第3章　数据组织和架构

3.1　数据组织概述

自20世纪70年代起，传统数据库系统在事务处理上取得了显著成就，但在分析处理上却面临挑战。由于部门间数据抽取的差异、时间不一致、外部信息和分析程序的多样性，分析结果的可靠性受到影响；数据分散于多个数据库，导致分析处理效率低下；历史数据存储时间各异，数据集成性不足，加大了数据转化为信息的难度。因此，直接在事务处理环境中构建分析型处理应用并不理想。为解决这些问题，数据仓库(data warehouse)在20世纪90年代应运而生，它从事务处理环境中提取并重新组织分析型数据，构建了一个独立的分析处理环境。数据仓库有效地整合了跨业务、跨系统的数据，为管理分析和业务决策提供了统一的数据支持。与数据库系统不同，数据仓库专注于历史性、综合性和深层次的数据分析，提供多维查询分析，使用户能够更灵活、直观、简洁地进行数据操作，从而提高数据分析和决策的效率及有效性。

随着互联网尤其是移动互联网的迅速普及，数据规模急剧膨胀，远超传统数据仓库的扩展能力。同时，半结构化、非结构化数据的规模和类型迅速增加，使得数据仓库在存储容量和存储类型上捉襟见肘，难以满足企业的存储和分析需求。为应对这一挑战，数据湖(data lake)作为一种可扩展的新型数据存储库被提出，它以多样化原始数据为导向，实现了任意来源、速度、规模、类型数据的全量获取、全量存储、多模式处理与全生命周期管理，为数据挖掘、预测建模和机器学习等高级分析应用提供了强大的数据支持。然而，随着数据湖的广泛应用，数据割裂、资源重复浪费等问题逐渐浮现。为解决这些问题，2016年阿里巴巴率先提出了数据中台(data middle platform)的概念，其核心在于避免数据重复计算，通过数据服务化提升数据共享能力、赋能数据应用。

与此同时，向量数据库(vector database)作为一种先进的数据管理系统，专门设计用于处理和存储高维向量数据，这些数据通常来源于机器学习、深度学习或其他复杂的数据表示模型。与传统的关系型数据库和文档数据库不同，向量数据库的优势在于其高效的向量相似性搜索能力，这得益于其构建的KD树、球树或局部敏感哈希等复杂索引结构。随着人工智能和大数据技术的飞速发展，特别是在海量数据集中寻找与查询向量最接近的匹配项时，向量数据库的重要性日益凸显。向量可被视为多维空间中的点，每个维度代表数据的不同特征，这使得向量数据库在自然语言处理、图像和视频检索、目标检测、推荐系统、异常检测等多种应用场景中发挥着关键作用。随着技术的不断进步，向量数据库正逐渐展现出其在

实时响应查询、实时决策和个性化服务等方面的卓越性能。尽管向量数据库是一个相对较新的领域，其迅速发展已吸引了大量研究和商业投资，在未来的数据管理和人工智能应用中将占据核心地位。

作为信息系统的核心需求和数据管理的重要内容，几乎所有信息系统都需要有效地组织、存储、操纵和管理业务数据。本章围绕数据组织和架构的关键问题，讨论数据仓库、数据湖与数据中台，以及向量数据库，介绍数据仓库的概念、特征及体系结构，数据湖和数据中台的基本原理、体系结构及代表性的数据湖平台 Hudi，最后介绍向量数据库的基本概念、核心技术原理及代表性产品。

3.2 数据仓库

3.2.1 数据仓库的基本特征

一个普遍认可的数据仓库的定义为：数据仓库是一个面向主题的、集成的、时变的、不可更新的数据集合，用以支持管理层的决策制定过程和商业智能。该定义指出了数据仓库的 4 个重要特征。

1. 面向主题

数据仓库中的数据是围绕企业的关键主题而组织的。主题是一个抽象的概念，是在较高层次上将企业信息系统中的数据综合、归类后进行分析利用的抽象，因此，主题也称为高层实体。例如，客户、患者、学生、产品、时间等，都是经常使用的主题。在逻辑意义上，主题是对企业中某一宏观分析领域所涉及的分析对象，针对某一决策问题而设置。面向主题的数据组织方式，就是在较高层次上对分析对象的一个完整和一致的描述，能统一刻画各个分析对象所涉及的各项数据及数据之间的相互联系。相对面向应用的数据组织方式而言，按照主题进行数据组织的方式具有更高的数据抽象级别。

需要说明的是，数据在数据仓库中可如同在数据库中一样，以表的形式进行存储，但数据的组织和建模方式与数据库系统相比有了较大的改变。

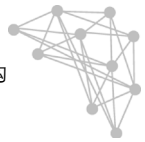
2. 集成

数据仓库中存储的数据使用一致的命名约定、格式、编码规则和相关特性来定义，这些相关特性可从内部记录系统或组织机构外部获得。

决策支持系统本身就需要全面、正确、集成的数据，相关数据收集越完整，得到的结果就越可靠。数据仓库将关系数据库、一般文件和联机事务处理记录等多个异构数据源集成在一起，在集成过程中使用数据清理和数据融合技术，确保命名约定、编码结构、属性度量等的一致性。一方面，原有数据库系统以事务处理方式记录每一项业务的数据，在进入数据仓库之前需经过综合处理，抛弃一些分析处理不需要的数据项，必要时还要增加一些相关的外部数据。另一方面，数据仓库的每一个主题所对应的源数据，在分散的数据源中可能存在重复或不一致的部分，需利用数据清理方法将这些数据转换为全局统一的定义，消除冗余、错误和不一致数据，以保证数据质量。

3. 时变

数据仓库中的数据包含一个时间维度，可用来研究趋势和变化。



数据仓库以分析为目的，需把不同时间段的业务情况记录下来，如同一组镜头中的一幅幅画面，将其连续播放就会产生动画效果。对于瞬间图像而言，时间并不是一个变量，而是静止不变的，但对于动画而言时间，则是不断变化的，动画可根据观看者的要求按任意方向以任意速度播放。数据仓库类似动画，业务决策者按照不同的时间观察数据的曲线及趋势，便于制定决策。

4. 不可更新

也称为“非易失”或“稳定”，即数据仓库中的数据从操作型系统加载和刷新，但不可由最终用户来更新。

如前所述，数据仓库的数据与事务处理的数据物理分离，从而不影响事务处理系统的使用。由于这种分离，数据仓库不需要事务处理、恢复和并发控制机制，通常只需要数据初始化装入和数据访问这两种数据处理。由于数据仓库中的数据是只读的，所以不存在多个用户同时对同一记录进行读写操作时被锁定的问题。也就是说，数据仓库所创建的是一个只读数据库系统，系统只为数据库不断增加新的记录，而系统中已经存在的记录不会被改动。

3.2.2　从操作型系统到信息型系统

作为一种信息型系统(informational system)，数据仓库中的数据来源于各分散的操作型系统(operational system)；独立的数据仓库，可避免操作型处理与信息型应用混淆时而导致对资源的争用。下面首先介绍这两类系统的基本概念，并从多个角度对其进行比较，也使读者进一步理解数据仓库的概念。

操作型系统是用来根据当前数据而实时运行业务的系统，也称为记录系统。例如，学生选课、销售订单处理、商品预订、患者挂号等都是典型的操作型系统，它们必须处理相对简单的读/写事务，并需要快速响应。信息型系统是为了支持依据历史时间点的瞬态数据和预测数据来执行决策而设计的系统，也是为复杂查询或数据挖掘应用而设计的系统。例如，销售趋势分析、客户与市场细分、人力资源规划等系统，都是典型的信息型系统，它们通过只读操作对数据进行分析处理。表 3.1 从不同特征的角度比较了操作型系统和信息型系统。

表 3.1　操作型系统和信息型系统的比较

特　　征	操作型系统	信息型系统
主要目的	执行当前业务	支持管理层决策制定
数据类型	当前状态的当前表示	历史时间点和预测数据
主要用户	职员、销售人员和管理员	管理人员、业务分析人员和客户
使用范围	较窄，有计划、简单的查询和更新	较宽，复杂查询和分析
设计指标	性能、吞吐量和可用性	易于灵活地访问和使用
数据量	持续不断地查询或更新一条或多条记录	定期分批地查询或更新很多记录或所有记录

