本文内容涉及到基本 SQL 语法，数据的基本存储原理，数据库一些概念、数据优化等。抱砖引玉，权当一个综合复习！

常见面试题目：

0. 基本 SQL 语法题目，在 正文“基础 SQL 语法”中有 13 道题，这里就略过了。
1. 索引的作用？她的优点缺点是什么？
2. 介绍存储过程基本概念和 她的优缺点？
3. 使用索引有哪些需要注意的地方？
4. 索引碎片是如何产生的？有什么危害？又该如何处理？
5. 锁的目的是什么？
6. 锁的粒度有哪些？
7. 什么是事务？什么是锁？
8. 视图的作用，视图可以更改么？
9. 什么是触发器(trigger)？触发器有什么作用？
10. SQL 里面 IN 比较快还是 EXISTS 比较快？
11. 维护数据库的完整性和一致性，你喜欢用触发器还是自写业务逻辑?为什么？

基础 SQL 语法

以下 SQL 所使用的实例数据库为 Sqlite（因为相当轻量），数据库文件（下载链接,test.db,6KB）,SQLite 数据库管理工具推荐 SQLite Expert Personal。

0. 创建表

定义如下表结构，后面的题目都以此表结构为依据。

```
Student(ID,Name,Age,Sex) 学生表
Course(ID,Name,TeacherID) 课程表
Score(StudentID,CourseID,Score) 成绩表
Teacher(ID,Name) 教师表
```

创建表的语法很简单，SQL 语句：

```
CREATE TABLE [Student] (
  [ID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  [Name] NVARCHAR(20),
  [Age] INT,
  [Sex] INT);

CREATE TABLE [Course] (
  [ID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  [Name] NVARCHAR(20),
  [TeacherID] INT)

CREATE TABLE [Score] (
  [Score] double,
  [StudentID] INT,
  [CourseID] INT)

CREATE TABLE [Teacher] (
  [ID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  [Name] NVARCHAR(20))
```

1. 查询语文“1”比数学“2”课程成绩高的所有学生的姓名

这是一个嵌套查询的题目，考察对子查询的使用，子查询结果作为一个集合可以当做一个独立的表来看待，子查询必须用括号括起来：

```
select st.[Name],c1.Score,c2.Score from
(select sc.[Score], sc.StudentID from Score sc  where sc.[CourseID]=1)c1,
(select sc.[Score],sc.StudentID from Score sc where sc.[CourseID]=2)c2
```

```
join Student st on st.[ID]= c1.[StudentID]
where c1.[Score]>c2.[Score] and c1.[StudentID]==c2.[StudentID]
```

2. 查询平均成绩大于 60 分的同学的学号和平均成绩

GROUP BY 语句用于结合合计函数，根据一个或多个列对结果集进行分组。GROUP BY 子句在 SELECT 语句的 WHERE 子句之后并 ORDER BY 子句之前。WHERE 关键字无法与合计函数一起使用，GROUP BY 后面不能接 WHERE 条件，使用 HAVING 代替。

```
select sc.[StudentID],avg(sc.Score) from Score sc
group by sc.[CourseID] having avg(sc.Score)>60
```

3. 查询所有同学的学号、姓名、选课数、总成绩；

```
select st.[ID],st.Name,count(sc.CourseID),sum(sc.Score) from Student st
left outer join Score sc on sc.[StudentID]=st.[ID]
group by st.[ID]
```

外连接的三种形式如下表，其中 outer 可以省略。与外连接对应的就是内连接 inner join ，要两个表同时满足指定条件。

联接类型	保留数据行
A left outer join B	all A rows
A right outer join B	all B rows
A full outer join B	all A and B rows

4. 查询姓“张”的老师的个数；

```
select count(t.ID) from Teacher t where t.Name like '张%'
```

SQL LIKE 子句使用通配符运算符比较相似的值。符合 LIKE 操作符配合使用 2 个通配符：

- 百分号（%）：百分号代表零个，一个或多个字符
- 下划线（_）：下划线表示单个数字或字符

5. 找出教师表中姓名重复的数据，然后删除多余重复的记录，只留 ID 小的那个。

```
select t.Name,count(t.Name) from Teacher t group by t.[Name] having count(t.Name)>1
```

删除多余的记录，写这种稍微复杂一点的 sql 的时候，要学会拆解，此题可以拆解为三个部分（删除+重复数据+重复数据中 ID 最小的数据），先分别把 3 个部分的 sql 写了，然后再一步步合并，这样就轻松多了。

```
delete from Teacher where Name in
      (select t2.Name from Teacher t2 group by t2.[Name] having count(t2.Name)>1)
and ID not in
      (select min(t3.ID) from Teacher t3 group by t3.Name having count(t3.Name)>1)
```

6. 按照成绩分段标示（<60 不及格，60-80 良，>80 优），输出所有学生姓名、课程名、成绩、成绩分段标示。

```
select st.Name,c.Name,sc.Score,(case
      when sc.Score > 80 then '优'
      when sc.Score < 60 then '不及格'
      else '良' end) as 'Remark'
from Score sc
inner join Student st on st.ID=sc.[StudentID]
inner join Course c on c.ID=sc.[CourseID]
```

1	张三	语文	50	不及格
2	张三	数学	75	良
3	李四	语文	66	良
4	李四	数学	90	优
5	小美	语文	88	优
6	小美	数学	55	不及格

