有匹配的，取出匹配的行放入结果集中

        b) 取出 row source 1 的 row 2（第二行数据），遍历 row source 2 的所有行并检查是否有匹配的，取出匹配的行放入结果集中

        c) 若 row source 1 （即驱动表）中返回了 N 行数据，则 row source 2 也相应的会被全表遍历 N 次。
      - 因为 row source 1 的每一行都会去匹配 row source 2 的所有行，所以当 row source 1 返回的行数尽可能少并且能高效访问 row source 2（如建立适当的索引）时，效率较高。
    - 嵌套循环如何提升连接效率
      - 🔴嵌套循环的表有驱动顺序，注意选择合适的驱动表。
      - 应尽可能使用限制条件（Where过滤条件）使驱动表（row source 1）返回的行数尽可能少，同时在匹配表（row source 2）的连接操作关联列上
        建立唯一索引（UNIQUE INDEX）或是选择性较好的非唯一索引，此时嵌套循环连接的执行效率会变得很高。
      - 若驱动表返回的行数较多，即使匹配表连接操作关联列上存在索引，连接效率也不会很高
  - b） HASH JOIN (哈希连接)

-
  - 🔴哈希连接只适用于等值连接（即连接条件为 =），先生成哈希值，再通过哈希值做匹配，哈希连接只能做等值连接
- 先将驱动表的数据按照条件字段以散列的方式放入内存，然后在内存中匹配满足条件的行。哈希连接需要有合适的内存，
  而且必须在CBO优化模式下，连接两表的WHERE条件有等号的情况下才可以使用。
- 🔴哈希连接在表的数据量较大，表中没有合适的索引可用时比嵌套循环的效率要高。
- 内部连接过程简述：
  -
    a) 取出 row source 1（驱动表，在HASH JOIN中又称为Build Table） 的数据集，然后将其构建成内存中的一个
    Hash Table（Hash函数的Hash KEY就是连接操作关联列），创建Hash位图（bitmap）

    b) 取出 row source 2（匹配表）的数据集，对其中的每一条数据的连接操作关联列使用相同的Hash函数，并找到对应的
    驱动表里的数据在 Hash Table 中的位置，在该位置上检查能否找到匹配的数据
  - 计算出整张被探查表关联字段的哈希值，这些哈希值和整张被探查表一起放入缓存区，然后从驱动表逐条取记录，
    计算出关联字段对应的哈希值，再与被探查表的哈希值匹配,匹配上了再精准匹配每一条记录。
- 延伸阅读：Hash Table相关
  - 散列（hash）技术：在记录的存储位置和记录具有的关键字key之间建立一个对应关系 f ，使得输入key后，可以得到对应的存储位置 f(key)，这个对应关系 f 就是散列（哈希）函数；
  - 采用散列技术将记录存储在一块连续的存储空间中，这块连续的存储空间就是散列表（哈希表）；
  - 不同的key经同一散列函数散列后得到的散列值理论上应该不同，但是实际中有可能相同，相同时即是发生了散列（哈希）冲突，解决散列冲突的办法有很多，比如HashMap中就是用链地址法来解决哈希冲突；
  - 哈希表是一种面向查找的数据结构，在输入给定值后查找给定值对应的记录在表中的位置以获取特定记录这个过程的速度很快。
- HASH JOIN的三种模式：
  - OPTIMAL HASH JOIN
  - ONEPASS HASH JOIN
  - MULTIPASS HASH JOIN
- c） SORT MERGE JOIN (排序-合并连接)

- 假设有查询：select a.name, b.name from table_A a join table_B b on (a.id = b.id)
- 内部连接过程：
  - a) 生成 row source 1 需要的数据，按照连接操作关联列（如示例中的a.id）对这些数据进行排序
  - b) 生成 row source 2 需要的数据，按照与 a) 中对应的连接操作关联列（b.id）对数据进行排序
  - c) 两边已排序的行放在一起执行合并操作（对两边的数据集进行扫描并判断是否连接）
- 数据量大，不等值连接的时候
- d)　CARTESIAN　PRODUCT(笛卡尔积)

- 表连接总结
  - 对于表的连接顺序，多数情况下使用的是嵌套循环，尤其是在索引可用性好的情况下，使用嵌套循环式最好的，但当ORACLE发现需要访问
    的数据表较大，索引的成本较高或者没有合适的索引可用时，会考虑使用哈希连接，以提高效率。排序合并连接的性能最差，但在存在排序
    需求，或者存在非等值连接无法使用哈希连接的情况下，排序合并的效率，也可能比哈希连接或嵌套循环要好。

