# 数据库与ER建模

## 1.1.1 数据库（DataBase）

数据库是按照数据结构来组织、存储和管理数据的仓库，是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合。

数据库是以一定方式存储在一起，能与多个用户共享，具有尽可能小的冗余度、与应用程序彼此独立的数据集合，可视为电子化的文件柜，存储电子文件的处所，用户可以对文件中的数据进行新增、查询、更新、删除等操作，数据组织主要是面向事务处理任务。

## 1.1.2 数据库三范式

关系数据库设计时，遵照一定的规范要求，目的在于降低数据冗余性和保证数据的一致性，这些规范就可以称为范式NF（Normal Form），大多数情况下，关系型数据库的设计符合三范式即可。

**第一范式（1NF）**：原子性，字段不可分

即表的列的具有原子性，不可再分解，即列的信息，不能分解。数据库表的每一列都是不可分割的原子数据项，而不能是集合，数组，记录等非原子数据项。如果实体中的某个属性有多个值时，必须拆分为不同的属性。通俗理解即一个字段只存储一项信息。

举例如下，有如下订单表，购买商品一列，可以拆分为商品和数量两列。



**第二范式（2NF）**：唯一性，一个表只能说明一个事物，有主键，非主键字段依赖主键

第二范式是在第一范式的基础上建立起来的，第二范式（2NF）要求数据库表中的每个实例或行必须可以被唯一地区分，为实现区分通常需要我们设计一个主键来实现，当存在多个主键的时候，不能存在这样的属性，它只依赖于其中一个主键，这就是不符合第二范式，通俗理解是任意一个字段都只依赖表中的同一个字段，举例如下：



学生信息表中的成绩依赖于学号和课程号两个主键，但是课程学分只是依赖课程的，可以拆分成右侧的两张表。

**第三范式（3NF）**：非主键字段不能相互依赖，不存在传递依赖

满足第三范式必须先满足第二范式，第三范式（3NF）要求一个数据库表中不包含已在其他表中已包含的非主键字段（某张表的某字段信息，如果能够被推导出来，就不应该单独的设计一个字段来存放）。

如果某一属性依赖于其他非主键属性，而其他非主键属性又依赖于主键，那么这个属性就是间接依赖于主键，这被称作传递依赖于主属性。第三范式中要求任何非主属性不依赖于其它非主属性，即不存在传递依赖。很多时候，我们为了满足第三范式往往会把一张表分成多张表

举例如下：



第一张表中“院校地址”是依赖于“院校”的，“院校”依赖于主键“学号”，存在传递依赖，不符合第三范式，那么需要拆解成对应的学生表和院校表两张表。

注意：三大范式只是一般设计数据库的基本理念，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，提供读性能，就必须降低范式标准，降低范式就是增加字段，减少了查询时的关联，提高查询效率，适当保留冗余数据，这就是反范式，反范式化一定要适度，并且在原本已满足三范式的基础上再做调整的。

## 1.1.3 ER实体关系模型

ER实体关系模型（Entity-Relationship）是数据库设计的理论基础，当前几乎所有的OLTP系统设计都采用ER模型建模的方式，这种建模方式基于三范式。在信息系统中，将事物抽象为“实体”、“属性”、“关系”来表示数据关联和事务描述。

**实体**（Entity）：实体是一个数据对象，指应用中可以区别的客观存在的事物。例如：商品、用户、学生、课程等。它具有自己的属性，一类有意义的实体构成实体集。在ER实体关系模型中实体使用方框表示。

属性：对实体的描述、修饰就是属性，即：实体的某一特性称为属性。例如：商品的重量、颜色、尺寸。用户的性别、身高、爱好等。在ER实体关系模型中属性使用椭圆来表示。

**关系**（Relationship）：表示一个或多个实体之间的关联关系 。实体不是孤立的，实体之间是联系的，这就是关系。例如：用户是实体，用户选购商品这个过程就会产生“选购商品数量”，“金额”这些属性，这就是关系。再如：学生是实体，课程是实体，学生选择课程这个过程就产生了“课程数量”、“分数”这些属性，这就是关系。在ER实体关系模型中关系使用菱形框表示，并用线段将其与相关的实体链接起来。

ER实体关系模型又叫E-R关系图，实体与实体之间的关系存在一对一的关系、一对多的关系、多对多的关系。

**一对一关系**：例如：“学生”是实体，“身份证”是实体，一个学生只能有一个身份证，一个身份证也只能对应一个学生。

**一对多关系**：一对多关系反过来就成了多对一的关系。例如：“学生”是实体，“账号”是实体，一个学生有多个账号，反过来就是多个账号对应一个学生。

**多对多关系**：例如：“学生”是实体，“课程”是实体，一个学生可以学习多个课程，

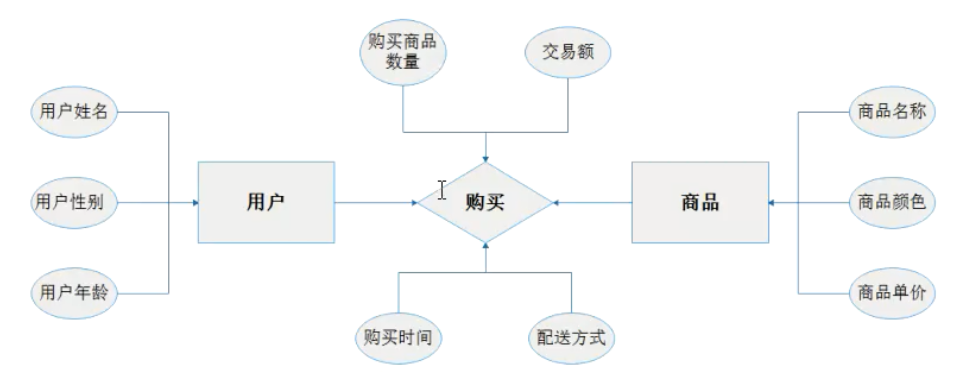
一个课程可以被多个学生学习，整体来看，学生学习课程就成了多对多的关系。

## 1.1.4 ER实体关系模型案例

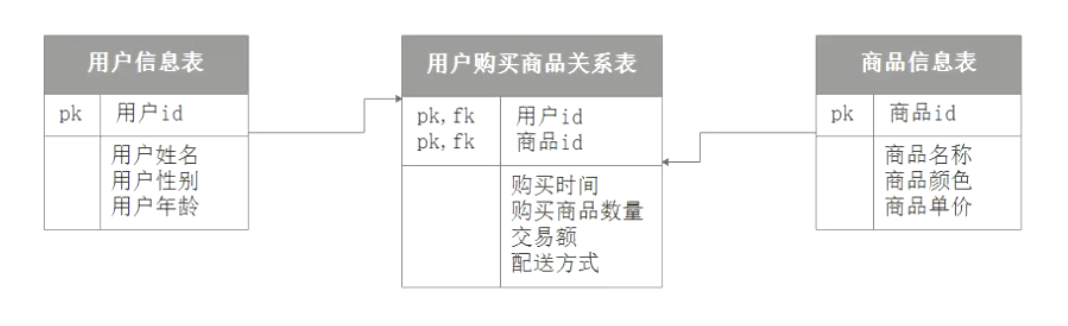
假设在电商购物系统中，对商品、用户设计ER实体关系模型图来表示商品信息、用户购买商品之间的业务联系，可以数据库逻辑模型设计。

