**数据库系统知识点整理**

# 一、基础查询、视图与数据完整性

## 1. 视图的定义

视图是一种**虚拟的表**，其本质是对应于一条SELECT语句并将结果集赋予一个名字。可以对视图进行增，删，改，查等操作。特别地，对视图的修改不影响基本表。相比多表查询，它使得我们获取数据更容易。

## 2. Drop、Delete与Truncate的区别

* DELETE用来删除表的全部或者一部分数据行，可以回滚，可以带WHERE子句，会触发触发器
* TRUNCATE删除表中的所有数据，不能回滚，不会触发触发器，不可带WHERE子句，速度更快
* DROP命令从数据库中删除表，不能回滚，不会触发触发器，不可带WHERE子句，速度最快

CREATE、TRUNCATE、DROP属于**DDL(*Data Definition Language*)**

INSERT、DELETE、UPDATE属于**DML(*Data Manipulation Language*)**

SELECT、FROM、WHERE属于**DQL(*Data Query Language*)**

GRANT、ROLLBACK、COMMIT属于**DCL(*Data Control Language*)**

## 3. 查询语句不同元素的执行先后顺序

* Select
* From：From的表关联的解析顺序是从右向左的，因此尽量将数据量小的表放在最右边
* Where：Where条件的解析顺序是自下而上的
* Group By
* Having
* Order By

## 4. 临时表

* 临时表用于保存一些临时数据，仅在当前连接可见，关闭连接时MySQL会自动删除表并释放所有空间，其SQL语句为*create temporary table*

## 5. 内连接、外连接、交叉连接（笛卡尔积）

* 内连接：用于返回连接表中符合连接条件和查询条件的数据行，通常有等值连接和不等连接
* 外连接：分左外连接(返回左表中的所有行，如果左表中行在右表中没有匹配行，则结果中右表中的列返回空值)、右外连接(返回右表中的所有行，如果右表中行在左表中没有匹配行，则结果中左表中的列返回空值)和全外连接(返回左表和右表中的所有行，当某行在另一表中没有匹配行，则另一表中的列返回空值)
* 笛卡尔积：返回左表中的所有行，左表中的每一行与右表中的所有行组合。

## 6. 内连接和WHERE的区别

* 使用JOIN和ON对两表进行连接时，在对两表的数据行进行匹配时，会先判断数据行是否符合ON语句后面的条件，再决定是否JOIN
* 使用FROM T1, T2 WHERE时，会先在内存中先生成一张数据量比较大的笛卡尔积表，再进行筛选

因此当两张表的数据量比较大，又需要连接查询时，应该使用JOIN

## 7. VARCHAR和CHAR的使用场景

**VARCHAR**

* 用于存储变长字符串，只占用必要的存储空间
* 应该使用最小符合需求的长度最为长度，存储效率低于CHAR
* 适合字符串最大长度比平均长度大很多，字符串列很少被更新

**CHAR**

* 定长字符串，会过滤末尾空格
* 适合存储短的、长度近似、经常更新的字符串

## 8. 触发器

触发器是**与表相关的数据库对象**，**在满足定义条件时触发**，并执行触发器中定义的语句集合。触发器的这种特性可以协助应用在数据库端**确保数据库的完整性**。

## 9. 主键、外键、超键和候选键

* 超键：在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。
* 候选键：是最小超键，即没有冗余元素的超键。（候选键相当于是超键的一个子集）
* 主键：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能为空值。（主键相当于是候选键的一个子集）
* 外键：在一个表中存在的另一个表的主键称此表的外键。