3.2.3　数据仓库体系结构

根据应用需求的不同和所采用技术本身的特点，数据仓库的体系结构可分为以下 4 种

类型：一般两层结构、独立数据集市、依赖数据集市和操作型数据存储、逻辑数据集市和实时数据仓库。

1. 一般两层体系结构

数据仓库的一般两层体系结构如图 3.1 所示，包括源数据系统、数据和元数据存储区两个层次。

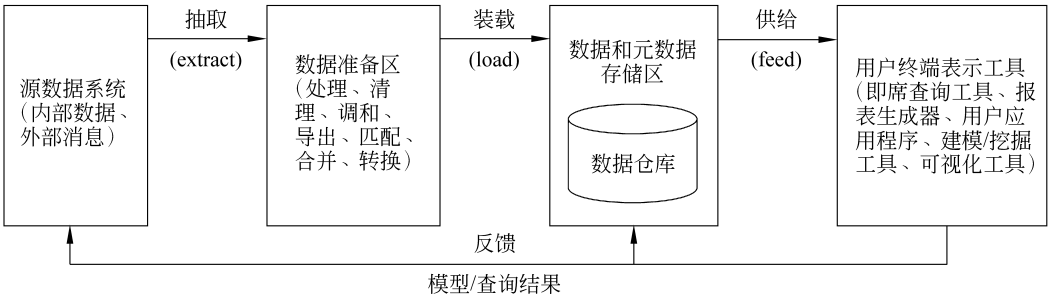


图 3.1　数据仓库的一般两层体系结构

构建这一体系结构的 4 个基本步骤如下。

（1）从各种内部及外部的源系统文件和数据库中抽取数据，在大型组织机构中，通常可能有几十个甚至数百个这样的文件和数据库。

（2）把来自各类源系统的数据加载到数据仓库之前进行转换和集成，事务可能被发送到源系统中，以纠正数据准备中发现的错误。

（3）将数据准备预处理后的数据装载到数据仓库中，包括详细数据和汇总数据。

（4）用户通过多种查询语言和分析工具来访问数据仓库，预测和预报等的结果会反馈到数据仓库和源系统中。

在以上步骤中，从源数据系统进行数据的抽取（extract）、转换（transform）和装载（load），简记为 ETL，是数据仓库构建的基本流程，由于其重要性而被称为数据仓库的“血液”。经过多年的发展，目前已有许多有效的 ETL 工具，这些工具的功能越来越全面，多种数据源的处理能力和自动化处理功能也日益增强。具体而言，数据仓库的 ETL 工具一般包括以下功能部件。

1）数据抽取

从源系统中提取出与数据仓库主题相关的必需数据。例如，若某超市确定以分析客户的购买行为为主题建立数据仓库，则只需将与客户购买行为相关的数据抽取出来，而对于其他数据，没有必要抽取并存储到数据仓库中。数据抽取一般包括静态（static）抽取和增量（incremental）抽取这两种类型，静态抽取是在某一时间点捕获所需要源数据快照的一种方法，用来在初始时填充数据仓库；增量抽取是一种只捕获自上次捕获之后源数据所发生变化的方法，用于数据仓库的维护。

2）数据清理

保证足够的数据质量是 ETL 面临的最大挑战，直接影响后续的分析处理和所得到信息的有效性。从数据仓库的角度，对不同源系统中冗余、不一致或不同步的数据，在进入数据仓库之前就予以更正或删除，以保证数据仓库中数据的质量，是数据清理的主要目的。

3）数据转换

将不同源系统中同类信息的不同数据类型及不同数据格式转换为统一的表示，数据转换功能可分为记录级功能和字段级功能。最重要的记录级功能包括选择（根据条件分割数据集）、连接（合并各种数据源）、规范化（消除数据异常）和聚合（获得概要级别的数据）；字段级功能将数据从源记录中的给定格式转换为目标记录中的不同格式，包括从单一源字段转换为单一目标字段，以及从一个或多个源字段转换到一个或多个目标字段。例如，对零售业务记录按照商店、商品及日期等进行汇总，以得到总销售额；将时间格式“年/月/日”和“月/日/年”统一转换为“日—年—月”。

4）数据装载

将预处理后的数据按照物理数据模型定义的结构装入数据仓库，包括清空数据域、填充空格和有效性检查等操作。刷新（refresh）和更新（update），是将数据装载到数据仓库的两种基本模式。刷新模式采用定期批量重写目标数据来填充数据仓库，而更新模式只将源数据中的变化写入数据仓库。最初创建数据仓库时，一般采用刷新模式来填充数据仓库，而在随后的维护中，一般采用更新模式；刷新模式与静态抽取一并使用，而更新模式与增量抽取一并使用。此外，为了高效地管理和检索数据仓库中的数据，需要为其创建索引，数据仓库常用的索引是位图索引（bitmap index）和连接索引（join index）。索引的概念，这里不做赘述。

2. 独立数据集市的数据仓库体系结构

在图 3.1 给出的一般两层体系结构中，数据仓库中存储了按照不同主题而组织的整个企业的信息。然而，针对实际中一些特定的分析任务，若只涉及某一个或几个主题，则没有必要检索数据仓库中所有主题的数据，而只需对相关主题的部分数据进行分析，这样也可提高分析处理的效率。例如，市场发展规律的分析主题主要由市场部门的人员使用，可在逻辑上或物理上将这部分数据分离出来，当市场部门人员需要信息时，只需在与市场部相关的部门数据上进行分析。因此，为了特定用户群的决策制定，关注面向企业中某个部门（主题）的数据子集，可从数据仓库中划分出来，或从源数据系统由独立的 ETL 过程获得，这样的数据子集称为数据集市（data mart，DM）。

由于组织机构可能只关注一系列短期有利的业务目标，因此通常需要创建独立数据集市，图 3.2 给出了包含独立数据集市的数据仓库体系结构。在这一体系结构中，数据集市中的数据相分离，每个数据集市对应了独立的 ETL 过程，使得用户访问独立数据集市中的数据时较复杂，且可能导致各数据集市中数据的不一致性。鉴于该体系结构的实际需求及技术上的缺点，人们对独立数据集市的价值展开了激烈争论，包括数据集市是否应该从企业范围内决策支持数据的一个子集以自底向上的方式发展，针对分析处理的数据集市数据库应该规范化到什么程度，等等。

3. 依赖数据集市和操作型数据存储体系结构——三层结构

从技术和可伸缩性的角度，针对独立数据集市体系结构的局限性，人们提出数据仓库三层体系结构，其出发点主要包括：降低冗余数据及其处理的较高代价，提供一个清晰的企业级数据视图，具备向下钻取细节信息的分析能力，降低扩大规模和保持分离数据一致性的成本。

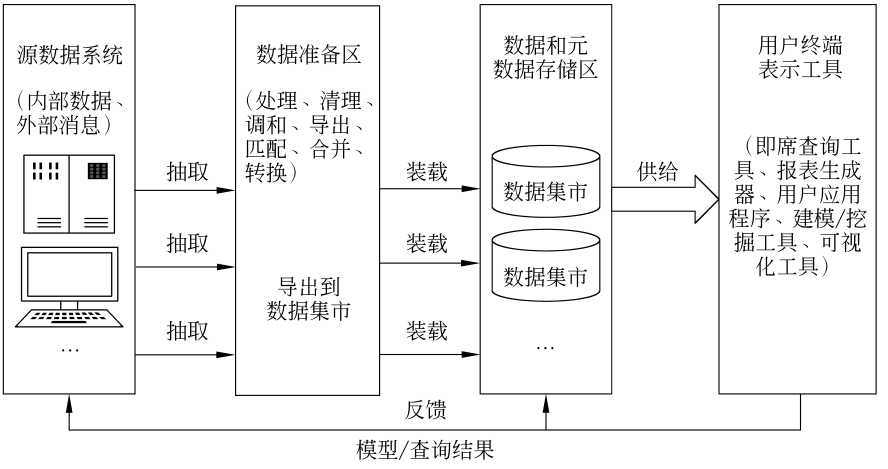


图 3.2　独立数据集市的数据仓库体系结构

采用“源数据系统—操作型数据存储—企业数据仓库和依赖数据集市”表示的三层结构，是较早出现用来解决独立数据集市局限性最普遍的一种方法，如图 3.3 所示。与独立数据集市体系结构相比，新增加的一层是“操作型数据存储”，而“数据和元数据存储层”则重新进行了配置。这种体系结构也被称为“中心辐射型”（hub and spoke）结构，其中，企业数据仓库（enterprise data warehouse，EDW）是中心、源数据系统和数据集市输入/输出辐射的两端。

