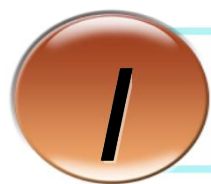


ORACLE数据库知识分享

—SQL优化

东方龙马数据库工程师：张红记



SQL语句基础概念



SQL语句为什么会慢



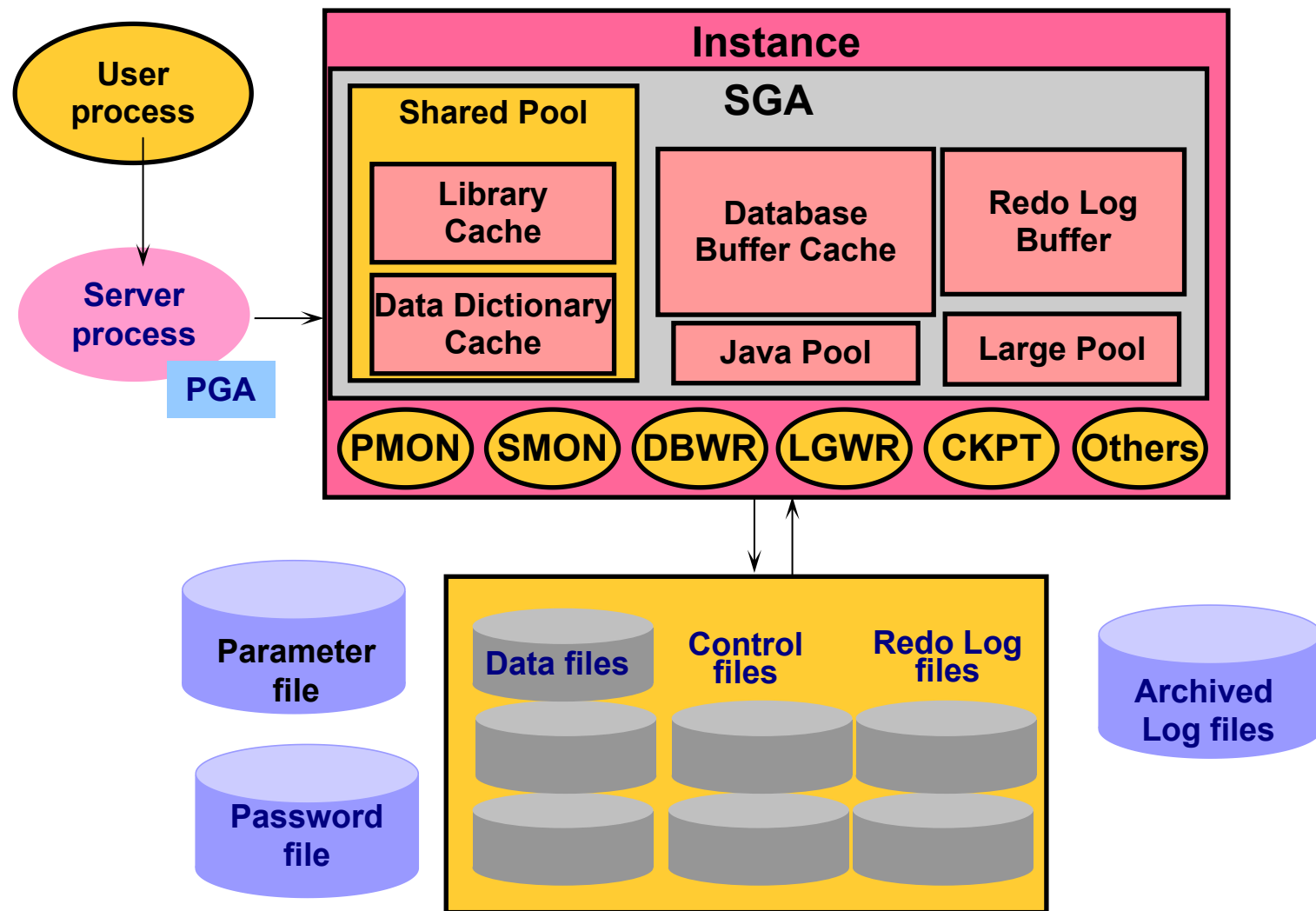
SQL语句怎么写会快



低效率语句对数据库的影响

ORACLE经典结构图

ORACLE数据库服务器，由ORACLE数据库和ORACLE数据库实例组成。



Library cache

Library cache是Shared pool的一部分，它几乎是Oracle内存结构中最复杂的一部分，主要存放shared cursors (SQL) 和PLSQL对象 (function, procedure, trigger) 的信息，以及这些对象所依赖的table, index, view等对象的信息。

