# 数据库设计

**为什么进行数据库设计**

**数据库实体就是数据库管理系统中的不同管理对象。**
**需求分析--逻辑分析--物理设计--维护优化**

| 需求分析 | 逻辑分析- | 物理设计 | 维护优化 |
| --- | --- | --- | --- |
| 数据库需求的作用点<br>1 数据是什么<br>2 数据有哪些属性<br>3 数据和属性各自的特点有哪些 | 使用ER图对数据库进行逻辑建模 | 根据数据库自身的特点把逻辑设计转换为物理设计 | 1新的需求进行建表<br>2 索引优化<br>3大表优化 |

**为什么要进行需求分析**
不进行需求分析的话，设计出来的数据库很可能不是最优化
1 了解系统中的数据有哪些
2 了解数据的存储特点
3 了解数据的生命周期

要更搞清楚的一些问题
1 实体与实体之间的关系（1对1 1对多，多对多）
2 实体的所包含的属性有什么？
3 那些属性或属性的组合可以唯一标识一个实体

## 逻辑设计

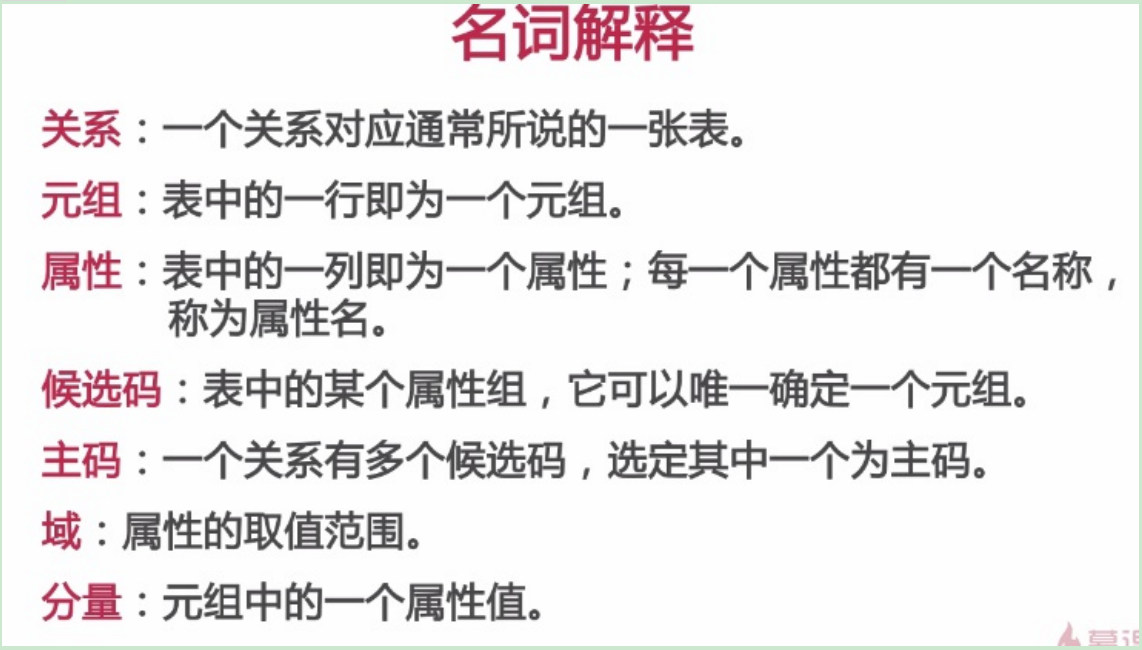
**逻辑设计师做什么的**

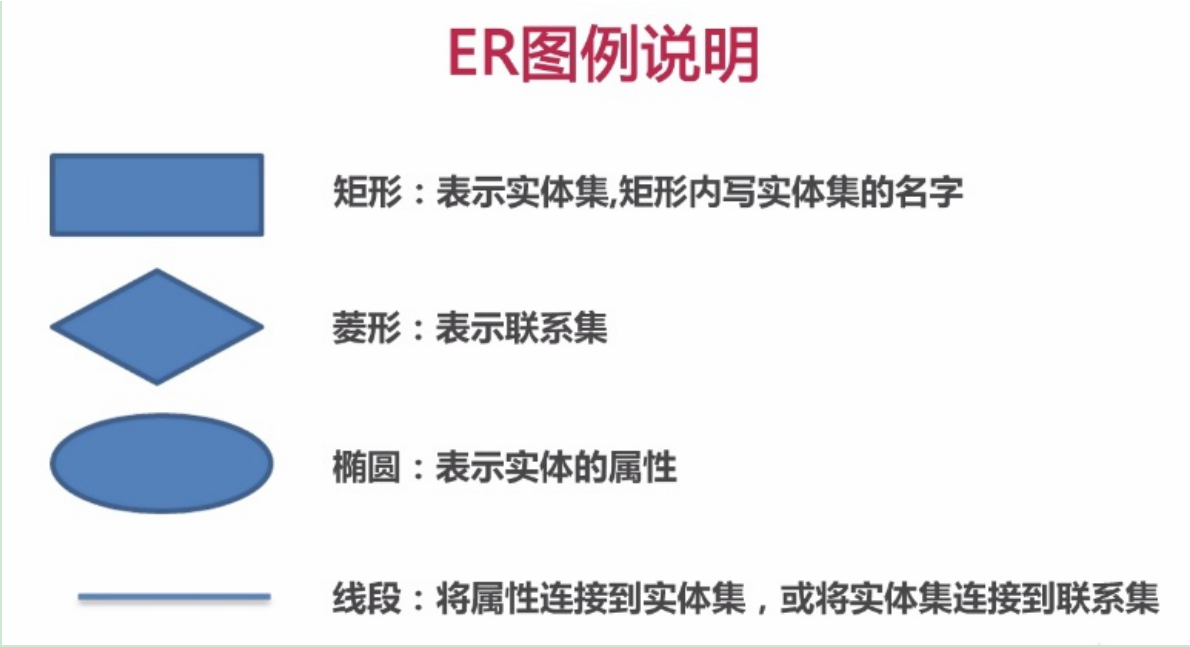
1 将需求转换为数据库的逻辑模型

2 通过ER图的型式对逻辑设计模型进行展示

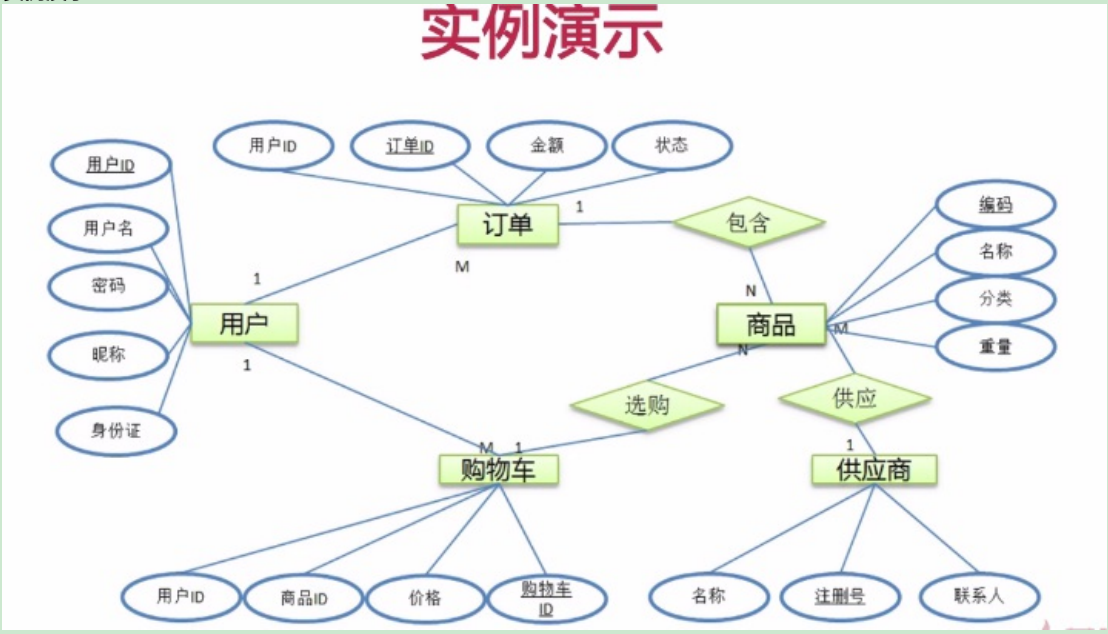
3 同所选用的具体DBMS系统无关

**ER图**






**实例演示**




例如用户ID加了下标就为主键

**数据库设计的范式**

常见的有 第一范式 第二范式 第三范式以及BC范式

