**1.DDL/DML/DCL区别**

DDL：data definition language，用于操作对象和对象的属性，包括：create、drop、alter（修改数据表定义和属性），操作对象---表

DML:data manipulation language，用于操作数据库对象中包含的数据，包括：insert、delete、update语句，操作对象---记录

DCL:data control language，控制数据库对象的权限，包括：grant、revoke，操作对象---用户

**2. 数据模式、数据模型定义区别联系，举例**

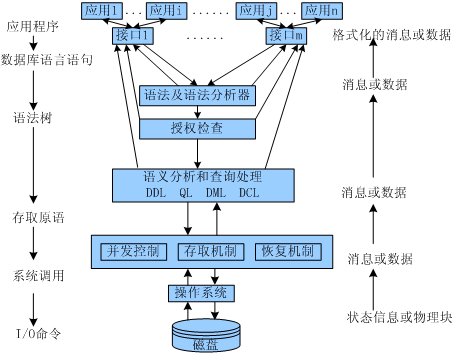
数据模型：用来描述数据的一组概念和定义，如关系数据模型，它规定了属性、元组等概念来描述现实世界中的数据。

数据模式：对某一类数据的结构、联系和约束的描述。例如学生信息记录可以定义为姓名、学号、性别等属性和关系的形式，就是数据模式。

区别：数据模型是描述数据的手段，数据模式是用给定数据模型对具体数据的描述

联系：数据模式需要使用数据模型来描述，数据模式是结果，数据模型是手段，例如c语言类似于数据模型，c语言编写的程序类似于数据模式。

（数据模型是对数据进行概念层次上的描述，是计算机中用来描述现实世界的方法。而数据模式是对特定数据集中数据的结构和联系的描述。数据模式是用给定的数据模型对具体数据的描述。）



**3.在DBMS中，通常采用多级数据模式，例如概念模式、外模式和内模式，简述数据库系统中的多级数据模式对数据独立性的影响**

外模式、概念模式、内模式，有效地组织、管理数据，提高了数据库的逻辑独立性和物理独立性。用户级对应外模式，概念级对应概念模式，物理级对应内模式，使不同级别的用户对数据库形成不同的视图。

数据独立性分为逻辑独立性和物理独立性.物理独立性是指内模式改变时,概念模式保持不变,逻辑独立是指概念模式改变时,外模式不变,从而使应用程序保持不变.当内模式改变时,DBMS只要通过改变概念模式到内模式映射,即可使概念模式保持不变,从而实现了数据的物理独立性.而逻辑独立的实现正好相反.

（简化表述即为：多级数据模式提高了数据独立性

三层数据模式---->二层数据映射----->两级独立性：

逻辑独立性：应用基于外模式开发而逻辑结构变化不会影响应用

物理数据独立性：存储结构的变化不会影响逻辑结构

）

逻辑数据独立性是指局部逻辑数据结构(外视图即用户的逻辑文件)与全局逻辑数据结构(概念视图)之间的独立性。当数据库的全局逻辑数据结构(概念视图)发生变化(数据定义的修改、数据之间联系的变更或增加新的数据类型等)时，它不影响某些局部的逻辑结构的性质，应用程序不必修改。

物理数据独立性是指数据的存储结构与存取方法(内视图)改变时，对数据库的全局逻辑结构(概念视图)和应用程序不必作修改的一种特性，也就是说，数据库数据的存储结构与存取方法独立。

数据独立性的好处：数据的物理存储设备更新了，物理表示及存取方法改变了，但数据的逻辑模式可以不改变。数据的逻辑模式改变了，但用户的模式可以不改变，因此应用程序也可以不变。这将使程序维护容易，另外，对同一数据库的逻辑模式，可以建立不同的用户模式，从而提高数据共享性，使数据库系统有较好的可扩充性，给 DBA 维护、改变数据库的物理存储提供了方便。

