$r(R)$ is a relation on the relation schema R
- E.g., *customer(Customer_schema)*

Find the largest account balance

$\Pi_{balance}(account)$
- $\Pi_{account.balance}$
$(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank

$\Pi_{customer\_name}(\sigma_{branch\_name = "Perryridge"}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$
- $\Pi_{customer\_name}(depositor)$

Find all customers who have at least one account from each branch of "Downtown" and "Uptown"
- $\Pi_{CN}(\sigma_{BN="Downtown"}(depositor \bowtie account)) \cap \Pi_{CN}(\sigma_{BN="Uptown"}(depositor \bowtie account))$

Consider the view all_customer consisting of branches and their customers

create view *all_customer* **as**
$\Pi_{branch\_name,\ customer\_name}(depositor \bowtie account)$
$\cup\ \Pi_{branch\_name,\ customer\_name}(borrower \bowtie loan)$

Aggregation function takes value as the result
- **avg**: average value
- **min**: minimum value
- **max**: maximum value
- **sum**: sum of values
- **count**: number of values

Relation *account* grouped by *branch_name*:

branch_name	account_number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$branch\_name\ \mathcal{G}_{sum(balance)}(account)$

branch_name	balance
Perryridge	1300
Brighton	1500
Redwood	700

Relation schemas & primary keys
- **Bookstore** (BSName, BSaddress, BSTel, BSManager)
- **Book** (BookNo, BName, BPrice, Author, Publisher, PublishYear, Version)
- **Warehouse** (WNo, WName, WAddress, Wadministrator)
- **Sale** (BSName, BookNo, SaleDate, SaleQuantity)
- **Stock** (BookNo, WNo, StockQuantity)

alter table *r* ***add*** *A D*
- All tuples in the relation are a attribute

The alter table command can elation

alter table *r* ***drop*** *A*

分别使用关系代数表达式和**SQL**语句查询
BookNo and quantity of each book written by "Abraham", and the total stock in all warehouses is less than 50 items

```
select BookNo, sum(Stock.StockQuantity) as total_stock
from Book, Stock
where Book.BookNo=Stock.BookNo and
      Book.Author="Abraham"
group by BookNo
having sum(Stock.StockQuantity) <50
```

```
create table instructor
    (ID          varchar(5),
     name        varchar(20) not null,
     dept_name   varchar(20),
     salary      numeric(8, 2),
     primary key (ID),
     check (salary >= 0) )
```

- Add a new tuple to table instructor

insert into *instructor* ***values*** *('10211', 'Smith', 'Computer Science', 66000)*
  - Insertion fails if any integrity constraint is violated
- Delete all tuples from table instructor

delete from *instructor*

```
select *
from instructor
order by salary desc, name asc
```

找出工资至少比Biology系某一位教师工资高的教师的姓名

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology'
```

Find all customers who have both a loan and an account.

```
(select customer_name from depositor)
intersect [all]
(select customer_name from borrower)
```

找出Computer Science系的平均工资

```
select avg (salary)
from instructor
where dept_name = 'Computer Science'
```

找出在2021年有上课的教师总数

```
select count (distinct ID)
from teaches
where year=2021
```

例如：find the names of all branches where the average account balance is more than $1,200.

```
select branch_name, avg (balance)
from account
group by branch_name
having avg (balance) > 1200
```

E.g., find the average balance for each customer who lives in Harrison and has at least three accounts

```
select depositor.customer_name, avg (balance)
from depositor, account, customer
where depositor.account_number=account.account_number and
      depositer.customer_name=customer.customer_name and
      customer_city='Harrison'
group by depositor.customer_name
having count(distinct depositor.account_number) >=3
```

嵌套子查询： Set Membership（续）

Find all customers who have both an account and a loan at the Perryridge branch

```
select distinct customer_name
from borrower, loan
where borrower.loan_number=loan.loan_number and
      branch_name="Perryridge" and
      (branch_name, customer_name) in
      (select branch_name, customer_name
       from depositor, account
       where depositor.account_number =account.account_number)
```

Find all branches that have greater assets than some branch located in Brooklyn

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and
      S.branch_city = 'Brooklyn'
```

找出平均工资最高的系

```
select dept_name
from instructor
group by dept_name
having avg (salary) >= all (select avg (salary)
                            from instructor
                            group by dept_name);
```

A view consisting of branches and their customers

```
create view all_customer as
(select branch_name, customer_name
from depositor, account
where depositor.account_number = account.account_number)
  union
(select branch_name, customer_name
from borrower, loan
where borrower.loan_number = loan.loan_number)
```

