开发成功数据库应用的要点

- 需要理解数据库体系结构
- 需要理解锁和并发控制特性
  - 每个数据库都以不同的方式实现
- 不要把数据库当“黑盒”
- 性能、安全性都是适当的被设计出来的
- 用尽可能简单的方法解决问题
  - “创造”永远追不上开发的步伐
- DBA和RD之间的关系

数据库体系结构的差异

- **不能把数据库当成“黑盒”使用，因为每个数据库都是非常不同的**
- Oracle和MySQL的差别，类似
  - Windows和Linux的差别
  - iOS和Android的差别
  - 虽然都是DBMS，但它们也有相当的差异
- 了解这种差异，了解你所使用数据库的特性，是开发成功数据库应用的基础

并发控制的问题

- 现实存在并发，我们需要保持数据的一致性，所以要做并发控制
- 锁机制，使得并发控制成为可能
- **不同的数据库，实现锁机制是不一样的**

You

不同数据库的锁机制不同，做一张表达这个含义的图

< 2 / 2 >

ChatGPT

Here's the illustration showing the variety of lock mechanisms used by different database systems, with each lock type symbolizing the unique approach to ensuring data integrity and managing concurrency. If there's anything more you'd like to adjust or another concept you want to explore, please let me know!

比如：Oracle的锁机制就比较特别

- 这个时候，Transaction1能提交吗？

时序	Transaction 1	Transaction 2
T1	Begin	
T2	Read（X）x=1000	Begin
T3		Read（X）x=1000
T4		Write（X）x=x-800=200
T5		Submit
T6	**Submit**	

Oracle存在有时读不到正确数据的现象

- Oracle的多版本控制，读一致性的并发模型
  - 读一致查询：对于一个时间点（point time），查询会产生一致的结果
  - 非阻塞查询：查询不会被写入器阻塞，但在其它数据库中可能不是这样的

有一个最简单的方式演示Oracle中的多版本：

```
Create table t
As
Select * from all_users
/

Variable x refcursor
Begin
Open :x for select * from t;
End;
/
```

```
Declare

Begin

Delete from t;

Commit;

End;

/
```

Oracle这种锁机制的好处是什么？

- *例子：ACCOUNTS（account_number, account_balance）一个银行的账户余额*

  为了简单，只考虑一个仅有四行的表（同时假设每个数据库块中只存放一行数据。）

  帐号，余额

  1；500

  2；250

  3；400

  4；100

- 我们想运行一个日报表，了解银行里有多少钱。下面是一个非常简单的查询。
  - Select sum(account_balance) from accounts;

Oracle/MySQL和其它数据库在并发上的差别

帐号，余额
1；500
2；250
3；400
4；100

Oracle的做法

	R&W	W
T1	Read（1）	
T2	Read（2）	Write（1）500-400
T3	Read（3）	Write（4）100+400
T4	**Read（4）**	commit
T5	Sum（1+2+3+4）	
T6	**Commit**	

