boss說一共6道題目，順序不定，很簡單。

1.為性能而設計：a.沒有好的設計 b.數據庫之間的差別很大 c.知道一些提高數據庫性能的方法（比如異步和同步）好和坏的方面

2.sql優化：買BMW的優化(仔細看那道BMW的題，有類似的題，估計只會改一點點)

3.為什麼要使用索引，為什麼沒有使用索引，舉例

4.物理組織的優化（如何訪問堆文件，過程是什麼），堆文件和IOT，分區的特性

5.處理層次結構，數據庫設計（3種模型：臨界，物化路徑，嵌套集合：比較優劣和適合的場景）

6.ER模型到表結構是一個完整的映射，1對1（部分參與），1對多（部分參與），多對多。

本次考试满分是120分，其中20分是附加题，并不要求完成。

复习要点：

1、在PPT上没有某些考试题目的答案；

2、注意复习数据库中讲授过的基本文件的结构和读取方式；

3、注意复习数据库中讲授过的基本索引的结构和读取方式；

4、注意复习较为复杂的ORM（对象关系映射）或ER-表结构的映射规则

1．为性能而设计

◦ 在技术挑战的刺激下

◦ 忽视目标◦ 重视手段

◦ 忽视数据的质量◦ 重视按期交付功能

(1)  数据结构的设计。

对数据表进行规范化，可以保持适当的冗余量，建立有用的索引，尽量减少表与表之间的关联，运用存贮过程等。

(2)  处理流程

采用异步处理可以允许用户发出操作之后去做其他工作，数据库进行集中的批处理，当处理完毕后通知用户。

采用同步处理时用户必须等待数据库操作并返回结果，期间不能做其它操作。适合数据库进行实时交易。

要根据系统的需求选择一种合适的手段，这会影响物理结构的设计。

(3)  数据集中化

分布式数据存储具有更灵活的体系结构，但是有缺点：

                                    远程数据的透明引用访问代价很高。

                                    不同的数据源数据结合极为困难。

                                    存取结构复杂，保密性不易控制。

因此将数据集中化会提高数据库速度，并且在部署上服务离数据越近，速度会越快。

离数据越近，访问速度越快。代码里DBMS核心越近运行越快,充分利用每次数据库访问。Block使用充分，命中率高，所需io操作少

(4)  保持数据库连接的稳定。链接稳定减少交互。

(5)  充分利用每次的数据库访问，减少服务与数据库之间的交互。

(6)  优化SQL语句

把逻辑放入到查询SQL中而不是SQL宿主语言中；优化SQL语句的结构。

7．谨慎地使用自定义函数，不当的位置会使自定义函数执行次数过多而造成性能下降；

(7)  构建稳定的应用需要防御式编程，但在合理的情况下可以使用进攻式编程。

如通过left join返回的各种情况做判断（检查那个字段错误）

限用boolean型字段

约束应明确说明：数据中存在隐含约束是一种不良设计

2.sql優化：

(1)  注意在做否定意义的查询是小心进入陷阱：

如，没有选修‘B2’课程的学生 ：

　　select students.\*

　　from students, grades

　　where students.sno=grades.sno

　　AND grades.cno <> ’B2’

　　上面的查询方式是错误的，正确方式见下方：

　　select \* from students

　　where not exists (select \* from grades

where grades.sno=students.sno AND cno='B2')

(2)

* JOIN: 如果表中有至少一个匹配，则返回行（INNER JOIN）
* LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行
* RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行
* FULL JOIN: 只要其中一个表中存在匹配，就返回行

(3)                  oracle里的rownum

不应该

select empname, salary

from employees

where status != 'EXECUTIVE'

and rownum<= 5

                                    order by salary desc

而应该

select \*

from (select empname, salary

from employees

where status != 'EXECUTIVE'

* order by salary desc)