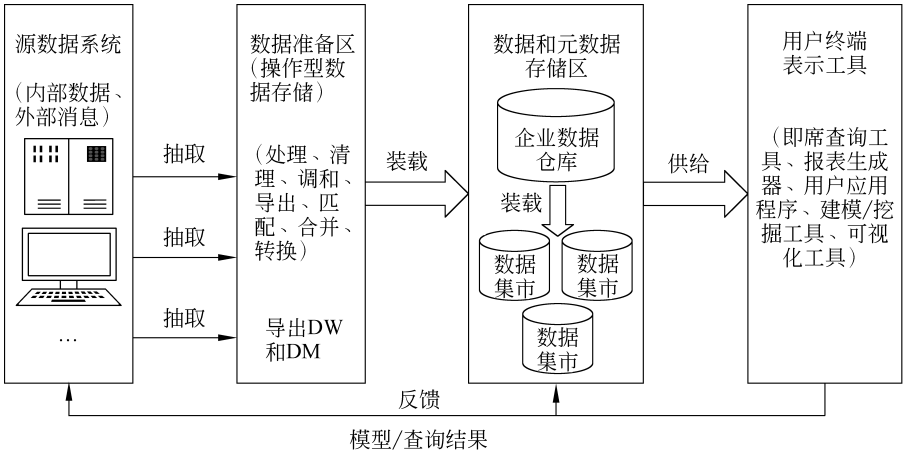


图 3.3　依赖数据集市和操作型数据存储的三层体系结构

EDW 是一个集中式的集成数据仓库，是决策支持应用程序最终用户可获得的所有数据的控制点和唯一“真实版本”；依赖数据集市（dependent data mart）是从 EDW 中以互斥方式而填充的数据集市。通过从 EDW 把数据根据部门或主题装载到依赖数据集市，解决了冗余数据和企业数据视图的局限性。一是 EWD 提供了一个简化的、高性能的环境，与特定部门用户群的决策制定需求相一致；二是用户群可访问其数据集市，同时需要其他数据时，用户仍可访问 EDW；三是由于每个数据集市都以同步方式从 EDW 这一相同数据源装载而

来，跨依赖数据集市的冗余在计划之内，且冗余的数据是一致的。

操作型数据存储(operational data store，ODS)是一个集成、面向主题、可更新、具有当前值、企业范围的详细数据库，其设计目的是为操作型用户执行决策支持处理提供服务，为操作型数据提供综合来源，克服了向下钻取细节信息方面的局限性。一般情况下，ODS 是一个关系数据库，其规范化针对决策制定应用程序，通过类似关系数据库规范化的方式进行。ODS 包含易变的当前数据，而不包含历史数据，因此，对 ODS 的相同查询在不同时间可能产生不同的结果。ODS 可立即从源数据系统接收数据，也可在一定延迟后接收数据，无论何种情况，ODS 都起着数据准备区的作用，通过它将数据装载到 EDW。

图 3.3 给出的依赖数据集市和操作型数据存储体系结构，也称为企业信息工厂(corporate information factory)，被视为组织机构数据的综合视图，支持用户对数据的所有需求。

4. 逻辑数据集市和实时数据仓库体系结构

针对中等规模的数据仓库或使用高性能数据仓库技术的情形，可使用逻辑数据集市和实时数据仓库体系结构，如图 3.4 所示。其中，操作型数据存储和企业数据仓库并不分开，比常用的三层体系结构少了一层。事实上，如果 EDW 和 ODS 为同一个存储实体，用户自底向上或自顶向下的钻取都会更容易实现。

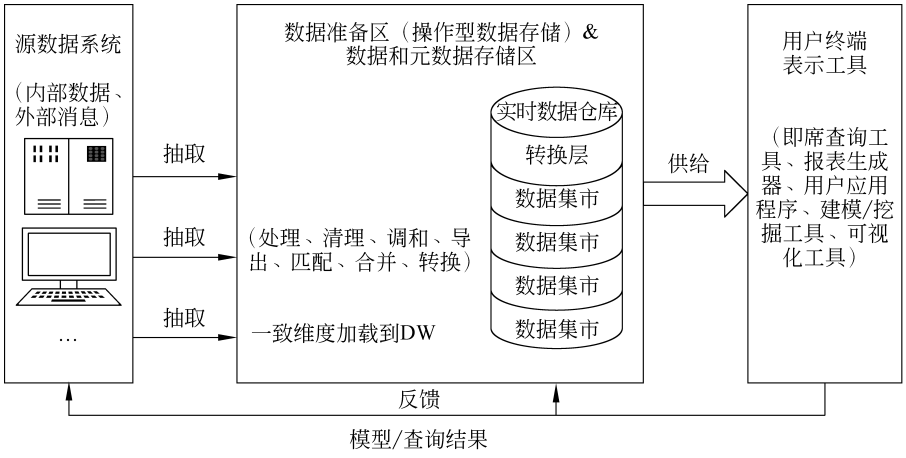


图 3.4　逻辑数据集市和实时数据仓库体系结构

从逻辑数据集市的角度，该体系结构具有如下特征。

(1) 逻辑数据集市(logical data mart)并不是物理上分离的数据库，而是具有一定反规范化特征的关系型数据仓库的不同关系视图。

(2) 数据从源数据系统抽取后被转移到数据仓库，而不是转移到与数据仓库相分离的数据准备区，从而可利用数据仓库技术的高性能计算能力来执行清理和转换任务。

(3) 由于数据集市是数据仓库的视图，因此，一方面，数据集市可被快速地创建，并不需为其创建物理数据库，也不需专门编写装载例程；另一方面，视图被使用时其中的数据才被创建，数据集市总是最新的，避免了由于数据重复存储而造成的不一致。此外，若用户需频繁执行相同的查询来处理数据集市的相同实例，可使用物化视图机制来进行查询优化。

从实时数据仓库的角度，该体系结构具有如下特征。

(1) 实时数据仓库(real-time data warehouse)，是一种近乎实时地接受源数据系统供给事务数据的企业数据仓库。它分析数据仓库中的数据，并近乎实时地将新的业务规则传递到数据仓库和源数据系统，以便立即采取行动来响应业务事件。

(2) 针对需要对组织机构当前的、综合的状态做出快速响应的情形，该体系结构保证了源数据系统、决策支持服务和数据仓库可接近实时的速度来交换数据和业务规则。也就是说，实时数据仓库通过所有信息的共同作用减少了从事件发生到采取行动之间的延迟，缩短了业务事件发生时捕获客户数据、分析客户行为、预测客户对可能采取行动的反应、制定优化客户交互的规则等的处理时间。

邮政快递和包裹投递服务是实时数据仓库的一个典型范例，通过频繁地扫描包裹，可精确地知道包裹在传输系统中所处的位置；根据包裹数据、定价、客户服务等级协议及物流时机等，通过实时分析，能自动为包裹重新选择路线，以较好地实现对客户所做的投递承诺。事实上，除了上述范例，实时数据仓库有许多有益的实际应用，例如：

(1) 及时运送。根据最新库存水平重新选择投递路线。

(2) 电子商务。在用户下线之前，通过弃置的购物车信息给用户发送电子邮件，告诉用户一些奖励或优惠政策。

(3) 营销监控。销售人员实时监控重要账户的关键绩效指标。

(4) 欺诈检测。对于信用卡的不平常事务模式，使系统向销售人员或在线购物车例程发出警报，以便采取相应的预防措施。

数据仓库的前述 3 种典型体系结构，都以数据集市中数据的装载、数据仓库与数据集市之间数据的流动为核心。作为本节内容的总结，表 3.2 给出数据仓库和数据集市的区别。

