[1. 为什么要有事务，事务的作用，事务有哪些特点，每个特点的含义是什么？ 2](#_Toc460253425)

[ 事务的定义 2](#_Toc460253426)

[ 事务的特点 2](#_Toc460253427)

[ 事务的状态 3](#_Toc460253428)

[2. 有哪些隔离级别，这几个隔离级别的关系是什么? 3](#_Toc460253429)

[3. 事务的并行执行和调度 4](#_Toc460253430)

[4. 数据库访问并发控制的方法，乐观锁和悲观锁的的区别和实现？ 4](#_Toc460253431)

[5. Join、left join、right join三者的区别 6](#_Toc460253432)

[6. union、union all两者的区别 6](#_Toc460253433)

[7. 写一个sql语句，输入为表a(aid, aname)、表b(bid, aid, value)，要求输出(aid, aname, avg(value))比如每个班级的平均成绩 6](#_Toc460253434)

[8. 什么是视图？ 6](#_Toc460253435)

[9. 什么是游标 7](#_Toc460253436)

[10. 分表技术包含哪些，怎样分表的？ 7](#_Toc460253437)

[11. 什么是数据表索引，怎样给一个数据表加上索引？ 7](#_Toc460253438)

[ 基本概念 7](#_Toc460253439)

[ 索引的优点 7](#_Toc460253440)

[ 索引的缺点 8](#_Toc460253441)

[ 应该在哪些列上创建索引 8](#_Toc460253442)

[ 不应该在哪些列上创建索引 8](#_Toc460253443)

[ 索引的分类 8](#_Toc460253444)

[ 顺序索引 9](#_Toc460253445)

[ 非聚集索引(辅助索引) 9](#_Toc460253446)

[ B+树索引 9](#_Toc460253447)

[ 散列索引 hash index 11](#_Toc460253448)

[ 顺序索引和散列的比较 11](#_Toc460253449)

[ SQL中的索引定义 11](#_Toc460253450)

[12. 数据库连接池是什么，它是怎样工作的？ 11](#_Toc460253451)

[13. 恢复系统 12](#_Toc460253452)

[14. 各个范式有何区别？ 13](#_Toc460253453)

[15. MySQL数据库锁 14](#_Toc460253454)

[16. 数据库中各个键的区别 14](#_Toc460253455)

[15. RowNUM 和 RowID 14](#_Toc460253456)

[16. 如果Mysql单个表的数据量很大，导致查询变慢，怎样优化？ 15](#_Toc460253457)

[17. Oracle大数据量时的优化 15](#_Toc460253458)

[18. 分页查询 15](#_Toc460253459)

[19. 回收站功能 16](#_Toc460253460)

[20. 表空间 16](#_Toc460253461)

[21. 数据库、段和区 16](#_Toc460253462)

1. 为什么要有事务，事务的作用，事务有哪些特点，每个特点的含义是什么？

* 事务的定义

事务(Transaction):是并发控制的单元，是用户定义的一个操作序列。这些操作要么都做，要么都不做，是一个不可分割的工作单位。通过事务，sql server 能将逻辑相关的一组操作绑定在一起，以便服务器 保持数据的完整性。

事务是数据库中一个单独的执行单元，通常由高级数据库操作语言或编程语言编写的用户程序的执行所引起的。当在数据库中更改数据成功时，在事务中更改的数据便会提交，不再更改，否则事物取消或回滚。数据库事务是为了保证在数据库更改失败的情况下还可以回到执行事务之前的状态。

SQL标准规定当一个SQL语句被执行，就隐含地开始了一个事务，事务最后被提交或回滚之后事务就结束，如果一个事务没有完成commit，其影响都可以被回滚，在断点和系统崩溃的情况下，回滚会在事务重启后执行。

每一个SQL语句默认为自成事务，且一旦执行完就提交。如果一个事务要要执行多条SQL语句，就必须关闭默认的每一个SQL语句自动提交。如何关闭自动提交也依赖于具体的SQL实现。

* 事务的特点

原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability).即ACID四种属性。

原子性：要求事务必须被原子性执行即要么不执行，要么全执行。避免执行一部分操作带来的错误。这个工作由事务管理部件处理。

一致性：一个事务执行前后，数据库必须保证一致性的状态。数据库的一致性状态应该满足**模式锁**指定的约束，那么在完整执行该事物后数据库仍然处于一致性状态。

隔离性：当两个或多个事务并发执行时，为了保证数据之间的安全性，将一个事物内部的操作隔离起来，不被其他正在执行的事物看到。确保隔离性是并发控制部件的责任。

持久性：也称永久性，事务完成以后，DBMS(数据库管理系统)保证它对数据库中的数据的修改是永久性的，当系统或介质发生变化时，该修改也永久保持。确保持久性是数据库系统中恢复管理部件的责任。

一般来说，事务的ACID四个特性都是由数据库管理系统进行保证的，在整个应用程序执行过程中，应用无需去考虑数据库的ACID属性。

一般情况下通过执行COMMIT或ROLLBACK语句来终止事务的，当执行COMMIT语句时，事务对数据库做的修改也就成了永久性的了，执行ROLLBACK语句时，事务启动以来对数据库做的修改就撤消了，并且数据库回到事物执行之前的状态。