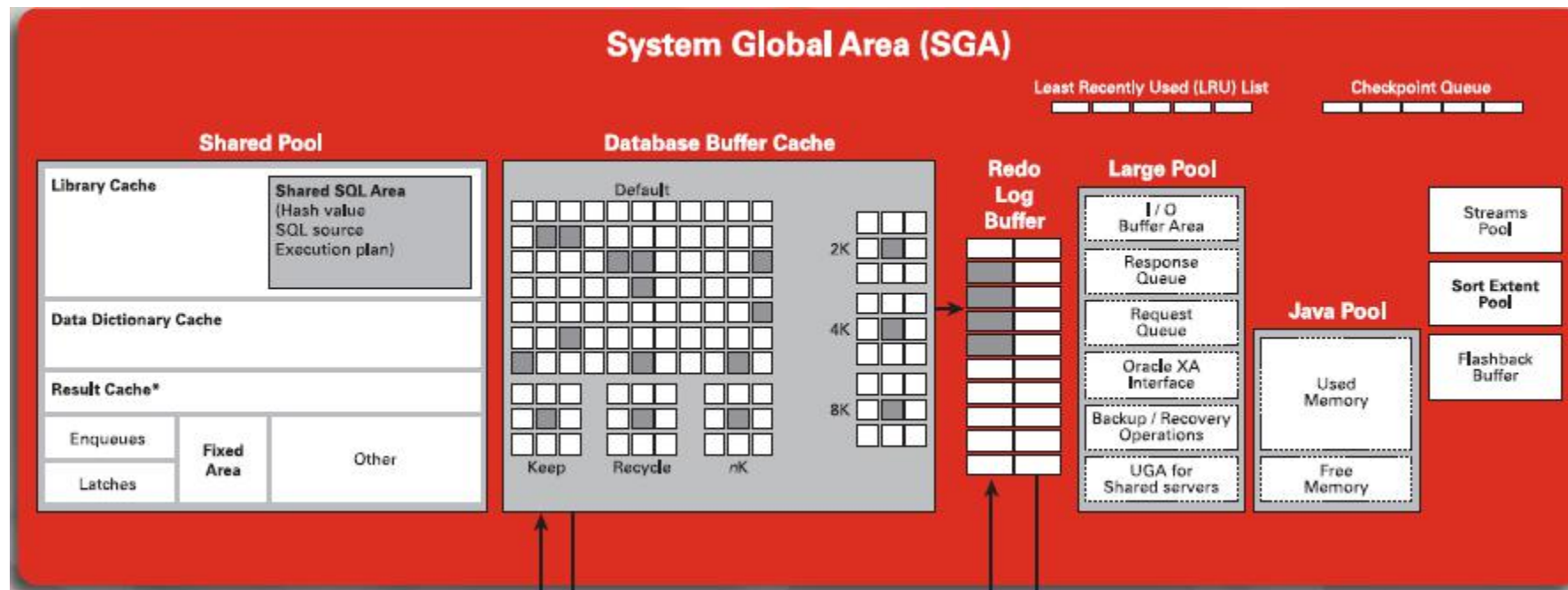
Library cache需要解决三个问题：

- 1.快速定位的问题
- 2.关系依赖的问题
- 3.并发控制的问题

Database buffer cache

由磁盘上的物理文件组成，不管在运行状态还是停止状态，这些文件一直存在。一旦创建数据库，数据库将永久存在。

SGA结构

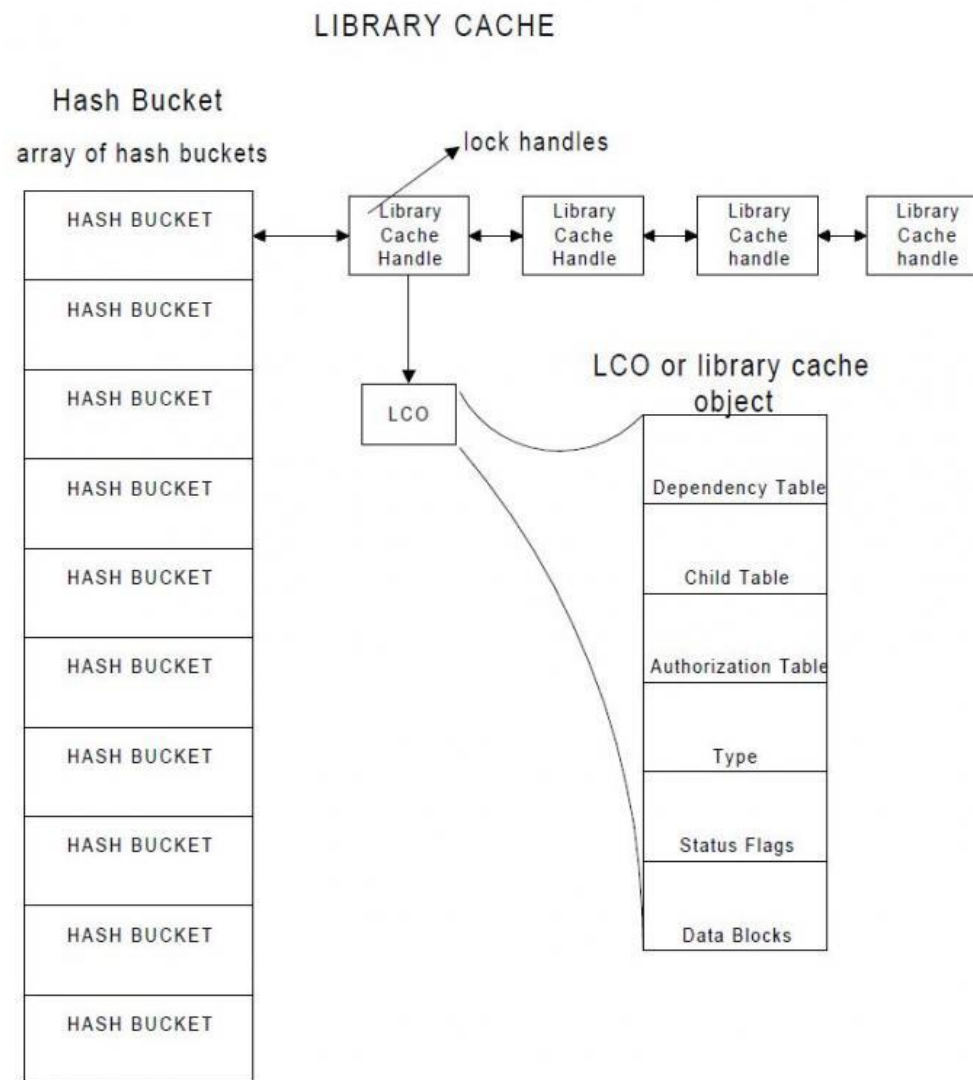


ORACLE LIBRARY CACHE结构

Library cache

- 1.快速定位的问题: Library cache中对象众多, Oracle如何管理这些对象, 以便服务进程可以迅速找到他们需要的信息。比如某个服务进程需要迅速定位某个SQL是否存在于Library cache中。
- 2.关系依赖的问题: Library cache中的对象存在复杂的依赖关系, 当某个object 失效时, 可以迅速将依赖其的对象也置为失效状态。比如某个表发生了结构变化, 依赖其的SQL语句需要重新解析。
- 3.并发控制的问题: Library cache中必须有一个并发控制的机构, 比如锁机制, 来管理大量共享对象的并发访问和修改的问题, 比如某个SQL在重新编译的同时, 其所依赖的对象不能被修改。