Find all customers of the Perryridge branch

```
select customer_name
from all_customer
where branch_name = 'Perryridge'
```

Find the maximum total balance across all branches

```
select max(tot_balance)
from (select branch_name, sum (balance)
      from account
      group by branch_name)
      as branch_total (branch_name, tot_balance))
```

From子句中的子查询

- **Derived Relations**
  - Find the average account balance of those branches where the average account balance is greater than $1200.

```
select branch_name, avg (balance)
from account
group by branch_name
having avg (balance) > 1200
```

```
select branch_name, avg_balance
from (select branch_name, avg (balance) as avg_balance
      from account
      group by branch_name)
where avg_balance > 1200
```

ACID 特性

原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)。ACID 是典型的强一致性要求

丢失更新：A 事务撤销时，把已经提交的 B 事务的更新数据覆盖了。幻读/不可重复读：指在一个事务 A 内，多次读同一个数据，但是事务 A 没有结束时，另外一个事务 B 则修改了该数据。那么事务 A 在 B 事务修改数据之后再次读取该数据，A 事务读到的数据可能和第一次读到的数据不一样

脏读：A 事务读取 B 事务尚未提交的数据，此时如果 B 事务发生错误并执行回滚操作，那么 A 事务读取到的数据就是脏数据

数据分片架构

- **主从架构**：主节点负责存储元数据和客户端访问接口，从节点负责存储数据分片，如HBase
- **对等结构**：无主节点，各个节点都可以接受客户端访问请求，如果自身没有存储相关分片，则该节点会向其他节点查询数据，如Cassandra

多副本解决方案

- 将数据存储为多个副本，副本存储在不同节点上。通常以数据分片为单位实现多副本
- 相对于原始文件或整个表格，分片体积较小，容易检测、拷贝
- 理论上n个副本都可以被读取，但n个副本是否可以被更新，则要视系统实现和用户策略而定
- 例如：HDFS采用基于"机架感知"的三副本机制

单个副本

- 假设系统中数据只有一个副本。则一致性（C）可以得到绝对的保障，由于在读写时不需要通过网络查询其他副本的情况，因此读写性能较高（A），但如果存储数据的节点故障则无法容错，即该设计兼顾CA

多个副本

- 假设系统中数据存在n个副本，采用“读写分离”机制，只有一个副本可以接受写请求
  - 对于写操作，一致性和可用性较好，因为只要写完一个副本，操作即为成功，但此时该写入节点无法实现分区可用性，即兼顾CA
  - 对于读操作，假设数据存在多个“只读”副本，客户端每次**只读取其中一个**，则该设计实现了读操作的分区可用性（多副本），可用性较好，但客户端无法判断该副本是否为最新的（考虑网络通信的不确定性），即只兼顾了PA
  - 对于读操作，假设客户端需要同时读取多个副本，并对比这些副本，以检查是否存在版本差异或版本冲突。则此时兼顾了PC，由于需要读取多个副本，因此客户端响应时间变长，可用性（A）变弱

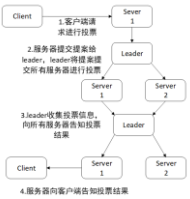
BASE

- **B**asically **A**vailable （基本可用）：部分节点或功能出现故障时，系统的核心部分仍然可用
- **S**oft-state(软状态）：系统允许出现“中间状态”，即多副本内容不一致
- **E**ventual consistency（最终一致性）：系统中的多副本数据经过一定时间后，最终保持一致
- BASE是一个和ACID相对比的概念，强调弱一致性
  - ACID指事务的强一致性，认为事务执行时不应存在中间状态，只有“成功”、“回滚”等最终状态
  - 在分布式环境下，涉及到网络通信的不可靠性，性能较差，且技术实现复杂

最终一致性

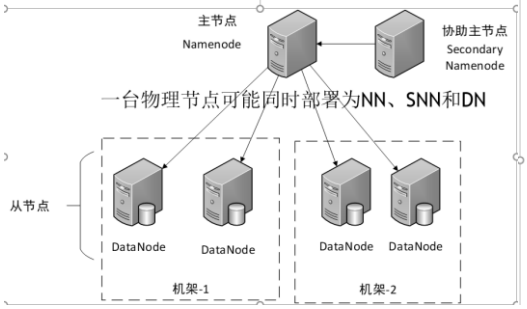
- BASE强调，在互联网等场景中，用户响应（即可用性）很重要，必须首先满足
  - 最终一致性，即事务存在中间状态，但经历一段时间之后，最终会一致
  - 最终一致性（在一些应用场景下）也可以看作NoSQL允许多个副本可以存在暂时的不同步（即异步更新），结合CAP理论，这种设计强调PA，可以提高响应速度

