# 二、数据模型与查询语言

数据模型是开发软件最重要的部分，对软件的编写方式和如何思考待解决的问题都有深远影响。

多数应用程序都是通过一层层叠加数据模型来构建的。每一层都面临关键问题：如何将其用下一层来表示？例如

1. 开发人员，观测现实世界（包括人员、组织、货物、行为、资金流动、传感器等），通过对象或数据结构，以及操作这些数据结构的API来对其建模。这些数据结构往往特定于该应用。
2. 当需要存储这些数据结构时，可以采用通用数据模型（JSON或XML文档、关系数据库中的表或图模型）来表示
3. 数据库工程师接着决定用何种内存、磁盘或网络的字节格式来表示上述JSON / XML / 关系 / 图形数据。数据表示需要支持多种方式的查询、搜索、操作和处理数据
4. 更下一层，硬件工程师需要考虑用电流、光脉冲、磁场等来表示字节

复杂应用可能有更多的中间层，例如基于API来构建上层API，但是基本思想相同：每层都通过提供一个简洁的数据模型来隐藏下层的复杂性。这种抽象机制使得不同的人群可以高效协作，例如数据厂商的工程师和使用数据库的应用程序开发人员一起合作。

许多不同类型的数据模型都有各自最佳使用的若干假设。用法支持与否、操作性能快慢，数据转换自然与否。

本章介绍一系列用于数据存储和查询的通用数据模型。比较关系模型、文档模型和一些基于图的数据模型。还将讨论多种查询语句并比较它们的使用场景。第三章将讨论存储引擎，即这些数据模型是如何实现的。

## 1. 关系模型与文档模型

现在最著名的数据模型可能是SQL。现在最著名的数据模型可能是SQL。它基于Edgar Codd在1970年提出的关系模型【1】：数据被组织成关系（SQL中称作表），其中每个关系是元组（SQL中称作行)的无序集合。

关系数据库的核心在于商业数据处理。关系模型的目标就是将实现细节隐藏在更简洁的接口后面。

**常见分类**

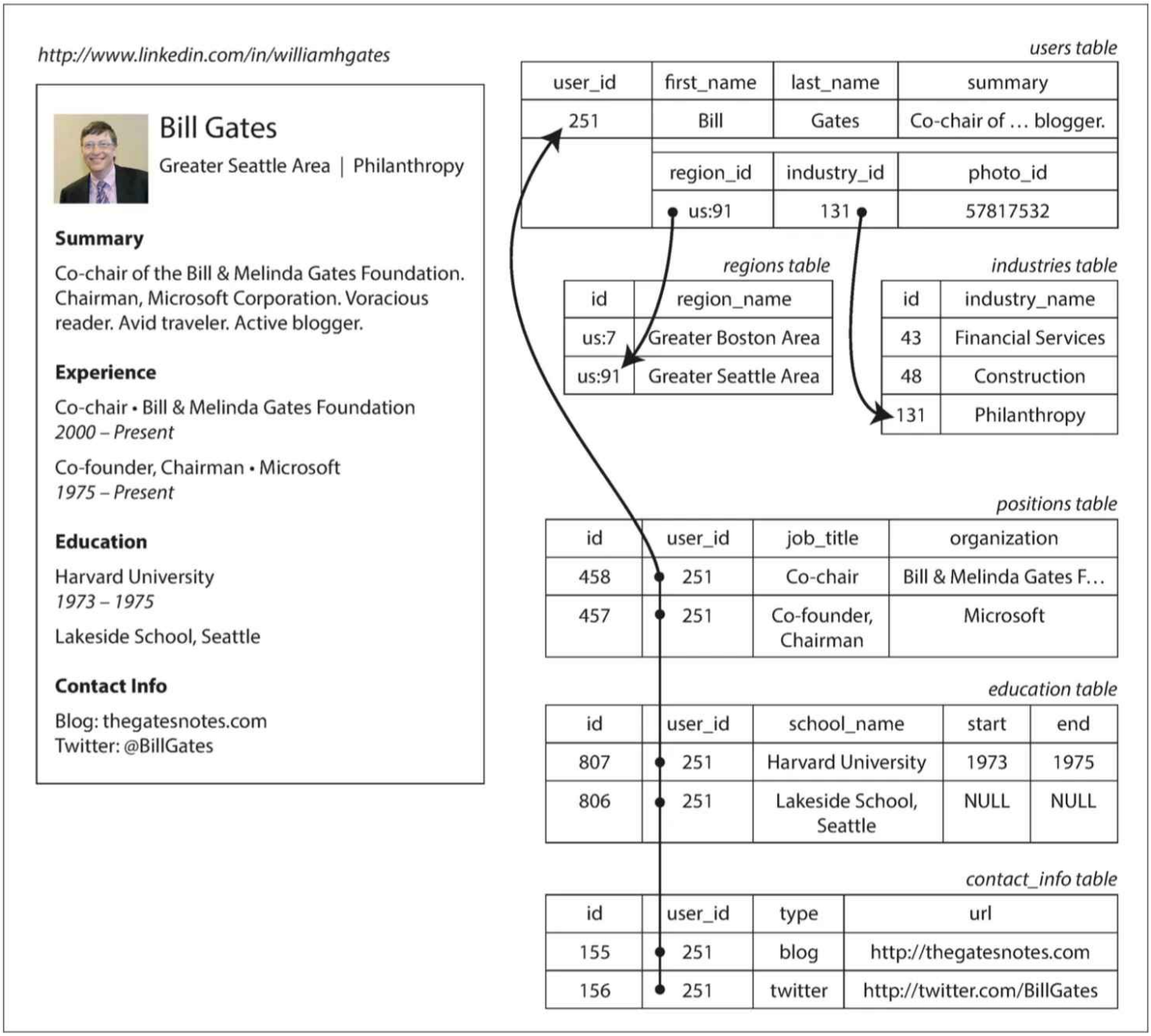
1. 事务型（TP）：银行交易、火车票
2. 分析型（AP）：数据报表、监控表盘
3. 混合型（HTAP）：