* 事务的状态

事务必须属于以下五个状态之一。

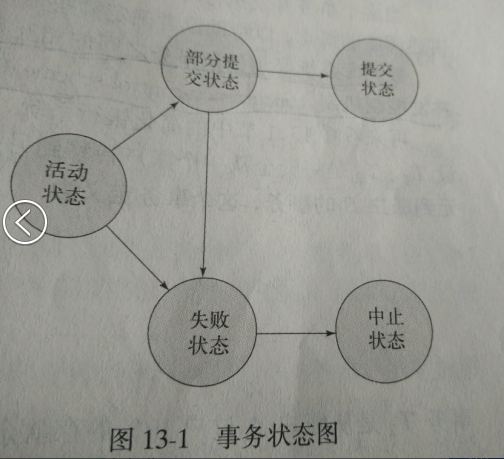
① 活动状态：事务执行时处于这个状态；

② 部分提交状态：最后一条语句执行之后；

③ 失败状态：发现正常的执行不能继续之后；

④ 中止状态：事务回滚并且已经恢复到事务开始执行之前的状态；

⑤ 提交状态：成功完成后。



事务进入中止状态之后可以选择重启事务或杀死事务。重启事务是仅当引起事务中止的是硬件错误或不是由事务的内部逻辑产生的软件错误时，重启事务看成是一个新的事物；杀死事务是由于事务的内部逻辑错误，只有重写应用程序才能改正，或者由于输入错误，或所需数据在数据库中没有找到。

注意:

BEGIN或START TRANSACTION；显示地开启一个事务；

COMMIT；也可以使用COMMIT WORK，不过二者是等价的。COMMIT会提交事务，并使已对数据库进行的所有修改称为永久性的；

ROLLBACK；有可以使用ROLLBACK WORK，不过二者是等价的。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；

SAVEPOINT identifier；SAVEPOINT允许在事务中创建一个保存点，一个事务中可以有多个SAVEPOINT；

RELEASE SAVEPOINT identifier；删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；

ROLLBACK TO identifier；把事务回滚到标记点；

SET TRANSACTION；用来设置事务的隔离级别。InnoDB存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ和SERIALIZABLE。

1. 有哪些隔离级别，这几个隔离级别的关系是什么?

数据库有四种类型的事务隔离级别：不提交的读、提交的读、可重复的读、串行化。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 读数据一致性 | 脏读 | 不可重复读 | 幻读 |
| 未提交读(脏读) | 最低级别，只能保证不读取物理上损坏的数据 | 是 | 是 | 是 |
| 已提交读(不可重复读) | 语句级 | 否 | 是 | 是 |
| 可重复读 | 事务级 | 否 | 否 | 是 |
| 序列化 | 最高级别，事务级 | 否 | 否 | 否 |

* 未提交读：一个事务在执行过程中可以看到其他事务没有提交的新插入的记录，也能看到其他事务没有提交的对已有记录的更新。也称为脏读。
* 已提交读：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录。也就是说一个事务从开始到提交之前对数据库做的修改对其他事务都是不可见的。但是其他事务可以修改这个事务正在读的行。
* 可重复读：不能读取已由其它事务修改了但是未提交的行，其它任何事务也不能修改在当前事务完成之前由当前事务读取的数据。但是对于其它事务插入的新行数据，当前事务第二次访问表行时会检索这一新行。因此，这一个隔离级别的设置解决了不可重复读取的问题，但是避免不了幻读。
* 序列化：当数据库系统使用序列化隔离级别时，不能读取已由其它事务修改但是没有提交的数据，不允许其它事务在当前事务完成修改之前修改由当前事务读取的数据，不允许其它事务在当前事务完成修改之前插入新的行。一个事务在执行过程中完全感觉不到其他事务对数据库所做的更新。事务实际上以串行化方式运行。

|  |  |
| --- | --- |
| 脏读 | 事务T1更新了一行记录的内容，但是并没有提交所做的修改。事务T2读取更新后的行，然后T1进行了回滚操作，取消了刚才所做的修改。现在T2读到的之前T1修改的行则是无效的。(一个事务读取了另一个事务未提交的数据) |
| 不可重复读 | 事务T1读取了一行记录，紧接着T2修改了刚才T1读取的那一行记录，然后T1再次读取这行记录，发现与刚才读取的结果不同。 |
| 幻读 | 事务T1读取一条指定的Where子句所返回的结果集，然后事务T2事务新插入一行记录，这行记录恰好可以满足T1所使用的查询条件。然后T1再次对表进行检索，但又看到了T2新插入的数据。 |

1. 事务的并行执行和调度

事务的并行可带来两个显著的好处：提高吞吐量和资源利用率、减少等待时间。数据库系统必须控制事务之间的相互影响，防止它们破坏数据库的一致性。系统通过称为并发控制机制的一系列机制来保证这一点。

各个事务之间的执行顺序称为调度(schedule)，表示指令在系统中执行的时间顺序。

当分别属于Ti和Tj的两条连续的指令Ii和Ij是不同事务对相同数据项的操作，并且Ii和Ij其中至少有一个是write指令时，就说是冲突的。

若调度S经过一系列非冲突指令交换成S’,称S和S’是冲突等价(conflict equivalent)。

若调度S和一个串行调度冲突等价，称调度S是冲突可串行化(conflict serializable)的。

1. 数据库访问并发控制的方法，乐观锁和悲观锁的区别和实现？

事务最基本的特性之一是隔离性。为保持事务的隔离性，系统必须对并发事务之间的相互作用加以控制，这是通过称为并发控制机制的一系列机制中的一种实现来的。后面讨论的并发控制的机制是保证调度是可串行化的。

① 基于锁的协议

一个事务在访问数据项时，必须拥有该数据项的锁。要访问一个数据项，事务Ti必须首先给该数据项加锁。如果该数据项已被另一个事务加上了不相容的锁，则在所有其他事务持有的不相容类型锁被释放之前，并发管理器不会授予锁。这样只好等待，直到所有其他事务持有的不相容的锁被释放。

