**1、本章面试题**

     什么是索引，按物理分类有哪些种类，分别有什么特点，实际使用例子

sql语句的优化

什么是SGA PGA,运行的体系结构如何（DBA会问）

**2、知识点**

**2.1、课程回顾**

游标cursor（使用过程：定义，打开， 提取数据，关闭 带参数的游标，游标变量，for简化游标）

触发器trigger (dml触发器，行级触发器：old :new ,使用触发器完善oracle功能（级联更新）, 调用存储过程)

**2.2、本章重点**

索引

语句优化

SGA PGA

**3、具体内容**

**3.1 索引**

**3.1.1 概念**

索引是关系型数据库中用于存放每一条记录的一种对象（关联的是表中的每一条数据）。主要目的是为了加快数据的查询速度和完整性校验。创建索引是技术性非常强的工作，一般在数据库设计阶段和建模时去创建。整个系统的使用的速度和索引有直接关系。

**3.1.2 分类**

**1)逻辑分类**

**单列索引和复合索引**

一个索引可以由一个列或者是多个列组成，只在一个列上创建的索引称为单**列**索引，在多个列创建叫复合索引

**唯一索引和非唯一索引**

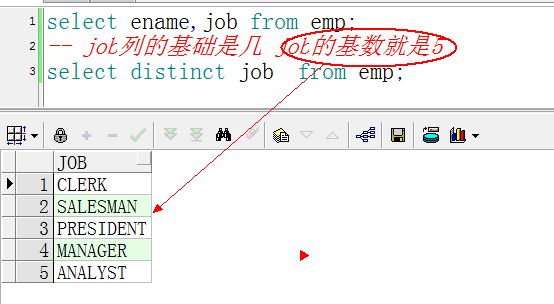
唯一索引是索引列的值不能重复，称为唯一索引，反之称为非唯一索引

**函数索引**

在oracle实际查询时不建议在列上使用函数，但是如果不可避免的情况可以使用函数，可以在函数上添加索引，该索引就叫函数索引。聚合函数(avg,min,max,sum等)是不可以使用函数索引。可以使用的：length,trim,substr,lower,upper,instr 等。

**2)物理分类**

**基数：在查询表数据是，一个列去掉重复项后得到的值就是该列的基数。**



性别列：sex gander 数据库无论有多少数据，基数都是2 中国行政省份：provence 基数都是34

**B树索引**

使用B树机制存储索引条目，可以保证最短路径访问到键和值，一般基数较大的列建议使用B树索引

**位图索引**

使用位图偏移的方式索引条目，一般基数较小的列建议使用位图索引

**3.1.3 使用原则**

  一般不需要为数据量很小的表创建索引

  对于数据量比较大的表，如果经常需要查询的记录数小于表中所有记录数的10%，则可以考虑为该表创建索引 100W 经常查询的是10W条

  对于基数大（取值范围较大）的列（如empno列），应该创建B树索引；对于基数小（取值范围较小）的列（如sex列），应该创建位图索引

  不能在CLOB(4G)或BLOB等大对象数据类型的列上创建索引(在企业中大对象类型使用时一定谨慎)

   如果在大部分情况下只需要对表执行只读操作（大数据框架对大量的日志数据进行分析），就可以为该表创建更多的索引以提高查询速度

如果在大部分情况下需要对表执行更新(增删改)操作，则应该为少创建一些索引，以提高更新速度

 至少要包含组合索引的第一列（即如果索引建立在多个列上，只有它的第一个列被where子句引用时，优化器才会使用该索引 job sal comm）, 使用create 组合index时要将最常查询的列放在最前面

经常进行连接查询的列(如同dept表和emp表中的deptno)应该创建索引

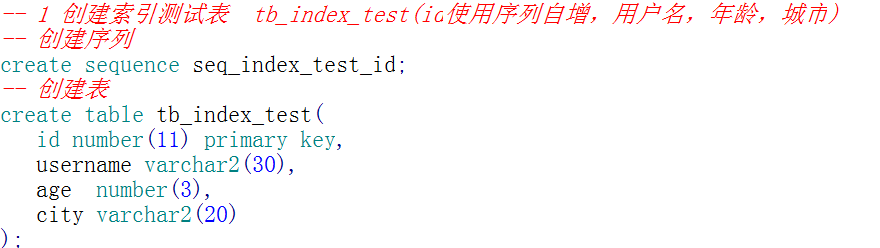
限制表中索引的数量（创建索引耗费时间，并且随数据量的增大而增大；索引会占用物理空间(牺牲空间换取时间做法)；当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度）这里需要权衡一个问题，建立索引的目的是为了提高查询效率的，但建立的索引过多，会影响插入、删除数据的速度，因为我们修改的表数据，索引也要跟着修改。这里需要权衡我们的操作是查询多还是修改多