7. 查询所有课程成绩小于 60 分的同学的学号、姓名信息

这个比较简单，下面给出了两种方法，使用 join 链接和子查询：

```
select distinct st.* from Student st
left join Score sc on sc.[StudentID]=st.ID
where sc.[Score]<60
-- --
select * FROM Student st WHERE st.ID in
      (SELECT s1.ID FROM Student s1,Score s2 WHERE s1.ID=s2.[StudentID] AND s2.Score<60)
```

8. 查询各科成绩最高和最低的分：以如下形式显示：课程名称，最高分，最低分

```
select c.Name as 课程,max(sc.Score) as 最高分,min(sc.Score) as 最低分 from Score sc
left join Course c on c.ID=sc.[CourseID]
group by sc.[CourseID]
```

9. 查询不同老师所教不同课程平均分从高到低显示

```
select t.Name,c.Name, avg(sc.Score) from Score sc,Teacher t,Course c
where sc.[CourseID]=c.ID and c.TeacherID=t.ID
group by sc.[CourseID] order by avg(sc.Score) desc
```

Name	Name_1	avg(sc.Score)
Click here to define a filter		
王老师	数学	73.3333333333333
张老师	语文	68

10. 查询和“1”号的同学学习的课程完全相同的其他同学学号和姓名

```
select s1.StudentID from Score s1 where s1.CourseID in(select s2.CourseID from Score s2 where s2.StudentID=1)
group by s1.StudentID having count(*)=(select count(*) from Score s3 where s3.StudentID=3)
```

11. 查询选修“张老师”老师所授课程的学生中，成绩最高的学生姓名及其成绩

```
select t.Name,c.Name,s1.Score from Score s1,Teacher t,Course c
      where t.ID=c.[TeacherID] and s1.CourseID=c.ID and t.Name='张老师'
      and s1.Score=(select max(Score) from Score where CourseID=c.ID)
```

Name	Name_1	Score
ick here to define a filter		
张老师	语文	99
张老师	体育	61

注意：多表连接查询有多种写法，比如本题目中多表连接可以有以下两种方式：

```
select t.Name,c.Name,s1.Score from Score s1,Teacher t,Course c where t.ID=c.[TeacherID] and s1.CourseID=c.ID
-- 下面的写法和上面的效果是一样的！ --
select t.Name,c.Name,s1.Score from Score s1
join Teacher t on t.ID=c.[TeacherID]
join Course c on s1.CourseID=c.ID
```

FROM TABLE1,TABLE2…效果等效于 FROM TABLE1 join TABLE2…，都是内连接 inner 操作。详细的 SQL 表连接操作可以参考：深入理解 SQL 的四种连接-左外连接、右外连接、内连接、全连接

12. 查询所有成绩第二名到第四名的成绩

```
select  s.[StudentID],s.Score from Score s order by s.Score desc limit 2 offset 2
-- SQL 2005/2008 中的分页函数是 ROW_NUMBER() Over (Order by 列...)--
select  t.[StudentID],t.Score from(
       select s2.[StudentID],s2.Score,ROW_NUMBER() OVER (ORDER BY s2.[Score]) AS rn from Score s2) t
where t.rn>=2 and t.rn<=4
```

这是一个分页的题目，上面这是 Sqlite 提供的内置方法 limite 进行分页，不同数据库的分页方式又有些差别，但都大同小异。基本的过程都是先根据条件查询所需数据（加上行号），然后再此基础上返回指定行区间段的数据。其实 SQlServer 也很简单，不同的版本也有些不同，可以参考：SQL Server 常用分页 SQL

12. 查询各科成绩前 2 名的记录:(不考虑成绩并列情况)

```
select * from Score s1 where s1.Score
```

```
      in(select s2.Score from Score s2 where s1.[CourseID]=s2.[CourseID] order by s2.Score desc limit 2 offset
0)
order by s1.[CourseID],s1.[Score] desc
-- 上面是 sqlite 中的语法，sqlite 中没有 top，使用 limit 代替，效果是一样的 --
select * from Score s1 where s1.Score
      in(select Top 2 s2.Score from Score s2 where s1.[CourseID]=s2.[CourseID] order by s2.Score desc)
order by s1.[CourseID],s1.[Score] desc
```

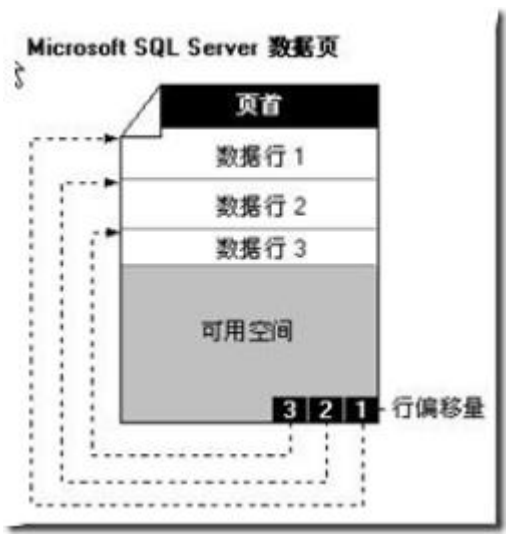
Score	StudentID	CourseID
lick here to define a filter		
99	4	1
88	3	1
90	2	2
75	1	2
61	4	3

