# 诗檀软件

# Oracle开发优化基础

汪伟华

DOC#: ZXW-7

古希腊的Delphi(世界中心)，屹立着Parnassus Mount(诗檀山)，山上有一座阿波罗神庙，庙中住着女祭司(Oracle)

# 议程

- 数据库开发员需要注意些什么
- 如何快速定位及认知数据库问题点
- 如何编写高性能SQL – 基础知识
- 如何编写高性能SQL – 执行计划
- 调整思路

# 数据库开发员需要注意些什么

当前应用开发现状 - 从业人员

- 应用开发人员的数据库基础参差不一
  - 大量1-3年工作经验的程序员
  - 缺乏数据库基础知识和优化经验
- 更注重应用功能实现
  - 不关注数据库性能
- 最懂数据库的优化人员成为救急人员

# 数据库开发员需要注意些什么

对SQL开发初期优化普遍关注度不够

- 数据库像个黑盒子？
  - 数据库总是访问通用API且其框架机制复杂隐蔽【SQL开发回避，不关心】
  - SQL实现在还未模块化就先上马运行【SQL开发没想透】
  - 如果SQL返回值正确，那就OK 【SQL开发普遍心态】

- 结果

  **在测试环境上运行良好的SQL语句，在生产环境却发生了性能问题！！**

# 数据库开发员需要注意些什么

RDB下的SQL开发的注意点

- 代码重用
  - 模块化
  - 共享池
- 使用适当方法
  - 针对数据特点，采用适当算法
- 消除浪费的处理
  - 减少重复(循环)处理

- SQL语句重用

提高Oracle内部的代码重用率

- 使用适当的索引

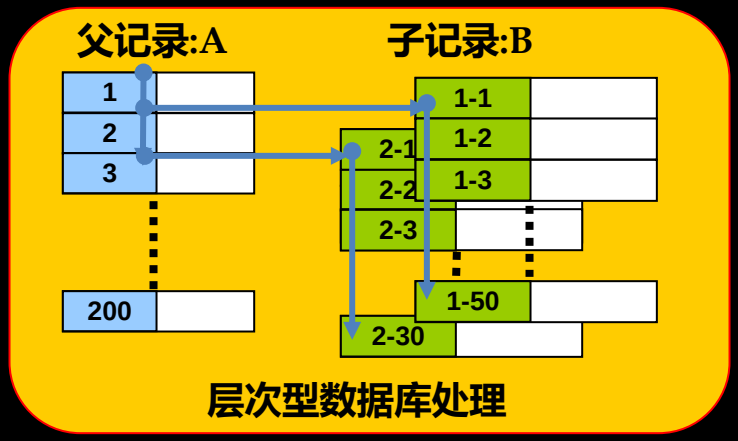
明确搜寻数据的条件

- 减少SQL执行时间

适当的表连接，在Oracle内部减少重复处理

# 数据库开发员需要注意些什么

## 对关系型数据库的理解




处理逻辑

```
WHILE (A的值为1或2) {
  查找并获得A对应信息(SELECT)
  WHILE (当B的键与A当前记录相关) {
  查找并获得B对应信息(SELECT)
  }
}
```

**发起Select语句的数量与循环的量相同**

基表:A

| AK | N1 |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| : | |
| 200 | |

从属表:B

| BK | AK | C1 |
|---|---|---|
| 1-1 | 1 | |
| : | : | |
| 1-50 | 1 | |
| 2-1 | 2 | |
| : | : | |
| 2-30 | 2 | |

关系型数据库处理

RDB使用SQL

```
SELECT A.N1, B.C1
  FROM A,B
 WHERE A.AK = B.AK
   AND A.AK IN (1, 2);
```

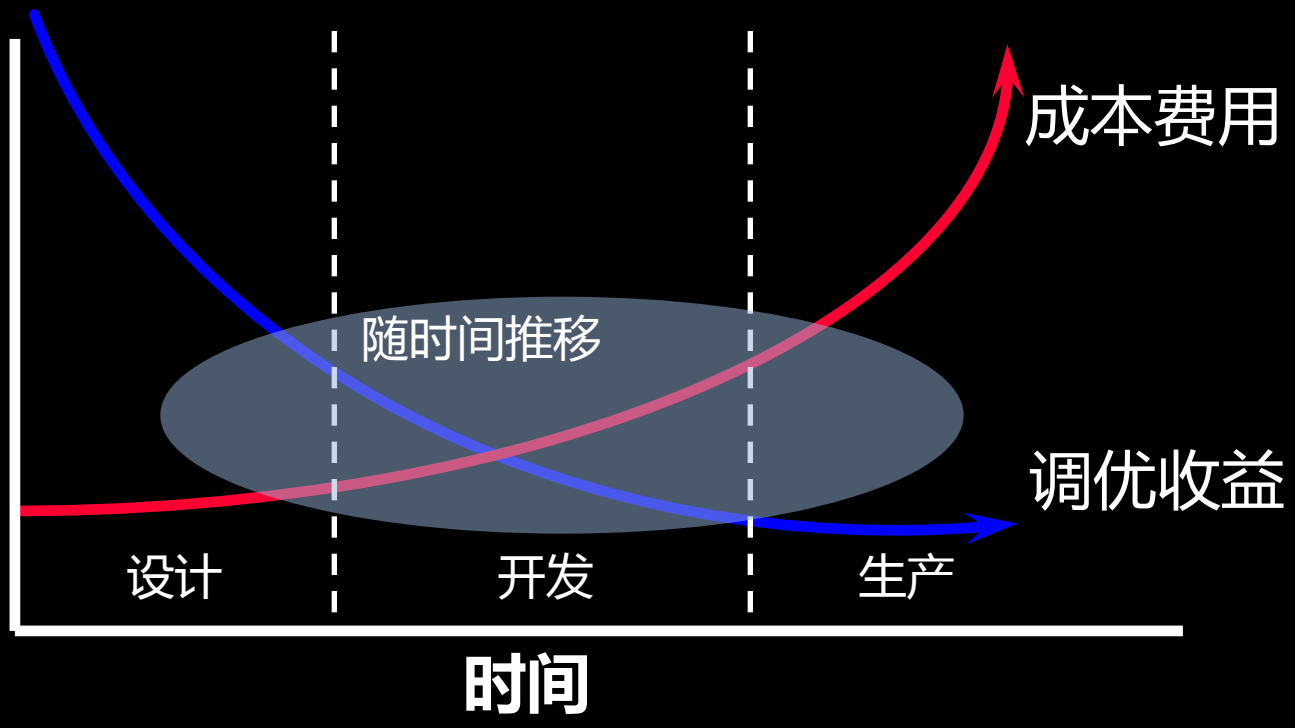
# 数据库开发员需要注意些什么

如何做到性能下降较少的DB应用开发

- 从DB处理角度来看潜在的应用问题
  - 了解并承认DB处理瓶颈
- 理解Oracle基本操作
  - 避免一些不佳的编码 (如单个SQL运行没问题，但在Oracle整体会有问题)
  - 了解优化器行为，从而编写Oracle所期望的SQL
- 了解操作中的所用到的数据
  - 尽管开发员在处理逻辑（条件分支）中已经意识到了这一点，但一般不会意识到所处理的数据量问题

# 数据库开发员需要注意些什么

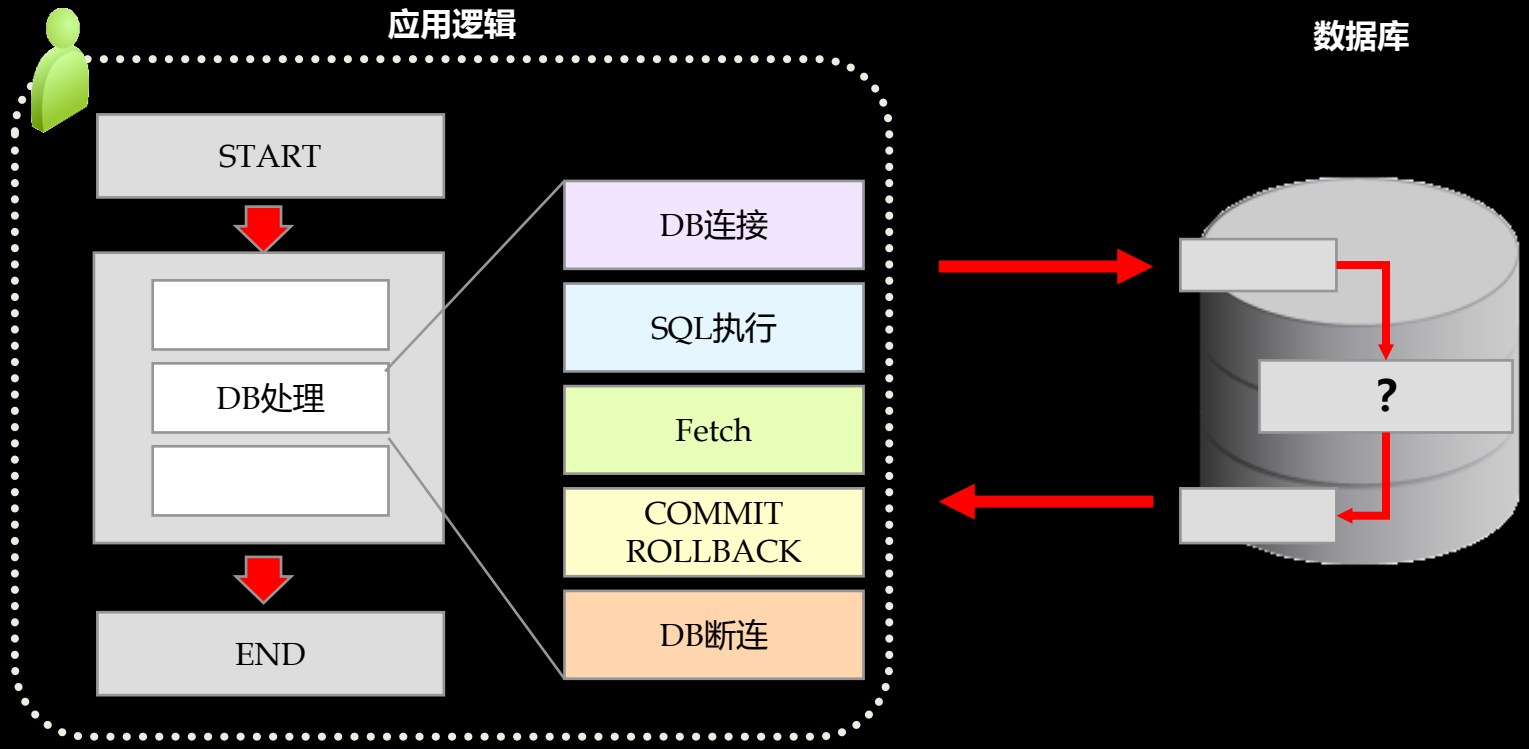
优化成本收益比(设计 > 开发 > 生产)