## 1.1 NoSQL 的诞生

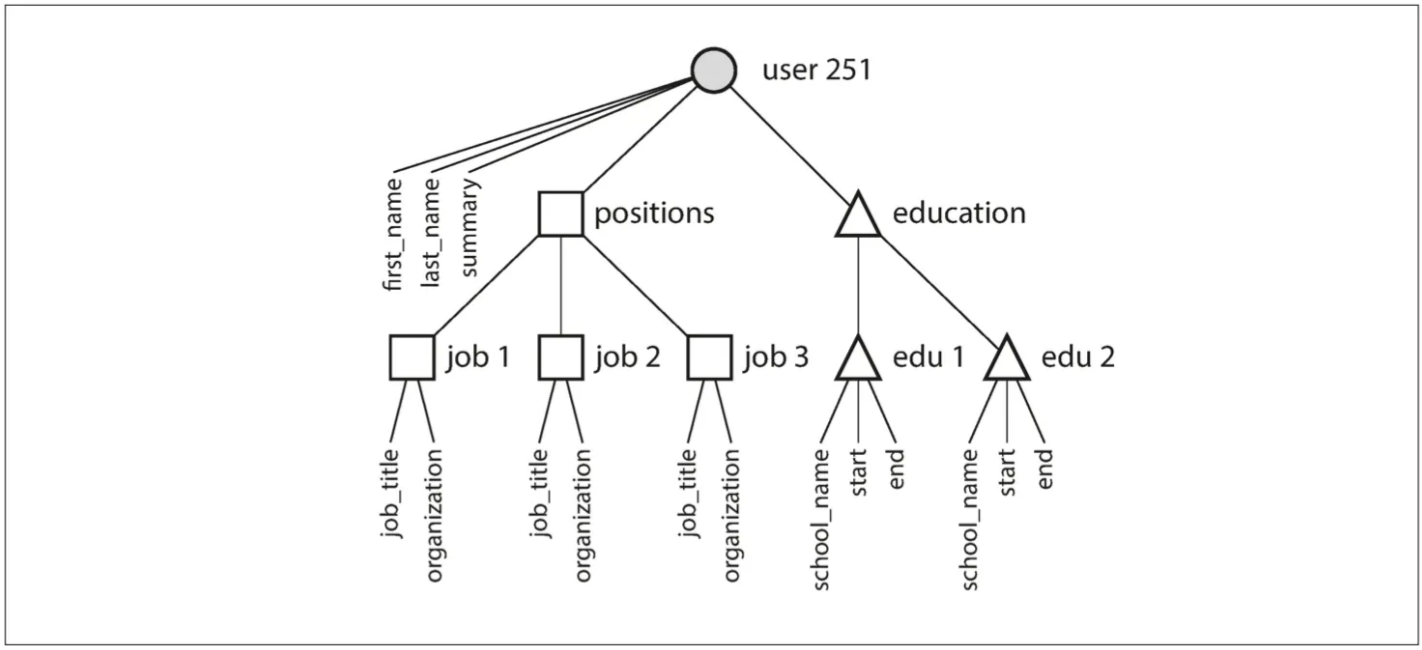
1. NoSQL：不仅是SQL（Not Only SQL）是对不同于传统的关系数据库的数据库管理系统的统称。根据 DB-Engines 排名，现在最受欢迎的 NoSQL 前几名为：MongoDB，Redis，ElasticSearch，Cassandra。
2. 采用NoSQL数据库的背后有几个驱动因素，其中包括：
    - 需要比关系数据库更好的可扩展性，包括支持超大数据集或非常高的写入吞吐量
    - 相比商业数据库产品，免费和开源软件更受偏爱。
    - 关系模型不能很好地支持一些特殊的查询操作
    - 受挫于关系模型的限制性，渴望更具动态性与表现力的数据模型
3. 在可预见的未来，关系数据库似乎可能会继续与各种非关系数据库一起使用 - 这种想法有时也被称为混合持久化（polyglot persistence）