设计ER实体关系模型图，步骤如下：

1. 抽象出实体
2. 找出实体之间的关系
3. 找出实体的属性
4. 画出E-R关系图



以上是ER实体关系图，为了方便，我们一般可以将ER实体关系图转换成如下数据库表格式，IDEFIX格式



# 数据仓库与维度建模

### 数据仓库（Data Warehouse）

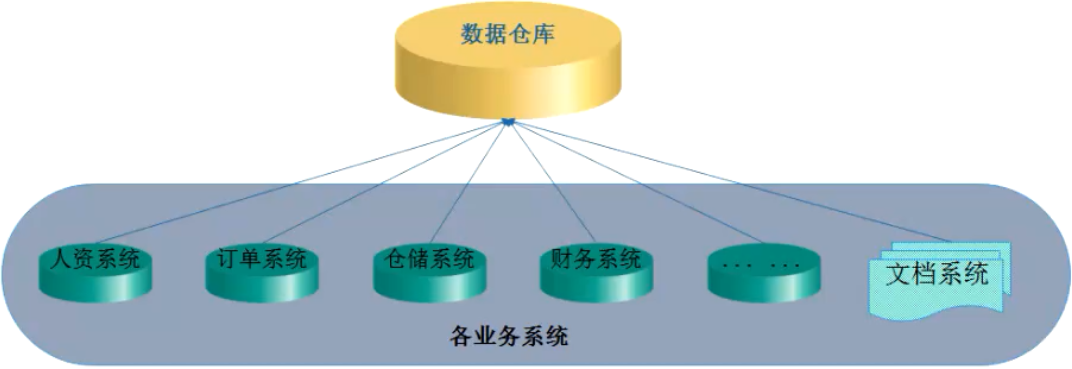
在日益激烈的商业竞争中，企业迫切需要更加准确的战略决策信息。在以往的关系型数据库系统中，企业拥有海量的数据，这些数据对于企业的运作非常有用的，但是对于商业战略决策和目标制定的作用甚微，不是战略决策要使用的信息。

关系型数据库很难将这些数据转换成企业真正需要的决策信息，原因如下：

1. 一个企业中可能有多个管理系统平台，企业数据分散在多种互不兼容的系统中。例如：一个银行中的系统分为：核心系统，信贷系统，企业贷款系统，客户关系系统，助学贷款系统，理财系统、反洗钱系统等，这些系统数据有可能存储在不同类型的关系型数据库中。
2. 关系型数据库中存储的数据一般是最基本的、日常事务处理的、面向业务操作的数据，数据一般可以更新状态，删除数据条目等。不能直接反应趋势的变化。例如：用户登陆网站购买商品，在关系型数据库中最终存储的数据是某个用户下了一个订单，订单状态为付款待发货，一般用户在网站浏览了什么商品，搜索了什么样的关键字，这些数据不会存储在关系型数据库中，往往这些数据更具有价值。
3. 对于战略决策来说，决策者必须从不同的商业角度观察数据，比如说产品、地区、客户群等不同方面观察数据，关系型数据库中数据不适合从不同的角度进行分析，只是面向基本的业务操作。

所以我们需要对企业中各类数据进行汇集，清洗，管理，找出战略决策信息，这就需要建立数据仓库。

数据仓库，Data Warehouse，可简写为DW或DWH。数据仓库是面向主题的、集成的（非简单的数据堆积）、相对稳定的、反应历史变化的数据集合，数仓中的数据是有组织有结构的存储数据集合，用于对管理决策过程的支持。



面向主题：主题是指使用数据仓库进行决策时所关心的重点方面，每个主题都对应一个相应的分析领域，一个主题通常与多个信息系统相关

例如：在银行数据中心平台中，用户可以定义为一个主题，用户相关的数据可以来自信贷系统、银行资金业务系统、风险评估系统等，以用户为主题就是将以上各个系统的数据通过用户切入点，将各种信息关联起来。如下图所示：

数据集成：数据仓库中的数据是在原有分散的数据库数据抽取、清理的基础上经过系统加工、汇总和整理得到的，必须消除源数据中的不一致性，以保证数据仓库内的信息是关于整个企业的一致的全局信息，这个过程中会有BTL操作，以保证数据的一致性、完整性、有效性、精确性。

例如某公司中有人力资源系、生产系统、财务系统、仓储系统等，现需要将各个系统的数据统一采集到数据仓库中进行数据分析，在人力系统中，张三的性别为“男”，坑你在财务系统中张三性别为“M”，在人力资源系统中张三职称为“生产部员工”，在生产系统中职称为“技术经理”，那么当我们将数据抽取到数据仓库中时，需要经过数据清洗，将数据进行统一、精确、一致性存储。

相对稳定：数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一旦某个数据进入数据仓库以后，一般情况下将被长期保留，也就是数据仓库中一般有大量的查询操作，基本没有修改和删除操作，通常只需要定期的加载、刷新。

例如：某用户在一天中多次登录某系统，关系型数据库中只是记录当前用户最终在系统上的状态是“在线”还是“离线”，只需要记录一条数据进行状态更新即可。但是在数据仓库中，当用户多次登录系统时，会产生多条记录，不会存在更新状态操作，每次用户登录系统和下线系统都会在数据仓库中记录一条信息，这样方便后期分析用户行为。

反应历史变化：数据仓库中的数据通常包含历史信息，系统地记录企业从过去某一时点（如开始应用数据仓库的时点）到当前的各个阶段的信息，通过这些信息，可以对企业的发展历程和未来趋势做出定量分析和预测。

例如：电商网站中，用户从浏览各个商品，到将商品加入购物车，直到付款完成，最终的结果在关系型数据库中只需要记录用户的订单信息。往往用户在网站中的浏览商品的信息

行为更具价值，数据仓库中就可以全程记录某个用户登陆系统之后浏览商品的浏览行为，加入购物车的行为，及付款行为。以上这些数据都会被记录在数据仓库中，这样就为企业分析用户行为数据提供了数据基础。