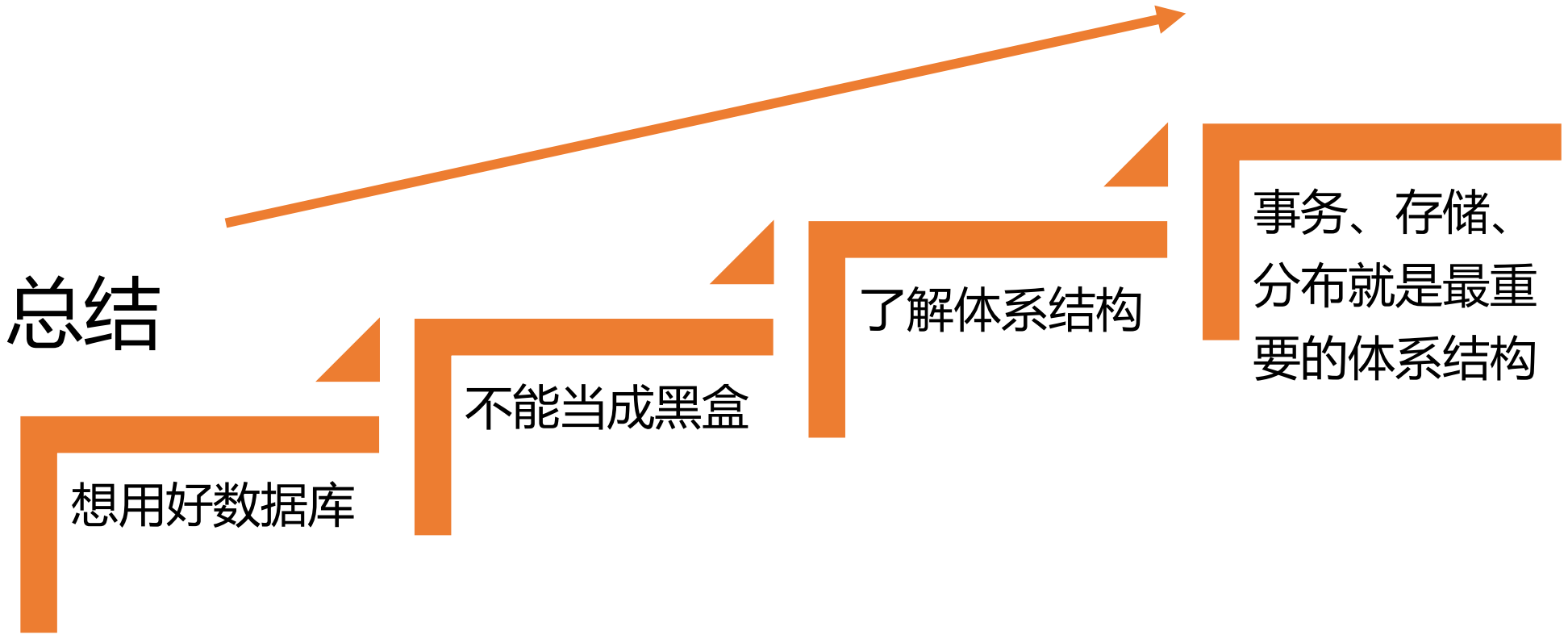
其它数据库的做法

	R&W	W
T0	**Lock（T）**	
T1	Read（1）	
T2	Read（2）	Write（1）500-400
T3	Read（3）	Write（4）100+400
T4	**Read（4）**	Commit **（Waiting and Roll Back）**
T5	Sum（1+2+3+4）	
T6	**Commit**	Roll Back

从体系结构和特性中了解具体数据库的锁机制

- 比如Oracle实现的锁机制
  - 只有修改才加行级锁
  - Read绝对不会对数据加锁
  - Writer不会阻塞Reader
  - 读写器绝对不会阻塞写入器

总结



Practice in class 2-1

- 对大多数码农而言，数据库锁机制好像都是自动和透明实现的，那么深入了解每个数据库的锁机制实现细节，对码农编码有什么影响嘛？
- 根据这节课Oracle的锁机制特征的分析，你尝试去了解一下MySQL、或者其他数据库，他们这些数据库锁机制实现的特征

开发成功数据库应用的要点-黑盒的问题

- 对大多数码农而言，数据库锁机制好像都是自动和透明实现的，那么深入了解每个数据库的锁机制实现细节，对码农编码有什么影响嘛？

一个技术的盔甲就是它的软肋

- Oracle的无阻塞设计有一个副作用，就是如果确实想保证一次最多只有一个用户访问一行数据，就得开发人员自己做一些工作

例如：一个资源调度程序主要有两张表：

Resources（Resource_name，other_data）

Schedules(resource_name, start_time, end_time)

往Schedules中插入一个房间预订之后，提交之前，应用将查询

```
Select count(*)
From schedules
Where resource_name = :resource_name   FOR UPDATE
        and (start_time < :new_end_time) and (end_time > :new_start_time)
```

Test:

A：(701, 10:00, 12:00)

B: (701, 11:00, 12:00)

不能把数据库当成黑盒使用

必须深入了解你所使用的数据库的体系结构和特征

什么时候，码农需要自己考虑并发的问题？

不知道，看情况

黑盒和数据库独立性的问题

- 黑盒——脱离实现级别的使用方法
- 我的观点是
  - 要构建一个完全数据库独立的应用，而且是高度可扩展的应用是极其困难的
  - 实际上，这几乎是不可能的
- 要构建一个完全数据库独立的应用
  - 你必须真正了解每个数据库具体如何工作
  - 如果你清楚每个数据库工作的具体细节，你就会知道，数据库独立性可能并不是你真正想要的

例如：Null值造成的数据库迁移障碍

例子：在表T中，如果不满足某个条件，则找出X为NULL的所有行，如果满足就找出X等于某个特定值的所有行。

```
Declare
  L_some_variable varchar2(25)
Begin
  If(some_condition)
  Then
    L_some_variable := f(…);
End if;
For x in (select * from t where x=l_some_variable)
Loop
…
```

*Select * from t where(x = l_some_variable OR(x is null and l_some_variable is NULL)*

select * from t where nvl(x,-1) = nvl(l_some_variable,-1)

创建一个基于函数的索引：create index t_idx on t(nvl(x,-1))