数据异常

操作异常分为

1 插入异常 ：某实体随另一实体的存在而存在，缺少某个实体时无法表示这个实体。这个表存在插入异常

2 更新异常 ：更改表所对应的某个实体实例的单独属性时，需要将多行更新，这个表存在更新异常

3 删除异常 ：删除表的某一行来反映某实体实例。失效时导致另一个不同实体实例信息丢失。这个表存在删除异常

数据冗余：

是指相同的数据在多个地方存在，或者说表中的某个列可以由其他列计算得到，这样就说表中存在着数据冗余。

第一范式：

定义：

数据库中的所有字段都是单一属性，不可再分的。

这个单一属性是由基本的数据类型所构成的，如整数，浮点类，字符串等。

第一范式要求数据库中的表都是二维表。

例子：

| 用户ID | 用户名 | 密码 | 姓名 | 电话 |
|---|---|---|---|---|
| 1 | Zhang3 | ******* | 张三 | 1388888 |

https://www.zhihu.com/question/24696366这是关于三大范式的具体讲解

**第二范式：**
定义：
数据库中的表中不存在非关键字段对任一候选关键字段的部分函数依赖。
部分函数依赖是指存在着组合关键字中的某一关键字决定非关键字的情况
换名话说：所有单关键字段的表都符合第二范式
**不符合第二范式的存在问题**
：插入异常
：删除异常
：更新异常
：数据冗余
如果两个或两个以上信息组合的字段称为组合关键字段。非关键字段对组合关键字段存在依赖关系就叫做部分函数依赖。单关键字段的表都是符合第二范式的。
第二范式：比第一范式要求更高
