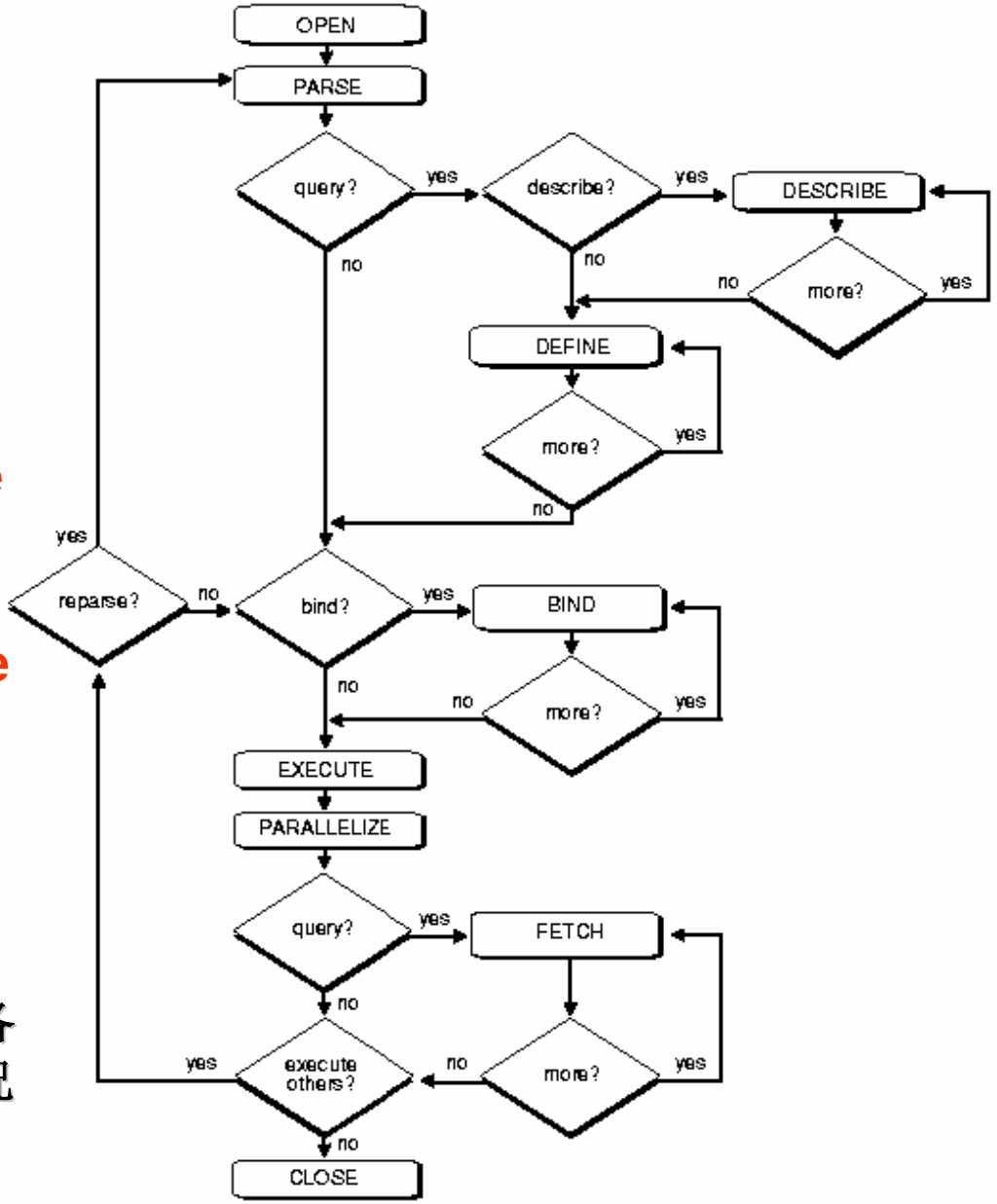
SQL语句的优化

SQL语句处理过程

- 查询语句处理：select
- DML语句处理：insert、update
  delete
- DDL语句处理：create、drop、
  alter、truncate
- TCL语句处理：commit
  rollback
- DCL语句处理：grant、revoke

说明：
图中列出了处理一个SQL语句需要的各个重要阶段，不同的语句，在不同情况下可能顺序不同。



SQL语句处理过程

1. 创建游标(Create a Cursor)
   - 任何SQL语句都会创建它，特别在运行DML语句时，都是自动创建游标的，不需要开发人员干预。
2. 分析语句(Parse the Statement)

在语法分析期间，SQL语句从用户进程传送到Oracle，SQL语句经语法分析分别执行下列操作：

- 翻译SQL语句，验证它是合法的语句，即书写正确
- 实现数据字典的查找，以验证是否符合表和列的定义
- 在所要求的对象上获取语法分析锁，使得在语句的语法分析过程?验证为存取所涉及的模式对象所需的权限是否满足
- 决定此语句最佳的执行计划
- 将它装入共享SQL区
- 对分布的语句来说，把语句的全部或部分路由到包含所涉及数据

SQL语句处理过程

3. 描述查询结果(Describe Results of a Query)

- 描述阶段只有在查询结果的各个列是未知时才需要；例如，当查询由用户交互地输入需要输出的列名。在这种情况要用描述阶段来决定查询结果的特征（数据类型，长度和名字)

4. 定义查询的输出数据(Define Output of a Query)

- 在查询的定义阶段，你指定与查询出的列值对应的接收变量的位置、大小和数据类型，这样我们通过接收变量就可以得到查询结果。如果必要的话，Oracle会自动实现数据类型的转换。这是将接收变量的类型与对应的列类型相比较决定的