## 1.2 对象 - 关系不匹配

1. 应用程序使用面向对象的语言，需要一个转换层，才能转成 SQL 数据模型：被称为阻抗不匹配。
2. Hibernate这样的 对象关系映射（ORM object-relational mapping） 框架可以减少这个转换层所需的样板代码的数量，但是它们不能完全隐藏这两个模型之间的差异。
3. 对于一份简历而言，关系型模型需要一对多（比如工作经历）。




而表述这样的简历，使用 JSON 是非常合适的。JSON 比多表模式有更好的局部性，可以一次查询出一个用户的所有信息。JSON 其实是一棵树。




换另一个角度来说，关系模型很难直观的表示一对多的关系。比如简历上，一个人可能有多段教育经历和多段工作经历。

文档模型：使用 Json 和 XML 的天然嵌套。

关系模型：使用 SQL 模型就得将职位、教育单拎一张表，然后在用户表中使用外键关联。

在简历的例子中，文档模型还有几个优势：

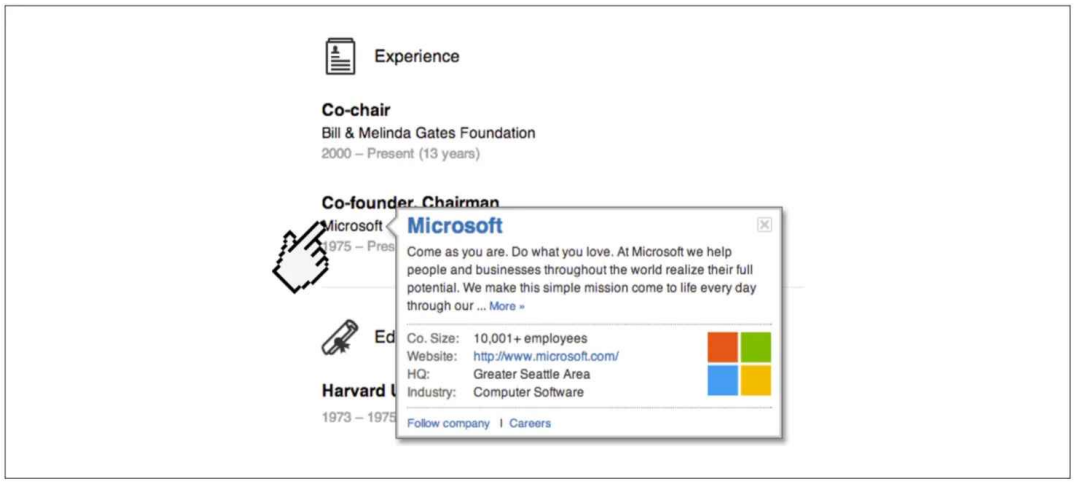
1. 模式灵活：可以动态增删字段，如工作经历。
2. 更好的局部性：一个人的所有属性被集中访问的同时，也被集中存储。
3. 结构表达语义：简历与联系信息、教育经历、职业信息等隐含一对多的树状关系可以被 JSON 的树状结构明确表达出来。

