数据处理算法的七十二变

公益讲座11：00准时开始，请大家先浏览云技术微信公众号技术文章。资料会在各群同步发布，已入群客户请勿重复入群！

20-21

数据库和云讲座群

甲骨文云技术公众号

B站专家系列课程

数据处理算法的七十二变
RWP 谈性能优化系列

甲骨文技术公益课 - 数据库专场

2023 年 6 月 9 日 11:00

线上直播

基于 Oracle 数据库
免费企业数据健康检查

- 及时了解数据库健康状况，发现并解决潜在问题
- 维护数据库系统良好状态，保护数据资产的安全
- 提升数据库性能、稳定性和安全性，降低业务风险

免费咨询热线：

400-699-8888

* 活动最终解释权归甲骨文公司所有

议程

- 数据处理技术
- 应用程序算法及demo
- 总结

小数据遇到大的乘数
每毫秒处理一行需要多长时间?

表	响应时间（秒）	时钟时间
千行	1	1 秒
百万行	1,000	17 分钟
十亿行	1,000,000	12 天
万亿行	1,000,000,000	32 年

数据处理技术

Row by row 逐行

一个线程/进程，一次处理一行数据

Arrays 数组

一个线程/进程，一次处理一组数据

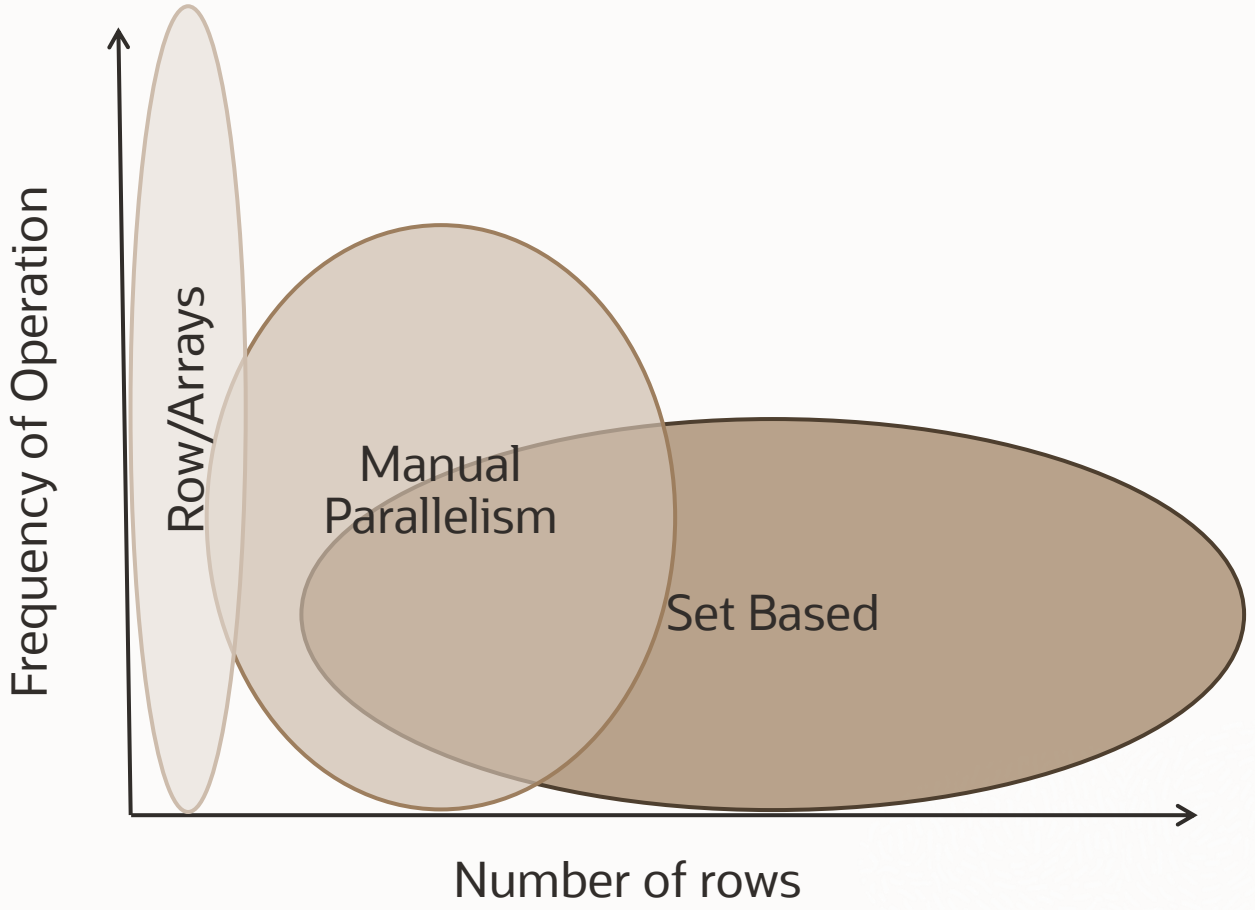
Manual Parallelism 手动并行

多个线程/进程各自执行逐行或数组处理

Set based processing 基于集合的处理

利用并行进程按组或行集处理数据

应用程序算法



议程

- 数据处理技术
- 应用程序算法及demo
- 总结

数据处理技术

Row by row 逐行

- 面向过程的编程
- 单个进程循环处理数据，每次处理一行
- 对于小的数据集非常适合
- 通常从数据库中检索数据，在中间层进行处理，然后在数据库中执行插入，更新 或删除操作
- 对于大型数据集，处理时间可能会很长
  - 以每行1毫秒的速度处理100万行需要1,000秒或17分钟

数据处理技术 - Row by row 例子

```
declare
  cursor c is select s.* from emp s;
  r c%rowtype;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
    if r.deptno = 20 then
      insert into west d values r;
    else
      insert into east d values r;
    end if;
    commit;
  end loop;
  close c;
end;
/
```


数据处理技术

Arrays 数组

- 把row by row的处理改成array的方式处理通常相当简单
- Array的方式允许数据库一次处理一组行，由此而带来的诸多好处
  - 减少网络往返的时间
  - 减少COMMIT的时间
  - 减少客户端和服务器中的code path