把索引与对应的表放在不同的表空间。当读取一个表时表与索引是同时进行的。如果表与索引和在一个表空间里就会产生资源竞争，放在两个表这空就可并行执行。

建索引的时候应该根据具体的业务SQL来创建，特别是where条件，还有where条件的顺序(尽量将过滤大范围的放在后面，因为SQL执行是从后往前的)。

**3.1.4 索引的示例**

1，创建索引测试表 tb\_index\_test(id使用序列自增，用户名，年龄，城市)



2， 使用jdbc插入1000W条数数据（oracle）

创建项目index\_demo 引入jar包

<!--oracle驱动包-->

<dependency>

<groupId>oracle</groupId>

<artifactId>oracle-jdbc</artifactId>

<version>12.1.0.2</version>

</dependency>

直接引入dependency下不来包，使用安装命令安装(MAVEN已经安装成功并且配置成功 找到oracle的驱动包（F:\app\Administrator\product\11.1.0\db\_1\jdbc\lib\ojdbc6.jar ）拷到D盘下)

 mvn install:install-file -Dfile=D:\ojdbc6.jar -DgroupId=oracle -DartifactId=oracle-jdbc -Dversion=12.1.0.2  -Dpackaging=jar

实现代码：

package com.aaa.oraclejdbc.demo;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Random;

/\*\*

\* fileName:BatchAddTBIndexTest

\* description:

\* author:zz

\* createTime:2020/3/12 11:40

\* version:1.0.0

\*/

public class BatchAddTBIndexTest {

public static void main(String[] args) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

//加载驱动类 反射获取class对象一种方式

// oracle.jdbc.driver.OracleDriver

Class.forName("oracle.jdbc.driver.OracleDriver");

//DriverManager获取连接Connection

connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "scott",

"tiger");

//设置事务提交方式为手动提交 默认为true

connection.setAutoCommit(false);

//获取PreparedStament对象

preparedStatement = connection.prepareStatement(

"insert into tb\_index\_test values(seq\_index\_test\_id.nextval,?,?,?)");

//实例化城市数组

String[] cityArray = {"海口","香港","澳门","深圳","广州","南宁","桂林","贵州","武汉","郑州","黑龙江"};

//实例化一个随机数对象

Random random = new Random();

//使用该对象批量添加数据 插入100W条数据

for (int i = 1; i <=1000000; i++) {

System.out.println("当前添加"+i);

//设置不同的参数

preparedStatement.setString(1,"scott"+i);

// random.nextInt(120) 随机0-119之间任意一个数

preparedStatement.setInt(2,random.nextInt(120));

// cityArray.length 长度11 random.nextInt(11) 随机0-10之间的任意一个数 cityArray[2]=澳门

preparedStatement.setString(3,cityArray[random.nextInt(cityArray.length)]);

//先把数据放入批处理缓冲区

preparedStatement.addBatch();

//为了不影响数据库性能，分批提交

if(i%1000==0){ //每1000条提交一次到数据

preparedStatement.executeBatch();//执行批量操作

connection.commit();//持久化

preparedStatement.clearBatch();//清空已经批量缓存的语句

}

}

System.out.println("数据添加成功！！！！");

} catch (Exception e) {

e.printStackTrace();

System.out.println("数据添加失败！！！！");

}finally {

//关闭使用资源finally处理

try {

if(preparedStatement!=null){

preparedStatement.close();