5. 绑定变量(Bind Any Variables)

- 执行到此时，Oracle知道了SQL语句的意思，但仍没有足够的信息用于执行该语句。Oracle 需要得到在语句中列出的所有变量的值。通过某个变量对某列的值进行限定，这个过程就称绑定变量。

SQL语句处理过程

6. 并行执行语句(Parallelize the Statement)

- ORACLE 可以在SELECT,DML语句中执行相应并行查询操作
- 对于某些DDL操作，如创建索引、用子查询创建表、在分区表上的操作，也可以执行并行操作。
- 并行化可以导致多个服务器进程为同一个SQL语句工作，使该SQL语句可以快速完成
- 但会耗费更多的资源，所以除非很有必要，否则不要使用并行查询。

7. 执行语句(Run the Statement)

- 此时，Oracle拥有所有需要的信息与资源，可以真正运行SQL语句。如果该语句为SELECT查询或INSERT语句，则不需要锁定任何行，因为没有数据需要被改变。然而，如果语句为UPDATE或DELETE语句，则该语句影响的所有行都被锁定，防止该用户提交或回滚之前，别的用户对这些数据进行修改。这保证了数据的一致性。
- 对于某些语句，你可以指定执行的次数，这称为批处理(array processing)。指定执行N次，则绑定变量与定义变量被定义为大小为N的数组的开始位置，这种方法可以减少网络开销，也是优化的技巧之一。

SQL语句处理过程

8. 取出查询的行(Fetch Rows of a Query)

- 在fetch阶段，行数据被取出来，每个后续的存取操作检索结果集中的下一行数据，直到最后一行被取出来。上面提到过，批量的fetch是优化的技巧之一

9. 关闭游标(Close the Cursor)

- SQL语句处理的最后一个阶段就是关闭游标

SQL语句处理过程

- DDL语句处理(create、drop、alter)
  - DDL语句的执行不同与DML语句和查询语句的执行，这是因为DDL语句执行成功后需要对数据字典数据进行修改。对于DDL语句，语句的分析阶段实际上包括分析、查找数据字典信息和执行。
  - 事务管理语句、会话管理语句、系统管理语句只有分析与执行阶段，为了重新执行该语句，会重新分析与执行该语句。
- TCL语句处理(commit、rollback)
  - **Commit顺序：**
    - 服务器为每个COMMIT产生一个SCN。使改变永久化。
    - LGWR进程将日志缓冲区数据并带有SCN一起写到重做日志文件。
    - 服务器释放表级和行级锁。
    - 用户被提示COMMIT完成。
    - 服务器使事务已完成。
  - Rollback顺序
    - 服务器进程不做任何的改变。
    - 服务器释放表级和行级锁。
    - 服务器使事务已完成。

Oracle优化器(Optimizer)

- 基于规则的优化器--Rule Based (Heuristic) Optimization(简称RBO)
  - 优化器在分析SQL语句时,所遵循的是Oracle内部预定的一些规则（15条）。
  - 常见规则有：有索引时使用索引而不考虑其它影响性能的因素。
  - 某些情况下所选择的执行计划性能并不是最佳。

- 基于代价的优化器 -- Cost Based Optimization(简称CBO)
  - Oracle把一个代价引擎(Cost Engine)集成到数据库内核中，用来估计每个执行计划需要的代价，该代价将每个执行计划所耗费的资源进行量化，从而CBO可以根据这个代价选择出最优的执行计划。
  - 一个查询耗费的资源被分成3个基本组成部分：I/O代价、CPU代价、network代价。

优化器的模式

- **RULE**
  - 使用**RBO**优化器
- **CHOOSE**
  - 为缺省值,它是根据实际情况，如果数据字典中包含被引用的表的统计数据，即引用的对象已经被分析，则就使用**CBO**优化器，否则为**RBO**优化器
- **FIRST_ROWS**
  - 使用**CBO**优化器，是以数据的响应时间为主要目标，以便快速查询出开始的几行数据。
- **FIRST_ROWS_[1 | 10 | 100 | 1000]**
  - 使用**CBO**优化器让优化器选择一个能够把响应时间减到最小的查询执行计划，而迅速产生查询结果的前 **n** 行（**9i**中引入）,常用于联机应用处理。
- **ALL_ROWS**
  - 使用**CBO**优化器，是以数据的吞吐量为主要目标，以便可以使用最少的资源完成语句，常用于**DDS**。

优化器的使用

- Instance级别
  - 通过在init.ora文件中设定OPTIMIZER_MODE=RULE、OPTIMIZER_MODE=CHOOSE、OPTIMIZER_MODE=FIRST_ROWS、OPTIMIZER_MODE=ALL_ROWS去来设定，默认用的是Choose这种方式。
- Sessions级别
  - 通过SQL> ALTER SESSION SET OPTIMIZER_MODE=<Mode>;来设定 。
- 语句级别
  - 在语句中添加一些提示，即需要用到Hint 。
  - [SELECT | DELETE|UPDATE] /*+ [hint | text] */
- 数据分析语句
  - SQL> ANALYZE TABLE table_name COMPUTE STATISTICS;
  - SQL> ANALYZE INDEX index_name ESTIMATE STATISTICS;
  - SQL>analyze table table_name delete statistics;

查看执行计划

- 什么是执行计划：
  - **Oracle** 用来运行一个语句的步骤就叫做执行计划（**execution plan**），执行计划包含了语句所涉及的每个表的访问路径和连接顺序。