关于黑盒的问题总结几点

- 数据库是不同的。在一个数据库上取得的经验也许可以部分应用于另一个数据库，但是必须有心理准备，二者之间可能存在一些基本差别，可能还有一些细微的差别。
- 细微的差别（比如对NULL的处理）与基本差别（如并发控制机制）可能有同样显著的影响。
- 了解数据库，知道它是如何工作的，他的特性如何实现，这是解决这些问题的唯一途径。

我能不能找一个大牛帮我调优

- 先把程序写出来，之后再让专家在生产环境中帮我调优
  - 这个想法是错误的.....
- 性能调优（目前情况下性能优化至最优）
  - 根据当前CPU能力、可用内存、I/O子系统等资源情况来设置相应参数
  - 通过索引、物理结构、SQL的优化，具体提高某一个查询的性能

如果有个专家能通过一些参数、技巧提高了你的系统一个数量级的性能，

不能说这个专家牛逼，大概只能说明你的程序太烂了。

性能拙劣的罪魁祸首是错误的设计

- 提高整体性能
  - 技巧决定系统性能的下限
  - 设计决定系统性能的上限
- 比如，新闻的门户网站
  - 动态页面vs静态页面
  - 静态页面+内容管理系统

性能优化要考虑整体

- 性能指标都是有成本的、安全和优化中寻找平衡
- 性能指标是什么？（一个事务多少秒？还是……？）
- 性能指标要考虑整体性
  - 优化手段本身就有很大的风险，只不过你没意识到罢了
  - 任何一个技术可以解决一个问题，但必然存在另一个问题的风险
  - 对于带来的风险，控制在可接受的范围才是有成果
  - 性能优化技术，使得性能变好，维持和变差是等概率的事件

使用优化工具

MySQL常用的工具
MySQLadmin
MySQLshow
SHOW [SESSION | GLOBAL] variables
SHOW [SESSION | GLOBAL] STATUS
Information_schema
SHOW ENGIN INNNODB STATUS
SHOW PROCESSLIST
Explain
Show index
Show log

MySQL不常用，但是好用的工具	
zabbbix	监控主机、系统、数据库
Pt-query-digest	分析慢日志
MySQLslap	分析慢日志
sysbench	压力测试工具
MySQL profiling	统计数据库整体状态工具
Performance Schema	性能状态统计的数据
Workbench	管理、备份、监控、分析、优化工具

整体层面的性能优化考虑

- 问题一：CPU负载高，IO负载低
  - 内存不够
  - 磁盘性能差（磁盘问题、raid设计不好、raid降级）
  - **SQL的问题**
  - **并发锁机制的问题**
  - **事务设计问题，大量小数据IO**
  - **大量的全表扫描**

整体层面的性能优化考虑

- 问题二：IO负载高，CPU负载低
  - **大量小的IO执行写操作**
  - **Autocommit，产生大量小IO**
  - **大量大的IO执行写操作**
  - **SQL的问题**
  - IO/PS磁盘限定一个每秒最大IO次数