原理: 将对象信息 (比如SQL) hash定位到某个hash bucket中, 然后顺序扫描bucket中的 List, 实现快速定位对象的目的。



常用的查询语句

总数: `select count(*) totalcount from hr.employees;`

求和: `select sum(salary) sumvalue from hr.employees a where a.salary>2000;`

平均: `select a.department_id,avg(salary) avgvalue
from hr.employees a
where a.salary>2000
group by a.department_id ;`

最大: `select a.department_id,max(salary) maxvalue
from hr.employees a
group by a.department_id
order by a.department_id;`

最小: `select b.department_name, min(salary) minvalue
from hr.employees a, hr.departments b
where a.department_id = b.department_id
group by b.department_name
order by 2;`

```
select b.department_name, min(salary) minvalue
from hr.employees a, hr.departments b
where a.department_id = b.department_id
group by b.department_name
order by 2;
```

	DEPARTMENT_NAME	MINVALUE
1	Shipping	2100
2	Purchasing	2500
3	IT	4200
4	Administration	4400
5	Marketing	6000
6	Sales	6100
7	Human Resources	6500
8	Finance	6900
9	Accounting	8300
10	Public Relations	10000
11	Executive	17000

常用的查询语句

BETWEEN 的使用

between限制查询数据范围时包括了边界值，not between不包括
select * from hr.employees where salary between 12000 and 24000;
select * from hr.employees where salary not between 7000 and 9000;

select * from hr.employees where salary >= 12000 and salary <= 24000;
select * from hr.employees where salary < 7000 or salary > 9000;

IN 的使用


select * from hr.employees a where a.department_id in (80,90,100);
select * from hr.employees a where a.department_id not in (80,90,100);

EXISTS 的使用

select a.employee_id, a.first_name, a.department_id from hr.employees a
where not exists (select * from hr.departments b
 where b.department_id = a.department_id);

select a.employee_id, a.first_name, a.department_id from hr.employees a
where a.department_id not in
 (select b.department_id from hr.departments b)

```
select a.employee_id, a.first_name, a.department_id
from hr.employees a
where not exists (select *
                  from hr.departments b
                  where b.department_id = a.department_id);
```



	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
▶ 1	178	Kimberely	...

```
select a.employee_id, a.first_name, a.department_id
from hr.employees a
where a.department_id not in
     (select b.department_id from hr.departments b)
```



	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
--	-------------	------------	---------------

表关联查询

内连接

内连接，只连接匹配的行

```
Select a.c1,b.c2 from a join b on a.c3 = b.c3;
```

```
Select a.c1,b.c2 from a ,b where a.c3 = b.c3;
```

左外连接

包含左边表的全部行以及右边表中全部匹配的行

```
Select a.c1,b.c2 from a left join b on a.c3 = b.c3;
```

```
Select a.c1,b.c2 from a ,b where a.c3 = b.c3(+);
```

右外连接

包含右边表的全部行以及左边表中全部匹配的行

```
select a.c1,b.c2 from a right join b on a.c3 = b.c3;
```

```
Select a.c1,b.c2 from a ,b where a.c3(+) = b.c3;
```

内连接,外连接区别:

内连接是保证两个表中所有的行都要满足连接条件，而外连接则不然。

在外连接中，某些不满条件的列也会显示出来，也就是说，只限制其中一个表的行，而不限制另一个表的行。

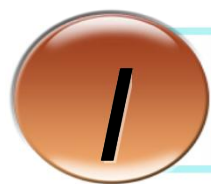
关联子查询

显示每个部门工资最高的员工信息

```
select a.*  
  from hr.employees a,  
       (select department_id, max(salary)  
        from hr.employees  
        group by department_id) b  
 where a.department_id = b.department_id  
       and a.salary = b.salary;
```

```
select b.department_name, count(*)  
  from hr.employees a, hr.departments b  
 where a.department_id = b.department_id(+)  
 group by b.department_name  
 order by 1;
```

	DEPARTMENT_NAME	COUNT(*)
1	Accounting	2
2	Administration	1
3	Executive	3
4	Finance	6
5	Human Resources	1
6	IT	5
7	Marketing	2
8	Public Relations	1
9	Purchasing	6
10	Sales	34
11	Shipping	45
12		1



SQL语句基础概念



SQL语句为什么会慢



SQL语句怎么写会快



低效率语句对数据库的影响

SQL语句为什么会慢

就当前已经在系统上运行的SQL语句，语句运行的快慢取决于该SQL所需要的资源是否被及时的获取到并被有效合理使用。

本身业务数据多

由业务原因要求SQL语句返回数据结果较多导致的语句执行缓慢，如业务批量处理，内存数据库初始化等。

SQL语句效率差

SQL语句执行过程中无效的数据扫描或不必要的计算导致资源浪费。



业务扫描次数多

SQL本身对资源消耗不大，但是执行频率非常高，如每小时执行百万次以上。

主机或者数据库资源紧张

SQL执行过程中主机资源是否繁忙，数据库参数设置是否合理，是否有锁存在。

SQL语句效率差

案例：某平台数据库主机CPU使用率达到99.6%，大量相关应用进程超时，业务页面无法正常运行，增加索引后数据库CPU使用率下降至10%左右，业务恢复正常。

检查数据库连接等待，发现数据库中存在大量热块读取（latch:cache buffers chains），即大量应用进程在读取相同的数据块。查看对应语句执行计划，确认对应语句大部分为全表扫描。因此可以基本确认，热块读的原因并不是因为用户对某一块进行频繁读取，而是由于每次扫描全表读取了大量无用数据，SQL语句返回结果命中率差，从而导致严重资源争用。