**4.sql和关系代数的联系与区别**

关系代数（以关系代数为基础的数据库语言是过程性的）、SQL、关系演算(元组关系演算和域关系演算)，它们的非过程化程度依次递增，主要应用领域也不同.SQL 是关系数据库的标准语言，关系代数和关系演算是它的理论基础

联系：关系代数是sql的理论基础。

区别：sql是结构化查询语言，是数据库具体的技术标准和规范。关系代数是数学理论。

**5.现代数据库怎么管理数据模式的？数据模型怎么影响系统性能？什么是结构化数据，半结构化数据，非结构化数据？**

数据库划分为三个层次，分别为概念模式、外模式、内模式，都存于数据目录中，是数据目录最基本的内容，DBMS通过数据目录管理和访问数据模式

数据模型包括数据结构、数据操作、数据完整性约束，影响性能的主要因素是数据结构的复杂度和数据操作的可优化程度......(可自行扩展)

结构化数据：结构化数据：数据整体结构化，通过数据模型描述

半结构化数据：单条记录内部的数据有结构，数据文件间无联系，整体无结构

非结构化数据：数据间，数据文件间都没有结构

**6.现代数据库如何实现数据的独立性？数据库设计时，数据模式遵循的范式是越高越好吗？**

实现数据独立性：**结合第三题适当自我总结**

数据库的范式主要目的是防止数据冗余、更新异常、插入异常和删除异常，因此，如果达到了该目的也就可以了，但范式越高可能带来处理速度缓慢和处理逻辑复杂的问题，因此需要权衡考虑。并不是应用的范式越高越好，要看实际情况而定。应用的范式等级越高，则表越多。表多会带来很多问题：1 查询时要连接多个表，增加了查询的复杂度 2 查询时需要连接多个表，降低了数据库查询性能

**7.从查询优化的角度分析，为什么SQL查询where子句应该尽量避免使用“OR”**

为了加快查询速度，优化查询效率，主要原则就是应尽量避免全表扫描，尽量在 where 及 order by 涉及的列上建立索引。然而在 where 子句中使用 or 来当连接条件时，在OR连接的诸条件中，只要有一个条件无合适的存取路径，就只能用顺序扫描，会导致引擎放弃使用索引而进行全表扫描（改用 Union 后，性能会大大提高。）

**8.相对于层次、网状数据库系统，查询优化对关系型数据库系统更为重要，对吗？为什么？**

对

层次和网状数据库系统的语言是过程性的，用户不仅要说明需要什么数据，还要告诉数据库系统获得这些数据的过程，二者使用指针表示属性间的关系，这样的结构固定了两种数据库的查询路径，进行优化的空间有限。

而关系数据库系统的语言是非过程性的，用户只用说明需要什么数据，而如何获取由系统来实现。对于同一个查询语句，对应的关系代数等价的不同表达式的查询效率有着很大的差异；集合操作不同的执行规则和策略也对查询效率有着很大的影响；同时关系数据在物理存储形式和存取方式和路径上都没有限制。因此对于关系数据库系统来说，查询优化就更为重要，它对系统的性能有着很大的影响

所以查询优化对于关系数据库系统非常重要。

**9.索引对关系型数据库查询优化的意义，什么时候使用索引？是不是在任何情况下使用索引都能有益处？举例说明**

关系型数据库查询优化的途径之一是依赖于存取路径的优化，而在关系型数据库中索引是用得最多的一种存取路径，建立合适的索引是实现查询优化的首要前提。索引提供了对数据的快速访问，根据操作建立合适的索引能够很大程度上优化存取路径，从而提高查询效率。

使用索引的意义：1. 大大加快数据的检索速度 ;2.创建唯一性索引，保证数据库表中每一行数据的唯一性;3.加速表和表之间的连接;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。

使用索引情况：数据量较大的时候，需要频繁进行查询、排序或分组的属性

并非所有情况都适合使用索引：当列很少使用到时，建立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；当对于小文件上的顺序查找使用堆文件的形式的开销不大，使用索引反而可能会增加开销。因为索引本身需要占用一定的存储空间，而且维护索引也需要一定的开销。对于增删改操作频繁的属性上建立索引可能会起到反作用。