## 10. 存储过程

存储过程是一组**预编译的SQL语句**，其只在创建时进行编译，而一般SQL语句每执行一次就编译一次，因此存储过程的效率高于连续执行同等的SQL语句。

**优点：**

* 执行效率高
* 可重复调用
* 可供外部程序调用，如JAVA程序

**缺点：**

* 开发调试差（没有良好的IDE开发工具）
* 移植性差（每个数据库的存储过程的语法都不一样，难以维护）

**与函数的区别：**函数是由一个或多个SQL语句组成的子程序，可用于封装代码以便重新使用。

* 存储过程在数据库中完成特定的操作（如插入、删除等），函数则用于返回特定的数据。
* 存储过程声明用PROCEDURE，而函数用FUNCTION声明。
* 函数中不能用临时表，只能用表变量。
* 函数中必须返回一个变量（可以使用RETURN），而存储过程可以返回多个变量或不返回变量。

## 11. SQL约束

* NOT NULL: 用于控制字段的内容一定不能为空（NULL）。
* UNIQUE: 控制字段内容不能重复，一个表允许有多个 Unique 约束。
* PRIMARY KEY: 也是用于控制字段内容不能重复，但它在一个表只允许出现一个。
* FOREIGN KEY: 用于预防破坏表之间连接的动作，也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。
* CHECK: 用于控制字段的值范围。

## 12. 通配符

* %通配符能够匹配连续多个字符
* \_通配符只能匹配一个字符

## 13. COUNT

* COUNT(\*)与COUNT(1)都是对行的数目进行计算，包含NULL
* COUNT(COL)对特定的列的值具有的行数进行计算，不包含NULL值。

# 二、关系数据库设计与规范化

## 1. 关系型数据库：采用关系模型来组织数据结构的数据库

* 容易理解
* 使用方便，能够支持多表连接查询等复杂操作
* 数据的完整性和一致性高
* 但是不能够很好地满足并发需求(因为SQL语句的解析、加锁、磁盘IO会有很大的消耗)

## 2. 非关系型数据库：指非关系型的，分布式的，且一般不保证遵循ACID原则的数据存储系统

* 高并发、读写能力强
* 结构灵活，有良好的可扩展性
* 但是通用性差(没有通用的SQL语句)，容易出错，且没有外键关联等复杂操作

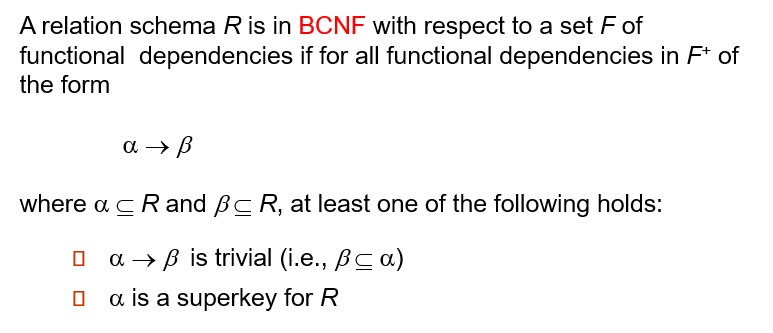
## 3. 数据库三大范式

* 第一范式：（原子性）数据库表的每一列都是不可分割的原子数据项
* 第二范式：在第一范式的基础上消除了部分依赖
* 第三范式：在第二范式的基础上消除了传递依赖

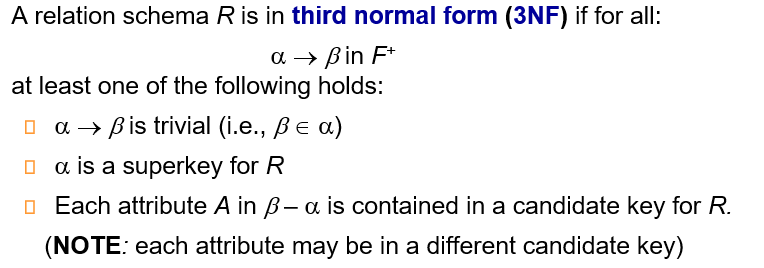
**部分依赖：**假设某表(A, B, C, D, E, F)的候选码为(A, B, C)，但存在C→D的函数依赖，则该表不符合2NF