查看执行计划

- **SQLPULS自动跟踪**
  - **1、conn /as sysdba;**
  - **2、@$ORACLE_HOME/sqlplus/admin/plustrce.sql**
  - **3、grant plustrace to user_name**
  - **4、@$ORACLE_HOME/rdbms/admin/utlxplan.sql**
  - **5、create public synonym plan_table for plan_table;**
  - **6、grant all on plan_table to user_name ;**

  注：以上步骤只需配置一次即可。
  - **7、conn username/password;**
  - **8、set autotrace traceonly（如果想看到结果集：set autotrace on，关闭：set autotrace off）**

  注：set autotrace traceonly　explain /statistics　（计划/统计）
  - **9、set timing on（如果想同时看到语句执行的时间）**
  - **10、运行你的SQL**
- **通过OEM和TOP_SQL查看**

CBO访问路径

- 访问路径
  - 访问路径就是从数据库中检索数据的方式。通常来说，检索一个表中少量的数据行应该使用索引访问，但是检索大量数据时全表扫描可能优于索引
- 全表扫描（Full Table Scans）
  - 全表扫描将读取**HWM**（高潮标记）之下的所有数据块，访问表中的所有行，每一行都要经**WHERE**子句判断是否满足检索条件。当**Oracle**执行全表扫描时会按顺序读取每个块且只读一次，因此如果能够一次读取多个数据块， 可以提高扫描效率，初始化参数**DB_FILE_MULTIBLOCK_READ_COUNT** 用来设置在一次**I/O** 中可以读取数据块的最大数量。
- 索引扫描
  - 索引不仅包含被索引字段的值，还包含表中行的位置标识**rowid**，如果语句只检索索引字段，**Oracle**直接从索引中读取该值而不去访问表，如果语句通过索引检索其他字段值，则**Oracle**通过**rowid**访问表中记录。

全表扫描（FTS）

- 优化器何时进行全表扫描
  1. 无可用索引
  2. 大量数据
  3. 小表
  4. 并行
  5. 全表扫描hints

ORACLE内部操作

当执行查询时,采用了内部的操作. 下表显示了几种重要的内部操作.

ORACLE 子句	内部操作
ORDER BY	SORT ORDER BY
UNION	UNION-ALL
MINUS	MINUS
INTERSECT	INTERSECT
DISTINCT,MINUS,INTERSECT,UNION	SORT UNIQUE
MIN,MAX,COUNT	SORT AGGREGATE
GROUP BY	SORT GROUP BY
ROWNUM	COUNT or COUNT STOPKEY
Queries involving Joins	SORT JOIN,MERGE JOIN,NESTED LOOPS
CONNECT BY	CONNECT BY

Rowid扫描

- Rowid扫描
  - Rowid就是一个记录在数据块中的位置，由于指定了记录在数据库中的精确位置，因此rowid是检索单条记录的最快方式。如果通过rowid 来访问表，Oracle 首先需要获得被检索记录的rowid，Oracle 可以在WHERE 子句中得到rowid，但更多的是通过索引扫描来获得，然后Oracle 基于rowid来定位被检索的每条记录。
  - 优化器何时使用Rowid
    - 并不是每个索引扫描都伴随着rowid的访问，如果索引中包含了被访问的所有字段，则不再需要通过rowid来访问表。
  - 注意：
    - Rowid 是Oracle 表示数据存储的内部方法，它可能会由于版本的改变而改变。不推荐通过在WHERE 中指定rowid 来访问数据，因为行迁移和行链接会导致rowid 变化，exp/imp 也会使rowid变化。

索引扫描

- 索引扫描类型：
  - 索引唯一扫描（Index Unique Scans）
  - 索引范围扫描（Index Range Scans）
  - 索引降序范围扫描（Index Range Scans Descending）
  - 索引跳跃扫描（Index Skip Scans）
  - 全索引扫描（Full Scans）
  - 快速全索引扫描（Fast Full Index Scans）
  - 索引连接（Index Joins）
  - 位图连接（Bitmap Joins）

共享SQL语句

- 共享的语句必须满足三个条件：

① 字符级的比较
  - 要求：当前被执行的语句和共享池中的语句必须完全相同

② 两个语句所指的对象必须完全相同
  - 名称相同并不代表所指的是同一个对象

③ 两个SQL语句中必须使用相同的名字的绑定变量(bind variables)

常见的连接方式

nested-loops 嵌套循环连接

- 处理过程
  - Oracle从第一个行源中读取第一行，然后和第二个行源中的数据进行对比，所有匹配的记录放在结果集中，
  - 接着Oracle将读取第一个行源中的下一行，继续操作，直到第一个数据源中的所在行都经过处理
  - 第一个行源通常称为外部表或驱动表，第二个行源称为内部表或基表，
  - 使用嵌套循环连接是一种从连接结果中提取第一批记录的最快速的方法。
- 特点
  - 在驱动表(就是您正在查找的记录)较小、或者基表已连接的列有唯一的索引或高度可选的非唯一索引时， 嵌套循环连接效果是比较理想的。
  - 嵌套循环连接比其他连接方法有优势，它可以快速地从结果集中提取第一批记录，而不用等待整个结果集完全确定下来。这样在理想情况下终端用户就可以通过查询屏幕查看第一批记录，而在同时读取其他记录。
  - 不管如何定义连接的条件或者模式，任何两行记录源可以使用嵌套循环连接，所以嵌套循环连接是非常灵活的。
- 缺点：
  - 然而，如果内部行源表(读取的第二张表)已连接的列上不包含索引，或者索引不是高度可选时， 嵌套循环连接效率是很低的，如果驱动行源表(从驱动表中提取的记录)非常庞大时，其他的连接方法可能更加有效。