**若在内关系的连接属性上建有索引？是否一定能够提高内关系和外关系的匹配效率？**

当每次循环所选的匹配元组数在内关系中占有较大比例（例如超过15%）时，用无序索引甚至还不如用顺序扫描的方法。内关系的连接属性上有簇集索引时，索引对减少连接所需I/O次数的作用最明显。

**10.稠密索引是否一定能够提高针对索引属性查询的效率？**

不一定

1. 查询小文件中全部或大多数记录时，使用索引并不能提高查询效率，反而会因为索引查询增加开销
2. 如果稠密索引为次索引，且不是簇集索引，则再最低索引中，每个键值对应的不是一个地址而是一个地址集。很可能一个键值对应的多条记录分散在不同的物理块中；当一个键值对应的记录较多时，取这些记录时访问物理块的 I/O 开销反而会降低查询的效率。

**11.建立簇集索引的条件**

聚集索引：该索引中键值的逻辑顺序决定了表中相应行的物理顺序。

聚集索引确定表中数据的物理顺序。聚集索引类似于电话簿，后者按姓氏排列数据。由于聚集索引规定数据在表中的物理存储顺序，因此一个表只能包含一个聚集索引该索引可以包含多个列（组合索引），就像电话簿按姓氏和名字进行组织一样。

非聚集索引：数据存储在一个地方，索引存储在另一个地方，索引带有指针指向数据的存储位置。

非聚集索引中的项目按索引键值的顺序存储，而表中的信息按另一种顺序存储（这可以由聚集索引规定）。对于非聚集索引，可以为在表非聚集索引中查找数据时常用 每个列创建一个非聚集索引。有些书籍包含多个索引。例如，一本介绍园艺的书可能会包含一个植物通俗名称索引，和一个植物学名索引，因为这是读者查找信息的两种最常用的方法。



**12.建立索引和不建立索引的条件**

在一个或者一些字段需要频繁用作查询条件，并且表数据较多的时候，创建索引会明显提高查询速度，因为可由全表扫描改成索引扫描。（无索引时全表扫描也就是要逐条扫描全部记录，直到找完符合条件的，索引扫描可以直接定位）

对于需要频繁新增记录的表，最好不要创建索引，没有索引的表，执行insert、append都很快，有了索引以后，会多一个维护索引的操作，一些大表可能导致insert 速度非常慢

**13.判断并发事务是否正确执行的标准是？封锁法基本思想？ 怎么保证并发事务正确执行？采用封锁法必须解决的问题是？**

标准是对并发事务运行的调度是否可串行化

基本思想：并发事务对同一数据对象操作前，向系统发出请求对操作对象加锁。事务对数据对象的加锁请求获准后，它便对该对象有了一定的控制，在这个事务释放它的锁之前，其他事务对该数据对象的锁请求不能获准，无法对其进行操作，，从而避免了锁冲突，保证并发事务正确执行。

问题：由于事务之间的循环等待导致的死锁问题以及部分事务一直处于等待状态的活锁问题。

**14.假设运行记录与数据库的存储磁盘有独立失效模式。介质失效恢复时,对运行记录中上一检查点以前的已提交事物应该redo否?为什么?**

应该redo

理由：介质失效会丢失数据库中所有的数据，恢复时需要再加载最近的后备副本后，根据运行记录中的后像，重做最近后备副本以后提交的所有更新事务。因此最近一次检查点以前提交的事务也要做 redo 操作。