一旦事务循环等待释放锁，就会发生死锁，发生死锁之后必须进行事务回滚。事务发生回滚时，就会释放该事务锁住的数据项，其他事务就可以访问这些数据项，继续自己的执行。

锁的粒度也有所不同，有的是单元数据项加锁，也可以给行加锁，表加锁，甚至是数据库加锁。

② 基于时间戳的协议

对于系统中的每个事务Ti，把一个唯一的固定时间戳和它联系起来，此时间戳即为TS(Ti)。该时间戳是在事务Ti开始执行前由数据库系统赋予的。若事务Ti已被赋予时间戳TS(Ti)，并且一旦有新的事务Tj进入系统，则TS(Ti)< TS(Tj)。事务的时间戳决定了串行化顺序。

事务的时间戳决定了串行化顺序。因此，若TS(Ti)< TS(Tj)，则系统必须保证所产生的调度等价于事务Ti出现在事务Tj之前的某个串行调度。

要实现这个机制，每个数据项Q需要与两个时间戳值相关联。

W-timestamp(Q)表示成功执行write(Q)的任意事务的最大时间戳。

R-timestamp(Q)表示成功执行read(Q)的任意事务的最大时间戳。

每当新的read(Q)或write(Q)指令执行时，这些时间戳就更新。

时间戳排序协议保证任何有冲突的read或write操作按时间戳顺序执行。某个事务开始执行时会先判断本次执行是否会有冲突或有效。

③ 基于有效性协议

为进行有效性检测，需要知道事务Ti的各个阶段何时进行。为此，将三个不同的时间戳与事务Ti相关联。

Start(Ti):事务Ti的开始执行时间；

Validation(Ti):事务完成读阶段并开始其有效性检查阶段的时间；

Finish(Ti):事务Ti完成写阶段的时间。

利用时间戳Validation(Ti)的值，通过时间戳排序技术决定可串行化顺序，因此，值TS(Ti) = Validation(Ti)，若TS(Ti)< TS(Tj)，则系统必须保证所产生的调度等价于事务Ti出现在事务Tj之前的某个串行调度。选择Validation(Ti)而不是Start(Ti)做为事务的时间戳，因为在事务Ti冲突频度很低的情况下可望有更快的响应时间。

在并发环境下，一般可以采用多种机制来保证数据的一致性，例如oracle系统提供的事务级的一致性、行级锁、表级锁等。

对行级锁，提交读和串行性事务都采用行级锁，都会在试图修改一个没有提交的并行事务更新的行时产生等待。第二个事务等待其他事务提交或者撤消来释放它的锁，才能更新给定的行。不管等待的是什么级别的隔离事务，若其他事务回滚了，都可以对以前锁定的行进行修改。

悲观锁(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

乐观锁(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

实现方法例子：

悲观锁：普通的aspx页面，当用户点提交后，直接将提交及相关按钮的enabel改为false，直到提交事件完成后，再改回来。另外在数据层那一块，每次提交数据更改时，都需要判断数据以前的状态是否改变，以防止有并发改变的情况出现。

乐观锁：SQL中的事务来实现数据操作不成功就回滚。

1. Join、left join、right join三者的区别

Join(内连接、自然连接)根据连接条件查出两张表中满足连接条件的数据，查出两个表中相同的数据；

left join，以左表为主；

right join，以右表为准；

left join和right join为外连接。

1. union、union all两者的区别

union在对表求并集后会去掉重复的记录，所以会对产生的结果集进行排序运算，删除重复的记录后再返回结果。

Union all 则只是将简单地将两个结果集合并之后就返回结果。因此，如果union的两个结果集中有重复的记录，那么返回的结果集就会包含重复的数据。

1. 写一个sql语句，输入为表a(aid, aname)、表b(bid, aid, value)，要求输出(aid, aname, avg(value))比如每个班级的平均成绩

Select a.aid aname avg(value) from a,b where a.aid = b.aid GROUP BY a."aid";

1. 什么是视图？

视图是由从数据库的基本表中选出来的数据组成的逻辑窗口，与基本表不同，它是一个虚表。在数据库中，存放的只是视图的定义，而不存放视图中包含的数据项，这些数据项仍然存放在它原来的物理表中。

视图的作用主要有四点：

① 可以简化数据查询语句；

我们在使用查询时，在很多时候我们要使用聚合函数，同时还要 显示其它字段的信息，可能还会需要关联到其它表，这时写的语句可能会很长，如果这个动作频繁发生的话，我们可以创建视图，这以后，我们只需要select \* from view就可以啦，这样很方便。

② 可以使用户从多角度看待同一数据；

③ 通过引入视图可以提高数据的安全性；(定义视图以将表与表之间的复杂的操作连接和搜索条件对用户不可见，用户只需要简单地对一个视图进行查询即可，故增加了数据的安全性，但是不能提高查询的效率)