语句一：

```
select count(*)
  from norn_ggk_geter
 where jdate between to_date(to_char(sysdate, 'yyyy-MM-dd'), 'YYYY-mm-dd')
              and to_date(to_char(sysdate + 1, 'yyyy-MM-dd'), 'YYYY-mm-dd')
              and hid = '8a5c83904e7213d1014e7236d16a0010'
              and platform = '1'
```

语句二：

```
select count(*)
  from norn_ggk_record
 where jtime between to_date(to_char(sysdate, 'yyyy-MM-dd'), 'YYYY-mm-dd')
              and to_date(to_char(sysdate + 1, 'yyyy-MM-dd'), 'YYYY-mm-dd')
              and tel = '18847162790'
              and platform = '1'
              and hid = '8a5c83904e7213d1014e7236d16a0010'
```

语句三：

```
select ...
  from tp_userinfo this_
 where this_.platform = :1
        and this_.platformId = :2
```

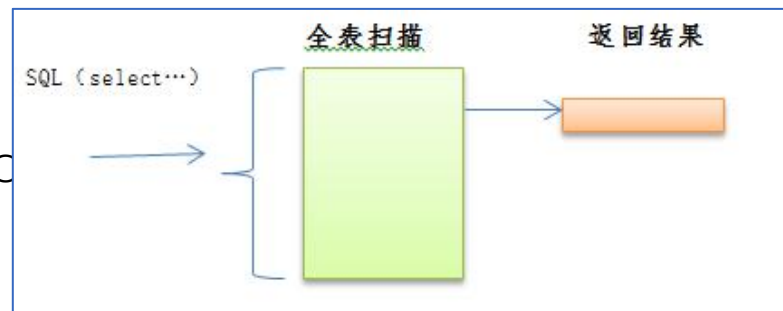


SQL语句效率差 – 创建有效索引

优化分析：平时活跃用户不是特别多，即使每次扫描全表读取了大量无用数据，SQL语句返回结果命中率差，但资源仍然能够及时得到响应，因此该业务能够正常运行。出现异常正是当时赠送量大，活跃用户激增，而且相关表迅速增长，导致资源争用。增加适当索引后，SQL语句的命中率提高，语句几乎不再访问无效的数据，因此降低了资源争用，CPU使用率及时恢复正常，业务也恢复正常运行。

索引一：

```
CREATE INDEX NORN.IND_GGK_GETER_TL  
ON NORN.NORN_GGK_GETER(hid,platform,jdate) TABLESPACE NMGORC
```



索引二：

```
CREATE INDEX NORN.IND_GGK_RECORD_tel  
ON NORN.NORN_GGK_RECORD(tel) TABLESPACE NMGORC_DATA
```

索引三：

```
CREATE INDEX NORN.IND_TP_USERINFO_ptid  
ON NORN.TP_USERINFO(platformId) TABLESPACE NMGORC_DATA
```



优化就这么简单

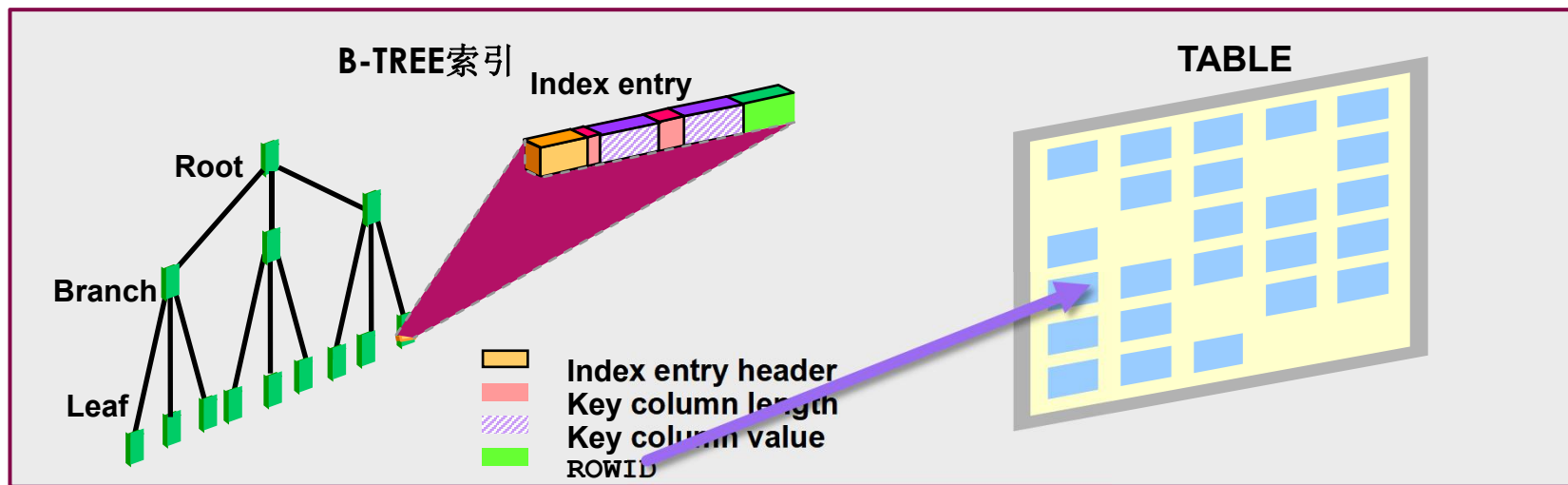
SQL语句效率差 – 创建有效索引

索引类似于书的目录，索引可以加快数据的检索速度，建索引允许指定单个列或者是多个列。但是索引在一定程度上减慢了数据录入的速度。

创建: `create index idx_emp_id on hr.employees_new (employee_id);`

删除: `drop index idx_emp_id name;`

重建: `alter index idx_emp_id rebuild;`



```
select *
  from all_ind_columns
 where table_owner = 'HR'
    and table_name = 'EMPLOYEES_NEW'
```

