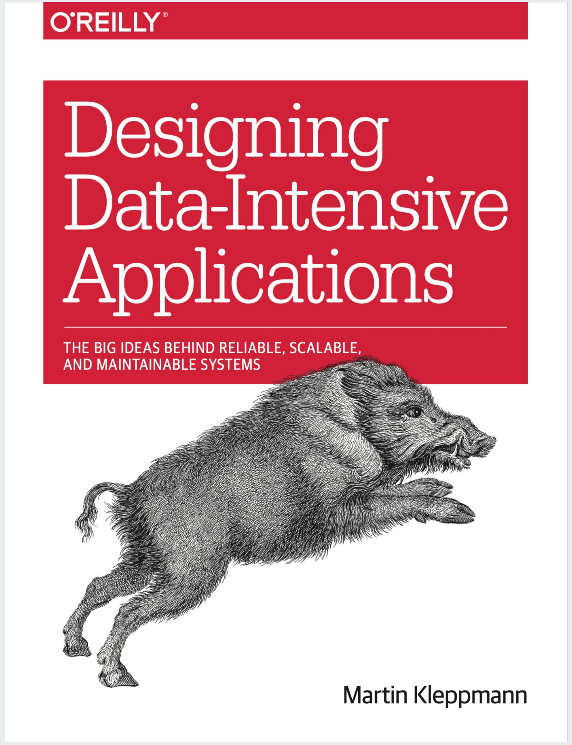
Design Data Intensive Application

The Big Ideas Behind Reliable, Scalable, and Maintainable Systems


O'REILLY

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

Martin Kleppmann

Chapter2:
Data Models and Query Languages

Overview

Data models are perhaps the most important part of developing software: not only on how the software is written, but also on how we think about the problem that we are solving.

Most applications are built by layering one data model on top of another. For each layer, the key question is: how is it represented in terms of the next-lower layer? each layer hides the complexity of the layers below it by providing a clean data model.

General-purpose data models for data storage and querying

Relational DB: IBM DB2, MS SQL Server, PostgreSQL

Document DB: MongoDB, RethinkDB, CouchDB, Espresso

Graph-based DB: Neo4j, Titan, InfiniteGraph

Relational Model Versus Document Model

The best-known data model today is probably that of SQL: data is organized into relations (called tables in SQL), where each relation is an unordered collection of tuples (rows in SQL).

The goal of the relational model was to hide that implementation detail behind a cleaner interface.

