# 为性能而设计

## 现状：

在技术挑战的刺激下，忽视目标，重视手段，忽视数据质量，重视按期交付的功能。

## 提高性能的方法：

（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，运用存储过程等。

（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，期间不能做其他操作，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，这回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熟读，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的访问。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。连接稳定减少交互

（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）谨慎的使用自定义函数

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

（8）构建稳定的应用需要防御是编程，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（检查哪个字段错误）

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

## 提高性能的原则

（1）减少数据访问（减少磁盘访问）

a.创建并使用正确的索引

建立索引通常应该考虑：

索引对于查询的效率提高多少

索引对DML(INSERT,UPDATE,DELETE)附加的开销有多少

索引应该建在哪些列上

哪些情况能用到索引

b.只通过索引访问数据

需要检索的数据全部在索引中

c.优化SQL执行计划

设计适合业务逻辑的良好SQL（表数量，join数量）

d.代码中使用函数始终应该首选DBMS自带函数，代码的执行越接近DBMS核心，则代码执行速度越快

e.慎用自定义函数。（自定义函数中可能隐藏有查询，阻碍了基于代价的优化器（CBO）对整个查询的优化效果，子查询隐藏在函数中，优化器“鞭长莫及”）

（2）返回更少数据（减少网络传输或磁盘访问）（不需要的数据不要取出来）

a.数据分页处理（限制每次取出数据的数量）

客户端（应用程序或浏览器）分页

应用服务器分页

数据库SQL分页

b.只返回需要的字段

通过去除不必要的返回字段可以提高性能

减少数据在网络上传输开销

减少服务器数据处理开销

减少客户端内存占用

字段变更时提前发现问题，减少程序bug

如果访问的所有字段刚好在一个索引里面，则可以使用纯索引访问提高性能

（3）减少交互次数（减少网络传输）（需要的数据以尽量少的交互次数全部取出）

a.批量处理数据，每次数据库连接应该完成尽可能多的工作，充分利用每次数据库访问

b.使用INList 代替多个 where，减少SQL查询的次数

c.设置每次返回结果的Fetch Size

可以加大fetch size,这样可以减少结果数据传输的交互次数及服务器数据准备时间，提高性能

d.使用存储过程

大型数据库一般都支持存储过程，合理的利用存储过程也可以提高系统性能。

（各个DBMS内部变成方式有较大差异，存储过程移植性不好，存储过程中会包含一定的业务逻辑，使得系统维护和管理更加困难，所以普通业务逻辑尽量不要使用存储过程）

（4）减少服务器CPU开销（减少CPU及内存开销）

a.使用绑定变量

绑定变量是指SQL中对编号的值采用变量参数的形式提交，而不是在SQL中直接拼写对应的值（java里面的preparedStatement）

b.合理使用排序

你需要注意是否一定要这么做了，大记录集排序不仅增加了CPU开销，而且可能会由于内存不足发生硬盘排序的现象，当发生硬盘排序时性能会急剧下降。

c.减少模糊查找，例如like

d.不要把太多的复杂计算交给数据库

大量复杂运算在数据库里不利于数据库高并发处理

（5）优化数据库物理存储及其环境

a.选择性能优良的数据库服务器

b.数据库物理存储的考虑

客户离数据越近，访问速度越快

数据分散磁盘存储，减小单个磁盘的访问压力，增加数据库对于并发的处理能力

c.数据表的分区

## 各个DB的比较

1.MySQL

MySQL是最受欢迎的开源SQL数据库管理系统，

MySQL是一个快速的、多线程、多用户和健壮的SQL数据库服务器。MySQL服务器支持关键任务、重负载生产系统的使用，也可以将它嵌入到一个大配置(mass- deployed)的软件中去。

与其他数据库管理系统相比，MySQL具有以下优势：

(1)MySQL是一个**关系数据库管理系统**。

(2)MySQL是开源的。

(3)MySQL服务器是一个快速的、可靠的和易于使用的数据库服务器。

(4)MySQL服务器工作在客户/服务器或嵌入系统中。

(5)有大量的MySQL软件可以使用。

2.SQL Server

SQL Server是由微软开发的数据库管理系统，是Web上最流行的用于存储数据的数据库，它已广泛用于电子商务、银行、保险、电力等与数据库有关的行业。

目前最新版本是SQL Server 2005，它只能在Windows上运行，操作系统的系统稳定性对数据库十分重要。并行实施和共存模型并不成熟，很难处理日益增多的用户数和数据卷，伸缩性有限。

