- **MySQL 篇**

9.1.0 主键 超键 候选键 外键

9.1.1 数据库事务的四个特性及含义

参考答案：

数据库事务 transanction 正确执行的四个基本要素。ACID,原子性(Atomicity)、一致性(Correspondence)、隔离性(Isolation)、持久性(Durability)。

- 原子性:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
- 一致性:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。
- 隔离性:隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。
- 持久性:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

9.1.2 视图的作用，视图可以更改么？

参考答案：

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。

视图不能被索引，也不能有关联的触发器或默认值，如果视图本身内有 order by 则对视图再次 order by 将被覆盖。

创建视图：create view XXX as XXXXXXXXXXXXXX;

对于某些视图比如未使用联结子查询分组聚集函数 Distinct Union 等，是可以对其更新的，对视图的更新将对基表进行更新；但是视图主要用于简化检索，保护数据，并不用于更新，而且大部分视图都不可以更新。

9.1.3 drop,delete 与 truncate 的区别

参考答案：

drop 直接删掉表 truncate 删除表中数据，再插入时自增长 id 又从 1 开始 delete 删除表中数据，可以加 where 字句。

（1） DELETE 语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行进行回滚操作。TRUNCATE TABLE 则一次性地从表中删除所有的数据并不把单独的删除操作记录记入日志保存，删除行是不能恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。

（2） 表和索引所占空间。当表被 TRUNCATE 后，这个表和索引所占用的空间会恢复到初始大小，而 DELETE 操作不会减少表或索引所占用的空间。drop 语句将表所占用的空间全释放掉。

（3） 一般而言，drop > truncate > delete

（4） 应用范围。TRUNCATE 只能对 TABLE；DELETE 可以是 table 和 view

（5） TRUNCATE 和 DELETE 只删除数据，而 DROP 则删除整个表（结构和数据）。

（6） truncate 与不带 where 的 delete ：只删除数据，而不删除表的结构（定义）drop 语句将删除表的结构被依赖的约束（constrain),触发器（trigger)索引（index);依赖于该表的存储过程/函数将被保留，但其状态会变为：invalid。

（7）delete 语句为 DML（data maintain Language),这个操作会被放到 rollback segment 中,事务提交后才生效。如果有相应的 tigger,执行的时候将被触发。

（8） truncate、drop 是 DLL（data define language),操作立即生效，原数据不放到 rollback segment 中，不能回滚

（9） 在没有备份情况下，谨慎使用 drop 与 truncate。要删除部分数据行采用 delete 且注意结合 where 来约束影响范围。回滚段要足够大。要删除表用 drop;若想保留表而将表中数据删除，如果于事务无关，用 truncate 即可实现。如果和事务有关，或老师想触发 trigger,还是用 delete。

（10） Truncate table 表名 速度快,而且效率高,因为: truncate table 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。DELETE 语句每次删除一行，并在事务日志中为所删除的每行记录一项。

TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。

（11） TRUNCATE TABLE 删除表中的所有行，但表结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用 DELETE。如果要删除表定义及其数据，请使用 DROP TABLE 语句。

（12） 对于由 FOREIGN KEY 约束引用的表，不能使用 TRUNCATE TABLE，而应使用不带 WHERE 子句的 DELETE 语句。由于 TRUNCATE TABLE 不记录在日志中，所以它不能激活触发器。

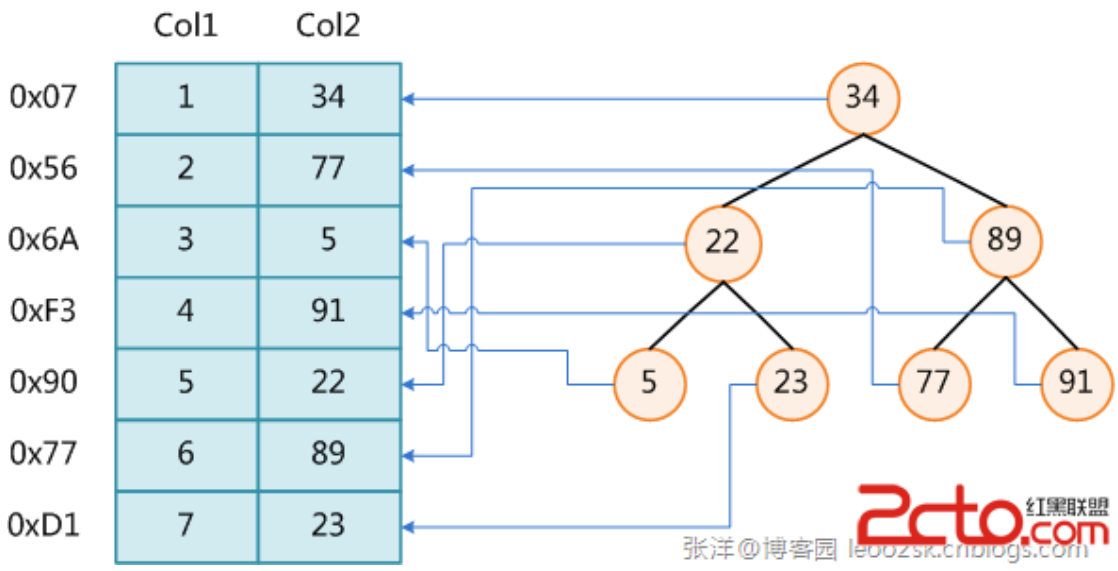
9.1.4 索引的工作原理及其种类

参考答案：

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。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。