如果某属性依赖组合关键字的一部分，则不符合要求——>将表拆分成两个表和一个关系

**第三范式(对第二范式的扩展)**

第三范式是在第二范式的基础上定义的，**如果数据表不存在非关键字，**
**对任意候选关键字段的传递函数依赖，则符合第三范式.**
例子
(商品名称)--（分类）--（分类描述）
非关键字段" 分类描述"对关键字段“商品名称”的依赖函数依赖

| 商品名称 | 价格 | 商品描述 | 重量 | 有效期 | 分类 | 分类描述 |
|---|---|---|---|---|---|---|
| 可乐 | 3.00 | | 250ml | 2014.6 | 酒水饮料 | 碳酸饮料 |
| 苹果 | 8.00 | | 500g | | 生鲜食品 | 水果 |

**存在的问题**
（分类，分类描述）对于每一个商品都会进行记录，所以存在数据冗余
同时，也还存在数据的插入，更新与删除异常

**问：第二范式和第三范式如何区别？**
第二范式：非主键列是否依赖主键（包括一列通过某一列 间接依赖主键），要是有依赖关系的就是第二范式；
第三范式：非主键列是否是直接依赖主键，不能是那种通过传递关系的依赖的。要是符合这种就是第三范式；

**BC范式(对第三范式的扩展)**
**Boyce.Codd范式(BCNF)**

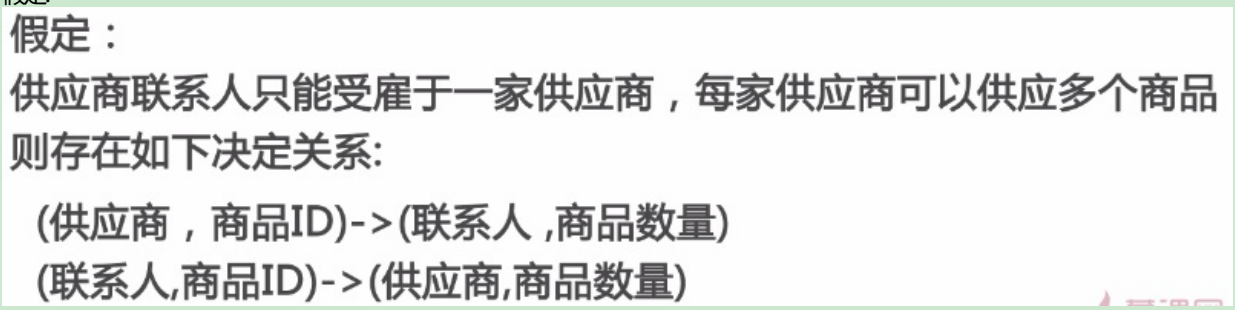
BC范式，比较奇怪，是在第三范式的基础之上定义的，应该叫第四范式，不过四与死同音，所以就这样叫了，这个范式的要求更为严格，他的意思是讲，如果一张表的关键字是一个复合的关键字，字段之间也不能有依赖关系，这样的表结构的设计才是符合BC范式的。