- 由于效率的提高，使得单个进程处理数据更快

Arrays Racing Demo

演示: Array的处理效率

数据处理技术

Manual Parallelism 手动并行

- 把row by row的程序并行化
- 多个线程/进程各自执行row by row或array方式的处理
- 通常由中间层或ETL服务器控制
- 实施起来很复杂，在多个线程上平均分配工作负载就很复杂
- 多个线程对数据库对象执行相同操作时可能出现争用问题

Array vs Manual Parallelism Demo

演示: 对比Array与手动并行

数据处理技术

Set based processing 基于集合的处理

- 按组或行集处理数据
- 使用SQL定义结果
- 以集合为单位，数据库可以高效的处理数据，得益于
  - 网络上没有数据移动
  - Hash joins
  - 并行查询和DML
  - 一体机平衡的CPU和IO能力可以得以充分发挥

数据处理技术 – set based processing 例子

```
insert /*+ append */ into west
select *
from   emp
where  deptno = 20;

commit;

insert /*+ append */ into east
select *
from   emp
where  deptno != 20;

commit;
```


数据处理技术 – set based processing 例子

```
insert /*+ append */ first
  when deptno = 20 then
    into west values …
  else
    into east values …
select *
from   emp;

commit;
```


数据处理技术 – set based processing

使用SQL定义结果
让数据库决定怎样去执行

- 一些set based的SQL操作
  - create table as select
  - insert /*+ append */ select
  - intersect
  - minus
  - exists
  - not exists
  - window functions
  - multi-table inserts
  - outer joins

Set Programming Demo: Loading

演示: Set-Based数据加载

示例代码 – 为什么选择PL/SQL

PL/SQL是将SQL提交到数据库的好方法

- 有许多功能可以管理SQL的执行
- 消除网络开销
- 优化提交时间

所有示例均使用PL/SQL
当示例代码运行缓慢时，所有时间都在SQL中

- 不够优化的，是应用的算法，而不是
  - PL / SQL
  - SQL

示例代码

示例代码演示了不同的技术

示例代码并不代表Best Practice!

