# 在数据库中集成结构化和非结构化查询

# 1. 引言

## 1.1. 结构化查询与非结构化查询概述

在数据领域，查询是信息检索的核心，它使用户能够从庞大的数据存储中提取有意义的洞察。结构化查询（如SQL，Structured Query Language）用于操作存储在关系型数据库中的有组织数据。这类查询依赖预定义的模式（schema），非常适合精确执行诸如joins、聚合（aggregations）和过滤（filtering）等复杂操作。

相反，非结构化查询旨在从缺乏固定结构的数据（如文档、图片、视频或对话日志）中提取洞察。这类查询利用自然语言处理（Natural Language Processing, NLP）、向量检索（vector search）和机器学习等技术来发现模式和检索相关内容。结构化查询在清晰性和效率上具有优势，而非结构化查询则在灵活性和适应性上表现卓越，两者在现代数据系统中都不可或缺。

补充示例：在讨论结构化与非结构化查询的差异时，可以增加具体场景示例：

结构化查询： 在一个电商平台数据库中，查询出“某商品在过去一个月的销售总量和区域分布”。

非结构化查询： 在客户服务聊天记录中，检索包含“延迟发货”抱怨的文本。

语义层面：讨论结构化和非结构化数据结合的场景，例如知识图谱的构建如何将非结构化数据转化为结构化数据

## 1.2. 查询集成在数据库系统中的重要性

结构化查询和非结构化查询的集成代表了数据库技术的一次变革性飞跃。传统数据库设计用于处理结构化数据，对于非结构化数据的上下文和多样化特性往往表现不足。而专注于非结构化查询的系统则缺乏处理结构化数据操作时所需的精确性和逻辑性。

通过融合两种查询模式，查询集成实现了一种统一的数据检索方法，使组织能够在关系表与文本、视觉或多模态（multi-modal）数据集之间无缝查询。这种协同作用支持更全面的洞察，助力复杂分析工作流，并增强决策能力。此外，依托Retrieval-Augmented Generation (RAG) 和声明式查询框架（declarative query frameworks）等前沿技术的混合系统正在重新定义我们与数据库交互的方式，提供更自然、智能和可扩展的数据探索能力。

## 1.3. Wiki 的范围与主要目标

本Wiki旨在为理解和实现数据库系统中结构化查询与非结构化查询的集成提供全面的指南。结合最新的研究、工具和方法论，本文深入探讨了实现这种集成的关键技术，如SQL增强、基于LLM（Large Language Model）的查询生成，以及混合检索技术。

主要目标包括：

解释结构化查询与非结构化查询的基本概念与区别。

探讨统一查询范式的前沿工具和系统，如声明式查询引擎（declarative query engines）和向量相似性检索（vector similarity search）框架。

提供实际案例研究和示例，以展示真实场景中的应用。

提供设计、实现和优化混合查询系统的最佳实践。

****\*本Wiki能为您提供必要的知识和工具，帮助您掌握查询集成的最新趋势，充分挖掘现代数据库的潜力。\*****

# ****2. 查询基础****

## 2.1. 定义与关键区别

查询是数据库操作的核心，使用户能够高效地提取、分析和操作数据。主要有两种类型的查询——结构化查询和非结构化查询，两者各有用途。

结构化查询（SQL）：  
结构化查询基于预定义的模式（schema），利用\*\*Structured Query Language (SQL)\*\*与关系型数据库交互。这类查询设计用于处理表格式数据，可精确执行如过滤、连接（joins）和聚合（aggregations）等操作。例如，SQL可用于提取特定月份的所有销售记录，或计算按产品类别划分的总收入。

非结构化查询：  
非结构化查询用于操作没有固定模式的数据，如文档、视频或图片。这类查询依赖于如\*\*自然语言处理（Natural Language Processing, NLP）和向量检索（vector search）\*\*等高级技术来检索和处理信息。例如，非结构化查询可能涉及搜索包含特定情感的客户评论，或从视频数据中提取特定对象。