常见的连接方式

- 排列合并连接 (Sort Merge Join)
  - 在排列合并连接中,Oracle分别将第一个源表、第二个源表按它们各自要连接的列排序,然后将两个已经排序的源表合并。如果找到匹配的数据,就放到结果集中。
- 特点：
  - 在缺乏数据的选择性或者可用的索引时,或者两个源表都过于庞大(超过记录数的5%)时，排序合并连接将比嵌套循环连更加高效。
  - 但是，排列合并连接只能用于等价连接(如WHERE D.deptno=E.deptno，而不是WHERE D.deptno>=E.deptno)。
- 缺点：
  - 排列合并连接需要临时的内存块,以用于排序(如果SORT_AREA_SIZE设置得太小的话)。这将导致在临时表空间占用更多的内存和磁盘I/O。

常见的连接方式

- 哈希连接 （HASH　JOIN)
  - 当内存能够提供足够的空间时,哈希(HASH)连接是Oracle优化器通常的选择。在哈希连接中，Oracle访问一张表(通常是较小的表),并在内存中建立一张基于连接键的哈希表。然后它扫描连接中其他的表（通常是较大的表），并根据哈希表检测是否有匹配的记录。
- 特点：
  - 只有在数据库初始化参数HASH_JOIN_ENABLED设为True，并且为参数PGA_AGGREGATE_TARGET设置了一个足够大的值的时候，Oracle才会使用哈希连接这和嵌套循环连接有点类似——Oracle先建立一张哈希表以利于操作进行。当使用ORDERED提示时,FROM子句中的第一张表将用于建立哈希表。
  - 当缺少有用的索引时，哈希连接比嵌套循环连接更加有效。哈希连接可能比排序合并连接更快，因为在这种情况下只有一张源表需要排序。哈希连接也可能比嵌套循环连接更快，因为处理内存中的哈希表比检索B Tree索引更加迅速。
  - 和排序合并连接、群集连接一样，哈希连接只能用于等价连接。和排序合并连接一样，
- 缺点：
  - 哈希连接使用内存资源，并且当用于排序内存不足时，会增加临时表空间的I/O（这将使这种连接方法速度变得极慢）。最后，只有基于代价的优化器才可以使用哈希连接。

常见的连接方式

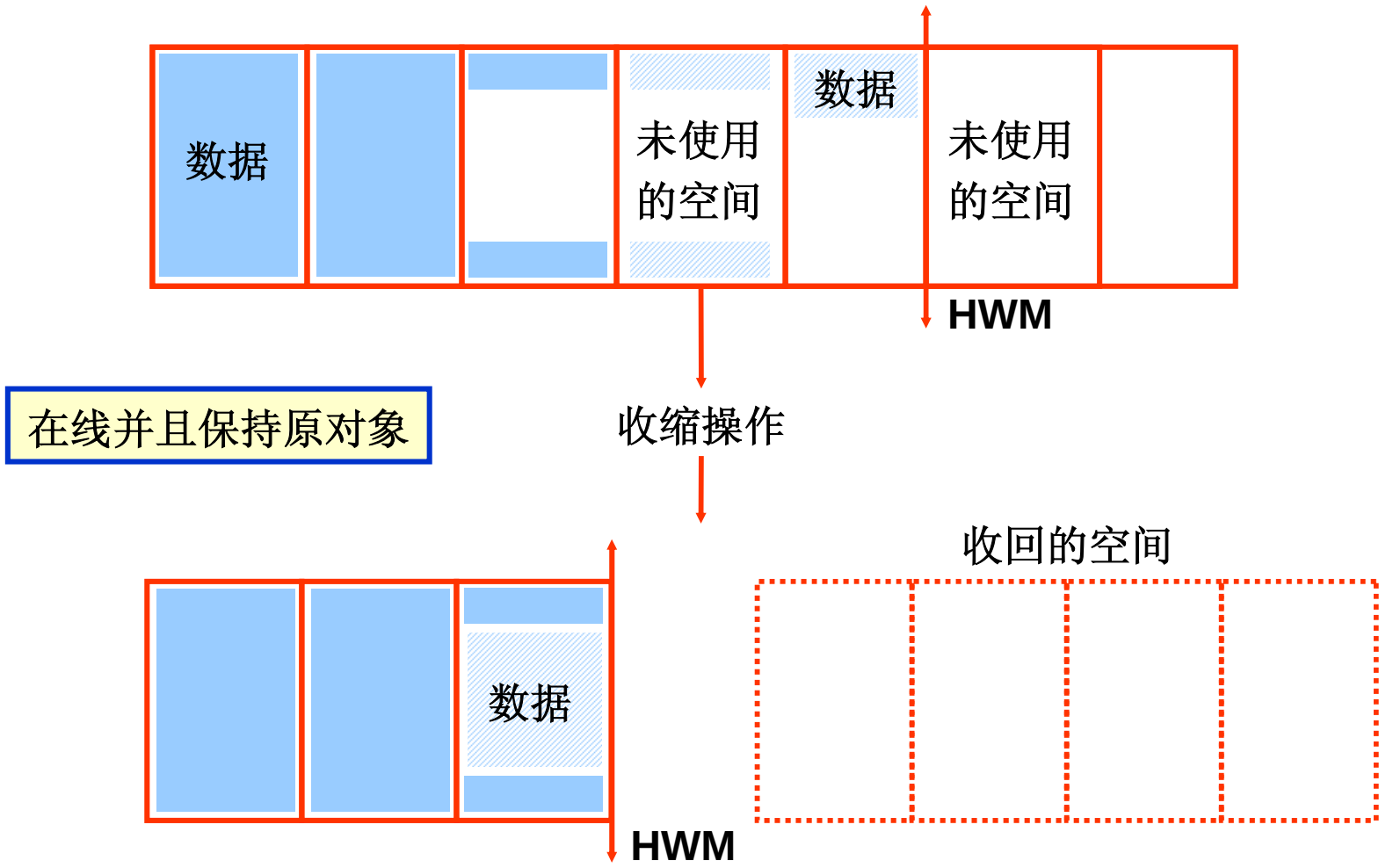
- 索引连接
  - 从**Oracle8i**起，如果一组已存在的索引包含了查询所需要的所有信息，那么优化器将在索引中有选择地生成一组哈希表。可通过范围或者快速全局扫描访问到每一个索引，而选择何种扫描方式取决于**WHERE**子句中的可用条件。在一张表有大量的列，而您只想访问有限的列时，这种方法非常有效。**WHERE**子句约束条件越多，执行速度越快。因为优化器在评估执行查询的优化路径时，将把约束条件作为选项看待。