表 3.2　数据仓库和数据集市的区别

	数 据 仓 库	数 据 集 市
范围	独立于应用	特定决策支持应用
	企业级，集中式	部门级，区域分散
数据	大量的历史数据	适度的历史数据
	最细的粒度	较粗的粒度
	稍微反规范化	高度反规范化
	大规模	开始小规模，逐渐变大
主题	企业多个主题	部门或特殊的分析主题
来源	多个内部或外部来源	少数几个内部或外部来源
其他	面向数据	面向项目
	生命周期长	生命周期短
	复杂的单一结构	相对简单的多种结构，合在一起较复杂
	海量数据分析处理和探索	便于访问分析，快速查询

需要说明的是，数据量并不是区分数据仓库和数据集市的唯一尺度；单一数据集市的复杂程度低于数据仓库，但是建立数据集市的过程很耗时，几乎与建立一个数据仓库一样，需要相同的计划和管理，且需要把数据模型化，因此，建立数据集市并不比建立数据仓库容易；由于数据集市针对特殊的业务需求而采取面向特定应用的数据模型，数据集市的伸缩、追加数据、扩展数据宽度都非常困难，因此，数据集市并不能较容易地升级到数据仓库。

3.3　数　据　湖

3.3.1　数据湖概述

从 20 世纪 90 年代起，数据仓库就开始大量用于企业决策，经过多年的发展，已成为企业信息集成和辅助决策应用的关键技术。然而，随着 Web 2.0 的兴起，非结构化数据迅速增加，数据仓库作为一个面向主题、以关系型数据库为基础、用以支持管理决策的数据集合，逐渐难以满足企业的数据存储和处理需求，而以数据湖的方式构建的数据存储和处理系统可面向海量、多源、异构的数据进行决策分析。例如，可通过对论坛和微博等非结构化数据进行综合分析，梳理或检验其中蕴含的事实和趋势，揭示隐藏的内容和信息传播的规律，对事件发展做出预测，以协助相关部门有效应对各种突发事件。

数据湖的概念最早由 Pentaho 首席技术官 James Dixon 于 2010 年提出，他将数据集比喻为大自然的水，各个江川河流的水未经加工、源源不断地汇聚到数据湖中，为企业带来各种分析和探索的可能性。随着数据湖研究和应用的不断深入，一个普遍认可的数据湖定义是：数据湖是一种灵活的、可扩展的数据存储库和管理系统，以原始数据格式接收和存储数据，并提供按需处理和数据分析的能力。该定义指出了数据湖的两个重要特征。

数据湖存储原始数据：数据湖可存储未经任何处理的数据，且支持存储任意类型的数据，包括结构化数据（如关系型数据库表）、半结构化数据（如 JSON 和 XML 文档等）和非结构化数据（如电子邮件和视频等）。

数据湖支持按需处理和数据分析：当把数据写入数据湖时，不需提前定义数据的存储结构和模式，只有应用程序需要数据时才会根据需求定义数据的存储结构和模式，因此数据处理更加灵活，可满足按需处理和数据分析的要求。

由上述定义可知，数据湖与数据仓库形成鲜明对比，数据仓库通常以关系型数据库为基础，在数据类型和数据量上都有很大限制，而数据湖是一种灵活且可扩展的数据存储库，可将任意大小和类型的数据存储到数据湖中；数据仓库遵循传统的 ETL 流程，首先从源数据系统中抽取数据，然后对数据进行转换，并将转换后的数据装载到数据仓库中，数据在装载到数据仓库之前需定义好数据的存储结构和模式，即“写时模式”（schema-on-write）。而数据湖以原始格式存储数据，只有当准备使用数据时，才对数据的存储结构和模式进行定义，即“读时模式”（schema-on-read）。因此，数据湖与传统数据仓库的 ETL 思想不同，数据湖首先从源数据系统中抽取数据，并将数据装载到数据湖中，直到应用程序需要数据时才会根据需求执行转换操作。总的来说，数据湖的主要特点是可存储任意类型的数据，利用“读时模式”的方法，以 ELT 流程处理数据，避免了复杂、昂贵的数据建模和数据集成工作，可满足用户需求灵活多变的高效率分析需求。这也是数据湖近年来被广泛关注的原因，它从设计理

念和具体实现等方面都与数据仓库有本质区别，如表 3.3 所示。

表 3.3　数据仓库与数据湖的区别

	数据仓库	数据湖
数据类型	结构化数据	结构化、半结构化、非结构化数据
处理模式	写时模式	读时模式
性能	较高成本的存储获得最快的查询结果	低成本存储获得较快的查询结果
数据访问	通过 SQL 语句访问数据	通过开发人员创建的程序访问数据
可靠性	可靠性高	可靠性较低
主要用户	业务分析师	数据工程师、数据科学家

3.3.2　数据湖与数据中台

中台的概念源于芬兰的游戏公司 Supercell，是以数据共享及功能开发为主的、面向前台的架构，旨在减少开发成本、提高业务创新能力。中台在实践中衍生出了很多新的概念，如业务中台和数据中台等。业务中台是中心化的能力复用平台，通过整合后台业务资源，将企业核心能力以共享服务的形式沉淀，形成业务中心，实现后台业务资源到前台易用能力的转换，避免系统重复功能建设和维护带来的资源浪费；数据中台以数据建设和数据治理等数据管理活动为特征，将企业各个业务应用场景中形成的数据资源经过整合加工，以共享的方式将数据复用到不同业务，从而提高企业数据的利用率和整合程度，在同时开展数量较多的业务时，可明显提升管理效率，加快组织间的协作能力，快速响应前台业务需求，使得数据更加充分有效地利用，解决企业面临不同业务数据互通受限、数据管理困难、数据应用开发冗长等问题。

随着数字化进程的不断推进，数字化转型进入了一个新的阶段，数据湖和数据中台作为一种新型数据存储架构和数据处理机制，成为了支撑企业数字化转型的数据底座和战略核心，能提供数据驱动和精准决策等全方位技术支持。企业针对不同业务开发的信息系统，难以做到数据的互联互通，导致企业内部形成“数据孤岛”，分散在各个“孤岛”的数据无法很好地支撑企业的经营决策，也无法很好地应对快速变化的前端业务。数据湖是一个可扩展的集中存储库，能存储任意规模和任意类型的数据，将企业各个业务应用场景中形成的数据资源进行整合，并支持按需处理和分析数据，助力企业高效低成本地完成数字化转型。数据中台能聚合企业内部和外部的数据，将用户的共同需求抽象出来并转换为数据资产，构建基于平台、组件化的系统能力，并以接口和组件等形式服务各个业务单元，使企业能快速、灵活地调用资源，以数据洞察来驱动决策和运营，为前台业务提供快速响应，精细化运营和服务支撑，从而促进数字化转型。

数据湖和数据中台都以大数据技术作为底层技术支撑，二者也存在一些相似的功能，例如，数据湖和数据中台都能将底层不同类型的数据统一综合到一个平台上处理，解决“数据孤岛”等问题。因此，人们通常会将数据湖与数据中台进行比较，从定义来看，数据中台与数据仓库、数据湖等数据存储和组织技术不同，数据中台是一套可持续“让企业的数据用起来”

的机制，是一种战略选择和组织方式，是依据企业特有的业务模式和组织架构，通过有形的产品和实施方法论支撑，构建的一套持续不断把数据变成资产并服务于业务的机制，由人工智能技术、大数据平台、数据治理产品与数据管理系统等组成，可持续不断地将数据进行资产化、价值化，并应用到业务；从实际用途来看，数据仓库和数据湖是为了支持管理决策和数据分析，而数据中台则是将数据进行服务化之后提供给业务系统，目标是将数据服务能力渗透到企业各个业务环节，不仅限于决策分析场景；从技术层面来看，数据仓库和数据湖都是集中存储库，而数据中台是加速企业从数据到业务价值的中间层，是位于数据底座与上层数据应用之间的一整套体系，包含数据体系建设，可建立在数据仓库和数据湖之上。

3.3.3　数据湖体系结构

数据湖的体系结构描述了数据湖中数据在概念上的组织方式，然而目前尚无完全成熟且得到广泛认可和应用的统一结构。一般而言，根据应用需求的不同和所采用技术本身的特点，数据湖的体系结构可分为数据池(data pond)和数据区域(data zone)两种体系结构。