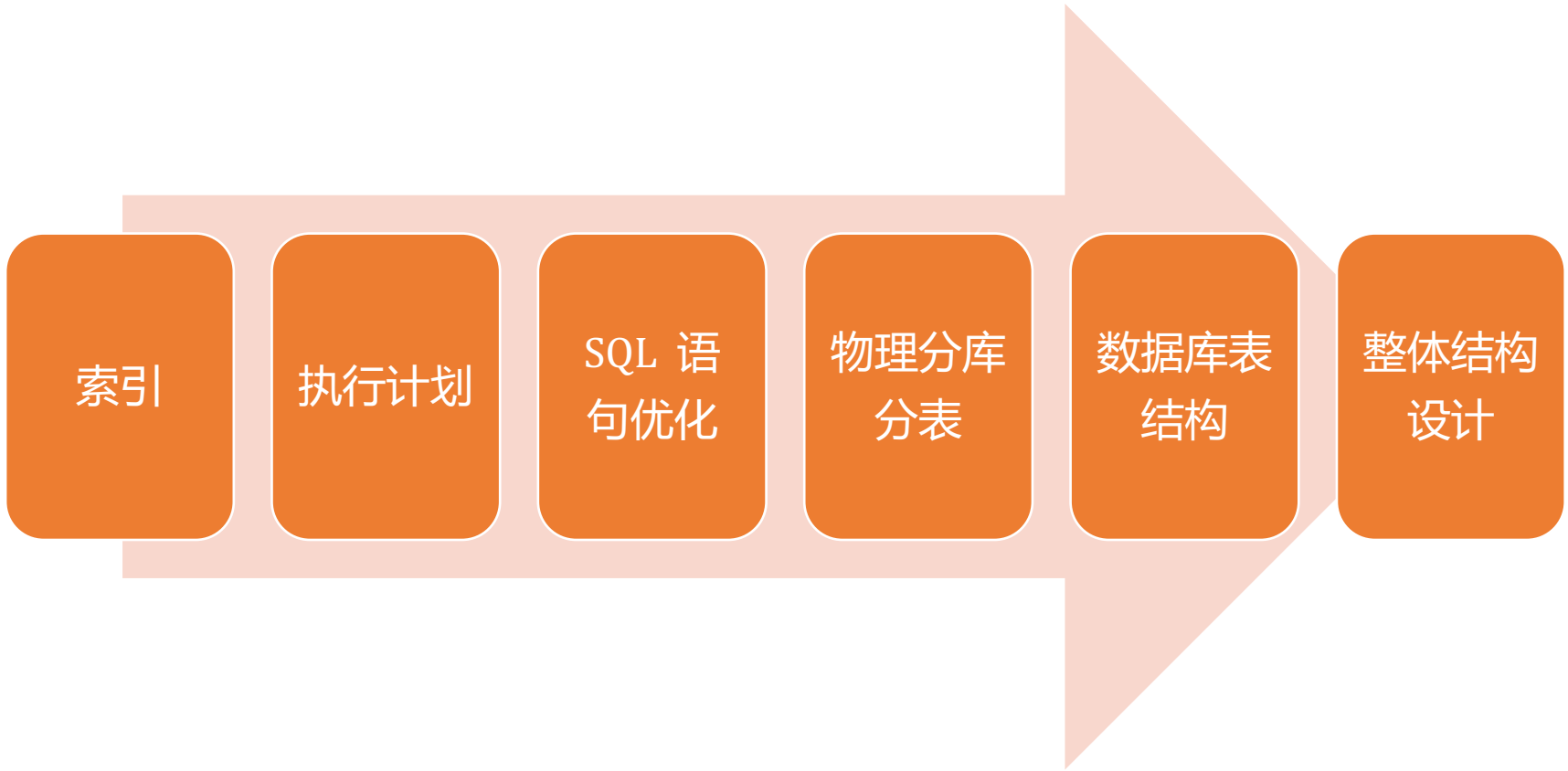
整体层面的性能优化考虑

- 问题三：IO和CPU负载都高
  - 硬件不够用了
  - **SQL存在问题**

性能问题，90%的问题来源都是程序员的问题

开发环境到生产环境是一场灾难

SQL优化的方向



差异：限用Boolean型字段

- SQL中并不存在Boolean类型
- 实现flag表示标志位的Y/N或T/F
  - 例如：order_completed
  - 但是...往往增加信息字段能包含更多的信息量
  - 例如：completion_date completion_by
  - 或者增加order更多状态标示
- 极端的例子：四个属性取值都是T/F，可以用0-15这16个数值代表四个属性所有组合状态
  - 技巧可能违反了原子性的原则
  - 为数据而数据，是通向灾难之路

和编程的差异在，SQL 绝大多数时候，不需要“特别的思考和技巧”

差异：过于灵活的危险性

- “真理向前跨一步就是谬误”
- 不可思议的四通用表设计
  - Objects(oid, name), Attributes(attrid, attrname,type)
  - Object_Attributes(oid,attrid,value)
  - Link(oid1,oid2)
- 随意增加属性，避免NULL
- 成本急剧上升，性能令人失望

差异：约束应明确说明

- 数据中存在隐含约束是一种不良设计
- 字段的性质随着环境变化而变化时设计的错误和不稳定性
- 数据语义属于DBMS，别放到应用程序中

设计：理解子类型（SubType）

- 表过“宽”（有太多属性）的另一个原因，是对数据项之间的关系了解不够深入
- 一般情况下，给子类型表指定完全独立于父表主键的主键，是极其错误的

设计：如何处理历史数据

- 历史数据：例如：商品在某一时刻的价格
- Price_history
  - (article_id，effective_from_date，price)
- 缺点在于查询当前价格比较笨拙
- 其他方案
  - 定义终止时间
  - 同时保持价格生效和失效日期，或生效日期和有效天数等等
  - 当前价格表+历史价格表