# 如何快速定位及认知数据库问题点

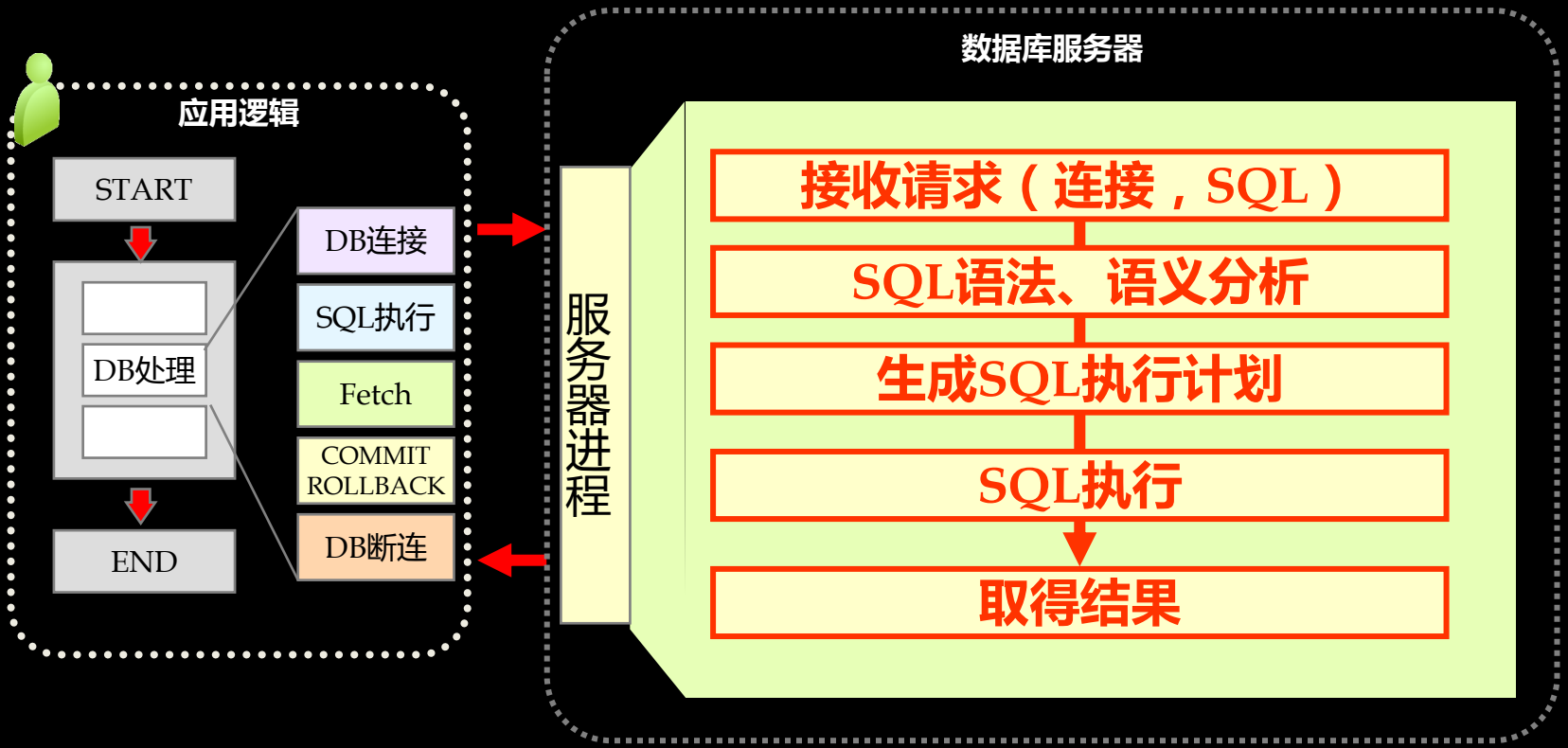
## 对于应用数据库

- 从应用角度出发，用过DB连接驱动程序(JDBC驱动等)，访问数据库



# 如何快速定位及认知数据库问题点

## 应用程序SQL发出后的DB内部处理

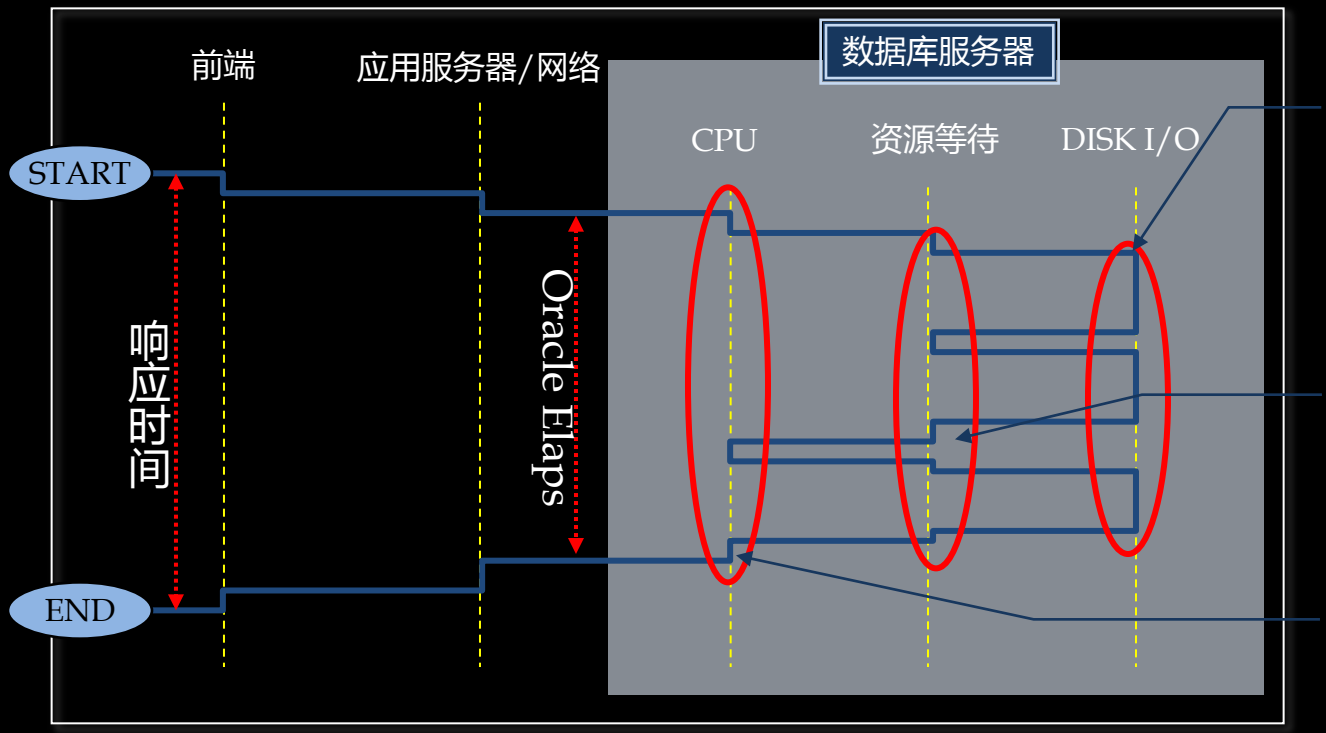


# 如何快速定位及认知数据库问题点

## 数据库处理时间

- 通过下图，了解哪些因素可能会成为瓶颈。可以看到数据库处理时间被细分成三部分：

  1）CPU处理时间　2）等待时间(资源等待，同步处理)　3）等待时间(磁盘I/O处理)