1. 数据池

数据池体系结构将原始数据分类存储到不同的数据池，并在各数据池中进行优化整合，转换成容易分析的统一存储格式，可根据需求从不同数据池中获取数据。数据池的体系结构如图 3.5 所示，包括初始数据池(row data pond)、模拟数据池(analog data pond)、应用程序数据池(application data pond)、文本数据池(textual data pond)和归档数据池(archival data pond)。

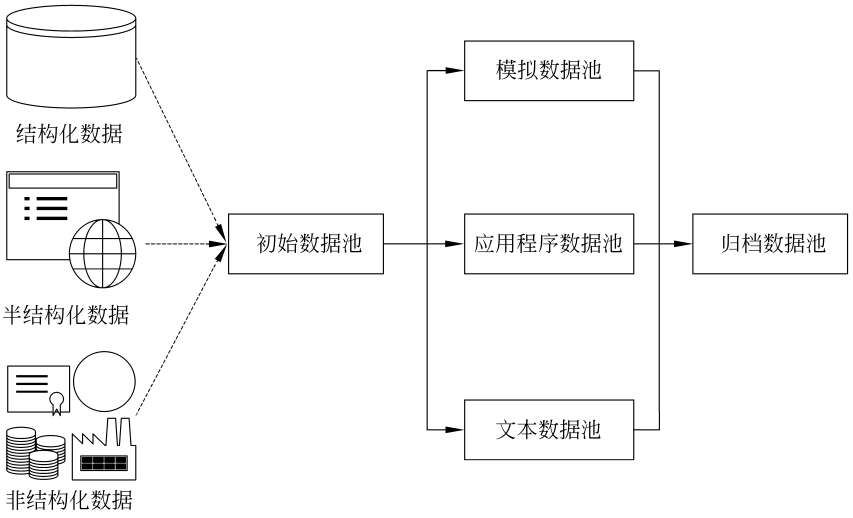


图 3.5　数据池体系结构

1) 初始数据池

摄取的结构化、半结构化和非结构化原始数据，首先存储在初始数据池中，作为其他数据池的数据暂存区，随后传输到所支持的数据池内(如模拟数据池、应用程序数据池或文本数据池)，并将原始数据从初始数据池中清除。

2）模拟数据池

用来存储和处理半结构化数据，通常半结构化数据由于数据量大而难以进行处理，因此模拟数据池中的一个重要处理步骤为数据缩减，即将半结构化数据的数据量减少到可管理和可分析的数据量。数据缩减的常用方法有数据去重、数据采样和数据压缩。数据去重的关键是如何判断 2 条记录是否重复，相似重复记录的识别算法（也称匹配算法）主要有字段匹配法（field matching）、编辑距离法（edit distance）和 N-gram 算法等。下面简单介绍编辑距离法的基本思想。

编辑距离是一种常用的尺度，是将一个串转换为另一个串的过程中需插入、删除及替换字符个数的最小值。设 A 为不同字母构成的有限字母表，$x^T \in A^T$ 表示字母表 A 上长度为 T 的任意一个串，x_i^j 表示 x^T 中位置 i 而终于位置 j 的子串，将单位长度的子串 x_i^i 简记为 x_i，将 x^T 中长度为 t 的前缀子串简记为 x^t。编辑距离用三元组$[A,B,c]$描述，其中：

（1）A 和 B 为有限字母表。

（2）c 为代价函数，$c:E\rightarrow R_+$。其中，$\mathcal{R}_+$ 为非负实数集，$E=E_s\cup E_d\cup E_i$ 为编辑操作构成的字母表，$E_s=A\times B$ 为替换（substitution）的集合，$E_d=A\times\{\varepsilon\}$ 为删除（delection）的集合，$E_i=\{\varepsilon\}\times B$ 为插入（insertion）的集合，ε 为空字符。

（3）每个三元组包括一个距离函数 $d_c:a^*\times b^*\rightarrow\mathcal{R}_+$，将一对串映射到一个非负值。子串 $x^t\in a^*$ 和 $y^v\in b^*$ 之间的距离递归地定义为

$$d_c(x^t,y^v)=\min\begin{cases}c(x^t,y_v)+d_c(x^{t-1},y^{v-1})\\ c(x^t,\varepsilon)+d_c(x^{t-1},y^v)\\ c(\varepsilon,y^v)+d_c(x^t,y^{v-1})\end{cases}\tag{3-1}$$

3）应用程序数据池

本质上为一个数据仓库，存储来自应用程序产生的结构化数据。在应用数据池中，由于不同应用程序产生的数据格式可能不同，需对来自不同应用程序的数据进行整合和转换，从而实现有价值的数据分析。数据转换功能可分为记录级别的功能和字段级别的功能。最重要的记录级功能包括选择（根据条件分割数据集）、连接（合并各种数据源）、规范化（消除数据异常）和聚合（获得概要级别的数据）；字段级功能将数据从源记录中的给定格式转换为目标记录中不同的格式，包括从单一源字段转换为单一目标字段，以及从一个或多个源字段转换到一个或多个目标字段。例如，对零售业务记录按照商店、商品及日期等进行汇总，从而得到总销售额；将时间格式“年/月/日”和“月/日/年”统一转换为“日—年—月”。

4）文本数据池

用来存储和处理非结构化的文本数据，例如电子邮件或光学字符识别（optical character recognition）产生的数据。文本消歧是文本数据池的一个重要处理步骤，通过对非结构的文本数据进行标准化处理，产生可作为分析记录存储在数据库中的数据。文本消歧的常用方法是通过机器学习或深度学习等模型，对池中的文本数据进行情感分类（情感分类模型将在 6.3 节详细介绍）。

5）归档数据池

用来存储来自模拟数据池、应用程序数据和文本数据池中使用频率较小的数据。

2. 数据区域体系结构

数据区域体系结构存在着多种变体，不同变体的区域数量和功能都有所不同，尽管没有统一的数据区域体系结构，但其核心思想都是根据数据处理程度将数据分配到不同区域。一个经典的数据区域架构如图 3.6 所示，包括初始区域（row zone）、黄金区域（gold zone）、工作区域（work zone）和敏感区域（sensitive zone）。

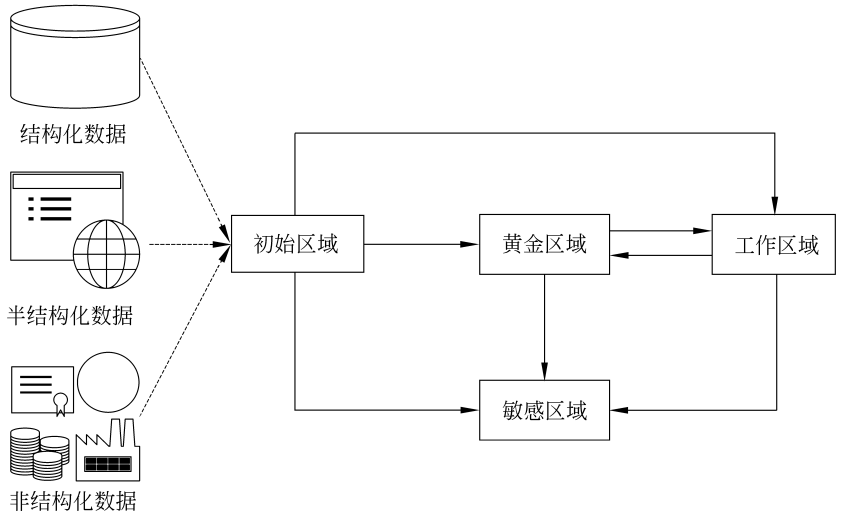


图 3.6　数据区域体系结构

1）初始区域

是数据区域体系结构的第一个存储区域，用于存储原始数据类型，即结构化数据、半结构化数据和非结构化数据。该区域主要面向技术开发人员、数据工程师和数据科学家。

2）黄金区域

存储数据工程师对初始区域中原始数据进行数据清理和数据转换处理后的数据，主要面向数据分析师和数据科学家。在黄金区域常用的数据清理方法为错误数据清理（错误数据是指数据源中的记录值与实际值不相符，即记录字段的值异常或不符合概念一致性、值域一致性和格式一致性等业务规则），主要包括基于孤立点检测的方法和基于业务规则的错误数据清理方法。下面简单介绍基于业务规则的错误数据清理方法的 4 个步骤。

