## 关系模型 的 关系 ：

　　关系模式是对关系的描述。

　　关系实际上就是关系模式在某一时刻的状态或内容。也就是说，关系模式是型，关系是它的值。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。但在实际当中，常常把关系模式和关系统称为关系，读者可以从上下文中加以区别。关系模式可以形式化地表示为：

　　R（U，D，dom，F）

　　R 关系名，U 组成该关系的属性名集合，D 属性组U中属性所来自的域，dom 属性向域的映象集合，F 属性间的数据依赖关系集合。例如：导师和研究生出自同一个域——人，取不同的属性名，并在模式中定义属性向域的映象，即说明它们分别出自哪个域：

　　dom（SUPERVISOR-PERSON）= dom（POSTGRADUATE-PERSON）=PERSON

　　关系模式通常可以简记为：

　　R (U) 或 R (A1，A2，…，An)

　　R 关系名，A1，A2，…，An 属性名，注：域名及属性向域的映象常常直接说明为属性的类型、长度。

　　关系数据库系统是支持关系模型的数据库系统。

关系模型所具有的特点是：概念单一、规范化、以二维表格表示。

1. 关系模型的一致性

如何保证数据库的查询结果跟实际情况一样具有有效性？

如何建立一个合适的关系模式使得不会出现

1、数据冗余

如果一个教师教几门课程，那么这个教师的地址就要重复几次存储；

2、修改异常

必须同时修改几个主键相同的元组的相关信息；

3、插入异常

如果一门课程没有教师来教，则无法插入到数据库中去（缺少主关键字）；

4、删除异常

如果删除一个教师的教学任务，同时也会删除教师信息。

你应该这么改造：

原来的：

数据库模式S〈U，F〉

U = { SNO，SDEPT，MN，CNAME，G }

F = {SNO→SDEPT，SDEPT→MN，(SNO，CNAME)→G}



改造后:

S〈SNO,SDEPT, SNO→SDEPT〉；   
SG〈SNO,CNAME,G, (SNO,CNAME)→G〉；   
DEPT〈SDEPT,MN, SDEPT→MN〉；

改造的标准做法是：规范化

## 规范化

1. 在关系模式R中的每一个具体关系r中，如果每个属性值都是不可再分的最小数据单位，则称R是第一范式（First Normal Form）的关系模式。

反例：



1. 若R∈lNF,且每一个非主属性完全函数依赖于关键属性。

不允许部分依赖的存在。

反例：

设有关系模式SC(S#，C#，GRADE，CREDIT),其中CREDIT表示学分。存在函数依赖：(S#，C#) **→** GRADE， C# **→** CREDIT，候选码是(S#，C#)。

应进行分解:

SCl(S#，C#，GRADE)

C2(C#，CREDIT)

1. 如果关系模式R（U，F）中的所有非主属性对码都不存在传递依赖（第二种定义），则称关系只是第三范式的。

反例：

关系模式Sl(S#，SNAME，D#，DNAME，LOCATION)，关键字是S#。location会随着学生的增多产生冗余，原因是因为S#->D#，而D#->LOCATION。

这是由S#与D#是一对多造成的。

1. 如果关系模式R（U，F）的所有属性都不传递依赖于R的任何候选码，那么称关系RBCNF。

1NF 确保原子性（Atomicity）

◦原子性的粒度、原子性的价值

2NF 检查对键的完全依赖

◦价值在在于控制数据冗余和查询性能

3NF 检查属性的独立性

规范化的价值

◦合理规范化的模型可应对需求变更

◦规范化数据重复降至最少

## 其他原则

1. 极端的例子：四个属性取值都是T/F，可以用0-15这16个数值代表四个属性所有组合状态

◦技巧可能违反了原子性的原则

◦为数据而数据，是通向灾难之路

约束应明确说明

数据中存在隐含约束是一种不良设计

字段的性质随着环境变化而变化时设计的错误和不稳定性

数据语义属于DBMS，别放到应用程序中

1. 过于灵活的危险性

性能与正确性令人失望

例：四通表

1. 如何处理历史数据

## 数据库性能调优

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。

1. 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。

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。

1. 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。

1. 保持数据库连接的稳定。
2. 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。
3. 优化SQL语句