## 1.3 多对一与多对多的关系

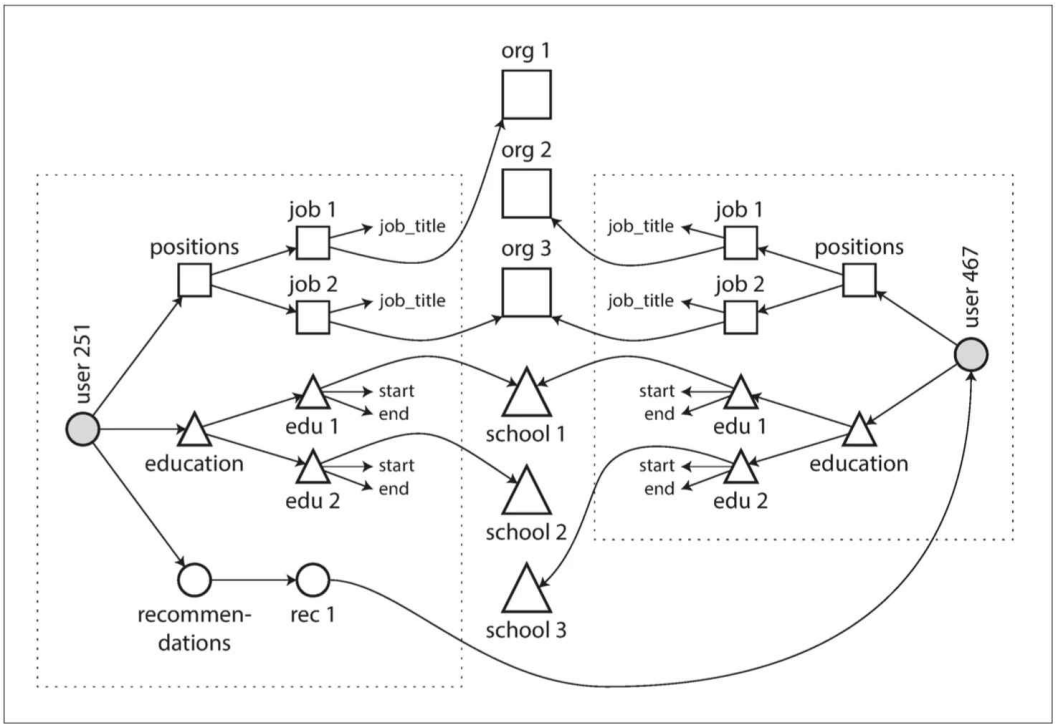
1. 为什么在 SQL 中，地域和公司都以 ID，而不是字符串进行标识呢？
    - ID 对人类没有任何意义，所以永远不需要改变，可以规范化人类的信息。那么就会存在多对一的关系（多个人对应了同一个 ID）。
    - 在关系数据库 SQL 中，所有使用它的地方可以用 ID 来引用其他表中的行；
    - 但是文档数据库（比如 JSON），对连接支持很弱。
2. 如果数据库不支持连接，那么就需要在应用代码中，对数据库执行多个查询进行模拟。执行连接的工作从数据库被转移到应用程序代码上。

哪怕最开始的应用适合无连接的文档模型，但是随着功能添加，数据会变得更加互联，比如对简历修改：

1. 组织和学校作为实体：假如组织和学校有主页




2. 推荐：给别人做推荐，当别人的信息更改的时候，所有地方要同步更新。




**文档 vs 关系**

1. 对于一对多关系，文档型数据库将嵌套数据放在父节点中，而非单拎出来放另外一张表。
2. 对于多对一和多对多关系，本质上，两者都是使用外键（文档引用）进行索引。查询时需要进行 join 或者动态跟随。

## 1.4 文档数据库是否在重演历史？

20 世纪 70 年代，最受欢迎的是层次模型（hierarchical model），它与文档数据库使用的JSON模型有一些惊人的相似之处。它将所有数据表示为嵌套在记录中的记录树。虽然能处理一对多的关系，但