步骤 1：分析具体业务，在规则库中定义业务规则。

步骤 2：通过数据查询技术获得待清理数据，并根据要检查的记录来检索规则库中的规则。

步骤 3：根据检索出的业务规则，根据字段域对每一条记录进行检测；根据同一记录中字段之间存在的函数依赖或业务规则等关系，对每条记录的多个字段进行检测。

步骤 4：通过步骤 3 可判断每条记录是否符合所定义的业务规则，如果不符合，则说明该记录含有错误数据，进一步从规则库中调用相关规则来改正该记录中的错误字段值；若无合适的规则，就由人工来处理。

3）工作区域

用于存储数据工程师从黄金区域中复制的数据，并根据业务需求进一步加工处理，为数据科学家提供更加精练的数据，因此大部分数据分析业务都发生在工作区域。工作区域中经处理和分析后的结果可写入黄金区域，以供进一步分析。该区域主要面向数据工程师和数据科学家。

4）敏感区域

用于存储初始区域、黄金区域和工作区域的重要数据文件，只有数据管理员和有权限的数据分析师才能访问敏感区中的数据，例如，财务人员可访问财务数据。一般而言，敏感区域的访问控制可通过对文件设置用户访问权限实现，例如，构建在 Hadoop 分布式文件系统（Hadoop distributed file system）之上的数据湖，可通过设置系统的文件访问控制列表实现。

在数据池体系结构中，当数据离开初始数据池时，原始数据会丢失，这与数据湖的概念相矛盾，而在数据区域体系结构中，无论何时都可从初始区域中访问原始数据，因此数据区域体系结构使用更加广泛。除了以上介绍的两种数据湖体系结构，近年来人们提出了湖仓一体（data lakehouse）这一新的开放式体系结构，通过将数据仓库的高性能及可靠性与数据湖的灵活性相结合，在数据湖的低成本存储之上提供数据仓库的事务管理和数据管理功能，并支持实时查询和分析，被认为是未来数据库发展的新趋势，感兴趣的读者可自行查阅相关文献学习。

3.3.4 Apache Hudi 简介

Hudi（Hadoop upserts deletes and incrementals）是一个开源的数据湖管理框架，最初于 2016 年由 Uber 开发，以实现在 HDFS 之上高效的数据增量更新，该项目于 2017 年开源，并于 2020 年被 Apache 软件基金会收录为顶级项目，正式进入 Hadoop 生态系统。Hudi 根据内部维护的时间轴（timeline）和文件管理机制，能有效地管理数据生命周期，并提高数据质量，通过 Spark 和 Flink 等分布式计算引擎，可对 Hudi 中不同类型的表进行更新、查询等操作。下面分别介绍 Hudi 的时间轴、文件管理、表类型和查询方式这 4 个核心概念，并结合 PySpark 给出 Hudi 程序示例。

1. 时间轴

Hudi 的核心是维护 Hudi 表在不同时刻执行的所有操作的时间轴，这有助于提供表的即时视图，同时还能根据数据的到达顺序高效地检索数据，例如仅查询某个时间点之后成功提交的数据，可有效避免扫描更大范围的数据。每一次对数据的操作都会在 Hudi 表的时间轴上生成一个 Hudi Instant 实例，每个 Instant 实例由即时操作（instant action）、即时时间（instant time）和即时状态（instant state）3 个部分组成。

1）即时操作

指在表上执行的操作类型，包括 COMMITS、CLEANS、DELTA_COMMIT、COMPACTION、ROLLBACK 和 SAVEPOINT。

（1）COMMITS：将一批数据原子性地写入表中。

（2）CLEANS：清除表中过期的数据，限制表占用的存储空间。

（3）DELTA_COMMIT：将一批数据原子性地写入增量日志文件中，仅适用于读时合

并类型的表(后续介绍 Hudi 中表类型的定义)。

(4) COMPACTION：协调 Hudi 中差异数据结构的后台活动，合并基于行的日志文件与基于列的数据文件、生成新的基于列的数据文件。

(5) ROLLBACK：在 COMMITS 或 DELTA_COMMIT 操作执行失败时进行回滚，并删除执行过程中产生的文件，保证 COMMITS 和 DELTA_COMMIT 操作的原子性。

(6) SAVEPOINT：标记需要备份的文件，在执行 CLEANS 操作时不会清除标记的文件；当需要数据恢复时，可将数据还原到标记时的状态。

2) 即时时间

按照即时操作开始执行时间顺序单调递增的时间戳。

3) 即时状态

在指定时间点对表执行操作后表所处的状态，包括已调度但未初始化的状态(REQUESTED)、当前正在执行的状态(INFLIGHT)和操作执行完成的状态(COMPLETED)。

2. 文件管理和索引

Hudi 中每个表都有相应的分布式文件系统目录，表被分为多个分区，用以确定表目录的各子目录中数据的分布。例如，表 T 的数据目录在/data/T 中，表 T 按日分为 2022-09-27 和 2022-09-28 两个分区，那么 2022 年 9 月 28 日产生的数据将存储在/data/T/2022-09-28 中。

在每个分区中，文件被组织为文件组(file group)，每个文件组包含多个文件切片(file slice)，每个文件切片包含一个数据文件和多个日志文件。每个文件组对应一个唯一的文件 ID，Hudi 写入新数据时，通过数据的记录键(record key)与分区路径组合映射到文件组的文件 ID 定位旧数据所在的文件组。在文件组内，Hudi 默认通过数据文件携带的布隆过滤器(bloom filter)来判断写入数据所属的数据文件，避免读取不需要的数据文件，从而实现高效的更新操作。

布隆过滤器由一个长度为 $n(n>0)$ 的数组 A 和一组哈希函数 $H=\{h_i(x)\mid i\in[1,k]\}$ 组成，数组中的元素初始化为 0。对于含有 m 个元素的集合 $S=\{s_j\mid j\in[1,m]\}$，布隆过滤器使用 H 中 k 个相互独立的哈希函数，将 S 中任意元素 s 的哈希值映射到数组 A 的 k 个单元中，也就是将 A 中 $h_i(s)\%n$ 位置相应的元素置为 $1(1\leqslant i\leqslant k)$。在判断某个元素 w 是否在集合 S 时，布隆过滤器取出 $\{h_i(w)\mid i\in[1,k]\}$ 位置上的 k 个值，若存在 0，则元素不在集合 S 中，若均为 1，则元素可能存在集合 S 中。下面通过一个简单的例子来介绍布隆过滤器在 Hudi 中的应用。

例 3.1　数据文件中存在数据记录 A、B 和 C，根据数据的记录键，使用长度为 18 的数组和 3 个哈希函数构建布隆过滤器。在写入新数据 D 时，加载存储于数据文件中的布隆过滤器，并根据简单的计算即可确定数据 D 是否存在于该数据文件中。如图 3.7 所示，可发现布隆过滤器中数组的第 15 位元素为 0，进而确定数据 D 不在该数据文件中。

3. 表类型

表类型定义了如何在分布式文件系统中写入或查询数据，Hudi 提供了写时复制(copy on write，COW)和读时合并(merge on read，MOR)这两种类型的表。

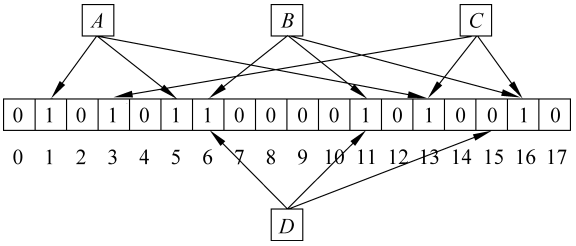


图 3.7　布隆过滤器示例

1）写时复制表

数据存储在基于列的数据文件中，数据更新时会重写数据所在的文件，生成新版本的数据文件，具有非常大的 I/O 资源消耗；读取数据时只需读取对应分区的一个数据文件，效率较高，因此写时复制表适用于读取密集型的场景。