### 数据仓库技术的发展历程

1. **萌芽阶段**（1978~1988年）。

数据仓库概念最早追朔到20世纪70年代，MIT（麻省理工）的研究员致力于研究一种优化的技术架构，该架构试图将业务处理系统和分析系统分开，即将业务处理和分析处理分为不同层次，针对各自的，特点采取不同架构设计原则，MIT的研究员认为这两种信息处理的方式具有显著差别，以至于必须采取完全不同的架构和设计方法。但受限于当时的信息处理能力，这个研究仅仅停留在理论层面。

1. **探索阶段**。

20世纪80年代中后期，DEC公司（美国数字设备公司）结合MIT的研究结论，建立了TA2（Technical Architecture2）规范，该规定定义了分析系统的四个组成部分：数据获取、数据访问、目录和用户服务。其中的数据获取和数据访问目前大家都很清楚，而目录服务是用于帮助用户在网络中找到他们想要的信息，类似于业务元数据管理；用户服务用以支持对数据的直接交互，包含了其他服务的所有人机交互界面，这个系统架构的一次重大转变，第一次明确提出分析系统架构并将其运用于实践。

1. **雏形阶段**（1988年）

1988年，为解决全企业集成问题，IBM公司第一次提出了信息仓库（InformationWarehouse）的概念：“一个结构化的环境，能支持最终用户管理其全部的业务，并支持信息技术部门保证数据质量”，并称之为VITAL规范（VirtuallyIntegrated Technical Architecture Lifecycle）。并定义了85种信息仓库的组件，包括数据抽取、转换、有效性验证、加载、Cube开发和图形化查询工具等。至此，数据仓库的基本原理、技术架构以及分析系统的主要原则都已确定，数据仓库初具雏形。

1. **确立阶段**（1991年）

1991年，比尔·恩门（Bill Inmon）出版了他的第一本关于数据仓库的书《Building the Data Warehouse》,标志着数据仓库概念的确立。该书定义了数据仓库非常具体的原则，包括：

数据仓库是面向主题的（Subject-Oriented）、

集成的（Integrated）、

包含历史的（Time-variant）、

不可更新的（Nonvolatile）、

面向决策支持的（Decision Support）、

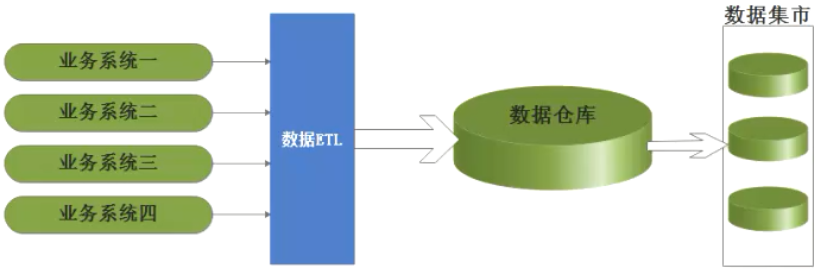
面向全企业的（Enterprise Scope）、

最明细的数据存储（Atomic Detail）、

数据快照式的数据获取（Snap Shot Capture）

这些原则到现在仍让是指导数据仓库建设的最基本原则。凭借着这本书，Bill Inmon被称为“数据仓库之父”。

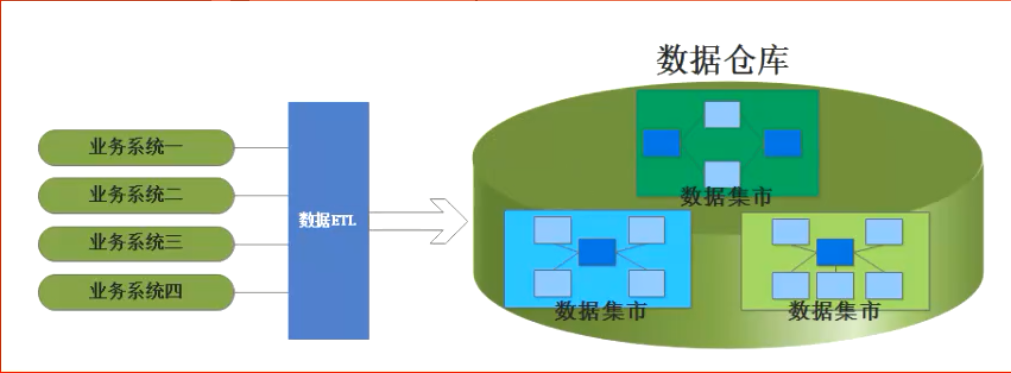
Bill Inmon 主张自上而下的建设企业级数据仓库EDW（Enterprise Data Warehouse）,认为数据仓库是一个整体的商业智能系统的一部分，一家企业只有一个数据仓库，数据集市（数据集市中存储为特定用户需求而预先计算好的数据，从而满足用户对性能的需求）的信息来源出自数据仓库，在数据仓库中，信息存储符合第三范式，大致结构如下：



由于企业级数据仓库的设计、实施很困难，使得最早吃数据仓库螃蟹的公司遭到大面积的失败，除了常见的业务需求定义不清、项目执行不力之外，很重要的原因是因为其数据模型设计，在企业级数据仓库中，Inmon推荐采用3范式进行数据建模，但是不排除其他的方法，但是Inmon的追随者固守OLTP系统的3范式设计，从而无法支持决策支持（DSS-Decision Suport System）系统的性能和数据易访问性的要求。

1994年前后，实施数据仓库的公司大都以失败告终，这时，拉尔夫·金博尔（Ralph Kimball）出现了，他的第一本书《The DataWarehouse Toolkit》掀起了数据集市的狂潮，这本书提供了如何为分析进行数据模型优化详细指导意见。

Ralph Kimball主张自下而上的建立数据仓库，极力推崇建立数据集市，认为数据仓库是企业内所有数据集市的集合，信息总是被存储在多维模型当中，为传统的关系型数据模型和多维OLAP之间建立很好的桥梁，其思路如下



1. **争吵与混乱**（1996-1997年）

Bill Inmon（比尔·恩门）的《Building the Data Warehouse》主张建立数据仓库时采用自上而下方式，以第3范式进行的数据仓库模型设计，而Ralph Kimball（拉尔夫·金博尔）在《the DataWarehouse Tookit》则是主张自下而上的方式，力推数据集市建设。企业级数据仓库还是部门级数据集市？关系型还是多维，Bill Inmon 和 Ralph Kimball一开始就争论不休，其各自的追随者也唇舌相向，形成对立的两派：Inmon派和Kimball派

在初期，数据集市的快速实施和较高的成功率让Kimball派占了上风，由于数据集市仅仅是数据仓库的某一部分，实施难度大大降低，并且能够满足公司内部部分部门的迫切需求，在初期获得了较大的成功。但是很快，他们也发现自己陷入了某种困境：随着数据集市的不断增多，这种架构的缺陷也逐步显现，公司内部独立建设的数据集市由于遵循不同的标准和建设标准，以致多个数据集市的数据混乱和不一致。解决问题的方法只能是回归数据仓库的最初的基本建设原则上。