**传递依赖：**假设某表(A, B, C, D, E, F)的候选码为(A, B, C)，存在ABC→D，D→E的函数依赖，由2NF的判定规则得知该数据表符合2NF因为其不存在部分依赖，但其不符合3NF，因为存在ABC→E的依赖。

## 4. BCNF范式



**BCNF范式是严格版的3NF：3NF只消除了非主码的传递依赖，而BCNF范式还消除了主码的传递依赖**



假设某表(A, B, C, D, E, F)的候选码为(A, B, C, D)，但其中存在如下函数依赖AB→C，C→D，则其符合3NF因为两个函数依赖都满足3NF判定规则中的第三条，但其不满足BCNF范式。

# 三、事务和并发控制

## 1. 事务四大特性（ACID）

* 原子性(Atomicity)：事务被视为不可分割的最小单元，事务的所有操作要么全部提交成功，要么全部失败回滚
* 一致性(Consistency)：在一致性状态下，所有事务对一个数据的读取结果是相同的
* 隔离性(Isolation)：一个事务的修改在最终提交以前，对其他事务是不可见的
* 持久性(Durability)：一旦事务提交，其所作修改将会永远保存到数据库中

## 2. 事务隔离：指多个并发的事务同时访问一个数据库时，一个事务不应该被另一个事务所干扰，每个并发的事务间要相互进行隔离。

* 脏读(Dirty Read)：如果事务A访问了数据库并写入一条记录R，但是没有提交事务，如果此时事务B来读取数据时，返回结果中会出现记录R，这称为脏读
* 不可重复读(Unrepeatable Read)：两次读操作之间有别的事务修改（事务A访问了数据库并查询了ID=1的一条记录，但此时事务B也访问了数据库，修改了ID=1的记录的Name字段并提交了事务，而接着事务A再想查询ID=1的记录时，发现两次读取的Name字段不相同，这称为不可重复读）
* 幻读(Phantom Problem)：由于事务B在事务A的两次读操作之间进行了增删操作，导致事务A两次读取到的数据不一致
* 丢失修改：事务A和B都对数据进行修改，A先修改，B后修改，则B覆盖了A的修改

## 3. 数据库的隔离级别

* 读未提交(READ UNCOMMITTED)：事务A还没有提交，别的事务可以看到事务A对数据修改的结果
* 读已提交(READ COMMITTED)：只能读到已经提交了的内容，是SQL Server默认的隔离级别
* 可重复读(REPEATABLE READ)：当事务启动时，不允许其他事务进行修改操作，是Mysql的默认隔离级别
* 可串行化(SERIALIZABLE)：是数据库的最高隔离级别，在这种级别下，事务按照串行化顺序执行，但效率非常低。



## 4. 悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作，其在查询数据的时候就把事务锁起来，直到提交事务

* 共享锁(S): 发生在数据查找之前，多个事务的共享锁之间可以共存
* 排他锁(X): 发生在数据更新之前，排他锁是一个独占锁，与其他锁都不兼容
* 更新锁(U): 发生在更新语句中，更新锁用来查找数据，当查找的数据不是要更新的数据时转化为S锁，当是要更新的数据时转化为X锁

## 5. 乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性，其在修改数据的时候把事务锁起来，通过VERSION等方式来进行锁定

* 版本号
* 时间戳
* 待更新字段
* 所有字段

# 四、物理存储和索引

**1. MySQL的各种索引区别**

* 普通索引：最基本的索引，没有任何限制。
* 唯一索引：与普通索引类似，不同的是索引列的值必须唯一，但允许有空值。
* 主键索引：是一种特殊的唯一索引，不允许有空值。
* 全文索引：仅可用于 MyISAM 表，针对较大的数据，生成全文索引很耗时耗空间。
* 组合索引：为了更多的提高MySQL效率可建立组合索引，遵循最左匹配原则。