Row 1	Fields
INDEX_OWNER	SYSTEM
INDEX_NAME	IDX_EMP_ID
TABLE_OWNER	HR
TABLE_NAME	EMPLOYEES_NEW
COLUMN_NAME	EMPLOYEE_ID
COLUMN_POSITION	1
COLUMN_LENGTH	22
CHAR_LENGTH	0
DESCEND	ASC

SQL语句效率差导致执行速度慢

不能合理有效的利用资源是SQL语句慢的主要原因。大部分情况创建有效的索引并且正确使用可以加速SQL语句的处理速度。

1. 重复扫描，浪费IO资源

```
UPDATE ZG.CRD_ACCT_CREDIT_0483 d
SET (acct_name,cust_id) =
    (SELECT acct_name, NVL( cust_id,0)
     FROM ZG.CRM_ACCOUNT s
     WHERE s.so_nbr > :so_nbr AND s.so_nbr <= :max_sonbr
          AND d.acct_id = s.acct_id
          AND ( d.acct_name != s.acct_name OR d.cust_id != s.cust_id ))
WHERE EXISTS ( SELECT 0
               FROM ZG.CRM_ACCOUNT s
               WHERE s.so_nbr > :so_nbr AND s.so_nbr <= :max_sonbr
                    AND d.acct_id = s.acct_id
                    AND ( d.acct_name != s.acct_name OR d.cust_id != s.cust_id ))
```

将表ZG.CRD_ACCT_CREDIT_0483的数据与ZG.CRM_ACCOUNT数据重复查询后再次查询ZG.CRM_ACCOUNT数据后修改。

为什么不一次性查出来后统一修改呢？

2. 增加千万次to_char计算，浪费CPU资源

```
Select count(*)
from GPRS_20170720 A
WHERE TO_CHAR(start_date,'YYYYMMDD HH24:MI:SS') >='20170720 15:00:00'
      AND TO_CHAR(start_date,'YYYYMMDD HH24:MI:SS') <'20170720 15:30:00';
```

3. 因为没有合理的索引引起语句扫描了大量不必要的数据

```
select *
from CNEXT_OPER_tab
where sts = :1
      and rownum <=:2
      and valid_date <= :3
```


业务扫描次数多&SQL语句效率差

- 由业务原因要求SQL语句返回数据结果较多导致的语句执行缓慢，如业务批量处理，内存数据库初始化，客户流量详单查询等；
- SQL本身对资源消耗不大，但是执行频率非常高，如每小时执行百万次以上；
- 业务正在处理的数据与历史数据在同一表中导致表中冷数据不断上涨，SQL语句扫描到了无用的数据。

SQL orderd by CPU Time

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total	% Total DB Time	SQL Id	SQL Module	SQL Text
3,470	3,465	4,864	0.71	4.17	2.11	chwbdz02pga6v	sim_proc_file[0:12029]@CRMAPP4	select FILE_ID, FILE_TYPE_DTL,...
3,442	3,450	9,747	0.35	4.14	2.10	g1y3uydgknc8t	JDBC Thin Client	select sum(stock) from tm_stor...
3,405	3,419	61	55.82	4.09	2.08	dvk6kf36g8siv	manage_next_op[0:15326]@CRMAPP4	select SO_NBR, REGION_CODE, BU...
2,611	2,614	117	22.32	3.14	1.59	aqayys00g3sdz	JDBC Thin Client	SELECT SO_NBR, MSO_NBR, SERV_I...
2,334	4,439	6	388.92	2.80	2.70	5vm4ur0n68zf	BusiResMgmt_sv[0:18641]@CRMAPP4	SELECT T.RES_ATTRI_ID, T.RES_...
2,242	2,283	14,195,255	0.00	2.69	1.39	4jamxbky7mm38	manage_call_ba[0:1164]@CRMAPP3	select to_char(sysdate, :''SYS_...
1,982	1,967	275	7.21	2.38	1.20	8x8wg8vqv5r0	JDBC Thin Client	SELECT COUNT(imei_id) FROM TM_...
1,437	1,426	276	5.21	1.73	0.87	8vngvctdmecst	JDBC Thin Client	SELECT COUNT(imei_id) FROM TM_...
1,391	1,399	1,994	0.70	1.67	0.85	g7uyg8bixpizk	JDBC Thin Client	select ec_id, group_id, serv_c...



awr案例.html

aqayys00g3sdz	SELECT SO_NBR, MSO_NBR, SERV_ID, SE_ID, SE_NAME, PHONE_ID, SERV_CODE, BU... PROPERTY, CREATE_DATE, VALID_DATE, EXPIRE_DATE, REMARK, COMMIT_DATE, PROCESS_DATE, STS, S... REV2, REV3, ACT_TYPE, EC_NAME, ORD_TIME, SERV_NAME, EC_ID, SERV_TYPE, PROCESS_SEQ, FILE_SEQ... REV5, REV6, REV7, REV8, REV9, REV10 FROM (select from ZK.I_USER_IAGW where BUSI_TYPE = :1 and STS =... rownum < :3
---------------	--

STS值标记生产数据与历史数据之间的区别。生产与历史数据在同一表中。

4jamxbky7mm38	select to_char(sysdate, :''SYS_B_0'') from dual
---------------	---

SQL执行一次占用时间为0秒，
每小时执行1400万次以上。

主机或者数据库资源紧张

- 当主机发生异常或者数据库不在正常工作状态中会出现SQL语句执行缓慢;
- 锁的不合理使用会严重阻塞SQL执行速度, 锁包括: 表锁 (TM), 行锁 (TX), 锁 (latch);
- RAC跨界点相同数据访问会影响执行速度。