数据库基本存储原理

基本存储单元——页

数据库文件存储是以页为存储单元的，一个页是 8K（8192Byte），一个页就可以存放 N 行数据。我们表里的数据都是存放在页上的，这种叫数据页。还有一种页存放索引数据的，叫索引页。

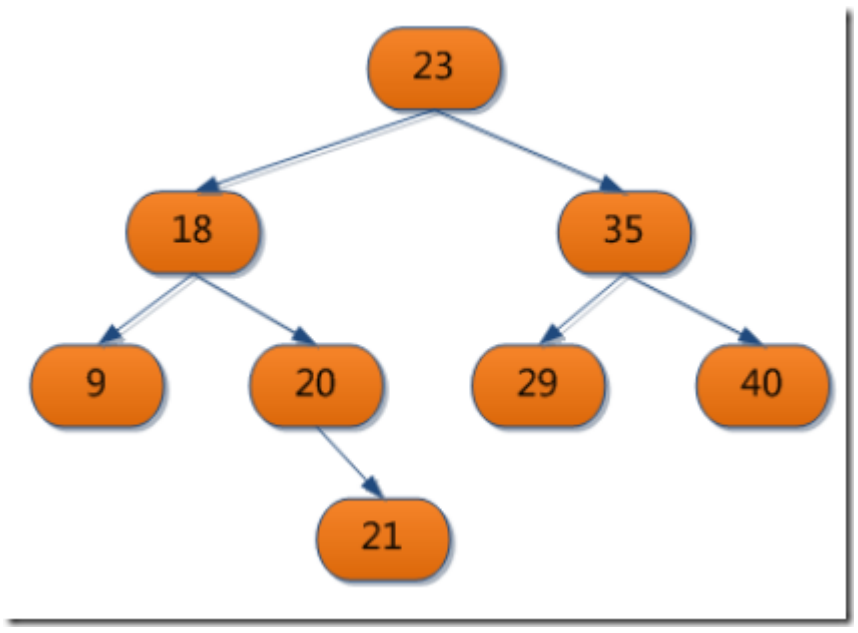
同时，页也是 IO 读取的最小单元（物理 IO 上不是按行读取），也是所有权的最小单位。如果一页中包含了表 A 的一行数据，这页就只能存储表 A 的行数据了。或是一页中包含了索引 B 的条目，那这页也仅仅只能存储索引 B 的条目了。每页中除去存储数据之外，还存储一些页头信息以及行偏移以便 SQL Server 知道具体每一行在页中的存储位置。



数据库的基本物理存储单元是页，一个表由很多个页组成，那这些页又是如何组织的呢？我们一般都会对表创建索引，这些索引又是如何存储的呢？不要走开，请看下文。

表/索引的存储结构

如下图，是一个 B 树（二叉搜索树）的示例，都是小的元素放左边，大的元素放右边，依次构造的，比如要查找元素 9，从根节点开始，只要比较三次就找到他了，查询效率是非常高的。



B+树和 B-树都是 B 树的变种，或者说是更加高级的复杂版本（关于 B 树的资料，有兴趣可以自己去学习，这里只是抛砖引玉）。B+树和 B-树是数据库中广泛应用的索引存储结构，它可以极大的提高数据查找的效率。前面说了数据库存储的基本单元是页，因此，索引树上的节点就是页了。

因此不难看出，索引的主要优点和目的就是为了提高查询效率。

为了保证数据的查询效率，当新增、修改、删除数据的时候，都需要维护这颗索引树，就可能会出现分裂、合并节点（页）的情况（这是树的结构所决定的，想要更好理解这一点，可以尝试自己代码实现一下 B-树 B+树）。好！重点来了！