**可成为瓶颈的因素**

- 非规范化表设计
- 数据放置不合理
- 执行计划不正确
- 数据碎片化

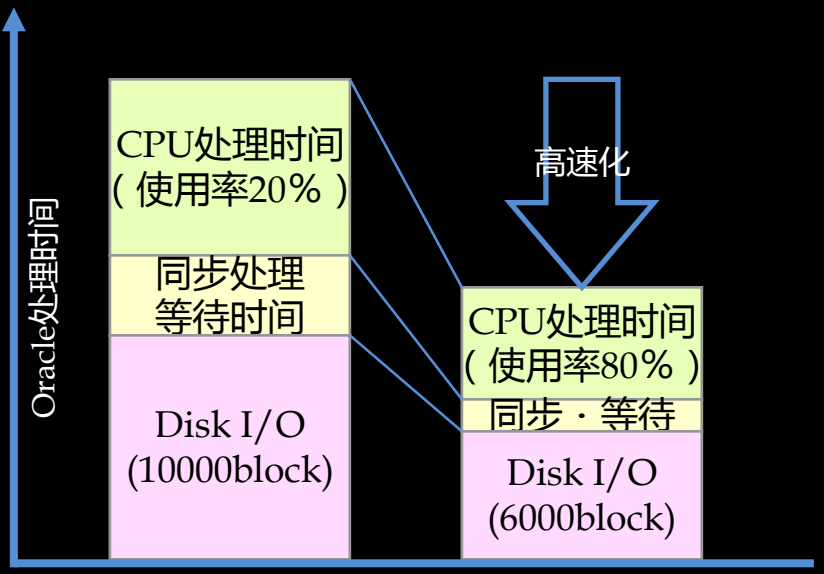
- 不必要的SQL解析
- 并发(资源竞争)
- 资源枯竭

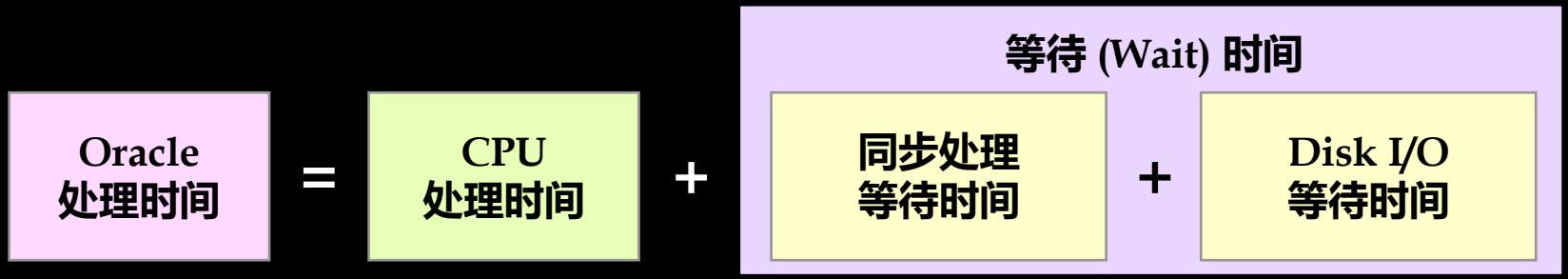
- 不必要的SQL解析
- 结合负载排序

# 如何快速定位及认知数据库问题点

如何使数据库处理得更快

- 有效利用CPU资源
- 减少等待时间
  - 降低磁盘I/O
  - 降低同步处理所需时间



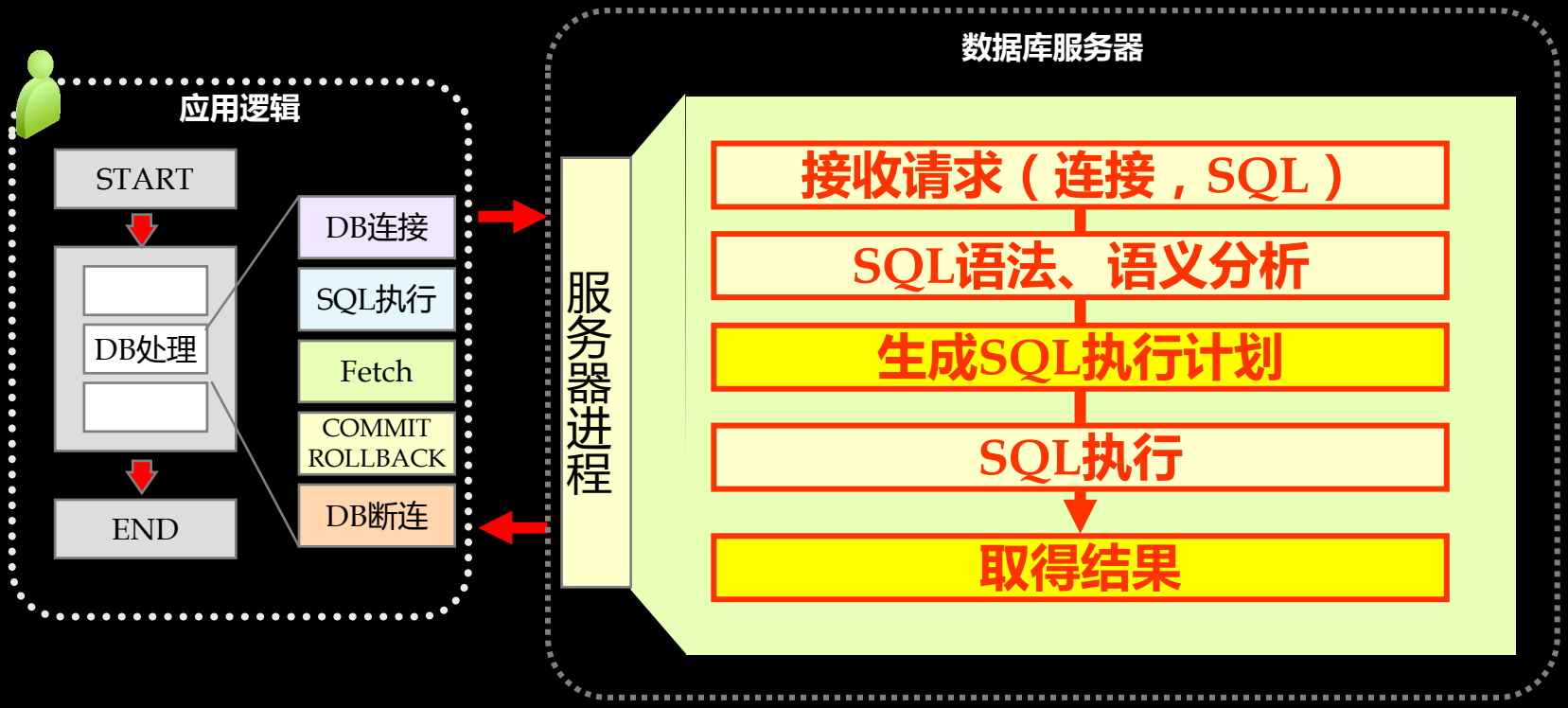



# 如何快速定位及认知数据库问题点

## 如何有效利用CPU

- 在于怎样写好SQL语句，并优化数据库内部处理
  - SQL语句的重用
  - 表连接

非常有效的SQL调优



# 如何快速定位及认知数据库问题点

- 处理时间

（其中CPU的处理使用状态）

Time Model System Stats DB/Inst: ORACLE10/oracle10g Snaps: 1-2
-> Ordered by % of DB time desc, Statistic name

| Statistic | Time (s) | % of DB time |
|---|---|---|
| sql execute elapsed time | 83.6 | 92.7 |
| DB CPU | 13.9 | 15.4 |
| parse time elapsed | 9.7 | 10.7 |
| hard parse elapsed time | 9.6 | 10.7 |
| connection management call elapsed | 0.1 | .1 |
| PL/SQL execution elapsed time | 0.1 | .1 |
| PL/SQL compilation elapsed time | 0.0 | .0 |
| repeated bind elapsed time | 0.0 | .0 |
| failed parse elapsed time | 0.0 | .0 |
| DB time | 90.2 | |
| background elapsed time | 14.6 | |
| background cpu time | 0.7 | |

CPU是否得到了有效使用？

- 等待时间

（资源等待，等待完成处理的状态）

某处是否存在瓶颈？

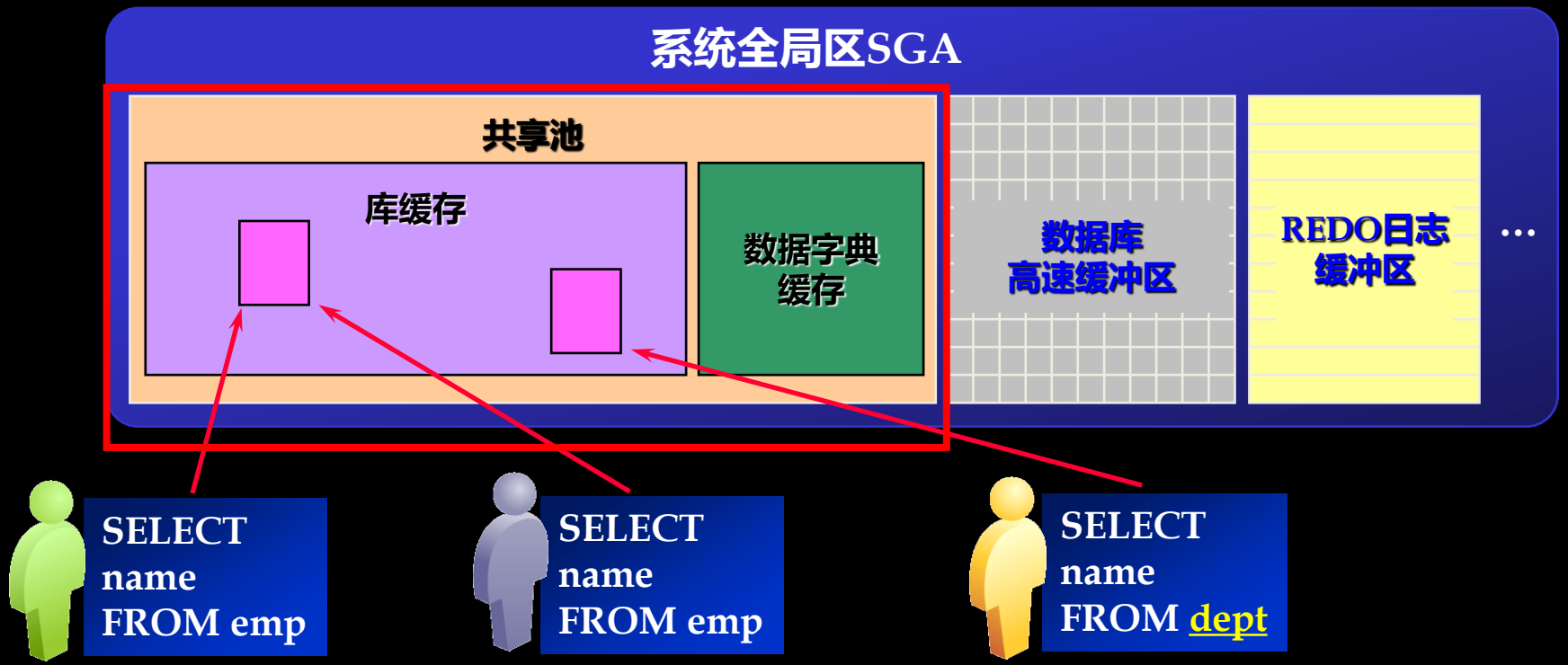
Wait Events DB/Inst: ORACLE10/oracle10g Snaps: 1-2
-> s - second, cs - centisecond, ms - millisecond, us - microsecond
-> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0
-> Only events with Total Wait Time (s) >= .001 are shown
-> ordered by Total Wait Time desc, Waits desc (idle events last)

| Event | Waits | %Time -outs | Total Wait Time (s) | Avg wait (ms) | Waits /txn |
|---|---|---|---|---|---|
| read by other session | 2,404 | 0 | 31 | 13 | 72.8 |
| db file scattered read | 1,019 | 0 | 25 | 24 | 30.9 |
| db file sequential read | 2,323 | 0 | 10 | 4 | 70.4 |
| db file parallel write | 1,355 | 0 | 6 | 5 | 41.1 |
| library cache load lock | 45 | 0 | 5 | 103 | 1.4 |
| control file sequential read | 601 | 0 | 3 | 6 | 18.2 |
| control file parallel write | 152 | 0 | 2 | 12 | 4.6 |
| log file parallel write | 47 | 0 | 1 | 12 | 1.4 |
| log file sync | 13 | 0 | 0 | 15 | 0.4 |
| row cache lock | 51 | 0 | 0 | 4 | 1.5 |
| SQL*Net more data to client | 475 | 0 | 0 | 0 | 14.4 |
| cursor: pin S wait on X | 3 | 100 | 0 | 11 | 0.1 |
| direct path write temp | 4 | 0 | 0 | 8 | 0.1 |
| os thread startup | 1 | 0 | 0 | 12 | 0.0 |
| latch free | 6 | 0 | 0 | 1 | 0.2 |
| SQL*Net message from client | 448 | 0 | 684 | 1528 | 13.6 |
| Streams AQ: qmn slave idle wait | 13 | 0 | 364 | 27999 | 0.4 |
| Streams AQ: qmn coordinator idle | 26 | 50 | 364 | 13999 | 0.8 |
| jobq slave wait | 121 | 96 | 363 | 2997 | 3.7 |
| virtual circuit status | 12 | 100 | 360 | 30000 | 0.4 |

