# OpenStore设计思想

## 系统目标和开发模式的颠覆

引言：当前软件开发的系统，数据存储占据着相当重要的部分。从几十年前一统天下的关系数据库存储数据，到如今大数据、人工智能大背景之下的文档型数据库（以mongodb为首）、KV数据库（以redis为首）、列式存储（hbase，rcfile文件等）、图形存储（neo4j、cayley等）。每一个数据存储方式，都直接决定了软件系统的设计架构、开发工具、开发模式、甚至直接影响了程序猿们的“开发习俗”（注：用“习俗”而不用“习惯”，一字之差，却能真实表达这个事实，习惯代表了个个人的常态，而习俗代表了一个团体，一个社区的约定俗成，虽然这个约定俗成在现在看来，并非最好或最正确）。

而在这些底层存储机制之上，又不断涌现出各种开发框架和平台，例如：关系数据库的各类数据库管理系统：sqlserver系列\mysql等；graphsql图形计算平台等。这些工具的推广和成型，直接导致了开发者，既享受了这些工具和平台带来的关于“数据存储“方面的操作便捷性，统一性。又不得不陷入到这些平台工具的”习俗“之中。

### 传统 vs 当下

关于数据存储，正如引言中谈到。我们可以在这里将“传统“缩小一定的范围，将其只限定于”企业应用“范围。对企业应用来说，”传统“的数据存储机制，毫无疑问，当属《关系型数据库》，没有之一。这除了历史原因决定外，还是由“关系数据库”本身的特性所决定的：

1. 对企业应用来说，大部分的业务数据都习惯性的按照“表格”方式组织，企业中每个部门，每个员工在执行日常事务时，都习惯了这种描述数据，存储数据的方式。
2. 大部分的企业应用中的“表格”式数据之间，业务上都是有“关联”的，并且这种关联，在企业以及企业员工的“思维定式”中，都是自动具有“强制性关联“的。例如：购物车id一定关联着购物车中的商品id
3. 传统技术的“教育“，在设计企业应用时，为了数据所谓的”完整性“、”一致性“等这些数据”约束“，关系数据库将这些概念植入了设计和开发人员的骨髓，数据的schema模式、事务、一对多，一对一等概念已经成为了”数据设计“的”基本常识“
4. 由于DBMS管理系统的普及和强大，以及操作方式SQL语言的统一便捷，使得开发企业应用数据存储部分的开发人员，不必费时费力去思考特定环境下“数据”的各种存储、查询等特性。直接一条简单的sql语句，即能完成相对比较复杂的企业应用中数据的新增、搜索、排序等操作

而当下，随着大数据和人工智能的流行，企业应用中的各种功能、业务、逻辑也面临着各种挑战和改变。同时，导致了企业应用中的《数据存储》部分，也有了翻天覆地的变化，主要表现在如下几方面：

1. 企业组织机构的复杂以及业务的延伸，导致企业每天产生的数据量呈几何级数增长，这导致了之前关系数据库中很难得出现的分库、分表、集群等“小场景“应用功能提上了日程。而关系库或关系库DBMS中的分库、分表、集群等操作和配置，又是及其复杂的
2. 由于当前互联网时代的快速发展，导致企业要面临的业务随时需要灵活的频繁变化，这导致了企业应用中的各种“数据“，并非一定是要模式化的，结构化的。【无模式】、【无类型】的由用户可以灵活定义的”数据“变得原来越受欢迎
3. 企业应用中所面临的现代“数据“，并非一切都是”扁平结构“的二维表所能表达的。或者说，用扁平结构的两维表有时候很难清晰简单的表达企业应用中涉及的数据，例如：分层次的企业组织机构和各类角色数据、企业客户的网络关系分布图等这些数据，用两维表来表达这种数据，【表结构】以及【表之间的关联】将不得不设计的非常复杂，从而导致从用户的“视角”根本无法直观的理解
4. 在当前互联网快速响应的整体风格下，企业应用中对数据的管理，有时并非要求的非常严格。也就是说，传统关系数据库要求的各种范式、事务、强一致性、外键约束等被越来越多的“弱化”！用户提交的一条评论，管理员一定要实时的查看到么？售后服务人员删除的一条数据，查询页面立马就要消失么？如果订单提交失败，预付款是否需要立即回滚？删除了订单，订单项中的商品如果存在，就一定非法么？当前的实践证明，关系数据库提出的严格数据范式，太理论化了。现实世界中的数据，应该将关系弱化、应该将一定范围的错误包容，才能为灵活的企业业务提供强有力的支持！

