ACID

A:原子性
C:一致性
I:隔离性
D:耐久性

范式

第一范式

二维数据表，是第一范式才是数据库

第二范式

第一范式+主键+其他列完全依赖主键

第三范式

第二范式+属性不依赖于非主键(不存在非主属性的传递函数依赖)

BC范式

第三范式+不存在任何字段对任一候选关键字段的传递函数依赖

DDBMS

查询引擎
事务管理
数据集成/多数据
复制/并行

DBMS优点

独立：与上层应用分离
缓存管理：性能优势（即使有数据丢失也可以通过日志恢复）
ACID
辅助函数

DDBMS优点

数据独立性：
高可靠/高可用：可靠指的是出错少，可用指的是出错后可以尽快恢复
高性能：
可伸缩：

DDBMS缺点

没有标准
复杂
管理困难
安全性

DDBMS Architectures

ANSI/SPARC Architecture

用户-外模式-概念模式-内模式

C/S DDBMS

SQL接口、程序接口、缓存-网络-目录、查询分解
客户端-应用服务器-数据服务器-数据库

P2P DDBMS

外模式-GCS(Global Conceptual Schema)-LCS(Local Conceptual Schema)-LIS(Local Internal Conceptual)

MDBMS Architectures

MDBMS

模式、内模式与外模式

模式

定义：也称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。

一个数据库只有一个模式
是数据库数据在逻辑级上的视图
数据库模式以某一种数据模型为基础
定义模式时不仅要定义数据的逻辑结构（如数据记录由哪些数据项构成，数据项的名字、类型、取值范围等），而且要定义与数据有关的安全性、完整性要求，定义这些数据之间的联系

内模式

定义：也称子模式（Subschema）或用户模式，是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。
一个数据库可以有多个外模式
外模式就是用户视图
外模式是保证数据安全性的一个有力措施

外模式

定义：也称存储模式（Storage Schema），它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式（例如，记录的存储方式是顺序存储、按照B树结构存储还是按hash方法存储；索引按照什么方式组织；数据是否压缩存储，是否加密；数据的存储记录结构有何规定）。
一个数据库只有一个内模式
一个表可能由多个文件组成，如：数据文件、索引文件
它是数据库管理系统(DBMS)对数据库中数据进行有效组织和管理的方法，可以减少数据冗余，实现数据共享，还可以减少数据冗余，实现数据共享

自顶向下设计(Top-down Design)

数据分解

水平分解：表结构不变，数据分成不同数据集
垂直分解：分解成多个子表
优点：
查询快
缺点：
性能的影响
完整性保证
分解粒度大小
更新慢

如何确保正确性

完备性
不相交性
可重构性

生命周期

从数据到大数据

大是相对的，和数据量，难度，处理时间等相关
无政府主义抬头
价值密度稀疏
OLTP在线事务→OLAP在线分析→BI商务智能
大致流程：获取数据→抽取清洗→集成聚合→分析建模→解释展示

大数据技术体系

分表、分区、分片与分库

Allocation Alternatives

优势：访问快，可靠性高
劣势：更新慢，易出错，必须同时更新所有

为什么分割

查询效率
可靠性和可用性
安全性

水平分割(Horizontal Fragmentation)

根据特定Property进行分割
用于分割的简单谓词集合P_r应当是最小而且完备的// EX:$\sigma_{BUDGET<1000}(PROJ)$
如何获取最小完备的简单谓词集合
主要是把重复的给消去，比如$B < 100$和$B \geq 100$

Primary Horizontal Fragmentation

$R_i = \sigma_{F_i}(R), 1 \leq i \leq w$
其中，F_i应当是最小项谓词

Derived Horizontal Fragmentation

输入:属从关系的划分集合、成员关系、属主和成员之间的半连接谓词

垂直分割(Vertical Fragmentation)

需要复制主键，按属性切割

信息需求

$Q = \{q_1, q_2, ..., q_q\}$是访问关系$R(A_1, A_2, ..., A_n)$的用户查询(应用)
属性使用值:

$$use(q_i, A_j) = \begin{cases} 1 \ if \ attributes \ A_j \ referenced \ by \ query \ q_i \\ 0 \ otherwise \end{cases}$$

简而言之，如果A_j使用到了q_i(比如select中使用到)，那么$use(q_i, A_j) = 1$
下面定义亲和度矩阵(Attribute Affinity Matrix, AA)

$$aff(A_i, A_j) = \sum_{k|use(q_k,A_i)=1 \wedge use(q_k,A_j)=1} \sum_{\forall S_l} ref_l(q_k) * acc_l(q_k)$$

其中$ref_l(q_k)$表示站点S_l中执行q_k时访问属性(A_i, A_j)的次数
$acc_l(q_k)$表示应用的访问频率度量

聚类算法

目标:调整属性排列顺序，使得属性亲和度值大小相近的在一起，使得全局亲和度度量(Global Affinity Measure,AM)最大
算法:BEA算法-$O(n^2)$