# 如何编写高性能SQL – 基础知识

## SQL语句重用 – 共享池

- 解析SQL语句的执行计划被存储在共享SQL区(Shared SQL Area)
- 如果每个用户已经运行相同的SQL，并使用相同的共享SQL区



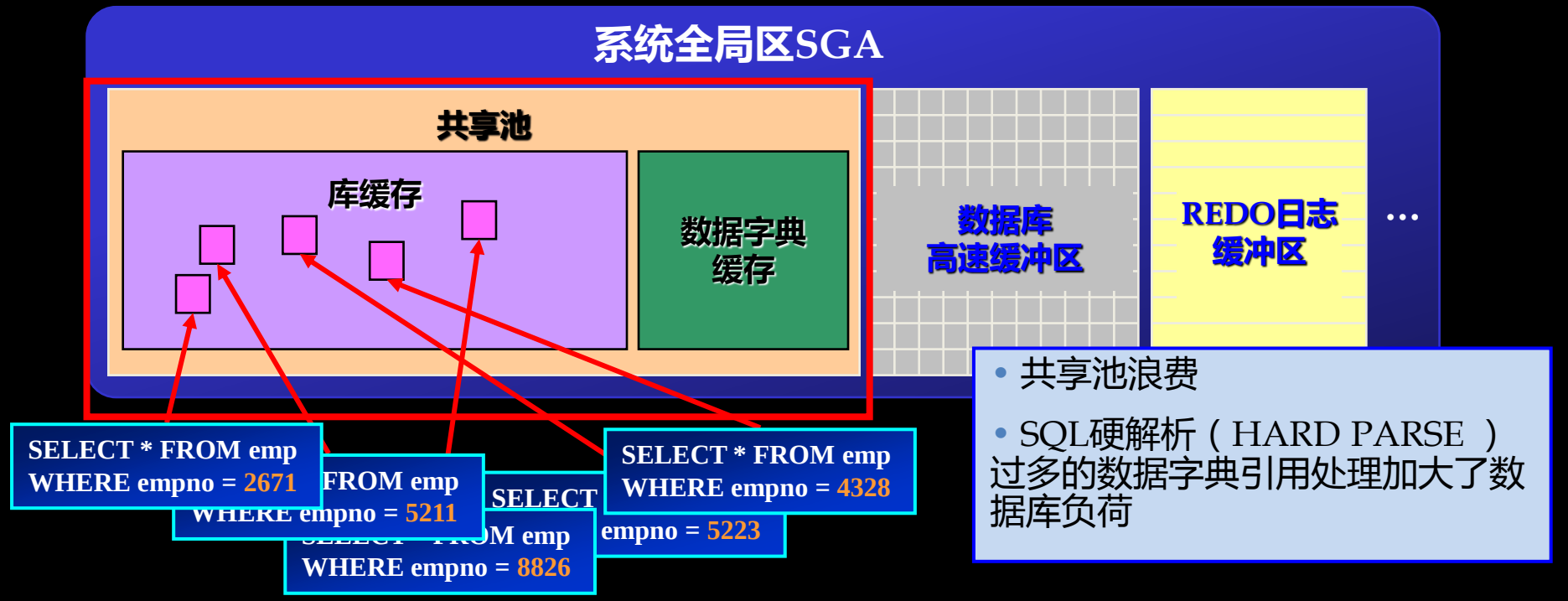
# 如何编写高性能SQL – 基础知识

## SQL语句重用（1）

- 使用绑定变量

```
SELECT * FROM EMP WHERE EMPNO = :v_empno
```

- 不使用绑定变量



# 如何编写高性能SQL – 基础知识

SQL语句重用（2）

- 创建一个SQL编码规范，并按规范进行编码。
  （**请注意: 下列看似相同的语句，Oracle并不认为其相同！！因此并不会得到重用**）

```
SELECT * FROM EMP WHERE EMPNO = :v_empno
```

```
SELECT * FROM EMP
WHERE EMPNO = :v_empno
```

```
SELECT * FROM EMP
WHERE EMPNO = :v_empno
```

# 如何编写高性能SQL – 基础知识

根据表的特点及数据增长趋势来判断（1）

- 特性表

| 类型 | 行数 | 更新时间频率 | 数据标识列 |
| --- | --- | --- | --- |
| 基表 | 少 | 每天，频率：小 | 主键 |
| 表显示了基表的相互（交叉表）的关系（*1） | 少 | 相关切换时间（年，月），频率：小 | 相关的主表的关键列 |
| 从属表（详细） | 多 | 每天，频率：大 | 列数据（标志类型），日期（交易的日期等）的状态 |
| 历史表 | 多 | 日报（添加数据）;每年，每月（数据删除） | 主键+日期 |

- 根据增长趋势来判断

数据量　　该列值的分布，即表示数据的状态

# 如何编写高性能SQL – 基础知识

## 根据表的特点及数据增长趋势来判断（2）

- 仔细参照业务分析人员对表业务的定义
  - 大致的数据编号，表定义，索引定义
- 所获取信息，这将对后继优化有帮助（按DBA要求）
  - 数量：

```
SELECT count(*) FROM <表名>;
```

  - 数据分布：

```
SELECT count(distinct A_id) FROM <表名>;
SELECT <列名>, count(*) FROM <表名>
 GROUP BY <列名>;
```

- 除了上面提到的
  - 索引信息：

```
SELECT i.table_name,i.index_name,
       ic.column_position,ic.column_name
  FROM user_indexes i, user_ind_columns ic
 WHERE i.index_name = ic.index_name
 ORDER BY i.table_name,
          i.index_name,
          ic.column_position;
```

# 如何编写高性能SQL – 基础知识

## 细化数据

- 哪些字段常被用于检索？
  - 订单表
- 表分析：
  - 所有订单：增加5000000张（五年），增长速度约为2500张/每天
  - 客户数量：50,000
  - 订单状态：“1”（订单），‘2’ - ‘4’（其它状态），‘9’（完成）

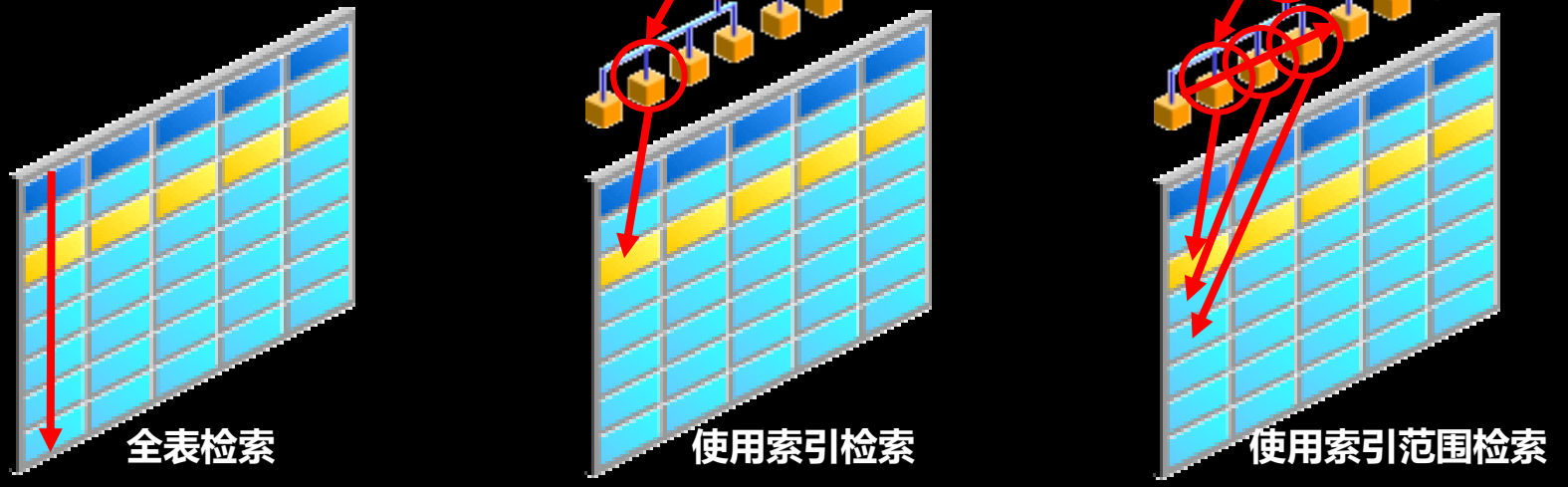
| 订单号 | 订购日期 | 客户号 | 订单状态 |
|---|---|---|---|
| 1000001 | 2007/10/10 | 2345 | 1 |
| 1000002 | 2007/10/10 | 8733 | 9 |

| 需查询的列名 | 数据统计 | 查询率 |
|---|---|---|
| 订单编号 | 按订单划分： 1张 | 1/5,000,000 |
| 订购日期 | 按1天划分： 2,500张 | 1/2,000 |
| 客户号 | 按客户均匀划分： 100张 | 1/5,0000 |
| 订单状态 | 状态为 ‘9’ 的比率假设为90%：<br>状态‘1’ - ‘4’ 500,000张<br>状态‘9’ 4,500, 000张 | <br>状态'1'-'4'占全部： 1/10<br>状态 ‘9’ 占全部： 9/10 |

# 如何编写高性能SQL – 基础知识

## 索引与数据检索

- 创建索引以进行有效率的数据查询
  - 数据量少时，即便没有索引，全表搜索也不成问题
  - 数据量大时，搜索性能问题（特别是全表搜索）才会产生



# 如何编写高性能SQL – 基础知识

## 数据检索中对索引的使用

- 查询中使用了条件列限制，但列相关索引似乎没起作用
  - 当条件的选择性较差，满足条件的数据比例较多时（如查询3年运营中其中1年的数据）

  一般查询量为30%或更多时，全表扫描往往比用索引更高效
  - 查询列DISTINCT唯一值较少的情况(如男女性别区分查询)

  特别对于数据仓库(DWH)处理，则有必要建立bitmap索引

**对于Where条件中的查询列，不要盲目地添加索引**

- 查询中未使用条件限制，即便存在索引
  - SQL语句查询不会使用此索引

**如何让SQL语句查询用到索引（开发者的责任）**

# 如何编写高性能SQL – 基础知识

## 通过索引进行数据检索

- 参考之前的例子创建索引

| | 需查询的列名 | 数据统计 | 查询率 |
|---|---|---|---|
| 索引 | 订单编号 | 按订单划分： 1张 | 1/5,000,000 |
| 索引 | 订购日期 | 按1天划分： 2,500张 | 1/2,000 |
| 索引 | 顾客号 | 按客户均匀划分： 100张 | 1/5,000 |
| | 订单状态 | 状态为 ‘9’ 的比率假设为90%：<br>状态 ‘1’ - ‘4’ 500,000张<br>状态 ‘9’ 4,500, 000张 | <br>状态'1'-'4'占全部： 1/10<br>状态 ‘9’ 占全部： 9/10 |