Relational Model Versus Document Model: The Birth of NoSQL

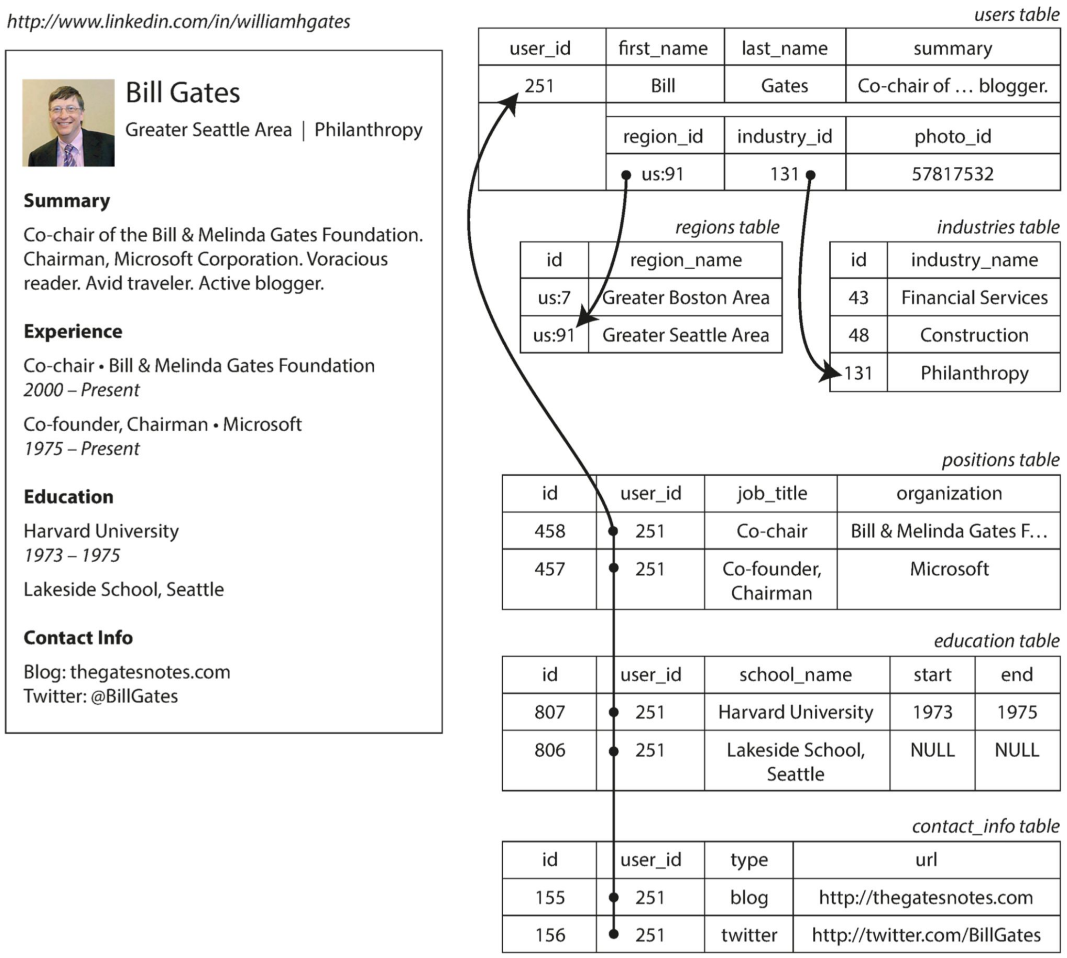
(Not Only SQL)

- A need for greater scalability than relational databases can easily achieve, including very large datasets or very high write throughput
- A widespread preference for free and open source software over commercial database products
- Specialized query operations that are not well supported by the relational model
- Frustration with the restrictiveness of relational schemas, and a desire for a more dynamic and expressive data model

Polyglot persistence: It therefore seems likely that in the foreseeable future, relational databases will continue to be used alongside a broad variety of nonrelational datastores

Relational Model Versus Document Model: The Object-relational Mismatch

If data is stored in relational tables, an awkward translation layer is required between the objects in the application code and the database model of tables, rows, and columns. The disconnect between the models is sometimes called an **impedance mismatch**.



http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Relational Model Versus Document Model: The Object-relational Mismatch