**15.如事务不遵守 ACID 准则，则对数据库产生何种后果？为什么一般不涉及数据库的程序中不提 ACID 准则？**

若事务不遵守 ACID 准则，数据库中会产生脏数据，数据不一致、难以恢复等情况。

不涉及数据库的程序不提 ACID 是因为它们很少需要满足数据库这样的高并发性和一致性需求。一个事务是由应用程序中对数据库的一组操作序列组成的。如果事务不遵守 ACID 准则，则数据库中数据的完整性和一致性等就可能会因为事务的执行而遭到破坏。而一般不涉及数据库的程序不存在多用户之间数据的共享问题，所以在一般不涉及数据库的程序中不提 ACID 准则。

**16.分布式数据库的全局死锁产生原因？**

由于系统提供的资源数比多个进程所需的资源数少，并且系统的资源分配策略和进程并发执行的速度不当。死锁是占有资源并申请资源的事务之间循环等待造成的。（举一个例子，T1 握有 T2 需要的资源的同时等待 T2 的握有的资源

**17.试分析分布式数据库系统出现的技术背景和应用背景。它与后来出现的联邦式数据库系统的类似之处和本质区别是什么**

背景：分布式处理技术的发展，当时网络宽还不足，因此出现了由网络连接的多台计算机共同协作解决大量数据的存储、管理、查询的需求，通过将数据就近存放提高访问效率。

本质区别：前者在物理上分布的，但逻辑上却是集中的。这种分布式数据库只适宜用途比较单一的、不大的单位或部门。而联邦式分布数据库系统在物理上和逻辑上都是分布的。由于组成联邦的各个子数据库系统是相对“自治”的，这种系统可以容纳多种不同用途的、差异较大的数据库，比较适宜于大范围内数据库的集成。

相似之处：分布式数据库系统的不同类别。是在集中式数据库系统的基础上发展来的。是数据库技术与网络技术结合的产物。包含分布式数据库管理系统(DDBMS)和分布式数据库(DDB)。

**18.数据查询和数据挖掘的区别联系？**

两者期望得到的结果不同，数据查询得到的是些表面的东西（比如上个月的销售额是多少，哪些商品卖得不好等等），数据挖掘得到的是更深层次的内容（比如下个月的销售额将会是多少，导致某些商品销量不佳的因素是什么）

**19.选择操作选择存取路径**

1. 小关系，无索引、散列等存取路径可用，选取的元组数占比大于20%且有关属性上无簇集索引--------顺序扫描
2. 主键等值选择----------簇集或散列
3. 、非主键等值选择元组占比<20%-----无序索引
4. 其余用簇集索引或顺序扫描
5. 对于范围条件，先通过索引找到范围的边界，再通过索引的顺序集沿相应方向搜索，如中选的元组数在关系中所占比例较大，宜采用簇集索引或顺序扫描。
6. AND连接的合取选择条件：
   1. 优先选择多属性索引
   2. 多个可用的次索引，用预查找处理后做其余条件检查
   3. 个别条件用3、4、5之一
   4. 顺序扫描
7. 用OR连接的条件---顺序扫描

**20.连接操作实现和优化**

1. 嵌套循环：最原始----类似于二重循环，有内外关系将占用物理块少的关系作为外关系，优化：ch4 p4背面
2. 利用索引或散列：在内关系上有合适的存取路径（连接属性上的索引散列等）避免内关系上的顺序扫描，减少I/O
3. 归并排序：
4. 散列连接法：连接属性R.A和S.B应具有相同的值域，用相同的散列函数，把R和S散列到同一散列文件中。符合连接条件的元组必然在同一通中（注意：同一桶中的元组未必都满足连接条件）。只需把桶中的匹配元组取出即可获得连接结果。
   1. 需要建立一个供连接用的散列文件，需要对R、S各扫描一次，且R、S一般不会对连接属性进行簇集
   2. 减少I/O次数： 可以在桶（散列文件）中不填入R、S的实际元组，而是代之以元组的tid，从而大大的缩小散列文件，使其有可能在内存中建立，而仅需对R、S各扫描一次
   3. 在取实际元组时，为减少物理块访问，可将各桶中，匹配元组的tid按块分类，一次集中取出同一块中所需的所有元组，当然这需要较大的内存开销。