                                    where rownum<= 5

**SQL优化**

(1)  在关系操作层完成尽量多的工作，对于不完全的关系操作，加倍留意查询的编写。

(2)  标准sql执行顺序：

1. select  2. from 3. where  4. group by  5. order by

4 1 2 3 5

(3) 避免在高层distinct，在实现 DISTINCT的过程中，同样也是需要分组的，然后再从每组数据中取出一条返回给客户端。

(4) IN 先产生中间结果集；EXIST相当于循环进行子查询

使用依据：如果子查询结果集较大则用EXIST，否则用IN。原则就是IN是否能过滤掉较多的数据。

1. **1.  例子：BMW  
   关于DISTINCT，嵌套 SQL，EXISTS 和 IN 的差异**

优化器借助关系理论提供的语义无误的原始查询进行有效的等价变换，在数据处理真正被执行的时候发生。

优化器的有效范围：

优化器需要借助数据库中找到的信息

能够进行数学意义上的等价变换

优化器考虑整体响应时间

优化器改善的是独立的查询

（如果是若干小查询，会各个优化；如果是一个大查询，会整体优化）

select distinct c.custname

from customers c

join orders o

on o.custid= c.custid

join orderdetailod

on od.ordid= o.ordid

join articles a

on a.artid= od.artid

where c.city= ‘Nanjing'

and a.artname= ‘BMW'

and o.ordered>= somefunc/\*函数，返回六个月前的具体日期\*/

古老的自然连接方式：

select distinct c.custname

from customers c,

orders o,

orderdetailod,

articles a

where c.city= ‘Nanjing'

and c.custid= o.custid

and o.ordid= od.ordid

and od.artid= a.artid

and a.artname= ‘BMW'

and o.ordered>= somefunc

摆脱distinct 关联子查询：o.custid需要有索引

select c.custname

from customers c

where c.city= ‘Nanjing'

and exists(select null

from orders o,

orderdetail od,

articles a

where a.artname= ‘BMW'

and a.artid= od.artid

and od.ordid= o.ordid

**and o.custid= c.custid**

and o.ordered>= somefunc)

非关联子查询：内层查询不在依赖外层查询，只需要执行一次；o.custid不需要有索引

select custname

from customers

where city = ‘Nanjing'

and custid in (select o.custid

from orders o,

orderdetail od,

articles a

where a.artname= ‘BMW'

and a.artid= od.artid

and od.ordid= o.ordid

and o.ordered>= somefunc)

exits适合内小外大的查询，in适合内大外小的查询

select custname from custom c where city = naj and custid in( select o.cust.id from rders 0 where ordered >=func orderid in(selct od.artid from od o a where ) )

SQL优化：

1)   避免在最高层使用distinct

2)   针对分辨率最强的条件使用exist或in

3)   尽快剔除不需要的数据，查询后续阶段必须处理的数据量就越少，查询效率就越高

4)   所有影响聚合函数结果的条件都应在having子句中

5)   任何无关聚合条件都应该放在where子句中

6)   减少group by必须执行排序操作处理的数据量

select c.custname, o.ordid, os.status, os.statusdate from customers c,

orders o,

orderstatus os where o.ordid = os.ordid

and not exists (select null from orderstatus os2

where os2.status = 'COMPLETE' and os2.ordid = o.ordid) and os.statusdate = (select max(statusdate)

from orderstatus os3

where os3.ordid = o.ordid) and o.custid = c.custid

select c.custname, o.ordid, os.status, os.statusdate from customers c,

orders o, orderstatus os, (select ordid, max(statusdate) laststatusdate

from orderstatus group by ordid) x

where o.ordid = os.ordid and os.statusdate = x.laststatusdate and os.ordid = x.ordid and os.status != 'COMPLETE' and o.custid = c.custid

**索引**

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。

索引分为**聚簇索引**和**非聚簇索引**两种，聚簇索引 是按照数据存放的物理位置为顺序的，而非聚簇索引就不一样了；聚簇索引能提高多行检索的速度，而非聚簇索引对于单行的检索很快。