**2. 最左匹配原则**

比如我们建立了一个以(*a,b,c*)为组合的索引，那么将会得到 (*a, ab,abc*)三种索引。若我们按列(*b*)进行查找，或者按列(*bc*)查找，都不会使用到索引，只有以上三种索引可以使用。

**3. 索引失效的情况**

* 条件中有OR或是LIKE查询中以%开头
* 不满足最左匹配原则的多列索引
* 列类型是字符串且在条件中没有将数据使用引号引用起来
* MySQL估计使用全表扫描要比使用索引快

**4. B+树索引与Hash索引的区别**

* B+树索引可以进行范围查询而Hash索引仅能进行等值查询
* Hash索引是一次性定位数据，不需要向B+树索引一样进行多次IO访问，检索效率高。但当Hash索引遇到大量相等Hash值得情况下效率反而会比B+树低
* 对于组合索引中的多个键，Hash索引在索引时要么全部使用，要么全部不使用

**5. MySQL存储引擎(主要从事务处理和锁级别来说明区别)**

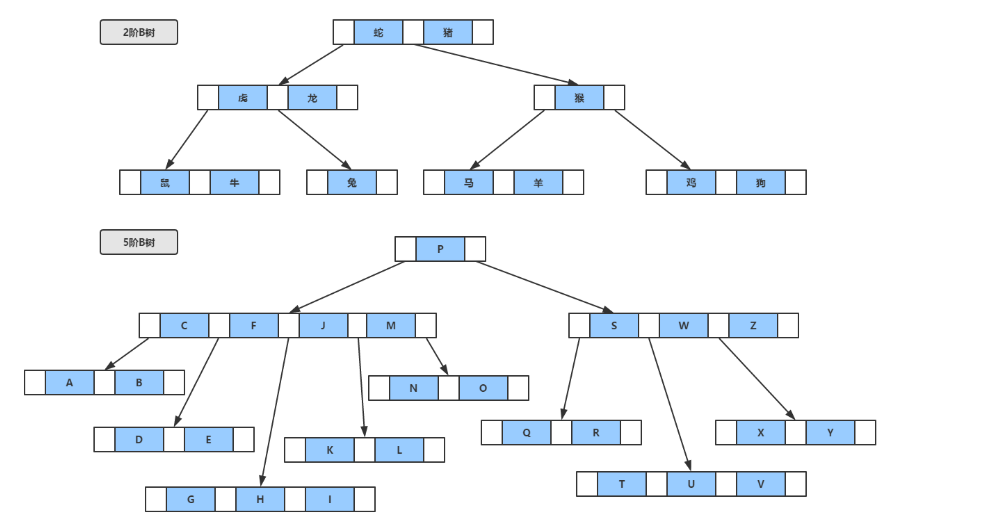
* ***InnoDB***：事务型，支持行级锁、外键、热备份，崩溃恢复时速度快且不易损坏，使用哈希索引，存储容量上限64TB。
* ***MyISAM***：非事务型，提供高速存储和检索以及全文搜索的能力，只支持表级锁，不支持外键，使用全文索引，存储容量没有上限。
* ***MEMORY***：使用存储在内存中的内容来创建表，而且数据全部放在内存中。文件中只存储表的结构，其数据存储在内存中，有利于数据的快速处理，但安全性不高。MEMORY默认使用哈希索引。