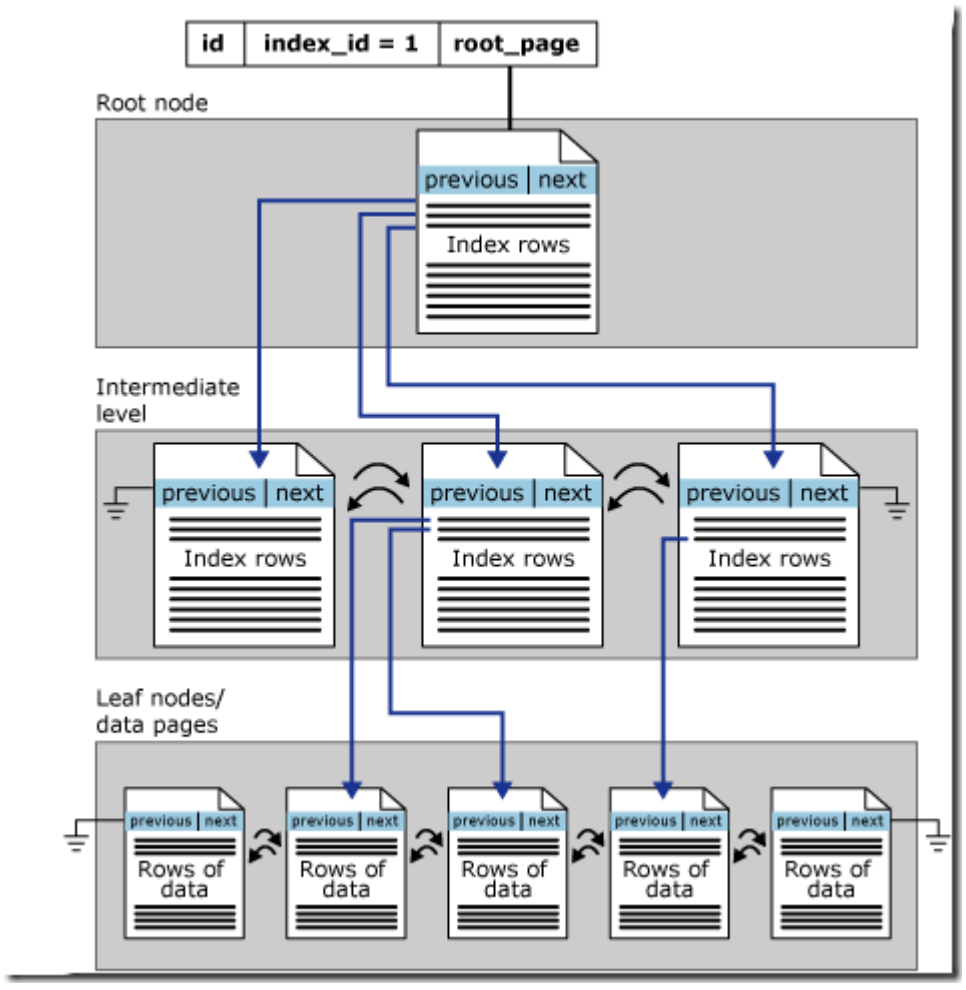
索引的缺点：

- 当新增、修改、删除数据的时候，需要维护索引树，有一定的性能影响；
- 同上面，在频繁的树维护过程中，B 树的页拆分、合并会造成大量的索引碎片，又会极大的印象查询效率 ，因此索引还需要维护；
- 非聚集索引需要额外的存储空间，不过这个一般问题都不是很大，但是需要注意的一个问题；

聚集索引

聚集索引决定了表数据的物理存储顺序，也就是说表的物理存储是根据聚集索引结构进行顺序存储的，因此一个表只能有一个聚集索引。如下图，就是一个聚集索引的树结构：

- 所有数据都在叶子节点的页上，在叶子节点（数据页）之间有一个链指针，这是 B+树的特点；
- 非叶子节点都是索引页，存储的就是聚集索引字段的值；
- 表的物理存储就是依据聚集索引的结构的，一个表只能有一个聚集索引；

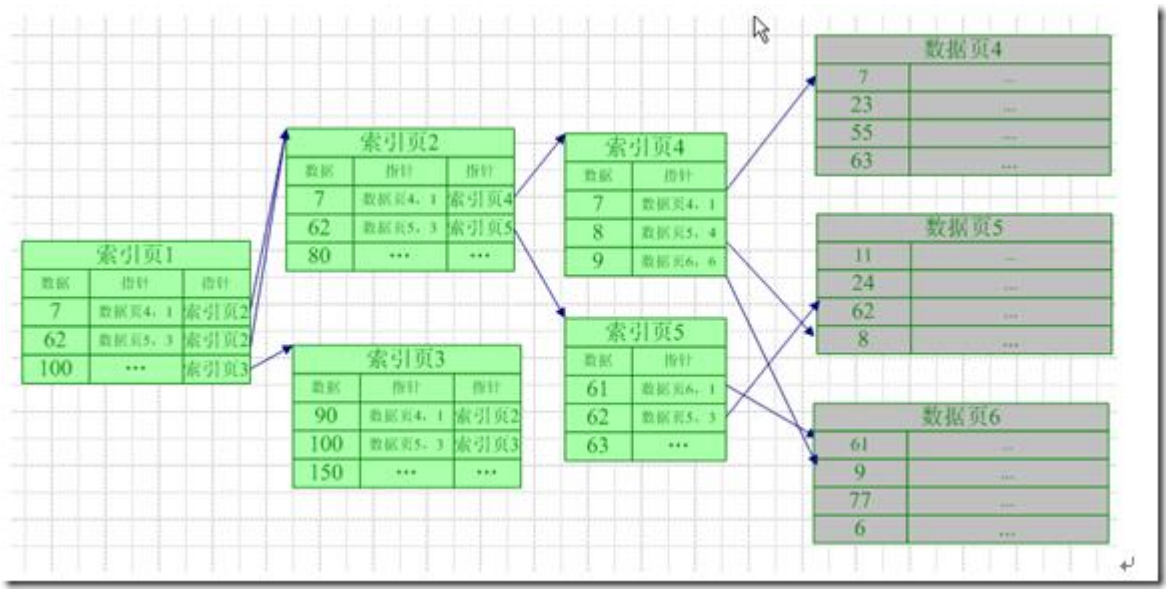


聚集索引的所有的数据都存储在叶子节点上，数据查询的复杂度都是一样的（树的深度），按照聚集索引列查找数据效率是非常高的。上面说了，聚集索引决定了表的物理存储结构，那如果没有创建聚集索引，会如何呢？——**表内的所有页都无序存放，是一个无序的堆结构**。堆数据的查询就会造成表扫描，性能是非常低的。