**定义:**
在第三范式的基础上，数据库表如果不存在任何字段对任一候选关键字段的传递参数则符合Bc范式

**也就是说如果是复合关键字，则复合关键字之间也不能存在函数依赖关系。**（以商品供应商的关系表来说明BCNF）

| 供应商 | 商品ID | 供应商联系人 | 商品数量 |
|---|---|---|---|
| 饮料一厂 | 1 | 张三 | 10 |
| 饮料一厂 | 2 | 李四 | 20 |
| 饮料二厂 | 1 | 王五 | 20 |

假定:



存在下列关系因此不符合BCNF要求：

(供应商)->(供应商联系人)

(供应商联系人)->(供应商)

并且存在数据操作异常及数据冗余

## 物理设计

1 选择合适的数据库管理系统
oracle,sqlServer.mysql....
2 定义数据库，表及字段的命名
3 根据命名规范的DBMS系统选择合适的字段类型
4 反范式化设计
。反范式，故名思义，跟范式所要求的正好相反，在反范式的设计模式，我们可以允许适当的数据的冗余，用这个冗余去取操作数据时间的缩短
(。**也就是用空间来换时间,把数据冗余在多个表中，当查询时可以减少或者是避免表之间的关联；**)

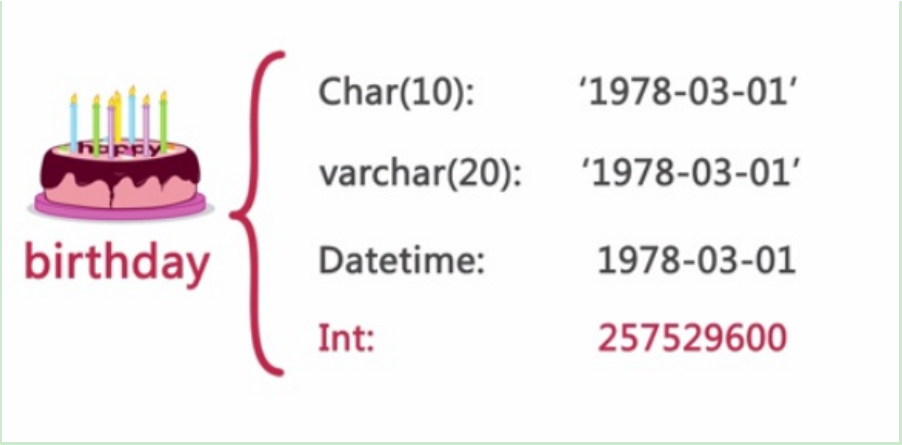
**常见的DBMS系统**
**商业数据库(更适合企业级项目):**
Oracle 性能高 适合大的事务操作 同时可以运行在window下也可以运行在linux下
SQL Server 只支持window上运行 开发语言适合.net
**开源数据库(适用于互联网项目)：**
Mysql 同时可以运行在window下也可以运行在linux下
Pgsql 同时可以运行在window下也可以运行在linux下

**mysql常用的存储引擎**

| 存储引擎 | 事务 | 锁粒度 | 主要应用 | 忌用 |
| --- | --- | --- | --- | --- |
| MyISAM | 不支持 | 支持并发插入的表级锁 | SELECT,INSERT， | 读写操作频繁 |
| MRG_MYISAM | 不支持 | 支持并发插入的表级锁 | 分段归档，数据仓库 | 全局查找过多的场景 |
| Innodb | 支持 | 支持MVCC的行级锁 | 事务处理 | 无 |
| Archive | 不支持 | 行级锁 | 日志记录,只支持 insert,select | 需要随机读取，更新,删除 |
| Ndb cluster | 支持 | 行级锁 | 高可用性 | 大部份应用 |

在使用mysql集群的情况下才使用Ndb cluster mysql集群是一种内存性的集群 如果数量比较大的话 超过内存的大小，对这种情况就不适用Ndb集群，所以就不可能使用Ndb的存储引擎
主要使用的引擎还是innodb