SQL Server 提供了众多的Web和电子商务功能，如对XML和Internet标准的丰富支持，通过Web对数据进行轻松安全的访问，具有强大的、灵活的、基于Web的和安全的应用程序管理等。而且，由于其易操作性及其友好的操作界面，深受广大用户的喜爱。

3.Oracle

最新的Oracle 8主要增加了对象技术，成为关系—对象数据库系统。目前，Oracle产品覆盖了大、中、小型机等几十种机型，Oracle数据库成为世界上使用最广泛的关系数据系统之一。

Oracle数据库产品具有以下优良特性。

(1)兼容性

Oracle产品采用标准SQL，并经过美国国家标准技术所(NIST)测试。与IBM SQL/DS、DB2、INGRES、IDMS/R等兼容。

(2)可移植性

Oracle的产品可运行于很宽范围的硬件与操作系统平台上。可以安装在70种以上不同的大、中、小型机上;可在VMS、DOS、UNIX、Windows等多种操作系统下工作。

(3)可联结性

Oracle能与多种通讯网络相连，支持各种协议(TCP/IP、DECnet、LU6.2等)。

(4)高生产率

Oracle产品提供了多种开发工具，能极大地方便用户进行进一步的开发。

(5)开放性

Oracle良好的兼容性、可移植性、可连接性和高生产率使Oracle RDBMS具有良好的开放性。

4.Sybase

Sybase主要有三种版本：一是UNIX操作系统下运行的版本; 二是Novell Netware环境下运行的版本;三是Windows NT环境下运行的版本。对UNIX操作系统，目前应用最广泛的是SYBASE 10及SYABSE 11 for SCO UNIX。

Sybase数据库的特点：

(1)它是基于客户/服务器体系结构的数据库。

(2)它是真正开放的数据库。

(3)它是一种高性能的数据库。

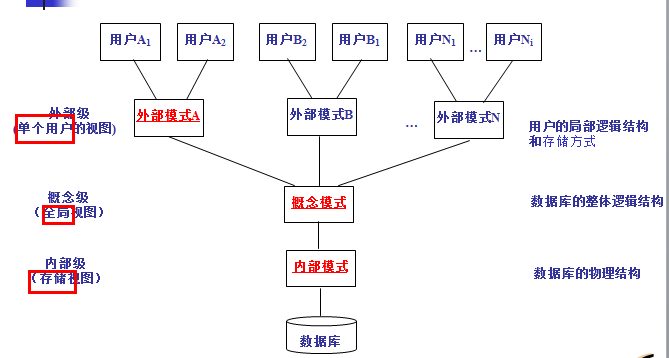
5.DB2

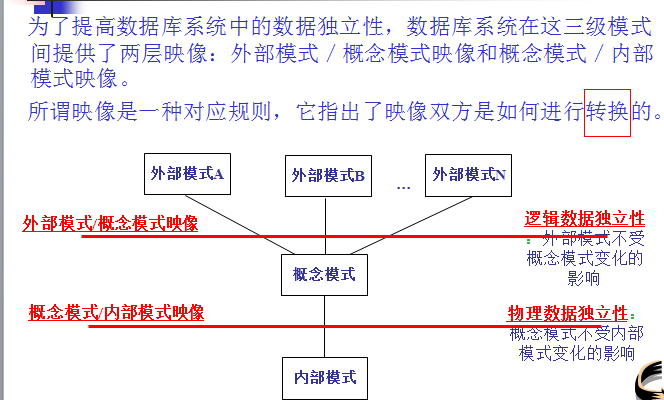
DB2是内嵌于IBM的AS/400系统上的数据库管理系统，直接由硬件支持。它支持标准的SQL语言，具有与异种数据库相连的GATEWAY。因此它具有速度快、可靠性好的优点。但是，只有硬件平台选择了IBM的AS/400，才能选择使用DB2数据库管理系统。