- In the traditional SQL model (prior to SQL:1999), the most common normalized representation is to put positions, education, and contact information in separate tables, with a foreign key reference to the users table.
- Later versions of the SQL standard added support for structured datatypes and XML data; this allowed multi-valued data to be stored within a single row, with support for querying and indexing inside those documents.
- A third option is to encode jobs, education, and contact info as a JSON or XML document, store it on a text column in the database, and let the application interpret its structure and content.

```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University",       "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog":    "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```

Relational Model Versus Document Model: The Object-relational Mismatch

The JSON representation has better locality than the multi-table schema.

If you want to fetch a profile in the relational example, you need to either perform multiple queries or perform a messy multiway join between the users table and its subordinate tables.

In the JSON representation, all the relevant information is in one place, and one query is sufficient.

Relational Model Versus Document Model: Many-to-one and Many-to-Many Relationships

plain-text vs standardized lists for region/industrial

- Consistent style and spelling across profiles
- Avoiding ambiguity (e.g., if there are several cities with the same name)
- Ease of updating—the name is stored in only one place, so it is easy to update across the board if it ever needs to be changed (e.g., change of a city name due to political events)
- Localization support—when the site is translated into other languages, the standardized lists can be localized, so the region and industry can be displayed in the viewer’s language
- Better search—e.g., a search for philanthropists in the state of Washington can match this profile, because the list of regions can encode the fact that Seattle is in Washington (which is not apparent from the string "Greater Seattle Area")

Relational Model Versus Document Model: Many-to-one and Many-to-Many Relationships

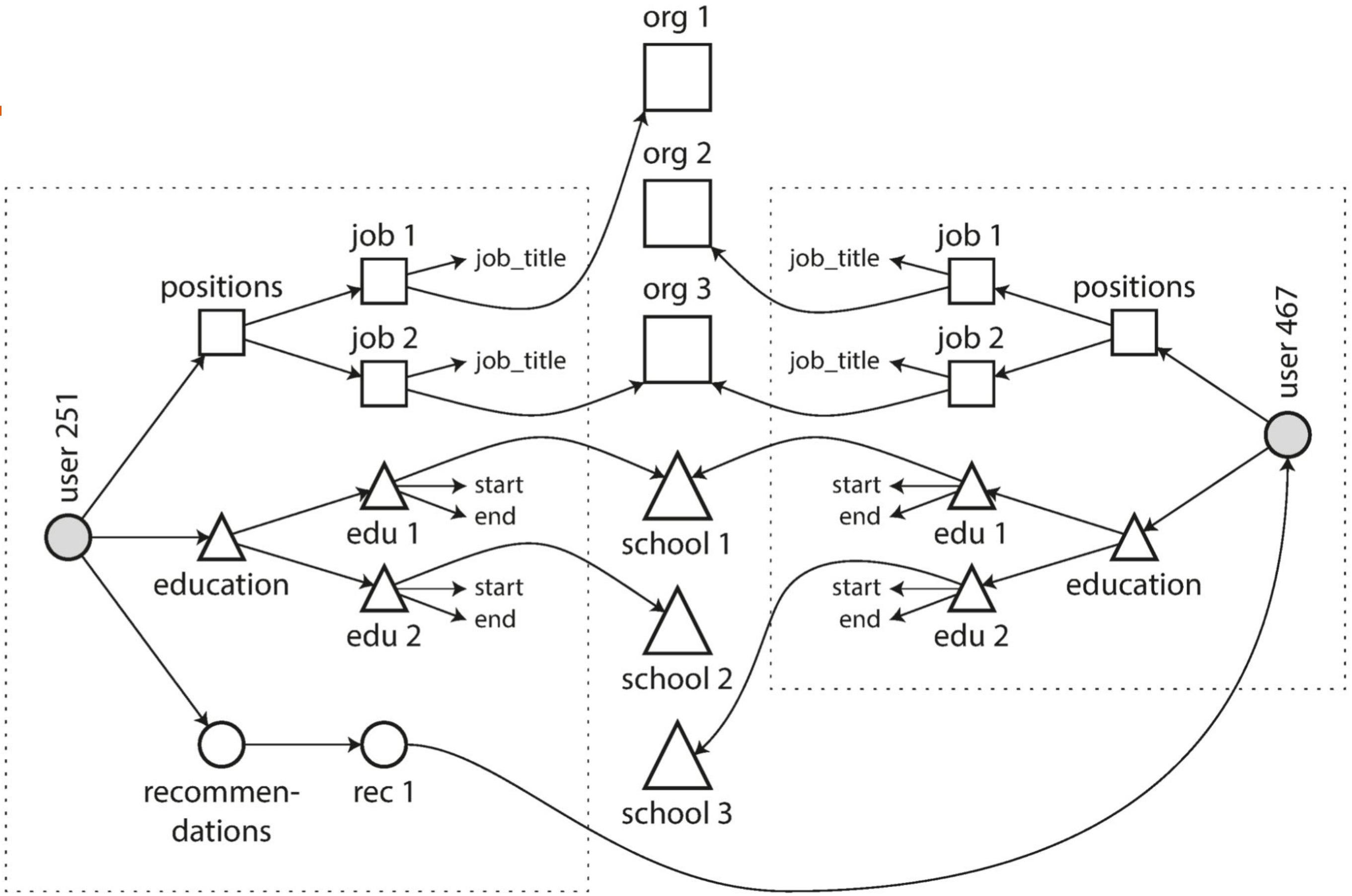
- When you store the text directly, you are duplicating the human-meaningful information in every record that uses it.
- The advantage of using an ID is that because it has no meaning to humans, it never needs to change
- Removing such duplication is the key idea behind **normalization** in databases.

Relational Model Versus Document Model: Many-to-one and Many-to-Many Relationships

Normalizing this data requires many-to-one relationships, which don’t fit nicely into the document model.

Organizations and schools as entities: In the previous description, organization (the company where the user worked) and school_name (where they studied) are just strings. Perhaps they should be references to entities instead?

Recommendations: Say you want to add a new feature: one user can write a recommendation for another user. The recommendation is shown on the résumé of the user who was recommended, together with the name and photo of the user making the recommendation. If the recommender updates their photo, any recommendations they have written need to reflect the new photo.


org 1
org 2
org 3
job 1
job_title
job 2
job_title
positions
user 251
education
