数据库技术与应用

# 数据库系统发展史

授课教师：计算机系王健楠

授课学期：2026年（春季）

# 半个世纪的数据库系统

（1960年代 – 2020年代）

| 时间 | 事件 |
|---|---|
| **1960年代早期 – 1970年代早期** | 导航数据库帝国 |
| **1970年代中期 – 1980年代中期** | 数据库第一次世界大战 |
| **1980年代中期 – 2000年代早期** | 关系数据库帝国 |
| **2000年代中期 – 至今** | 数据库第二次世界大战 |

时间轴：1960年代至今

**参考资料**

- https://en.wikipedia.org/wiki/Database#History
- 《历史的轮回》（迈克尔·斯通布雷克，乔·赫勒斯坦）
- VLDB 40周年座谈会

# 导航数据库帝国时代

（1960年代早期 – 1970年代早期）

## 导航型数据模型

**如何组织数据？**

将数据组织成多维空间（即记录的空间）。

**如何访问数据？**

通过跟随记录之间的“指针”进行导航访问。

## 学生选课场景

在导航数据库中，程序员必须像“导航员”一样，显式地顺着指针路径查找数据。例如：查找“张三”选修了哪门课？

| 学生记录 | | 选课记录 | | 课程记录 |
|---|---|---|---|---|
| **姓名：张三** | 指针跳转 → | **成绩：85分** | 指针跳转 → | **课程：数据库** |
| PTR: 0x1A | | PTR: 0x2B | | END |

路径：首先定位学生“张三”，通过指针找到关联的选课记录，再通过选课记录中的指针找到对应的课程信息。

## 查尔斯·巴赫曼

(Charles Bachman)

导航数据库之父

**1973年 ACM 图灵奖**

计算机界最高荣誉

**图灵奖演讲题目**

*《程序员作为导航员》*

*(The Programmer As Navigator)*

# 导航数据库帝国时代

代表性系统与标准化进程

## IDS 系统

1964年 • 通用电气

导航数据库的鼻祖

**应用场景：制造业零件管理**

就像管理一辆汽车的成千上万个零件，IDS 首次实现了复杂的“零件清单（BOM）”管理，让工厂能精确追踪每个螺丝钉属于哪个组件。

## IMS 系统

1966年 • IBM

导航数据库的巅峰

**应用场景：阿波罗登月计划**

为了把人类送上月球，NASA 需要管理土星五号火箭的数百万个组件。IMS 系统就像一个超级目录树，确保了登月任务海量数据的零差错管理。

## IDMS 系统

1973年 • 古德里奇

商业化的大成者

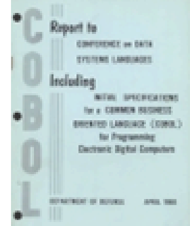
**应用场景：企业级商业管理**

完全遵循标准化的设计，使其成为当时最流行的商业数据库，广泛用于大型企业的库存、财务和人事管理，是关系数据库出现前的市场霸主。

## CODASYL 组织

数据系统语言会议 (1969)

在它之前，各家数据库互不兼容（像有的车靠左开，有的靠右开）。CODASYL 制定了统一的**导航数据库标准接口**，让大家有章可循。




**历史里程碑**

1969年发布 DBTG 报告

确立了导航模型作为行业标准

# 关系模型的诞生

（1970年代）

## 如何组织数据？（数据被组织成二维表）

学生表

| 学号 | 姓名 |
|---|---|
| 001 | 张三 |
| 002 | 李四 |

选课表

| 学号 | 课号 | 成绩 |
|---|---|---|
| 001 | 101 | 85 |
| 002 | 102 | 90 |

课程表

| 课号 | 课程名 |
|---|---|
| 101 | 数据库 |
| 102 | 算法 |

**数据被**组织成独立的二维表，表之间通过“值”（**如学号、**课号）建立逻辑关联，而非物理指针。

## 如何访问数据？（声明式语言）

任务：查找“张三”选修了哪门课？

```
SELECT 课程.课程名, 选课.成绩
FROM 学生, 选课, 课程
WHERE 学生.姓名 = '张三'
AND 学生.学号 = 选课.学号
AND 选课.课号 = 课程.课号;
```