架构：处理流程

- 操作模式（operating mode）
  - 异步模式处理（批处理）
  - 同步模式处理（实时交易）
- 处理数据的方式会影响我们物理结构的设计

架构：数据集中化（Centralizing）

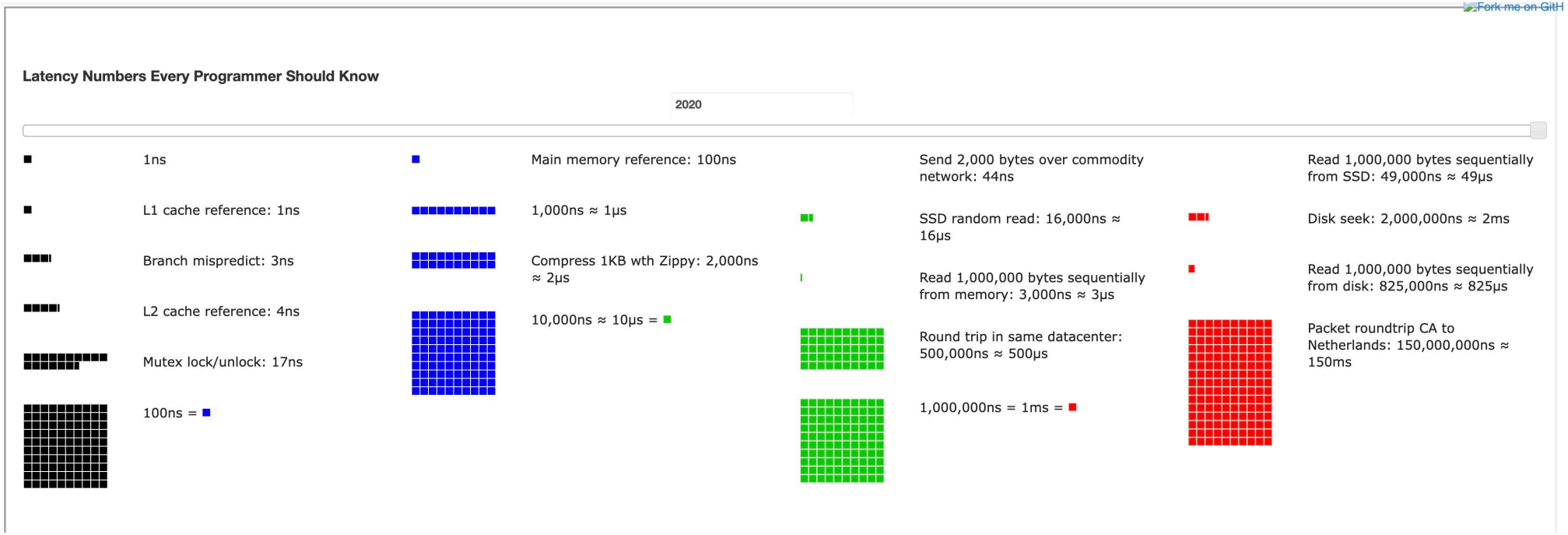
- 分布式数据系统复杂性大大增加
  - 远程数据的透明引用访问代价很高
  - 不同数据源数据结合极为困难
    - Copy的数据传输开销
    - 无法从数据规划中获益（物理结构，索引）
- 数据库该如何部署呢？
  - 中庸、分析、决策
- 离数据越近，访问速度越快

架构：系统复杂性

- 数据库的错误很多
  - 硬件故障
  - 错误操作...
- 数据恢复往往是RD和DBA争论焦点
  - DBA，即便确保数据库本身工作正常，依然无法了解数据是否正确
  - RD，在数据库恢复后进行所有的功能性的检查

存储：内存数据库和磁盘数据库

- 内存访问比磁盘访问快几个数量级
  - https://github.com/colin-scott/interactive_latencies Published at https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html



存储：不同的结构，不同的选择

- 内存数据库与磁盘数据库
  - 各自有啥优势或者特征？
- 面向行和面向列的数据库
  - MySQL 、 PostgreSQL
  - Apache Parquet 、 Apache ORC 、 RCFile（面向列的文件格式）
  - Apache Kudu 、 ClickHouse（面向列的存储）
- 宽列式存储（ BigTable 或 Hbase ）