### 数据存储的开发模式

当要制作一个企业应用时（这里只描述数据部分），传统的设计方法如下：

* 需求分析人员分析出系统业务所涉及的数据结构，将客户方提出的无模式、非结构化的“业务数据”模式化、结构化
  + 1. 将客户直接的“业务概念”归类，例如：商品、订单、库存
    2. 将归类的数据分解成“组成字段“：例如，商品中包括名称、SKU、品牌等
    3. 组织归类数据之间的关系：例如，一个订单中包括多个商品
* 开发（设计）人员，根据需求分析员提供的“结构化“数据，开始设计”表结构“
  + 1. 将归类概念设计成“表“
    2. 将组成字段，设计成字段，并设定字段的类型，例如SKU为整数、名称为64位字符串
    3. 为了表达表之间的关系，在表字段中添加额外的【关系型】字段：外键，例如品牌id，订单id等【这些字段对业务方毫无意义，但对系统运行有着至关重要的意义】
    4. 为了企业应用上的一些特殊逻辑，添加额外的字段（我们称其为：计算字段）。例如：为了表达数据的“被删除”行为，额外添加：isDeleted字段、为了表达一些算法而添加的一些字段等
    5. 为了建立一些表之间的关系，纯粹添加一些毫无意义的“中间表“，这些表纯粹只为提供一些映射关系，没有任何业务意义
    6. 为了配合关系库的“约束”，设置一些字段的“数据约束”，例如：用户评论不能超过255个字符、商品订单id不能为null等
    7. 有时，为了sql语句查询的方便，以及降低sql语句语法的复杂度，特殊设计一些字段来降低sql风险。例如left join，right join，子查询优化等

传统“数据开发模式“，带来的”习俗“的好处是不言而喻的，但弊端随着时代的变化也越来越明显：

1. 业务数据和物理数据是脱离的。在用户视角，根本无法理解他的“业务数据”和数据库中表、字段、关系、约束有什么对应关系。
2. 一个完整的“业务数据“，纯粹为了系统和”习俗“，而被人为分解成了多张表进行了存储，而这些分散的表，又是通过”业务无法理解“的方式，关联在一起。例如，一个订单可能被分解成了三四张表进行了存储，而这三四张表通过用户不理解的各种主外键id关联
3. 数据的“主体内容”和“附加属性“都被添加到了一起，不仅占用了空间还降低了查询性能。例如：一个订单，有其主体内容，而订单的”创建者“、”修改者“、”创建时间“等，不过是订单的”附加属性“而已。但表结构中，也必须设计字段存储这些附加属性
4. 单纯从实现上来说，表结构是紧密存储结构而并非稀疏存储结构。因此，一张表无法简单稀疏化的分布在各个不同的节点上

基于以上的讨论，openstore如何改变传统的开发模式呢？openstore主要针对以下几点进行改变：

* 一个“业务概念“对应一个FileFamily文件族，一个FileFamily文件族代表着【业务上相同的概念】。例如：业务上，“采购单据”都是相同的概念，因此，他应该被存储在Family为PurchseOrder的一个或多个File中（也就是说，一个Family可以由一个文件存储，也可以有多个文件存储）。一个“文件族”和业务上的某一个“概念”是一对一对等的，例如：企业应用有如下概念：采购单、出库单、入库单、售后服务单、配件申请单、外呼任务单、核销单….那么openstore中就有7个FileFamily
* 每一个FileFamily都需要指定一个Format文件格式，文件格式就是在一个文件系统中的文件类型，如：Excel、Word、PDF、txt….。Openstore通常支持普通人员能够理解的常用文件（比如excel，pdf等），也支持专业开发能够理解的文件（json、avro序列化文件），同时也支持大数据环境下的各类文件格式（如：hadoop mapfile、sequenceFile、RCFile，ORCFile、hbasefile等），自定义文件格式的解析器，也可以随时加入openstore进行扩展。
* 每一个“业务概念“中的具体某一项，都对应一个File文件。

例如：

1. 某个具体的1号采购单，就是一个采购单.0001.xls文件
2. 名称为：XXXXX外呼任务单，就是一个”外呼任务.orc.it”文件（orc文件中的一条记录）

* Openstore中File都是逻辑上的概念，也就是说，一个FileFamaily只是一个逻辑上的File集合，一个File只是一个逻辑上的“文件“，实际一个File数据存储在物理上是否是一个实际存储盘上的一个或多个物理文件，通常可以自定义调整，或者根据每种Family的Format默认决定；如：orc文件格式，就只有一个文件。

总结一下，如果使用openstore来设计企业应用数据存储，可以按照如下整体流程进行：

找出业务概念【如：采购单】

使用openstore生成对应的FileFamily

具体生成细节，后续章节描述

选择FileFamily文件格式

普通：excel/pdf/txt…

专业：json/avro…

大数据mapfile/sequenceFile/orcfile

定义File各组成部分

实际创建File，向File输入数据

就像你在C盘上新建了一个《采购单001.xls excel文件，随后的输入数据，就是在此excel文件中编辑采购单各个组成部分