6、**合并**（1998-2201年）

两种思路和观点在实际的操作中都很难成功完全项目交付，1998年Bill Inmon提出了新 的BI架构CIF（Corporation infomation factory）,把Kimball的数据集市也包容进来。CIF的核心是将数仓架构划分为不同的层次以满足不同场景的需求，比如常见的ODS、DW、DM等，每层根据实际场景采用不同的建设方案，现在CIF已经成为建设数据仓库的框架指南，但自上而下还是自下而上的进行数据仓库建设，并未统一。

以上就是到目前为止，整体数仓建设来源的过程。

## 维度建模

维度建模主要源自数据集市，主要面向分析场景。

维度建模以分析决策的需求出发构建模型，构建的数据模型为分析需求服务，因此它重点解决用户如何更快速完成分析需求，同时还有较好的大规模复杂查询的响应性能。它与实体-关系（ER）建模有很大的区别，实体-关系建模是面向应用，遵循第三范式，以消除数据冗余为目标的设计技术。维度建模是面向分析，为了提高查询性能可以增加数据冗余，反规范化 的设计技术。

Ralph Kimball（拉尔夫·金博尔）推崇数据集市的集合为数据仓库，同时也提出了对数据集市的维度建模，将数据仓库中的表划分为事实表、维度表两种类型

·**事实表**

发生在现实世界中的操作型事件，其所产生的可度量数值，存储在事实表中。例如，一个按照地区、产品、月份划分的销售量和销售额的事实表如下：



1. 在以上事实表的示例中，“地区ID”、“产品ID”、“月份ID”为键值列，“销售量”、“销售额”为度量列，所谓度量列就是列的数据可度量，度量列一般为可统计的数据列。事实表中每个列通常要么是键值列，要么是度量列。
2. 事实表中一般会使用一个代号或者整数来代表维度成员，而不使用描述性的名称，例如：ID代号，上表中的“地区ID”、“产品ID”、“月份ID”就是维度列，就是观察数据的角度。使用代号或整数来代表维度成员的原因是事实表往往包含很多数据行，使用代号或整数这种键值方式可以有效减少事实表的大小。
3. 在事实表中使用代号或者整数键值时，维度成员的名称需要放在另一种表中，也就是维度表。通常事实表中的每个维度都对应一个维度表。
4. 在数据仓库中，事实表的前缀为“fact”

·**维度表**

维度表包含了维度的每个成员的特定名称。维度成员的名称称为“属性”（Attribute），假设“产品ID”维度表中有3种产品，例如：



1. 如上图，“产品名称”是产品维度表中的一个属性，维度表中可包含很多属性列。
2. 产品维度表中的“产品ID”与事实表中的“产品ID”相匹配，称为“键属性”，在当前产品维度表中一个“产品ID”只有一个“产品名称”，显示时使用“产品名称”来替代，所以“产品名称”也被认为是“键属性”的一部分。
3. 每个维度表中的键值属性都与事实表中对应的维度相匹配，在维度表中“产品ID”类似关系型数据库中的主键，在事实表中“产品ID”类似关系型数据库中的外键，维度表和事实表就是安装键值属性“产品ID”进行关联的。在维度表中出现一次的每个键值都会在事实表中出现多次。例如上图中，产品ID中的1111在事实表中对应多行。
4. 在数据仓库中，维度表的前缀为“dim”

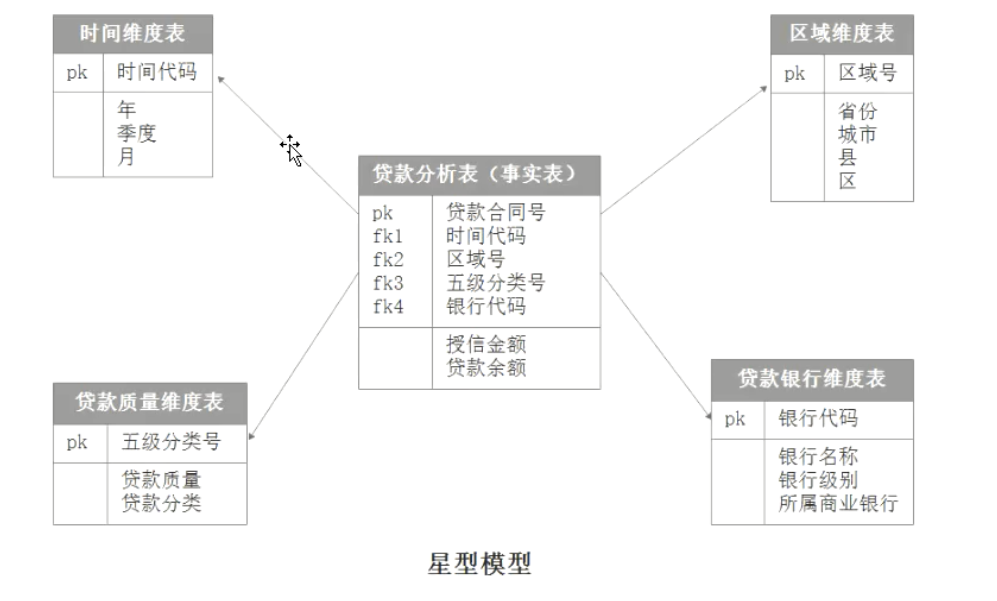
总结：

在数据仓库中事实表就是我们需要关注的内容，维度表就是我们从哪些角度观察这些内容。例如，某地区商品的销量，是从地区这个角度观察商品销量的。事实表就是销量表，维度表就是地区表。