Paxos共识算法



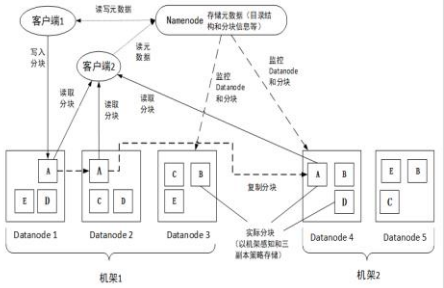
- **HDFS & HBase**
  - HDFS: 分布式文件系统
  - HBase: 分布式列数据库
- **Cassandra**
  - 分布式Key-Value数据库
- **MongoDB**
  - 分布式文档数据库
- **OpenGauss或者OceanBase**
  - NewSQL数据库
- **数据库系统体系结构**

HDFS架构



Namenode的数据结构

- **NN以文件方式存储元数据**
  - fsimage：只读状态，在NN启动时整体读入内存
  - editlog：以日志方式存储文件信息的变动情况
- **NN启动时**
  - 构造新的fsimage，并将其读入内存
  - 在该过程中，HDFS接受读请求，不接受写请求，
  - 该过程称为“安全模式”（safe mode）
- **editlog 需要定期合并，形成新的fsimage**
  - NN在每次启动之前进行fsimage 的更新工作
  - SNN可以代替NN完成该工作



HBase的数据模型

实际存储时，不同列族存储为HDFS上的不同文件

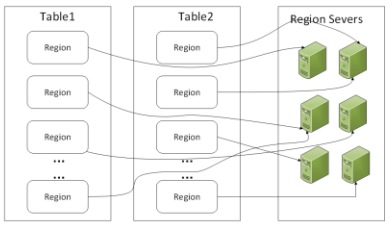
ROWKEY	Name	Value	Timestamp
001	playername	Micheal Jordan	1270073054
001	Uniform Number	23	1270073054
001	Position	Shooting guard	1270073054
002	Firstname	Kobe	1270084021
002	Lastname	bryant	1270084021
002	Uniform Number	8	1270084021
002	Position	SG	1270084021
002	Uniform Number	24	1270164055

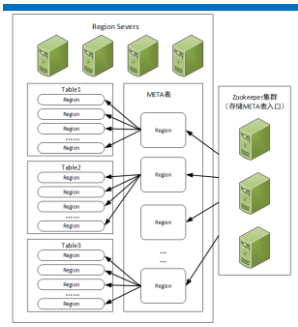
HBase的拓扑结构

- **HMaster节点**
  - 所有Hregionserver的管理者，负责对Hregionserver的管理范围进行分配
  - 不负责管理用户数据表
- **Hregionserver节点**
  - 用户数据表的实际管理者，在分布式集群中，数据表会进行水平分区，每个Hregionserver只会对一部分分区进行管理——负责数据的写入、查询、缓存和故障恢复等
  - 用户表最终是以文件形式存储在HDFS上，但如何将写入并维护这些文件，则由Hregionserver负责

水平分区原理

HBase对大数据表进行**水平分割**，形成不同**区域**(region)，由不同的Regionserver进行管理，分区过程可以自动进行，不需要用户干预





Zookeeper->META表->Regionserver->Region

Amazon Dynamo

Dynamo是一个基于点对点模式的分布式键值对存储系统

- **节点对称：**各个节点的角色、权重相同，简化整个集群系统的配置和维护
- **去中心化：**在对称的基础上，避免通过主节点对集群进行集中控制
- **水平扩展性：**以主机为单位实现横向扩展，扩展方式较简单，对集群整体影响较小
- **支持异构设备：**在扩展节点时，可以使用和原节点配置不同(如性能更高)的主机，即集群中可以存在多种配置的主机
- **多副本机制：**采用多副本数据机制，但强调弱一致性和高可用性，即CAP理论中的AP。Dynamo和Cassandra中的一致性和可用性权重可以根据用户策略调整
- **数据模式：**采用基于键值对和列族等概念的数据模式