两者的核心区别在于关注点：结构化查询强调表格式数据的精确性和逻辑性，而非结构化查询则提供处理多样化、非组织化数据的灵活性。

## 2.2. 各类查询的常见使用场景

结构化查询（Structured Query）：

财务报告：计算利润率或趋势。

库存管理：识别特定商品的库存水平。

用户分析：检索网站的用户活动日志。

非结构化查询（Unstructured Query）：

情感分析：从社交媒体中提取用户观点。

多媒体检索：在视频数据集中定位特定场景。

文本挖掘：总结客户反馈以改进产品。

通过发挥各自的优势，结构化查询和非结构化查询解决了数据分析中互补的需求。

## 2.3. 查询集成的挑战

尽管两种查询类型各自擅长不同领域，但其集成面临显著挑战：

模式不匹配： 结构化查询依赖严格的模式，而非结构化查询处理异构格式，使得统一的数据表示变得困难。

举例说明：例如在数据库中，一个表格可能有“产品ID”这种结构化字段，但评论内容（如客户反馈）却是非结构化数据，需要通过语义分析来建立关联。

提到处理方案：利用NLP技术自动对非结构化内容进行分类或标签化。

性能瓶颈： 将结构化操作（如连接）与计算密集型非结构化查询（如NLP）结合，可能导致延迟和效率问题。

通过实例展示如何通过优化查询计划减少查询时间，例如在多模态数据查询时，优先从文本数据索引中筛选出候选内容，再匹配图片或视频。

语义理解： 将非结构化查询转化为结构化操作，或反之，需要实现上下文和语义的对齐，这往往较难实现。

系统兼容性： 现有数据库通常针对结构化或非结构化数据进行了优化，在混合处理上存在技术差距。

### 总结

克服这些挑战需要采用如\*\*Retrieval-Augmented Generation (RAG)和声明式查询框架（declarative query frameworks）\*\*等混合解决方案，为统一查询系统铺平道路。

# 3. 技术与方法

## 3.1. 结构化查询系统 (structured Query Systems)

结构化查询系统基于关系型数据库的规则化数据模式，主要利用\*\*SQL (Structured Query Language)\*\*执行高效、精确的数据操作。这类系统的技术不断演进，以支持更复杂和优化的查询需求。

SQL优化技术（SQL Optimization Techniques）：  
SQL查询优化技术旨在提升查询性能和效率。通过索引优化、查询计划调整（Query Plan Optimization）以及语句重写，数据库可以大幅降低响应时间。例如，使用索引来加速搜索，或通过重写复杂的子查询以减少计算开销。

高级SQL特性（Advanced SQL Features）：  
现代SQL支持越来越多的高级功能，如递归查询（Recursive Queries）和窗口函数（Window Functions）。递归查询可用于遍历分层数据（如组织结构树），而窗口函数可以计算排名、累计总和等统计数据，而无需改变数据表结构。

****窗口函数的应用：在分析用户行为时，计算每月新增用户的累计总和：****

SELECT month, SUM(new\_users) OVER (ORDER BY month) AS cumulative\_users    
FROM user\_activity;

声明式查询框架（Declarative Query Frameworks, e.g., F1 Query）：  
声明式查询框架（如F1 Query）支持跨多种数据源的联合查询（Federated Queries），并融合了OLTP（联机事务处理）与OLAP（联机分析处理）的特点。它们允许用户通过声明式语法定义业务逻辑，而无需编写复杂的管道代码，极大地简化了查询流程。

## 3.2. 非结构化查询系统 (Unstructured Query Systems)

非结构化查询系统专注于处理缺乏固定模式的数据，如文本、图像和视频。这些系统通过智能技术来解析和检索有意义的信息。

自然语言查询处理（Natural Language Query Processing）：  
自然语言处理（NLP）使用户能够通过日常语言与数据库交互。这种方式对非技术用户尤为友好。通过NLP，系统可以解析用户输入的自然语言问题，并将其转换为结构化查询，完成如搜索文档或回答问题等任务。