在多维分析的商业智能解决方案中，根据事实表和维度表的关系，又可将常见的模型分为星型模型和雪花模型。

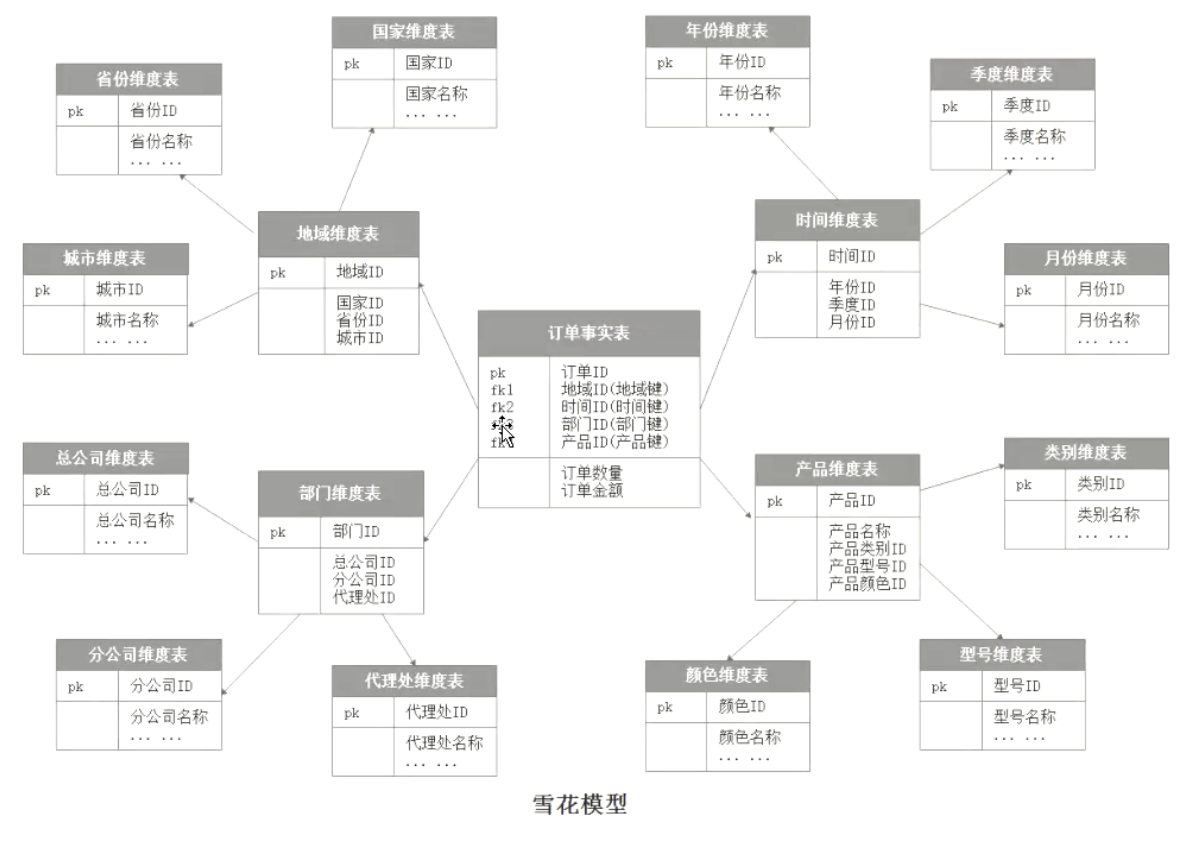
**星型模型**：

当所有的维度表都由连接键连接到事实表时，结构图如星星一样，这种分析模型就是星型模型，如下图，星型架构是一种非正规化的结构，多维数据集的每一个维度都直接与事实表相连接，不存在渐变维度，所以数据有一定的冗余，如在下图中，时间维中存在A年1季度1月，A年1季度2月两条经理，那么A年1季度被存储了2次，存在冗余。



**雪花模型**：

当有一个或多个维表没有直接连接到事实表上，而是通过其他维表连接到事实表上时，其结构图就像雪花连接在一起，这种分析模型就是雪花模型。如下图，雪花模型是对星型模型的扩展。它对星型模型的维表进一步层次化，原有的各维表可能被扩展为小的事实表，形成一些局部的“层次”区域，这些被分解的表都连接到主维度表而不是事实表。如下图中，将地域维表又分解为国家，省份，城市等维表。它的优点是：通过最大限度地减少数据存储量以及联合较小的维表来改善查询性能，雪花型结构去除了数据冗余。



**星型模型与雪花模型对比**：

星型模型和雪花模型主要区别就是对维度表的拆分，对于雪花模型，维度表的设计更加规范，一般符合三范式设计；而星型模型，一般采用降维的操作，维度表设计不符合三范式设计，反规范化，利用冗余牺牲空间来避免模型过于复杂，提供易用性和分析效率。

星型模型因为数据的冗余所以很多统计查询不需要做外部的连接，因此一般情况下效率比雪花型模型要高。星型结构不用考虑很多正规化的因素，设计与实现都比较简单。

雪花型模型由于去除了冗余，有些统计就需要通过表的联接才能生产，所以效率不一定有星型模型高。正规化也是一种比较复杂的过程，相应的数据库结果设计、数据的ETL、以及后期的维护都要复杂一些。因此在冗余可以接受的前提下，数仓构建实际运用中星型模型使用更多，也更有效率。