“告诉系统你**想要什么**（What），而不是**怎么做**（How）。”

## 埃德加·科德

(E.F. Codd)

关系数据库之父

- **1923** 出生于英国
- **1965** 获得计算机博士学位
- **1970** 发表里程碑论文 《**大型共享数据**库的关系数据模型》
- **1981** ACM 图灵奖

# 半个世纪的数据库系统

（1960年代 – 2020年代）

| 时间 | 事件 |
|---|---|
| **1960年代早期 – 1970年代早期** | 导航数据库帝国 |
| **1970年代中期 – 1980年代中期** | 数据库第一次世界大战 |
| **1980年代中期 – 2000年代早期** | 关系数据库帝国 |
| **2000年代中期 – 至今** | 数据库第二次世界大战 |

时间轴：1960年代至今

**参考资料**

https://en.wikipedia.org/wiki/Database#History

《历史的轮回》（迈克尔·斯通布雷克，乔·赫勒斯坦）

VLDB 40周年座谈会

# 数据库世界大战 I：背景

导航模型 VS 关系模型

## 一方：导航模型

领军人物

**查尔斯·巴赫曼**

1973年图灵奖得主

当时优势

**系统已成熟：**
拥有构建完善的成熟系统（如IDS）

**市场绝对主导：**
几乎垄断了当时的数据库市场

**工业界标准：**
由 CODASYL 制定了官方标准

VS

## 另一方：关系模型

领军人物

**泰德·科德**

IBM 数学程序员

当时劣势

**仅有理论论文：**
尚无任何实际构建的系统

**缺乏支持：**
初期并未获得IBM内部的重视

**思维范式挑战：**
纯数学理论，与当时主流观念冲突

# 数据库世界大战 I：三大战役

## 1. 理论战役

1970年代中期

理论上
哪种数据模型更好？

## 2. 实践战役

1970年代末 – 1980年代初

实践中
哪种数据模型更好？

## 3. 商业战役

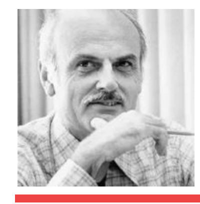
1980年代初 – 1980年代中期

商业上
哪种数据模型更好？

# 1. 理论战役

1974年 ACM SIGFIDET 辩论（SIGMOD 前身）

## 理论战役：哪种数据模型更好？



### 关系模型方

### 导航模型方

**数据组织：结构过于复杂**

数据结构复杂，维护成本高，像迷宫一样。

**仅是特殊情况**

关系模型被认为是导航模型的一种简单、特殊的退化形式。

**数据组织：缺乏声明式语言**

没有高级查询语言，程序员必须编写复杂的路径导航代码。

**声明式语言不可行**

当时没有系统能证明声明式语言在性能上是可行的。

# 2. 实践战役

（1970年代末 – 1980年代初）

**核心问题**

**关系数据**库系统的性能能否与导航数据库相媲美？

**Ingres 项目**

**早期先锋** • **开**创性系统

**System R 项目**

被认为“**做对了**”关键点 • **奠定**现代基础

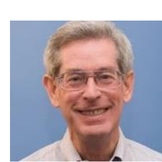
## System R 团队的关键贡献者

**帕特里夏·格里菲斯**

(Patricia Griffiths / Selinger)

**查询优化**

基于成本的优化器



**唐纳德·张伯林**

(Donald Chamberlin)

**SQL 语言**

结构化查询语言发明者

**吉姆·格雷**

(Jim Gray)

**事务处理**

1998年图灵奖得主

# 3. 商业战役

（1980年代早期 – 1980年代中期）

? 为什么关系数据库最终赢了？

### 1. 小型机革命

1977年

小型机（如VAX）的普及创造了全新的市场空间。

### 2. 竞争产品移植困难

技术壁垒

竞争产品（如IDMS）难以移植到新的小型机平台。

### 3. 导航数据库战略失误

战略失误

未能及时向导航数据库系统添加关系型前端。

# 我们学到了什么？

历史的经验与教训

1 好理论不代表好系统

2 好系统不代表好产品

# 半个世纪的数据库系统

（1960年代 – 2020年代）

| 时间 | 事件 |
| --- | --- |
| **1960年代早期 – 1970年代早期** | 导航数据库帝国 |
| **1970年代中期 – 1980年代中期** | 数据库第一次世界大战 |
| **1980年代中期 – 2000年代早期** | 关系数据库帝国 |
| **2000年代中期 – 至今** | 数据库第二次世界大战 |