【PS-1】*InnoDB*和*MyISAM*两种引擎都使用**B+树**作为索引的数据结构。

【PS-2】*InnoDB*引擎是**基于索引**完成行级锁的。

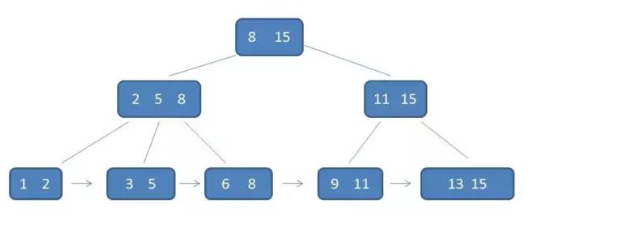
## 6. M阶B树索引的结构特征

* 根节点的孩子数为
* 每个中间节点包含个孩子和个元素，其中*k*的取值范围为
* 所有的叶子节点位于同一层，且都包含个元素，其中*k*的取值范围为



## 7. M阶B+树索引的结构特征

* 根节点的孩子数为
* 每个中间节点包含个孩子和个元素，其中*k*的取值范围为
* 所有的叶子节点位于同一层，且都包含个元素，其中*k*的取值范围为



## 8. B树和B+树结构特征的比较

* 由于B+树的中间节点没有卫星数据，因此同样大小的磁盘页可以容纳更多的节点元素，即B+树相比B树更加矮胖，因此查询时的I/O次数更少。
* 由于B树在查找时最好情况是根节点，最差情况是叶子节点；B+树都是查找到叶子节点，所以B+树的查找更加稳定。
* 对于范围查找来说，B+树只需遍历链表即可，而B树需要进行连续地中序遍历。

**卫星数据：**索引元素所指向的数据记录，比如数据库中的某一行。在B树中无论是中间节点还是叶子节点都带有卫星数据，但是在B+树中只有叶子节点带有卫星数据，中间节点仅仅是索引。

## 9. 聚集索引与非聚集索引的区别（主要从结构、查询和插入的效率）

* 根本区别是表记录的排列顺序和与索引的排列顺序是否一致
* 聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续
* 聚集索引查询数据速度快，插入数据速度慢；非聚集索引反之。

**聚集索引：**索引的逻辑顺序与磁盘上行的物理存储顺序相同，一个表中只能拥有一个聚集索引。

**非聚集索引：**索引的逻辑顺序与磁盘上行的物理存储顺序不同，一个表中可以拥有多个非聚集索引。

## 10. B+树索引与Hash索引的区别

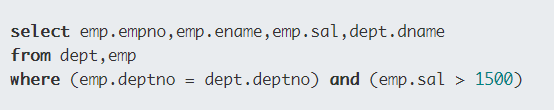
* B+树索引可以进行范围查询而Hash索引仅能进行等值查询
* Hash索引是一次性定位数据，不需要向B+树索引一样进行多次IO访问，检索效率高。但当Hash索引遇到大量相等Hash值得情况下效率反而会比B+树低
* 对于组合索引中的多个键，Hash索引在索引时要么全部使用，要么全部不使用

# 五、查询处理和优化

## 1. SQL语句的优化

RULE1：数据库的解析器按照**从右到左**的顺序处理FROM子句中的表名，FROM子句中写在最后的表将被最先处理，因此在FROM子句包含多个表的情况下，选择记录条数最少的表放在最后。

RULE2：数据库采用**从右到左**的顺序解析WHERE子句，因此表之间的连接条件必须写在其他WHERE条件之左，那些可以过滤掉最大数量记录的条件必须写在WHERE子句的之右。