Amazon Dynamo

数据一致性

- 用户可以在（读）写完部分副本（设为R或W）而非全部N个副本时，就返回写入成功
  - 出现部分节点故障时，提高系统可用性
  - R、W、N可由用户配置，R、W为1，可用性最强，一致性最差；R、W为N，可用性最差，一致性最强
  - 在一致性要求高时，推荐R+W>N，而实时性要求高时，则R+W<N。在实际应用中，经常设置为2、2、3
- Dynamo使用**矢量时钟**机制来记录数据对象更新的时序关系
  - 矢量时钟：一个键值对列表，结构为（node, counter）
  - 当Dynamo更新一个数据时，必须记录更新自哪个版本
  - 如果发现新旧两个版本的数据，则可以进行语法协调，并且一般会选择较新版本的数据
  - 如果发现多个不能语法协调的分支，将返回分支处的所有数据对象，其包含与上下文相应的版本信息。客户端决定如何协调

- **MerkleTree机制**
  - 用于副本一致性检测
  - 叶子是各个键的哈希值。树中较高的父节点存储各自子节点信息哈希值的汇总哈希值

如果两个副本（树）的根哈希值相等，则对应的副本不需要同步，如果根哈希值和某树枝的哈希值不同，则意味着该分支下的一些副本的值是不同的，只要继续比较子节点的哈希值，就可以识别出不同步的键值对，由此缩减同步过程中需要传输的数据量

Cassandra和Dynamo

Cassandra is the daughter of Amazon **Dynamo DB** and Google **Bigtable**

- Cassandra在分布式结构设计上充分借鉴Dynamo，也采用了环结构、Gossip协议、暗示移交等机制
- Dynamo的数据模型类似于“文档型”，Cassandra的数据模型类似于HBase的键值对和面向列模型
- Cassandra早期没有虚拟节点概念，担心数据迁移和维护时的效率问题
- Cassandra没有采用MerkelTree和向量时钟机制。因为面向列的模型建立MerkelTree较为繁琐，需要为每个列建立树（Dynamo相当于一行只有一类，因此树结构相对简单）
- Cassandra采用时间戳解决冲突，最新时间戳获胜
- 暗示移交、（W、R、N）设置等机制存在细节策略上的不同

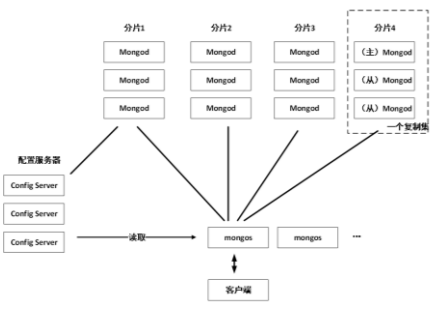
Cassandra的数据模型类似于Bigtable和HBase具有行键、列、列族、时间戳等概念。在用户建表时需要提前建立列，而HBase不需要Key:表示行键

Column: 表示列，列中存储的数据为三元组（name, value, timestamp）

MongoDB分片机制和集群架构

复制集（Replication Set）机制

- 面向分片的主从分布式结构
  - **主节点（Primary Node）**：负责数据的**写入**和**更新**。主节点在更新数据的同时，会将操作信息写入日志，称为oplog
  - **从节点（Secondary Node）**：监听主节点oplog的变化，根据其内容维护自身的数据更新，使之和主节点保持一致（最终一致性）
- 不会出现写冲突的情况，但可能出现（暂时的）**主节点单点失效**问题
- 用户即可以由主节点读取数据，也可以由从节点读取数据（是否一样？）
- 从节点可以配置为多个，各节点相互通信、相互检测心跳信
- 当主节点宕机时，从节点会检测到错误，并通过选举方式（一半以上的从节点投票通过），使某一台从节点提升为主节点，接管数据更新操作



MongoDB支持的索引类型 稀疏索引(sparse: true) 唯一索引(unique: true)

全文索引对字符串类型（包括字符串数组类型）的字段建立 “text”类型的索引

建立全文索引db. mycol. createIndex ({"item1": "text"})

全文索引可以针对多个字段建立db.mycol.createIndex ({"item1": "text", "item2": "text"})

每个集合只能建立一个全文索引

全文检索db.mycol.find({$text:{$search: "first"}}) 只能针对建立全文索引的字段进行

地理位置索引

位置数据的表示db.geo_db.insert({"name" : "school", "loc" : [10,20]})

合法的经纬度坐标取值范围为从[-179,-179]到[180,180]，否则使用时会报错

建立地理信息索引db.geo_db.createIndex({"loc" : "2d"})

根据距离排序db.geo_db.find({loc:{$near:[10,20]}})

设定最大距离限制db.geo_db.find({loc:{$near:[10,20],$maxDistance:50}})

50为两点经纬度的数值差异之和，并非精确值

维度差异所代表的实际距离会随着维度高低而不同，这种简单求差的计算方式必然存在误差