2）读时合并表

数据存储在基于列的数据文件和基于行的日志文件中，数据更新时，首先以追加的方式写入日志文件，由于日志文件较小，写入成本也较低。而读取数据时需通过 COMPACTION 操作合并数据文件和日志文件，消耗大量 I/O 资源，因此读时合并表适用于写入密集型的场景。

4. 查询类型

Hudi 支持快照查询（snapshot query）、增量查询（incremental query）和读优化查询（read optimized query）这 3 种不同的查询方式，其中写时复制表仅支持快照查询和增量查询，读时合并表支持所有查询方式。需要说明的是，Hudi 本身不具备查询处理的功能，需借助分布式计算引擎，例如将 Hudi 表转为 Spark 支持的 DataFrame 类型，从而在 Spark 上执行高效的查询操作。

1）快照查询

用于查询给定操作后表的最新快照。对于写时复制表，直接读取所有分区下每个文件组内最新版本的数据文件，然后根据给定条件进行过滤而得到查询结果，查询延迟较低。对于读时合并表，需动态合并每个文件组内的日志文件和数据文件，然后根据给定条件进行过滤而得到查询结果，查询延迟较高。图 3.8 为日志文件和数据文件合并的示意图，Hudi 从日志文件读取所有增量数据形成一个数据集合后与数据文件通过 Join 操作进行合并更新，生成新版本的数据文件。

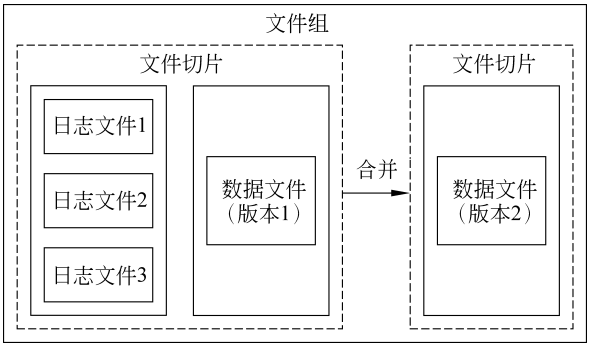
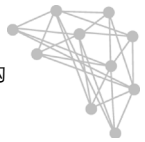


图 3.8　日志文件和数据文件合并示意图



2）增量查询

用于查询给定时间戳后的新写入数据。对于写时复制表，读取所有分区下每个文件组内最新版本的数据文件，数据文件中的每条数据都有一个关于该数据的提交时间，若给定查询的时间戳大于数据提交的时间戳，则跳过该数据。对于读时合并表，可根据给定查询的时间戳跳过某些数据文件和日志文件，从而实现更高效的增量查询。

3）读优化查询

与快照查询方式不同，读优化查询用于优化读时合并表的读取性能，该查询方式仅读取所有分区下的数据文件，然后根据给定条件进行过滤而得到查询结果，不执行动态合并文件操作，因此查询延迟较低。

例 3.2　图 3.9 为数据写入 MOR 表并执行 3 种不同查询方式的示意图。在时间戳为 1 的时刻执行 COMMITS 操作，将数据记录 A、B、C、D 和 E 写入数据文件；在时间戳为 2 的时刻执行 DELTA_COMMIT 操作，将更新数据记录 A1 和 D1 写入日志文件；在时间戳为 3 的时刻执行 DELTA_COMMIT 操作，将更新数据记录 A2、E1 和数据 F 写入日志文件；在时间戳为 4 的时刻执行 COMPACTION 操作，将数据文件和日志文件合并为新的数据文件。

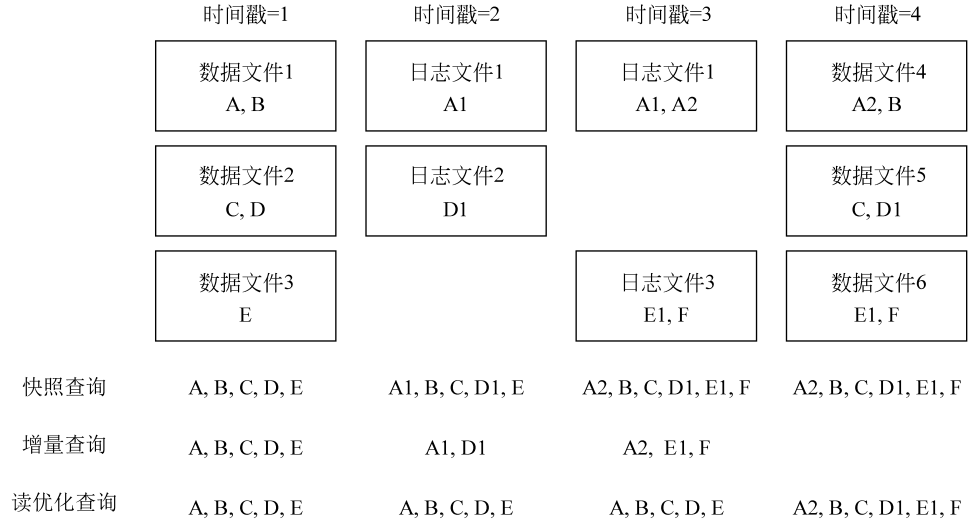


图 3.9　读时合并表执行查询示意图

3.4　向量数据库

3.4.1　向量数据库概述

随着互联网的普及、物联网设备的激增及数据存储技术的快速发展，全球数据量呈指数级增长。在大数据时代，数据不再仅仅是数字和文本，还包括了图像、视频、声音和各种传感器数据，这些数据往往蕴含着丰富的信息，给数据的处理和分析创造了新的机遇，也带来了前所未有的挑战。与此同时，以深度学习为代表的当代人工智能技术推动了机器学习模型达到前所未有的复杂度和精度，使得机器能够理解、分析并生成复杂的数据形式。

向量数据库是一种专门为存储、管理和查询高维向量数据而设计的数据库系统，这些向量数据通常来源于对图像、音频、文本等非结构化数据的特征提取，经过卷积神经网络、循环神经网络和 BERT 等机器学习模型转换而来。向量数据库能有效处理大量高维数据点，提供高效的数据检索和相似性搜索能力，其特点主要包括：

维度。向量数据具有多个维度，每个维度代表特征的一个方面。例如，一个文本向量可能有数百或数千个维度，每个维度对应一个词频或词嵌入的特定位置。

特征表示。向量数据以数学向量的形式来表示数据的特征，这使得数据可进行向量间的距离或内积计算等数学运算，从而度量数据点之间的相似度。

向量数据库的出现弥补了传统数据库在处理高维向量数据时的不足，特别是在需进行快速相似性搜索的场景下，提供了更为专业和高效的解决方案。与传统数据库相比，向量数据库在数据存储、查询机制、索引技术等方面具有显著区别，主要包括：

数据存储。传统的关系型数据库（如 MySQL）和文档数据库（如 MongoDB），主要处理结构化和非结构化数据，使用表格或文档作为数据存储单元，通常不直接支持高维向量的存储和高效检索。向量数据库则专注于存储和检索高维向量数据，且能高效地实现这些数据的相似性搜索。

查询机制。传统数据库中的查询通常是基于键值对的精确匹配或范围查询，而向量数据库的查询更多地依赖于向量间的距离或相似度计算（如最近邻搜索），这意味着向量数据库需采用特殊的数据结构和算法对相似性搜索过程进行优化。

索引技术。传统数据库使用 B^+ 树和哈希索引等技术来加速查询，而向量数据库则使用更复杂的索引结构，这些索引结构专门针对高维空间中的近似最近邻搜索进行了优化。

3.4.2 向量数据库的索引技术

向量数据库的核心在于其高效的索引技术，这些技术使得数据库能在海量的向量数据中快速定位到相似或相关的向量。由于现有的索引技术适用于不同的应用场景和数据类型，实际应用中需根据数据特点、查询需求和计算资源等因素进行选择和优化。下面介绍向量数据库中 3 种常用的索引技术。

1. KD 树

KD 树（K-dimension tree）是一种对 K 维空间中的数据点进行存储，以便对其进行快速检索的树形数据结构。它是一种特殊的二叉树，主要用于多维空间关键数据的搜索（如范围搜索和最近邻搜索）。KD 树的构建采用递归的方式，通常从根节点开始，选择某一维进行划分，将数据点按照该维的值进行排序，然后选择中位数作为划分点，将数据点划分为两个子集，并分别作为左子树和右子树的根节点。递归地在每个子集中选择新的维度进行划分，直到所有数据点都被划分到叶子节点为止。下面通过一个例子展示 KD 树的构建过程。