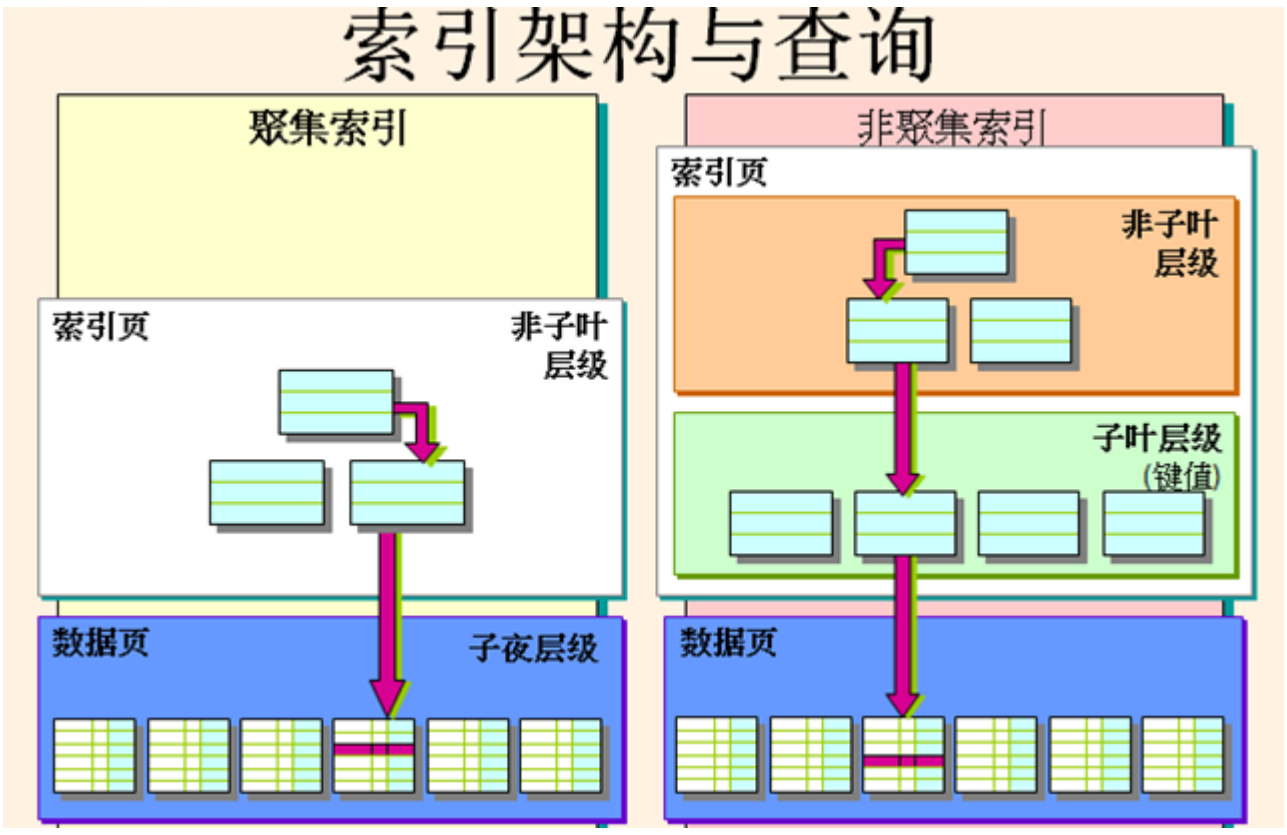
因此聚集索引的的重要性不言而喻，一般来说，大多会对主键建立聚集索引，大多数普通情况这么做也可以。但实际应用应该尊从一个原则就是“频繁使用的、排序的字段上创建聚集索引”

非聚集索引

除了聚集索引以外的其他索引，都称之为非聚集索引，非聚集索引一般都是为了优化特定的查询效率而创建的。非聚集索引也是 B 树（B+树和 B-树）的结构，与非聚集索引的存储结构唯一不一样的，就是非聚集索引中不存储真正的数据行，因为在聚集索引中已经存放了所有数据，非聚集索引只包含一个指向数据行的指针即可。



