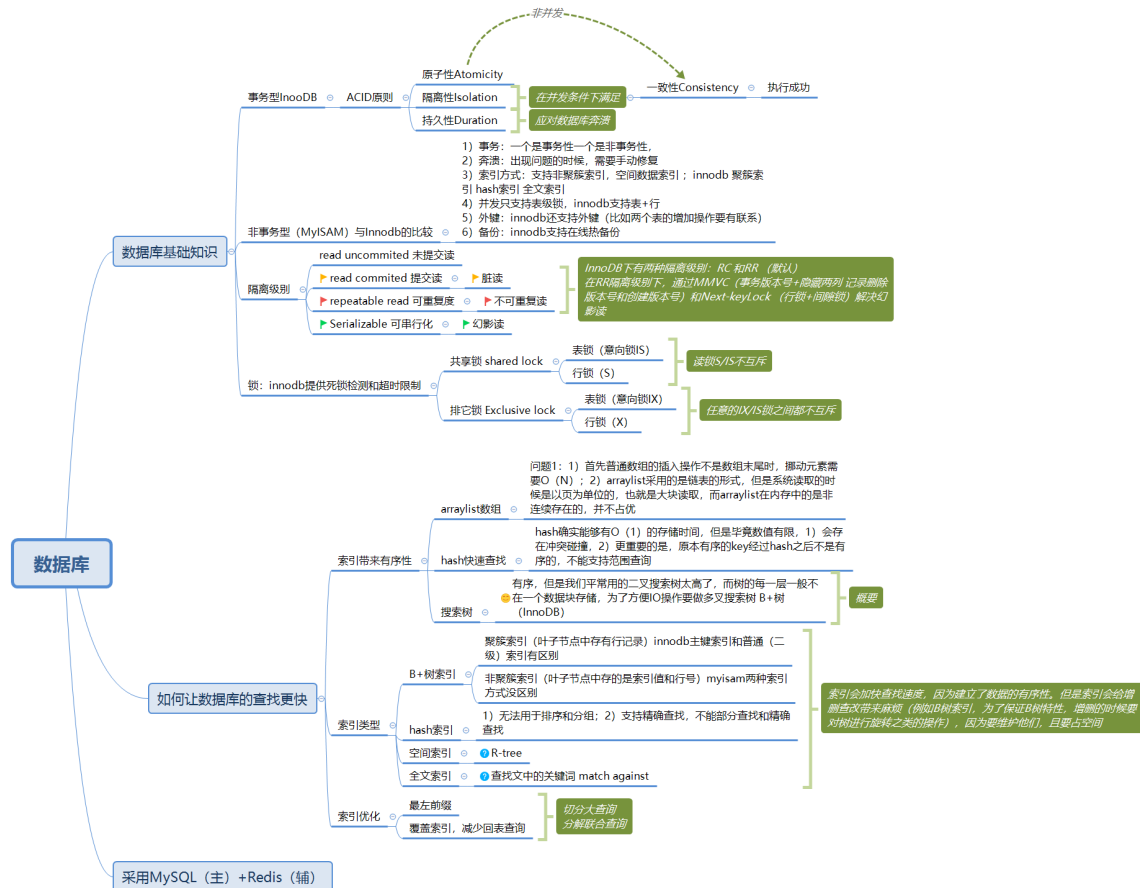


# 数据库

数据库 .....	1
1. 数据库基础知识 .....	3
1.1. 事务型InnoDB.....	3
1.1.1. ACID原则 .....	3
1.2. 非事务型（MyISAM）与InnoDB的比较 .....	4
1.2.1. 1) 事务：一个是事务性一个是非事务性， 2) 崩溃：出现问题的时候，需要手动修复 3) 索引方式：支持非聚簇索引，空间数据索引；innodb 聚簇索引 hash索引 全文索引 4) 并发只支持表级锁，innodb支持表+行 5) 外键：innodb还支持外键（比如两个表的增加操作要有联系） 6) 备份：innodb支持在线热备份 .....	4
1.3. 隔离级别.....	4
1.3.1. read uncommitted 未提交读 .....	4
1.3.2. read committed 提交读.....	4
1.3.3. repeatable read 可重复度 .....	4
1.3.4. Serializable 可串行化 .....	5
InnoDB下有两种隔离级别：RC 和RR（默认） 在RR隔离级别下，通过MMVC（事务版本号+隐藏两列 记录删除版本号和创建版本号）和Next-keyLock （行锁+间隙锁）解决幻影读(read committed 提交读, repeatable read 可重复度) .....	5
1.4. 锁：innodb提供死锁检测和超时限制 .....	5
1.4.1. 共享锁 shared lock .....	5
1.4.2. 排它锁 Exclusive lock .....	5
任意的IX/IS锁之间都不互斥(排它锁 Exclusive lock) .....	5
2. 如何让数据库的查找更快 .....	5
2.1. 索引带来有序性.....	5
2.1.1. arraylist数组 .....	5
2.1.2. hash快速查找.....	6
2.1.3. 搜索树 .....	6
概要(搜索树).....	6
2.2. 索引类型.....	6
2.2.1. B+树索引 .....	6
2.2.2. hash索引 .....	6
2.2.3. 空间索引 .....	6