- 说明：
  - 您必须在合适的列（那些满足整个查询的列）上建立索引，这样可以确保优化器将索引连接作为可选项之一。这个任务通常牵涉到在没有索引，或者以前没有建立联合索引的列上增加索引。相对于快速全局扫描，连接索引的优势在于：快速全局扫描只有一个单一索引满足整个查询。索引连接可以有多个索引满足整个查询。
  - 如果两个索引（一个在**ENAME**上，一个在**DEPTNO**上）创建于执行相应的查询之前。注意该查询不需要直接访问表！

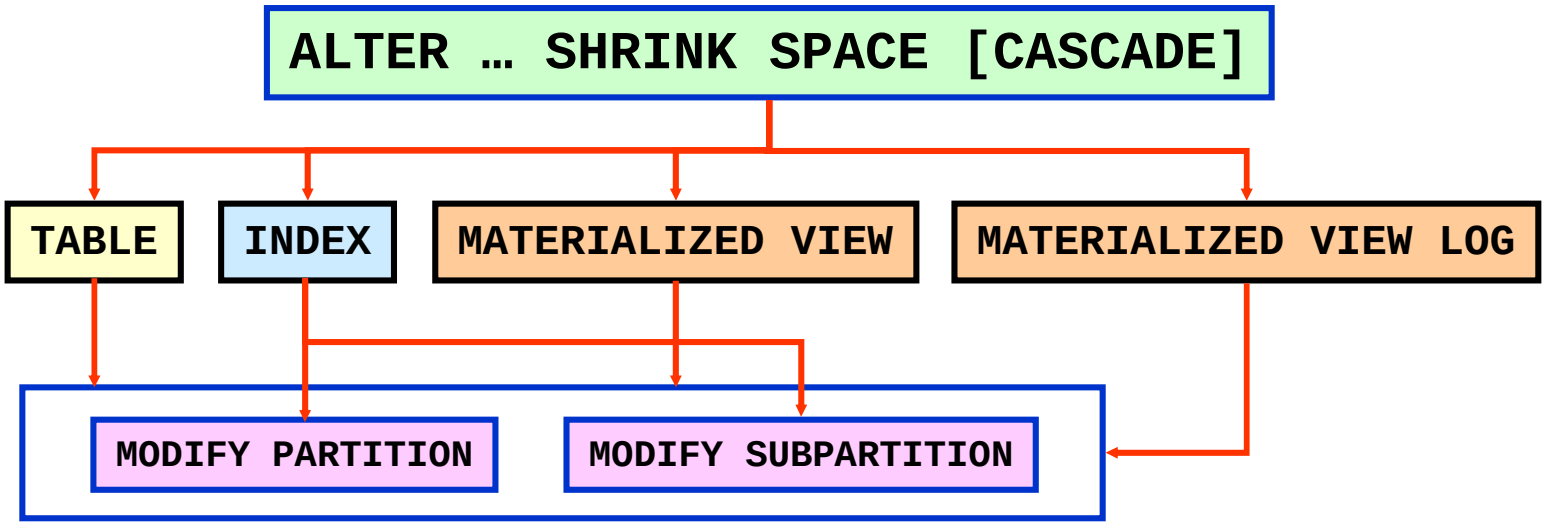
常见连接总结

- **Sort Merge Join:**
  - 对于非等值连接，这种连接方式的效率是比较高的。
  - 如果在关联的列上都有索引，效果更好。
  - 对于将2个较大的row source做连接，该连接方法比NL连接要好一些。
  - 但是如果sort merge返回的row source过大，则又会导致使用过多的rowid在表中查询数据时，数据库性能下降，因为过多的I/O。
- **Nested Loops:**
  - 如果driving row source(外部表)比较小，并且在inner row source(内部表)上有唯一索引，或有高选择性非唯一索引时，使用这种方法可以得到较好的效率。
  - NESTED LOOPS有其它连接方法没有的的一个优点是：可以先返回已经连接的行，而不必等待所有的连接操作处理完才返回数据，
    这可以实现快速的响应时间。
- **Hash Join:**
  - 只用在CBO优化器中，一般来讲优于以上两种连接，但需要合适的hash_area_size参数。
  - 在2个较大的row source之间连接时会取得相对较好的效率，在一个row source较小时则能取得更好的效率
- **笛卡儿乘积(Cartesian Product):**
  - 无关联条件时即作cartesian product