是很难应对多对多的关系，并且不支持链接。 提出的解决方案：

1. 关系模型（relational model）（它变成了SQL，统治了世界）
2. 网络模型（network model）（最初很受关注，但最终变得冷门）

### 1.4.1 网络模型

1. 支持多对多，每条记录可能有多个父节点。
2. 网络模型中记录之间的链接不是外键，而更像编程语言中的指针（同时仍然存储在磁盘上）。访问记录的唯一方法是跟随从根记录起沿这些链路所形成的路径。这被称为访问路径（access path）。
3. 最简单的情况下，访问路径类似遍历链表：从列表头开始，每次查看一条记录，直到找到所需的记录。但在多对多关系的情况中，数条不同的路径可以到达相同的记录，网络模型的程序员必须跟踪这些不同的访问路径。
4. 缺点：查询和更新数据库很麻烦。

### 1.4.2 关系模型

1. 数据：一个 关系（表） 只是一个 元组（行） 的集合，很简单。
2. 在关系数据库中，查询优化器自动决定查询的哪些部分以哪个顺序执行，以及使用哪些索引。这些选择实际上是“访问路径”，但最大的区别在于它们是由查询优化器自动生成的，不需要程序猿考虑。

### 1.4.3 文档数据库的比较

但是，在表示多对一和多对多的关系时，关系数据库和文档数据库并没有根本的不同：在这两种情况下，相关项目都被一个唯一的标识符引用，这个标识符在关系模型中被称为外键，在文档模型中称为文档引用。

## 1.5 关系数据库与文档数据库现状

- 支持文档数据模型的主要论据是架构灵活性，因局部性而拥有更好的性能，以及对于某些应用程序而言更接近于应用程序使用的数据结构。
- 关系模型通过为连接提供更好的支持以及支持多对一和多对多的关系来反击。

**哪种数据模型的应用代码更简单？**

文档模型：

- 优点：
  - 如果应用程序中的数据具有类似文档的结构（即，一对多关系树，通常一次性加载整个树），那么使用文档模型可能是一个好主意。
- 缺点：
  - 不能直接引用文档中的嵌套的项目，而是需要说“用户251的位置列表中的第二项”（很像分层模型中的访问路径）。但是，只要文件嵌套不太深，这通常不是问题。
  - 文档数据库对连接的糟糕支持也许或也许不是一个问题，这取决于应用程序。
  - 如果应用程序使用多对多关系，那么文档模型就没有那么吸引人了。
  - 对于高度相联的数据，选用文档模型是糟糕的，选用关系模型是可接受的，而选用图形模型是最自然的。

**文档模型的模式灵活性**

1. 文档模型是「读时模式」
   - 文档数据库有时称为无模式（schemaless），但这具有误导性，因为读取数据的代码通常假定某种结构——即存在隐式模式，但不由数据库强制执行
   - 一个更精确的术语是读时模式（schema-on-read）（数据的结构是隐含的，只有在数据被读取时才被解释），相应的是写时模式（schema-on-write）（传统的关系数据库方法中，模式明确，且数据库确保所有的数据都符合其模式）
   - 读时模式类似于编程语言中的动态（运行时）类型检查，而写时模式类似于静态（编译时）类型检查。
2. 模式变更
   - 读时模式变更字段很容易，只用改应用代码
   - 写时模式变更字段速度很慢，而且要求停运。它的这种坏名誉并不是完全应得的：大多数关系数据库系统可在几毫秒内执行ALTER TABLE语句。MySQL是一个值得注意的例外，它执行ALTER TABLE时会复制整个表，这可能意味着在更改一个大型表时会花费几分钟甚至几个小时的停机时间，尽管存在各种工具来解决这个限制。

**查询的数据局部性**

- 文档通常以单个连续字符串形式进行存储，编码为JSON，XML或其二进制变体
- 读文档：
  - 如果应用程序经常需要访问整个文档（例如，将其渲染至网页），那么存储局部性会带来性能优势。

    - 局部性仅仅适用于同时需要文档绝大部分内容的情况。
- 写文档：
    - 更新文档时，通常需要整个重写。只有不改变文档大小的修改才可以容易地原地执行。
    - 通常建议保持相对小的文档，并避免增加文档大小的写入

