项目管理-数据库设计

在对软件进行设计的过程中，数据库的设计是一项重要的内容，软件中主要的处理对象就是各类业务数据，通过对业务数据的处理，实现各种功能。我们经常说的，写程序，说到底就是增删改查，而增删改查的对象就是各种数据。数据都存储在数据库中，其重要性不言而喻，对于数据库的设计也是软件设计的一个重要基础。

1. 数据库选型

开发的软件，使用什么样的数据库，是由具体的业务数据要求决定。主要考虑的因素包括需要处理哪些数据，数据的规模如何，是否为结构化数据，对数据的安全性要求如何，数据是否需要实时进行处理等多个方面。

数据库类型有基于SQL的关系型数据库，主要用来处理结构化数据，由一个个表构成，可以通过SQL语句进行查询，如MySQL、PostgreSQL、Oracle、MS SQL Server等。

NoSQL数据库，用来处理非结构或半结构化数据，也有自己的查询语言。NoSQL数据库根据存储数据结构的不同，又可以分为文档存储数据库，常见的文档数据库有MongoDB、CouchDB、DocumentDB。列数据库，数据按列进行存储，常见的列存储数据库有 Cassandra。键值对存储数据库，用来存储键值对数据，常见的键值对数据库有 Redis、Memcached。图数据库，以三元组的形式，存储规模较大的数据，常见的图数据库有Neo4j、InfiniteGraph、virtuoso等。

分类	Examples举例	典型应用场景	数据模型	优点	缺点
键值(key.value)	Tokyo Cabinet/Tyrant Redis Voldemort Oracle BDB	内容缓存，主要用于处理大量数据的高访问负载，也用于一些日志系统等等。	Key指向Value的值对，通常用hashtable来实现	查速度快	数据无结构化，通常只被当作字符串或者二进制数据
列存储数据库	Cassandra, HBase, Riak	分布式的文件系统	以列式存储，同一列数据存在一起	查换速度快：可扩展性强，更容易进行分布式扩展	功能相对局限
文档型数据库	CouchDB, MongoDb	Web应用(与Key-Value类似，Value是结构化的，不同的是数据库能够了解value的内容)	Key-Value对应的值对：Value为结构化数据	数据结构要求不严格，表结构可变：不需要像关系型数据库一样需要预先定义表结构	查询性能不高，而且缺乏统一的查询语法。
图数据库	Neo4J Info Grid Infnite Graph	社交网络，推荐系统等。专注于构建关系图谱	图结构	利用图结构相关算法。比如最短路径寻址，N度关系查找等	很多时候需要对整个图做计算才能得出需要的信息：而且这种结构不太好做分布式的集群方案

选择合适的数据库，就是使用数据库的特点和优点，满足实际的业务场景。也有很多时候，是将多种数据库混合使用，结合多种数据库的优点。比如Mysql配合Redis使用，在软件开发中比较普遍，把Mysql表中经常访问的记录放在了Redis中，然后用户查询时先去查询Redis再去查询Mysql，实现了读写分离，也就是Redis只做读操作。由于缓存在内存中，所以查询会很快。