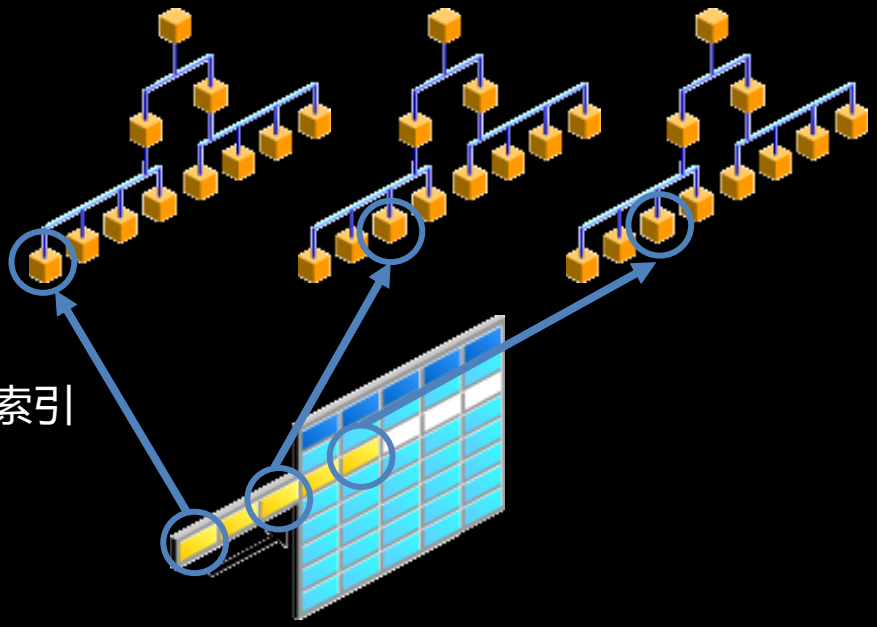
- 通过状态区分订单，若查询条件订单状态=‘9’，则更适用于全表扫描的情况。

# 如何编写高性能SQL – 基础知识

## 表索引数量问题

- 索引太多会导致更新表的速度变慢
  - 表的更新时，维护索引会有I/O负荷发生
  - 特别是批处理时，性能下降尤其明显

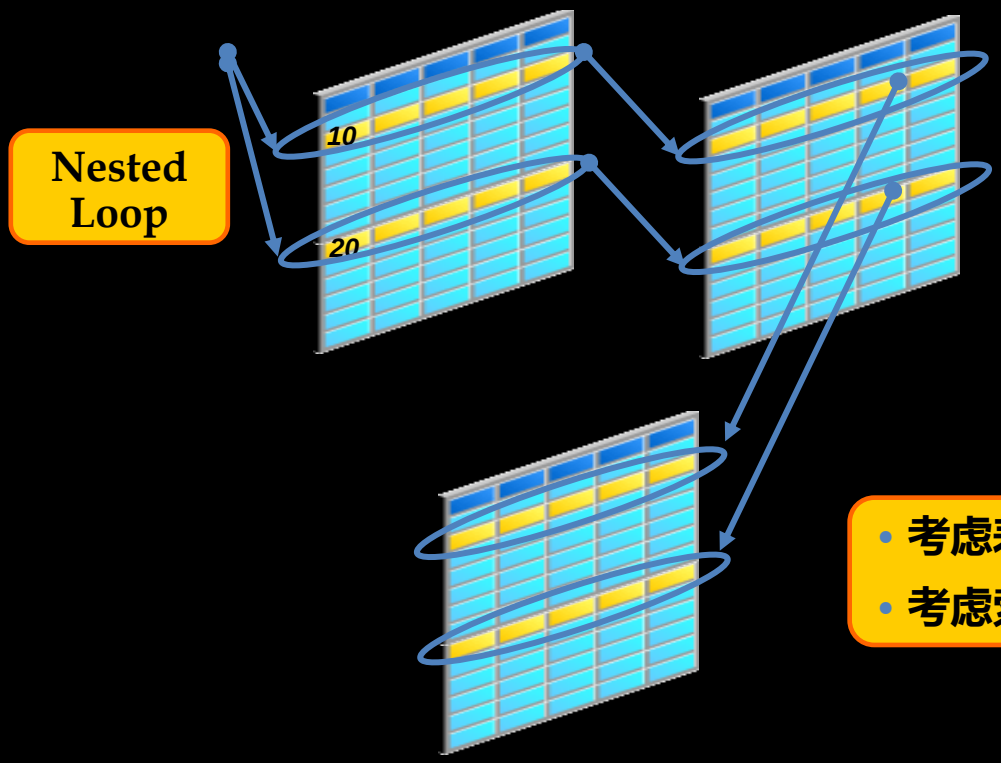
- Disk空间消耗量变大
  - 与表相比，索引中为使用的部分将更多
  - 如图，此表的INSERT操作需要维护3个索引



# 如何编写高性能SQL – 基础知识

## 表连接

- 多表连接，会进行内部排序循环处理




```
SELECT A.xx, B.yy, C.zz
  FROM A, B, C
 WHERE A.COL1 = B.COL1
   AND B.COL2 = C.COL2
   AND A.KEY in (10, 20);
```

- 考虑表扫描的先后顺序
- 考虑索引的有效性使用

# 如何编写高性能SQL – 基础知识

## 表连接（Nested Loop）

- 查询流程：
  - 通过优化器确定驱动表（外部表）
  - 进行以下的循环处理：
    - 提取驱动表中有效数据的一行（可以访问索引，也可以无索引）
    - 在其他表（内部表）查找匹配的有效数据并提取（访问索引）
  - 将数据返回到Oracle客户端

- 注意事项：
  - 被驱动表(内部表)如果没有索引的话，查询性能将很差

# 如何编写高性能SQL – 基础知识

表连接（Sort Merge）

- 查询流程：
  - 对表A排序
  - 对表B排序
  - 对排序后的表进行合并处理

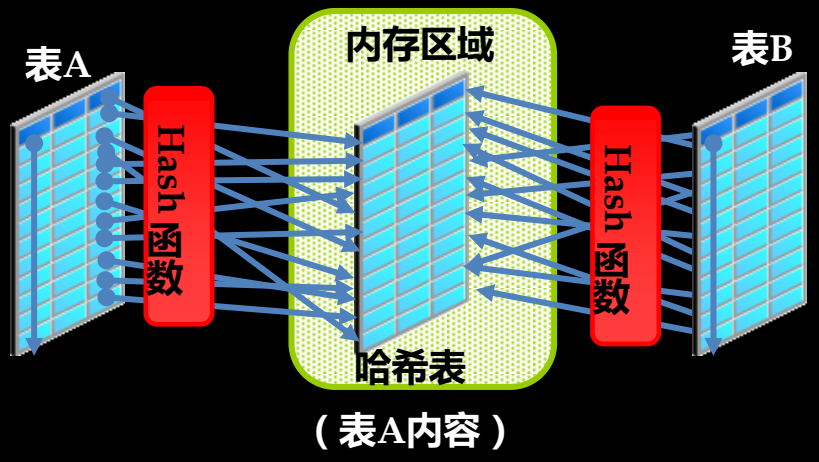

- 注意事项：
  - 大数据量的sort merge需要注意
  - OLTP场景下使用sort merge需要注意

# 如何编写高性能SQL – 基础知识

## 表连接（Hash Join）

- 查询流程：
  - 对数据量小的表进行全表读取
  - 在内存中创建一个对应的哈希表
  - 对大表进行读取并Hash（检查哈希表，找到匹配行哈希值后返回大表的对应行）

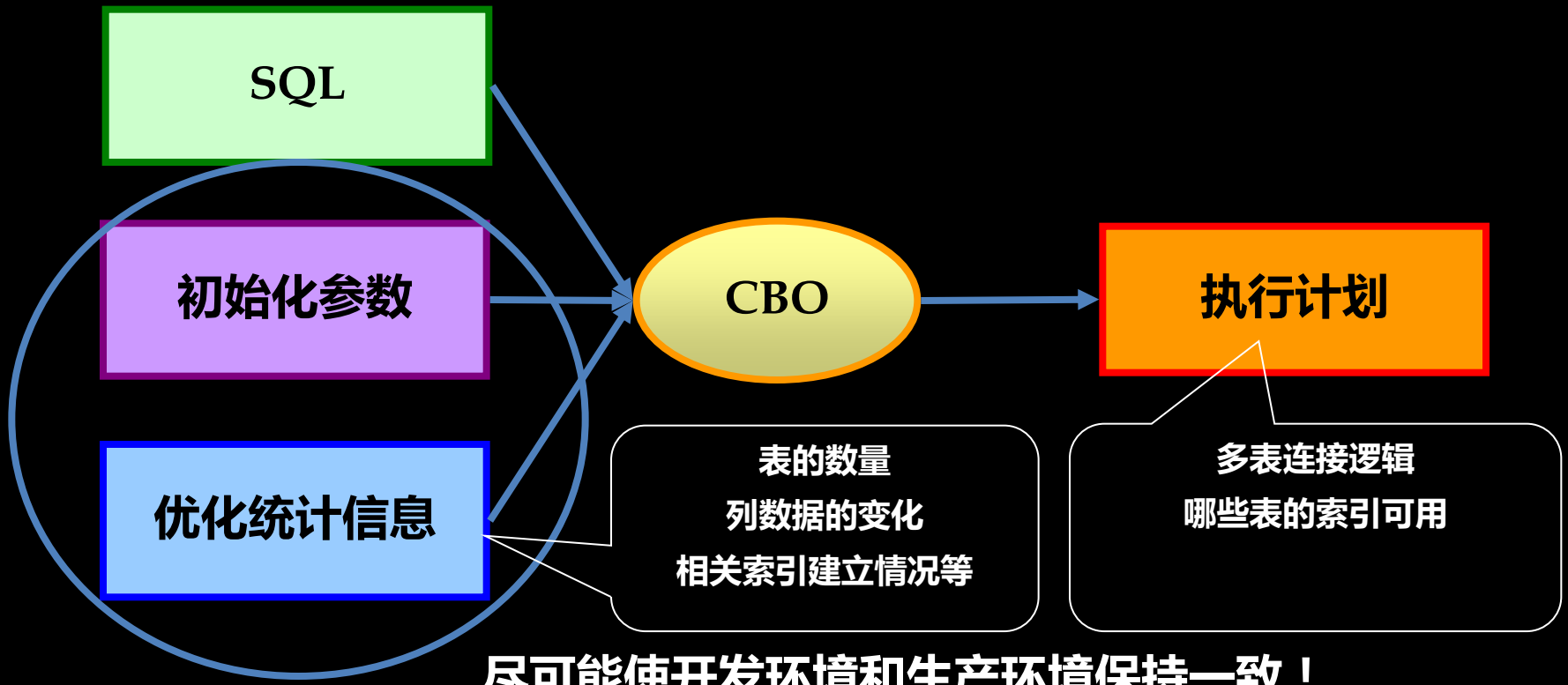
- 注意事项：
  - 当连接条件是非等价的键（范围指定）连接，则不推荐使用哈希联接。
  - OLTP场景下哈希连接需要注意。



# 如何编写高性能SQL – 执行计划

优化器的执行计划

- 通过基于成本优化器（CBO）的统计信息，以获得最优的执行计划




**尽可能使开发环境和生产环境保持一致！**

# 如何编写高性能SQL – 执行计划

在确认执行计划之前

- 将生产环境的优化统计信息导入到开发环境中
  - 请**不要收集**开发环境中的优化统计信息
- 优化器统计信息导入/导出

| | |
|---|---|
| **生产环境下统计信息导出** | **DBMS_STATS.EXPORT_*_STATS** |
| **开发环境下统计信息导入** | **DBMS_STATS.IMPORT_*_STATS** |

- 在开发环境下，关闭自动统计信息收集 (从10g开始会进行自动收集)

```
EXECUTE  DBMS_STATS.LOCK_TABLE_STATS ( 'SCOTT','EMP' ) ;
EXECUTE  DBMS_STATS.LOCK_SCHEMA_STATS ( 'SCOTT' ) ;
```