**文档数据库与关系数据库的融合**

- MySQL 等逐步增加了对 JSON 和 XML 的支持
- 关系模型和文档模型的混合是未来数据库一条很好的路线。

# 2. 数据查询语言

- 关系模型包含了一种查询数据的新方法：SQL是一种 声明式 查询语言，而IMS和CODASYL使用 命令式 代码来查询数据库。
- 命令式语言：告诉计算机以特定顺序执行某些操作，比如常见的编程语言。
- 声明式查询语言（如SQL或关系代数）：你只需指定所需数据的模式 - 结果必须符合哪些条件，以及如何将数据转换（例如，排序，分组和集合） - 但不是如何实现这一目标。数据库系统的查询优化器决定使用哪些索引和哪些连接方法，以及以何种顺序执行查询的各个部分。
    - SQL相当有限的功能性为数据库提供了更多自动优化的空间。
    - 声明式语言往往适合并行执行。

## 2.1 Web上的声明式查询

- 声明式语言更加泛化，不用关心底层的数据存储变化
- 在Web浏览器中，使用声明式CSS样式比使用JavaScript命令式地操作样式要好得多。另外，浏览器厂商可以在不破坏兼容性的情况下提高CSS和XPath的性能
- 类似地，在数据库中，使用像SQL这样的声明式查询语言比使用命令式查询API要好得多。