- 没有异常处理
- 没有重新执行功能

使用逐行方法加载

```
declare
  cursor c is select s.* from ext_scan_events s;
  r c%rowtype;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
    insert into stage1_scan_events d values r;
    commit;
  end loop;
  close c;
end;
```


使用数组方法加载

```
declare
  cursor c is select * from ext_scan_events;
  type t is table of c%rowtype index by binary_integer;
  a t;
  rows binary_integer := 0;
begin
  open c;
  loop
    fetch c bulk collect into a limit array_size;
    exit when a.count = 0;
    forall i in 1..a.count
      insert into stage1_scan_events d values a(i);
    commit;
  end loop;
  close c;
end;
```


使用手动并行方法加载

```
declare
  sqlstmt varchar2(1024) := q'[
-- BEGIN embedded anonymous block
  cursor c is select
                s.*
               from
                ext_scan_events_${thr} s;
  type t is table of c%rowtype
    index by binary_integer;
  a t;
  rows binary_integer := 0;
begin
  for r in (select
              ext_file_name
             from
              ext_scan_events_dets
             where ora_hash(file_seq_nbr,${thrs}) = ${thr})
  loop
...
```

使用手动并行方法加载（续）

```
...
  loop
    execute immediate
      'alter table ext_scan_events_${thr} location ' ||
        '(' || r.ext_file_name || ')';
    open c;
    loop
      fetch c bulk collect into a limit ${array_size};
      exit when a.count = 0;
      forall i in 1..a.count
        insert into stage1_scan_events d values a(i);
      commit;
    end loop;
    close c;
  end loop;
end;
-- END   embedded anonymous block
]';
```

使用手动并行方法加载（续）

```
...
begin
  sqlstmt := replace(sqlstmt,'${array_size}'
                             ,to_char(array_size));
  sqlstmt := replace(sqlstmt,'${thr}',thr);
  sqlstmt := replace(sqlstmt,'${thrs}',thrs);
  execute immediate sqlstmt;
end;
```


使用set based方法加载

```
alter session enable parallel dml;

insert /*+ APPEND */ into
 stage1_scan_events d
select
 s.*
from
 ext_scan_events s;

commit;
```


Set Programming Demo: De-duplicating

演示: Set-Based去重

使用逐行方法去重

```
declare
  cursor c is select
               s.*
              ,cast(null as number(10)) as error_ind
              ,rowid as rid
              from
               stage1_scan_events_ref s;
  r c%rowtype;
  dups number;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
...
```


使用逐行方法去重（续）

```
...
    select
     count(*)
    into
     dups
    from
     stage1_scan_events_ref s
    where s.loc_code                = r.loc_code
      and s.rtl_trx_seq_nbr         = r.rtl_trx_seq_nbr
      and s.trx_line_item_seq_nbr = r.trx_line_item_seq_nbr
      and ( s.file_seq_nbr          > r.file_seq_nbr
        or ( s.file_seq_nbr         = r.file_seq_nbr
         and s.rowid                < r.rid
           )
         );
...
```


使用逐行方法去重（续）

```
...
    if dups = 0
    then
      insert into stage2_scan_events d values (r ...);
    else
      r.error_ind := 1;
      insert into stage1_scan_events_err d values r;
    end if;
    commit;
  end loop;
  close c;
end;
```


使用数组方法去重

```
  loop
    fetch c bulk collect into a limit array_size;
    exit when a.count = 0;
    a_count := a.count;
    for i in 1..a_count
    loop
      select
...
      if dups = 0
      then
        null;
      else
        a(i).error_ind := 1;
        insert into stage1_scan_events_err d values a(i);
        a.delete(i);
      end if;
    end loop;
    forall i in indices of a
      insert into stage2_scan_events d values (a(i) ... );
    commit;
  end loop;
```