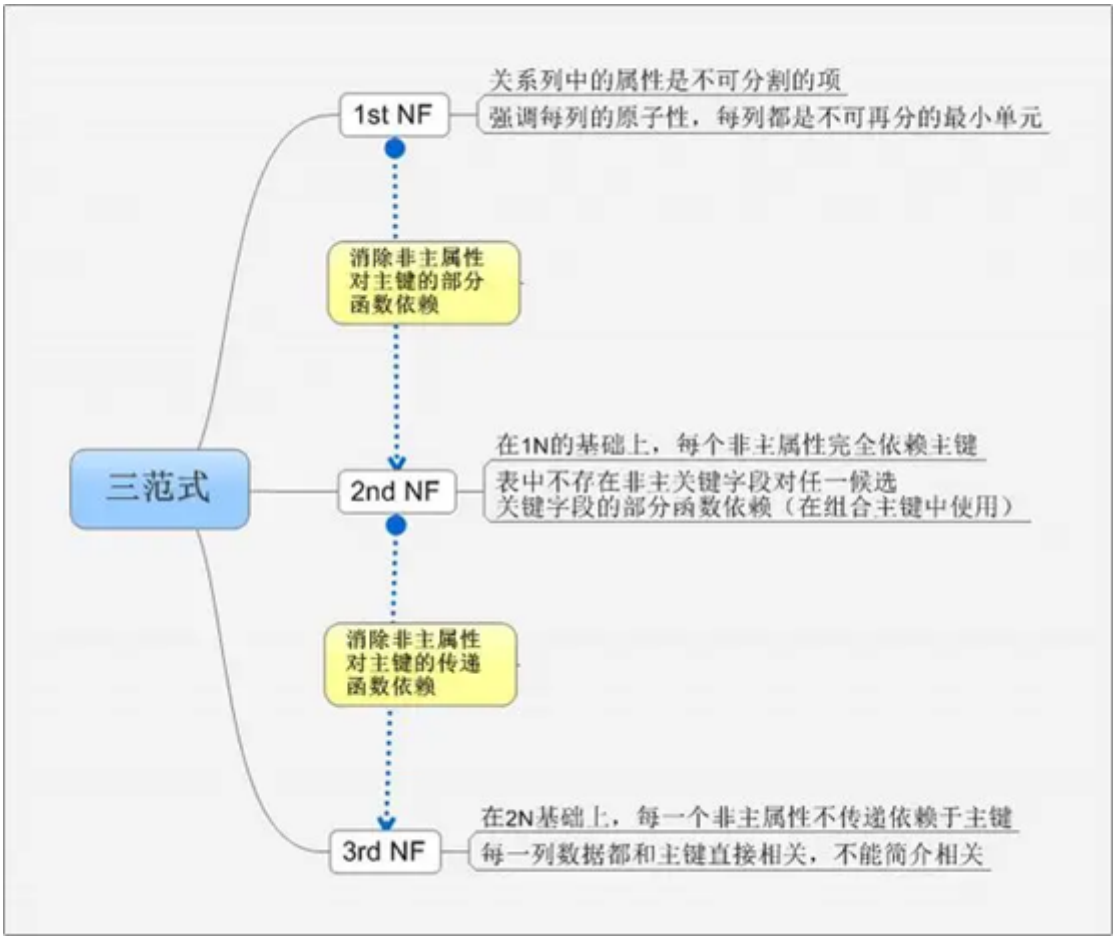
2. 数据库设计原则

数据库的设计，遵循一定的规范，一般是按照三范式的规则进行设计。理论上还有第四范式和第五范式，实际中用的相对较少。

第一范式（1NF）是每一列不可再分，保证列的原子性。其实这也是和业务挂钩的，根据业务需要，没有必要再分，也就符合了1NF。比如人的姓名，一般情况就不用再分，已经是原子属性了。但对一些特殊业务场景，就可能会继续拆分，比如家谱系统，需要区分姓和名，这时候姓名就需要进一步划分，这里的原子性还是针对业务需求来说的。

第二范式（2NF）是在满足1NF的基础上，确保表的非主键列都和主键相关，也就是表的一行只记录一种数据，实际上是一个拆表的动作。这样行的所有数据都和主键相关。比如学生表中班级信息，如果班级包括多个字段（年级，班级名称，班主任等），班级信息需要单独做一个表。需要加外键的动作，将班级id加到学生表，建立学生与班级的关联。

第三范式（3NF）是在满足2NF的基础上，保证表的非主键列和主键直接相关，而不是间接相关，而且非主键列之间不能存在依赖关系。比如学生表：学号，姓名，年龄，班主任，班主任电话。其中关键字为单一关键字"学号"。存在依赖传递:：(学号) → (班主任) → (班主任电话) 。将学生表与班主任表进行拆分，学生表：学号， 姓名， 年龄， 班主任；班主任表：班主任姓名，电话。