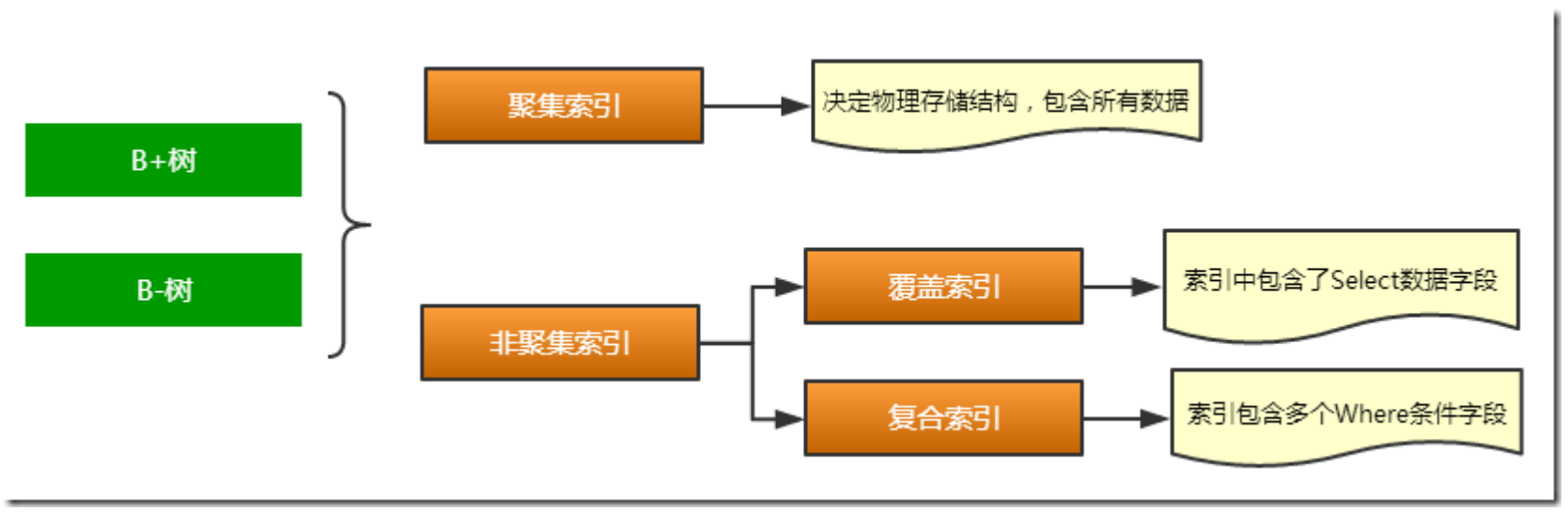
- 非聚集索引的创建会单独创建索引文件来存储索引结构，会占用一定存储空间，就是用空间换时间；
- 非聚集索引的目的很单纯：提高特定条件的查询效率，一个表有可能根据多种查询需求创建多个非聚集索引；



数据查询 SQL 简单来看，分为两个部分：SELECT****　和　Where ****，因此索引的创建也是根据这两部分来决定的。根据这两点，有两种主要的索引形式：复合索引和覆盖索引，在实际使用中，根据具体情况可能都会用到，只要能提高查询效率就是好索引。

覆盖索引：就是在索引中包含的数据列（非索引列，SELECT 需要的列），这样在使用该索引查询数据时就不会再进行键查找（也叫书签查找）了。

复合索引：主要针对 Where 中有多个条件的情况，索引包含多个数据列。在使用复合索引时，应注意多个索引键的顺序问题，这个是会影响查询效率的，一般的原则是唯一性高的放前面，还有就是 SQl 语句中 Where 条件的顺序应该和索引顺序一致。



索引碎片

前面说过了，索引在使用一段时间后（主要是新增、修改、删除数据，如果该页已经存储满了，就要进行页的拆分，频繁的拆分，会产生较多的索引碎片）会产生索引碎片，这就造成了索引页在磁盘上存储的不连续。会造成磁盘的访问使用的是随机的 i/o，而不是顺序的 i/o 读取，这样访问索引页会变得更慢。如果碎片过多，数据库是可会能不使用该索引的（她嫌弃你太慢了，数据库会选择一个更优的执行计划）。