使用手动并行方法去重

```
cursor c is select
             s.*
            ,cast(null as number(10)) as error_ind
            ,rowid as rid
            from
             stage1_scan_events_ref s
            where ora_hash(loc_code,threads) = thread;
```

使用set based方法去重

```
alter session enable parallel dml;

insert /*+ APPEND */ first
when error_ind = 1 then into
 stage2_scan_events
values ( ... )
else into
 stage1_scan_events_err
select
 s.*
,row_number() over (
partition by
 loc_code,rtl_trx_seq_nbr,trx_line_item_seq_nbr
order by
 file_seq_nbr desc,rowid
) as error_ind
,rowid as rid
from
 stage1_scan_events_ref s;

commit;
```


Set Programming Demo: Transformation

演示: Set-Based数据转换

使用逐行方法转换

```
declare
  cursor c is select
               s.*
              ,cast(null as number(10)) as error_ind
              from
               stage2_scan_events_ref s;
  r c%rowtype;
  dk dim_day.day_key%type;
  lk dim_loc.loc_key%type;
  pk dim_prod.prod_key%type;
  day_code dim_day.day_code%type := '20130922';
  day_key dim_day.day_key%type :=
    sd_convert.day_code_to_key(day_code);
  error_ind number(10);
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
...
```

使用逐行方法转换（续）

```
...
    error_ind := 0;
    if r.day_code != day_code
    then
      dk := null;
    else
      dk := day_key;
    end if;
    if dk is null then error_ind := error_ind + 1; end if;
    lk := sd_convert.loc_code_to_key(r.loc_code);
    if lk is null then error_ind := error_ind + 2; end if;
    pk := sd_convert.prod_code_to_key(r.prod_code);
    if pk is null then error_ind := error_ind + 4; end if;
    if error_ind = 0
    then
      insert into stage3_scan_events d values (r ... );
    else
      r.error_ind := error_ind;
      insert into stage2_scan_events_err d values r;
    end if;
    commit;
  end loop;
  close c;
end;
...
```

使用数组方法转换

```
...
      if dk is null then error_ind := error_ind + 1; end if;
      lk := sd_convert.loc_code_to_key_rc(a(i).loc_code);
      if lk is null then error_ind := error_ind + 2; end if;
      pk := sd_convert.prod_code_to_key_rc(a(i).prod_code);
      if pk is null then error_ind := error_ind + 4; end if;
...
```


使用手动并行方法转换

```
...
  cursor c is select
                s.*
               ,cast(null as number(10)) as error_ind
               ,cast(null as number(10)) as day_key
               ,cast(null as number(10)) as loc_key
               ,cast(null as number(10)) as prod_key
               from
                stage2_scan_events_ref s
               where ora_hash(loc_code,threads) = thread;
...
```


使用set based方法转换

```
alter session enable parallel dml;

insert /*+ APPEND */ first
when error_ind = 0 then into
 stage3_scan_events
values ( ... )
else into
 stage2_scan_events_err
values ( ... )
with
 dim_day_current
as (select * from dim_day where day_code = '20130922')
select
 s.*
,d.day_key
,l.loc_key
,p.prod_key
...
```


使用set based方法转换（续）

```
...
,(
  case when day_key is not null  then 0 else 1 end
+ case when loc_key is not null  then 0 else 2 end
+ case when prod_key is not null then 0 else 4 end
 ) as error_ind
from
 stage2_scan_events_ref s
left outer join
 dim_day_current d
on s.day_code = d.day_code
left outer join
 dim_loc l
on s.loc_code = l.loc_code
left outer join
 dim_prod p
on s.prod_code = p.prod_code;

commit;
```