你考虑的因素除了价格、持久化、备份机制、数据库实现的数据结构还有什么？科技发展（非易失性存储器 NVM [ARULRAJ17]）

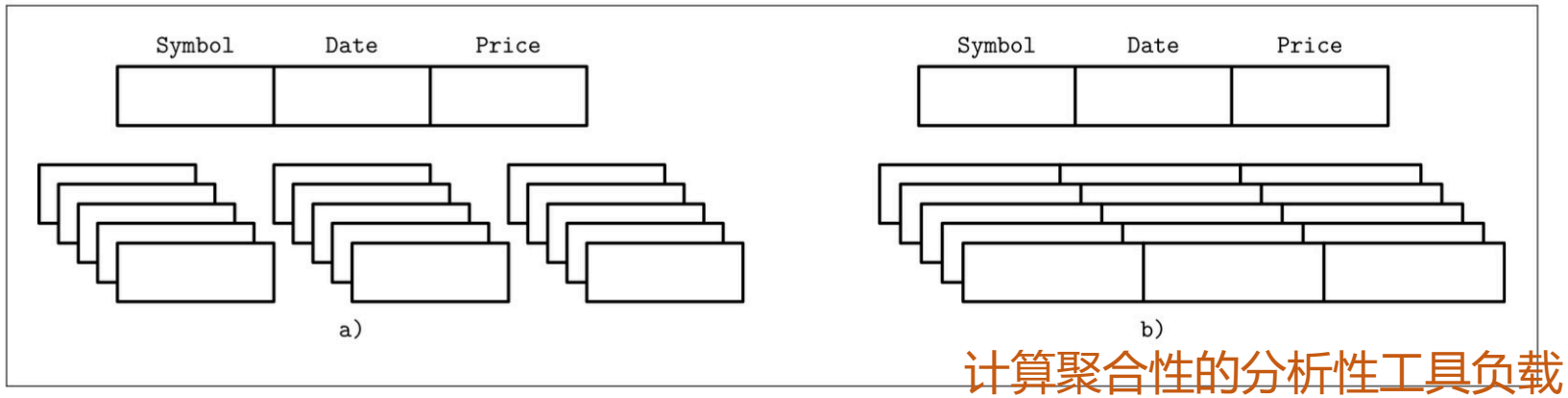
分析的逻辑——存储 - 读取 - 计算（聚合的分析型工作负载）

存储：面向列与面向行的数据库

和访问相关，不仅仅是需求，同时还有效率

整行存储可以提高空间局部性[DENNING68]

- 存储结构的选择，到底和什么相关 ？
- 磁盘之类的持久性介质上的数据通常是按块（block）访问的
  - 换句话说，磁盘访问的最小单位是块
- 对访问整个用户数据有利，不利于某个字段查询



计算聚合性的分析性工具负载（趋势、平均值等）

ID	Symbol	Date	Price
1	DOW	08 Aug 2018	24,314.65
2	DOW	09 Aug 2018	24,136.16
3	S&P	08 Aug 2018	2,414.45
4	S&P	09 Aug 2018	2,232.32

```
Symbol: 1:DOW; 2:DOW; 3:S&P; 4:S&P
Date:   1:08 Aug 2018; 2:09 Aug 2018; 3:08 Aug 2018; 4:09 Aug 2018
Price:  1:24,314.65; 2:24,136.16; 3:2,414.45; 4:2,232.32
```

存储：数据布局只是针对应用可能的优化步骤之一

- 存储效率上，行存储和列存储的差异？ 很多时候，没有绝对的答案，要理解这里的复杂
- 压缩效率上，这里有明显的差异，相同数据类型可以用到同一类压缩算法
- 从同一列中读取多个值可以显著提高缓存利用率和计算效率
  - 向量化指令可以使单条 CPU 指令一次处理多个数据点[DREPPER07]

具体来说：访问模式、工作负载、计算聚合

存储：宽列式存储（如 BigTable 或 Hbase）

- 数据表示为多维映射，列被分组为列族（通常存储相同类型的数据）
- 在每个列族上，数据被逐行存储
- 适合啥？存储由一个键或者一组键来检索的数据

BigTable 的论文[CHANG06] 的 WebTable

```
{
    "com.cnn.www": {
        contents: {
            t6: html: "<html>..."
            t5: html: "<html>..."
            t3: html: "<html>..."
        }
        anchor: {
            t9: cnnsi.com: "CNN"
            t8: my.look.ca: "CNN.com"
        }
    }
    "com.example.www": {
        contents: {
            t5: html: "<html>..."
        }
        anchor: {}
    }
}
```

Column Family: contents

Row Key	Timestamp	Qualifier	Value
com.cnn.www	t3	html	"<html>..."
com.cnn.www	t5	html	"<html>..."
com.cnn.www	t6	html	"<html>..."
com.example.www	t5	html	"<html>..."

Column Family: anchor

Row Key	Timestamp	Qualifier	Value
com.cnn.www	t8	cnnsi.com	"CNN"
com.cnn.www	t5	my.look.ca	"CNN.com"

存储：数据文件和索引文件

- 数据也是文件存储的，为什么不是文件系统？ 不依赖于目录和文件的文件系统层次结构来定位数据
  - 存储效率（文件已最小化单个存储数据记录开销的方式进行存储）
  - 访问效率（用尽可能少的步骤来定位记录）
  - 更新效率（记录更新可以以某种特定方式执行，以使磁盘更改次数最小化）
- 数据文件（Primary File）
  - Iot，heap，hash
- 索引文件（Index File）

存储：缓冲、不可变性和有序性

- 存储结构基于数据结构
- 问题在于，数据结构并不描述缓存、恢复、事务性和存储引擎的其他语义
- 存储结构的三大变量
  - 是否使用缓存——B 树节点的内存缓冲区；双组件 LSM 树和 B 树之间不同的缓冲形式
  - 可变文件还是不可变文件——同一位置是否可以更新，还是只能附加
  - 是否按顺序存储——有序性决定了扫描的范围，是写入优化还是读取优化