**表及字段的命名规则**
1.可读性原则（使用大小写来格式化的库对象名以获得良好的可读性）
2.表意性原则（对象的名字应该能描述出它所标识的的对象）
3.长名原则（尽可能少使用或不使用缩写）
**字段类型的选择规则**




列的数据类型一方面影响数据存储空间的开销，另一方面也会影响数据查询的性能。当一个列可以选择多种数据库类型时，应该优先考虑数字类型，其次是日期或二进制类型，最后是字符类型。对于相同级别的数据类型，应该优先选择占用空间小的数据类型。
优先使用int 然后是Datetime 然后是char(10),最后是varchar(20);

| 列类型 | 存储空间 |
| --- | --- |
| TINYINT | 1 字节 |
| SMALLINT | 2 个字节 |
| MEDIUMINT | 3 个字节 |
| INT | 4 个字节 |
| BINGINT | 8 个字节 |
| DATE | 3 个字节 |
| DATETIME | 8 个字节 |
| TIMESTAMP | 4 个字节 |
| CHAR(M) | M字节，1 <= M <= 255 |
| VARCHAR(M) | L+1 字节, 在此L <= M和1 <= M <= 255 |

TIMESTAMP有限制的，最多存储到2037年 存储方式为int 由于4个字节的限制

在对数据进行比较(查询条件,JOIN条件及排序)操作时:同样的时间，字符处理往往比数字处理慢.
在数据库中，数据处理已页为单位，列的长度越小，利于性能提升

**char与vachar如何选择**
原则:
1 如果列中要存储的数据长度差不多一致的，
则应该考虑用char；否则应该用varchar。
2 如果列中的最大数据长度小与508Byte,则一般也考虑用char。
3 一般不宜定义大于50Byte的char 类型列

**decimal与float如何选择**
1.decimal用于存储精确数据，而float只能用于存储非精度数据。
2.由于float的存储空间开销一般比demimal小。
故非精度数据优先选择float

**时间类型如何存储**
1.使用int来存储时间字段的优缺点
优点：字段长度比datetime小
确定：使用不方便，要进行函数转换。
限制:只能存储到2038-1-19 11:14:07即2^32为2147483648
2.需要存储的时间粒度
年月日小时分秒周

**数据库设计其他注意事项**

# 如何选择主键

## 1. 区分业务主键和数据库主键

业务主键用于标识业务数据，进行表与表之间的关联；

数据库主键为了优化数据存储(Innodb会生成6个字节的隐含主键)

## 2. 跟据数据库的类型,考虑主键是否要顺序增长

有些数据库是按主键的顺序逻辑存储的

## 3. 主键的字段类型所占空间要尽可能的小

对于使用聚集索引方式存储的表，每个索引后都会附加主键信息。

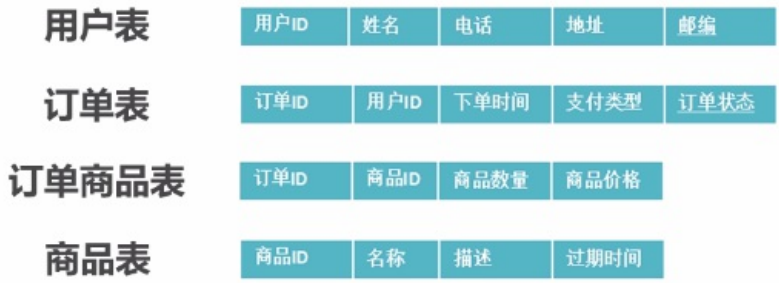
**避免使用外键约束**
1 降低数据导入的效率
2增加维护成本
3 虽然不建议使用外键约束，但是相关联的列上一定要建立索引

**避免使用触发器**
1 降低数据导入的效率
2 可能会出现意向不到的数据异常
3 使业务逻辑变的复杂

**关于预留字段**
1 无法准确的知道预留字段的类型
2 无法准确的知道预留字段所存储的内容
3 后期维护预留所要的成本
同增加一个字段所需要的成本是相同
4 严禁使用预留字段

**什么是反范式**
反范式化是针对范式化而言的，所谓的反范式化就是为了性能和读取效率的考虑而适当的对第三范式的要求进行违反，而允许存在少量的数据冗余
换句话来说反范式化就是使用空间来换取时间
通过对写的工作的增多，增加对读的操作

## 符合范式化的设计



## 反范式化的设计


## 如何查询订单详情

```
SELECT b.用户名,b.电话,b.地址
,a.订单ID
,d.商品名称,d.过期时间,c.商品数量
,c.商品价格
 FROM `订单表` a
JOIN `用户表` b
   ON a.用户ID=b.用户ID
JOIN `订单商品表` c
ON c.订单ID=b.订单ID
JOIN  `商品表` d on d.商品ID=c.商品ID
```

```
SELECT b.用户名,b.电话,b.地址
,a.订单ID
,c.商品名称,c.过期时间,c.商品数量
,c.商品价格
 FROM `订单表` a
JOIN `用户表` b
   ON a.用户ID=b.用户ID
JOIN `订单商品表` c
ON c.订单ID=b.订单ID
```