- 🔴在优化sql语句的时候，经常会需要去查看sql怎么样执行消耗更少

三、HINTS

- 1、通过查看执行计划，分析性能原因，通过hints可以优化sql性能
- 2、并行：/*+PARALLEL(8)*/
  - #什么是并行#
    - 对于一个大的任务，一般的做法是利用一个进程，串行的执行，如果系统资源足够，可以采用parallel技术，把一个大的任务分成若干个小的任务，
      同时启用n个进程/线程，并行的处理这些小的任务，这些并发的进程称为并行执行服务器(parallel executeion server)，这些并发进程由一个称
      为并发协调进程的进程来管理
  - 启用Parallel前的忠告
    - 只有在需要处理一个很大的任务，如需要几十分钟，几个小时的作业中，并且要有足够的系统资源的情况下
      (这些资源包括cpu，内存，io),您才应该考虑使用parallel。否则，在一个多并发用户下，系统本身资源负担已经很大的情况下，启用parallel，
      将会导致某一个会话试图占用了所有的资源，其他会话不得不去等待，从而导致系统系能反而下降的情况，一般情况下，oltp系统不要使用parallel，olap系统中可以考虑去使用。
  - 并行的写法：
    - SELECT|INSERT|DELETE|UPDATE|MERGE　/*+PARALELL（表名 并行数）*/

- 关于执行效率，建议还是多按照index的方法来提高效果。Oracle有自带的explan road的方法，在执行之前，先看下执行计划路线，
  对写好的SQL tuned之后再执行。实在没办法了，再用parallel方法。

- 3、常用的hints
  - （1）/*+ PARALLEL(表名,并行数) */ --指定开启多少个并行
  - （2）/*+ INDEX(表名 索引名) */ --指定索引
  - （3）/*+ FULL(表名) */ --指定全表扫描
  - （4）/*+ USE_NL(表名1 表名2) */ --指定用NESTED LOOP连接
  - （5）/*+ USE_HASH(表名1 表名2) */ --指定用HASH连接
  - （6）/*+ USE_MERGE(表名1 表名2) */ --指定用SORT MERGE JOIN （排序-合并连接）
  - （7）/*+ LEADING(表名1 表名2) */ --指定表1作为驱动表
  - （8）/*+ APPEND */ --数据直接插入到高水位上面(与insert连用)

- 四、sql语句的执行步骤
  - 1）语法分析，分析语句的语法是否符合规范，衡量语句中各表达式的意义。
  - 2）语义分析，检查语句中涉及的所有数据库对象是否存在，且用户有相应的权限。
  - 3）视图转换，将涉及视图的查询语句转换为相应的对基表查询语句。
  - 4）表达式转换， 将复杂的 SQL 表达式转换为较简单的等效连接表达式。
  - 5）选择优化器，不同的优化器一般产生不同的“执行计划”
  - 6）选择连接方式， ORACLE 有三种连接方式，对多表连接 ORACLE 可选择适当的连接方式。
  - 7）选择连接顺序， 对多表连接 ORACLE 选择哪一对表先连接，选择这两表中哪个表做为源数据表。
  - 8）选择数据的搜索路径，根据以上条件选择合适的数据搜索路径，如是选用全表搜索还是利用索引或是其他的方式。
  - 9）运行“执行计划”

- 五、SQL优化
  - （一）表设计的优化：在创建表时，先把索引、分区表写好
  - （二）SQL的优化
    - 1、从扫描方式优化：全表扫描的时候，有索引的走索引扫描，没索引的创建索引
    - 2、优化连接方式：多数情况下使用嵌套循环，尤其是在索引性能好的情况下，使用嵌套循环最好，但当数据表较大，索引成本较高或没有合适的索引时，考虑哈希连接，排序-合并连接性能最差，但存在排序需求时，或存在非等值连接无法使用哈希连接时，排序合并连接效率高
  - （三）使用HINTS进行优化
  - （四）硬件的加速

- （五）并行：/*+PARALLEL(8)*/，效果显著，但是慎用

六、SQL语句的优化

- （一）避免在select 后使用'*' 符号
- （二） DISTINCT是性能最差的去重，group by或者EXISTS替代
  - 当提交一个包含一对多表信息(比如部门表和雇员表)的查询时，避免在SELECT子句中使用DISTINCT。
    一般可以考虑用EXIST替换

```
例子如下：
------------DISTINCT------------------
SELECT DISTINCT D.DEPTNO,
       D.DNAME
 FROM DEPT D
 JOIN EMP E
  ON D.DEPTNO = E.DEPTNO;

------------EXISTS------------------
SELECT D.DEPTNO,
   D.DNAME
 FROM DEPT D
WHERE EXISTS (SELECT 1
       FROM EMP
       WHERE DEPTNO = D.DEPTNO);
```

