# 1 数据库基础知识

## 数据库范式

数据库的设计范式是数据库设计所需要满足的规范。只有理解数据库的设计范式，才能设计出高效率、优雅的数据库，否则可能会设计出错误的数据库。

范式可以避免数据冗余，减少数据库的空间，减轻维护数据完整性的麻烦，但是操作困难，因为需要联系多个表才能得到所需要数据，而且范式越高性能就会越差。

关系的描述称为关系模式（Relation Schema）它可以形式化地表示为：

R（U，D，dom，F）

其中R为关系名，U为组成该关系的属性名集合，D为属性组U中属性所来自的域，dom为属性向域的[映象](http://baike.baidu.com/view/540438.htm" \t "_blank)集合，F为属性间数据的依赖关系集合。

通常简记为：

R(U)或R(A1，A2，…,An)

其中R为关系名，U为属性名集合，A1，A2，…,An为各属性名。

第一范式：如果关系模式R的每个关系R的属性都是不可分的数据项，那么就称R是第一范式的模式。每个属性原子项不可分割。

案例：student table

学生编号 姓名 性别 联系方式 地址

0001 chen 1 [xxxx@qq.com,188888888888](mailto:xxxx@qq.com,188888888888) 中国四川省成都市武侯区

第二范式：如果关系模式R是1NF，且每个非主键属性都完全依赖主键，那么R就是第二范式。

案例：student table 主键（学生、课程）

学生 课程 教师 教师职称 教材 教室 上课时间

Zhang java li 高级 corejava 303 15:00

第三范式：如果关系模式R是2NF，且每个非主键属性都独立其他非关键字列，并依赖于关键字，不能存在传递依赖。

案例：student table 主键（学生、课程）

学生 课程 教师 教师职称 教室 上课时间

Zhang java li 高级 303 15:00

总结：1NF原子性，2NF消除部分依赖，3NF消除传递依赖

## 事务特性

A---atomicity（原子性）:一个事务必须被视为一个不可分割最小工作单元，事务中的操作只要有一个失败则全部回滚（可以理解为，一件事情，要么全做，有一点调点不具备那么全部都不做。）

C---Consistency(一致性)：在一个转账事务中，A账户的减少必定对应着B账户的增加，这个状态转换的过程必须保持一致，那么这就是事务的一致性。

I---Isolation（隔离性）：通常来说，一个事物所做的修改在最终提交之前，对其他事务是不可兼得。也就是说，事务内部的操作是不会被外部所看到的，只有最终提交之后，我们才能在银行系统中看到两个账户金额的变化。

D---durability（持久性）：一旦失误提交，所有的修改会永久的保存在数据库中，即使系统崩溃，服务器被损坏，只要硬盘还在，数据就依旧存在。

## 1.3 事务隔离与加锁

[并发控制](https://zh.wikipedia.org/w/index.php?title=Concurrency_control&action=edit&redlink=1)描述了数据库处理隔离以保证数据正确性的机制，以事务为单元，并发控制通常使用封锁机制，锁与多版本机制。

### 1.3.1数据库加锁方式

共享锁：如果事务T获得了数据A的共享锁，T对A可读不可写，其他事务只能对A加共享锁，不能加排他锁。

排他锁：如果事务T获得了数据A的排他锁，则T对A可写可读，其他事事务不能再对A进行加锁。

意向共享锁

意向排他锁

一次封锁：在方法开始时预先知道会使用哪些数据，然后全部锁住，方法执行完毕再解锁。这种加锁方式可以避免循环死锁，但是在数据库不适用，事务开始时数据库不知道使用哪些数据。

两阶段锁：加锁阶段与解锁阶段，在加锁阶段，对读操作加共享锁，在进行写操作之前申请获取排他锁，加锁不成功事物进行等待，直到加锁成功。加锁阶段不能进行解锁，事务释放第一个锁后，事务进入解锁阶段，在该阶段只能进行解锁不能加锁。容易出现死锁，加锁阶段多次申请同一个锁。