段收缩概述



如何才能收缩段呢？




```
ALTER TABLE employees ENABLE ROW MOVEMENT;    1
```

```
ALTER TABLE employees SHRINK SPACE CASCADE;   2
```

索引的优化

- 索引分为：b tree索引和bitmap索引
- 索引使用的优化原则及注意事项：
  - 用于平衡查询和DML
  - 小表不使用索引，对查询获得数据的范围不超过2%-4%的表上建立索引
  - 表上索引的个数，一般不超过5个，索引树的高度一般不超过4
  - 创建大的索引可以使用nologging
  - 复合索引优于单列索引，函数索引优于索引列运算

索引的优化

- 通常将数据和索引分开存储，以提高系统运行速度，索引表空间常为数据表空间的1/2
- 索引的initrans比其对应表的稍大一些
- 合理的使用反向键索引
- 索引字段不可以参与运算
- 索引的定期重整
  - 当索引的的高度大于4层时考虑重建。
  - 当索引的消耗大于20%时需要重建。
  - 当表被MOVE操作时需要重建
- 其它

索引的详细优化

- 索引列不能参与运算
  - 导致索引失效
  - 常见的运算(如：列为Id)
    - **Id+1=990**
    - **Id <> 9**
    - **Id like ‘%900’**
    - **substr(id,1,4)=xxxx**
    - **Id is null**
    - **Id is not null**
    - **……**

索引的详细优化

- 索引列的类型转换
  - 有时默认的类型转换会导致索引失效
  - 如：**where charcol>100**
- 增加查询的范围限制
  - 通过适当的范围限制，避免全范围查询

Where mytime <to_date(….) and mytime > to_date(….)

索引的详细优化

- Where 子句中使用IN或OR索引可能会失效
- 使用索引提示“/*+ INDEX(表名称，索引名称）*/”
- 使用nls_date_format
  - select * from record where to_char(myTime,'mm')='12‘
  - SQL>alter session set nls_date_formate=’MM’;
  - select * from record where myTime='12'

索引的详细优化

- 当有两个或以上的索引具有相同的等级，有时为了提高查询效率可以强制只使用一个。
  - 假如emp表中的列empno有一个非唯一索引，优化器会使用该索引， 这是目前唯一的选择. 如果,一段时间以后, 另一个非唯一性建立在job上,优化器必须对两个索引进行选择,在通常情况下,优化器将使用两个索引并在他们的结果集合上执行排序及合并. 然而,如果其中一个索引（empno）接近于唯一性而另一个索引（job）上有很多个重复的值， 排序及合并就会成为一种不必要的负担， 在这种情况下，你希望使优化器屏蔽掉empno索引。
  - 使用索引时避免在索引上使用计算
  - 避免在索引上使用NOT
  - 某些情况下优化器会自动将NOT转化成相对应的关系操作符

索引的维护

- 分析索引

  analyze index scott.a_in validate structure

- 查看索引信息

  select * from index_stats;

- 统计消耗百比分

  select del_lf_rows_len/lf_rows_len from index_stats;

- 重建索引

  alter index scott.a_in rebuild;

重整所有索引的方法

```
Spool D:\index.sql
SELECT 'alter index ' || index_name || ' rebuild;'
   FROM all_indexes
   WHERE owner = USER;
Spool off
```

索引的迁移

- 如果索引与数据存放于同一个表空间可用下列命令将索引迁移到索引表空间中
  - **Alter index index_name rebuild tablespace table_space_name storage(parameter ….)**

多个平等的索引

- 当SQL语句的执行路径可以使用分布在多个表上的多个索引时, ORACLE会同时使用多个索引并在运行时对它们的记录进行合并, 检索出仅对全部索引有效的记录
- 在ORACLE选择执行路径时,唯一性索引的等级高于非唯一性索引. 然而这个规则只有当WHERE子句中索引列和常量比较才有效，如果索引列和其他表的索引列相比较。 这种子句在优化器中的等级是非常低的。
- 如果不同表中两个想同等级的索引将被引用， FROM子句中表的顺序将决定哪个会被率先使用， FROM子句中最后的表的索引将有最高的优先级。
- 如果相同表中两个想同等级的索引将被引用，WHERE子句中最先被引用的索引将有最高的优先级。

用EXISTS替代IN

- IN操作等价于OR操作
- 减少使用OR或IN操作

  在许多基于基础表的查询中,为了满足一个条件,往往需要对另一个表进行联接.在这种情况下, 使用EXISTS(或NOT EXISTS)通常将提高查询的效率

用NOT EXISTS替代NOT IN

- 高效的关联子查询

  在子查询中,NOT IN子句将执行一个内部的排序和合并， 无论在哪种情况下，NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历)。 为了避免使用NOT IN ,我们可以把它改写成外连接(Outer Joins)或NOT EXISTS。

用表连接替换EXISTS

- 高效的表连接操作

  通常来说，采用表连接的方式比EXISTS更有效率 。

用EXISTS替换DISTINCT

- 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EXISTS替换

- 注意:

  EXISTS 使查询更为迅速,因为RDBMS核心模块将在子查询的条件一旦满足后,立刻返回结果。

Is null 和 Is not null

- 判断字段是为空
  - 判断字段是否为空一般是不会应用索引的，因为索引中不包含空值。
  - 优化思路：
    - 用其它相同功能的操作运算代替
    - 如：**a is not null** 改为 **a>0** 或**a>’’**等