时间轴：1960年代至今

**参考资料**

https://en.wikipedia.org/wiki/Database#History

《历史的轮回》（迈克尔·斯通布雷克，乔·赫勒斯坦）

VLDB 40周年座谈会

# 关系数据库帝国

（1980年代中期 – 2000年代初）

# 半个世纪的数据库系统

（1960年代 – 2020年代）

| 时间 | 事件 |
| --- | --- |
| **1960年代早期 – 1970年代早期** | 导航数据库帝国 |
| **1970年代中期 – 1980年代中期** | 数据库第一次世界大战 |
| **1980年代中期 – 2000年代早期** | 关系数据库帝国 |
| **2000年代中期 – 至今** | 数据库第二次世界大战 |

时间轴：1960年代至今

**参考资料**

https://en.wikipedia.org/wiki/Database#History

《历史的轮回》（迈克尔·斯通布雷克，乔·赫勒斯坦）

VLDB 40周年座谈会

# 数据库世界大战 II：背景

## 2000年初 · 互联网繁荣 (Internet Boom)

Internet Boom 带来的挑战

### 数据量暴增

→ 单机无法存储

### 更新频繁

→ 单机无法处理

▼

已有系统的困境

### 商用系统昂贵

→ 成本难以承受

### 开源能力不足

→ 不支持分布式

# OLTP

OnLine Transaction Processing

### 主要特点

更新模式
高频更新

查询模式
小型查询

**例如**：电商订单、银行转账

# OLAP

OnLine Analytical Processing

### 主要特点

更新模式
低频更新

查询模式
大型查询

**例如**：商业智能、销售报表

# 数据库世界大战 II：两大战役

第一战役

## OLTP 战场

NoSQL

VS

传统关系型数据库

第二战役

## OLAP 战场

MapReduce

VS

传统关系型数据库

# OLTP 发生了什么？

**1970 – 至今**

## OldSQL

传统关系型数据库

**2000 – 至今**

## NoSQL

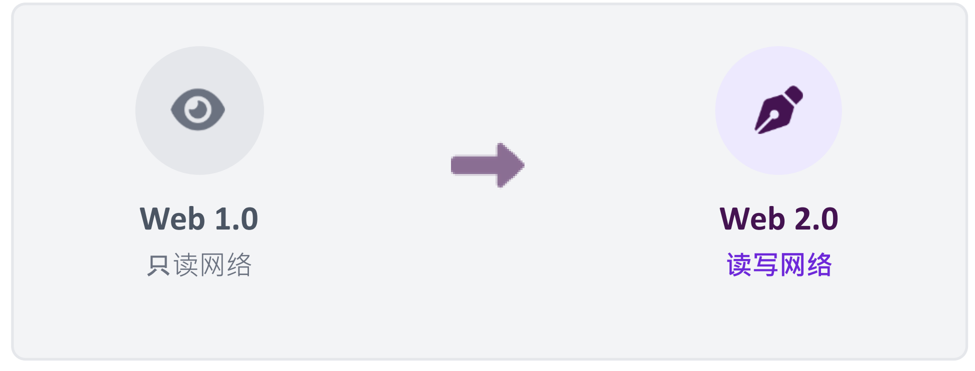
非关系型分布式数据库

**2010 – 至今**

## NewSQL

新一代高性能关系型数据库

# Web 1.0 到 Web 2.0 的变革




典型 Web 2.0 应用

## Web 2.0 应用的核心技术要求

### 1. 高可扩展性

系统需要支撑百万级用户并发和千台服务器规模的集群扩展。

### 2. 高可用性

服务必须保证 7×24 小时全天候不间断运行，即便是故障期间。

### 3. 高灵活性

能够适应灵活多变的模式（Schema）和多样化的非结构化数据类型。

# NoSQL 先驱

分布式存储的早期探索

## Memcached 2004

基于内存的键值缓存，解决数据库读取瓶颈，实现极高的扩展性。

**应用场景：Facebook 会话缓存**

支撑十亿级用户的秒级登录验证，减轻后端数据库的读取压力。

## BigTable 2006

分布式的结构化数据存储系统，支持持久化存储扩展至数千个节点。

**应用场景：Gmail 邮件存储**

管理数十亿邮箱的海量数据，提供跨服务器的可靠存储与快速检索。

## Dynamo 2007

采用最终一致性模型，牺牲强一致性以换取极致的可用性。

**应用场景：亚马逊购物车**

确保“永远可写”，即使在部分网络故障时，用户仍能添加商品到购物车。

# NoSQL 分类

| NoSQL 类型 | 数据模型 | 代表系统 |