T1: SQL> select employee_id,salary from hr.employees a where a.employee_id=101;

EMPLOYEE_ID	SALARY
101	17000.00

T2: SQL> update hr.employees a set a.salary = 18000 where a.employee_id = 101;
1 row updated

T3: SQL> update hr.employees a set a.salary = 19000 where a.salary=17000;

T2: SQL> commit;

T3: 5 row updated
SQL> commit;

直到T2提交后T3
才会修改完成

T1: SQL> select employee_id,salary from hr.employees a where a.employee_id=101;

EMPLOYEE_ID	SALARY
101	1

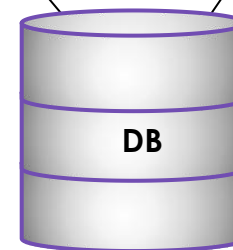
猜猜这是几??

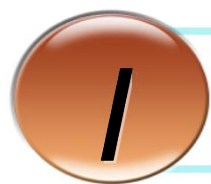
实例1

INSERT INTO A
VALUES
INSERT INTO A
VALUES
.....

实例2

SELECT * FROM A
WHERE
DELETE FROM A
WHERE
.....





SQL语句基础概念



SQL语句为什么会慢



SQL语句怎么写会快



低效率语句对数据库的影响

SQL语句怎么写会快

就当前已经在系统上运行的SQL语句，语句运行的快慢取决于该SQL所需要的资源是否被及时的获取到并被有效合理使用。SQL优化的实质在于保证结果正确的前提下，充份利用索引，减少表扫描的 I/O 次数，尽量少访问数据块，尽量避免全表扫描和其他额外开销。

业务架构时期，要求将生产数据表与历史数据表分开，并将历史数据分区或者分表以便于及时清理过期数据。

需要超高频率访问的数据在应用层做缓存以减少数据库负担。对繁重的业务进行批处理。

要求数据库表功能清晰，降低不必要的多表连接查询。建立有效索引。SQL语句书写规范，应用开发人员有一定的SQL基础，及时优化SQL语句。

对于RAC系统，对于同一对象操作尽量控制在同一个节点，降低集群间负担。同一事务处理完成后及时提交，并注意应用流程的先后顺序，避免跨数据库死锁。

保证数据的有效性

业务架构时期，要求将生产数据表与历史数据表分开，并将历史数据分区或者分表以便于及时清理过期数据。

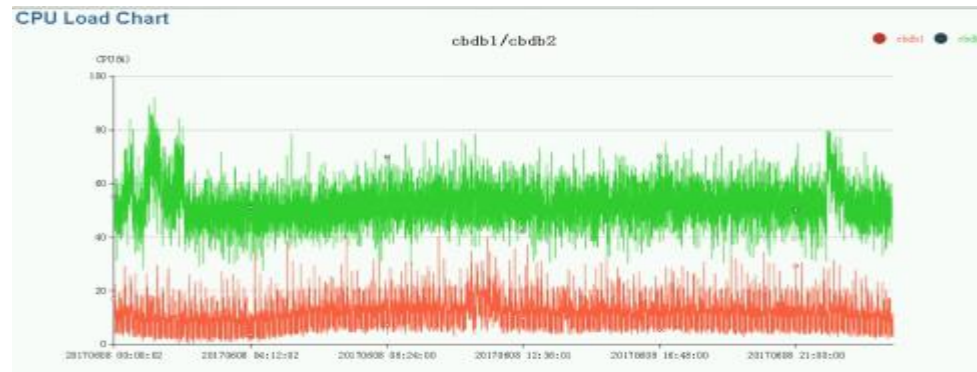
本案例为将表I_USER_RADIUS_INDEX 所有已处理数据清理后，主机CPU使用率下降20%以上。

如何清理无效数据需要业务人员提出，由DBA配合实施。

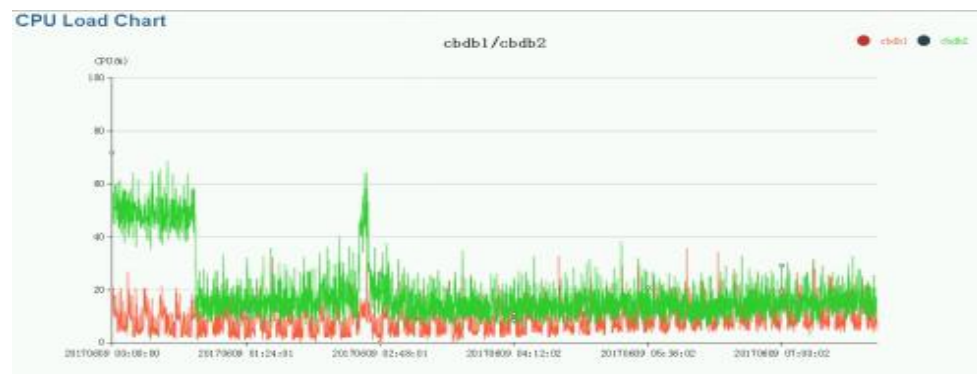
相关SQL语句

```
SELECT .....  
FROM (select * from I_USER_RADIUS_INDEX  
      where BUSI_TYPE = :1  
        and STS = :2  
        and MSO_NBR = :3  
        and mod (BILL_ID, :4)=:5  
        order by COMMIT_DATE, SO_NBR  
      )  
where rownum <= :6
```

清理前



清理后



SQL语句加速技巧

由于DBA并不了解业务逻辑，因此SQL语句效率差是从数据库上体现最直接的性能问题。优化SQL语句是DBA帮助业务性能提升的最有效方式，但效果非常有限。

以下是一些加速SQL语句执行速度的小技巧。

进行过计算或者函数运算后，不能使用索引

SQL> set autotrace traceonly

SQL> select * from hr.employees z where z.employee_id+1 = 101;