# 如何编写高性能SQL – 执行计划

## 如何获取执行计划（1）

- 通过命令行来获取
  - 运行以下脚本命令来建立PLAN_TABLE (10g之前，10后默认已经安装)
    - $ORACLE_HOME/rdbms/admin/utlxplan.sql

```
SQL> select owner, object_name, object_type from dba_objects where object_name like '%PLAN_TABLE%';

OWNER                          OBJECT_NAME                    OBJECT_TYPE
------------------------------ ------------------------------ -----------------------------
SYS                            SQL_PLAN_TABLE_TYPE            TYPE
SYS                            PLAN_TABLE$                    TABLE
PUBLIC                         SQL_PLAN_TABLE_TYPE            SYNONYM
PUBLIC                         PLAN_TABLE                     SYNONYM
EXFSYS                         EXF$PLAN_TABLE                 TABLE
```

  - 在SQL语句前加

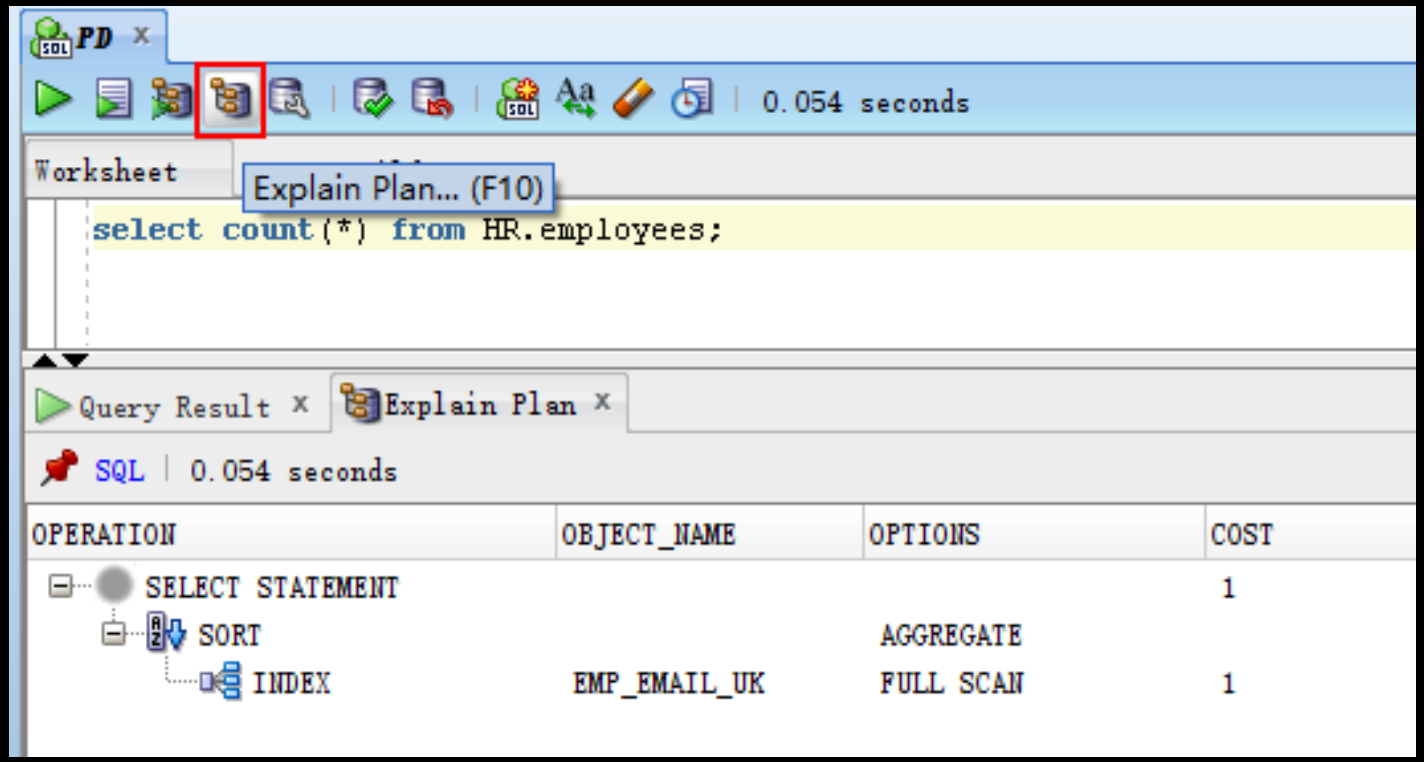
【explain plan for】并执行

```
explain plan for
SELECT d.dname,e.empno,e.ename
  FROM emp e, dept d
 WHERE e.deptno = d.deptno;
```

# 如何编写高性能SQL – 执行计划

## 如何获取执行计划（2）

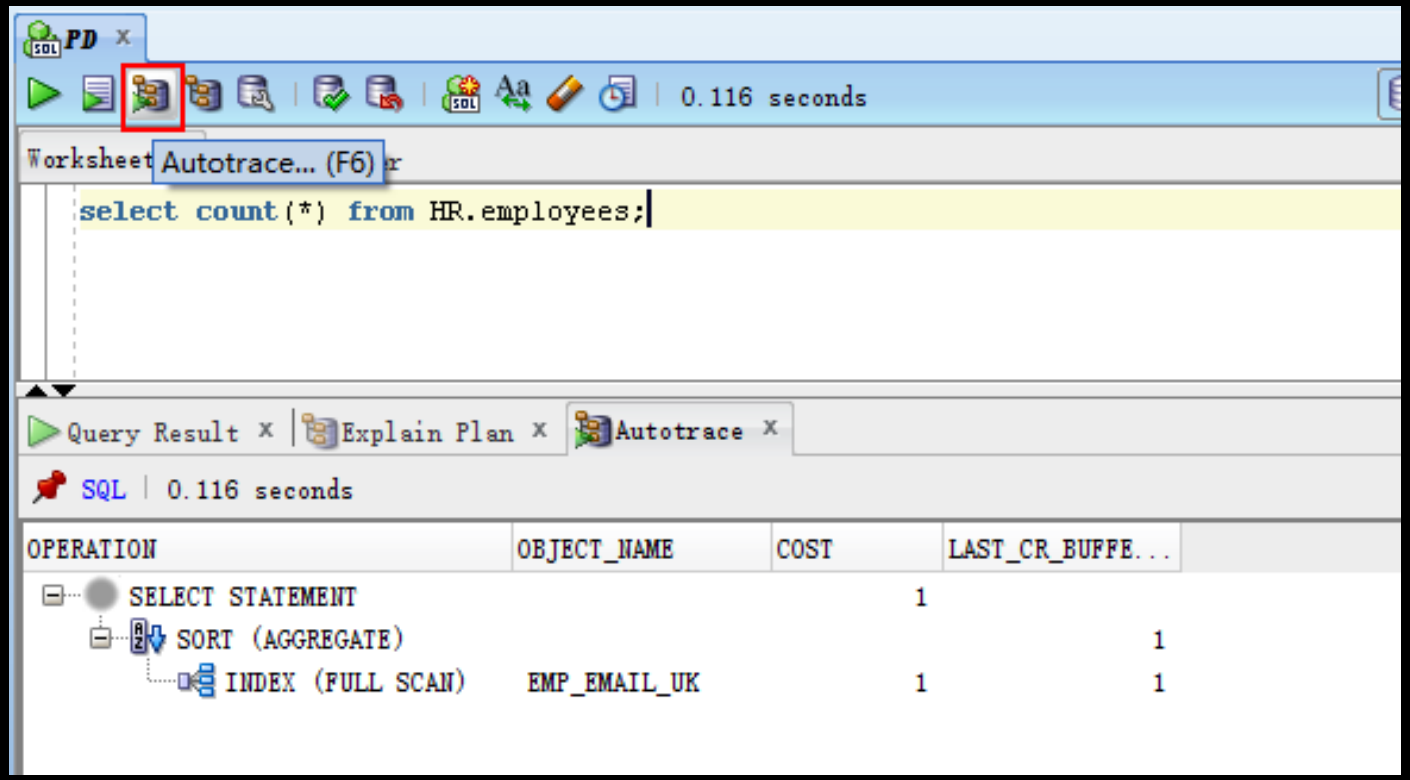
- 使用SQL Developer 【Explain Plan】按钮



# 如何编写高性能SQL – 执行计划

## 如何获取执行计划（3）

- 查看SQL Developer 【Autotrace】



# 如何编写高性能SQL – 执行计划

举例: Nested Loop

```
SELECT d.dname,e.empno,e.ename
  FROM emp e, dept d
 WHERE e.deptno = d.deptno
     AND e.empno between 7000 and 7500;
```

```
 Id   Operation                          Name
  0   SELECT STATEMENT
  1    NESTED LOOPS
* 2   ② TABLE ACCESS BY INDEX ROWID      EMP
* 3     ① INDEX RANGE SCAN               PK_EMP
  4   ④  TABLE ACCESS BY INDEX ROWID     DEPT
* 5     ③ INDEX UNIQUE SCAN              PK_DEPT

Predicate Information (identified by operation id):
   2 - filter("E"."DEPTNO" IS NOT NULL)
   3 - access("E"."EMPNO">=7000 AND "E"."EMPNO"<=7500)
   5 - access("E"."DEPTNO"="D"."DEPTNO")
```

在①~④ 反复循环执行

# 如何编写高性能SQL – 执行计划

举例: Merge Join

```
SELECT d.dname,e.empno,e.ename
  FROM emp e, dept d
 WHERE e.deptno = d.deptno;
```

```
 Id   Operation                          Name
  0   SELECT STATEMENT
⑤ 1    MERGE JOIN
  2   ② TABLE ACCESS BY INDEX ROWID      DEPT
  3     ①  INDEX FULL SCAN               PK_DEPT
* 4   ④ SORT JOIN
* 5     ③  TABLE ACCESS FULL             EMP

Predicate Information (identified by operation id):
   4 - access("E"."DEPTNO"="D"."DEPTNO")
       filter("E"."DEPTNO"="D"."DEPTNO")
   5 - filter("E"."DEPTNO" IS NOT NULL)
```

①、②在执行后、
③、④再执行
最后⑤进行合并处理

# 如何编写高性能SQL – 执行计划

举例: Hash Join

```
SELECT m.empno, m.ename, w.empno
  FROM employees m, employees_wk1 w
 WHERE m.ename=w.ename;
```

```
Id   Operation                    Name
  0  SELECT STATEMENT
* 1③ HASH JOIN
  2 ①  TABLE ACCESS FULL  EMPLOYEES_WK1
  3 ②  TABLE ACCESS FULL  EMPLOYEES

Predicate Information (identified by operation id):
  1 - access("M"."ENAME"="W"."ENAME")
```

表行数少

①对EMPLOYEES_WK1表做全表扫描并创建一个哈希表

②对EMPLOYEES表进行检索以找到哈希表对应匹配行

# 如何编写高性能SQL – 执行计划

举例: 多表连接

```
SELECT t1.c1
  FROM t1, t2, t3, t4, t5
 WHERE t1.c1 = t2.c1
    AND t2.c1 = t3.c1
    AND t3.c1 = t4.c1
    AND t4.c1 = t5.c1;
```