Like、<>操作符

- **LIKE操作符可以应用通配符查询，需要注意的是**
  - **Like ‘%xxxxx’　索引失效，全表扫描**
  - **Like ‘xxxx%’　使用索引**

- **<> 操作符（不等于）**
  - **不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。**
  - **优化思路：用其它相同功能的操作运算代替，如**

  a<>0 改为 a>0 or a<0

  a<>’’ 改为 a>’’

有效的表名顺序（RBO）

- 在RBO优化器下
  - ORACLE的解析器按照从右到左的顺序处理FROM子句中的表名
  - 因此FROM子句中写在最后的表(基础表 driving table)将被最先处理
  - 在FROM子句中包含多个表的情况下，你必须选择记录条数最少的表作为基础表
  - 当ORACLE处理多个表时， 会运用排序及合并的方式连接它们.首先,扫描第一个表(FROM子句中最后的那个表)并对记录进行排序，然后扫描第二个表(FROM子句中最后第二个表),最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。

WHERE子句中的连接顺序

- 在RBO优化器下
  - ORACLE采用自下而上的顺序解析WHERE子句
  - 根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾

SELECT子句中避免使用‘*’

- PL/SQL操作中少用’*’
  - 在SELECT子句中列出所有的COLUMN时，经常使用动态SQL列引用 ‘*’
  - 这是一个非常低效的方法.
  - ORACLE在解析的过程中, 会将'*' 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间.

    Select * from emp;

减少访问数据库的次数

- 减少访问数据库的次数：
  - 当执行每条SQL语句时, ORACLE在内部执行了许多工作: 解析SQL语句, 估算索引的利用率, 绑定变量 , 读数据块等等. 由此可见, 减少访问数据库的次数 , 就能实际上减少ORACLE的工作量
  - 使用物化视图、with子句等操作

注意:

在SQL*Plus , SQL*Forms和Pro*C中重新设置ARRAYSIZE参数, 可以增加每次数据库访问的检索数据量 ,建议值为200

使用DECODE函数来减少处理时间

- 使用Oracle自带的函数
  - 使用DECODE函数可以避免重复扫描相同记录或重复连接相同的表。
  - DECODE函数用法：

DECODE(条件,值1,翻译值1,值2,翻译值2,...值n,翻译值n,缺省值)

该函数的含义如下：

```
IF 条件=值1 THEN
RETURN(翻译值1)
ELSIF 条件=值2 THEN
RETURN(翻译值2)
......
ELSIF 条件=值n THEN
RETURN(翻译值n)
ELSE
RETURN(缺省值)
END IF
```

删除重复记录

- 最高效的删除重复记录方法（因为使用了**ROWID**）

```
DELETE FROM EMP E

WHERE E.ROWID > (SELECT MIN(X.ROWID)

                 FROM EMP X

                 WHERE X.EMP_NO = E.EMP_NO);
```

用TRUNCATE替代DELETE

- 说明：

  当删除表中的记录时,在通常情况下, 回滚段(rollback segments)用来存放可以被恢复的信息. 如果你没有COMMIT事务,ORACLE会将数据恢复到删除之前的状态(准确地说是恢复到执行删除命令之前的状况)

- 注：而当运用TRUNCATE时, 回滚段不再存放任何可被恢复的信息. 当命令运行后,数据不能被恢复.因此很少的资源被调用,执行时间也会很短，truncate是DDL语句。
- 删除全表或分区

尽量多使用COMMIT

- 合理的事务处理
  - 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。

    注：使用COMMIT时必须要注意到事务的完整性, 特别注意发生死锁时，慎用commit.

计算记录条数

- 统计计算性能
  - 通常 **count(*)** 比**count(1)**稍快
  - 当然如果可以通过索引检索,对索引列的计数仍旧是最快的.
  - 例如 **COUNT(EMPNO)** 。
  - 注：跟机器的硬件配置和环境有一定关系，三者的性能无明显的差别。

用Where子句替换HAVING子句

- 尽能先过滤再统计
  - 避免使用HAVING子句, HAVING 只会在检索出所有记录之后才对结果集进行过滤. 这个处理需要排序,合计等操作.
  - 如果能通过WHERE子句限制记录的数目,那就能减少这方面的开销.
  - 注意:

    HAVING 中的条件一般用于对一些集合函数的比较,如COUNT() 等等. 除此而外,一般的条件应该写在WHERE子句中。

减少对表的查询

- 在子查询中减少对表的查询
  - 在含有子查询的SQL语句中,要特别注意减少对表的查询
  - With子句的使用
  - Insert all子句的使用
  - Merge子句的使用

通过内部函数提高SQL效率

- 分析函数的主要用法：

  Analytic-Function(<Argument>,<Argument>,...)

  OVER (

  <Query-Partition-Clause>

  <Order-By-Clause>

  <Windowing-Clause>

)

  - 其中**over()**被称为开窗函数，开窗函数指定了分析函数工作的数据窗口大小，这个数据窗口大小可能会随着行的变化而变化，如：
    - **over（order by salary）** 按照**salary**排序进行累计，**order by**是个默认的开窗函数
    - **over（partition by deptno）**按照部门分区
    - **over（order by salary range between 50 preceding and 150 following）**

      每行对应的数据窗口是之前行幅度值不超过**50**，之后行幅度值不超过**150**

通过内部函数提高SQL效率

- **Over函数的说明：**
  - **over（order by salary rows between 50 preceding and 150 following）**
    每行对应的数据窗口是之前50行，之后150行
  - **over（order by salary rows between unbounded preceding and unbounded following）**
    每行对应的数据窗口是从第一行到最后一行，等效：
  - **over（order by salary range between unbounded preceding and unbounded following）**
- **常用的分析函数**
  - **AVG**
  - **COUNT**
  - **SUM**
  - **CUME_DIST**
  - **DENSE_RANK**

通过内部函数提高SQL效率

- 常用的分析函数
  - FIRST_VALUE
  - LAG
  - LEAD
  - LAST_VALUE
  - RANK
  - DENSE_RANK
  - ROW_NUMBER

使用表的别名(Alias)

- 合理使用别名：
  - 当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误.