更多概念的阅读

- *Database architecture*
  - Hellerstein, Joseph M., Michael Stonebraker, and James Hamilton. 2007. “Archi- tecture of a Database System.” *Foundations and Trends in Databases* 1, no. 2 (Feb- ruary): 141-259. *https://doi.org/10.1561/1900000002*.
- *Column-oriented DBMS*
  - Abadi, Daniel, Peter Boncz, Stavros Harizopoulos, Stratos Idreaos, and Samuel Madden. 2013. *The Design and Implementation of Modern Column-Oriented Database Systems*. Hanover, MA: Now Publishers Inc.
- *In-memory DBMS*
  - Faerber, Frans, Alfons Kemper, and Per-Åke Alfons. 2017. *Main Memory Data- base Systems*. Hanover, MA: Now Publishers Inc.

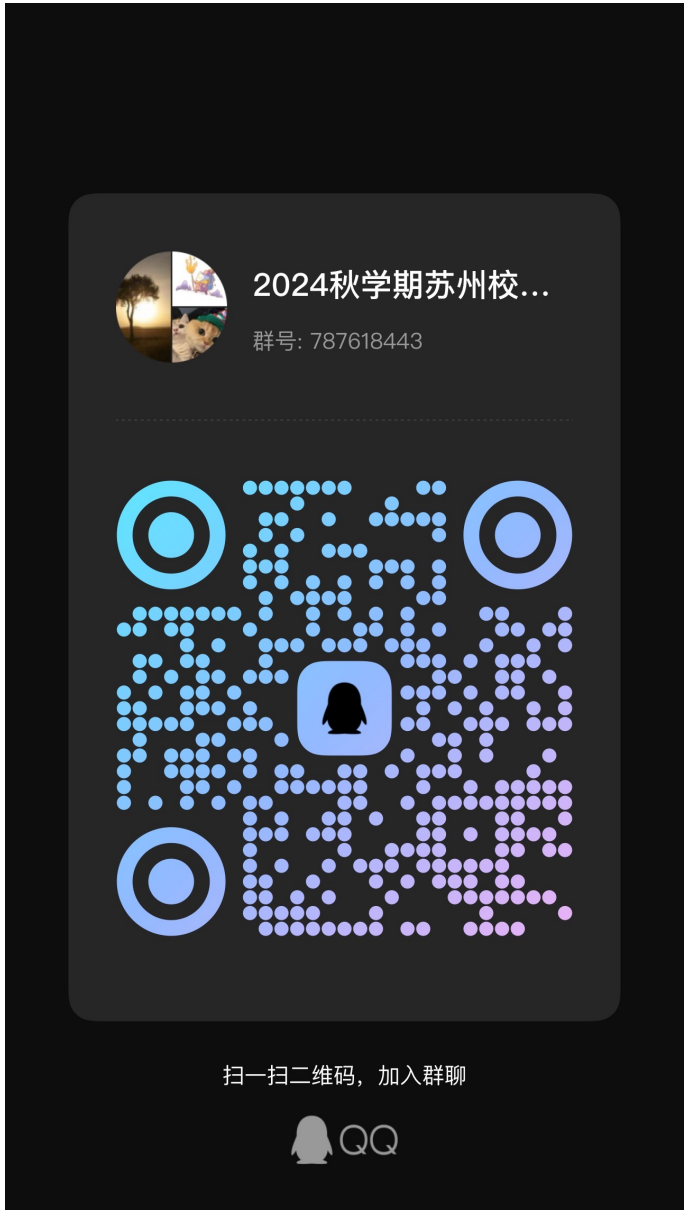
Practice in class 2-2

- 你对你常用的关系数据库关系系统中，去寻找一些针对优化的工具，去尝试使用一些性能的分析和监控工具（查看数据库官方Reference，首先使用官方的命令和工具）

Practice in class 2-3

- 关于把数据库当成黑盒使用的错误，其实也会在你学习软件开发中遇到类似的问题，比如，对操作系统的黑盒化，比如对某些开发框架的黑盒化等等，请你思考一下，你的学习过程中，还能找到类似的例子嘛？