向量搜索与相似性匹配（Vector Search and Similarity Matching, e.g., VBase）：  
向量搜索技术通过将数据（如文本、图像）编码为高维向量，计算向量间的相似性，帮助识别最相关的内容。例如，VBase是一个统一在线向量相似性搜索系统，支持快速和高效的多模态查询。

多模态查询规划（Multi-Modal Query Planning, e.g., CAESURA）：  
CAESURA等系统支持多模态数据的联合查询，如文本与图像的结合。它们利用不同数据类型的嵌入模型来整合语义信息，并生成复杂的查询计划，适用于跨文档、音视频的检索任务。

## 3.3. 混合系统 (Hybird Systems)

混合系统通过结合结构化和非结构化查询技术，提供了一种更全面的解决方案。这类系统整合了多种技术，既能处理关系型数据，又能支持上下文驱动的非结构化数据查询。

检索增强生成（Retrieval-Augmented Generation, RAG）：  
RAG是一种结合信息检索和生成模型的技术。在查询处理过程中，系统先从外部数据源检索相关内容，再利用生成式语言模型（如LLM）生成答案。它非常适用于处理需要外部知识支持的复杂任务。

SQL与LLM查询的结合（Combining SQL and LLM-Based Queries）：  
通过将SQL的结构化查询功能与大语言模型（LLM）的自然语言处理能力相结合，混合系统能够支持更灵活的查询方式。例如，用户可以通过自然语言输入问题，系统将其解析为SQL语句，完成结构化数据的查询。

视频与复杂数据查询的声明式线索（Declarative Clues for Video and Complex Data Queries）：  
复杂数据（如视频分析）的查询需要结合声明式语法和智能算法。例如，通过声明式线索框架，用户可以指定某些关键事件或目标对象，系统则基于这些指令从视频中提取所需的信息。

混合查询系统的真实场景应用：

零售分析：  
将销售数据（结构化）与社交媒体趋势（非结构化）结合，用于预测需求。

SQL 检索销售数据，NLP 模型处理推文情感。

医疗领域：  
搜索患者记录（结构化）与医生笔记（非结构化）以识别治疗模式。

示例：“在过去6个月内报告疲劳的糖尿病患者。”

### 总结

通过结合这些技术和方法，数据库系统正在逐步突破传统限制，支持更复杂和多样化的数据查询需求，同时大幅提升查询效率与用户体验。

# 4. 关键应用

## 4.1. 企业数据管理 (Enterprise Data Management)

企业中的数据管理需要结合结构化和非结构化查询技术，以支持高效的数据分析与决策过程。以下是几个关键场景：

企业场景中的Text-to-SQL（Text-to-SQL in Enterprise Scenarios）：  
在企业环境中，Text-to-SQL 技术使非技术用户能够通过自然语言提出查询请求，系统自动将其转换为SQL语句并运行。例如，业务经理可以通过输入“过去一个季度的销售总额是多少？”来获取对应的财务数据。  
在实际应用中，企业数据库的复杂性（如多表关联和定制化规则）对Text-to-SQL系统提出了更高要求。这需要结合上下文信息（如数据库schema和历史查询记录）以提高转换准确性。

语义列类型检测 (Semantic Column-Type Detection)：  
此技术通过分析数据列的内容，为其分配一个具有实际意义的标签（如“人口”、“地址”或“收入”）。例如，在一个拥有数千个表的企业数据仓库中，语义列类型检测可以快速帮助用户识别目标列并加快数据整合和分析流程。  
这种方法通常结合预训练模型和上下文提示，以提高对非标准化企业数据的适应性。

## 4.2. 信息检索 (Information Retrieval)

信息检索是将非结构化数据和复杂知识集成到查询系统中的核心任务之一。

知识图谱集成 (Knowledge Graph Integration)：  
知识图谱通过将实体（如人物、地点）和关系（如“位于”或“拥有”）连接起来，提供了语义丰富的上下文信息。在复杂查询中，知识图谱可用于增强查询的语义理解能力。例如，通过将用户输入的问题映射到知识图谱中的节点和路径，系统可以回答类似“某公司在过去五年收购了哪些企业？”的问题。  
技术上，知识图谱可以与语言模型（如LLMs）集成，用于动态扩展图谱或提升推理能力。