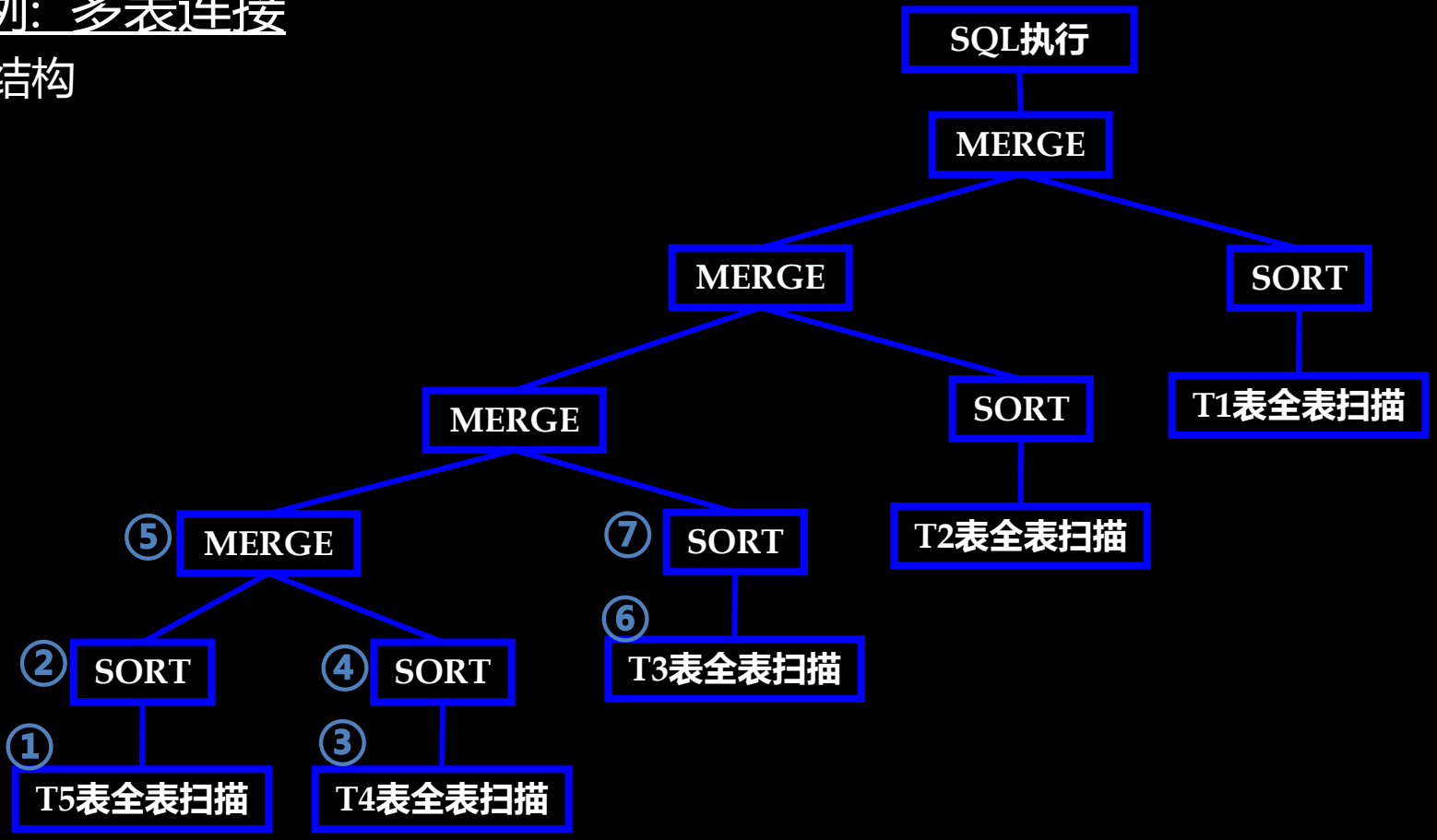
**执行计划**

```
----------------------------------------------------------
 0     SELECT STATEMENT
 1   0  MERGE JOIN
 2   1    MERGE JOIN
 3   2      MERGE JOIN
 4   3        MERGE JOIN
 5   4          SORT (JOIN)
 6   5            TABLE ACCESS (FULL) OF 'T5'
 7   4          SORT (JOIN)
 8   7            TABLE ACCESS (FULL) OF 'T4'
 9   3        SORT (JOIN)
10   9          TABLE ACCESS (FULL) OF 'T3'
11   2      SORT (JOIN)
12  11        TABLE ACCESS (FULL) OF 'T2'
13   1    SORT (JOIN)
14  13      TABLE ACCESS (FULL) OF 'T1'
```

# 如何编写高性能SQL – 执行计划

举例: 多表连接

- 结构



# 如何编写高性能SQL – 执行计划

举例: 数据过滤及外连接

```
select *
  from customer, order
 where order.cust_id(+) = customer.cust_id
   and order.cust_id    = '012345';
```

```
0    SELECT STATEMENT
1  0   FILTER
2  1    NESTED LOOPS (OUTER)
3  2     TABLE ACCESS (FULL) OF 'CUSTOMER'
4  2     TABLE ACCESS (BY INDEX ROWID) OF ''
5  4      INDEX (RANGE SCAN) OF 'IND_ORDER'(NON-UNIQUE)
```

外连接

过滤去除非匹配值

# 调整思路

应用变得反应很慢！数据库整体变慢？

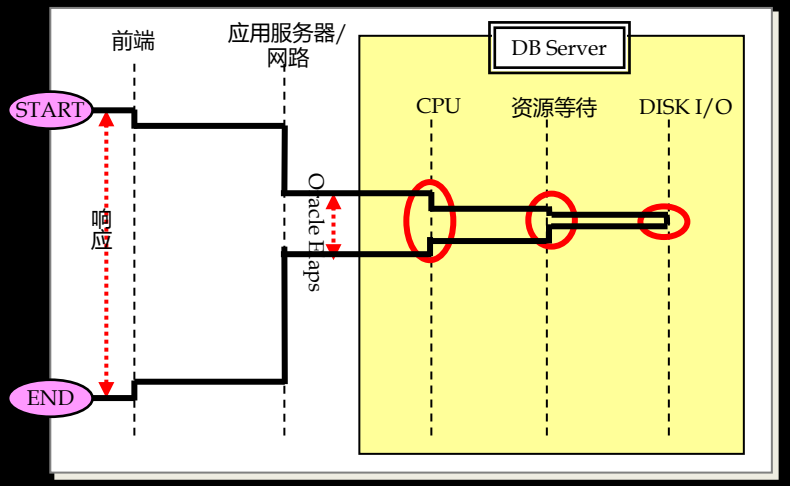
- 由于应用程序影响导致数据库整体变慢
  - 对应用程序进行调整

- Oracle数据库在变慢
  - 数据库调整（参数调整，扩大硬件内存等）
  - 修改应用程序 (绑定变量， 查询调优等)

# 调整思路

<u>应用变得反应很慢！是因为？</u>

- 非SQL原因
  - ➢ 需对应用逻辑进行审查

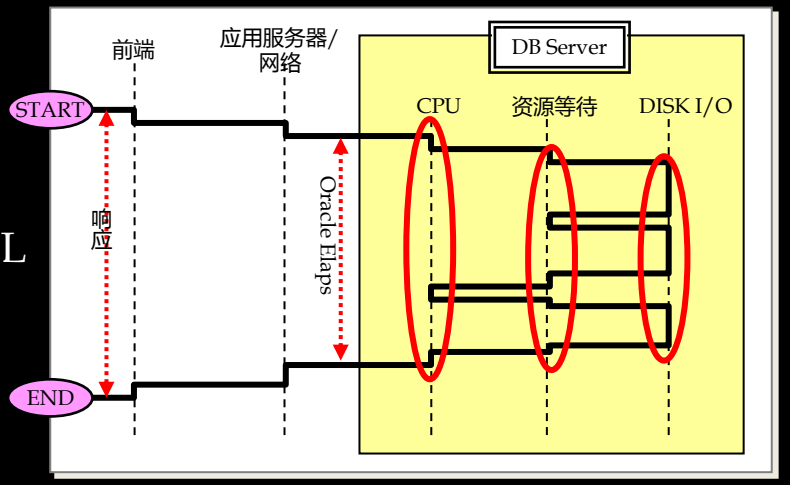



- SQL原因
  - ➢ 仅一两个SQL需要优化

(索引，增加查询条件限制)

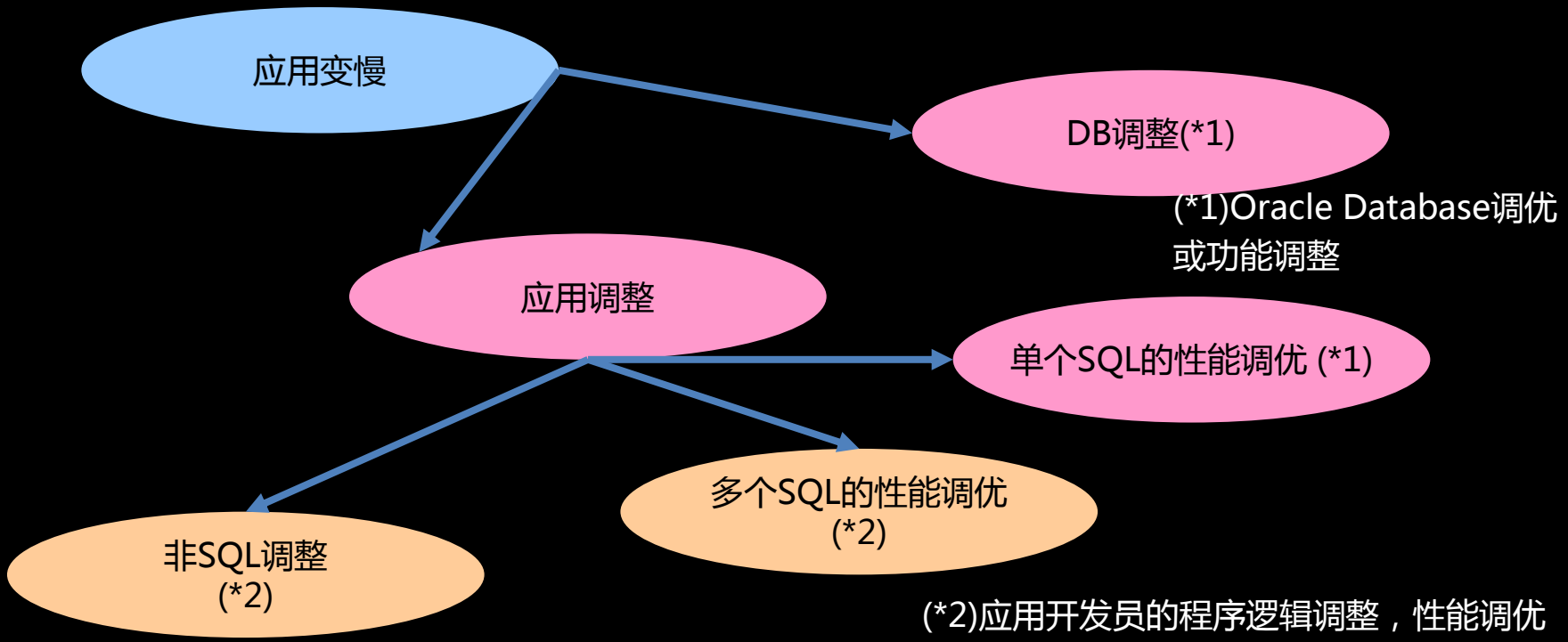
  - ➢ 非常多的SQL需要优化，如多个SQL的单一化SQL修改等

(则需要审查整个应用程序逻辑)