DB2能在所有主流平台上运行(包括Windows)，最适于海量数据。

DB2在企业级的应用最为广泛，在全球的500家最大的企业中，几乎85%以上都用DB2数据库服务器，而国内到1997年约占5%。

## DB的体系结构





# SQL优化：

买BMW的优化（仔细看那道BMW的题，有类似的题，估计只会改一点点）

关于DISTINCT，嵌套 SQL，EXISTS 和 IN 的差异

（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’课程的学说：

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，也从坐标返回所有的行

RIGHT JOIN：即使坐标中没有匹配，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（41235）

（3）避免在高层distinct，在实现distinct的过程中，同样也是需要分组的，然后再从魅族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

IN：先进行内层子查询，再进行外层查询（有一个中间结果集，外层查询的时候使用中间结果集进行查询）。内外层查询间无关联，内层查询过程不使用外层的变量。

EXIST：先通过外层查询条件进行过滤，再进行内层子查询，内外层查询之间有关联。一般内层中会使用到外层的变量。

选择两者的原则：尽量在查询初期过滤掉更多的数据（IN首先通过子查询条件过滤，EXIST首先通过外层查询条件过滤）。

**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.aritd

and a.artname = ‘BMW’

and o.ordered >= somefunc

摆脱dixtinct 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 >= somfunc)

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 = ‘Nanjing’ and custid in(select o.custid

from orders o

where ordered >= somefunc

ordered in(select od.artid

from orderdetail od,article 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）尽快提出不需要的数据，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。

**索引目的：提高查询效率**

结构：B树

什么时候使用B数索引：

仅当要通过索引访问表中很少一部分行，比如处理表中多行，可以使用索引而不用表

**索引的几个关键点：**

使用索引可能可以提高查询效率，但是只是可能，没有适当设计的索引可能会降低 查询速度，有时使用所以带来的维护代价可能高于使用它因为加快查询获得的好处

索引是一种以原子粒度访问数据的手段，而不是为了检索大量数据的

在某些列上使用索引可能会导致其他列的查询变慢

系统地对表的外键加上索引的做法非常普遍，提高并发

为每个外键建立索引，可能会导致多余索引

**索引的5种优点：**

1.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

2.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

3.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

4.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

5.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。

**为什么不每一列建立索引**

1.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

2.索引需要占物理空间，除了数据表占数据空间之外，每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。

3.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

**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，其制定的范围 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，加快调价的判断速度

**这些列不应该建立索引：**

1.对于那些在查询中很少使用或者参与的列不应该创建索引。这事因为，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。

2.对于那些只有很少数据值的列也不应该增加索引。这事因为，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。

3.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。

4.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）**

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。

总结：归根到底，不适用索引的通常原因就是不能使用索引，使用索引会返回不正确的 结果，或者，不该使用索引，如果使用了索引就会变得更慢。

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。

# 物理组织的优化

物理组织的优化（如何访问堆文件，过程是什么），堆文件和IOT，分区的特性

## 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。

特性？如何提高查询效率？

## 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速度。

特性：对IOT表插入的效率也许低于堆文件

IOT的用途：全索引表、代码查找表、高频度的一组关联数据查询

IOT最大优点：记录是排序的

任何有序数据便于某些处理的同时，必将对其他处理不利

可能会造成有些列的查询效率降低。例如table(a,b,c)是ITO，且b跟a的相关度 不高，则可能出现对b的查询效率变低的情况

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， 它再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、空间的OLAP程序。索引组织表的使用情况：

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，从而曾倩系统架构的可伸缩性

问题：分区是如何提高查询效率的？

分区技术用的好的话可以提高性能，是因为一方面分区把一大块数据分成了n小块，这样查询的时候很快定位到某一小块上，在小块中寻址要快很多；另一方面CPU比磁盘IO快很多倍，而硬件上又有多个磁盘，或者是RAID（廉价磁盘冗余阵列），可以让数据库驱动CPU同时去读写不同的磁盘，这样才有可能可以提高效率。

分区表：

当表中数据量不断增大，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制定的表空间中，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）将数据基于范围映射到每一个分区，这个范围是指你在 创建分区时制定的分区键决定的。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**

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IO：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