Set Programming Demo: Aggregation

演示： Set-Based汇总

使用逐行方法汇总

```
declare
  cursor c is select /*+ INDEX(s) */
                s.*
              from
               stage3_scan_events_ref s
              order by
               day_key
              ,loc_key
              ,prod_key;
  r_this c%rowtype;
  r_last c%rowtype;
  r_running_total fact_sales%rowtype;
...
```

使用逐行方法汇总（续）

```
...
  procedure update_running_total
  (
   r c%rowtype
  )
  is
  begin
    if r.actn_code = 'Sale'
    then
      r_running_total.qty :=
        r_running_total.qty + r.qty;
      r_running_total.extended_amt :=
        r_running_total.extended_amt + r.extended_amt;
      r_running_total.discount_amt :=
        r_running_total.discount_amt +
        r.extended_amt * (100 - r.discount_pct) / 100;
    else
      r_running_total.qty :=
        r_running_total.qty - r.qty;
      r_running_total.extended_amt :=
        r_running_total.extended_amt - r.extended_amt;
      r_running_total.discount_amt :=
        r_running_total.discount_amt -
        r.extended_amt * (100 - r.discount_pct) / 100;
    end if;
  end;
...
```


使用逐行方法汇总（续）

```
...
  procedure start_running_total
  (
   r c%rowtype
  )
  is
  begin
    r_running_total.day_key      := r.day_key;
    r_running_total.loc_key      := r.loc_key;
    r_running_total.prod_key     := r.prod_key;
    r_running_total.qty          := 0;
    r_running_total.extended_amt := 0;
    r_running_total.discount_amt := 0;
    update_running_total(r);
  end;
  procedure flush_running_total
  (
   r c%rowtype
  )
  is
  begin
    insert into fact_sales values r_running_total;
    commit;
    start_running_total(r);
  end;
...
```

使用逐行方法汇总（续）

```
...
begin
  open c;
  fetch c into r_last;
  if not c%notfound
  then
    start_running_total(r_last);
    loop
      fetch c into r_this;
      exit when c%notfound;
      if     r_this.day_key  = r_last.day_key
         and r_this.loc_key  = r_last.loc_key
         and r_this.prod_key = r_last.prod_key
      then
        update_running_total(r_this);
      else
        flush_running_total(r_this);
        r_last := r_this;
      end if;
    end loop;
    flush_running_total(r_last);
  end if;
  close c;
end;
```

使用数组方法汇总

```
...
  procedure flush_running_total
  is
  begin
    forall i in 1..a_running_totals.count
      insert into fact_sales values a_running_totals(i);
    commit;
    a_running_totals.delete;
  end;
  procedure buffer_running_total
  (
   r c%rowtype
  )
  is
  begin
    a_running_totals(a_running_totals.count + 1) :=
      r_running_total;
    if a_running_totals.count = array_size
    then
      flush_running_total;
    end if;
    start_running_total(r);
  end;
...
```

使用数组方法汇总（续）

```
...
    loop
      fetch c bulk collect into a limit array_size;
      exit when a.count = 0;
      for i in 1..a.count
      loop
        r_this := a(i);
        if     r_this.day_key  = r_last.day_key
           and r_this.loc_key  = r_last.loc_key
           and r_this.prod_key = r_last.prod_key
        then
          update_running_total(r_this);
        else
          buffer_running_total(r_this);
          r_last := r_this;
        end if;
      end loop;
    end loop;
    buffer_running_total(r_this);
    flush_running_total;
...
```

使用手动并行方法汇总

```
alter session enable parallel dml;

insert /*+ APPEND */ into
 fact_sales d
select
 *
from
 table(sd_aggregation.sum_scan_events
         (
         cursor(select * from stage3_scan_events_ref s)
         )
       );

commit;
```


使用手动并行方法汇总（续）

```
create or replace package body
 sd_aggregation
as
function sum_scan_events
(
 c refc
)
return t
pipelined
cluster c by (day_key,loc_key,prod_key)
parallel_enable
(
partition c by hash (day_key,loc_key,prod_key)
)
is
  r fact_sales%rowtype;
begin
...
```