课程群二维码



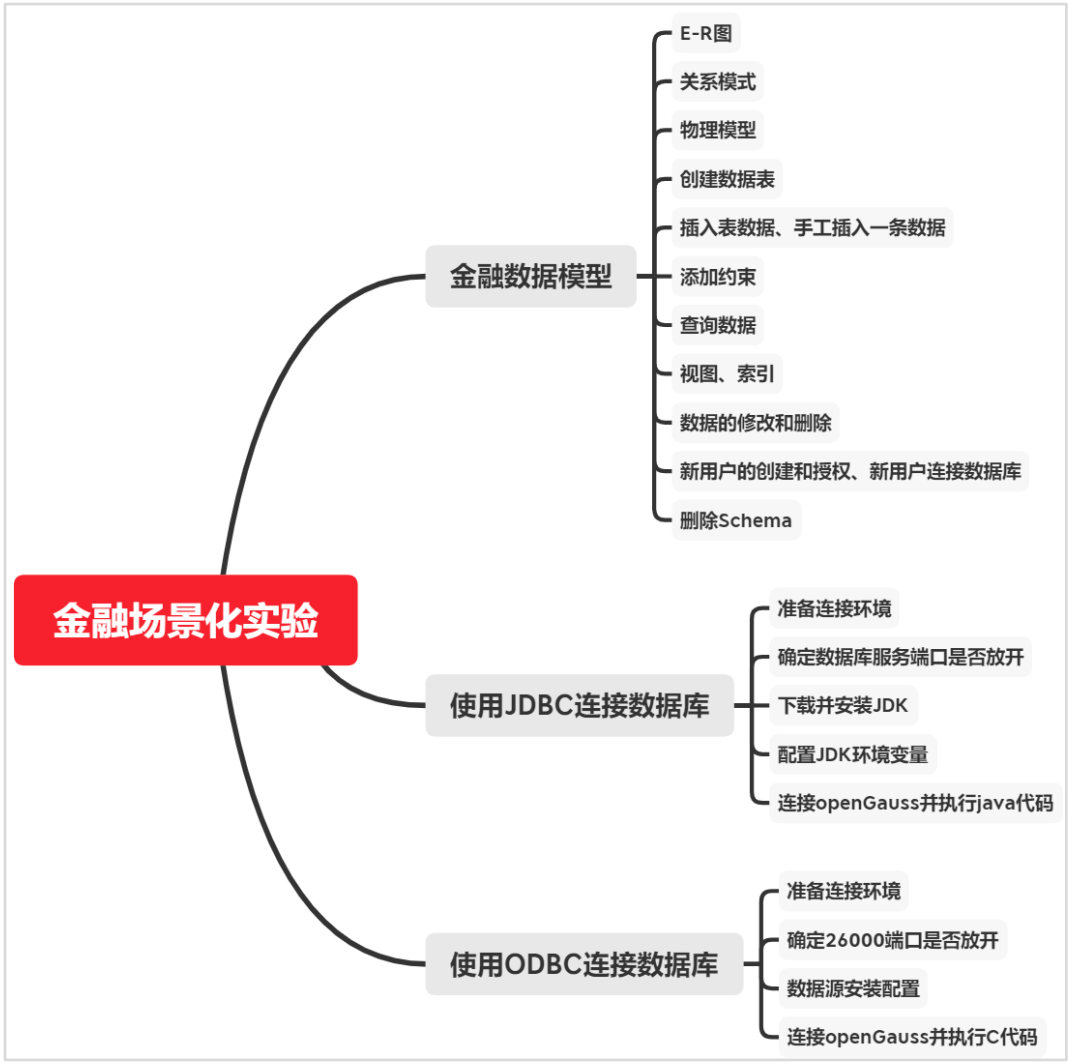
1. 如有问题欢迎在群内提问和讨论
2. 助教们会尽力回复同学们的提问
3. 如果不愿意在群内发消息也欢迎通过邮件zheyuanlin@smail.nju.edu.cn联系

openGauss课程实验

实验要求：

1. 学习openGauss数据库基本操作及OS基本操作。
2. 根据《指导手册》购买ECS弹性云服务器，安装openEuler操作系统构建openGauss数据库部署环境。
3. 根据《openGauss场景化综合应用实验》完成openGauss数据库的编译和安装，并完成金融场景化实验，体验应用过程。
4. 不会很难，学习为主。

时间： 国庆节节后(10.8)发布资源申请链接和具体任务细节。



SQL在线练习和考试

练习系统：

1. 请在校园网或校园网vpn 环境下，访问SQL Platform 网址:http://172.29.4.28。
2. 注册并自由练习。
3. 练习题目不提供统一的标准答案，请同学们独立解决。如有需求可以与助教进行探讨。
4. 题目可能有bug，会热修(请随时发邮件拷打助教谢谢)，不算分，请大家放心。

考试系统：

1. 请在校园网或校园网vpn 环境下，访问SQL Exam http://172.29.4.69/
2. 考前会发布测试考试，考试系统和练习系统有一些不一样。
3. 11月中旬进行考试，考试内容为SQL的编程题，难度不会超过SQL Platform。

End

下一节课再见