④ 视图提供了一定程度的逻辑独立性。

1. 什么是游标

游标提供了一种对从表中检索出的数据进行操作的灵活手段，它实际上是一种能从包含多条数据记录的结果集中每次提取一条记录的机制。

游标由结果集和结果集中指向特定记录的游标位置组成的。

在select返回的行集合中，游标不允许程序对整个行集合执行相同的操作，但对每一行数据的操作不作要求。游标有以下两个优点：

① 在使用游标的表中，对行提供删除和更新的操作；

② 游标将面向集合的数据库管理系统和面向行的程序设计连接了起来。

视图和游标的区别：

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

1. 分表技术包含哪些，怎样分表的？

分表技术顾名思义，就是把若干个存储相同类型数据的表分成几个表分表存储，在提取数据的时候，不同的用户访问不同的表，互不冲突，减少锁表的几率。比如，目前保存用户分表有两个表，一个是user\_1表，还有一个是 user\_2 表，两个表保存了不同的用户信息，user\_1 保存了前10万的用户信息，user\_2保存了后10万名用户的信息，现在如果同时查询用户 heiyeluren1 和 heiyeluren2 这个两个用户，那么就是分表从不同的表提取出来，减少锁表的可能。

数据库分表就是把一张表中的记录分到多张表上。查到的一个分表的算法就是hash分表：通过一个原始目标的ID或者名称通过一定的hash算法计算出数据存储表的表名，然后访问相应的表。

1. 什么是数据表索引，怎样给一个数据表加上索引？

http://blog.csdn.net/kennyrose/article/details/7532032

* 基本概念

索引（id）是一种提高数据库查询速度的机制，它是一个在数据库的表或视图上按照某个关键字段的值，升序或降序排序创建对象。当用户查询索引字段时，它可以快速地进行检索操作，不需要遍历整张表的内容就可以快速找到要检索的数据。

索引是与表或试图关联的磁盘结构，即对表中列值排序的一种结构，可以加快从表或视图中检索行的速度，执行查询时不必扫描整张表就能更加快速地访问数据库中的信息。

一条索引记录包含键值和索引指针。创建索引时，系统分配一个索引页。在表中插入一行记录时，同时也向该索引页插入一条索引记录。索引记录包含的索引字段值比真实数据量小，节省了空间。

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

* 索引的优点

创建索引可以大大提高系统的性能：

* 1. 创建唯一性的索引，可以保证数据库中每一行数据的唯一性；
  2. 通过索引，可以大大加快数据的检索速度；
  3. 通过索引可以加快表和表之间的连接，从而有效实现数据的参考完整性；
  4. 在使用分组和排序子句进行数据检索时，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。
* 索引的缺点

过多的索引会带来的问题？

1. 创建索引和维护索引要耗费时间、空间。
2. 除了数据表中的记录要占用空间之外，每一个索引还要占用一定的物理内存；
3. 当对表中的数据进行增加、修改、删除操作时，索引也要动态地维护，从而降低了数据的维护速度。

* 应该在哪些列上创建索引

一般来说，应该在这些列上创建索引：

① 在经常**需要搜索**的列上，可以加快搜索的速度；

② 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；

③ 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；

④ 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；

⑤ 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；

⑥ 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。

* 不应该在哪些列上创建索引

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的的这些列具有下列特点：

① 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。

②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。

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。

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。

* 索引的分类

**顺序索引**(ordered index)：基于值的顺序排序。

**散列索引**(hash index)：基于将值平均分布到若干散列的桶中，一个所属的散列桶是由一个函数决定的，该函数称为散列函数。

在开始后面的介绍之前，先解释一个名词搜索码(搜索键)：用于在文件中查找记录的属性或属性集。

为了快速随机存取文件中的记录，可以使用索引结构。每个索引结构与一个特定的搜索码关联。一个文件可以有多个索引，分别对应不同的搜索码。如果包含记录的文件按照某个搜索码指定的顺序排序，那么该搜索码对应的索引称为聚集索引(clustering index)。聚集索引也被称为主索引(primary index)。主索引看起来是指建立在主码上的索引，但是实际上它可以建立在任何搜索码上。通常情况下聚集索引的搜索码是主码，但也并非总是如此。搜索码指定的顺序与文件中记录的物理顺序不同索引称作是非聚集索引(nonclustering index)或辅助索引(secondary)。

* 顺序索引

称在某个搜索码上有聚集索引的文件为索引顺序文件。

索引记录(index record)或称为索引项(index entry)由一个搜索码值和指向具有该搜索码值的一个或多个记录的指针构成。指向记录的索引指针包括磁盘块的标识和标识磁盘块内记录的块内偏移量。

可以使用的顺序索引有稠密索引(dense index)和稀疏索引(sparse index)。稠密索引为每一个搜索码都建立一个索引记录，在稠密索引中，索引记录包括搜索码值以及指向具有该搜索码值的第一个数据记录的指针，具有相同搜索码值的其余记录顺序地存储在第一个数据记录之后，由于该索引是聚集索引，所以记录根据相同的搜索码排序。

稀疏索引是只为搜索码的某些值建立索引记录。和稠密索引一样，每个搜索码记录也包括搜索码值和指向具有该搜索码值的第一个数据记录指针，但是为了定位一条记录，需要找到其最大搜索码值小于或等于所找记录的搜索码值的索引项，然后从这个索引项指向的记录开始向下找，直到找到要查询的结果为止。

稠密索引和稀疏索引各有好处，稠密索引可以更快速地定位一条记录，但是稀疏索引占的空间小。

如果索引本身变得很大而难于有效管理，可以给索引再建立索引，采用多级索引(multilevel)的形式来管理。

使用索引顺序文件组织的最大缺点在于随着文件的增大，索引查找性能和数据顺序扫描性能都会下降。虽然这种性能下降可以通过对文件进行重新组织来弥补，但是实际中却不希望频繁的进行重组。

* 非聚集索引(辅助索引)