把逻辑放入到查询SQL中而不是SQL宿主语言中；

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；

优化SQL语句的结构。

1. 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

网络答案：<http://blog.csdn.net/yzsind/archive/2010/12/06/6059209.aspx>

## SQL语句注意点

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')



* 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

1. 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

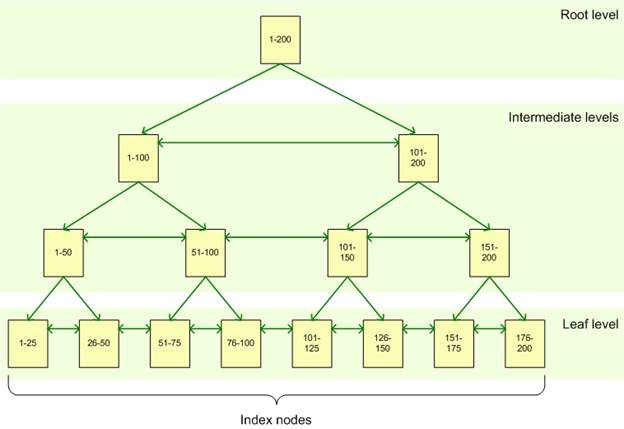
4 1 2 3 5

(3) 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。

## 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)

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）

联合索引是由多个字段组成的索引。

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index\_name

[USING index\_type]

ON tbl\_name (index\_col\_name,...)

index\_col\_name:

col\_name [(length)] [ASC | DESC]

如果你经常要用到多个字段的多条件查询，可以考虑建立联合索引，一般是除第一个字段外的其它字段不经常用于条件筛选情况，比如说a,b 两个字段，如果你经常用a条件或者a+b条件去查询，而很少单独用b条件查询，那么可以建立a,b的联合索引。如果a和b都要分别经常独立的被用作查询条件，那还是建立多个单列索引。

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

通常情况下，如果不使用基于函数的索引，那么当SQL语句在的Where子句中队存在索引的列使用函数时，这会让数据库的优化器忽略掉这些索引。也就是 说，这种情况下即使只存在着少量的复合条件的信息，数据库仍然会对这张表进行全表扫描，以获取相关的数据。这主要是因为这些索引实际上已经改变了被索引列 的值。

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。

## 数据库的物理组织

### IOT

索引组织表(IOT)不仅可以存储数据，还可以存储为表建立的索引。索引组织表的数据是根据主键排序后的顺序进行排列的，这样就提高了访问的速度。但是这是由牺牲插入和更新性能为代价的(每次写入和更新后都要重新进行重新排序)。

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，所以就等于是约定了数据存放的时候是按照严格规定的，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。

#### 不适用情况

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。

如果不是经常使用主键访问表，就不要使用IOT.

使用IOT的好处：  
1、由于索引项和数据存储在一起，所以无论是基于主键的等值查询还是范围查询都能大大节省磁盘访问时间。  
2、为了能够更快地访问那些频繁访问的列，可以使用溢出存储选项将那些访问不频繁的列放在B树叶结点数据块之外的溢出堆空间中。这样一来便可以得到更小的B树，以及包含更多行的叶结点  
3、和堆组织表和索引不同，主键不需要被存储两次。  
4、ROWID伪列是基于主键值的逻辑rowid，而不是物理rowid，即使表被重新组织过，造成了基表行的迁移，二级索引仍然可用，不需要重建。

### 堆文件的组织

表空间->段->块

freelist

## 堆文件和IOT的比较

一个传统表存储数据是使用“heap”的结构，或者是一大堆数据。索引组织表(IOT)存储数据是使用B-tree索引数据结构。在一个heap表中，数据没有特定的顺序;在一个IOT中，数据是按照主键值来排序。