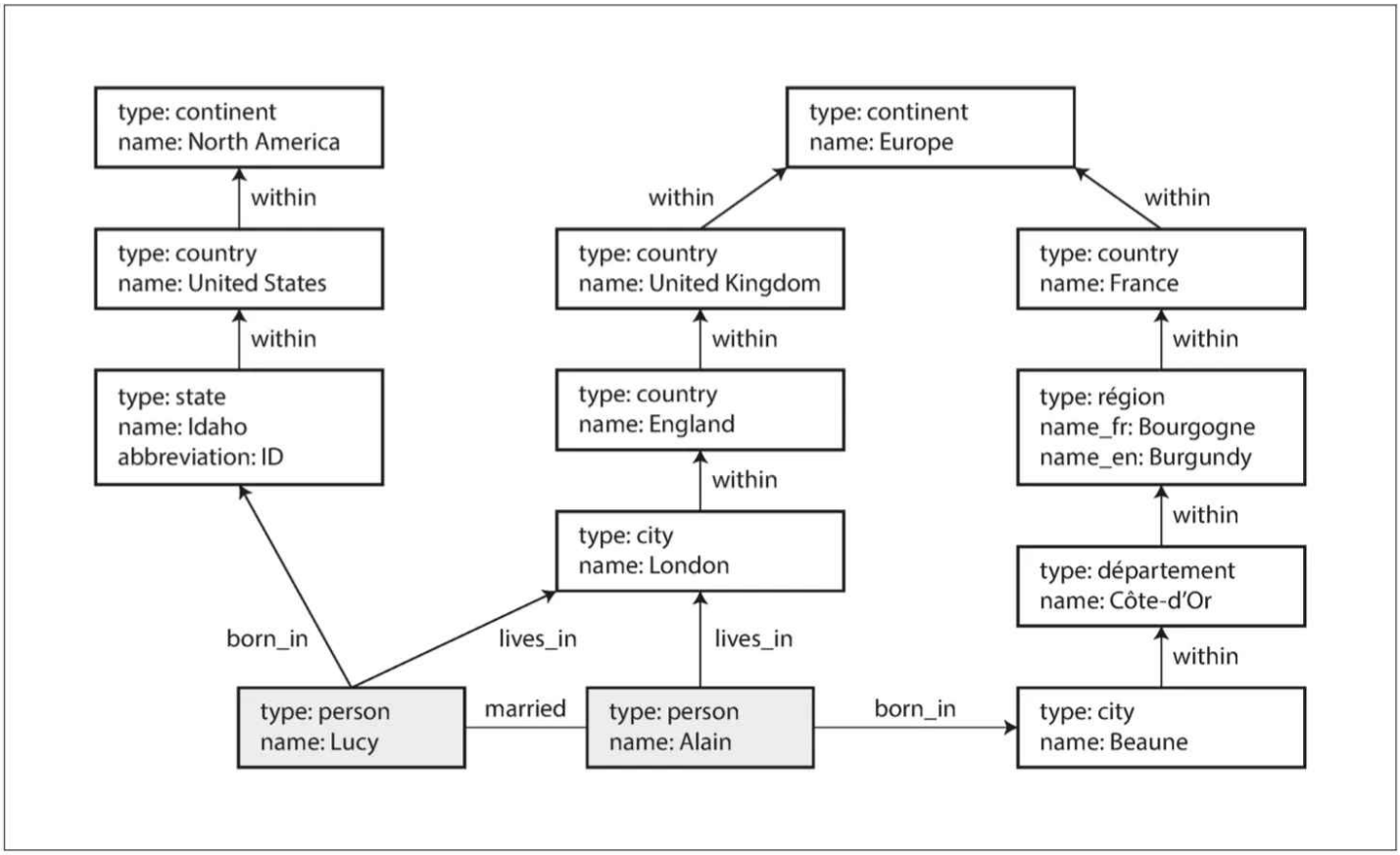
## 2.2 MapReduce查询

- MapReduce是一种编程模型，用于在许多机器上批量处理海量数据。一些NoSQL数据存储（包括MongoDB和CouchDB）支持有限形式的MapReduce，作为在多个文档中执行只读查询的机制。
- MapReduce既不是一个声明式的查询语言，也不是一个完全命令式的查询API，而是处于两者之间
    - 查询的逻辑用代码片段来表示，这些代码片段会被处理框架重复性调用。

    - 它基于map（也称为collect）和reduce（也称为fold或inject）函数，两个函数存在于许多函数式编程语言中。
- map和reduce函数在功能上有所限制：
    - 它们必须是纯函数，这意味着它们只使用传递给它们的数据作为输入，它们不能执行额外的数据库查询，也不能有任何副作用。
    - 这些限制允许数据库以任何顺序运行任何功能，并在失败时重新运行它们。
    - MapReduce是一个相当底层的编程模型，用于计算机集群上的分布式执行。像SQL这样的更高级的查询语言可以用一系列的MapReduce操作来实现，但是也有很多不使用MapReduce的分布式SQL实现。
    - MapReduce的一个可用性问题：必须编写两个密切合作的JavaScript函数，这通常比编写单个查询更困难。此外，声明式查询语言为查询优化器提供了更多机会来提高查询的性能。基于这些原因，MongoDB 2.2添加了一种叫做聚合管道的声明式查询语言的支持

## 3. 图数据模型

1. 多对多关系是不同数据模型之间具有区别性的重要特征。
2. 文档模型：适合数据有一对多关系、不存在关系
3. 图数据模型：适合多对多关系
4. 一个图由两种对象组成：
    a. 顶点（vertices）（也称为节点（nodes） 或实体（entities））
    b. 边（edges）（ 也称为关系（relationships）或弧 （arcs））。
5. 举例：*社交图谱，网络图谱，公路或铁路网络*
6. 图数据结构示例（以社交网络为例）




7. 存储方式：属性图，三元组

## 3.1 属性图

在属性图模型中，每个**顶点（vertex）**包括：

- 唯一的标识符
- 一组 出边（outgoing edges）
- 一组 入边（ingoing edges）
- 一组属性（键值对）

每条 边（edge） 包括：

- 唯一标识符
- 边的起点/尾部顶点（tail vertex）
- 边的终点/头部顶点（head vertex）
- 描述两个顶点之间关系类型的标签
- 一组属性（键值对）

使用关系模式来表示属性图

```
1 CREATE TABLE vertices (
```

```
  vertex_id  INTEGER PRIMARY KEY,
  properties JSON
);CREATE TABLE edges (
  edge_id     INTEGER PRIMARY KEY,
  tail_vertex INTEGER REFERENCES vertices (vertex_id),
  head_vertex INTEGER REFERENCES vertices (vertex_id),
  label       TEXT,
  properties  JSON
);CREATE INDEX edges_tails ON edges (tail_vertex);CREATE INDEX edges_heads ON
edges (head_vertex);
```

关于这个模型的一些重要方面是：

1. 任何顶点都可以有一条边连接到任何其他顶点。没有模式限制哪种事物可不可以关联。
2. 给定任何顶点，可以高效地找到它的入边和出边，从而遍历图，即沿着一系列顶点的路径前后移动。（这就是为什么例2-2在tail_vertex和head_vertex列上都有索引的原因。）
3. 通过对不同类型的关系使用不同的标签，可以在一个图中存储几种不同的信息，同时仍然保持一个清晰的数据模型。