死锁：T1 slockB xlockA

T2 slockA xlockA

树形协议：数据集合T{t1,t2,t3…..tn}满足一个偏序关系，访问数据按照偏序关系先后进行,如t1🡪t2，要访问t2得先访问t1。可以对任何数据项进行加锁，加锁顺序进行。解锁倒序进行。

规则：事务T第一次加锁对任何数据进行，此后事务T对数据A进行加锁需要先持有A的父亲数据项的锁。对数据解锁可以随时进行，事务T对数据A进行加锁解锁之后不能再被事务T加锁。

时间戳排序协议：每个事务都有一个唯一的时间戳，时间戳有大小，以此来保证事务可串行化。对于每个数据项A，有两个时间戳与之绑定，一个是W-TS(A)最近写事务时间戳,一个是R-TS(A)最近读时间戳。Thomas规则如下：

若事务T1发起一个write(A)，则

如果TS(T1)<R-TS(A)表明事务T1准备写入的值来不及写入A就被提前读取，拒绝TS的写入，操作回滚。

如果TS(T1)<W-TS(A)表明事务T1准备写入过期数据，其他数据已经写回到A上，拒绝T1的写入，操作回滚。

若事务T2发出read(B)：

如果TS(T2)<W-TS(B)，则T2需要读取的数据B已被覆盖，read操作拒绝，T2回滚

如果TS(T2)>=W-TS(B)，则执行read操作。

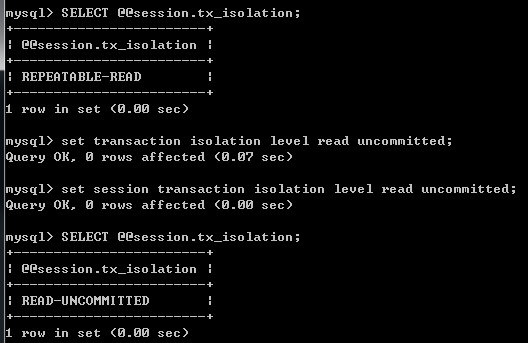
其他情况操作被允许。

### 1.3.2事务隔离

<https://zh.wikipedia.org/wiki/%E4%BA%8B%E5%8B%99%E9%9A%94%E9%9B%A2#.E6.9C.AA.E6.8F.90.E4.BA.A4.E8.AF.BB.28Read_uncommitted.29>

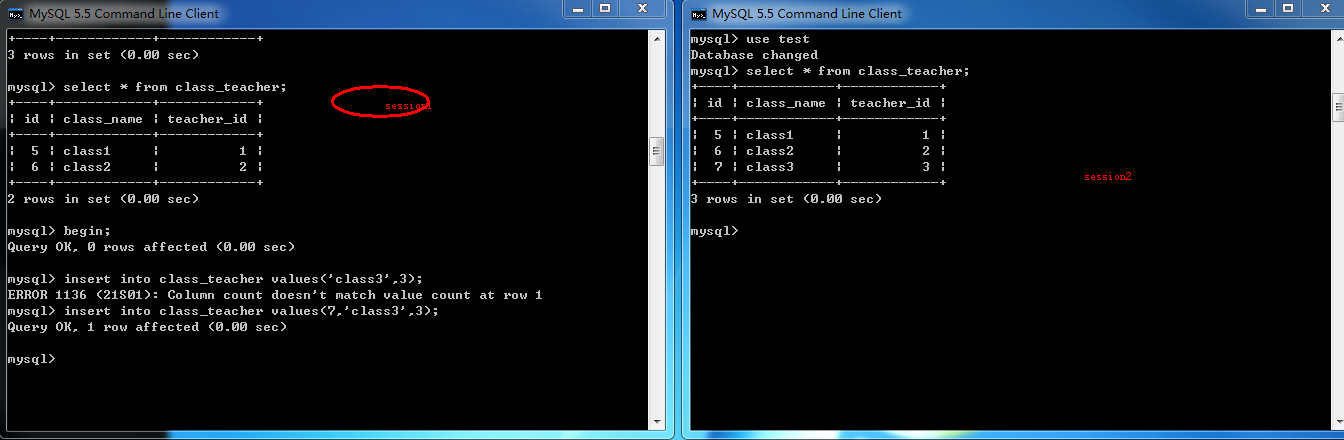
<http://www.cnblogs.com/zhoujinyi/p/3437475.html>