解决这个问题主要是两种方法：

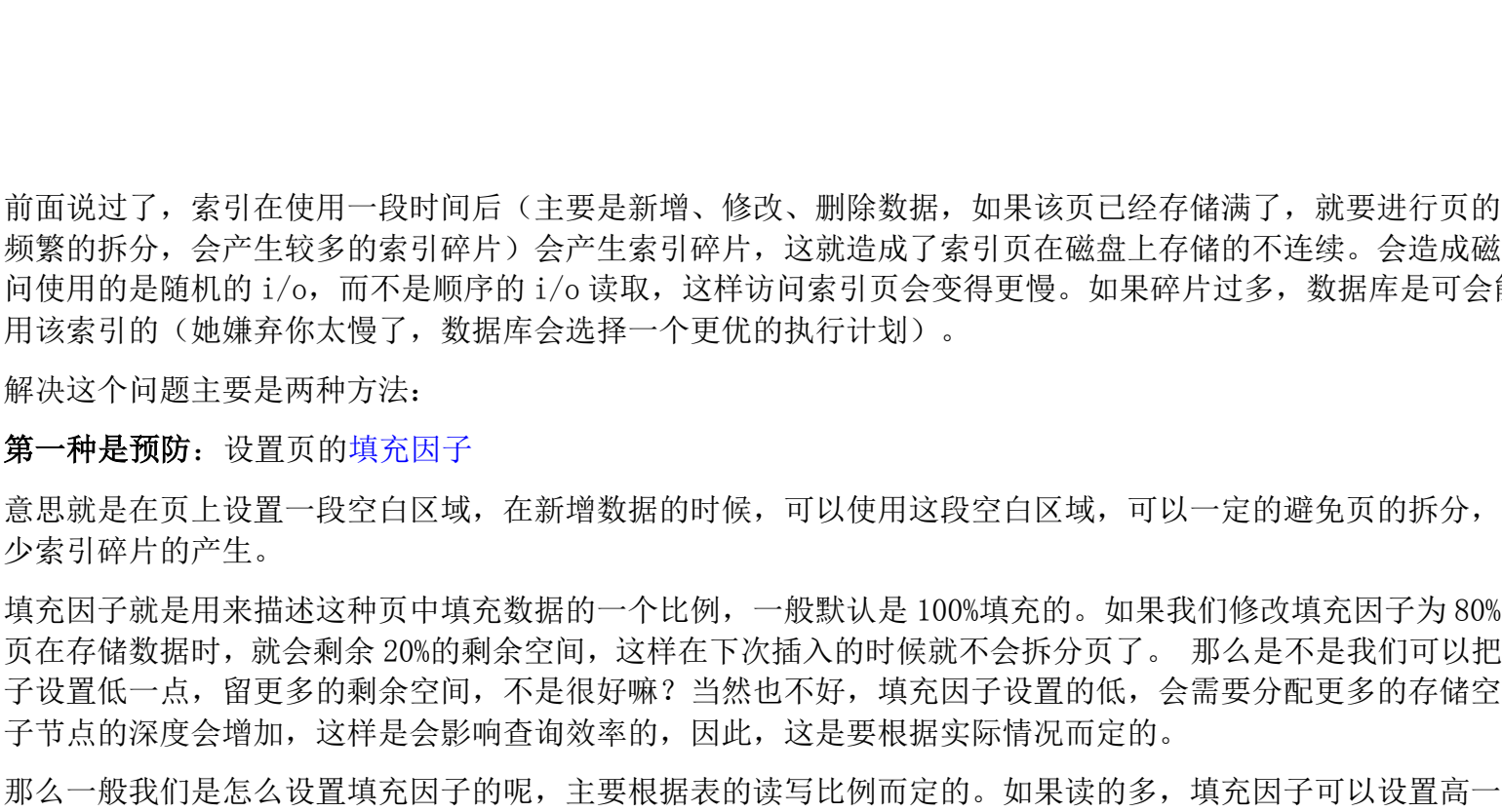
第一种是预防：设置页的填充因子

意思就是在页上设置一段空白区域，在新增数据的时候，可以使用这段空白区域，可以一定的避免页的拆分，从而减少索引碎片的产生。

填充因子就是用来描述这种页中填充数据的一个比例，一般默认是 100%填充的。如果我们修改填充因子为 80%，那么页在存储数据时，就会剩余 20%的剩余空间，这样在下次插入的时候就不会拆分页了。 那么是不是我们可以把填充因子设置低一点，留更多的剩余空间，不是很好嘛？当然也不好，填充因子设置的低，会需要分配更多的存储空间，叶子节点的深度会增加，这样是会影响查询效率的，因此，这是要根据实际情况而定的。

那么一般我们是怎么设置填充因子的呢，主要根据表的读写比例而定的。如果读的多，填充因子可以设置高一点，如 100%，读写各一半，可以 80~90%；修改多可以设置 50~70%。

第二种是索引修复：定期对索引进行检查、维护，写一段 SQL 检查索引的碎片比例，如果碎片过多，进行碎片修复或重建，定期执行即可。具体可以参考本文末尾的相关参考资料。



索引使用总结

- 创建索引的的字段尽量小，最好是数值，比如整形 int 等；
- 对于频繁修改的字段，尽量不要创建索引，维护索引的成本很高，而且更容易产生索引碎片；
- 定期的索引维护，如索引碎片的修复等；
- 不要建立或维护不必要的重复索引，会增加修改数据（新增、修改、删除数据）的成本；
- 使用唯一性高的字段创建索引，切不可在性别这样的低唯一性的字段上创建索引；
- 在 SQL 语句中，尽量不要在 Where 条件中使用函数、运算符或表达式计算，会造成索引无法正常使用；
- 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；
- 应尽量避免在 where 子句中使用!=或<>操作符，否则将导致引擎放弃使用索引而进行全表扫描；

事务与锁

事务就是作为一个逻辑工作单元的 SQL 语句，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上个节点。为了确保要么执行，要么不执行，就可以使用事务。而锁是实现事务的关键，锁可以保证事务的完整性和并发性。

这部分的理论性太强了，不如看看下文章末尾的参考资料更好（博客园的高质量文章还是相当多的），简单总结一下就好了！

和在.NET 中的锁用途类似，数据库中的锁也是为了解决在并发访问时出现各种冲突的一种机制。

题目答案解析:

1. 索引的作用？和它的优点缺点是什么？

索引就一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。索引很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。缺点是它减慢了数据录入的速度，同时也增加了数据库的尺寸大小。

2. 介绍存储过程基本概念和 她的优缺点

存储过程是一个预编译的 SQL 语句，他的优点是允许模块化的设计，也就是说只需创建一次，在该程序中就可以调用多次。例如某次操作需要执行多次 SQL，就可以把这个 SQL 做一个存储过程，因为存储过程是预编译的，所以使用存储过程比单纯 SQL 语句执行要快。缺点是可移植性差，交互性差。

3. 使用索引有哪些需要注意的地方？

- 创建索引的的字段尽量小，最好是数值，比如整形 int 等；
- 对于频繁修改的字段，尽量不要创建索引，维护索引的成本很高，而且更容易产生索引碎片；
- 定期的索引维护，如索引碎片的修复等；
- 不要建立或维护不必要的重复索引，会增加修改数据（新增、修改、删除数据）的成本；
- 使用唯一性高的字段创建索引，切不可在性别这样的低唯一性的字段上创建索引；
- 在 SQL 语句中，尽量不要在 Where 条件中使用函数、运算符或表达式计算，会造成索引无法正常使用；
- 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；
- 应尽量避免在 where 子句中使用!=或<>操作符，否则将导致引擎放弃使用索引而进行全表扫描；