| --- | --- | --- |
| 键值存储 | 哈希表 | DynamoDB, Riak, Redis, Membase |
| 文档存储 | 文档对象 | MongoDB, CouchBase, SimpleDB |
| 宽列存储 | 列族 | HBase, Cassandra, HyperTable |
| 图数据库 | 图结构 | Neo4j, InfoGrid, GraphBase |

NoSQL 系统种类繁多（100+），**主要根据其数据模型和**应用场景进行分类。每种类型都在特定的读写模式和扩展性需求上进行了优化。

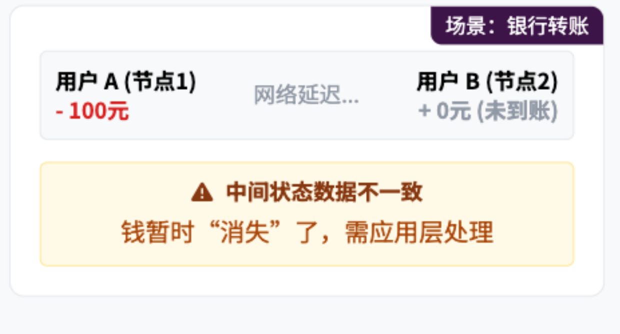
# NoSQL 的局限

## 1. 低层次语言

缺乏声明式查询(SQL)，需编写复杂代码处理逻辑。

## 2. 一致性较弱

采用最终一致性，数据在各节点间可能暂时不同步。




## 3. 缺乏标准化

百余种系统各不相同，学习与迁移成本极高。

# NewSQL

传统架构的终结与重生

里程碑论文 (2008)

The end of an architectural era:(it's time for a complete rewrite)
M Stonebraker, S Madden, DJ Abadi… - Proceedings of the 33rd …, 2007 - dl.acm.org
Abstract In previous papers [SC05, SBC+ 07], some of us predicted the end of" one size fits all" as a commercial relational DBMS paradigm. These papers presented reasons and experimental evidence that showed that the major RDBMS vendors can be outperformed ...
Cited by 580 Related articles All 55 versions Cite Save

## 架构时代的终结

传统数据库在现代硬件上表现不佳，必须**彻底重写**架构，才能适应大内存与多核环境。

## 时间都去哪儿了？

| 90% 系统内部开销 (Overhead) | 10% |
| --- | --- |

↑ 大部分CPU时间浪费在辅助机制上

有效数据处理

### 缓冲区管理

页面置换算法开销

### 锁与并发控制

死锁检测与等待

### 日志与恢复

数据持久化代价

# 架构变革：如何消除三大开销？

针对现代硬件的数据库重新设计

[PDF] OLTP Through the Looking Glass, and What We Found There
S Harizopoulos, M Stonebraker, S Madden, DJ Abadi - 2008 - people.csail.mit.edu
ABSTRACT Online Transaction Processing (OLTP) databases include a suite of features—disk-resident B-trees and heap files, locking-based concurrency control, support for multi ...
Cite

## 核心观点：彻底抛弃旧架构

Stonebraker等人提出：为了适应现代硬件（大内存、多核），必须重新设计数据库引擎。传统架构是为"**慢速磁**盘"设计的，在"**全内存**时代"**反而成了**累赘。