# 调整思路

## 应用调整


应用变慢
DB调整(*1)
(*1)Oracle Database调优
或功能调整
应用调整
单个SQL的性能调优 (*1)
多个SQL的性能调优
(*2)
非SQL调整
(*2)
(*2)应用开发员的程序逻辑调整，性能调优

# 总结

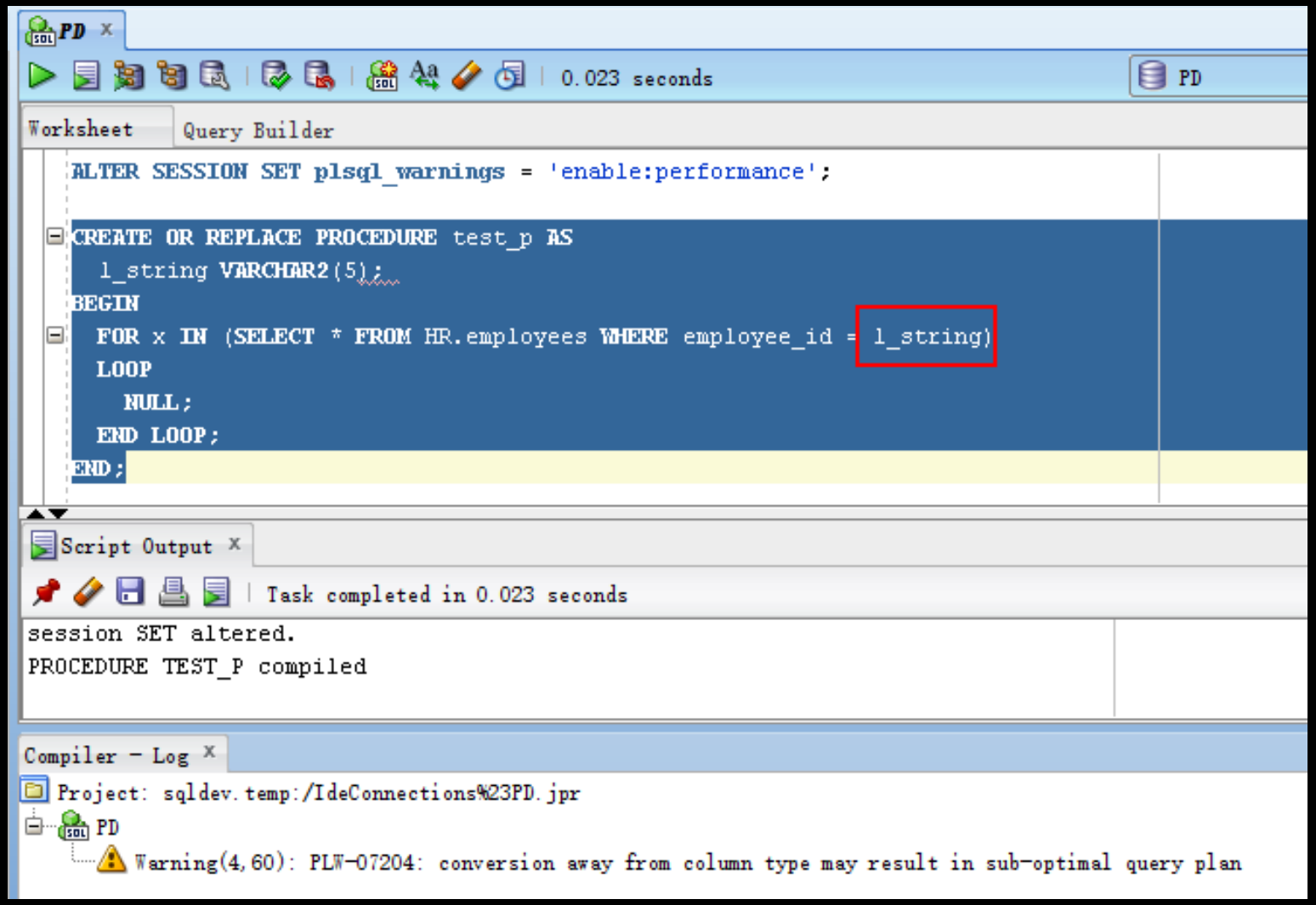
- 在应用程序开发中消除不必要的SQL处理。俯瞰整个应用程序，从而写出一个高效程序
- 应用程序开发人员应该关心实际的SQL操作
- 不仅仅将SQL作为一种语言，更要了解SQL在数据库的运作，从而实现有效的编码
- 更多得去了解表中的数据，清楚表的行数大小和查询条件。

# Tips - 1

plsql_warnings

- 尝试使用plsql_warnings参数来得到一些性能问题建议
  - Severe: 代码可能会产生的一些预料之外的错误及行为 (如 Exception语法中没有Raise exception)
  - Performance: 不当的条件编码可能会造成的一些性能问题
  - Informational: 非语法性错误，由写法不对造成的（如 Null is not NULL）

# Tips - 1


PD
0.023 seconds
PD
Worksheet
Query Builder
ALTER SESSION SET plsql_warnings = 'enable:performance';
CREATE OR REPLACE PROCEDURE test_p AS
l_string VARCHAR2(5);
BEGIN
FOR x IN (SELECT * FROM HR.employees WHERE employee_id = l_string)
LOOP
NULL;
END LOOP;
END;
Script Output
Task completed in 0.023 seconds
session SET altered.
PROCEDURE TEST_P compiled
Compiler - Log
Project: sqldev.temp:/IdeConnections%23PD.jpr
PD
Warning(4,60): PLW-07204: conversion away from column type may result in sub-optimal query plan

# Tips - 2

## DBMS_STATS

- 作为CBO基础，主动收集参数信息（对于新表和大量更新表）
  - DBMS_STATS 替代过去ANALYZE命令
  - estimate_percent : NULL (不抽样，全表统计) ，值 (按此作为抽样比例)
  - 可对TABLE / INDEX / SCHEMA 级别收集
  - 可将收集导入到其他数据库

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(ownname          => 'HR',
                                tabname          => 'employees',
                                estimate_percent => NULL,
                                cascade          => TRUE); -- gather index statics
END;
```

```
Script Output x
Task completed in 0.741 seconds
anonymous block completed
```

# Tips - 3

## 索引与NULL值

- 无论单列唯一索引或复合唯一索引，对于可为NULL的列或复合NULL值，Oracle不会为其存储索引值。
- 故在基于单列创建B树唯一索引或多列B树唯一索引的情况下：
- 当列上允许NULL值时
  - Where子句使用基于is null的情形，其执行计划使用全表扫描
  - Where子句使用基于is not null的情形，其执行计划使用索引扫描（索引范围扫描或索引全扫描）
- 当列上不允许为NULL值时，存在非NULL约束
  - Where子句使用基于is null的情形，其执行也走索引扫描
  - Where子句使用基于is not null的情形，其执行计划也走索引扫描。

# Tips - 4

## 动态SQL – SQL*Plus

- 用于重复SQL执行，通过降低硬解析提升性能
- SQL*Plus使用中的动态SQL

```
-- SQL*Plus中对以下每一行都需进行硬解析
SELECT * FROM geo_street WHERE street_id = 11;
SELECT * FROM geo_street WHERE street_id = 12;
SELECT * FROM geo_street WHERE street_id = 14;
```

```
-- 通过绑定变量使得能够重用执行计划
VARIABLE street_id NUMBER

EXEC :street_id := 11;
SELECT * FROM geo_street WHERE street_id = :street_id;

EXEC :street_id := 12;
SELECT * FROM geo_street WHERE street_id = :street_id;

EXEC :street_id := 14;
SELECT * FROM geo_street WHERE street_id = :street_id;
```

# Tips - 4

## 动态SQL – EXECUTE IMMEDIATE

- 用于重复SQL执行，通过降低硬解析提升性能
- 比DBMS_SQL更易用，且速度更快（ DBMS_SQL性能低5倍，避免使用DBMS_SQL ）

```
DECLARE
  sql_stmt  VARCHAR2(200);
  dept_id   NUMBER(2) := 50;
  dept_name VARCHAR2(20) := 'STATISTICS CANADA';
  dept_loc  VARCHAR2(13) := 'OTTAWA';
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE table_a (AID NUMBER(5), amount NUMBER)';
  --
  sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
  EXECUTE IMMEDIATE sql_stmt
    USING dept_id, dept_name, dept_loc;
END;
```

注意语句中不要使用分号

# Tips - 4

## JDBC开发改良1

- 改良前

```
conn = ds.getConnection(userid, password);

 for (i = 0; i < 1000000; i++) { {
   pstmt =
conn.prepareStatement("SELECT name FROM employees WHERE id = " + i);
   rset = pstmt.executeQuery();

   ......
}

conn.close();
```

每次执行都使用拼凑的SQL语句，未启用绑定变量，默认情况下每次均硬解析hard parse

# Tips - 4

## JDBC开发改良1

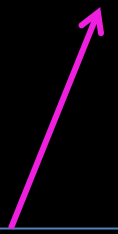
- 改良后

```
conn = ds.getConnection(userid, password);

 for (i = 0; i < 1000000; i++) { {
   pstmt =
     conn.prepareStatement("SELECT name FROM employees WHERE id = ?");
   pstmt.setInt(1, i);
   rset = pstmt.executeQuery();

   ......
}

conn.close();
```



使用绑定变量，避免每次执行都硬解析

# Tips - 4

## JDBC开发改良2

- 正确使用PrepareStatement

- PrepareStatement
  - SQL语句解析
- Bind
  - 变量绑定
- Execute
  - SQL语句执行

✕

```
For  {
     PrepareStatement
     Bind
     Execute
}
```

○

```
PrepareStatement
For {
     Bind
     Execute
}
```

# Tips - 4

## JDBC开发改良2

```
conn = ds.getConnection(userid, password);
pstmt =
  conn.prepareStatement("SELECT name FROM employees WHERE id = ?");

 for (i = 0; i < 1000000; i++) { {
   pstmt.setInt(1, i);
   rset = pstmt.executeQuery();

   ......
}

conn.close();
```

预解析，只解析一次

# Tips - 4

## JDBC开发改良2

- 最终效果
  - 响应时间: 1ms
  - 吞吐量: 25,000tps
  - 并发争用减少
  - CPU被充分使用

# ParnassusData™

www.parnassusdata.com

400-690-3643

# Thank You