edu 1
start
end
edu 2
start
end
recommen-
dations
rec 1
school 1
school 2
school 3
job_title
job 1
job_title
job 2
positions
user 467
start
end
edu 1
start
end
edu 2
education

The data within each dotted rectangle can be grouped into one document, but the references to organizations, schools, and other users need to be represented as references, and require joins when queried.

Relational Model Versus Document Model: Are Document Databases Repeating History?

While many-to-many relationships and joins are routinely used in relational databases, document databases and NoSQL reopened the debate on how best to represent such relationships in a database.

Relational Model Versus Document Model: Are Document Databases Repeating History?

Network Model

The CODASYL(Conference on Data Systems Languages) model was a generalization of the hierarchical model. In the tree structure of the hierarchical model, every record has exactly one parent; in the network model, a record could have multiple parents.

The links between records in the network model were not foreign keys, but more like pointers in a programming language. The only way of accessing a record was to follow a path from a root record along these chains of links. This was called an **access path**. It was difficult to make changes to an application’s data model.

Relational Model Versus Document Model: Are Document Databases Repeating History?

The Relational Model

What the relational model did, by contrast, was to lay out all the data in the open: a relation (table) is simply a collection of tuples (rows). In a relational database, the query optimizer automatically decides which parts of the query to execute in which order, and which indexes to use.

If you want to query your data in new ways, you can just declare a new index, and queries will automatically use whichever indexes are most appropriate. The relational model thus made it much easier to add new features to applications.

You only need to build a query optimizer once, and then all applications that use the database can benefit from it.

Relational Model Versus Document Model: Are Document Databases Repeating History?

Comparison to docuemnt databases

Document databases reverted back to the hierarchical model in one aspect: storing nested records within their parent record rather than in a separate table.

The related item is referenced by a unique identifier, which is called a foreign key in the relational model and a document reference in the document model. That identifier is resolved at read time by using a join or follow-up queries.

Relational Model Versus Document Model: Relational Versus Document Databases Today

Which data model leads to simpler application code?

If the data in your application has a document-like structure -> good idea to use a document model.

Limitation:

Cannot refer directly to a nested item (However, as long as documents are not too deeply nested, that is not usually a problem.)

The poor support for joins in document databases (depending on the application)

many-to-many relationships

Relational Model Versus Document Model: Relational Versus Document Databases Today

Schema flexibility in the document model

Most document databases, and the JSON support in relational databases, do not enforce any schema on the data in documents.

Schema-on-read (Document) is similar to dynamic (runtime) type checking in programming languages.