### 1. 消除缓冲区管理

**对策：全内存存储**

**为什么**：内存已经很便宜，不需要再把数据搬来搬去。
**怎么做**：直接在内存中读写数据，就像操作普通变量一样简单。

### 2. 消除锁机制

**对策：单线程串行执行**

**为什么**：内存操作极快（微秒级），加锁解锁反而比做事还慢。
**怎么做**：排好队一个接一个做，避免多线程打架和上下文切换。

### 3. 消除重型日志

**对策：命令日志与复制**

**为什么**：记录每一个字节的变动太浪费时间。
**怎么做**：只记录"**做了什么操作**"（如SQL语句），并通过多机备份保证安全。

# NewSQL 系统概览

代表性 NewSQL 系统

## NewSQL 的优点

**支持 SQL 与 ACID**

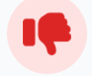
保留了关系数据库的易用性，支持标准 SQL 查询和强一致性事务。

**高性能扩展**

针对现代硬件（多核、大内存）优化，提供比传统数据库更好的水平扩展能力。

**兼容现有生态**

开发人员无需学习全新查询语言，现有工具和应用容易迁移。



## NewSQL 的挑战

**特定场景优化**

通常针对特定工作负载（如内存型OLTP）优化，不适合通用场景。

**生态系统较小**

相比传统数据库，社区支持较弱，工具链不够成熟。

**内存成本高昂**

许多 NewSQL 系统依赖全内存存储，在大数据量下硬件成本极高。

# 总结

不同时代的数据库技术对比

## 传统数据库

OldSQL (1970年代至今)

- ✓ **数据绝对准确**(不丢不错)
- ✓ **通用查询语言**(大家都会写)
- ✗ **很难加机器扩展**(单机瓶颈)
- ✗ **表结构必须固定**(改字段很麻烦)

**为什么选它？**

适合银行转账、订单管理等绝不能出错、结构稳定的核心业务。

## 非关系型数据库

NoSQL (2000年代至今)

- ✓ **轻松横向扩展**(加机器就能抗压)
- ✓ **存储格式灵活**(存什么都行)
- ✓ **始终可用**(系统不挂机)
- ✗ **数据可能暂时不一致**(最终才一致)

**为什么选它？**

适合社交网络点赞、日志分析、商品推荐等海量数据且允许短暂延迟的场景。

## 新一代数据库

NewSQL (2010年代至今)

- ✓ **数据准确 + 易于查询**(兼顾传统优势)
- ✓ **像NoSQL一样扩展**(解决性能瓶颈)
- ℹ **全新架构设计**(去掉了历史包袱)
- ⚠ **生态尚在发展**(不如老牌成熟)

**为什么选它？**

试图“鱼与熊掌兼得”：**既要**传统数据库的可靠便利，**又要**新技术的强力扩展。

# 数据库世界大战 II：两大战役

第一战役

## OLTP 战场

NoSQL

VS

传统关系型数据库

第二战役

## OLAP 战场

MapReduce

VS

传统关系型数据库

# Google 的使命与挑战

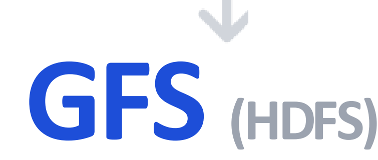
*"Organize the world's information and make it universally accessible and useful."*

整合全球信息，使人人皆可访问并从中受益

**10 PB**

= 1000GB × 10,000 台机器

如何利用 10,000 台机器 **存储** 10PB?




如何利用 10,000 台机器 **处理** 10PB?

# 为什么选择 MapReduce？(1) 容错性

## 电脑坏了怎么办？

如果你的程序运行到一半...

**突然死机了！**

*在大规模集群里，这是家常便饭*

## 为什么必须解决？

| 机器数量 | 死机概率 |
|---|---|
| 1 台 | 0.1% (很稳) |
| 100 台 | 9.5% (有点悬) |
| **10,000 台** | **~100%** |