辅助索引必须是稠密索引，对每个搜索码值都有一个索引项，而且对文件中的每个记录都有一个指针。

辅助索引能够提高使用聚集索引搜索码以外的码的查询性能。但是，辅助索引显著增加了数据库的更新开销。数据库设计者根据对查询和更新相对频率来决定哪些辅助索引是需要的。

* B+树索引

B+树索引结果使用是最广泛的，在数据插入和删除的情况下仍能保持其执行效率。B+树采用平衡树结构，其中树根到树叶的每条路径的长度相同即叶子结点都在同一层上。

B+树是稠密索引，各搜索码值都必须出现在某个叶子结点中。

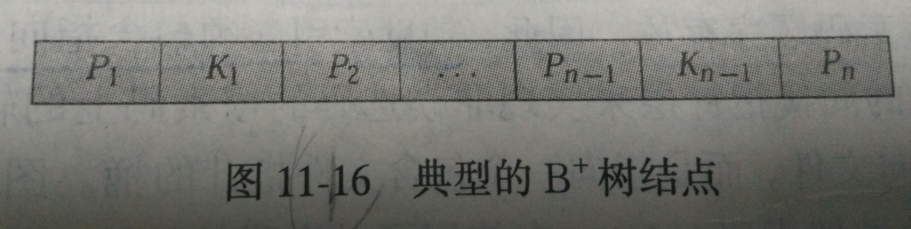
B+树索引的缺点：会增加文件插入和删除处理的性能开销，也会增加空间开销。但是即使对更新频率较高的文件来说，这种开销也是可以接受的，因为可以减少文件重组的开销。此外，由于结点有可能是半空的(如果他们具有最少子结点数的话)，这将造成空间的浪费，但是考虑到B+树所带来的性能提高，这种空间开销也是可以接受的。

B+树索引是一种多级索引，但是其结构不同于多级索引顺序文件。

对于一个n阶B+树来说，它的每个叶子结点最多可有n-1个值，允许叶结点包含的值的个数最少为(n-1)/2(向上取整)，各叶结点中值的范围互不重合。因此如果Li<Lj，那么Li中的所有搜索码值都小于Lj中的所有搜索码的值。

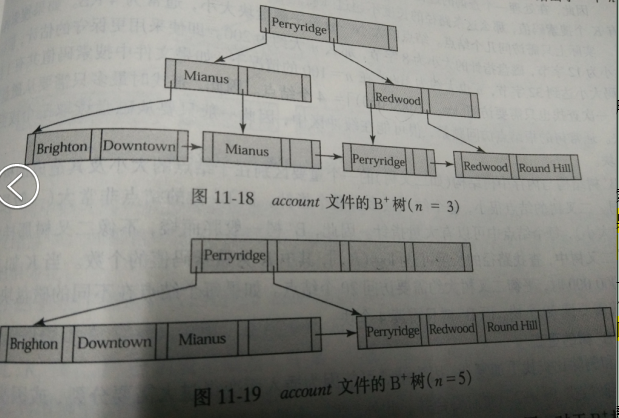
B+树的非叶结点形成叶结点上的一个多级(稀疏)索引。非叶结点的结构和叶结点的相同，只不过非叶结点中所有的指针都是指向树中结点的指针。一个非叶结点可以最多容纳n个指针，最少容纳n/2(向上取整)个指针。

典型的B+树结点指针如下：



对于包含m个指针的非叶子结点，P1指向的子树中所有搜索码值小于K1的那一部分的子树，指针Pm指向的子树中所有搜索码的值大于等于Km-1的那一部分子树。对i=2,3，…m-1，指针Pi指向一棵子树，该子树包含的搜索码的值小于Ki且大于等于Ki-1。即把Ki-1这个关键字的值存在了Pi指向的子树中, 把Ki这个关键字的值存在了Pi+1指向的子树中。

对于包含n个指针的叶结点，对i=1,2,3…n-1，将指针Pi指向具有搜索码值Ki的一个文件记录或指向一个指针桶，而桶中的每个指针指向具有搜索码值Ki的一个文件记录。桶结构只在搜索码不是候选码且文件不按搜索码顺序存放时才使用。用Pn将叶结点按搜索码顺序串在一起，这种顺序允许文件进行高效的顺序处理。如下图给出了B+树的例子。



B+树的查找：

假设要找出的搜索码值为V的所有记录。首先，检查根结点，找到大于V的最小搜索码值，假设我们找到的这个搜索码值是Ki。然后顺着指针Pi到达另一个结点。如果找不到这样的值，则V>Km-1,其中m是该结点中的指针个数。这种情况下，沿着Pm到达另一个结点。按上述方式到达的那个结点中，再次寻找大于V的最小搜索码值，并且按照上面那样沿着相应的结点向下。最终，我们将到达一个叶结点，如果在该叶结点中某个搜索码值Ki等于V，那么指针Pi指向我们需要的记录或指针桶。如果在该叶结点中找不到值V，则不存在搜索码值为V的记录。

因此在处理一个查询的过程中，需要遍历树中从根到某个叶结点的一条路径，如果文件中有K个搜索码值，那么这条路径的长度不超过Logn/2K.其中n为树的阶数或者说一个磁盘块内搜索码值的个数。

B+树的更新：插入和删除

插入：使用和查找一样的技术，找到搜索码将出现的叶子结点，如果该搜索码值已经存在于该叶子结点，则在文件中加入一个新纪录，并且必要时在指针桶中加入一个指向该记录的指针。如果搜索码值不存在，那么在此叶结点中加入该值，并决定该值的位置使搜索码仍按顺序排列。然后在文件中创建一个新纪录，必要时创建一个新的包含合适指针的指针桶。