# 处理层次结构，数据库设计

处理层次结构，数据库设计（3种模型：临界，物化路径，嵌套集合：比较优劣，适合场景）

表结构设计

**问题：这三种方式的性能区别？**

邻接模型：id,parent\_id（指向上机）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，计算由路径倒车不的层次不方便。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。树有两种实体类型，一个是节点，另一个是节点时间的连结。

在数据库设计中，树通常三种模型

**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

**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

**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：**

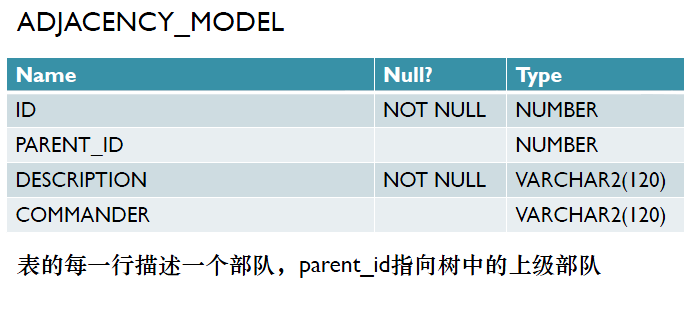
自顶向下：邻间>物化路径>嵌套集合

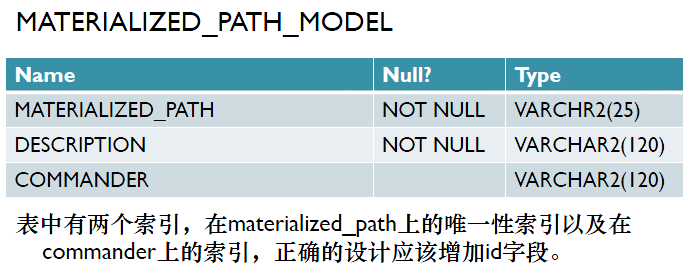
自底向上：邻间>物化路径>嵌套集合

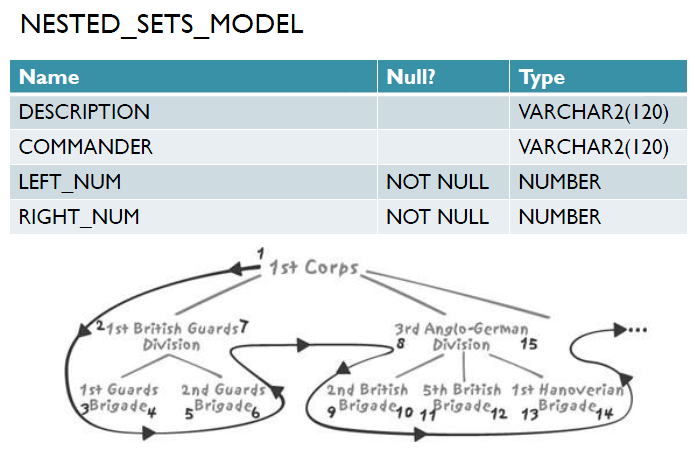
**例子：如何把一棵树存到二维表中？**

**邻接模型**

表的每一行描述一个部队，parent\_id指向树中的上级部队

**物化路径模型**

**嵌套集合模型**



**1.<id, pid>方式；1.物化路径；3.嵌套集合 问题：这三种方式的性能区别**

1. **自顶向下（Vandamme查询）**

邻接模式:递归实现，效率较低，使用CONNECT BY实现

|  |
| --- |
|  |

使用递归实现

|  |
| --- |
|  |

物化路径模型：path中前半段相同的即为结果，假设mp\_depth()函数返回当前节点深度

|  |
| --- |
|  |

嵌套集合模型：直接找到在left\_num和right\_num中间的所有节点，缩进比较难处理。

|  |
| --- |
|  |

**自顶向下效率：嵌套集合<物化路径<邻间模型**

**二、 自底向上（Highland查询）**

1. **邻间模型:**

|  |
| --- |
|  |

物化路径：

|  |
| --- |
|  |

嵌套集合模型：

|  |
| --- |
|  |

**自底向上：邻间>物化路径>嵌套集合**

6.ER模型到表结构是一个完整的映射，1对1（部分参与），1对多（部分参与），多对多。