迭代和递归检索 (Iterative and Recursive Retrieval)：  
对于需要多步推理的查询，迭代和递归检索能够逐步细化查询目标。例如，用户可能询问“在2020年总收入超过10亿美元的公司中，哪些公司投资了AI初创企业？”，系统需要分步检索相关数据并合并结果。  
通过结合检索增强生成（RAG）和图搜索技术，迭代与递归检索能够更高效地处理跨领域、多层级的复杂查询。

## 4.3. 新兴趋势 (Emerging Trends)

随着人工智能技术的迅速发展，数据库系统的功能与智能化水平也在持续提升。

LLMs在数据库系统中的集成 (Integration of LLMs in Database Systems)：  
大语言模型（LLMs）通过自然语言理解和生成能力，将传统数据库与智能查询结合。例如，用户可以使用非正式语句进行查询，如“展示过去一年利润最高的五个部门”。LLMs能够将此类查询转化为SQL，并结合非结构化数据生成更全面的答案。  
此外，LLMs在查询模糊匹配、语义解析和上下文生成方面表现出色，为混合查询系统提供了新方向。

利用AI进行查询优化 (Leveraging AI for Query Optimization)：  
人工智能在查询优化中扮演越来越重要的角色。通过机器学习模型，系统能够预测查询执行计划的性能，自动选择最优策略。  
例如，在大规模数据查询中，AI模型可以动态调整索引选择或缓存分配，从而减少查询时间和计算资源消耗。

### 总结

通过上述应用，数据库技术正从传统的静态系统向动态、智能化方向转变，为复杂数据查询和决策支持提供更强大的能力。

# 5. 案例研究与实际示例

## 5.1. 视频分析中的查询优化 (Query Optimization in Video Analysis)

随着多媒体数据量的快速增长，视频分析已成为现代数据库的重要应用场景之一。然而，由于视频数据的高维性和非结构化特性，其查询优化面临巨大挑战。

挑战：  
视频分析需要从长时视频中快速定位特定内容（如目标对象、动作或事件）。传统的线性扫描方法效率低下，无法满足实时需求。此外，用户查询通常是高层次的语义请求（如“寻找某人进入房间的时间点”），需要在底层数据中解析复杂的语义关系。

解决方案：  
使用声明式查询线索（Declarative Clues）框架，用户可以指定目标特性（如时间窗口或对象特征）。例如，通过优化索引和基于向量搜索的特征匹配，系统能够加速目标帧的检索。例如：

查询：“查找视频中5:00到10:00出现红色汽车的所有实例。”

解决方案：说明CAESURA如何通过预计算嵌入和时间索引优化查询。

此外，现代系统结合多模态数据（如视频中的音频和文本）进一步提升查询效率。例如，在一项基于视频会议记录的分析中，系统可以通过同步分析讲话者的音频和视频画面，快速定位相关内容。

## 5.2. 基于LLM的SQL生成 (LLM-Driven SQL Writing, e.g., PURPLE)

PURPLE 是一种专注于提升SQL生成质量的语言模型，它展示了如何结合LLM生成SQL以优化结构化数据的查询流程。

案例：  
在一个跨部门的企业应用中，用户需要查询“某一时间段内特定区域的销售增长趋势”。普通用户可能无法直接编写复杂的SQL语句。PURPLE通过自然语言解析用户的查询意图，将其转化为精准的SQL表达。  
例如，用户输入“给我2023年第一季度华南地区的销售增长数据”，系统生成以下SQL：

SELECT region, SUM(sales) AS total\_sales

FROM sales\_data

WHERE region = 'South China' AND date BETWEEN '2023-01-01' AND '2023-03-31'

GROUP BY region;