使用手动并行方法汇总（续）

```
...
    loop
      fetch c bulk collect into a limit array_size;
      exit when a.count = 0;
      for i in 1..a.count
      loop
        r_this := a(i);
        if     r_this.day_key  = r_last.day_key
           and r_this.loc_key  = r_last.loc_key
           and r_this.prod_key = r_last.prod_key
        then
          update_running_total(r_this);
        else
          pipe row (r_running_total);
          start_running_total(r_this);
          r_last := r_this;
        end if;
      end loop;
    end loop;
    pipe row (r_running_total);
...
```


使用set based方法汇总

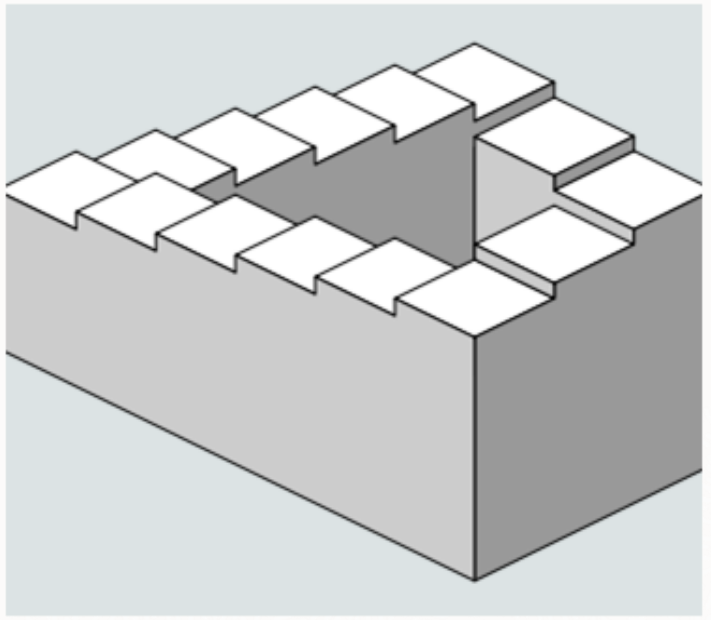
```
alter session enable parallel dml;

insert /*+ APPEND */ into
 fact_sales d
select
 day_key
,loc_key
,prod_key
,sum(case when actn_code = 'Sale' then 1 else -1 end *
     qty) as qty
,sum(case when actn_code = 'Sale' then 1 else -1 end *
     extended_amt) as extended_amt
,sum(case when actn_code = 'Sale' then 1 else -1 end *
     extended_amt * (100 - discount_pct) / 100) as discount_amt
from
 stage3_scan_events_ref s
group by
 day_key
,loc_key
,prod_key;

commit;
```