删除：使用和查找一样的技术，找到要删除的记录，并将其从文件中删除。如果该删除搜索码值没有对象的指针桶，或者由于删除使该指针桶为空，那么我们从该搜索码值所在叶结点中将其删除。

插入的时候可能结点因为插入而变得过大需要分裂，或因为删除变得过小需要合并。此外，当一个结点分裂或一对结点合并的时候，要保证B+树的平衡。

* 散列索引 hash index

在散列文件组织中，通过计算所需记录搜索码值的函数直接获取包含该记录的磁盘块地址。基于散列技术的文件组织使我们能够避免访问索引结构。

在散列索引组织中，我们把搜索码以及与它们相关联的指针组织成一个散列文件结构。

在对散列的描述中，使用术语桶(bucket)表示能存储一条或多条记录的一个存储单位。

散列的过程中可能出现桶溢出，出现桶溢出的原因有桶不足和桶偏斜。解决的办法是使用溢出桶(类似于采用链地址法解决冲突)。

散列不仅可以用于文件的组织，还可以用于索引结构的创建。散列索引将搜索码及其相应的指针组织成散列文件结构。构建散列索引的方法为：将散列函数作用于搜索码以确定对应的桶，然后将此搜索码以及相应的指针存入桶中(或溢出桶)。

为减少桶溢出的可能性，桶的数目选为(nr/fr)\*(1+d)，其中d是避让因子，其典型值约为0.2. nr 表示将要存储的记录总数 ，fr 表示一个桶中能存放的记录数目。

上面介绍的是一种静态的散列技术，静态散列会要求在前面就确定好桶的个数，但是在随着数据库的变大，就会带来很严重的问题。

集中动态散列技术允许散列函数动态改变，以适应数据库增大或缩小的需要。两种常用的动态散列技术是可扩充散列(extendable hashing)和线性散列(linear hashing)。

* 顺序索引和散列的比较

可以用索引顺序组织或B+树组织将记录文件组织成顺序文件，也可以用散列来组织文件。大多数数据库系统支持B+树索引，并且有可能还额外支持某种形式的散列文件组织或散列组织。

如果基于码值查找并且不执行范围查询，散列组织对于在查询执行过程中创建的临时文件来说特别有用。如果会频繁地使用范围查询，使用顺序索引相对来说更好一点。

* SQL中的索引定义

原则上数据库系统可以自动决定创建何种索引。但是，由于索引的空间代价和索引对更新操作的影响，在维护何种索引上自动地做出正确选择并不容易。因此，大多数SQL实现允许程序员通过数据定义语言的命令对索引的创建和删除进行控制。

**可以用create index创建索引**,如create index test\_idx on tablename (col1,col2)系统优化时会自动选择使整个查询开销最小的查询计划。如你的sql 为select \* from tablename where col1 = @col1 and col2 = @col2 时一般都会用到索引。

1. 数据库连接池是什么，它是怎样工作的？

建立数据库连接的部分通常是整个应用系统的瓶颈所在，因为建立数据库连接的速度很多时候都决定了整个系统的运行速度。原因就是在建立数据库连接的时候需要同数据库系统建立通讯、分配资源和进行权限认证等操作，这些操作对应用程序来说是很耗费时间的。

为了解决数据库连接的瓶颈问题，所以提出了数据库连接池的概念，核心理念就是在一个虚拟的容器中存放已经预先建立好的数据库连接对象，把这些数据库连接对象分配给需要的程序使用，当连接对象使用完之后不直接将这些连接关闭，而是再放回到这个池子中。

通过使用连接池的技术，大大缓解了数据库连接的瓶颈问题，让开发的java应用程序具有更快的运行效率。并且通过数据库连接池具有更好的安全性，并且可以根据应用系统的需要，实时的修改连接池中保存的数据库连接对象的数量，将更有效地使用数据库连接池中的数据库连接对象。

自己写的连接池是一个单例模式的connectionpool，然后初始化的时候根据配置文件中的连接数初始化好相应的数据库连接池中，用ArrayList维护这个连接。每次获取连接时从ArrayList中分配一个连接即可，然后把这个连接对象，等不用的时候再放到池子中。

现在很多WEB服务器(Weblogic, WebSphere, Tomcat)都提供了DataSoruce的实现，即连接池的实现。通常我们把DataSource的实现，按其英文含义称之为数据源，数据源中都包含了数据库连接池的实现。也有一些开源组织提供了数据源的独立实现：

* DBCP 数据库连接池：DBCP 是 Apache 软件基金组织下的开源连接池实现，要使用DBCP数据源，需要应用程序应在系统中增加如下两个 jar 文件① Commons-dbcp.jar：连接池的实现② Commons-pool.jar：连接池实现的依赖库。
* C3P0 数据库连接池：C3P0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。目前使用它的开源项目有Hibernate***，***Spring等。C3P0数据源在项目开发中使用得比较多。
* c3p0与dbcp区别

dbcp没有自动回收空闲连接的功能，c3p0有自动回收空闲连接功能。

1. 恢复系统

* 数据访问

每个事务Ti都有一个私有工作区，用于保存Ti所访问和更新的所有数据项的副本。该工作区在初始化时创建，在事务提交或终止时删除。当事务第一次需要访问数据项X时，他必须执行read(X)。然后，系统对X的所有更新操作都作用于xi。事务最后一个访问X后，必须执行write(X)，在数据库中反映X的变化。

* 基于日志的恢复

使用最为广泛的记录数据库修改的结构就是日志(Log),日志是日志记录(log record)的序列，记录数据库中的所有更新活动。日志记录有几种类型。一个更新日志记录(update log record)描述一次数据库写操作，具有如下字段：