关键优势：

提高非技术用户的查询能力。

自动优化生成的SQL语句以提升查询性能。

减少开发者对复杂语法的依赖，提升业务效率。

## 5.3. 基于图检索的多跳推理 (Multi-Hop Reasoning with Graph-Based Retrieval)

多跳推理（Multi-Hop Reasoning）是解决复杂查询任务的核心技术，特别是在需要跨多个数据源或推导复杂逻辑的场景中。

应用场景：  
在法律文档分析中，用户可能需要回答“哪些案件涉及特定法规，并且这些案件的最终裁定中包含对环境保护的具体引用？” 这种任务需要跨多个文档检索和关联信息。

技术实现：  
基于知识图谱（Knowledge Graph）的检索系统能够有效支持多跳推理：

图构建： 首先将法律文档的关键实体（如案件、法规、判决）和关系（如引用、适用）组织为知识图谱。

示例：查询“2023年哪些公司收购了AI初创企业？”通过将实体（公司）和关系（收购）映射到知识图谱，实现准确检索。

递归查询： 使用迭代方法在图中逐层检索。例如，系统先找到包含特定法规的案件，再进一步查询这些案件是否包含环境保护的具体引用。

语义扩展： 通过结合LLM，系统能够理解用户查询中的隐含语义，并生成更精确的推理路径。

实际效果：  
这种方法不仅提高了复杂查询的准确性，还显著降低了查询时间，使其能够适应实时业务需求。

### 总结

通过这些实际案例，可以看出现代数据库在处理复杂查询时的技术能力不断提升。无论是优化非结构化视频数据的查询，还是结合LLM提升SQL生成能力，或是利用图结构支持复杂推理，数据库系统正在为多样化的应用场景提供更智能、高效的解决方案。

# 6. 挑战与解决方案

## 6.1. 应对SQL的局限性

虽然SQL在处理结构化数据时表现出色，但其设计上的局限性也在复杂场景中暴露无遗。以下是改进SQL能力的一些关键技术：

SQL管道语法 (SQL Pipe Syntax)：  
传统SQL语法的顺序固定（如SELECT...FROM...WHERE...），可能会导致复杂查询难以表达或可读性差。管道语法（Pipe Syntax）通过允许操作以自然的顺序链接执行，优化了查询的灵活性和可读性。例如：

-- 传统嵌套查询

SELECT SUM(sales)

FROM (SELECT region, sales FROM sales\_data WHERE region = 'East');

-- 使用管道语法

FROM sales\_data

|> FILTER region = 'East'

|> AGGREGATE SUM(sales) AS total\_sales

|> ORDER BY total\_sales DESC;

这种语法减少了对嵌套子查询的依赖，使复杂逻辑的实现更加直观。

声明式扩展 (Declarative Extensions)：  
通过声明式查询框架（Declarative Query Frameworks），SQL可以无缝支持新的操作功能，而无需重新设计查询语法。例如，使用扩展语法支持窗口操作或递归查询，这对复杂分析任务尤为重要。同时，这些扩展通常与现有SQL生态系统兼容，降低了迁移成本。

## 6.2. 解决非结构化查询问题

非结构化查询需要处理多样化数据，其核心挑战在于数据的复杂性与检索精度。

数据检索准确性 (Data Retrieval Accuracy)：  
在非结构化数据中，信息分布广泛且缺乏统一格式，检索系统需要在保持高召回率的同时尽量提高精度。解决方案包括：

向量搜索技术（Vector Search）： 使用语义嵌入将数据映射到高维空间，计算查询与数据点之间的语义相似性。

多模态嵌入（Multi-Modal Embeddings）： 将文本、图像和音频等不同数据类型整合到统一的语义空间中，提高跨模态检索能力。

RAG系统中的噪声与幻觉问题 (Noise and Hallucination Mitigation in RAG Systems)：  
检索增强生成（Retrieval-Augmented Generation, RAG）系统常会因低质量检索或语言模型的“幻觉”（hallucination）生成不准确答案。改进方法包括：