2.2.4. 全文索引 .....	7
索引会加快查找速度，因为建立了数据的有序性。但是索引会给增删查改带来麻烦（例如B树索引，为了保证B树特性，增删的时候要对树进行旋转之类的操作），因为要维护他们，且要占空间(B+树索引, hash索引, 空间索引, 全文索引) .....	
2.3. 索引优化.....	7
2.3.1. 最左前缀 .....	7
2.3.2. 覆盖索引，减少回表查询 .....	7
切分大查询 分解联合查询(最左前缀, 覆盖索引，减少回表查询) .....	7
3. 采用MySQL（主）+Redis（辅） .....	7



## 1. 数据库基础知识

### 1.1. 事务型InnoDB

#### 1.1.1. ACID原则

原子性Atomicity

参见: [一致性Consistency \(非并发\)](#)

隔离性Isolation

持久性Duration

在并发条件下满足([隔离性Isolation](#))

一致性Consistency

参见: [原子性Atomicity \(非并发\)](#)

执行成功

应对数据库崩溃([持久性Duration](#))

## 1.2. 非事务型（MyISAM）与InnoDB的比较

- 1.2.1. 1) 事务：一个是事务性一个是非事务性，
- 2) 崩溃：出现问题的时候，需要手动修复
- 3) 索引方式：支持非聚簇索引，空间数据索引；innodb 聚簇索引 hash索引 全文索引
- 4) 并发只支持表级锁，innodb支持表+行
- 5) 外键：innodb还支持外键（比如两个表的增加操作要有联系）
- 6) 备份：innodb支持在线热备份

## 1.3. 隔离级别

1.3.1. read uncommitted 未提交读

1.3.2. read committed 提交读

脏读



1.3.3. repeatable read 可重复度



不可重复读



#### 1.3.4. Serializable 可串行化



幻影读



InnoDB下有两种隔离级别：RC 和RR（默认）

在RR隔离级别下，通过MMVC（事务版本号+隐藏两列

记录删除版本号和创建版本号）和Next-keyLock

（行锁+间隙锁）解决幻影读([read committed 提交读](#), [repeatable read 可重复度](#))

#### 1.4. 锁：innodb提供死锁检测和超时限制

##### 1.4.1. 共享锁 shared lock

表锁（意向锁IS）

行锁（S）

读锁S/IS不互斥([表锁（意向锁IS）](#), [行锁（S）](#))

##### 1.4.2. 排它锁 Exclusive lock

表锁（意向锁IX）

行锁（X）

任意的IX/IS锁之间都不互斥([排它锁 Exclusive lock](#))

## 2. 如何让数据库的查找更快

### 2.1. 索引带来有序性

#### 2.1.1. arraylist数组

问题1: 1) 首先普通数组的插入操作不是数组末尾时, 挪动元素需要 $O(N)$ ; 2) `arraylist`采用的是链表的形式, 但是系统读取的时候是以页为单位的, 也就是大块读取, 而`arraylist`在内存中的是非连续存在的, 并不占优

### 2.1.2. hash快速查找

hash确实能够有 $O(1)$ 的存储时间, 但是毕竟数值有限, 1) 会存在冲突碰撞, 2) 更重要的是, 原本有序的key经过hash之后不是有序的, 不能支持范围查询

### 2.1.3. 搜索树

有序, 但是我们平常用的二叉搜索树太高了, 而树的每一层一般不在一个数据块存储, 为了方便IO操作要做多叉搜索树 B+树 (InnoDB)



概要([搜索树](#))

## 2.2. 索引类型

### 2.2.1. B+树索引

聚簇索引 (叶子节点中存有行记录) `innodb`主键索引和普通 (二级) 索引有区别

非聚簇索引 (叶子节点中存的是索引值和行号) `myisam`两种索引方式没区别

### 2.2.2. hash索引

1) 无法用于排序和分组; 2) 支持精确查找, 不能部分查找和精确查找

### 2.2.3. 空间索引

R-tree



#### 2.2.4. 全文索引

查找文中的关键词 **match against**



索引会加快查找速度，因为建立了数据的有序性。但是索引会给增删查改带来麻烦（例如B树索引，为了保证B树特性，增删的时候要对树进行旋转之类的操作），因为要维护他们，且要占空间([B+树索引](#), [hash索引](#), [空间索引](#), [全文索引](#))

### 2.3. 索引优化

#### 2.3.1. 最左前缀

#### 2.3.2. 覆盖索引，减少回表查询

切分大查询

分解联合查询([最左前缀](#), [覆盖索引](#), [减少回表查询](#))

### 3. 采用MySQL（主）+Redis（辅）