20180308数据库常问

区分超键、候选键、主键和外键

区分很简单：主要看主键的个数

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：（多个主键+属性）

候选键(candidate key): 不含有多余属性的超键称为候选键。也就是在候选键中，若再删除属性，就不是键了！（多个主键）

主键(primary key): 用户选作元组标识的一个候选键程序主键（只选一个主键）

外键(foreign key)：如果关系模式R中属性K是其它模式的主键，那么k在模式R中称为外键。

咱们创建简单的两个表，说明一下各个键！

学生信息（学号 身份证号 性别 年龄 身高 体重 宿舍号）和 宿舍信息（宿舍号 楼号）

超键：只要含有“学号”或者“身份证号”两个属性的集合就叫超键，例如R1（学号 性别）、R2（身份证号 身高）、R3（学号 身份证号）等等都可以称为超键！

候选键：不含有多余的属性的超键，比如（学号）、（身份证号）都是候选键，又比如R1中学号这一个属性就可以唯一标识元组了，而有没有性别这一属性对是否唯一标识元组没有任何的影响！

主键：**就是用户从很多候选键选出来的一个键就是主键**，比如你要求学号是主键，那么身份证号就不可以是主键了！

外键：宿舍号就是学生信息表的外键

2、什么是事务？什么是锁?

事务就是要么提交，要么不提交（可回滚）。

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，或者是上有个节点。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：在所以的 DBMS 中，锁是实现事务的关键，锁可以保证事务的**完整性和并发性**。与现实生活中锁一样，它可以使某些数据的拥有者，在某段时间内不能使用某些数据或数据结构。当然锁还分级别的。

3、数据库事务的四个特性及含义

ACID

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。

4、什么是视图?

视图是虚拟的表，没有删（有增改查）。

用途： 1- 不希望获得整张表

2- 多表查询

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，试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。

如下两种场景一般会使用到视图：

（1）不希望访问者获取整个表的信息，只暴露部分字段给访问者，所以就建一个虚表，就是视图。

（2）查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异。

注：这个视图是在数据库中创建的 而不是用代码创建的。

5、触发器的作用?

跟踪表的变化，联级触发另一个表或者本表。

触发器是一中特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

6、 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？

先用约束（check，主键，外键，非空字段等来约束）效率最高 >>> 触发器 >>> 最后考虑业务逻辑

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**完整性和一致性**。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。

7、索引的作用？和它的优点缺点是什么？

索引是一种经过排序的数据结构，一般是用B树或者B+树实现的。

创建索引的好处是能快速索引，代价是需要额外的空间和增加、删除等操作时需要额外的维护时间

当频繁查询时，有外键： 用索引

当数据量大，很少查询：不用索引

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)。

创建索引可以大大提高系统的性能（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。

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：

（1）在**经常需要搜索**的列上，可以加快搜索的速度；

（2）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；

（3）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；

（4）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；

（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。

同样，对于有些列不应该创建索引：

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。

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。

8、drop,delete与truncate的区别

drop: 直接删除表，将表所占用的空间全部释放掉。表的结构（包括约束constraint，触发器(trigger),索引（index））都被删掉

truncate: 删除表中数据，再插入时，自增长id 又从1 开始。一次性删除，不能恢复。操作记录在日志中保存。表和索引恢复到初始空间大小。

delete: 删除表中数据，可以加 where 子句，可回滚，一次删除一行

一般而言，drop > truncate > delete

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。

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。

truncate 和 delete 的区别：

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。

什么时候使用?

用Delete: 想保留标识计数； 有外键，有触发

用Truncate: 想快速删表，但保留结构

Drop：直接删表

（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、连接的种类

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

举两个表作为例子：

```
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|
-----------------------------------------------
```

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)。

```
select * from table1 left join table2 on table1.id=table2.id
-------------结果-------------
id name     id      score
------------------------------
1  lee      1       90
2  zhang    2       100
4   wang    NULL    NULL
------------------------------
```

Left join: 右表中匹配的行加入左表, 以左表为基准，如果右表没有左表中相应的字段，则显示NULL值。

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-------------
id  name    id  score
------------------------------
1       lee     1   90
2       zhang   2   100
NULL    NULL    3   70
------------------------------
```

左表加如右表，左表中不存在的用NULL 代替

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-------------
id  name    id      score
------------------------------
1       lee     1       90
2       zhang   2       100
4       wang    NULL    NULL
NULL    NULL    3   70
------------------------------
```

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-------------
id      name    id      score
------------------------------
1       lee     1       90
2       zhang   2       100
------------------------------
```

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

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
```

10、数据库范式

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）**

11 .数据库优化的思路

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
```

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！

12.存储过程与触发器的区别

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。

13.面试回答数据库优化问题从以下几个层面入手

（1）、根据服务层面：配置mysql性能优化参数；

（2）、从系统层面增强mysql的性能：优化数据表结构、字段类型、字段索引、分表，分库、读写分离等等。

（3）、从数据库层面增强性能：优化SQL语句，合理使用字段索引。

（4）、从代码层面增强性能：使用缓存和NoSQL数据库方式存储，如MongoDB/Memcached/Redis来缓解高并发下数据库查询的压力。

（5）、减少数据库操作次数，尽量使用数据库访问驱动的批处理方法。

（6）、不常使用的数据迁移备份，避免每次都在海量数据中去检索。

（7）、提升数据库服务器硬件配置，或者搭建数据库集群。

（8）、编程手段防止SQL注入：使用JDBC PreparedStatement按位插入或查询；正则表达式过滤（非法字符串过滤）；