在数据库事务ACID四个属性中，隔离性往往容易被忽略。隔离级别会影响数据库系统的锁机制或者多版本并发控制，数据库系统定义了不同的隔离级别来控制锁的程度，多数数据库事务都避免使用高等级的隔离级别（可序列化）从而减少系统开销。



未提交读：允许脏读，事务可以看到其他事务尚未提交的修改。

案例：



提交读：提交读这种隔离级别保证了读到的任何数据都是提交的数据，避免读到中间的未提交的数据。但是不保证事务重新读的时候能读到相同的数据，因为在每次数据读完之后其他事务可以修改刚才读到的数据。

如何避免不可重复读？

可重复读：同一事务的多个实例在并发读取数据时，会看到同样的数据行。对读锁和写锁会保持到事务结束。但是不要求范围锁，所以可能会造成幻影读。在事务执行过程中，当两个完全相同的查询语句执行得到不同的结果集称为幻影读。

可序列化：最高的隔离级别，在可重复读级别增加了一个范围锁。

**隔离级别vs读现象**

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **脏读** | **不可重复读** | **幻影读** |
| 未提交读 | 可能发生 | 可能发生 | 可能发生 |
| 提交读 | - | 可能发生 | 可能发生 |
| 可重复读 | - | - | 可能发生 |
| 可序列化 | - | - | - |

**隔离级别vs 锁持续时间**

在基于锁的并发控制中，隔离级别决定了锁的持有时间。**"C"**-表示锁会持续到事务提交。 **"S"** –表示锁持续到当前语句执行完毕。如果锁在语句执行完毕就释放则另外一个事务就可以在这个事务提交前修改锁定的数据，从而造成混乱。

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别l** | **写操作** | **读操作** | **范围操作 (...where...)** |
| 未提交读 | S | S | S |
| 提交读 | C | S | S |
| 可重复读 | C | C | S |
| 可序列化 | C | C | C |

## 1.4 SQL主要组成部分

1.**DDL**（[**Data**](javascript:;)**Definition Language**）[**数据库**](javascript:;)**定义语言。**用于定义数据库的三级结构，包括外模式、概念模式、内模式及其相互之间的映像，定义数据的完整性、安全控制等约束。

CREATE、ALTER、DROP、COMMENT、RENAME

2.**DML**（**Data Manipulation Language**）**数据操纵语言。由DBMS提供，用于让用户或程序员使用，实现对数据库中数据的操作。**

SELECT、INSERT、UPDATE、DELETE

3.**DCL**（**Data Control Language**）**数据库控制语言**  授权，角色控制等。

4.**TCL**（**Transaction Control Language**）**事务控制语言**

## 1.5 数据库索引

**数据库索引**，是[数据库管理系统](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)中一个排序的[数据结构](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)，以协助快速查询、更新[数据库表](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8)中数据。

数据存储是

http://blog.jobbole.com/24006/

### 1.5.1 数据映射数据结构

#### 1.5.1.1 二叉查找树

#### 1.5.1.2 B树

#### 1.5.1.3 B+树

#### 1.5.1.4 总结

二叉查找树的结构不适合数据库，因为它的查找效率与层数相关。越处在下层的数据，就需要越多次比较。极端情况下，n个数据需要n次比较才能找到目标值。对于数据库来说，每进入一层，就要从硬盘读取一次数据，这非常致命，因为硬盘的读取时间远远大于数据处理时间，数据库读取硬盘的次数越少越好。

### 1.5.2 数据库模型

行存储模型

列存储模型

行列结合类存储模型

文档类存储模型

json格式存储模型

column family类存储模型

图形存储模型

# 2 数据库规范

# 3 MySQL数据库

## 3.1 mysql 日志

http://aceruser.blog.51cto.com/2772529/658842

### 3.1.1 binlog

MySQL binlog二进制数据库改变事件日志，表与数据的变化，查询日志不记录。主要用途用于用户数据同步与恢复。

### 3.1.2 relay-log

## 3.2 加锁分析

锁机制

锁表

<http://hedengcheng.com/?p=771>

http://hedengcheng.com/?p=828

## 3.3 并行度控制

MySQL并发控制Patch

<http://www.gpfeng.com/?p=499>

## 3.3 数据复制

并行复制

http://backend.blog.163.com/blog/static/20229412620135611215610/

# 4 数据库分区

# 5 高可用架构方案

http://mp.weixin.qq.com/s?\_\_biz=MzI0MjA1Mjg2Ng==&mid=400528497&idx=1&sn=6b068d924ba06d030bbb5b147265abfa