依据三范式设计规范，可以建立冗余较小、结构合理的数据库。在实际项目的数据库设计中，最重要的是看需求和性能，根据需求和性能来确定表的结构，也不能非常教条的去追求范式建立数据库。比如资产管理，最主要的资产信息和资产的资源，是两个表，由于前台显示需要将信息和资源一起显示，如果查询两个表，效率就会比较低，为了查询快速，可以将资产信息表和资产资源表合并为一个表，会有大量的冗余信息，是不符合三范式规则的，但有利于业务需求，还是合并处理比较好一些。

3. 设计约定

在数据库设计过程中，遵守的一些设计约定，可以在设计原则的基础上，进一步规范数据库设计的细节要求。三范式可以认为是战略上的要求，是一个总的原则，设计约定则是战术上的要求，很具体，需要在实践中执行。

1）表达是/否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint（0表示否，1表示是）

> 说明:检查is__xxx的命名方式是为了明确其取值含义与范围；任何字段如果为非负数，必须是unsigned
>
> 注意：使用tinyint类型，默认值为4，页可设为1，即tinyint(1)等价于boolean;POJO类中的任何布尔型的变量，都不允许加is前缀。

2）表名、字段名必须使用小写字母或数字，禁止出现数字开头，表的命名最好是加上“业务名称_表的作用”。

> 说明： MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。如，edu_teacher，sys_menu

3）表名不使用复数名词

> 说明： 表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

4）varchar 是可变长字符串，不预先分配存储空间，长度不要超过5000，如果存储长度大于此值，定义字段类型为 text，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

> 说明： 该表的命名以 原表名_字段缩写 的格式命名。

5）表必备字段： id， create_by，create_time，update_by， update_time，remark。也可以叫做Who字段，就是每个表里必须具备的字段。这些字段起到似metadata的作用。这些字段的作用很大，例如，数据分析的时候，可以使用update_time作为数据抽取的时间戳字段等。

id 必为主键，类型为 unsigned bigint、单表时自增、步长为 1。

create_time 是此条数据的创建时间，数据类型为datetime 类型。

update_time是此条数据的最后更新时间，数据类型为 datetime 类型。

create_by是此条数据的创建人。

update_by是此条数据的最后更新人。

remark是此条备注信息

除特别说明外，所有字段默认都设置不充许为空，需要设置默认值。

4. 数据库安全管理

数据库系统是整个系统的核心，是所有业务管理数据以及清算数据等数据存放的中心。数据库的安全直接关系到整个系统的安全。在本系统中对此考虑如下：

数据库管理员(SA)的密码应由专人负责，密码应该定期变换。

客户端程序连接数据库的用户绝对不能使用数据库管理员的超级用户身份。

客户端程序连接数据库的用户在数据库中必须对其进行严格的权限管理，控制对数据库中每个对象的读写权限。

利用数据库的审计功能，以对用户的某些操作进行记录。

充分使用视图以及存储过程，保护基础数据表。

对于不同的应用系统应建立不同的数据库用户，分配不同的权限。

5. 数据库备份管理

在系统运行过程中，经常会由于设备以及其他因素的原因，导致系统的崩溃，数据库的毁坏。为了系统数据安全，无论采用何种系统备份方案，也必须进行数据备份。在系统设计中，应建立一套有效的备份策略，建立完善的备份制度。在本系统中考虑如下：

备份方式可采用完全备份与增量备份相结合方式进行备份；

备份时间频度应结合系统的数据增量来确定，如每天一次、每周一次等；

对系统数据库也需定期备份，但备份时间可以是每月一次，但在系统表有所变化时，必须当天进行备份；

备份介质可为磁带、可擦写光盘或MO等可移动介质，绝对避免使用本机硬盘；

备份设备以及介质必须定期检查和维护，保证备份工作不能由于设备以及介质的原因而耽误；

定期对于备份的正确性和完整性进行检验；

备份工作必须由专人负责，备份介质专人保管，确保备份数据的安全；

当系统发生故障时，应及时利用备份文件，该系统恢复至最近的完整状态，并通知用户及时补充故障期间丢失的数据，直至恢复到系统发生故障前正确的状态。