**肯定会坏！必须有备案！**

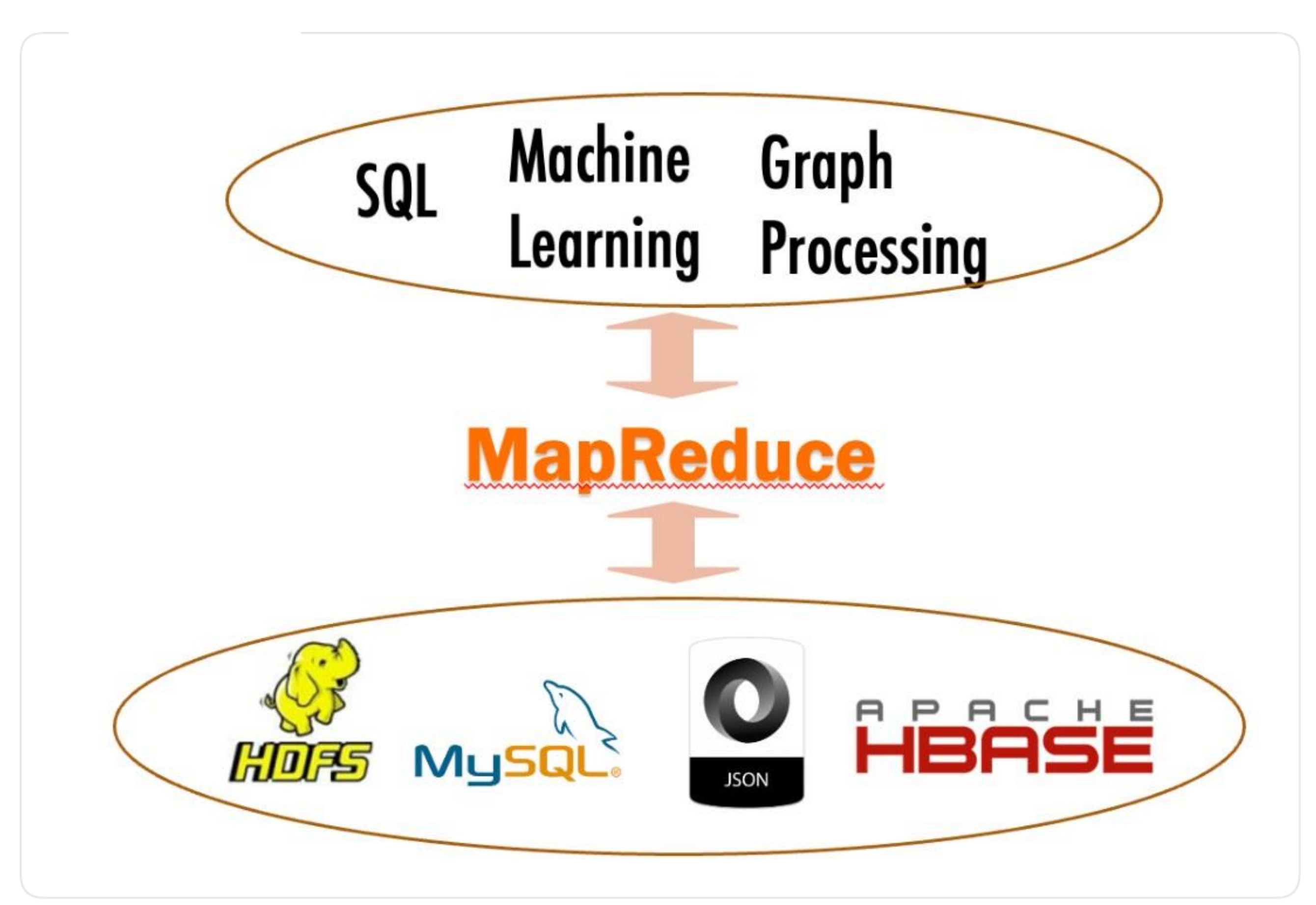
## MapReduce解决方案

1 **保存进度**
把做到一半的结果存硬盘

2 **断点续传**
哪台坏了，换台机器接着干

不用从头再来！

# 为什么选择 MapReduce？(2) 复杂分析+异构存储




MapReduce 连接异构存储与复杂分析应用

# MapReduce大辩论 (2008-2010)

## 学界 vs 产业界

这场辩论始于2008年初，不仅是技术路线之争，更是关于数据处理哲学的碰撞。它深刻影响了后续十年大数据系统的发展方向。

### 数据库学界

**迈克尔·斯通布雷克**
(Michael Stonebraker)

**大卫·德威特**
(David DeWitt)

核心观点

"MapReduce 是一次巨大的倒退"

VS

### 系统产业界

**杰夫·迪恩**
(Jeff Dean)

**桑杰·格玛瓦特**
(Sanjay Ghemawat)

核心观点

"MapReduce 是编程模型而非数据库"