4. 索引碎片是如何产生的？有什么危害？又该如何处理？

索引在使用一段时间后（主要是新增、修改、删除数据，如果该页已经存储满了，就要进行页的拆分，频繁的拆分，会产生较多的索引碎片）会产生索引碎片。

索引碎片会严重印象数据的查询效率，如果碎片太多，索引可能不会被使用。

碎片的处理方式主要有两种：

第一种是预防：设置页的填充因子

意思就是在页上设置一段空白区域，在新增数据的时候，可以使用这段空白区域，可以一定的避免页的拆分，从而减少索引碎片的产生。

填充因子就是用来描述这种页中填充数据的一个比例，一般默认是 100%填充的。如果我们修改填充因子为 80%，那么页在存储数据时，就会剩余 20%的剩余空间，这样在下次插入的时候就不会拆分页了。 那么是不是我们可以把填充因子设置低一点，留更多的剩余空间，不是很好嘛？当然也不好，填充因子设置的低，会需要分配更多的存储空间，叶子节点的深度会增加，这样是会影响查询效率的，因此，这是要根据实际情况而定的。

那么一般我们是怎么设置填充因子的呢，主要根据表的读写比例而定的。如果读的多，填充因子可以设置高一点，如 100%，读写各一半，可以 80~90%；修改多可以设置 50~70%。

第二种是索引修复：定期对索引进行检查、维护，写一段 SQL 检查索引的碎片比例，如果碎片过多，进行碎片修复或重建，定期执行即可。具体可以参考本文末尾的相关参考资料。

5. 锁的目的是什么？

主要解决多个用户同时对数据库的并发操作时会带来以下数据不一致的问题：

- 丢失更新，同时修改一条数据
- 读脏，A 修改了数据后，B 读取后 A 又取消了修改，B 读脏
- 不可重复读，A 用户读取数据,随后 B 用户读取该数据并修改,此时 A 用户再读取数据时发现前后两次的值不一致
- 还有一种是幻读，这个情况好像不多。

并发控制的主要方法是封锁,锁就是在一段时间内禁止用户做某些操作以避免产生数据不一致

6. 锁的粒度有哪些？

- 数据库锁：锁定整个数据库，这通常发生在整个数据库模式改变的时候。
- 表锁：锁定整个表，这包含了与该表相关联的所有数据相关的对象，包括实际的数据行(每一行)以及与该表相关联的所有索引中的键。
- 区段锁：锁定整个区段，因为一个区段由 8 页组成，所以区段锁定是指锁定控制了区段、控制了该区段内 8 个数据或索引页以及这 8 页中的所有数据行。
- 页锁：锁定该页中的所有数据或索引键。
- 行或行标识符：虽然从技术上将，锁是放在行标识符上的，但是本质上，它锁定了整个数据行。

7. 什么是事务？什么是锁？

事务就是被绑定在一起作为一个逻辑工作单元的 SQL 语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上个节点。为了确保要么执行，要么不执行，就可以使用事务。要将所有组语句作为事务考虑，就需要通过 ACID 测试，即原子性，一致性，隔离性和持久性。
锁是实现事务的关键，锁可以保证事务的完整性和并发性。

8. 视图的作用，视图可以更改么？

视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询；不包含任何列或数据。使用视图可以简化复杂的 sql 操作，隐藏具体的细节，保护数据；视图创建后，可以使用与表相同的方式利用它们。

视图的目的在于简化检索，保护数据，并不用于更新。

9. 什么是触发器(trigger)？触发器有什么作用？

触发器是数据库中由一定时间触发的特殊的存储过程，他不是由程序掉用也不是手工启动的。触发器的执行可以由对一个表的 insert,delete, update 等操作来触发，触发器经常用于加强数据的完整性约束和业务规则等等。

10. SQL 里面 IN 比较快还是 EXISTS 比较快？

这个题不能一概而论,要根据具体情况来看。IN 适合于外表大而内表小的情况;EXISTS 适合于外表小而内表大的情况。

如果查询语句使用了 not in，那么对内外表都进行全表扫描，没有用到索引；而 not exists 的子查询依然能用到表上的索引。所以无论哪个表大，用 not exists 都比 not in 要快。参考资料：
http://www.cnblogs.com/seasons1987/archive/2013/07/03/3169356.html

11. 维护数据库的完整性和一致性，你喜欢用触发器还是自写业务逻辑?为什么？

尽可能使用约束，如 check、主键、外键、非空字段等来约束。这样做效率最高，也最方便。其次是使用触发器，这种方法可以保证，无论什么业务系统访问数据库都可以保证数据的完整新和一致性。最后考虑的是自写业务逻辑，但这样做麻烦，编程复杂，效率低下。

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

参考资料:

书籍：CLR via C#

书籍：你必须知道的.NET

SQL 常考笔试题[转]
深入理解 SQL 的四种连接-左外连接、右外连接、内连接、全连接
博主以前写的：数据库索引及基本优化入门
SQL Server 索引的维护 - 索引碎片、填充因子〈第三篇〉
SQL Server 锁
SQL Server 事务语法
SQL Server 中的事务与锁

分类: C#.NET, 数据库(DataBase)