## 3.2 Cypher查询语言

1. Cypher是属性图的声明式查询语言，为Neo4j图形数据库而发明
2. 通常对于声明式查询语言来说，在编写查询语句时，不需要指定执行细节：查询优化程序会自动选择预测效率最高的策略，因此你可以继续编写应用程序的其他部分。
3. 查找所有从美国移民到欧洲的人的Cypher查询

```
MATCH
    (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United
States'}),
    (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location {name:'Europe'})
RETURN person.name
```

## 3.3 SQL的图查询

1. 用关系数据库表示图数据，那么也可以用SQL，但有些困难。
2. 在关系数据库中，你通常会事先知道在查询中需要哪些连接。在图查询中，你可能需要在找到待查找的顶点之前，遍历可变数量的边。也就是说，连接的数量事先并不确定。

3. 语法很复杂

## 3.4 三元存储与SPARQL

1. 在三元组存储中，所有信息都以非常简单的三部分表示形式存储（主语，谓语，宾语）。例如，三元组 (吉姆, 喜欢 ,香蕉) 中，吉姆 是主语，喜欢 是谓语（动词），香蕉 是对象。
2. 三元组的主语相当于图中的一个顶点。而宾语是下面两者之一：
   a. 原始数据类型中的值，例如字符串或数字。在这种情况下，三元组的谓语和宾语相当于主语顶点上的属性的键和值。例如，(lucy, age, 33)就像属性{“age”：33}的顶点lucy。
   b. 图中的另一个顶点。在这种情况下，谓语是图中的一条边，主语是其尾部顶点，而宾语是其头部顶点。例如，在(lucy, marriedTo, alain)中主语和宾语lucy和alain都是顶点，并且谓语marriedTo是连接他们的边的标签。
3. 当主语一样的时候，可以进行省略写法

```
@prefix : <urn:example:>.
_:lucy      a :Person;   :name "Lucy";          :bornIn _:idaho.
_:idaho     a :Location; :name "Idaho";         :type "state";   :within _:usa
_:usa       a :Loaction; :name "United States"; :type "country"; :within
  _:namerica.
_:namerica  a :Location; :name "North America"; :type "continent".
```

## 3.5 Datalog

# 小结

数据模型是一个庞大的主题，本章介绍了各种不同的数据模型。无法详细介绍每个模型的细节，但是希望可以激起你的兴趣，进一步寻找最适合应用需求的模型。

历史上，数据最初被表示为一棵大树（层次模型），但是不利于表示多对多关系，所以发明了关系模型来解决这个问题。最近，发现一些应用也不太适合关系模型。新的非关系 NoSQL 数据存储在两个主要方向上存在分歧：

1. 文档数据库的目的用例是数据来自于自包含文档，且一个文档与其他文档之间的关联很少
2. 图数据库则针对相反的场景，目标用例是所有数据都可能会互相关联

文档模型、关系模型和图模型，如今都有广泛使用，并且在各自的目标领域足够优秀。

文档数据模型和图数据库有一个共同点，它们通常不会对存储的数据强加某个模式，这可以使应用程序更容易适应不断变化的需求。

每个数据模型都有自己的查询语言或框架，例如：SQL、MapReduce、MongoDB的聚合管道、Cypher、SPARQL和Datalog。CSS和XSL/XPath，它们不属于数据库查询语言，但存在相似之处。

尚未提及的数据模型：

- 使用基因组数据的研究人员经常需要执行序列相似性搜索，意味着需要用一个非常长的字符串（代表一个DNA分子），与存在相似但不完全相同的大型字符串数据库进行匹配。以上介绍的数据库都不适用于这种场景，研究人员开发了专门的基因组数据库软件，如GenBank
- 粒子物理学家一直进行海量数据的超大规模数据分析，像大型强子对撞机（LHC）这样的项目，现在可以处理数百PB项目的数据。这种规模下，需要定制解决方案来避免硬件成本失控。
- 全文搜索是一种经常与数据库一起使用的数据模型。信息检查是一个很大的专业课题，第三章和第三部分中介绍搜索索引相关内容。