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]$$

其中

$$aff(A_0, A_j) = aff(A_{n+1}, A_j) = aff(A_i, A_0) = aff(A_i, A_{n+1}) = 0$$

简单来说，找到那些同时使用了A_i, A_j的q，把他们的应用频率乘以权重相加
考虑到属性亲和度矩阵的对称性($AA = AA^T$)

$$AM = \sum_{i=1}^{n}\sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$$

算法步骤:
1.初始化:任意选择一列，放入CA
2.迭代:注意选取剩下的列，把它分别插入CA的不同位置，找到使得AM最大的位置，进行插入
3.排序:没有列剩余时，调整相应行列的顺序(遇到相同值随机插入)

划分算法

目标：在聚类后找到合适的划分点

混合切割(Mixed Fragmentation)

综合上面二者进行切割

如何确保分割的正确性

完备性：每个item必须至少属于一张子表
不相交性：一个数据不能属于两个分割后的表
重构性：可以重构出原表

Data Fragmentation Design

简单谓词(Simple Predicate)

$P_j : A_i\Theta\ value\ where\ \Theta \in \{=, <, \leq, >, \geq, \neq\}$ Ex:NAME $\neq$ "MAIN"

最小项谓词(Minterm Predicate)

是简单谓词的结合，并且任取两项，要么相等，要么相反
Ex:
m1:NAME="A" $\wedge$ BUDGET $<$ 2000
m2:NOT(NAME="A") $\wedge$ BUDGET $<$ 2000
m3:NAME="A" $\wedge$ NOT(BUDGET $<$ 2000)
m4:NOT(NAME="A") $\wedge$ NOT(BUDGET $<$ 2000)

数据库集成

GCS：全局数据模式，也称中介模式
LCS：局部概念模式
OLTP:也叫联机事务处理(Online Transaction Processing),强调数据库内存效率，强调内存各种指标的命令率，强调绑定变量，强调并发操作
OLAP:也叫联机分析处理(Online Analytical Processing),强调数据分析，强调SQL执行市场，强调磁盘I/O，强调分区等

自底向上的设计方法

GCS与LCS的关系:局部作为视图(LAV,Local-As-View)和全局作为视图(GAV,Global-As-View)
LAV:系统定义GCS，把LCS看做GCS上定义的一个视图，查询主要受到局部DBMS的限制，可能不完全
GAV:系统基于多个LCS上的视图定义GCS，查询时，局部DBMS可能有更多的信息，但是会被GCS的定义限制

模式翻译

用于将组件数据库的模式翻译成规范的中间形式，这个规范表示模型需要有足够的表达能力，能够包含所有待集成数据库中的概念，可以选用：实体-关系模型，面向对象模型，图模型等

模式生成

利用中间模式生成GCS，包含以下步骤
1.模式匹配：决定已翻译的LCS元素之或是预定义的GCS元素与单个LCS元素之间的语法、语义关系
2.模式集成：将共同的模式元素集成到上未定义的全局概念模式中
3.模式映象：确定任一LCS元素与GCS元素之间的映像关系

模式匹配

规则(rule,r):包含两个元素之间的对应
对应(correspondence,c):可以直接表明两个概念是相似的，也可以是计算两个概念相似性的函数
条件谓词(predicate,p):对应关系成立的条件
相似性分值(similarity value,s):由某种方式定义和计算，范围0 1
一组匹配$M = \{r\}, r = \langle c, p, s\rangle$

软件栈

BASE

Basically
Available
Soft-state
Eventual consistency

Hadoop生态圈

利用MapReduce作为核心的生态圈
缺陷：频繁的IO

GFS/HDFS

分布式文件系统：GFS/HDFS
NoSQL：HBASE/Cassandra/MongoDB

NoSQL分类

Key/Value:
Schemaless:语义结构不够强，没有传统的ACID特性

CAP

C:Consistency一致性
A:Availability可用性
P:Partition Tolerance

Sparding(文件分块)

问题：容错性极地，但是只要有一块坏掉了，原始数据无法恢复
使用Replication(副本机制)：安全(相当于多一个备份)，性能更高
出现的问题：更新时一致性问题
解决方法：Master/Slave机制
写的操作由Master进行，读由slave进行
仍然存在的问题：
Master写入多个Slave仍然需要时间，可能存在延迟
Master工作量大
单点故障问题

解决方法：P2P
1.用户必须等所有更新完毕才能离开(保证数据一致性，但是速度慢，并且网络故障后会一直等待)
2.用户只需要更新一个节点，剩下的自行完成(无法保证数据一致性，可以通过全部读取选择最新来完成，但是仍然会受到网络故障的影响)

GFS

假设与目标

流数据读写：主要用于程序处理批量数据，而非与用户的交互或随机读写，所以主要是追加写
文件尺寸大

设计思路

1.分块：一个Chunk64M
2.通过冗余提高可靠性(多个副本)
3.通过单个master协调数据访问、元数据存储

问题

单点故障 性能瓶颈

解决方案

尽量减少Master参与程度
不使用Master读取数据，仅用于保存元数据
客户端缓存元数据
使用大尺寸数据块64M

Master的功能

存储元数据
文件系统目录管理与加锁
与ChunkServer进行周期性通信
数据块创建，复制与负载均衡
垃圾回收
陈旧数据快删除

元数据

只有三个类型的元数据:
1.文件和块的命名空间
2.从文件到块的映射
3.每个块的副本位置

GFS的特点

采用中心服务器模式

GFS的容错机制

三类元数据，前两类可以使用log恢复，最后通过备份恢复

title

数据仓库：价格贵，只支持结构化数据

g: s: m: