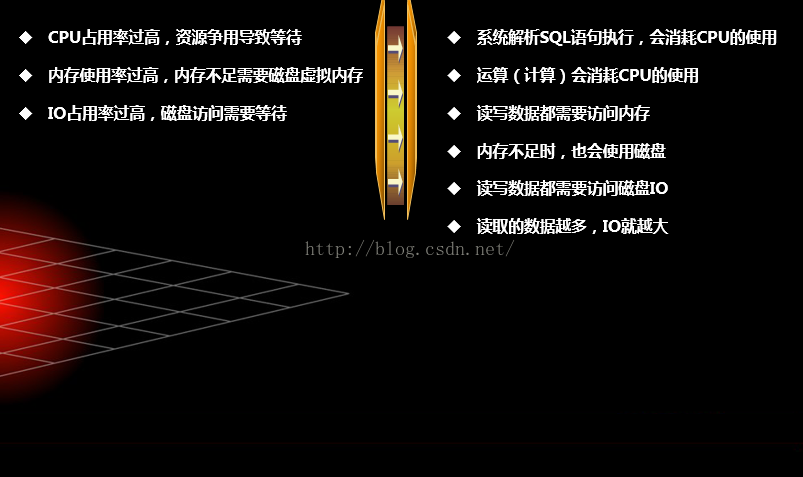
# 优化的理论基础



# 通过Select Count(?)进入优化之旅

不看百度或者GOOGLE说出下面3者的区别？

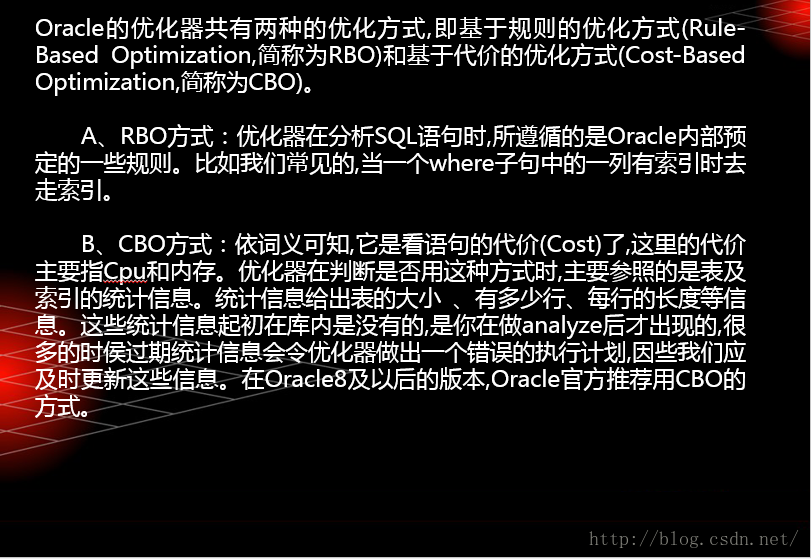
SELECT COUNT (\*)  
  
  
SELECT COUNT(1)  
  
  
SELECT COUNT(字段名）

## SELECT Count(?)的知识

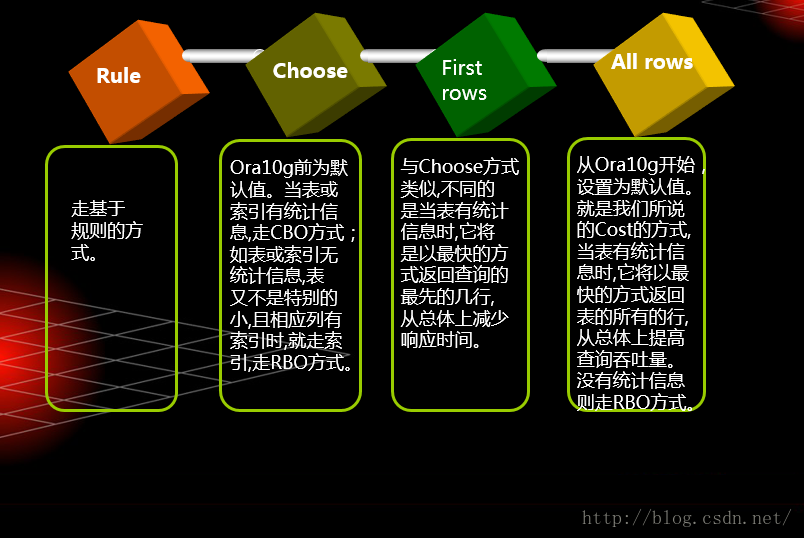


# ORACLE的优化器

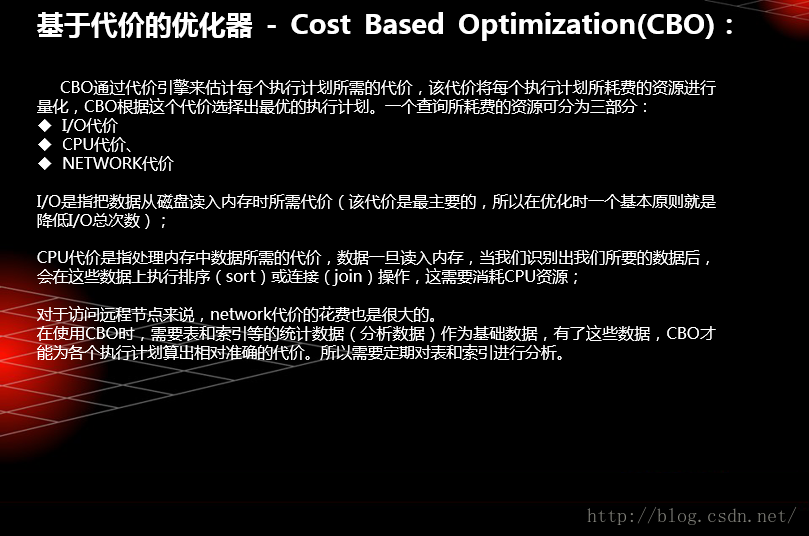
要说PLSQL优化，我们先需要来好好说一下Oracle优化器的知识：



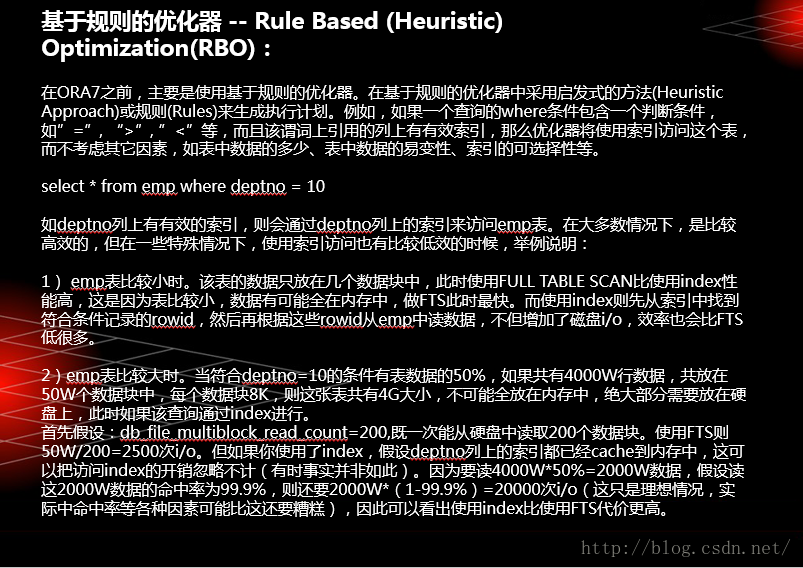
## 优化器的优化模式



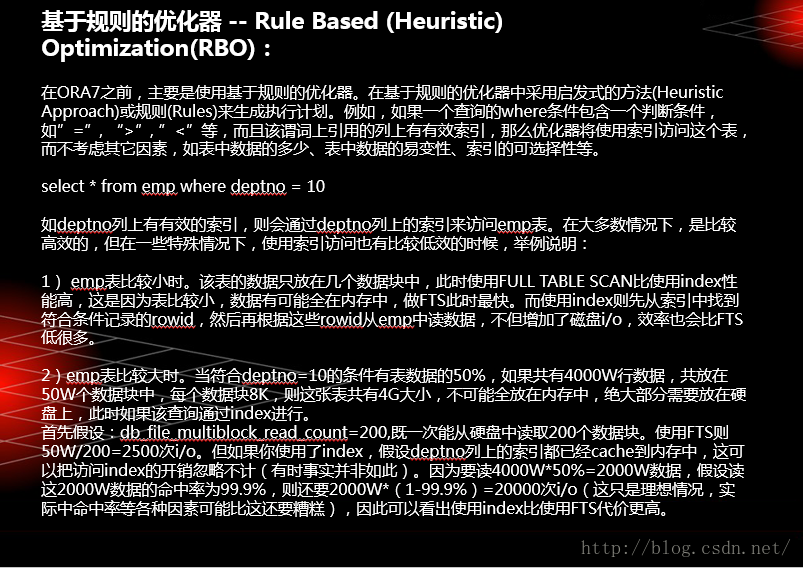
## CBO模式



## RBO模式



## 一起来看看oracle优化器的发展历程



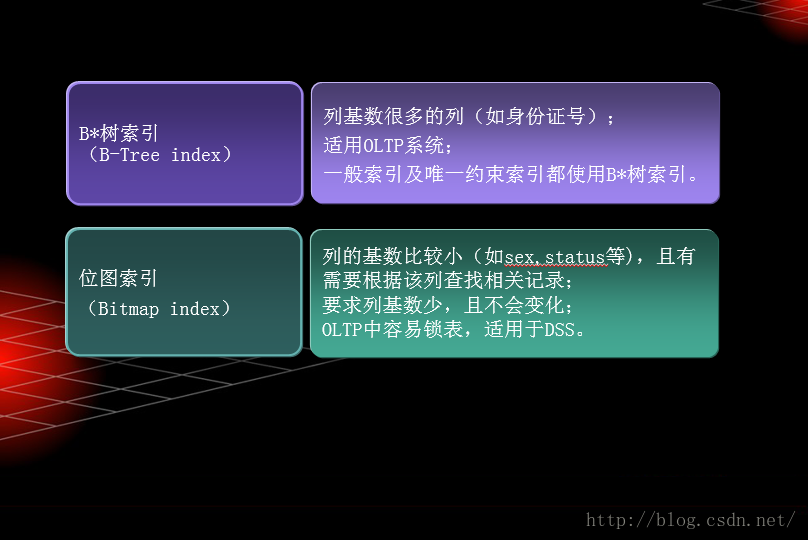
所以，我们知道ORACLE10后开始默认使用CBO，在CBO时ORACLE会自动来选择最优的执行计划，有时我们会认为：这个应该走索引更好啊，但是对于CBO来说，一个FULL TABLE ACCESS反而比索引更有效。

因此，在CBO的模式下，我们需要做的就是：

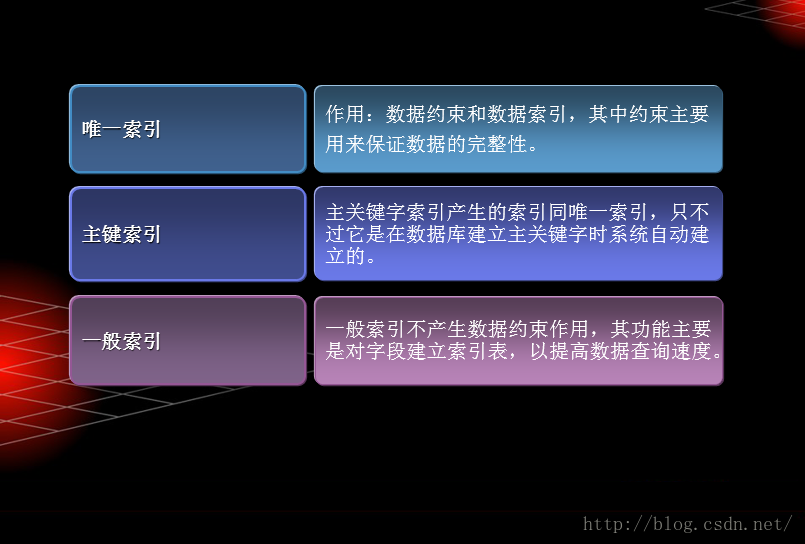
1. 做好数据库信息的相关统计
2. 合理建设我们的索引
3. 优化我们的SQL

索引是不需要修改SQL最直接带来性能提升的利器，何时该建索引，怎么建？怎么样让你的索引更合理？

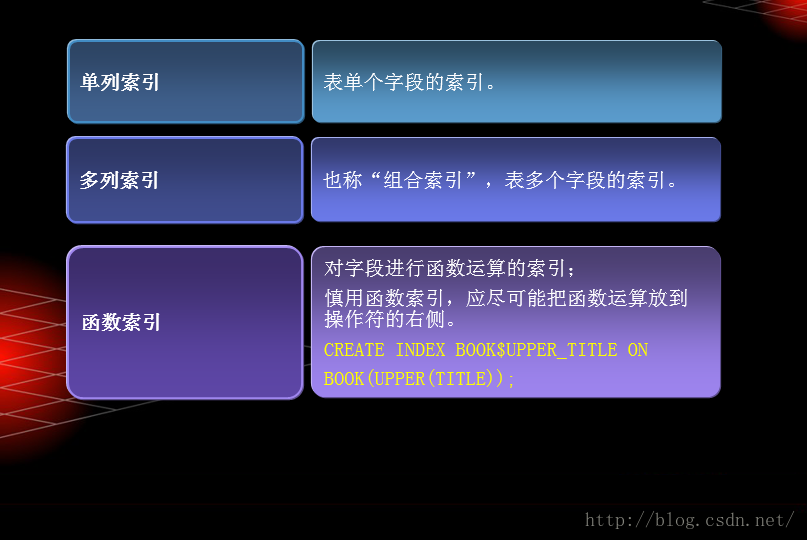
## 索引按内部结构分类



## 索引按功能分类

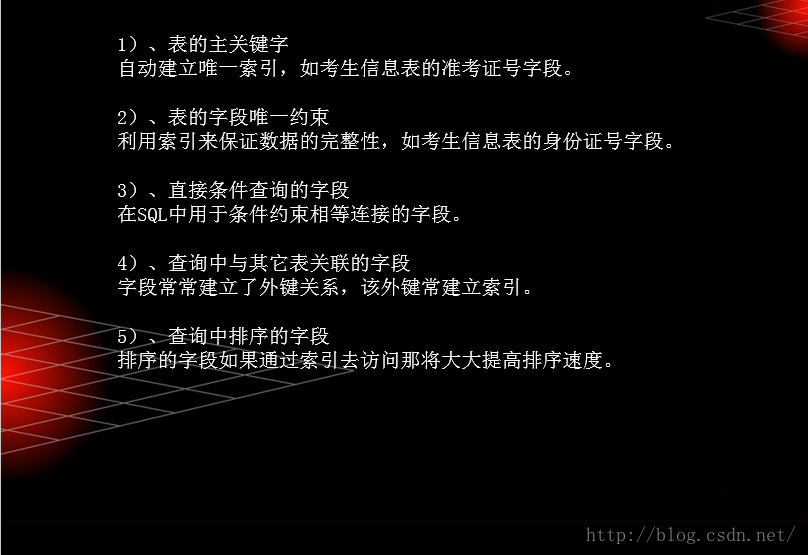


## 索引按索引对象分类



## 建立索引的方法论

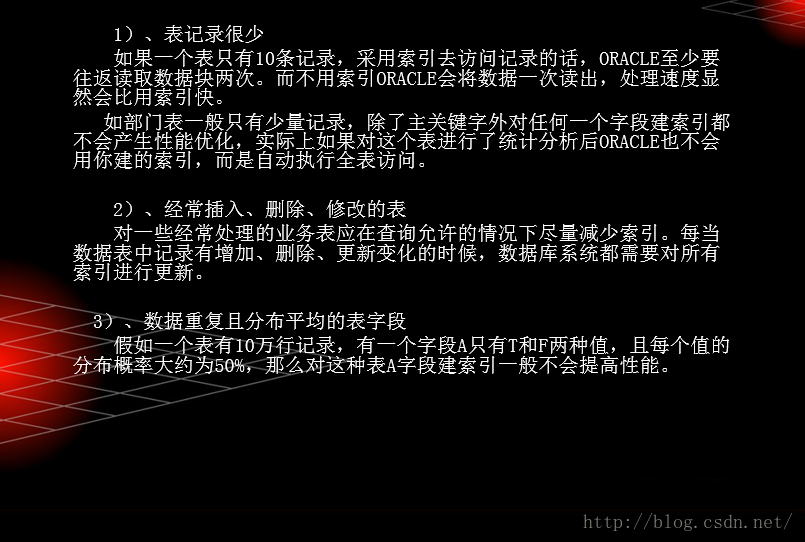
上面介绍了这么多索引的分类，下面来讲讲建立索引的方法论吧，大家可能较关心这个，因为这个是经验总结也是实战有用的利器哈。

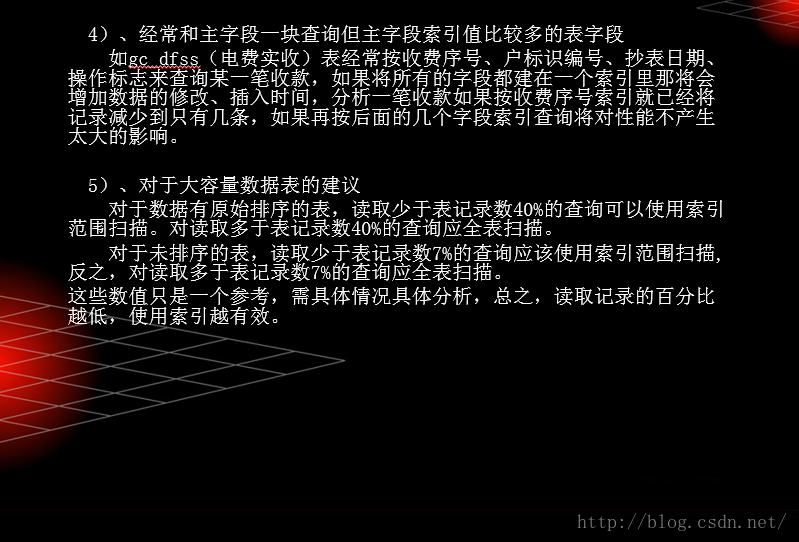




## 不建议建立索引的情况

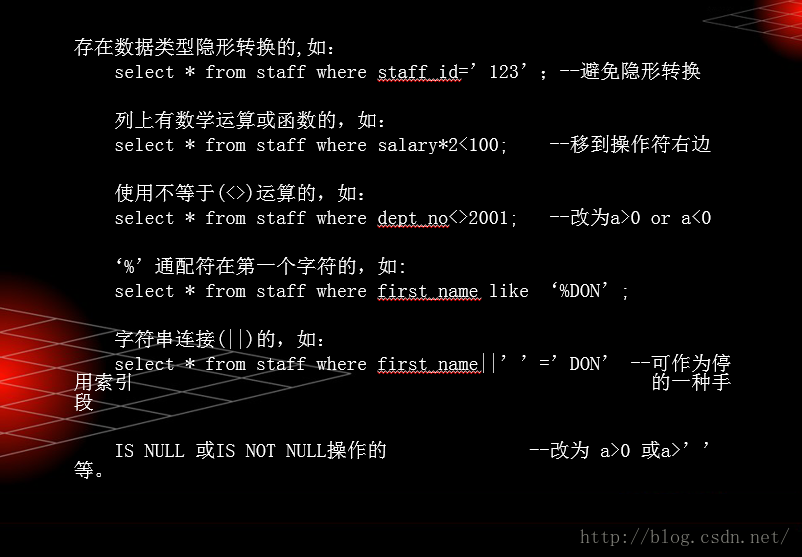
索引很神奇，可是索引不是万能，有时你建了索引也等于没用或者是白建、作无用功，为什么呢？我们看下去。





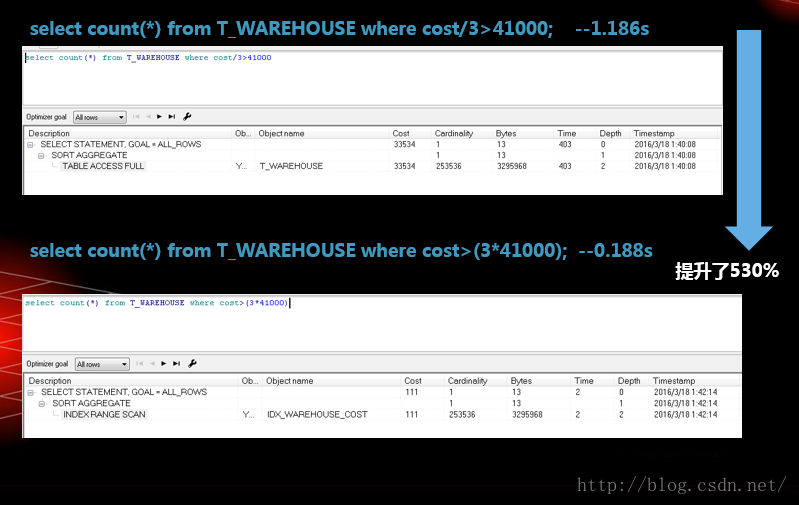
## 索引不会生效的情况

所以索引不要乱建，有时建了也是白建，为什么呢？来看看下面的案例分析吧：



## 以案例来说明

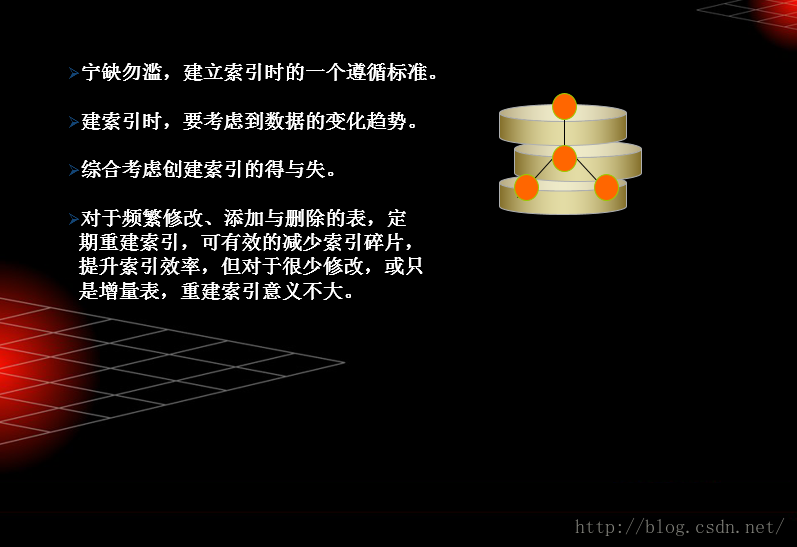
### PLSQL优化>一个不走索引的优化案例



这个例子说明了，如果你有一字参于WHERE条件查询的字段，但是它参于了运算符，因此它在ORACLE的内部执行计划中是不会走索引的，因此我们做了一个小小的变化，效率提升了多少倍？5.3倍，530%，呵呵！

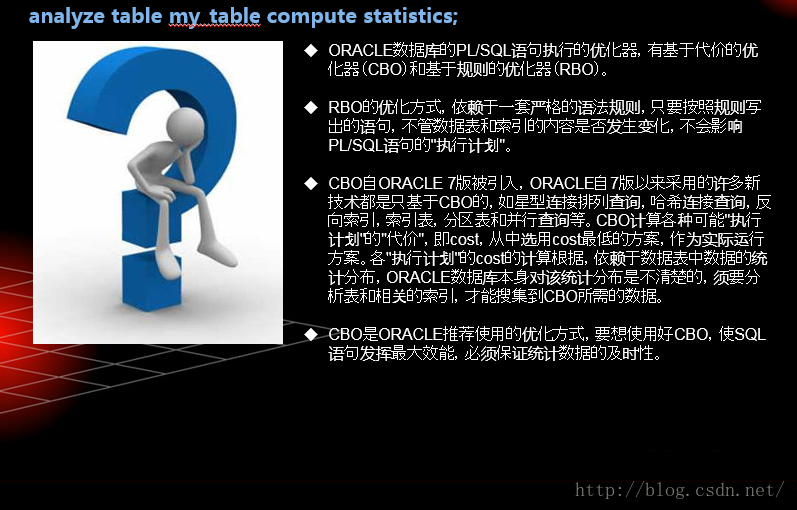
以上例我们可以为建立索引作一个总结。

## 建立索引的总结



# 

# Table Analyze



## Analyze Table VS DBMS\_STATS



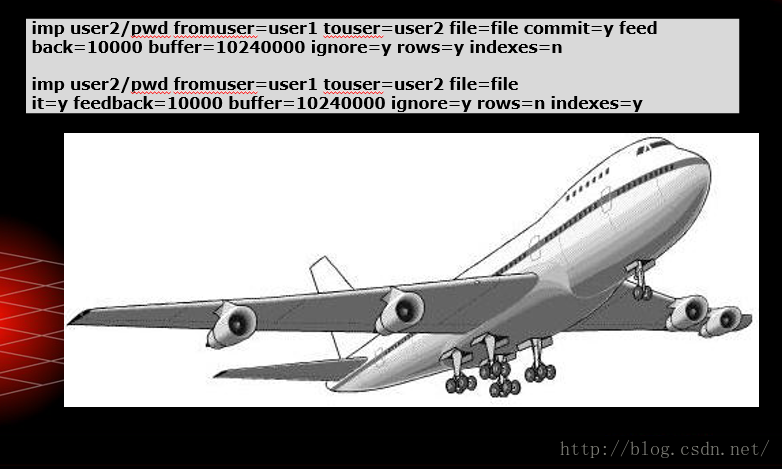
# Import & Export

说到Import & Export命令，大家会说。。。哎，这个不是很简单，就是：imp username/pwd@oraid file=path 吗？嘿嘿。。。试想：

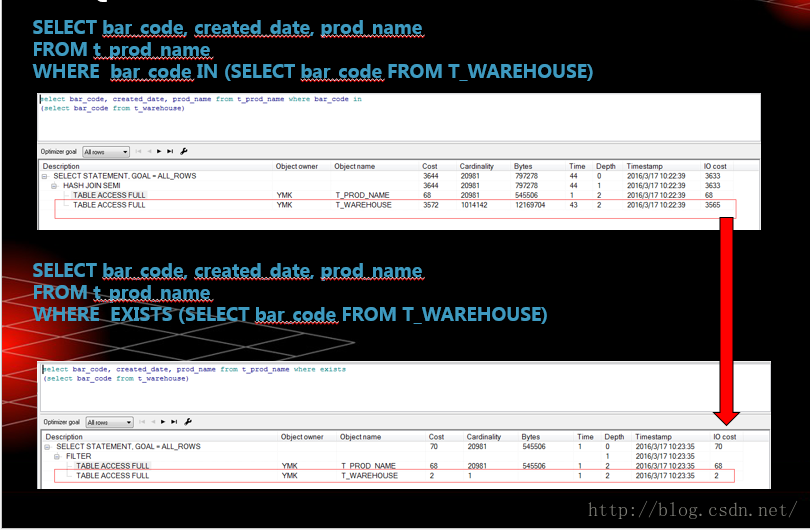
1. 你需要导入一个8GB左右的.dmp文件进入数据库
2. 你需要将一个库，其中含有至少30张表并且每张表都超过1200万条记录的数据进入一个.dmp文件

## Import的常规做法



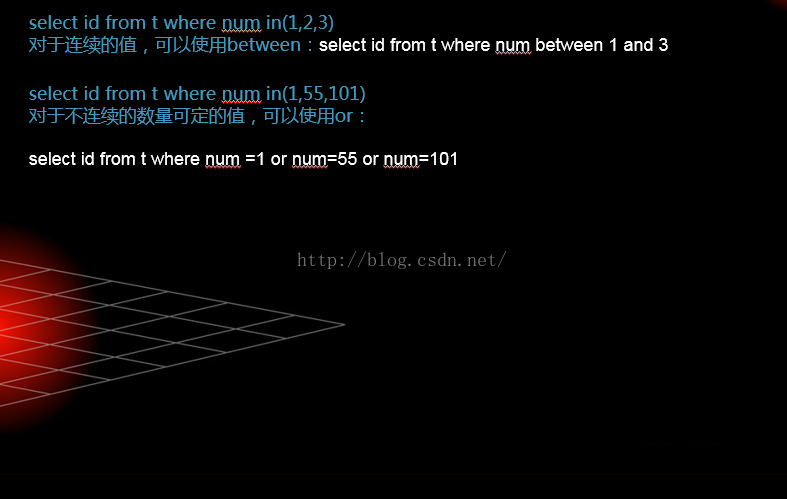


## PLSQL优化-SELECT IN 与SELECT EXISTS



这边提高了多少？光看IO就知道提高了多少了，呵 呵，很好玩吧？再来！

## PLSQL优化-SELECT IN的几种优化



## PLSQL优化-SELECT IN、OR、UNION的互转



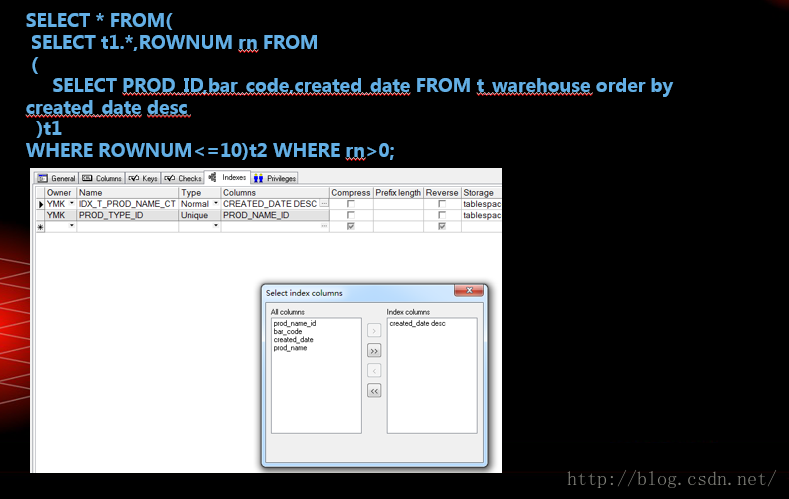
## PLSQL优化-分页语句中加入索引的优化

以下是一条分页语句，我们对created\_date做一个索引，等。。。。。。等等等，这边的索引不是一般的索引，我们把图形化工具建的索引翻译成SQL：

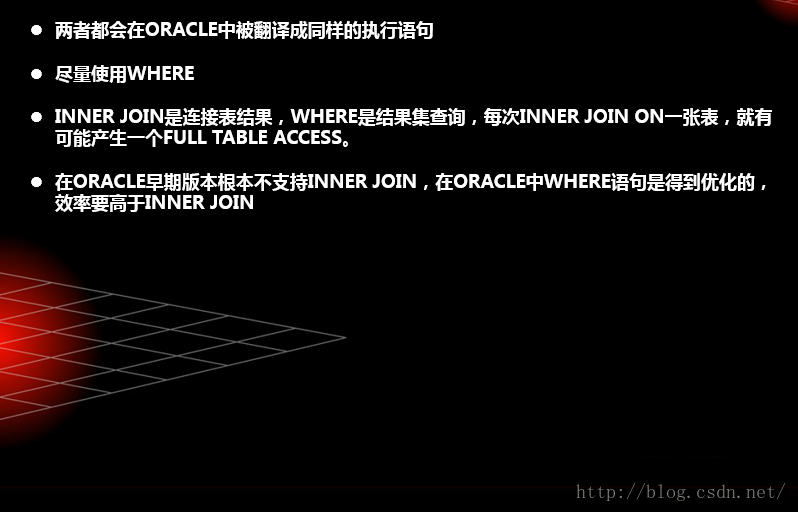
**[sql]** [view plain](http://blog.csdn.net/lifetragedy/article/details/51013069) [copy](http://blog.csdn.net/lifetragedy/article/details/51013069)

1. **create** **index** IDX\_WAREHOUSE\_CT **on** T\_WAREHOUSE(CREATED\_DATE **DESC**);

我们这个表是一个含有1000万条记录的表，仅此一招，整个SQL查询提高了300%-340%

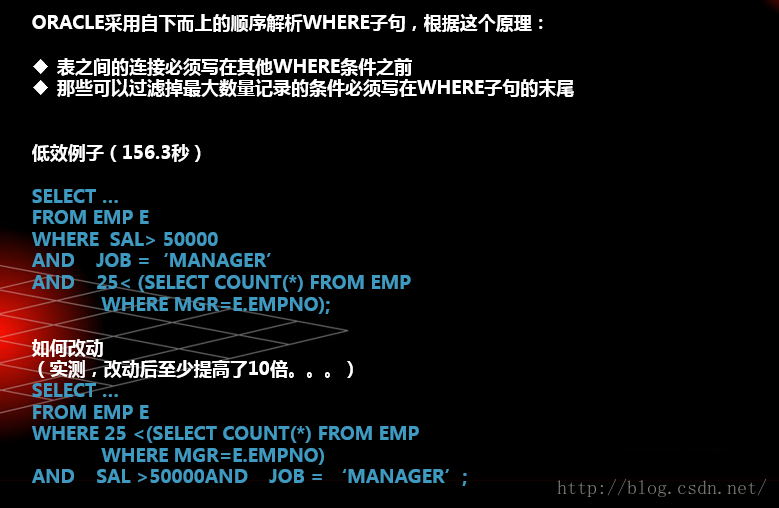


## PLSQL优化-INNER JOIN VS WHERE



## PLSQL优化-WHERE语句优化要点

注意下面这个例子，只是WHERE条件后的顺序上下颠倒一下，就提高了10倍的效率，呵呵。



### WHERE语句中选择最有效的表名顺序





PLSQL优化-用UNION取代OR



是不是写SQL时稍微注意一下。。。这个效率。。。这个性能 。。。123%。。。123%的提高啊。

**4.10 Partition Table（分区表）**



Oracle的表分区功能通过改善可管理性、性能和可用性，从而为各式应用程序带来了极大的好处。通常，分区可以使某些查询以及维护操作的性能大大提高。此外,分区还可以极大简化常见的管理任务，分区是构建千兆字节数据系统或超高可用性系统的关键工具。  
   
分区功能能够将表、索引或索引组织表进一步细分为段，这些数据库对象的段叫做分区。每个分区有自己的名称，还可以选择自己的存储特性。从数据库管理员的角度来看，一个分区后的对象具有多个段，这些段既可进行集体管理，也可单独管理，这就使数据库管理员在管理分区后的对象时有相当大的灵活性。但是，从应用程序的角度来看，分区后的表与非分区表完全相同，使用 SQL DML 命令访问分区后的表时，无需任何修改。  
   
什么时候使用分区表：

1. 表的大小超过2GB
2. 表中包含历史数据，新的数据被增加都新的分区中。

表分区有以下优点：

1. 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。
2. 增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用；
3. 维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可；
4. 均衡I/O：可以把不同的分区映射到磁盘以平衡I/O，改善整个系统性能。

缺点：   
分区表相关：已经存在的表没有方法可以直接转化为分区表。不过 Oracle 提供了在线重定义表的功能。

下面来看几种Oracle中分区的用法吧

### Range Partition（根据范围来分区）

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8215312) [copy](http://blog.csdn.net/lifetragedy/article/details/8215312)

1. PARTITION BY RANGE (CUSTOMER\_ID)
2. (
3. PARTITION CUS\_PART1 VALUES LESS THAN (100000) TABLESPACE
4. CUS\_TS01, PARTITION CUS\_PART2 VALUES LESS THAN (200000)
5. TABLESPACE CUS\_TS02
6. )

### Hash partition（HASH分区）这个最傻瓜了最好用了，不需要指定分区的条件的

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8215312) [copy](http://blog.csdn.net/lifetragedy/article/details/8215312)

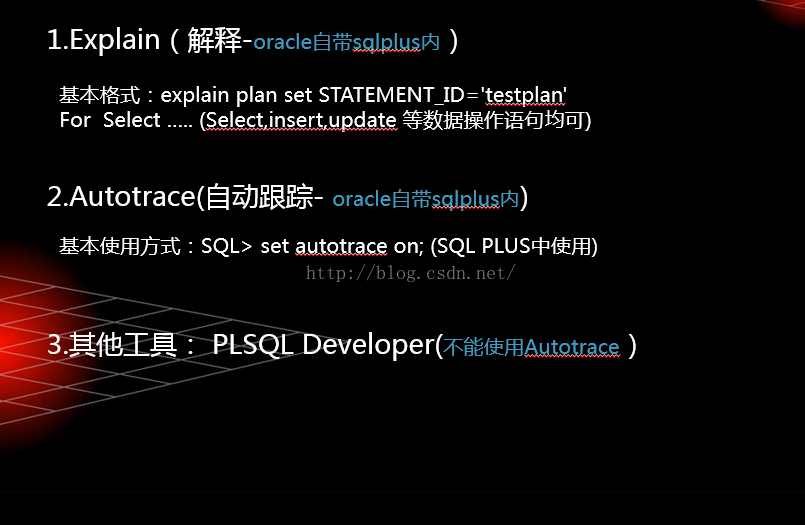
1. CREATE TABLE emp
2. (
3. empno NUMBER (4),ename VARCHAR2 (30),sal NUMBER
4. )PARTITION BY HASH (empno) PARTITIONS 8
5. STORE IN (emp1,emp2,emp3,emp4,emp5,emp6,emp7,emp8);

### Component Partition

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8215312) [copy](http://blog.csdn.net/lifetragedy/article/details/8215312)

1. create table dinya\_test
2. (
3. transaction\_id number primary key,item\_id number(8) not null,transaction\_date date
4. ) partition by range(transaction\_date)subpartition by hash(transaction\_id) subpartitions
5. 3 store in (dinya\_space01,dinya\_space02,dinya\_space03)
6. (
7. partition part\_01 values less than(to\_date(‘2006-01-01’,’yyyy-mm-dd’)),
8. partition part\_02 values less than(to\_date(‘2010-01-01’,’yyyy-mm-dd’)),
9. partition part\_03 values less than(maxvalue)
10. );

## 行计划的方式



## 伪列-ROWID

rowid是一个伪列，既然是伪列，那么这个列就不是用户定义，而是系统自己给加上的。对每个表都有一个rowid的伪列，但是表中并不物理存储ROWID列的值。不过你可以像使用其它列那样使用它，但是不能删除改列，也不能对该列的值进行修改、插入。一旦一行数据插入数据库，则rowid在该行的生命周期内是唯一的，即即使该行产生行迁移，行的rowid也不会改变。

## Recursive SQL

有时为了执行用户发出的一个sql语句，Oracle必须执行一些额外的语句，我们将这些额外的语句称之为‘recursive calls’或‘recursive SQL statements’。如当一个DDL语句发出后，ORACLE总是隐含的发出一些recursive SQL语句，来修改数据字典信息，以便用户可以成功的执行该DDL语句。当需要的数据字典信息没有在共享内存中时，经常会发生Recursive calls，这些Recursive calls会将数据字典信息从硬盘读入内存中。用户不比关心这些recursive SQL语句的执行情况，在需要的时候，ORACLE会自动的在内部执行这些语句。当然DML语句也都可能引起recursive SQL。简单的说，我们可以将触发器视为recursive SQL。

## Row Source and Predicate

* Row Source(行源)：用在查询中，由上一操作返回的符合条件的行的集合，即可以是表的全部行数据的集合；也可以是表的部分行数据的集合；也可以为对上2个row source进行连接操作(如join连接)后得到的行数据集合。
* Predicate(谓词)：一个查询中的WHERE限制条件

## Driving Table

Driving Table(驱动表)：该表又称为外层表(OUTER TABLE)。这个概念用于嵌套与HASH连接中。如果该row source返回较多的行数据，则对所有的后续操作有负面影响。注意此处虽然翻译为驱动表，但实际上翻译为驱动行源(driving row source)更为确切。一般说来a，是应用查询的限制条件后，返回较少行源的表作为驱动表，所以如果一个大表在WHERE条件有有限制条件(如等值限制)，则该大表作为驱动表也是合适的，所以并不是只有较小的表可以作为驱动表，正确说法应该为应用查询的限制条件后，返回较少行源的表作为驱动表。在执行计划中，应该为靠上的那个row source，后面会给出具体说明。

## Probed Table

Probed Table(被探查表)：该表又称为内层表(INNER TABLE)。在我们从驱动表中得到具体一行的数据后，在该表中寻找符合连接条件的行。所以该表应当为大表(实际上应该为返回较大row source的表)且相应的列上应该有索引。

## 组合索引(concatenated index)

由多个列构成的索引，如create index idx\_emp on emp(col1, col2, col3, ……)，则我们称idx\_emp索引为组合索引。在组合索引中有一个重要的概念：引导列(leading column)，在上面的例子中，col1列为引导列。当我们进行查询时可以使用”where col1 = ? ”，也可以使用”where col1 = ? and col2 = ?”，这样的限制条件都会使用索引，但是”where col2 = ? ”查询就不会使用该索引。所以限制条件中包含先导列时，该限制条件才会使用该组合索引。

## 可选择性(selectivity)

比较一下列中唯一键的数量和表中的行数，就可以判断该列的可选择性。如果该列的”唯一键的数量/表中的行数”的比值越接近1，则该列的可选择性越高，该列就越适合创建索引，同样索引的可选择性也越高。在可选择性高的列上进行查询时，返回的数据就较少，比较适合使用索引查询。

# oracle访问数据的存取方法（高实战）

## http://img.blog.csdn.net/20160505095552903?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

## 索引扫描的细分（Index Scan）





## 表连接



## 表连接（高实战2）



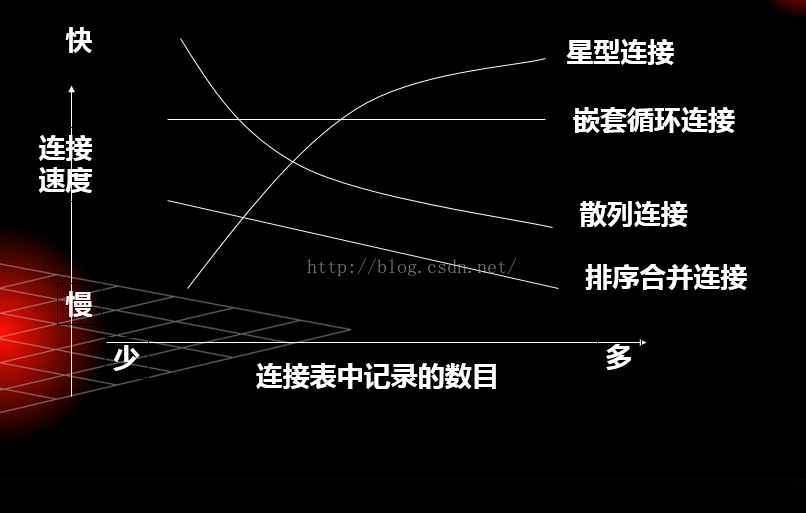
## 表连接（高实战3）

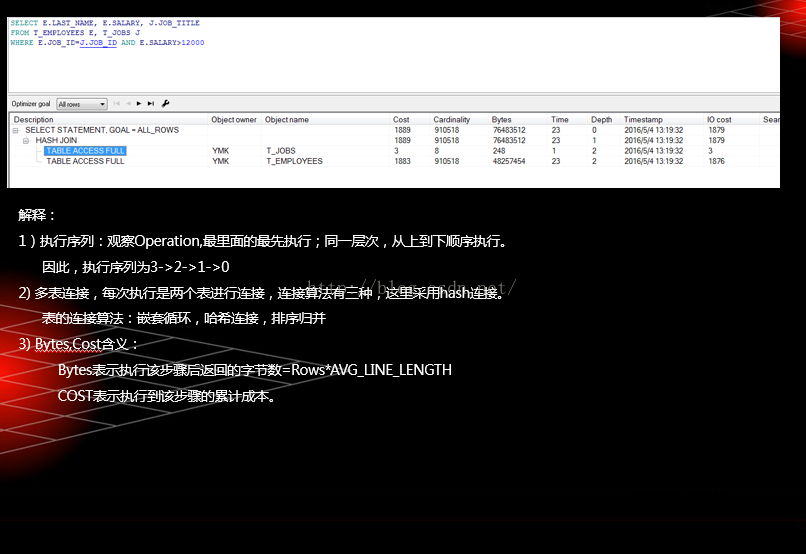


## 表连接（高实战4）

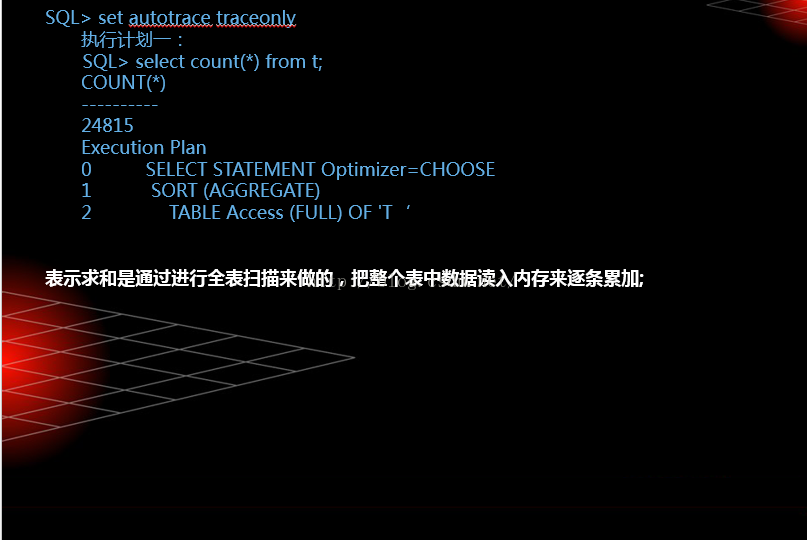


# 不同表连接的相对速度

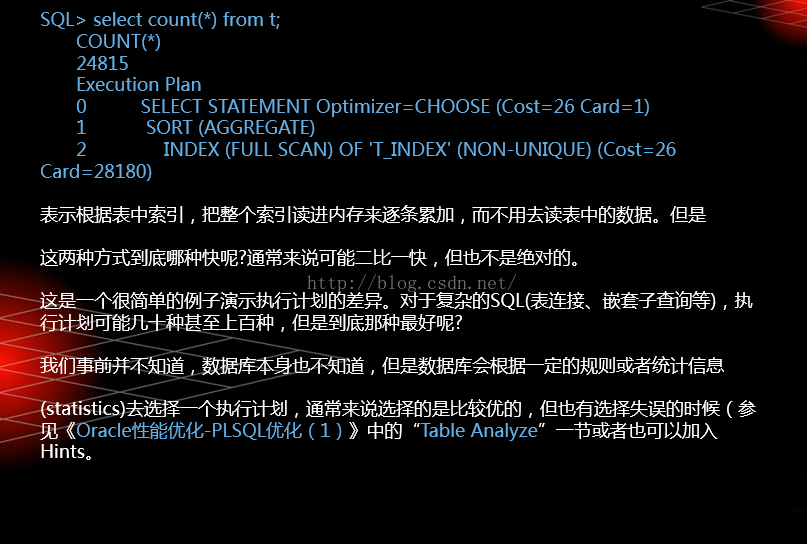




# 再来看2个执行计划（1）



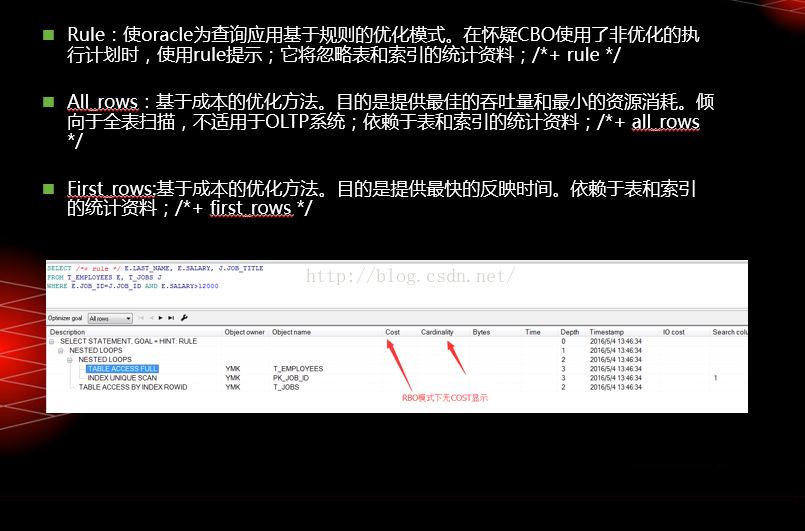
## 再来看2个执行计划（2）



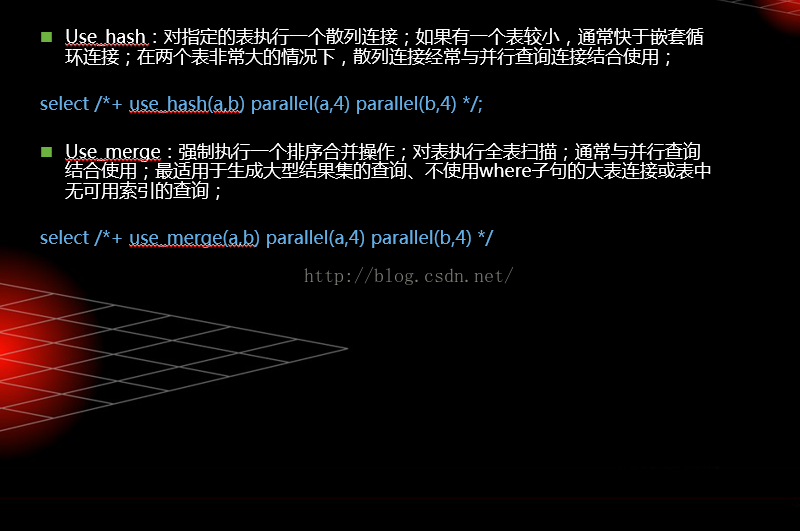
# Oracle中的Hints（提示）

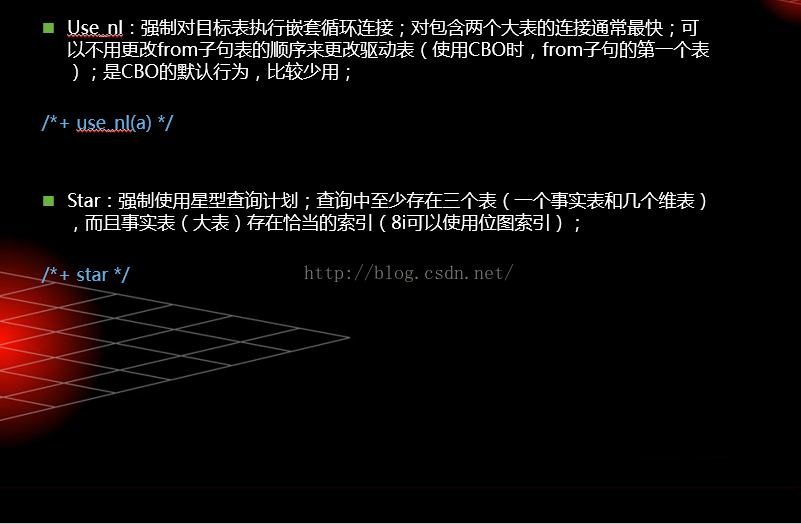


## 优化器提示

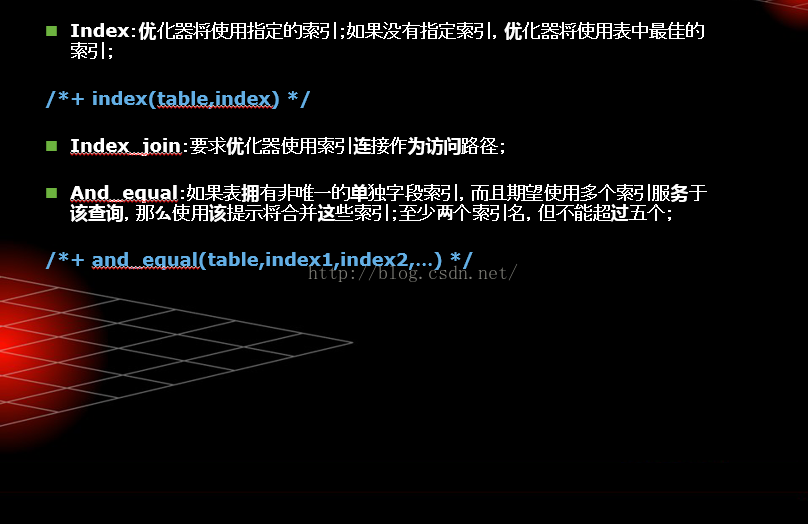


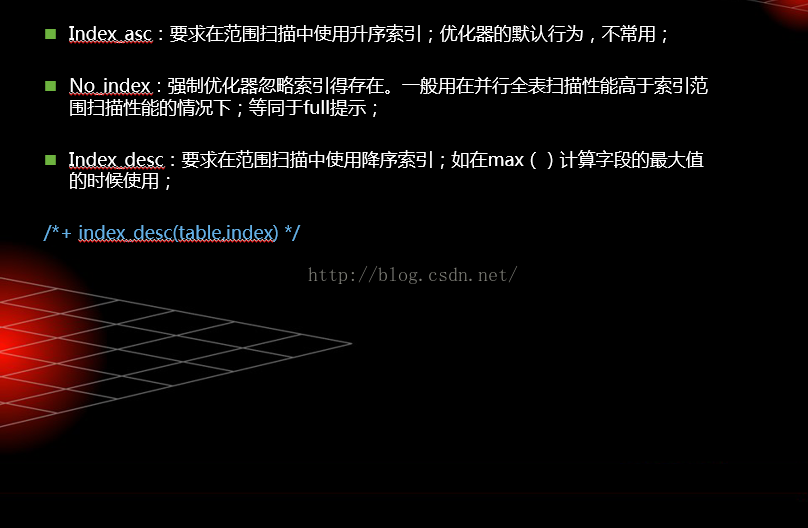
## 表连接提示

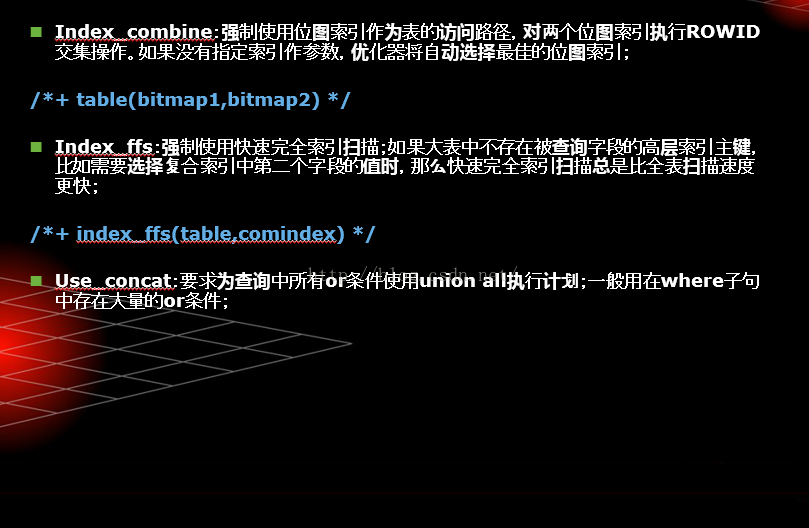




## 索引提示



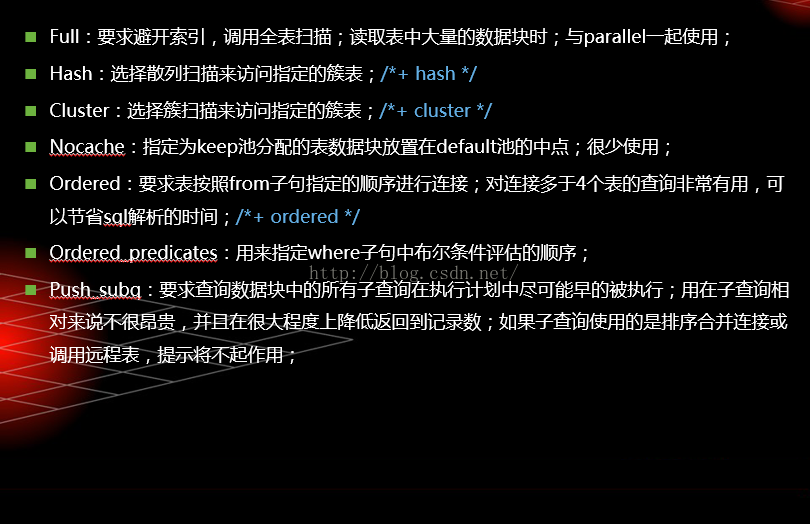




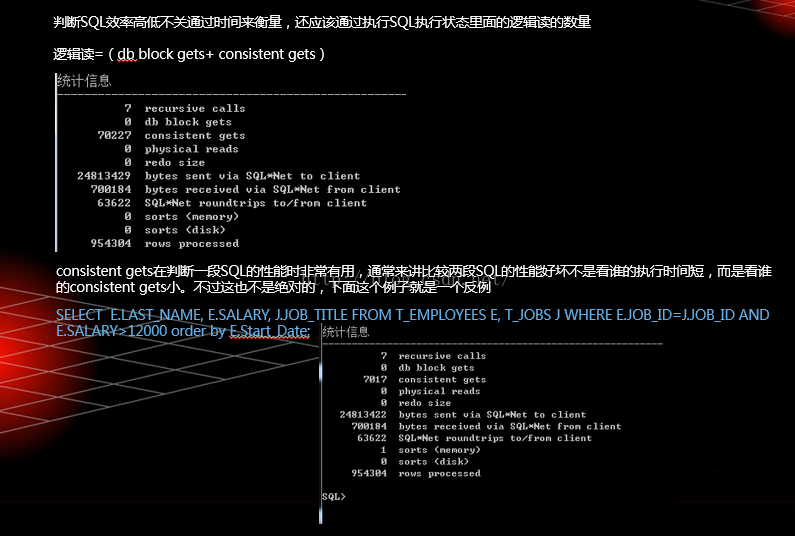
## 并行提示



## 表访问提示

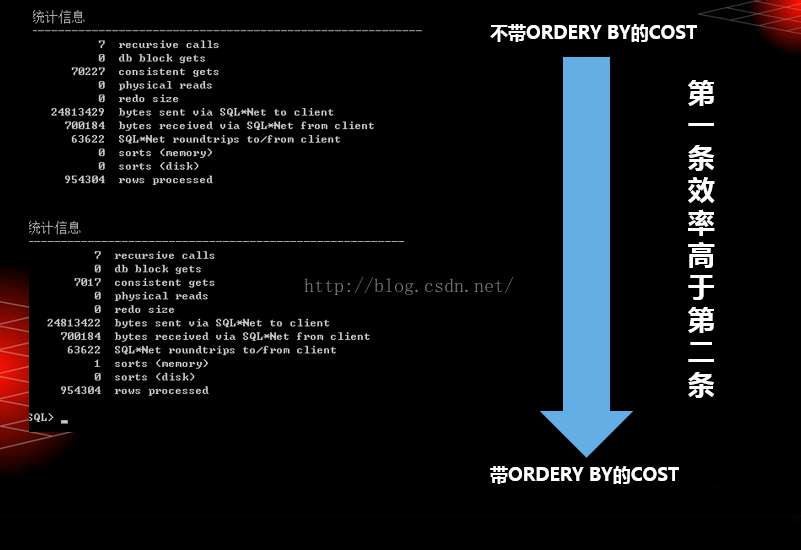


# 索引和SQL语句的正确使用



第1个不加order by的SQL肯定比第2个SQL效率高是毋庸置疑的。  
  
  
但是为什么第2个SQL的consistent gets如此之少?  
  
  
原因有如下两点：

1. 通常情况下，不在logical RAM buffer中的数据要通过physical reads来读取，而physical reads后通常会紧跟着一个consistent gets。因此一般情况下consistent gets是要比physical reads大的。但是有一个特例，如果physical reads得到的数据直接用于HASH或者SORT，则只记为physical reads不记为consistent gets。所以加上order by后有可能physical reads多但consistent gets少。不过这个原因不是我这里现象产生的原因，因为我这个实验里根本没有physical reads。
2. arraysize的影响。arraysize是指读取数据时一次读取得到的行数。这个值默认为15，使用show arraysize命令可以查看。一个数据块例如有100条记录，那么并不是读取这个块一次就能取到所有数据，以arraysize=15为例，就要有100/15=7次consistent gets。把arraysize设置得大一点可以降低consistent gets，不过有时候可能会消耗更多的资源。如果我们做select count(0) from test;操作，那么Oracle会把arraysize暂时设为test的行数，因此consistent gets会很少。很少：



# AUTOTRACE的几个常用选项

set autotrace off ： 不生成autotrace 报告，这是缺省模式

set autotrace on explain：  autotrace只显示优化器执行路径报告

set autotrace on statistics： 只显示执行统计信息

set autotrace on： 包含执行计划和统计信息

set autotrace traceonly： 同set autotrace on，但是不显示查询输

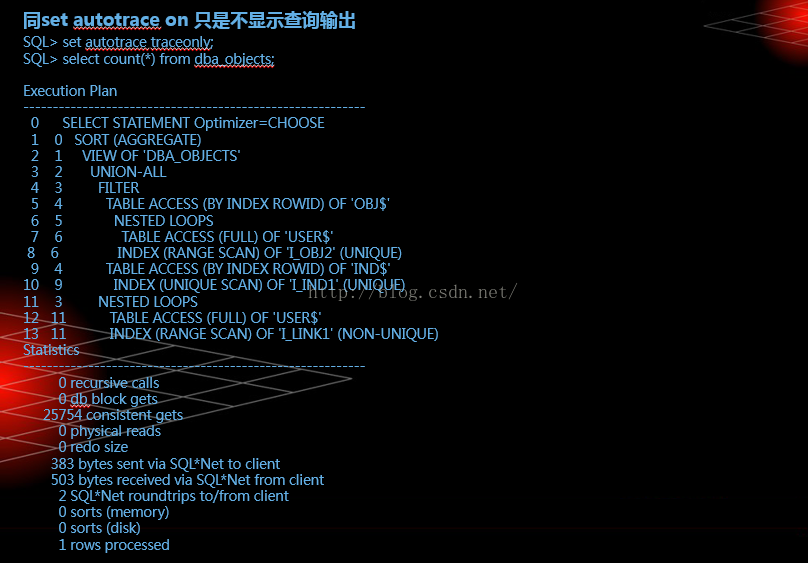
## set autotrace on explain



## set autotrace on statistics



## set autotrace traceonly



## set autotrace traceonly explain



## Explain Plan-查看执行方案