低效

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	68	3 (0)	00:00:01
* 1	TABLE ACCESS FULL	EMPLOYEES	1	68	3 (0)	00:00:01

SQL> select * from hr.employees z where z.employee_id=100;

高效

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	68	1 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	68	1 (0)	00:00:01
* 2	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1		0 (0)	00:00:01

SQL语句加速技巧

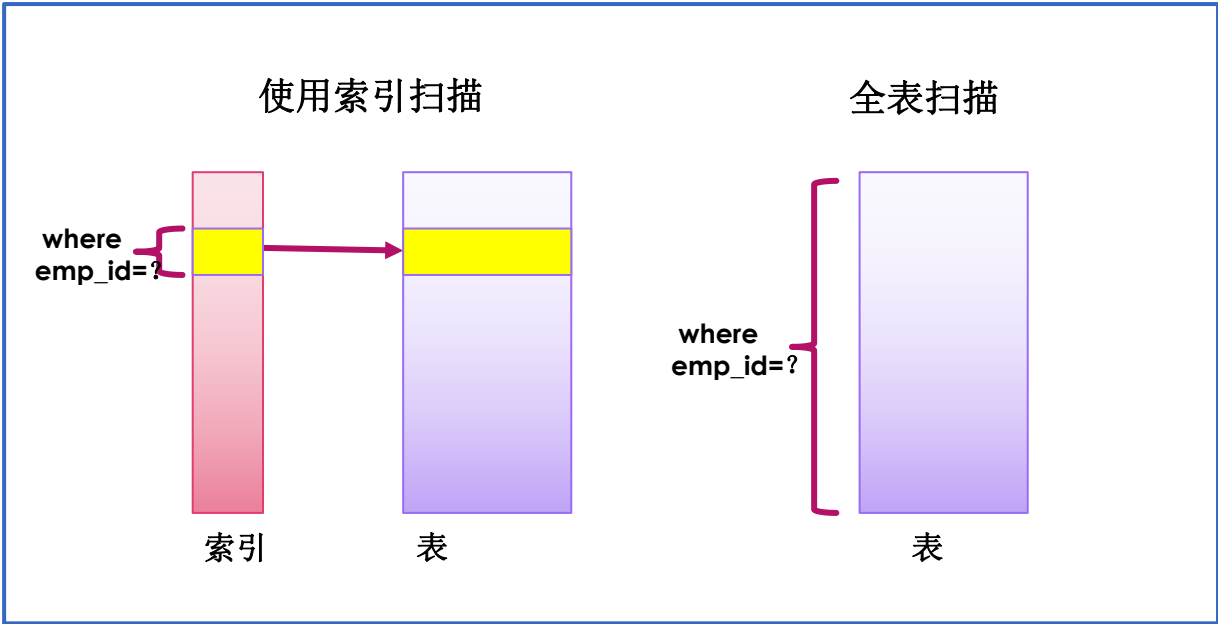
正确使用索引可以有效降低数据库扫描范围，加快对数据的访问速度。因此，为那些最经常出现在查询条件（WHERE column =）或排序条件（ORDER BY column）中的数据列创建索引尤为重要。

索引创建原则：选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据列）来创建索引。



```
SQL> select * from hr.employees a where a.employee_id=101;
```

Id	Operation	Name	Cost	(%CPU)
0	SELECT STATEMENT		1	(0)
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	(0)
* 2	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	0	(0)



SQL语句加速技巧

以下情形均无法使用索引

类型	举例
索引列的加工	<ul style="list-style-type: none">- 由于索引列使用函数等被加工- 输入的列值或常数中使用函数等，避免在索引列被加工- WHERE TO_CHAR(finish_date, 'YYYYMMDD') = '20121213'→ WHERE finish_date = TO_DATE('20121213', 'YYYYMMDD')- WHERE SUBSTR(name, 1, 6) = 'ENCORE'→ WHERE name LIKE 'ENCORE%'
数据类型发生内部变化	<p>进行比较的两列数据类型不同时，内部自动发生数据类型转换</p> <p>WHERE done_date LIKE TRUNC(SYSDATE) '%'</p> <p>→ WHERE done_date BETWEEN TRUNC(SYSDATE)</p> <p>AND TRUNC(SYSDATE+1) - 1/(60 * 60 * 24)</p> <p>→ WHERE done_date >= TRUNC(SYSDATE)</p> <p>AND done_date < TRUNC(SYSDATE+1)</p>
LIKE运算符的最前面使用'%'	<p>LIKE运算符的最前面使用 '%' 时，不能使用索引</p> <p>WHERE name LIKE '%ENCORE%'</p> <p>→ 生成域索引</p> <p>→ 与负责人协商是否可以去掉前面的 '%'</p>
NULL 搜索	<p>对 NULL 值的搜索</p> <p>WHERE done_date IS NULL</p> <p>→ done_date + Not Null 列添加到索引中</p> <p>→ 输入时用其他数据替换 NULL 值</p>
优化器的取舍选择	即使存在索引，但是优化器有时会生成不使用索引的执行计划

SQL语句加速技巧

复合索引(concatenated index)

由多个列构成的索引。

使用复合索引代替单列索引的原因
为对where条件中单个字段范围定义太广，使用多个列组合后可以增加索引扫描的有效性。

只有当复合索引的前导列出现在查询条件中时，才能有效使用到该索引。如果前导列不在查询条件中，复合索引可能被使用到，但是效率会很低，甚至不及全表扫描。

为不影响DML操作，复合索引字段不建议过多。

```
SQL> create index HR.EMP_NAME_IX on HR.EMPLOYEES (LAST_NAME, FIRST_NAME)
        tablespace USERS;
```

Index created.

```
SQL> select * from hr.employees a where a.first_name='Steven';
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		1	3 (0)
* 1	TABLE ACCESS FULL	EMPLOYEES	3	(0)