当只在WHERE语句中使用具体主键值查询表内容时，建议使用IOT。如果你在WHERE语句中查询其它非主键值列，那你会发现使用IOT会降低你的主机性能。

1. 对于经常需要更新的表，堆表比IOT快

2. 对于由主键的查询尤其是范围查询，IOT快。

3. 做笛卡尔乘积，堆表快。

## 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)，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。

## 检索和更改：双刃剑

聚集利于检索，分散利于更改

## 采用何种存储方式？

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

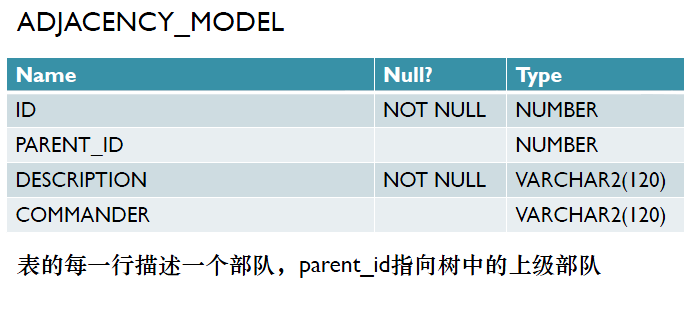
## 主从表与树状表

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

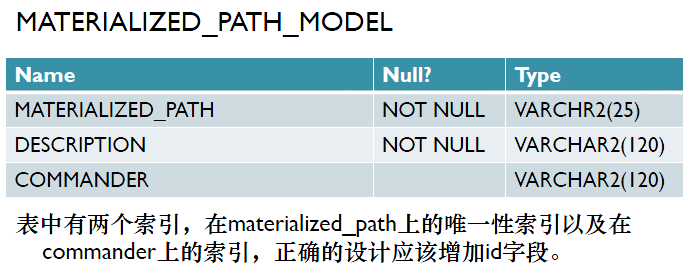
◦Adjacency model-邻接模型



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；删除子树难

◦Materialized path model-物化路径模型



对树中的每个节点，都要记录其在树中的位置信息

注意问题：

物化路径不该是KEY，即使他们有唯一性

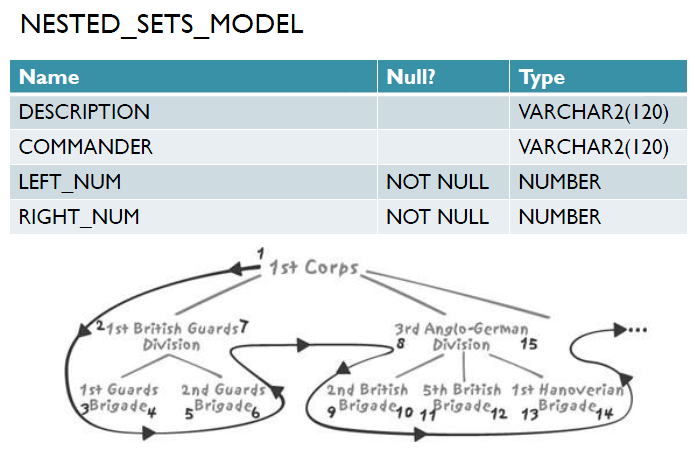
物化路径不该暗示任何兄弟节点的排序

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。



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

## 树的操作

vandamme查询

计算每一层人数

1. 提高数据库性能的原则？

①  数据集中化：分布式数据库系统大大增加系统复杂性，如分在数据一致性更新以及网络传输方面有代价；离数据越近，访问速度越快；

②  保持数据库连接稳定：数据稳定、减少交互；

③  充分利用每次数据库访问：在合理范围内每次数据库访问完成尽量多的工作；

④  接近DBMS核心：代码的执行越接近DMBS的核心执行就越快。 比如在Oracle中用SQL和用PL/SQL分别执行100万次数据插入，会发现前者更快，因为SQL比PL/SQL更接近DBMS核心