# MapReduce vs. SQL

```
SELECT Map(v)
FROM Table
```

```
Reduce (k, v_list)
```

```
SELECT Reduce(v)
FROM Table
GROUP BY k
```

# 数据库学界的五大批评

MapReduce：一次重大的倒退

## 来自数据库学界的五点严厉批评

**1. 走回头路**
放着好用的SQL不用，又回到写复杂代码的时代

**2. 性能太差**
比专业数据库慢很多（没有优化器，全是蛮力计算）

**3. 不是新东西**
40年前就有类似想法，只是重新包装了一下

**4. 功能太少**
缺少索引、事务等数据库的基本功能

**5. 不兼容现有工具**
无法与现有的报表、分析等数据库工具配合使用

### 批评来源

*博客文章：《MapReduce：一次重大的倒退》*

作者

**大卫·德威特**
**& 迈克尔·斯通布雷克**

发表日期

2008年1月17日

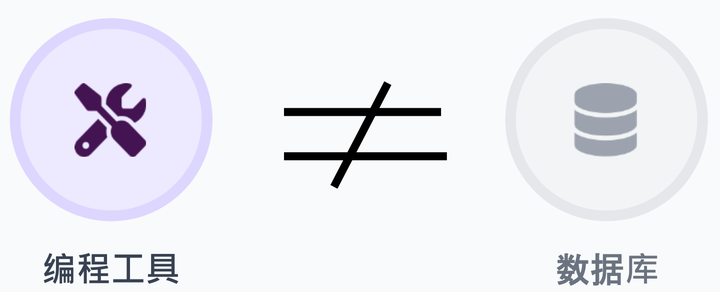
技术博客

# 产业界的回应

"你们搞错了！"

**MapReduce 不是数据库，**
**它只是一个编程工具！**

MapReduce **的**设计初衷是为了解决大规模数据的计算问题，
**从来就不是**为了替代传统的数据库系统。



COMMUNICATIONS OF THE ACM

CACM.ACM.ORG 01/2010 VOL.53 NO.01

From Stonebraker et al.

From Dean and Ghemawat

MapReduce complements DBMSs since databases are not designed for extract-transform-load tasks, a MapReduce specialty.

BY MICHAEL STONEBRAKER, DANIEL ABADI, DAVID J. DEWITT, SAM MADDEN, ERIK PAULSON, ANDREW PAVLO, AND ALEXANDER RASIN

# MapReduce and Parallel DBMSs: Friends or Foes?

MapReduce advantages over parallel databases include storage-system independence and fine-grain fault tolerance for large jobs.

BY JEFFREY DEAN AND SANJAY GHEMAWAT

# MapReduce: A Flexible Data Processing Tool

# 他们的共识

## MapReduce 的优势

**容错性**

机器坏了能继续跑，不用从头重来。

**支持复杂分析**

不只是简单查询，还能跑机器学习算法。

**支持多种数据源**

不用把数据导入数据库，Excel、日志都能直接算。

**无需数据加载**

省去漫长的导入时间，拿来就能用。

**核心共识**

**双方应该
相互学习
取长补短**

# 谁赢了这场辩论?

现在几乎**没有人**
在写 MapReduce 代码了。

许多新系统（如 Spark、HIVE）
都构建在 **MapReduce** 之上

# 趋势

### 趋势一：数据科学

通过数据处理、可视化与统计建模，从数据中发现规律、形成可信结论

### 趋势二：数据工程

通过 SQL、查询优化与数据建模，将数据处理流程从实验环境推进到生产级系统

### 趋势三：数据 + AI

数据库技术与人工智能的深度融合：AI4Data (AI 如何赋能数据库) 和 Data4AI（数据库支撑AI）

# 总结

数据库发展史上的三个核心问题：

### 为什么选择<br>关系数据库？

1970年代中期 – 至今

数据**独立性**与**声明式查询**的胜利

### 为什么选择<br>NoSQL？

2000年代中期 – 至今

应对 **Web 2.0** 的**高扩展性**与灵活性挑战

### 为什么选择<br>MapReduce？

2000年代中期 – 至今

大数据时代的**容错计算**与**复杂分析**