上下文验证 (Context Verification)： 在生成阶段验证检索内容的相关性，过滤掉噪声信息。

联合训练 (Joint Training)： 通过联合优化检索器与生成器，使两者更高效协作，减少误导性答案的生成。

## 6.3. 桥接结构化与非结构化系统

混合系统旨在整合结构化和非结构化查询，但这需要克服语义和技术上的多重障碍。

混合索引技术 (Hybrid Indexing Techniques)：  
混合系统需要同时支持关系型数据的索引（如B树、哈希索引）和非结构化数据的索引（如语义向量索引）。解决方案包括：

稀疏与密集索引结合 (Sparse and Dense Indexing)： 将传统关键字匹配（如BM25）与语义向量检索相结合，以在精确性和召回率之间取得平衡。

层次化索引 (Hierarchical Indexing)： 通过分层存储与检索，加快跨域查询速度。例如，先粗略定位目标文档，再在文档内部进行精细检索。

查询与文档对齐 (Query-Document Alignment)：  
跨模态数据的查询需要对用户意图和文档内容进行语义对齐。关键技术包括：

查询重写 (Query Rewriting)： 根据查询上下文动态调整检索策略，提高匹配质量。

生成式对齐 (Generative Alignment)： 使用生成模型预测潜在答案，再根据预测结果回溯检索路径，优化查询结果。

### 总结

通过应对SQL的语法和功能限制、优化非结构化数据的检索精度以及开发混合索引和对齐技术，现代数据库系统正逐步克服传统架构的局限性。这些创新不仅增强了查询系统的适应性和扩展性，也为多样化的数据分析需求提供了更全面的解决方案。

# 7. 最佳实践与工具

## 7.1. 查询规划与执行策略

高效的查询执行依赖于良好的规划和优化。以下是几个关键的策略：

查询优化 (Query Optimization):  
查询优化是提高查询性能的核心环节。通过使用查询计划生成器（Query Planner），数据库可以评估多种执行路径并选择成本最低的方案。例如，通过索引利用、分区裁剪（Partition Pruning）和重写嵌套查询，显著降低查询延迟。

分布式查询执行 (Distributed Query Execution):  
对于大规模数据，分布式执行策略（如MapReduce）能够将查询任务分解到多个节点进行并行处理，提高效率。例如，Google的F1 Query框架通过分布式执行实现了高效的跨表联邦查询（Federated Queries）。

自适应查询优化 (Adaptive Query Optimization):  
在动态数据环境中，系统可根据实时查询条件和数据分布，调整执行策略。例如，在实时流数据分析中，系统可以动态调整窗口大小以优化查询结果的准确性和延迟。

## 7.2. 查询集成工具

随着数据库系统的复杂性增加，选择合适的工具能够显著提升查询集成的效率和精度。

开源资源 (Open Source Resources):

GitHub Projects:  
GitHub上提供了许多优秀的开源项目，如LangChain 和 LlamaIndex，它们能够将LLMs与传统数据库集成，支持混合查询和多模态检索。

向量数据库:  
开源工具如Weaviate 和 Milvus 提供高效的向量检索功能，适合处理非结构化数据的相似性搜索。

示例：它如何支持文本和图像数据的相似性检索。

使用场景：搜索包含文本情感和关联图片的产品评论。

专有系统 (Proprietary Systems):

企业级解决方案:  
企业专有系统（如Google BigQuery和Amazon Redshift）支持大规模查询和数据分析，同时具备高效的查询优化功能。

集成服务:  
诸如Microsoft Azure Synapse Analytics 提供了数据湖与数据仓库的统一分析功能，适合跨模态数据的查询集成。

## 7.3. LLM驱动系统的效率与成本考量

LLM驱动的查询系统虽然功能强大，但也带来了显著的资源消耗和成本问题。

性能优化 (Performance Optimization):

上下文压缩: 为减少查询负担，利用上下文压缩技术优化提示输入（Prompts），降低LLM的计算复杂度。

增量检索 (Incremental Retrieval): 通过逐步细化检索范围，减少不必要的计算量。

成本控制 (Cost Control):