## 5.1 主从

<http://junwang.blog.51cto.com/5050337/1424711>

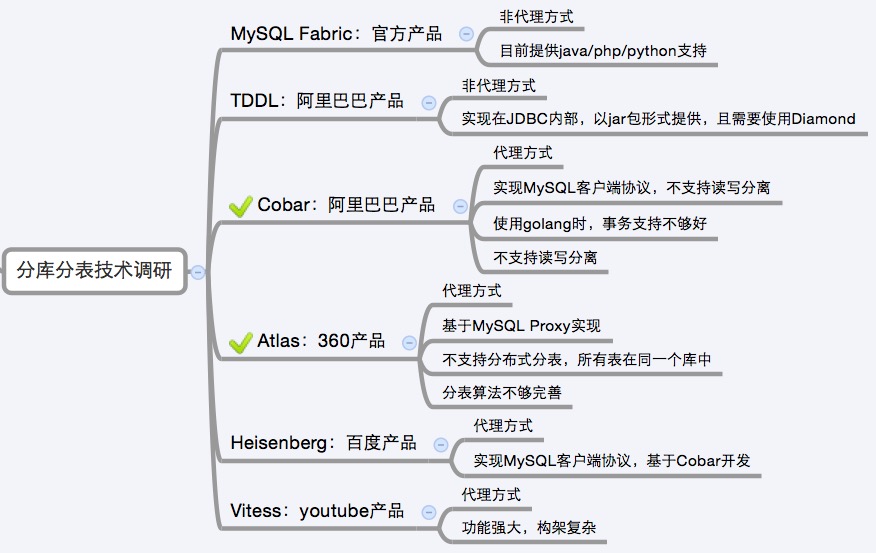
http://www.code123.cc/324.html

## 5.2 分库

http://mp.weixin.qq.com/s?\_\_biz=MzI0MjA1Mjg2Ng==&mid=400528497&idx=1&sn=6b068d924ba06d030bbb5b147265abfa

## 5.3 分表

## 5.4 水平扩展方案实现



<http://www.gpfeng.com/?p=657#comment-17562>

## 5.5 典型系统分析

google F1

google percolator

apache Hbase

apache cassandra

mongoDB

# 6 分布式数据库（CDS）

## 6.1 事物模型

## 6.2 分布式概念

6.2.1 CAP理论

6.2.2短板理论

## 6.3 分布式事物模型

## 6.5 数据一致性

一致性策略分析

弱一致性读写

强一致性写

两段提交改协议的问题

三段提交改协议的分析

PAXOS

少数服从多数

无主机活锁的避免

自动master选择策略

强一致性读(W+R>N)

## 6.6 JDBC

## 6.6 SQL解析

### 6.6.1 解析过程

### 6.6.2 SQL TopN过滤审查

### 6.6.3 超慢SQL拦截

## 6.7 数据切分

### 6.7.1 hash算法

简单Hash

一致性Hash

虚拟节点Hash

基于有序B树的数据切分

## 6.8 执行计划优化

## 6.9 反向索引

## 6.10 数据库主备和动态切换

### 6.10.1主备架构方案

### 6.10.2主备数据延迟方案

并行复制

<http://www.innomysql.net/article/16317.html>

<http://backend.blog.163.com/blog/static/20229412620135611215610/>

## 6.11 并发度流控

## 6.12 读写负载均衡

## 6.13 切分规则变更推送

## 6.14 数据合并

小表广播

分布式Join算法

合并排序

Hash分片Join

# 7 CDS监控

## 7.1 连接池管理

## 7.2 JVM监控

## 7.3 慢SQL监控

## 7.4 SQL报警