**为什么反范式化**

1 减少表的关联数量
2 增加数据的读取效率
3 反范式化一定要适度

## 维护与优化

1 维护数据字典
2 维护字典
3 维护表结构
4 在适当的时候对表进行水平拆分或垂直拆分

**如何维护数据字典**

1 使用第三方工具对数据字典进行维护
2 利用数据库本身的备注
以 mysql为例：

```
CREATE TABLE customer(
  cust_id INT AUTO_INCREMENT  NOT NULL COMMENT '自增ID
  cust_name VARCHAR(10) NOT NULL COMMENT '客户姓名',
  PRIMARY KEY (cust_id)
) COMMENT '客户表'
```

增加**comment**字段进行维护

## 3.导出数据字典

```
SELECT
a.table_name,b.TABLE_COMMENT,a.COLUMN_NAME,
a.COLUMN_TYPE,a.COLUMN_COMMENTFROM
information_schema.COLUMNS a JOIN information_schema.
TABLES b ON a.table_schema=b.table_schema AND
a.table_name=b.table_name
WHERE a.table_name='customer'
```

| table_name | TABLE_COMMENT | COLUMN_NAME | COLUMN_TYPE | | COLUMN_COMMENT |
|---|---|---|---|---|---|
| customer | 客户表 | cust_id | int(11) | 7B | 自增ID |
| customer | 客户表 | cust_name | varc... | 11B | 客户姓名 |

**如何维护索引**
1 出现在where从句，group by从句，order by从句中的列
2 可选择性高的列要放在索引的前面
3 索引不要包括太长的数据类型

**注意事项：**
1 索引并不是越多越好，过多的索引不但会降低写效率而且会降低读的效率
2 定期维护索引索引碎片
3 在SQL语句中不要使用强制索引关键字

**如何维护表结构**
注意事项
1 使用在线变更结构的工具
mysql 5.5之前可以使用 pt-online-schema-change (本身就有三个触发器 表中不要存在触发器 mysql一个表中一个触发器只有一个)
mysql 5.6之后本身支持在线表结构的变更
2 同时对数据字典进行维护
3 控制表的宽度与大小

**数据库适合的操作**

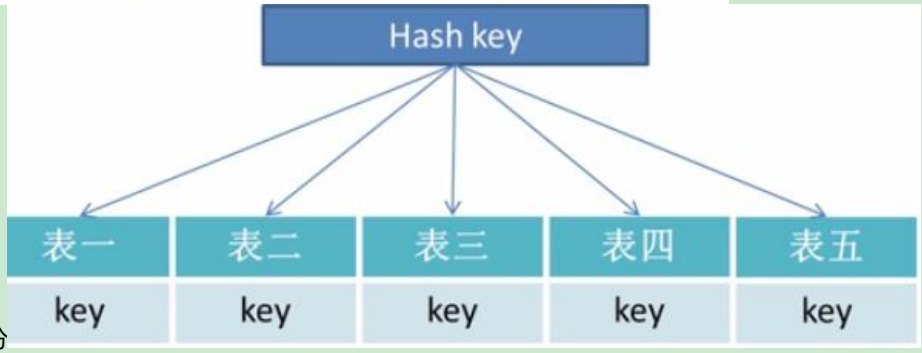
1. 批量操作 VS 逐条操作
2. 禁止使用Select * 这样的查询
3. 控制使用用户自定义函数
4. 不要使用数据库中的全文索引

1 数据库中适合批量操作 程序中适合逐条操作
2 造成IO的浪费 如果表结构进行了变更 这样也会对程序进行影响，这样会导致程序的出错
3 自定义函数的使用过多，会对索引造成影响（如果使用函数，则索引就不起作用 ）
4 全文索引要建立新的索引文件进行维护 另外对中文的支持不是太好

**（优化表的IO）数据库的垂直水平拆分**
**维护优化经常使用到方式**
**表的垂直拆分（解决表的宽度太长的问题）**
当我们的需求变得越来越多，当我们的列增长到了几十列的时候 这个时候就要考虑了

1 经常一起查询的列放到一起
2 text，blob等大字段拆分出附加表
（拆分后的数据量不应该有变化 ）

**水平拆分（解决数据量太多的问题）**


通过主键hash key的方式进行拆分

第一次复习增加：

https://www.zhihu.com/question/29619558关于写好SQL的建议