RULE3：SELECT子句中避免使用(\*)号，因为它要通过查询数据字典获取字段，这意味着将耗费更多的时间，同时使用(\*)号写出来的SQL语句也不够直观

RULE4：当删除全部记录时，用TRUNCATE代替DELETE。

RULE5：多使用内部函数提高SQL效率，如使用CONCAT函数会比使用||来进行拼接快，因为该函数已经被MYSQL优化过了。

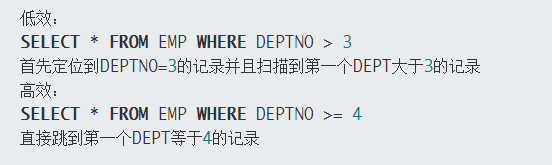
RULE6：表和列的名称应该尽可能地简短。

RULE7：在事务操作中，保证正确的情况下，多使用COMMIT，因为COMMIT会释放回滚点。

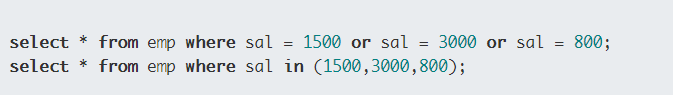
RULE8：编写SQL语句时，尽可能使用大写。

RULE9：索引相关

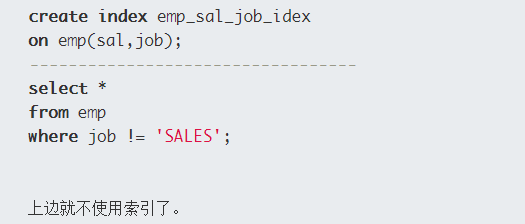
* 避免在索引列上使用NOT（ORACLE服务器遇到NOT后会转而执行全表扫描）
* 避免在索引列上使用计算（如果索引列是函数的一部分，优化器将不使用索引而使用全表扫描）
* 使用大于等于代替大于



* 使用IN代替OR



* 总是使用索引的第一个列：由最左匹配原则知，如果索引是建立在多个列上，只有在它的第一个列被WHERE子句引用时，优化器才会选择使用该索引。 当只引用索引的第二个列时，不引用索引的第一个列时，优化器使用了全表扫描而忽略了索引。



# 六、数据库恢复

## 1. 数据库崩溃恢复事务时UNDO和REDO日志

**UNDO:**

UNDO LOG为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方。然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用UNDO LOG中的备份将数据恢复到事务开始之前的状态。

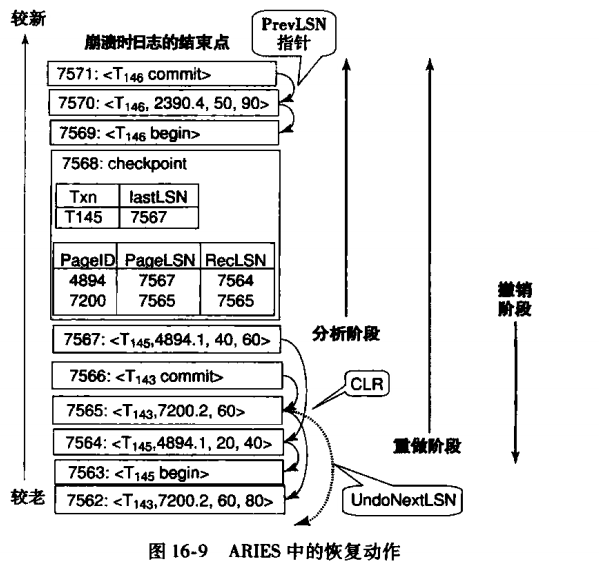
缺陷是每个事务提交前将数据和UNDO LOG写入磁盘，这样会导致大量的磁盘IO，因此性能很低。  
如果能够将数据缓存一段时间，就能减少IO提高性能。但是这样就会丧失事务的持久性。因此引入了另外一种机制来实现持久化，即REDO LOG。

* 反向扫描文件日志（从后往前扫描），查找该事务的更新操作。
* 对该事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库。
* 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。
* 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

**REDO:**

原理和UNDO LOG相反，Redo Log记录的是新数据的备份。在事务提交前，只要将REDO LOG持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是REDO LOG已经持久化。系统可以根据REDO LOG的内容，将所有数据恢复到最新的状态

## 2. ARIES恢复算法



# 七、缓存技术与NoSQL

**1. Redis**

Redis是一款**基于内存**的且支持**持久化**、高性能的**Key-Value**数据库，其支持丰富数据类型，常被用作缓存的解决方案。Redis具有以下显著特点：

* **速度快**：因为数据存在内存中，类似于HashMap。
* **丰富的数据类型**：支持String，List，Set，Sorted Set，Hash；
* **支持事务**：以原子性进行操作。
* **丰富的特性**：可用于缓存消息，按Key设置过期时间，过期后将会自动删除。