此外，在数据仓库中星座模型也使用比较多，当多个事实表共用多张维度表时，就构成星座模型。

## 维度建模案例

淘宝电商平台，经常需要对用户订单进行分析，以用户订单为例，使用维度建模的方式来设计模型。

以上涉及到的事实表有：订单明细表。

订单明细表中的维度有用户维度，区域维度，时间维度，商品维度，商户维度。度量有商品件数，商品金额。

用户维度表：用户ID，用户姓名，用户性别，用户年龄，用户职业，用户区域。

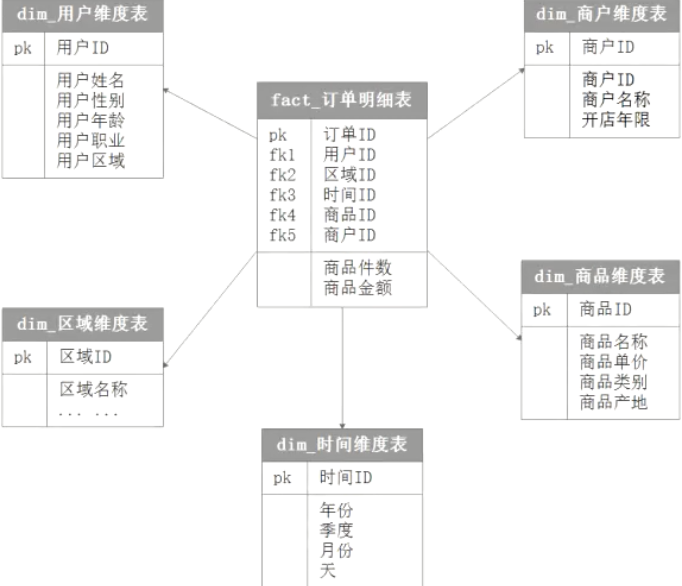
区域维度表：区域ID，区域名称。

时间维度表：时间ID，年份，季度，月份，天。

商品维度表：商品ID，商品名称，商品单价，商品类别，商品产地。

商户维度表：商户ID，商户名称，开店年限。

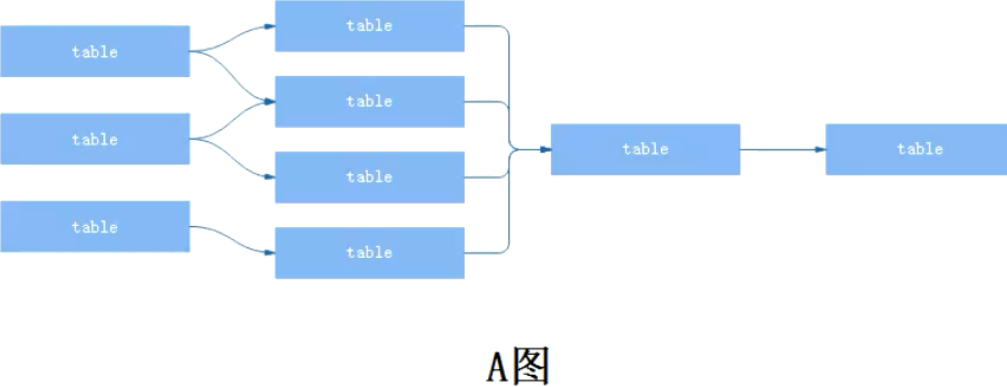
建模设计如下：

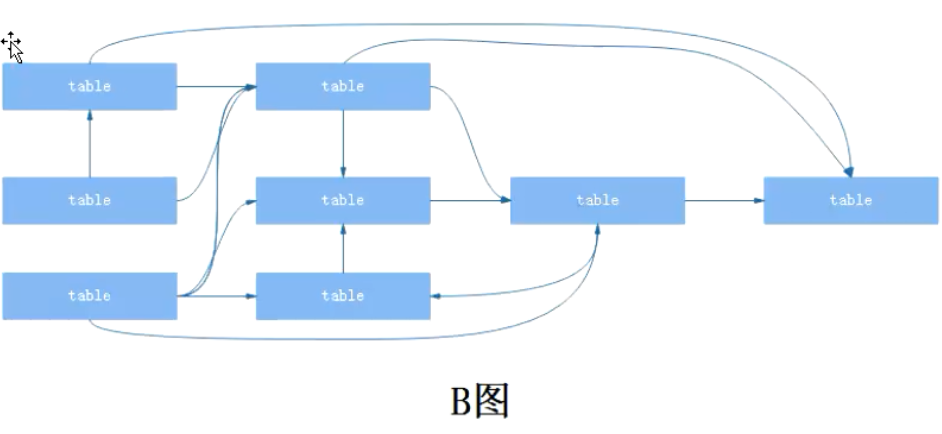


## 数据仓库的分层

### 数据仓库分层设计

在大数据分析中，我们希望我们分析的数据在整个分析流程中有秩序的流转，数据的整个流程能够清晰明确的被我们掌握使用。





上图中的数据流转我们更希望得到A图中的数据流转过程，所以我们希望有一套行之有效的数据组织和管理方法来让我们的数据更有序，这就需要在数据处理中进行数据分层的原因。基于大数据的数据仓库建设要求快速响应需求，同时需求灵活多变，对实时性有不同的要求，除了面向DSS、BI等应用外，还要响应用户画像、个性化推荐、机器学习、数据分析等各种复杂的应用场景，实际上在大数据开发中，涉及到数仓设计都会进行分层设计。

数据分层好处如下:

√ 清晰的数据结构：每层数据都有各自的作用域和职责，在使用表的数据更方便定位和 解释。

√ 减少重复开发：规范数据分层，开发一层公用的中间层数据，减少重复计算流转数据。

√ 统一数据出口：通过数据分层，提供统一的数据出口，保证对外输出数据口径一致。

√ 简化问题：通过数据分层，将复杂的业务简单化，将复杂的业务拆解为多层数据，每层数据负责解决特定的问题。



ODS（Operational Data Store）层 - 操作数据层

ODS层，操作数据层，也叫贴源层，本层直接存放从业务系统抽取过来的数据，这些数据从结构上和数据上与业务系统保持一致，降低了数据抽取的复杂性，本层数据大多是按照源头业务系统的分类方式而分类的。一般来讲，为了考虑后续可能需要追朔数据问题，因此对于这一层就不建议做过多的数据清洗工作，原封不动地接入原始数据即可。

DW（Data Warehouse）层 - 数据仓库层

数据仓库层是我们在做数据仓库时要核心设计的一层，本层将从ODS层中获得的数据安装主题建立各种数据模型，每个主题对应一个宏观的分析领域，数据仓库层排除对决策无用的数据，提供特定主题的简明视图。DW层又细分为DWD（Data Warehouse Detail）层、DWM（Data Warehouse Middle）层和DWS（Data Warehouse Service）层。

数据明细层：DWD（Data Warehouse Detail）

该层一般保持和ODS层一样的数据粒度，并且提供一定的数据质量保证，在ODS的基础上对数据进行加工处理，提供更干净的数据。同时，为了提供数据明细层的易用性，该层会采用一些维度退化手法，当一个维度没有数据仓库需要的任何数据时，就可以退化维度，将维度退化至事实表中，减少事实表和维表的关联。例如：订单id，这种量级很大的维度，没必要用一张维度表来进行存储，而我们一般在进行数据分析时订单id又非常重要，所以我们将订单id冗余在事实表中，这种维度就是退化维度。

数据中间表：DWM（Data Warehouse Middle）

该层会在DWD层的数据基础上，对数据做轻度的聚合操作，生成一系列的中间表，提示公共指标的复用性，减少重复加工处理数据。简单来说，就是对通用的维度进行聚合操作，算出相应的统计指标，方便复用。

数据服务层：DWS（Data Warehouse Service）

该层数据表会相对比较少，大多都是宽表（一张会涵盖比较多的业务内容，表中的字段较多）。按照主题划分，如订单，用户等，生成字段比较多的宽表，用于提供后续业务查询，OLAP分析，数据分发等。

在实际业务处理中，如果直接从DWD或者ODS计算出宽表的统计指标，会存在计算量太大并且维度太少的问题，因此一般的做法是，在DWM层计算出多个小的中间表，然后再拼接成一张DWS的宽表。由于宽和窄的界限不易界定，也可以去掉DWM这一层，只留DWS层，将所有的数据在放在DWS也没有问题。

DM（Data Mart）层 - 数据集市层

数据集市层，也可以称为数据应用层，基于DW上的基础数据，整合汇总成分析某一个主题域的报表数据，主要是提供给数据产品和数据分析使用的数据，一般会存放在ES、PostgreSql、Redis等系统中供线上系统使用，也可能存在Hive或者Druid中提供数据分析和数据挖掘使用。比如我们经常说的报表数据，一般就放在这里。

### 数据仓库分层案例

现在以购物网站电商日志为例，来说明数据仓库分层的使用，这里我们只是关注用户产生的日志这一部分数据。

某购物网站用户可以在pc端，ipad、手机app端、微信小程序登陆购物网站进行购物，用户购物会产生一些订单、浏览商品、地域登录登出等日志数据。为方便后期分析用户日志数据和订单数据更好的辅助决策，设计数据仓库分层如下：