3.為什麼要使用索引，為什麼沒有使用索引，舉例

**索引。**

**目的：提高查询效率。   
结构：B 树**

什么时候使用B树索引：

仅当要通过索引访问表中很少一部分行

如果要处理表中多行，而且可以使用索引而不用表

索引的5种优点

**索引优点**

第一，   通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，   可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，   可以加速表 和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，   在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，   通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

**为什么不为每一列建立索引**

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物 理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，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，这样就降低了数据的维护速 度。

**什么时候应该使用B树索引**

◦仅当要通过索引访问表中很少一部分行

◦如果要处理表中多行，而且可以使用索引而不用表

**这些列应该建立索引**

在经常需要搜索的列上，可以加快搜索的速度；

在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；外键建索引由于连接加快还会减少死锁几率。

在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

**这些列不应该建立索引**

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

　　第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

　　第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少,不利于使用索引。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

**为什么没有使用我的索引？(ppt)（失效情况）**

情况1：我们在使用B+树索引，而且谓词中没有使用索引的最前列

◦T，T(X,Y)上有索引，做SELECT \* FROM T WHERE Y=5

跳跃式索引（仅CBO）

情况2：使用SELECT COUNT(\*) FROM T，而且T上有索引，但是优化器仍然全表扫描

不带任何条件的count会引起全表扫描。

情况3：对于一个有索引的列作出函数查询

◦Select \* from t where f(indexed\_col) = value

情况4：隐形函数查询

不等于符”<>”会限制索引，引起全表扫描，如果改成or就可以使用索引了。

is null查询条件也会屏蔽索引。

情况5：此时如果用了索引，实际反而会更慢

数据量本来不够大  
oracle自己计算后认为不用索引更合算

则CBO不会选择用索引

情况6：没有正确的统计信息，造成CBO无法做出正确的选择

表分析就是收集表和索引的信息，生成的统计信息会存在user\_tables这个视图。CBO根据这些信息决定SQL最佳的执行路径。

其他：

1. 对于两个公有一字段的表，如果在做外表的表上对该字段建立索引，则该索引不会被使用因为外表的数据访问方式是全表扫描。

2. 查询使用了两个条件用or连接，如果条件1中的字段有索引而条件2中字段没有，则仍会全表扫描。

5．物理組織的優化（如何訪問堆文件，過程是什麼），堆文件和IOT，分區的特性

基本文件的结构和读取方式**堆文件**

文件结构：堆文件就是一般的表，使用“heap”的结构，数据没有特定的顺序;

读取/访问方式：获取表中的数据是按命中率来得到的。没有明确的先后之分，在进行全表扫描时，并不是先插入的数据就先获取。数据的存放是随机的，也可以根据可用空闲的空间来决定。

**IOT 索引组织表**

当索引中增加额外的字段（一个或多个，它们本身与实际搜索条件无关，但包含查询所需的数据），能提高某个频繁运行的查询的速度。

特性：对IOT表插入的效率也许低于堆文件

IOT最大优点：记录是排序的

文件结构：iot 就是类似一个全是索引的表，表中的所有字段都放在索引上，允许在主键索引中存储表中所有数据，表就是索引，所以就等于是约定了数据存放的时候按照严格规定，

提高效率方式：在数据插入以前其实就已经确定了其位置，所以不管插入的先后顺序，它在那个物理上的那个位置与插入的先后顺序无关。这样在进行查询的时候就可以少访问很多blocks，但是插入的时候，速度就比普通的表要慢一些。  
适用于信息检索、空间和OLAP程序。索引组织表的适用情况：  
1、 代码查找表。  
2、 经常通过主码访问的表。  
3、 构建自己的索引结构。  
4、 加强数据的共同定位，要数据按特定顺序物理存储。  
5、 经常用between…and…对主码或唯一码进行查询。数据物理上分类查询。如一张订单表，按日期装载数据，想查单个客户不同时期的订货和统计情况。  
经常更新的表当然不适合iot，因为oracle需要不断维护索引，而且由于字段多索引成本就大。**分区**