}

if(connection!=null){

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

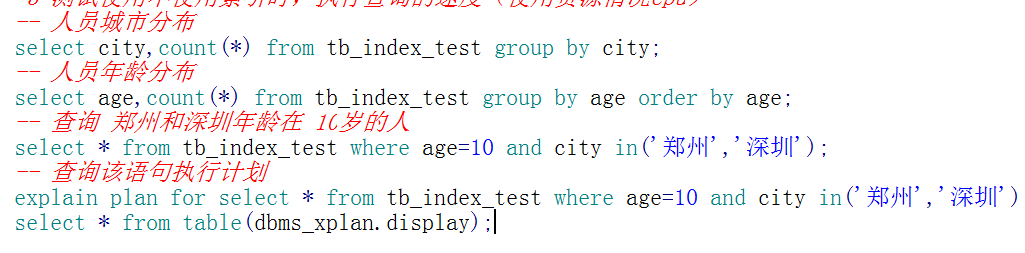
}

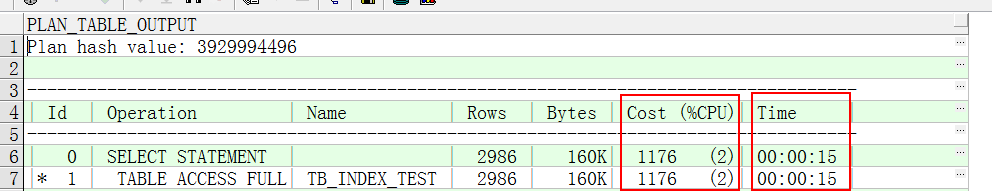
}

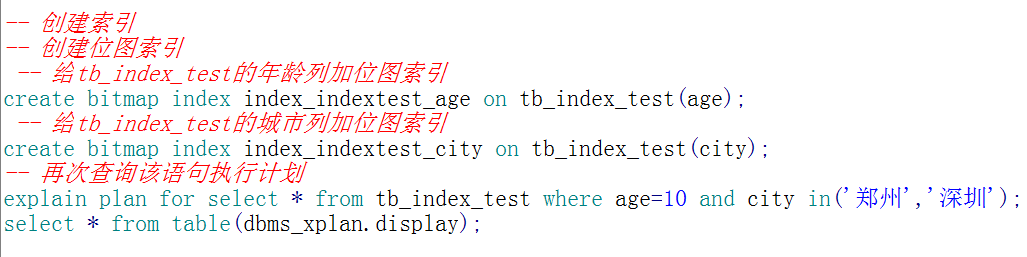
}

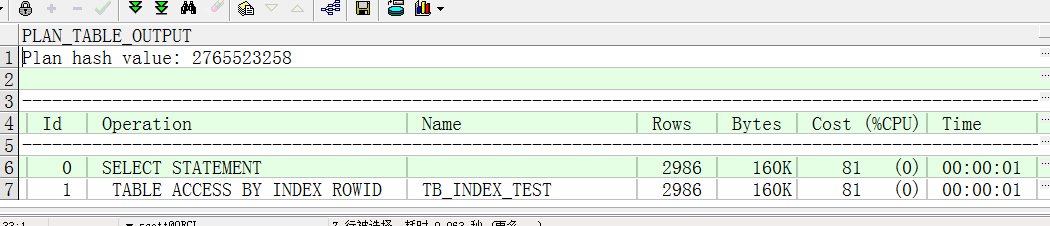
}

3，测试使用不使用索引时，执行查询的速度（使用资源情况cpu）









**3.2 sql语句优化**

对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

         select id from t where num is null

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

      select id from t where num=0

 应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将进行全表扫描。

  应尽量避免在 where 子句中使用 or 来连接条件，否则将导致全表扫描，如：

   select id from t where num=10 or num=20

   可以这样查询：

  select id from t where num=10

   union all

   select id from t where num=20

  in 和 not in 要慎用，否则会导致全表扫描，如：

   select id from t where num in(1,2,3)

   对于连续的数值，能用 between 就不要用 in 了：

   select id from t where num between 1 and 3

尽量避免在where子句中对字段进行函数操作，这将导致全表扫描。如：

   select id from t where substr(name,1,3)='abc'--name 以abc开头的id

   select \* from emp where add\_months(hiredate,120)<sysdate; 生成的id

  应改为:

   select id from t where name like 'abc%'

 select \* from emp where hiredate<to\_date('2010/3/13 10:06:33','yyyy-mm-dd hh24:mi:ss');

  全模糊查询也将导致全表扫描：

   select id from t where name like '%abc%'

   若要提高效率，可以考虑全文检索。

  应尽量避免在 where 子句中对字段进行表达式操作，这将进行全表扫描。如：

   select id from t where num/2=100

   应改为:

   select id from t where num=200

  很多时候用 exists 代替 in 是一个好的选择：

  select num from a where num in(select num from b)