## 数据库与数据仓库区别

数据库：传统关系型数据库的主要应用是OLTP（On-Line Transaction Processing）,主要用于业务类系统，主要供基层人员使用，进行一线业务操作。

数据仓库：数仓系统的主要应用主要是OLAP（On-Line Analytical Processing）,支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。OLAP数据分析的目标是探索并挖掘数据价值，作为企业高层进行决策的参考。



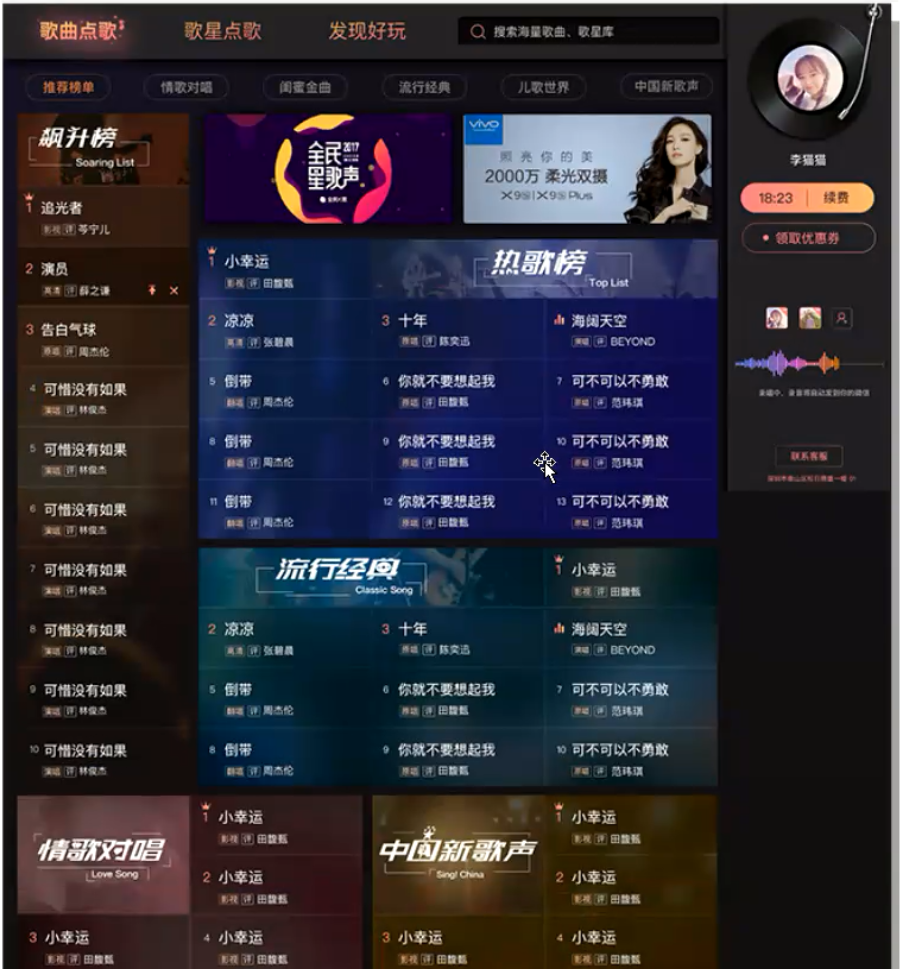
## 项目介绍

音乐数据中心数仓综合项目主要是针对公司过去收集到的用户点播、购买音乐等数据（包括业务数据与用户行为数据），为公司业务更健康的发展提供决策支持（BT商业决策）。













数据中心项目中包含业务系统数据和用户行为日志数据。

业务数据即业务系统数据，例如：系统中产生的订单、登录、点歌、广告展示等数据。

用户行为数据例如：用户在实体机器上操作的行为都是用户行为数据，点击、收藏、扫码等事件。

公司针对以上数据进行分析的结果主要有两个应用：

一是针对BI系统，商业智能中主要展示更多的报表给公司的运营人员参考，例如：每日歌曲点唱量，每日营收，机器分布，实时pv，uv，用户留存率、漏斗模型等。

另一个应用是数据服务，数据服务主要是针对分析后结果数据以接口的形式提供给业务系统来访问，例如：推荐系统，根据歌曲来推荐歌曲，根据歌手来推荐歌曲或者根据用户来推荐歌曲。

## 项目架构

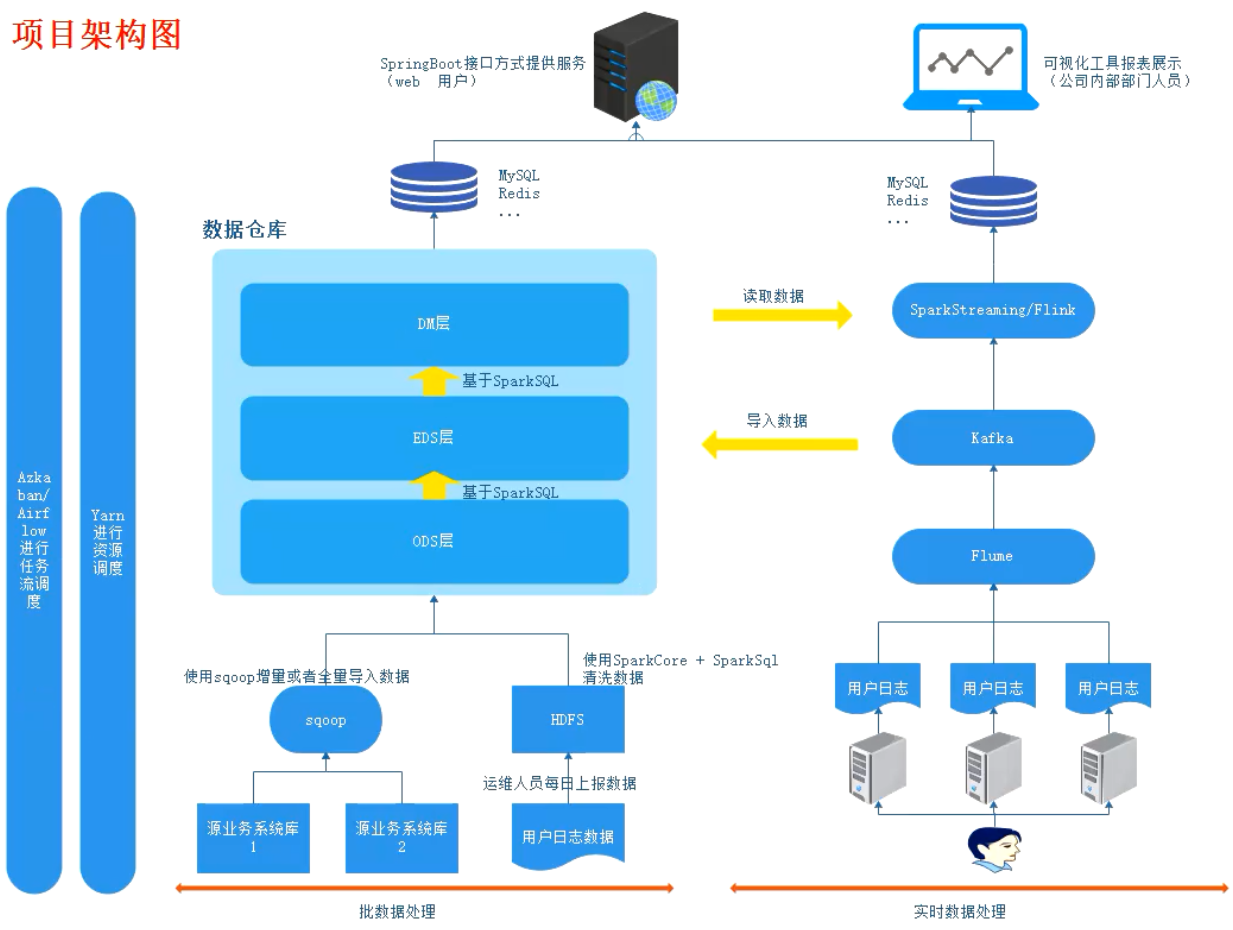
数据中心项目是Spark综合的数仓项目。分为离线处理和实时处理，其中用到的解释有MySQL、Sqoop、HDFS、Yarn、Hive、数据仓库模型设计、SparkCore、SparkSQL、SparkStreaming、Azkaban、Flume、Kafka、Redis、superSe、Redis、微信指数、高德API等。