图展示了一种可能的索引方式。左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。为了加快 Col2 的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在 O(log2n)的复杂度内获取到相应数据。

创建索引可以大大提高系统的性能。

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。

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。

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。

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，应该在这些列上创建索引：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；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。

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：

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。

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。

根据数据库的功能，可以在数据库设计器中创建三种索引：唯一索引、主键索引和聚集索引。

唯一索引

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在 employee 表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。 主键索引 数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中

使用主键索引时，它还允许对数据的快速访问。 聚集索引 在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

局部性原理与磁盘预读 由于存储介质的特性，磁盘本身存取就比主存慢很多，再加上机械运动耗费，磁盘的存取速度往往是主存的几百分分之一，因此为了提高效率，要尽量减少磁盘 I/O。为了达到这个目的，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。这样做的理论依据是计算机科学中著名的局部性原理：当一个数据被用到时，其附近的数据也通常会马上被使用。程序运行期间所需要的数据通常比较集中。

由于磁盘顺序读取的效率很高（不需要寻道时间，只需很少的旋转时间），因此对于具有局部性的程序来说，预读可以提高 I/O 效率。

预读的长度一般为页（page）的整倍数。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（在许多操作系统中，页得大小通常为 4k），主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后异常返回，程序继续运行。

B-/+Tree 索引的性能分析 到这里终于可以分析 B-/+Tree 索引的性能了。

上文说过一般使用磁盘 I/O 次数评价索引结构的优劣。先从 B-Tree 分析，根据 B-Tree 的定义，可知检索一次最多需要访问 h 个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次 I/O 就可以完全载入。为了达到这个目的，在实际实现 B-Tree 还需要使用如下技巧：

每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个 node 只需一次 I/O。

B-Tree 中一次检索最多需要 h-1 次 I/O（根节点常驻内存），渐进复杂度为 O(h)=O(logdN)。一般实际应用中，出度 d 是非常大的数字，通常超过 100，因此 h 非常小（通常不超过 3）。

而红黑树这种结构，h 明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性，所以红黑树的 I/O 渐进复杂度也为 O(h)，效率明显比 B-Tree 差很多。

综上所述，用 B-Tree 作为索引结构效率是非常高的。

9.1.5 连接的种类

参考答案：

查询分析器中执行：

```
--建表 table1,table2：
create table table1(id int,name varchar(10))
create table table2(id int,score int)
insert into table1 select 1,'lee'
insert into table1 select 2,'zhang'
insert into table1 select 4,'wang'
insert into table2 select 1,90
insert into table2 select 2,100
insert into table2 select 3,70
```

如表:

```
-------------------------------------------------
table1 | table2 |
-------------------------------------------------
id name |id score |
1 lee |1 90|
2 zhang| 2 100|
4 wang| 3 70|
-------------------------------------------------
```

以下均在查询分析器中执行 一、外连接 1.概念：包括左向外联接、右向外联接或完整外部联接

2.左连接：left join 或 left outer join (1)左向外联接的结果集包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值(null)。 (2)sql 语句

```
select * from table1 left join table2 on table1.id=table2.id
-------------结果-------------
idnameidscore
------------------------------
1lee190
2zhang2100
4wangNULLNULL
------------------------------
```

注释：包含 table1 的所有子句，根据指定条件返回 table2 相应的字段，不符合的以 null 显示

3.右连接：right join 或 right outer join (1)右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。 (2)sql 语句

```
select * from table1 right join table2 on table1.id=table2.id
-------------结果-------------
idnameidscore
------------------------------
1lee190
2zhang2100
NULLNULL370
------------------------------
```

注释：包含 table2 的所有子句，根据指定条件返回 table1 相应的字段，不符合的以 null 显示

4.完整外部联接:full join 或 full outer join (1)完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。 (2)sql 语句

```
select * from table1 full join table2 on table1.id=table2.id
-------------结果-------------
idnameidscore
------------------------------
1lee190
2zhang2100
4wangNULLNULL
NULLNULL370
------------------------------
```

注释：返回左右连接的和（见上左、右连接）

二、内连接 1.概念：内联接是用比较运算符比较要联接列的值的联接

2.内连接：join 或 inner join

3.sql 语句

```
select * from table1 join table2 on table1.id=table2.id
-------------结果-------------
idnameidscore
------------------------------
1lee190
2zhang2100
------------------------------
```