   用下面的语句替换：

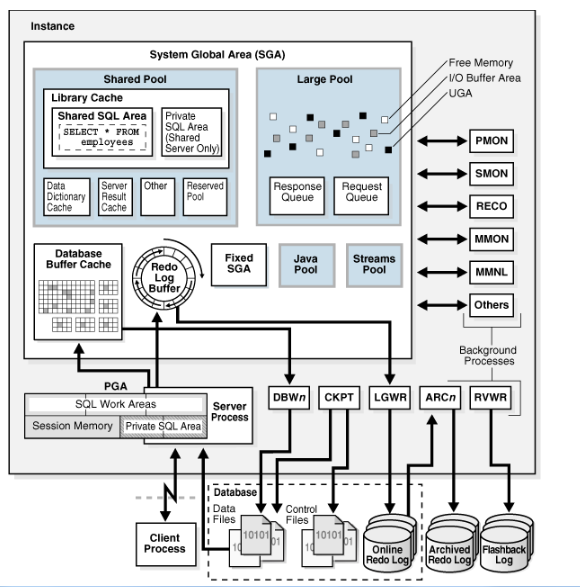
   select num from a where exists(select 1 from b where b.num=a.num)

  尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

  任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

**3.3 SGA PGA**

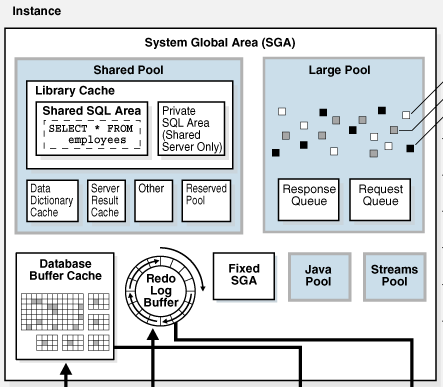
**3.3.1 数据库架构(官网:**<https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT914>**)**



SGA的右侧是后台进程PMON，SMON，RECO，MMON，MMNL等。在SGA之下是DBWn，CKPT，LGWR，ARCn和RVWR。SGA下方是PGA和服务器进程。服务器进程已连接到客户端进程。客户端进程右侧是数据库文件（数据文件，控制文件，联机重做日志），存档重做日志和闪回日志 。

**3.3.2 系统全局区域（SGA** System Global Area**）**

SGA是一组共享内存结构，其中包含一个数据库实例的数据和控制信息。SGA组件的示例包括缓存的数据块和共享的SQL区域。



该图显示了SGA。在SGA中是数据库缓冲区高速缓存，重做日志缓冲区，共享池，大型池，固定SGA，Java池和流池。

Database buffer cache：用于缓存从数据文件中检索出来的数据块，可以大大提高查询和更新数据的性能。提取数据时，服务器进程先在数据库高速缓冲区中搜寻所需数据，找到就直接使用，不进行磁盘操作；没找到就将进行磁盘操作，把数据文件中的数据读入到数据库高速缓冲区。Oracle采用LRU(least recently used)的队列(list)或算法(algorithm)来实现对数据库高速缓冲区的管理

查看数据库数据缓冲区的使用情况：

select name,value from v$sysstat where name in ('db block gets','consistent gets','physical reads');

sysstat  数据库启动后不断收集和统计数据存取的情况，并将其存放在动态性能视图V$SYSSTAT中

dbblock gets该统计值为数据请求总数。

consistent gets该统计值为对内存缓冲区存取技能满足的请求数。

physical reads该统计值为磁盘文件存取的总数。

计算出来数据缓冲区的使用命中率：

  命中率=1 —（physical reads / （db block gets + consistent gets））

这个命中率应该在90％以上，否则需要增加数据缓冲区的大小。

查看和改变数据缓冲区大小Sql代码:show parameter db\_cache\_size;

alter system set db\_cache\_size=52M;

Redo log buffer：缓存了写到磁盘之前的重做信息。为了提高日志文件的操做效率，Oracle数据库会在日志缓冲区中先记录日志，之后在特定的时间点，后台进程会批量写入到日志文件中.Oracle在使用任何DML或DDL操作改变数据(写入数据库高速缓冲区)之前都先写redo log。

1、服务进程从用户空间拷贝每条DML/DDL语句的redo条目到redo log buffer中。

2、redo log buffer是一个可以循环使用的buffer，服务进程拷贝新的redo覆盖掉redo log buffer中已通过LGWR写入磁盘（online redo log）的条目。

3、导致LGWR执行写redo log buffer到online redo log的条件

a.用户执行事务提交commit

b.每3秒钟或redo log buffer内已达到1/3满或包含1MB数据

c.DBWn进程将修改的缓冲区写入磁盘时（如果相应的重做日志数据尚未写入磁盘）

查看日志缓冲区的使用情况

SELECT name,value FROM v$sysstat WHERE name IN('redo entries','redo log space requests')