⑤  把逻辑放入查询中：尽量把条件逻辑放到SQL语句中，而不是SQL宿主语言中。很明显比如通过Java写数据库查询，则还得经过JDBC驱动层将请求转发给数据库服务器，数据库服务器查询到数据后又要经过JDBC驱动层返回给用户，查询效率肯定比不上直接在cmd下用SQL查询；

⑥  慎用自定义函数：如果自定义函数中包含select，则不能为CBO Optimizer优化。

 2.  SQL隐性优化：

这里只提下oracle中的rownum，PPT上的这条SQL语句：

|  |
| --- |
| select \*  from (select empname, salary  from employees  where status != 'EXECUTIVE' order by salary desc)  where rownum<= 5 |

是有问题的，因为先执行where，再from，得到的记录数可能小于5且就算得到5条记录，查到的数据也是不正确的。

关于DISTINCT: 避免在高层使用DISTINCT.

EXISTS和IN的差异：

* EXISTS子查询是关联子查询（内层查询依赖外层查询），IN子查询是非关联子查询
* EXISTS运行方式先运行主查询一次，根据主查询返回的记录再去子查询中查询是否满足EXISTS条件。所以如果主查询返回的记录比较少，且子查询中的表比较大，用EXISTS效率可能比较高。
* IN运行方式是先执行子查询产生结果集，然后主查询在结果集里寻找符合要求的字段列表。所以如果子查询结果集比较小且主查询中的表比较大时，用IN效率可能比较高。
* 一般情况下EXISTS效率比IN要高。

关于SQL语句多种变换：最古老的SQL连接查询、JOIN查询、EXISTS、IN、内嵌视图查询

大数据量查询：

所有影响聚合函数的条件应该放在having子句中；

任何无关聚合条件应该放在where子句中；

减少group by排序用到的数据的量。

附带SQL语句优化补充点：（全部以oracle为准）

* 表连接顺序：oracle解析器从右到左处理from子句中的表名，包含多表时应该将记录条数最少的表作为基础表（通俗理解为放在离from子句的末尾）；如果由3个以上的表进行连接查询，则选择交叉表（被其他表引用的表）作为基础表；
* Where子句中的连接顺序：oracle自下而上解析where子句，据此表之间的连接应写在其他where条件之前；可过滤最大数量的条件应该放在where子句的末尾。

 3.  索引：

索引目的：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，主要目的是为了提高查询效率。

B树索引结构：

B树索引由根节点、两个或两个以上的分支节点和叶节点组成。根节点指向分支节点，分支节点指向其他分支节点或叶节点。叶节点包含索引（INDEX）并且指向水平方向的其他叶节点。再画个图示意下，略。

为什么不用索引？（6条）

* 使用B+树索引，而且为此中没有使用索引的最前列。如：

表T中，T(X,Y)是索引，有SQL语句 SELECT FROM T WHERE Y=5 则索引是不会发挥作用的；

* 使用SELECT COUNT(\*) FROM T，则即使T上有索引，依然会进行全表扫描；
* 对一个索引列做函数查询：如SELECT \* FROM T WHER f(indexed\_col) = value
* 隐式函数查询：比如查询SQL中存在隐式转换比如no为VARCHAR类型，但是查询时把该字段当做INT类型查询：

SELECT \* FROM T WHERE no = 2124232

* 使用索引反而会更慢：比如经常更新数据，碎片太多，索引占用空间太大，查询优化器会认为没有必要使用索引了；
* 没有正确的统计信息，导致CBO无法做出正确的选择

 4.   数据库物理组织：

堆文件：

最简单最基本的数据库数据组织方式，新插入的记录直接被到堆文件末尾（插入效率较高），因此堆文件是无序的。

但查询时线性搜索整个文件，查询效率较差。但对于查询所有记录且访问顺序不重要情况下的查询效率比较高。

IOT：（索引组织表）

插入效率可能低于堆文件，最大特点是记录是排序的。在主键索引中存储表中所有数据。表即数据，在查询一组高频度关联数据时比较高效。

分区：

分区是将数据分组，提高了并发性和并行性。对于多用户并发访问情况，查询效率可能较高。分区的多种方式就不提了，分区的范畴太广。