事务标识符：执行write操作事务的唯一标识符；

数据项标识符：所写数据项的唯一标识符。通常是数据项在磁盘上的位置；

旧值：写之前数据项的值；

新值：写之后数据项的值；

一旦日志记录已存在，如果需要，就可以输出对数据库的修改，并且能够利用日志记录中的旧值字段来撤消已经对数据库的修改。

日志必须存放在稳定存储器上，下面介绍延迟的数据库修改和立即的数据库修改两种技术，可以利用日志保证即使发生故障也保持事务的原子性。

**延迟修改技术**：通过在日志中记录所有数据库修改，将一个事务的所有write操作拖延到事务部分提交时才执行，来保证事务的原子性。

**立即修改技术**：允许数据库修改在事务仍处于活跃状态时就输出到数据库中。活跃事务所做的修改称为未提交修改。

利用日志，系统可以解决任何不造成非易失性存储器上的信息丢失的故障。恢复机制使用下面两个恢复过程：

Undo(Ti):将事务所更新的所有数据项的值恢复成旧值；事务Ti需要undo的情况是如果日志中包含记录< Ti start>，但是不包含< Ti commit>;

Redo(Ti):将事务所更新的所有数据项更新为新值, 事务Ti需要redo的情况是如果日志中包含记录< Ti commit>，但是不包含< Ti start>.

* 检查点

周期性地执行检查点，需要执行的动作序列有：

① 将当前位于主存的所有日志记录输出到稳定存储器上；

② 将所有修改了的缓冲块输出到磁盘上；

③ 将一个日志记录<checkpoint>输出到稳定存储器上。

每次系统出现故障的时候，只需要查看最近的一次检查点之后的日志，按照记录的日志内容进行相应的恢复即可。

1. 各个范式有何区别？

引入数据库范式的目的是为了控制和减少数据库中的冗余数据和保证数据库中数据的一致性。范式级别越高，对表设计时的要求就越高，设计出来的表中数据的冗余性就越小，但是在查询的时候就会导致更多的表连接(对数据库效率有影响)。

① 第一范式(1NF)：指数据库中的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或重复的值。即第一范式要求属性值不可再分裂成更小部分；

比如Student(ID,NAME,AGE,ADDR);

② 第二范式(2NF)：在第一范式的基础上建立起来的，即满足第二范式必须满足第一范式。第二范式要求数据库表中的每个实例或行必须可以被唯一地区分即关系模式中的每一个非主属性完全函数依赖于R的某个候选键；

比如Student(ID,NAME,AGE,ADDR);

Book(ISBN,title,pubDate,publisher,contact)；

③ 第三范式(3NF)：如果关系模式R是第二范式，且每个非主属性都不传递依赖于R的候选键，则称R是第三范式；

比如在上面的Book表中，每一行由ISBN唯一确定，但是contact(联系人)依赖于publisher(出版社)，publisher(出版社)依赖于ISBN，出现了contact对主键的传递依赖，所以这个是不符合第三范式的，为了满足第三范式， 需要把Book表拆分成两张表；

Book(ISBN,title,pubDate,publisher)；

Contact(publisher,contact);

④ BCNF：它构建在第三范式的基础上，如果关系模式R是第一范式，且每个属性都不传递依赖于R的候选键，则称R为BCNF的模式。

与第三范式相比，BCNF不但要求非主属性不能传递依赖于R的候选键，也不能存在候选键依赖于候选键的情况。也就是说，一个表内的两个可能的候选键之间不能相互依赖。

比如学号(学号，身份证号，姓名，年龄)，在这表中学号和身份证号都可以唯一确定某一行，并且，学号决定了身份证号，身份证号决定了学号，存在候选键依赖于候选键的情况，所以这个表不符合BCNF，需要改成(学号，身份证号)和(学号，姓名，年龄)。可以消除删除异常、插入异常和更新异常。

⑤ 第四范式(4NF)：设R是一个关系模式，D是R上的多值依赖集合。如果D中存在非平凡多值依赖X🡪Y时，X必是R的超键，那么称R是第四范式的模式。

即第四范式要求不能出现有多个多值事实。

比如用户表(用户ID，固定电话，手机)，在这个表中一个用户可能有多个固定电话，也可能有多个手机，所以这个表就不符合第四范式，因为存在两个列都有多值事实。把这个表改成表1(用户ID，固定电话)和表2(用户ID，手机)就可以了。

⑥ 第五范式(5NF)

第四范式处理的是相互独立的多值情况，而第五范式则处理相互依赖的多值情况。

有一个销售信息表SALES（销售人员，供应商，产品）。SALEPERSON代表销售人员，在某些情况下，这个表中会产生一些冗余。可以将表分解成表1（销售人员，供应商）；表2（销售人员，产品）；表3（供应商，产品）。