  - 为什么EXISTS去重比DISTINCT性能好，其运行原理是怎样的?
    - 用EXISTS，只要找到第一个符合条件的值，就返回了，而不管后面有多少条符合条件的重复记录。
      而DISTINCT，是全扫描，必须查找全部符合条件的记录后，再返回唯一值。
  - 总结：
    用distinct需要每次都遍历匹配表进行比对，而使用exists只需要比对匹配表的一部分，在匹配表数据十分庞大时，
    这种性能差别就能更好的体现出来。
  - 注意，EXIST去重只适用于一对多查询时的去重，如果只是对单表数据进行去重，就无法使用EXIST!
- （三）连接和分组时先使用where条件过滤；用Where子句替换HAVING子句

```
用Where子句替换HAVING子句,案例：
SELECT DEPTNO,JOB,AVG(SAL)
 FROM EMP1
GROUP BY DEPTNO,JOB
HAVING JOB = 'MANAGER'

SELECT DEPTNO,AVG(SAL)
 FROM EMP1
 WHERE JOB = 'MANAGER'
GROUP BY DEPTNO
```

- （四）用TRUNCATE替换DELETE;

- （五）避免笛卡儿积；
- （六）避免索引失效：
  - 1.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：
    select id from t where num is null
    ```
    如：
    select id from t where num is null

    //可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：
    select id from t where num=0
    ```
  - 2.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描
  - 3.应尽量避免对索引字段进行算术运算；
  - 4.应尽量避免对索引字段进行函数操作；
  - 5.应避免在使用LIKE时，判断依据把'%'写在第一列；
- （七）使用SQL内置函数加快查询速度，如行列转换PIVOT，用DECODE 替换CASE WHEN;
- （八）用(NOT) IN 和 (NOT) EXITST 替换全连接JOIN,.
  (NOT) IN 和 (NOT)EXISTS 的性能大多数情况性能是差不多的，具体以执行计划为准;
- （九）尽量多使用COMMIT
  - 只要有可能,在程序中尽量多使用COMMIT, 这样程序的性能得到提高,需求也会因为COMMIT所释放的资源而减少:
    COMMIT所释放的资源:
  - a. 回滚段上用于恢复数据的信息.
  - b. 被程序语句获得的锁.
  - c. redo log buffer 中的空间
  - d. ORACLE为管理上述3种资源中的内部花费
- （十）使用WITH语句，对某段SQL建立临时表；
  - with table as 相当于建个临时表（用于一个语句中某些中间结果放在临时表空间的SQL语句），
    可以将查询中的子查询命名，放到SELECT语句的最前面。
  - 语法就是
    with tempname as (select ....)
    select ...
    ```
    例子：
    with t as (select * from emp where depno=10)
    select * from t where empno=xxx
    ```
  - 何时被清除
    with as临时表是查询完成后就被清除。

- 关于临时表的扩充
  - 在oracle中，临时表分为会话级别(session)和事务级别(transaction)两种。
  - 会话级临时表
    - 会话级的临时表在整个会话期间都存在，直到会话结束
    - 会话级临时表是指临时表中的数据只在会话生命周期之中存在，当用户退出会话结束的时候，Oracle自动清除临时表中数据。
    - 创建方式

      创建方式1：

      create global temporary table temp1(id number) on commit PRESERVE rows;

      insert into temp1values(100);

      select * from temp1;

      创建方式2：

      create global temporary table temp1 ON COMMIT PRESERVE ROWS as select id from 另一个表;

      select * from temp1;

      这个时候，在当前会话查询数据就可以查询到了，但是再新开一个会话窗口查询，就会发现temp1是空表。
  - 事务级临时表
    - 事务级别的临时表数据在transaction结束后消失，即commit/rollback或结束会话时，会清除临时表数据。
    - 创建方式

      创建方式1：

      create global temporary table temp2(id number) on commit delete rows;

      insert into temp2 values(200);

      select * from temp2;

      创建方式2：

      create global temporary table temp2 as select id from 另一个表;（默认创建的就是事务级别的）

      select * from temp2;

      这时当你执行了commit和rollback操作的话，再次查询表内的数据就查不到了。

- 会话级别和事务级别的区分
  - 1、事务级临时表 on commit delete rows; 当COMMIT的时候删除数据（默认情况）
    2、会话级临时表 on commit preserve rows; 当COMMIT的时候保留数据，当会话结束删除数据
- oracle的临时表创建完就是真实存在的，无需每次都创建。
- 若要删除临时表可以：

  truncate table 临时表名;
  drop table 临时表名;

- 总结

  SQL优化
  一、表设计
  1、建分区表
  2、建索引

  二、SQL语句优化
  1、写SQL注意要点
  2、SQL逻辑（减少join）

  三、执行计划（hints）
  1、查看执行计划，了解性能差的地方
  2、最后大招：并行（parallel）