查询出的结果可以计算出日志缓冲区的申请失败率：

申请失败率＝requests/entries，申请失败率应该接近于0，否则说明日志缓冲区开设太小，需要增加ORACLE数据库的日志缓冲区。

Shared pool：缓存了各用户间可共享的各种结构。用于缓存最近被执行的SQL语句和最近被使用的数据定义

Library Cache又包含共享SQL区与PL/SQL区：

a).共享SQL区保存了分析与编译过的SQL语句。

查看共享SQL区的使用率：select (sum(pins-reloads))/sum(pins) "使用率" from v$librarycache;

    Librarycache查询实例启动后所有库高速缓存的活动

pins  用于显示在库高速缓存中执行的次数或者请求数

reloads  用于显示在执行阶段库高速缓存不命中的次数

命中率应该在90％以上，否则需要增加共享池的大小。

查看和改变共享池大小Sql代码(命令窗口): show parameter shared\_pool\_size;

alter system set shared\_pool\_size=52M;

b).PL/SQL区保存了分析与编译过的PL/SQL块（过程和函数、包、触发器与匿名PL/SQL块）。

Data Dictionary Cache

保存了数据字典对象的定义。当Oracle执行sql语句时，服务器进程将数据文件，表，索引，列，用户和其他数据对象的定义和权限信息放入数据字典高速缓存。

查看数据字典缓冲区的使用率：

select(sum(gets-getmisses-usage-fixed))/sum(gets)"数据字典使用率"from v$rowcache;

rowcache  动态性能视图,借助它来检查、调整数据字典高速缓存的活动情况

gets 用于显示数据字典请求总数

getmisses  用于显示不命中的请求数

usage  用于显示有效数据的缓存项数目

fixed     用于显示固定的缓存项数目

这个使用率也应该在90％以上，否则需要增加共享池的大小。

Large pool：一个可选的区域，用来缓存大的I/O请求，以支持并行查询、共享服务器模式以及某些备份操作。

a.为I/O服务进程分配内存

b.为备份与恢复操作分配内存

c.为Oracle共享服务器模式与多个数据库间的联机事务分配内存。

通过从large pool中为共享服务器模式分配会话内存，可以减少share pool因频繁为大对象分配和回收内存而产生的碎片。将大的对象从share pool中分离出来，可以提高shared pool的使用效率，使其可以为新的请求提供服务或者根据需要保留现有的数据。

Java pool：保存java虚拟机中特定会话的数据与java代码。

1、JAVA\_POOL\_SIZE

　　通过JAVA\_POOL\_SIZE参数指定java pool大小。

保存了jvm中特定会话的java code和数据。

2、在编译数据库中的java代码和使用数据库中的java资源对象时，都会用到share pool。

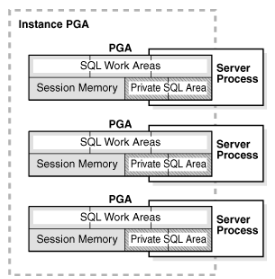
java的类加载程序对每个加载的类会使用大约8K的空间。

系统跟踪运行过程中，动态加载的java类，也会使用到share pool。

Streams pool：由Oracle streams使用。

**3.3.3 程序全局区域（PGA** Program Global Area**）**

对于与实例的每个用户连接，应用程序由客户端进程运行。每个客户端进程都与自己的服务器进程相关联。服务器进程具有自己的专用会话内存，称为程序全局区域（PGA）。



每个PGA都包含以下内容：SQL工作区，会话内存和专用SQL区域。

Sql Work Areas

PGA的一大部分被分配给Work Area，用来执行如下操作：

a.基于操作符的排序，group by、order by、rollup和窗口函数。

  　　　　参数为sort\_area\_size

　　　　b.hash散列连接，

  　　　　参数为hash\_area\_size

　　　　c.位图合并，

  　　　　参数为bitmap\_merge\_area\_size

　　　　d.位图创建，

  　　　　参数为create\_bitmap\_area\_size

　　　　e.批量装载操作使用的写缓存

**Private SQL Area**

1、保存了当前会话的绑定信息以及运行时内存结构。

2、每个执行sql语句的会话，都有一个private sql area。

3、当多个用户执行相同的sql语句，此sql语句保存在一个称为shared sql area。此share sql area被指定给这些用户的private sql area

4、共享服务器模式：private sql area位于SGA的share pool或large pool中

     专用服务器模式：private sql area位于PGA中

**Session memory**

   保存了会话的变量，如登录信息及其他与会话相关的信息，共享服务器模式下，Session memory是共享的。

**4、本章总结**

**4.1 总结本章知识点**

**4.2 面试题答案**

**4.3 预习下一章重点**