注释：只返回符合条件的 table1 和 table2 的列

4.等价（与下列执行效果相同）

```
A:select a.*,b.* from table1 a,table2 b where a.id=b.id
B:select * from table1 cross join table2 where table1.id=table2.id (注：cross join 后加条件只能用 where,不能用 on)
```

三、交叉连接(完全)

1.概念：没有 WHERE 子句的交叉联接将产生联接所涉及的表的笛卡尔积。第一个表的行数乘以第二个表的行数等于笛卡尔积结果集的大小。（table1 和 table2 交叉连接产生 3*3=9 条记录）

2.交叉连接：cross join (不带条件 where...)

3.sql 语句

```
select * from table1 cross join table2
-------------结果-------------
idnameidscore
------------------------------
1lee190
2zhang190
4wang190
1lee2100
2zhang2100
4wang2100
1lee370
2zhang370
4wang370
------------------------------
```

注释：返回 3*3=9 条记录，即笛卡尔积

4.等价（与下列执行效果相同）

```
A:select * from table1,table2
```

9.1.6 数据库范式

参考答案：

1 第一范式（1NF）

在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。 所谓第一范式（1NF）是指数据库表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者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。简而言之，第一范式就是无重复的列。

2 第二范式（2NF）

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键、主码。 第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

3 第三范式（3NF）

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept_id）、部门名称、部门简介等信息。那么在员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。（我的理解是消除冗余）

9.1.7 数据库优化的思路

参考答案：

这个我借鉴了慕课上关于数据库优化的课程。

1.SQL 语句优化

1）应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

2）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，如：

select id from t where num is null 可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：

select id from t where num=0 ========== liueleven 的评论 ========== 不是非我杠精，关于 null,isNull,isNotNull 其实是要看成本的，是否回表等因素总和考虑，才会决定是要走索引还是走全表扫描

也给大家找了一个作者的博文（MySQL 中 IS NULL、IS NOT NULL、!=不能用索引？胡扯！），仅供参考！！！

[zhiyong0804d 的意见] 之所以未把第二条删除还是考虑可能很多人都被误导了。那这样的组织能让大家兼听则明。

3）很多时候用 exists 代替 in 是一个好的选择

4）用 Where 子句替换 HAVING 子句 因为 HAVING 只会在检索出所有记录之后才对结果集进行过滤

2.索引优化 看上文索引

3.数据库结构优化 1）范式优化： 比如消除冗余（节省空间。。）

2）反范式优化：比如适当加冗余等（减少 join） 3）拆分表： 分区将数据在物

理上分隔开，不同分区的数据可以制定保存在处于不同磁盘上的数据文件里。这样，当对这个表进行查询时，只需要在表分区中进行扫描，而不必进行全表扫描，明显缩短了查询时间，另外处于不同磁盘的分区也将对这个表的数据传输分散在不同的磁盘 I/O，一个精心设置的分区可以将数据传输对磁盘 I/O 竞争均匀地分散开。对数据量大的时时表可采取此方法。可按月自动建表分区。

4）拆分其实又分垂直拆分和水平拆分： 案例： 简单购物系统暂设涉及如下表：

1.产品表（数据量 10w，稳定） 2.订单表（数据量 200w，且有增长趋势） 3.用户表 （数据量 100w，且有增长趋势） 以 mysql 为例讲述下水平拆分和垂直拆分，mysql 能容忍的数量级在百万静态数据可以到千万 垂直拆分：解决问题：表与表之间的 io 竞争 不解决问题：单表中数据量增长出现的压力 方案： 把产品表和用户表放到一个 server 上 订单表单独放到一个 server 上 水平拆分： 解决问题：单表中数据量增长出现的压力 不解决问题：表与表之间的 io 争夺 方案：用户表通过性别拆分为男用户表和女用户表 订单表通过已完成和完成中拆分为已完成订单和未完成订单 产品表 未完成订单放一个 server 上 已完成订单表盒男用户表放一个 server 上 女用户表放一个 server 上(女的爱购物 哈哈)

4.服务器硬件优化
这个么多花钱咯！

9.1.8 存储过程与触发器的区别

参考答案：

触发器与存储过程非常相似，触发器也是 SQL 语句集，两者唯一的区别是触发器不能用 EXECUTE 语句调用，而是在用户执行 Transact-SQL 语句时自动触发（激活）执行。触发器是在一个修改了指定表中的数据时执行的存储过程。通常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，触发器主要是通过事件执行触发而被执行的，而

存储过程可以通过存储过程名称名字而直接调用。当对某一表进行诸如 UPDATE、INSERT、DELETE 这些操作时，SQLSERVER 就会自动执行触发器所定义的 SQL 语句，从而确保对数据的处理必须符合这些 SQL 语句所定义的规则。