1. MySQL数据库锁

（1）行级锁定：

优点：锁定颗粒度很小，所以发生锁定资源争用的概率很小，挺高了并发性。

缺点：获取锁释放锁需要做更多的事情，容易发生死锁。

（2）表级锁定：

优点：锁定资源大，获取释放锁速度快，不容易发生死锁。

缺点：锁定资源争用的概率大。

1. 数据库中各个键的区别

超键(super key):在关系中能唯一标识元组的属性集称为关系模式的超键

候选键(candidate key):不含有多余属性的超键称为候选键

主键(primary key):用户选作元组标识的一个候选键做为程序主键

搜索键:用于在文件中查找记录的属性或属性值，在索引中用到的词。

比如一个小范围的所有人，没有重名的，考虑以下属性：

身份证、姓名、性别、年龄。

身份证 唯一 所以是一个超键

姓名 唯一 所以是一个超键

姓名，性别） 唯一 所以是一个超键

姓名，性别，年龄） 唯一 所以是一个超键

--这里可以看出，超键的组合是唯一的，但可能不是最小唯一的

身份证 唯一而且没有多余属性 所以是一个候选键

姓名 唯一而且没有多余属性 所以是一个候选键

--这里可以看出，候选键是没有多余属性的超键

15. Rownum 和 Rowid

Rowid和Rownum对于数据库开发人员来说基本很少用到，因为在企业数据库开发中大多都是进行数据批处理，但是对于其他数据库人员来说还是会用到的。

Rowid和Rownum都是虚列，但含义完全不同。Rowid是物理地址，用于定位oracle中具体数据的物理存储位置，而Rownum则是sql的输出结果排序。通俗的讲：Rowid是相对不变的，Rownum会变化，尤其是使用order by的时候。

1. 如果Mysql单个表的数据量很大，导致查询变慢，怎样优化？

① 对经常查询的列添加索引；

② 把经常查询的内容做个缓存；

③ hash分表，分区。

1. Oracle大数据量时的优化

http://www.cnblogs.com/easypass/archive/2010/12/08/1900127.html

* + 1. 编写优良的程序代码
    2. 对海量数据进行分区操作
    3. 建立缓存机制
    4. 优化查询SQL语句
    5. 避免使用耗费资源的操作:

带有DISTINCT,UNION,MINUS,INTERSECT的SQL语句会启动SQL引擎 执行耗费资源的排序(SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要执行两次排序. 通常, 带有UNION, MINUS , INTERSECT的SQL语句都可以用其他方式重写. 如果你的数据库的SORT\_AREA\_SIZE调配得好, 使用UNION , MINUS, INTERSECT也是可以考虑的, 毕竟它们的可读性很强

* + 1. 通常来说，如果语句能够避免子查询的使用，就尽量不用子查询。因为子查询的开销是相当昂贵的。
    2. 分页处理；
    3. 优化业务逻辑；
    4. 只返回需要的数据。

1. 分页查询

在我的项目中用到oracle数据库中的查询如下:

例子1：

String sql = "select \* from \"tbl\_SDB\_Bike\" ,\"tbl\_SDB\_Register\" where \"Bike\_eid\" = \"Register\_bikeEid\" order by \"Register\_time\" DESC ";

sql = "SELECT \* FROM ( SELECT A.\*, ROWNUM RN FROM ( " +sql+" ) A WHERE ROWNUM <= "+(pageIndex)\*pageSize+") WHERE RN >"+(pageIndex-1)\*pageSize;

例子2：

String sql = "select \* from \"tbl\_SDB\_Bike\" ,\"tbl\_SDB\_Register\" "

+ " where \"Bike\_eid\"=\"Register\_bikeEid\" order by \"Register\_time\" DESC ";

sql = "SELECT \* FROM ( SELECT A.\*, ROWNUM RN FROM ( " +sql+" ) A WHERE ROWNUM <= "+(pageIndex)\*pageSize+") WHERE RN >"+(pageIndex-1)\*pageSize;

MySql数据库中limit的用法：

select \* from table limit m,n

其中m是指记录开始的index，从0开始，表示第一条记录

n是指从第m+1条开始，取n条。

select \* from tablename limit 2,4

即取出第3条至第6条，4条记录

1. 回收站功能

为了防止删除表时误删除数据，Oracle数据库增加了回收站功能。表被删除后，并没有真正的从数据库中删除掉，而是将其重新命名，并将其相关系统信息存入回收站。如果发生了误删的操作，就可以把它恢复过来。

Oracle数据库其实就是一个数据字典库，其中存储了删除的数据库对象信息。

但是需要注意的是sys与system用户删除的对象，会直接删除，而不会再被放到回收站中。当表空间的空闲空间不够时，回收站中的对象所占用的空间会自动释放掉，分配给需要空间的其他对象。

1. 表空间

表空间可以看作是放置表的容器。表空间由一个或多个数据文件构成。

创建表空间的命令：Create tablespace tbs datafile ‘e:\oradata\tbs01.dbf’ size 10m autoextend on next 1m;

在创建表空间的时候可以指定数据文件的路径、文件初始大小、是否自增、大小是否有限制等。

表空间也可以由多个数据文件构成可以使得表或索引的数据存储在多个文件上，这样可以带来的好处有：

① 一个数据文件若分布于不同磁盘上，可以实现数据并行读写；

② 进行备份恢复操作时，可以对各个数据文件分期分批进行；

③ 可以根据数据文件中的数据重要性，只备份部分文件。

表空间可以分为系统表空间、undo表空间、临时表空间、用户表空间。

1. 数据库、区和区

数据块(data block)是Oracle数据库中最小的逻辑存储单位，在其他DBMS中也称为数据页(data page)。表、索引等对象的数据即存储于数据块中。

下一层次的逻辑存储单位是区(extent)，是Oracle数据库对象分配存储空间的单位，每个区由若干个物理上连续的数据库构成，表或索引等对象需要新的存储空间时，Oracle每次对其分配一个区的空间。

最后一个层次的逻辑存储单位是段(Segment)。对于一般的表或索引，都同时存在与其对应的段。对于表或索引，本质上，段与其对应的表或索引描述的是同样的一堆数据，只是表或索引强调的是逻辑数据，如表包含的行、列等属性，而段强调的是其分配到的存储空间属性，如区、数据库等属性。