离线处理：以Spark为主，其中很少使用了SparkCore的代码，主要是用SparkSQL构建数据仓库。项目使用airflow/Azkkaban进行调度，可以每日进行调度，也可以每月进行调度，每天定时触发调度。

实时处理：使用SparkStreaming实现实时处理，离线N+1的方式不能得到实时数据，运营活动中心有时需要实时的用户上线数据，针对客户端进行数据埋点，用户在客户端所有的行为都是事件，对事件进行埋点，当用户触发了一些事件时，判断用户是否满足了目标用户情况。

实时处理流程如下：

http请求 -> 数据采集接口 -> 数据目录 -> flume监控目录【监控的目录下的文件是按照日期分的】 -> Kafka【也会放在HDFS中，就是上面做的】-> SparkStreaming分析数据 ->Kafka【给运营中心使用】



## 集群配置&项目人数、周期

集群配置

生产环境：集群有50台服务器，16核32线程+128G+40T

20核40线程机器。32核64线程。56核112线程。

测试环境：5台测试机器，2核双线程+32G+1T磁盘

项目人数：

1个web人员+大数据开发4人（2个负责离线，1个负责实时，一个负责规划）+ 运维部门人员2人。

项目周期：

业务一直改变，目前2年

## 数据来源及采集

数据中心项目中的数据来源主要是商场中的KTV歌厅产生的数据和商场中跳舞机产生的数据。

以上两种设备产生的数据可以分为两类数据，一类是产生的订单数据，会记录到业务数据库，后期直接通过sqoop直接抽取MySQL中的数据到HDFS。另一类是通过http请求，上传到专门采集数据的日志服务器上，每天由运维人员将数据打包上传到数据中心平台某个目录下，然后由定时任务定时来执行Spark任务拉取数据，上传至HDFS中。这里读取压缩数据使用SparkCore进行处理，处理之后将数据以parquet格式或者json格式存储在HDFS中即可

## 数据仓库模型

数据仓库按照主题分为三个主题：用户、机器、内容（歌曲相关、歌手相关）。每个主题下面都有对应的表。数据仓库的设计分为三层，如下：

* ODS层

ODS文件中是从业务数据库中抽取出来数据表 的原数据，数据从关系型数据库MySQL中导入，转换成Parquet格式的文件存在HDFS中，后期方便使用SparkSQL处理

ODS层数据来源如下:

外部数据源：网易云爬取歌曲热度数据、歌手热度数据，爬取数据是json格式的数据。

内部数据源：主要有MySQL和客户端上传json数据。MySQL使用Sqoop抽取数据到 HDFS中，导入ODS层。客户端产生日志到客户端服务器，客户端服务器由运维人员每 天将数据压缩成包导入到HDFS路径中，也就是ODS层。

* EDS层

EDS层负责信息集成、轻度汇总类数据。简单理解就是将事务性的数据组织成便于分析

的仓库维度建模类型的数据，做一些轻度聚合，类似Hive 中的宽表。例如︰将ODS层 数据进行清洗，如果主题是用户主题，那么就按照用户id为粒度将数据组织在一起

如果主题是机器，那么就按照机器id为粒度将数据组织在一起。以上ODS层和EDS层 使用spark代码处理数据，然后利用SparkSQL读取ODS层数据，保存到Hive 的EDS层。

* DM层

DM层的数据有一部分是存储在Hive表中，或者保存分析结果到MySQL、HBase等。EDS 层数据是parquet格式的数据，放在Hive 的主要原因是后期使用Kyli查询一些业务， 数据放MySQL的都是结果数据，放在HBase 的原因是设涉及到大表的明细查询。以上 数据仓库模型的设计表对应关系都在“数据仓库模型.xlsx”文件中。

## 数据仓库命名规范

参照“数据仓库模型.xlsx”文件。

## Sqoop安装

由于项目中使用到了Sqoop将MySQL中的数据导入到Hive 中，所以这里，首先安装

Sqoop。

Sqoop是一款开源的工具，主要用于在Hadoop(Hive)与传统的数据库(MySQL、 Postgres...）间进行数据的传递，可以将一个关系型数据库（例如MySQL ,Oracle ,Postgres 等 ）中的数据导进到HDFS 中，也可以将HDES的数据导进到关系型数据库中。Sqoop 的 导入导出数据原理非常简单，就是将导入或导出命令翻译成mapreduce程序来实现 Sqoop安装︰

1．下载Sqoop

登录Sqoop官网:http: //sqoop.apache.org/ ，下载Sqoop,这里选择版本1.4.7 版本进行下载。

2.上传解压

将下载好的Sqoop，上传到集群中的某个节点上，这里选择mynode3,进行解压。 节点必须有Hadoop环境，并且配置好hadoop环境变量。

3．在mynode3节点上配置Sqoop的环境变量

在/etc/profile追加Sqoop的环境变量，保存后，执行source /etc/profile

使环境变量生效。