轻量模型 (Lightweight Models): 使用参数高效调优（如LoRA）或小型专用模型来处理特定任务，降低运行成本。

按需部署 (On-Demand Deployment): 在需要时动态调用LLM，避免持续运行带来的资源浪费。

缓存与复用 (Caching and Reuse): 对频繁查询结果进行缓存或部分复用，减少重复计算的开销。

资源规划 (Resource Planning):  
结合业务需求和数据规模，合理规划云服务实例的使用。例如，通过调整计算资源的配置以适应查询高峰，既能保障性能又避免资源浪费。

### 总结

通过优化查询规划、选择合适的工具以及合理控制资源成本，可以在实现强大功能的同时提高效率并降低费用。这些最佳实践和工具为复杂查询系统的设计与实现提供了有力支持，推动了数据库技术的进一步发展。

# 8. 结论

## 8.1. 数据库查询集成的未来方向

随着数据类型和规模的不断增长，数据库查询集成将面临更多挑战，同时也迎来了巨大的创新机遇。未来，以下几个方向值得关注：

更深度的多模态集成 (Deeper Multi-Modal Integration):  
支持文本、图像、视频、音频等多模态数据的统一查询，通过优化索引技术和语义建模，进一步提升查询能力。

智能化与自动化 (Intelligent and Automated Queries):  
利用LLMs和机器学习技术，实现更加智能化的查询解析、自动优化和实时调整，满足复杂业务场景的需求。

联邦学习与隐私保护 (Federated Learning and Privacy-Aware Queries):  
在支持跨组织数据共享的同时，注重用户隐私保护，构建更安全的查询集成架构。

## 8.2. 对实践者的建议

选择合适的工具和框架: 根据业务需求和数据特点选择合适的开源资源或专有系统（如LangChain、BigQuery等），避免盲目追求最新技术。

注重查询性能优化: 利用上下文压缩、分布式执行和动态索引等技术，平衡性能和成本，特别是在LLM驱动系统中。

结合业务需求定制查询流程: 确保技术实现能够直接服务于业务目标，并持续改进系统的适配性。

## 8.3. 学习总结

本Wiki围绕结构化与非结构化查询的集成，深入探讨了其技术方法、实际应用和未来方向。通过案例研究和最佳实践，我们了解到：

查询集成是现代数据库技术发展的必然趋势，其核心在于平衡精确性和灵活性。

混合查询系统（如RAG）结合了结构化查询的逻辑性与非结构化查询的语义能力，推动了数据分析的全面化。

高效的工具和优化策略不仅能提高查询性能，还能有效控制成本，为复杂应用场景提供支持。

# 9. 参考文献

## 9.1. 文献与论文

1. Yu et al., Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task, EMNLP, 2018.
2. Zhao et al., Retrieval-Augmented Generation (RAG) and Beyond: A Comprehensive Survey, arXiv, 2024.
3. Demiralp et al., Making LLMs Work for Enterprise Data Tasks, NEDBDay, 2024.
4. Pan et al., Unifying Large Language Models and Knowledge Graphs: A Roadmap, arXiv, 2023.
5. Zhong et al., Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation, arXiv, 2024.

## 9.2. 工具与开源资源

1. LangChain,<https://github.com/hwchase17/langchain>.
2. LlamaIndex,<https://github.com/jerryjliu/llama_index>.
3. Weaviate,<https://weaviate.io/>.
4. Milvus,<https://milvus.io/>.

## 9.3. 扩展阅读

1. Hellerstein et al., Ground: A Data Context Service, CIDR, 2017.
2. Borgeaud et al., Improving Language Models by Retrieving from Trillions of Tokens, ICML, 2022.
3. Gao et al., Retrieval-Augmented Generation for AI-Generated Content: A Survey, arXiv, 2024.

通过以上资源，实践者可以更深入地了解数据库查询集成的最新技术和实际应用，为复杂场景中的数据管理提供支持。