数据库性能不好的根本原因

处理大量数据的时候，

使用逐行或者数组的方法

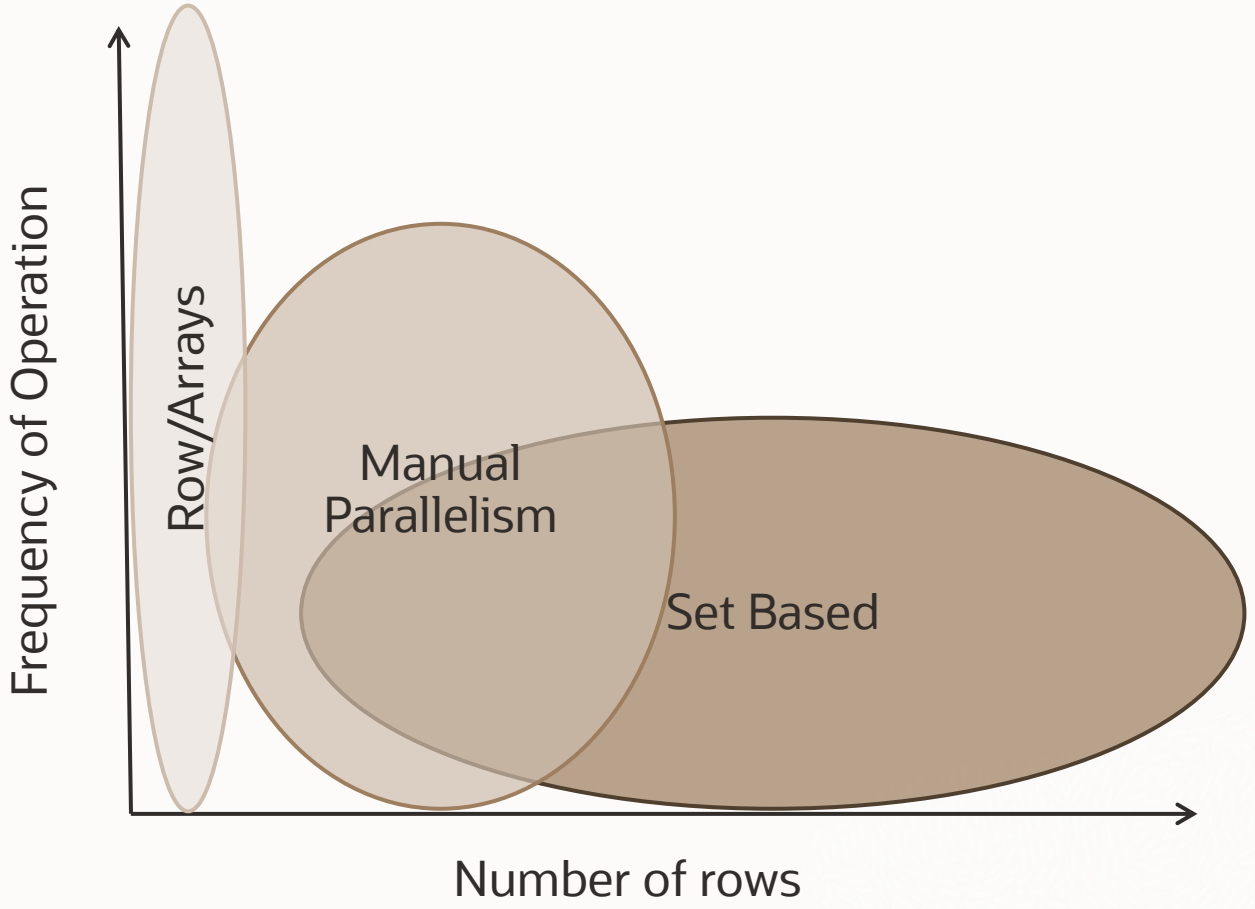
手动并行的方法



议程

- 数据处理技术
- 应用程序算法及demo
- 总结

应用程序算法



总结 – 处理少量数据

Row by row 逐行

- 一个线程/进程，一次处理一行数据
- 适合用OLTP场景中很少量数据的处理

Arrays 数组

- 一个线程/进程，一次处理一组数据
- 比row by row方法效率高
- 在OLTP场景中，能用arrays方法就用arrays方法

总结 –处理大量数据

Manual Parallelism 手动并行

- 实施起来很复杂
- 在多个线程上平均分配工作负载是个很大的挑战
- 多个线程对数据库对象执行相同操作时可能出现争用问题

Set based processing 基于集合的处理

- 使用SQL定义结果
- 在数据库里面处理数据，简单高效
- 在处理大量数据的时候，能用set based的方法就用set based的方法

基于 Oracle 数据库
免费企业数据健康检查

- 及时了解数据库健康状况，发现并解决潜在问题
- 维护数据库系统良好状态，保护数据资产的安全
- 提升数据库性能、稳定性和安全性，降低业务风险

免费咨询热线：

400-699-8888

* 活动最终解释权归甲骨文公司所有

数据库和云讲座群

20-21

甲骨文云技术公众号

技术专家1V1深入交流


ORACLE