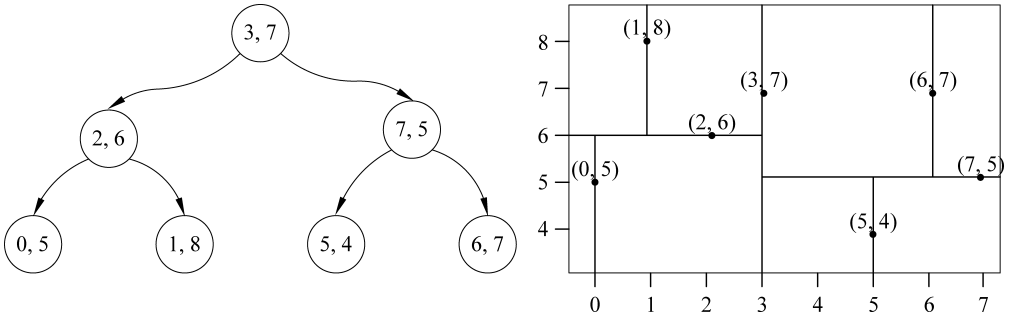
例 3.3 给定 7 个二维（$K=2$）的数据点组成的数据集 $X=\{(3,7),(2,6),(0,5),(1,8),(7,5),(5,4),(6,7)\}$，为其构建 KG 树。

首先，将 X 中的数据点按照第一维（如 x 轴）进行排序，得到{(7,5),(6,7),(5,4),(3,7),(2,6),(1,8),(0,5)}，选择中位数 3 对应节点(3,7)作为划分点，将数据点划分为子集{(7,5),(6,7),(5,4)}和{(2,6),(1,8),(0,5)}。对于根节点(3,7)，将其第 1 维数据 3 与待分配节

点(2,6)的第 1 维数据 2 进行比较,由于 2 小于 3,则该节点被分配到左子树并作为其根节点。

接着,继续递归地在每个子树上进行划分,但此时切换到第 2 维(如 y 轴)进行划分。将左子树根节点(2,6)的第 2 维数据 6 与所有待分配节点(1,8)的第 2 维数据 8 进行比较,由于 8 大于 6,则该节点被分配到右子树,(0,5)被分配到左子树。

同样地,对右子树进行类似的划分,递归地执行该过程,直到所有数据点都被分配到 KD 树的叶子节点上,最终得到的二维 KD 树如图 3.10 所示。不难看出,对于 KD 树中的任何一个非叶子节点,其左子树中的所有节点在指定维度(当前正在考虑的维度)上的值都小于该节点在相应维度上的值,而右子树中所有节点在指定维度的值都大于该节点在相应维度上的值。




动画 3-1

图 3.10　KD 树

对于给定的查询点,KD 树的查询基于最近邻搜索的思想,从根节点开始,沿着与查询点最近的子树方向进行搜索,直到达到叶子节点。然后回溯到父节点,检查另一个子树中是否存在距离查询点更近的节点。通过不断回溯和搜索,最终找到距离查询点最近的节点。KD 树在处理低维数据时表现良好,但在高维空间中由于“维数灾难”的存在,树的性能会急剧下降。因此,在实际应用中,需根据数据的维度和分布来选择合适的索引技术。

2. 局部敏感哈希

局部敏感哈希(locality-sensitive hashing,LSH)是一种用于高维空间数据近似最近邻搜索的哈希技术,其基本原理是,若两个数据点在原始空间中相近,则其经过哈希函数映射后得到的哈希值也相近。LSH 通过设计一系列哈希函数族将数据点映射到不同的哈希桶中,从而实现快速查找。哈希函数的选择是 LSH 的关键所在,一种常见的方法是随机投影法,将高维数据点随机投影到低维空间中,然后计算投影后的向量之间的距离。若两个数据点在原始空间中相近,那么它们在低维空间中投影向量之间的距离也较小。因此,可选择一个合适的阈值,将距离小于该阈值的投影向量映射到同一个哈希桶中。

LSH 的优点在于其可扩展性和处理高维数据的能力。然而,由于哈希函数的选择和阈值的设定对性能影响较大,因此需在实际应用中根据数据的特性进行调优。

3. 分层导航小世界图

分层导航小世界图(hierarchical navigable small world,HNSW)是一种基于图的索引技术,它结合了小世界网络和层次结构的优点,能在保持搜索效率的同时降低内存消耗。HNSW 的基本思想是将数据点组织成一个多层的图结构,每层图中的节点都与下一层图中

的多个节点相连。通过层次结构和导航策略，HNSW 能在图中快速定位到与查询点相近的节点。

HNSW 的构建过程包括初始化和优化两个阶段。在初始化阶段，随机选择一部分数据点作为初始节点并构建第一层的图结构，然后将剩余的数据点依次插入到图中，并更新图的结构。在优化阶段，通过迭代地选择图中的节点进行插入和删除操作，进一步优化图的结构和性能。

HNSW 的查询过程从最高层的图开始，逐层向下搜索与查询点相近的节点。在每一层利用图的连接关系和导航策略进行搜索，直到找到满足要求的节点或达到指定的搜索深度为止。HNSM 通过采用层次结构，将边按特征半径进行分层，从而将搜索复杂度降到了 $O(\log n)$，其中 n 为图中的节点数。

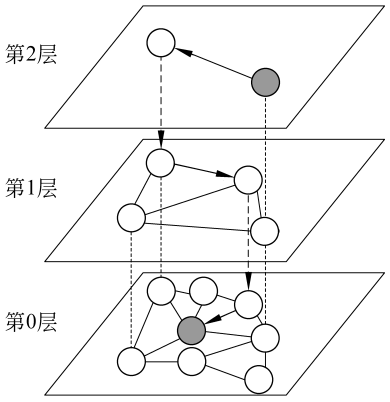


图 3.11　三层 HNSW

例 3.4　图 3.11 给出一个三层 HNSW，其中，第 2 层的黑色节点为遍历的入口点，第 0 层的黑色节点为目标点。从第 2 层开始，搜索按照箭头线的方向展开；在第 1 层利用图的连接关系和导航策略逐渐接近目标节点，最终在第 1 层匹配到目标节点。在该过程中，搜索半径逐渐减小。

HNSW 在处理高维数据和大规模数据集时表现出色，具有较低的内存消耗和较高的搜索效率。然而，其构建和优化过程相对复杂，计算开销也较大。

3.4.3　向量数据库的搜索技术

在自然语言处理、图像识别和推荐系统等领域，高维向量数据处理是其核心任务。如何在这些高维向量数据中高效准确地找到相似的向量（向量搜索），是向量数据库的技术关键。下面介绍近似最近邻搜索算法和常用的相似性度量方法。

1. 近似最近邻搜索

在向量数据库中，最近邻搜索是最基本、最重要的操作之一。然而，在高维空间中直接进行最近邻搜索往往面临计算复杂度高、效率低等问题。为了解决这一问题，人们提出了近似最近邻（approximate nearest neighbor，ANN）搜索算法，其基本思想是在保证一定精度的前提下，通过优化搜索策略来降低计算复杂度，即通过牺牲一定的精度来提高搜索效率。具体而言，ANN 算法通常在构建索引时采用聚类和哈希等启发式方法，以减少需搜索的向量数量。在查询时，ANN 算法首先根据索引找到可能与查询向量相似的候选向量，然后再对这些候选向量进行精确的相似度计算，从而找到最近邻向量。

根据不同的实现方式和应用场景，基于树结构的算法（如 KD 树、球树等）和基于哈希的算法（如局部敏感哈希、谱哈希等）是最常见的两类 ANN 算法。基于树结构的算法利用树形结构来组织向量数据，通过剪枝等策略来减少搜索空间；基于哈希的算法则通过哈希函数将高维向量映射到低维空间，然后在低维空间中进行搜索。

ANN 算法一方面希望算法能找到尽可能准确的最近邻向量，另一方面也希望算法能在