```
SQL> select * from hr.employees a where a.last_name='King';
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		1	2 (0)
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	2 (0)
* 2	INDEX RANGE SCAN	EMP_NAME_IX	1	1 (0)

```
SQL> select * from hr.employees a where a.last_name='King' and a.first_name='Steven';
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		1	2 (0)
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	2 (0)
* 2	INDEX RANGE SCAN	EMP_NAME_IX	1	1 (0)

SQL语句加速技巧

用 UNION ALL 代替 UNION

UNION 是最常用的集操作，使多个记录集联结成为单个集，对返回的数据行有唯一性要求，所以 oracle 就需要进行 SORTUNIQUE 操作（与使用 distinct 时操作类似），如果结果集又比较大，则操作会比较慢；

UNION ALL 操作不排除重复记录行，所以会快很多，如果数据本身重复行存在可能性较小时，用 union all 会比用 union 效率高很多！

低效

```
SQL> select * from hr.employees z
      union
      select * from hr.employees z ;
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		214	8 (63)
1	SORT UNIQUE		214	8 (63)
2	UNION-ALL			
3	TABLE ACCESS FULL	EMPLOYEES	107	3 (0)
4	TABLE ACCESS FULL	EMPLOYEES	107	3 (0)

高效

```
SQL> select * from hr.employees z
2  union all
3  select * from hr.employees z ;
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		214	6 (50)
1	UNION-ALL			
2	TABLE ACCESS FULL	EMPLOYEES	107	3 (0)
3	TABLE ACCESS FULL	EMPLOYEES	107	3 (0)

SQL语句加速技巧

减少对表的查询次数

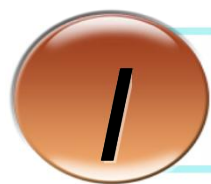
在含有子查询的SQL语句中, 要特别注意减少对表的查。

低
效

```
UPDATE ZG.CRD_ACCT_CREDIT_0483 d
  SET (acct_name,cust_id) =
      (SELECT acct_name, NVL( cust_id,0)
        FROM ZG.CRM_ACCOUNT s
       WHERE s.so_nbr > :so_nbr AND s.so_nbr <= :max_sonbr
          AND d.acct_id = s.acct_id
          AND ( d.acct_name != s.acct_name OR d.cust_id != s.cust_id ))
WHERE EXISTS ( SELECT 0
                FROM ZG.CRM_ACCOUNT s
               WHERE s.so_nbr > :so_nbr AND s.so_nbr <= :max_sonbr
                  AND d.acct_id = s.acct_id
                  AND ( d.acct_name != s.acct_name OR d.cust_id != s.cust_id ))
```

高
效

```
UPDATE (SELECT d.acct_name,d.cust_id,s.acct_name s_acct_name,NVL(s.cust_id,0) s_cust_id
        FROM ZG.CRM_ACCOUNT s, ZG.CRD_ACCT_CREDIT_0483 d
       WHERE s.so_nbr > :so_nbr
          AND s.so_nbr <= :max_sonbr
          AND d.acct_id = s.acct_id
          AND ( d.acct_name != s.acct_name OR d.cust_id != s.cust_id )
        )
SET   acct_name = s_acct_name,
       cust_id = s_cust_id
```



SQL语句基础概念



SQL语句为什么会慢



SQL语句怎么写会快



低效率语句对数据库的影响

低效率语句对数据库的影响

直接影响

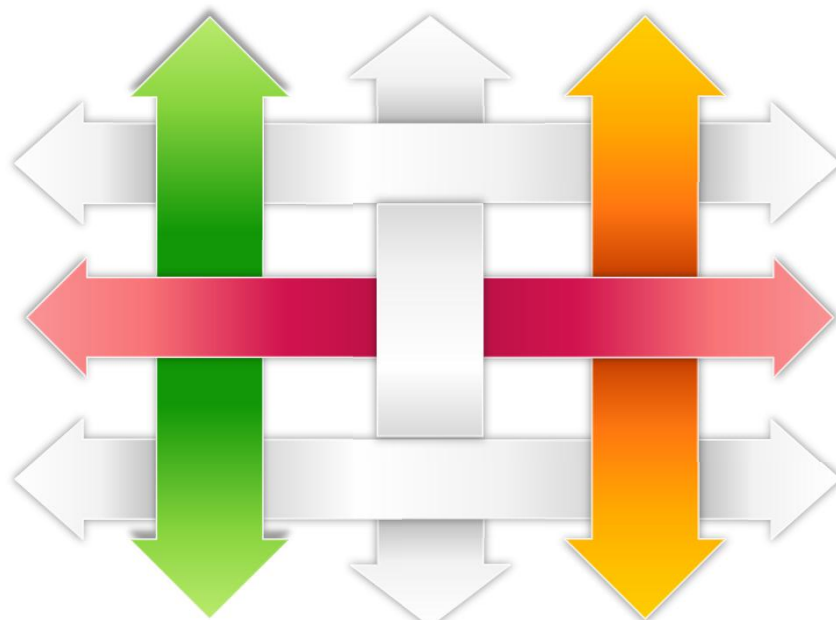
- 低效的SQL语句执行会直接降低前台业务页面的展现速度
- 低效的SQL语句会直接影响用户使用体验

影响其他业务

- 低效的SQL执行的同时会占用数据库大量资源，如CPU，IO，网络，TEMP表空间等
- 会影响到其他业务的高效运行

影响工作效率

- 低效的SQL会增加等待时间，影响工作人员工作效率
- 低效的SQL会增加工作人员工作强度



增加数据库风险

- 高频率且低效的SQL语句对大大增加数据库负担
- 高频率且低效的SQL语句有可能直接导致RAC数据库奔溃



**让我们的语句尽量跑的快一些
谢谢聆听！**

zhanghj@olm.com.cn