    注: Column歧义指的是由于SQL中不同的表具有相同的Column名,当SQL语句中出现这个Column时,SQL解析器无法判断这个Column的归属

识别'低效执行'的SQL语句

- 用下列SQL语句找出低效SQL

```
SELECT EXECUTIONS , DISK_READS, BUFFER_GETS,
    ROUND((BUFFER_GETS-DISK_READS)/BUFFER_GETS,2) Hit_radio,
   ROUND(DISK_READS/EXECUTIONS,2) Reads_per_run,
   SQL_TEXT
  FROM V$SQLAREA
  WHERE EXECUTIONS>0
     AND BUFFER_GETS > 0
     AND (BUFFER_GETS-DISK_READS)/BUFFER_GETS < 0.8
    ORDER BY 4 DESC
```

基础表的选择

- 基础表(Driving Table)是指被最先访问的表(通常以全表扫描的方式被访问)， 根据优化器的不同， SQL语句中基础表的选择是不一样的：
  - 如果你使用的是CBO (COST BASED OPTIMIZER)，优化器会检查SQL语句中的每个表的物理大小，索引的状态，然后选用花费最低的执行路径。
  - 如果你用RBO (RULE BASED OPTIMIZER)， 并且所有的连接条件都有索引对应， 在这种情况下， 基础表就是FROM 子句中列在最后的那个表。

等式比较和范围比较

- 在有等式比较和范围比较的WHERE子句中有索引列,ORACLE不能合并它们,ORACLE将用范围比较。
- 不明确的索引等级
  - 当ORACLE无法判断索引的等级高低差别,优化器将只使用一个索引,它就是在WHERE子句中被列在最前面的.

Where中用>=替代>

Select * from emp where empno>=4;

Select * from emp where empno>3;

- 两者的区别在于, 前者DBMS将直接跳到第一个empno等于4的记录而后者将首先定位到empno=3的记录并且向前扫描到第一个empno大于3的记录

用UNION替换OR (适用于索引列)

- 通常情况下，用UNION替换WHERE子句中的OR将会起到较好的效果。对索引列使用OR将造成全表扫描（CBO）

- 注意：
  - 以上规则只针对多个索引列有效，如果有COLUMN没有被索引查询效率可能会因为你没有选择OR而降低
  - 如果需要用OR则要将返回记录最少的索引列写在最前面。

总是使用索引的第一个列

- 如果索引是建立在多个列上，只有在它的第一个列(leading column)被where子句引用时，优化器才会选择使用该索引。

用UNION-ALL 替换UNION

- 根据前面的SQL语句内部的操作，我们可以看出当SQL语句需要UNION两个查询结果集合时，这两个结果集合会以UNION-ALL的方式被合并，然后在输出最终结果前进行排序，如果用UNION ALL替代UNION，这样排序就不是必要了。效率就会因此得到提高。
- 注意：

  UNION ALL 将重复输出两个结果集合中相同记录，因此还是要从业务需求分析使用UNION ALL的可行性UNION 将对结果集合排序，并且相同的只取其一，这个操作会使用到SORT_AREA_SIZE这块内存。对于这块内存的优化也是相当重要的。

使用常见的Hints

- 使用Hints是我们手工修改ORACLE优化器缺省执行路径的好方法
  - FULL和ROWID
  - CACHED
  - INDEX
  - USE_NL
  - USE_MERGE
  - USE_HASH

避免使用耗资源的操作

- SQL语句中使用带有DISTINCT，UNION，MINUS，INTERSECT，ORDER BY的SQL语句会启动SQL引擎去执行耗费资源的排序(SORT)功能，DISTINCT需要一次排序操作，而其他的至少需要执行两次排序。

日期操作的技巧

- 使用Oracle自定义的日期操作
  - Add_months()
  - Current_date()
  - Current_timestamp()
  - Dbtimezone()
  - Extract()
  - Last_day
  - Next_day
  - Months_between()
  - Trunc()

用WHERE替代ORDER BY

- **ORDER BY** 子句只在两种严格的条件下才使用索引
  - **ORDER BY**中所有的列必须包含在相同的索引中并保持在索引中的排列顺序。
  - **ORDER BY**中所有的列必须定义为非空。

- 注意：

  WHERE子句使用的索引和**ORDER BY**子句中所使用的索引不能并列.

避免改变索引列的类型

- 在oracle中当比较不同数据类型的数据时，ORACLE自动对列进行简单的类型转换。当对索引列的类型作转换时有时会使索引失效。

- 注：

  当有函数对索引操作时，使用基于函数的索引会会更好。

等于条件与其他条件位置顺序

- 通常将where子句中的等于条件放在其它条件之前这是因为：
  - 对于关联的查询，等于条件与其他条件位置顺序不会对查询结果造成影响。
  - 对于存在exists语句的查询，将where 后的等于条件放在exists 之前，将会提高查询的速度。

其它优化SQL语句