特性：

分区能够提高并发性和并行性

从而增强系统架构的可伸缩性

**问题：它们分别是如何提高查询效率的（考试可能考其中的一个）**

**分区表：**

当表中的数据量不断增大，查询数据的速度就会变慢，应用程序的性能就会下降，这时就应该考虑对表进行分区。表进行分区后，逻辑上表仍然是一张完整的表，只是将表中的数据在物理上存放到多个表空间(物理文件上)，

提升效率方法：这样查询数据时，不至于每次都扫描整张表。

(

   表空间：

    是一个或多个数据文件的集合，所有的数据对象都存放在指定的表空间中，但主要存放的是表， 所以称作表空间。

)

**分区的实现方式**

◦哈希分区（Hash-partitioning）

按散列决定记录分在哪个分区

◦范围分区（Range-partitioning）

范围分区将数据基于范围映射到每一个分区，这个范围是你在创建分区时指定的分区键决定的。这种分区方式是最为常用的

◦列表分区（List-partitioning）

该分区的特点是某列的值只有几个，基于这样的特点我们可以采用列表分区。每个分区的该列都是某一个值。

按什么字段进行分区要整体考虑，因为：

更新分区键会引起移动数据，应该避免这么做。

**表分区的优缺点**    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。

除了堆文件之外的任何存储方法，都会带来复杂性

选错存储方式会带来大幅度的性能降低

总结

◦A. 测试，测试，测试

◦B. 设计是最重要的

◦C. 任何设计都有时效性

5.處理層次結構，數據庫設計（3種模型：臨界，物化路徑，嵌套集合：比較優劣和適合的場景）

表结构设计。  
例子：如何把一棵树存到二维表中？  
一，<id, pid>方式；（邻接模型）二，物化路径；三，嵌套集合

问题：这三种方式的性能区别

邻接模型：id，parent\_id(指向上级)         自顶向下查询，递归实现。Connect by相当容易实现

物化模型：物化路径 计算由路径导出的层次不方便。仅找出适当的记录并缩排显示算容易，重复记录问题，顺序问题

嵌套集合模型：left\_path,right\_path 某节点后代的left\_num和right\_num都会在该节点的left\_num和right\_num范围内

例：user与order是主从表

树状结构的保存只需一个表。表与它本身之间有种主/从关系，不是两个类型不同的表的关系。  
与根节点的距离是层次结构的重要信息。主从关系中，不是主表、就是明细表。  
主/从关系中，可以有明确的外键完整性约束。  
多重父节点。数有两种实体类型，一个是节点，另一个是节点之间的连结。

**在数据库设计中，树通常三种模型**

◦Adjacency model-邻接模型

特性：邻接模式使用由下向上的记录方式，也就是在数据模型中每个节点会记录其父节点的信息

插入、移动、删除节点快捷；Connect by相当容易实现；删除子树难

◦Materialized path model-物化路径模型

特性：对树中的每个节点，都要记录其在树中的位置信息 。

计算由路径导出的层次不方便。仅找出适当的记录并缩排显示算容易，重复记录问题，顺序问题

注意问题：

物化路径不该是KEY，即使他们有唯一性

物化路径不该暗示任何兄弟节点的排序

所选择的编码方式不需要完全中立  
  
id name path   
  
1 oracle 1   
  
2 bin 1.1

◦Nested set model-嵌套集合模型

数据元素之间不再是点和线的关系，而是以容器和被容纳的方式

具体：每个节点被赋予一对数字，父节点的两个数字定义的间隔总是将其所有子孙所定义的间隔包含在其中。此模型思想是以两个数字为特定节点的路径编码，解释成有理 数（即分数）的分子和分母。计算量大，对存储程序要求高。它是基于指针的解决方案，设计关系方法的目标正是要逃离指针的沼泽。

某节点的后代的left\_num和right\_num都会在 该节点的left\_num和right\_num范围内

简单总结：

oracle为邻接模型提供了connect 语法，但删除子树难；物化路径模型是罪简单，操作最好的；而嵌套模型好理解，但操作难，只适合‘深度优先’遍历

效率总结：

自顶向下：邻间>物化路径>嵌套集合

自底向上：邻间>物化路径>嵌套集合