```
if (user && user.name && !user.first_name) {
    // Documents written before Dec 8, 2013 don't have first_name
    user.first_name = user.name.split(" ")[0];
}
```

Relational Model Versus Document Model: Relational Versus Document Databases Today

Schema flexibility in the document model

Schema-on-write (Relational) is similar to static (compile-time) type checking. Schema changes have a bad reputation of being slow and requiring downtime. Set to its default of NULL and fill it in at read time, like it would with a document database.

```
ALTER TABLE users ADD COLUMN first_name text;
UPDATE users SET first_name = split_part(name, ' ', 1);      -- PostgreSQL
UPDATE users SET first_name = substring_index(name, ' ', 1);      -- MySQL
```

Relational Model Versus Document Model: Relational Versus Document Databases Today

Data localiry for queries

If your application often needs to access the entire document, there is a performance advantage to this **storage locality**. The idea of grouping related data together for locality is not limited to the document model.

Relational Model Versus Document Model: Relational Versus Document Databases Today

Convergence of document and relational databases

It seems that relational and document databases are becoming more similar over time, and that is a good thing: the data models complement each other.

A hybrid of the relational and document models is a good route for databases to take in the future.

Query Languages for Data

An **imperative** language tells the computer to perform certain operations in a certain order.

In a **declarative** query language, like SQL or relational algebra, you just specify the pattern of the data you want—what conditions the results must meet, and how you want the data to be transformed —but not **how** to achieve that goal.

SQL is a **declarative** query language, whereas IMS and CODASYL queried the database using **imperative** code.

Query Languages for Data: Declarative Queries on the Web

CSS/XLS vs DOM API

Query Languages for Data: MapReduce Querying

MapReduce is a programming model for processing large amounts of data in bulk across many machines, popularized by Google.

MapReduce is neither a declarative query language nor a fully imperative query API, but somewhere in between

Graph-Like Data Models

If your application has mostly one-to-many relationships or no relationships between records, the document model is appropriate.

A graph consists of two kinds of objects: **vertices** and **edges**.

Social graphs: Vertices are people, and edges indicate which people know each other.

The web graph: Vertices are web pages, and edges indicate HTML links to other pages.

Road or rail networks: Vertices are junctions, and edges represent the roads or railway lines between them.

Graph-Like Data Models: Property Graphs

Each vertex consists of:

- A unique identifier
- A set of outgoing edges
- A set of incoming edges
- A collection of properties (key-value pairs)

Each edge consists of:

- A unique identifier
- The vertex at which the edge starts (the tail vertex)
- The vertex at which the edge ends (the head vertex)
- A label to describe the kind of relationship between the two vertices
- A collection of properties (key-value pairs)

Graph-Like Data Models: Property Graphs

Some important aspects of this model are:

- Any vertex can have an edge connecting it with any other vertex. There is no schema that restricts which kinds of things can or cannot be associated.
- Given any vertex, you can efficiently find both its incoming and its outgoing edges, and thus **traverse** the graph—i.e., follow a path through a chain of vertices —both forward and backward.
- By using different labels for different kinds of relationships, you can store several different kinds of information in a single graph, while still maintaining a clean data model.

Graph-Like Data Models: The Cypher Query Language

Cypher is a declarative query language for property graphs, created for the Neo4j graph database.

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```

find people who emigrated from the US to Europe

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

Graph-Like Data Models: Graph Queries in SQL

Yes, but with some difficulty.

Graph-Like Data Models: Triple-Stores and SPARQL

The triple-store model is mostly equivalent to the property graph model, using different words to describe the same ideas.

In a triple-store, all information is stored in the form of very simple three-part statements: (**subject**, **predicate**, **object**). The subject of a triple is equivalent to a vertex in a graph.

Graph-Like Data Models: Triple-Stores and SPARQL

The semantic Web

The triple-store data model is completely independent of the semantic web—for example, Datomic is a triple-store that does not claim to have anything to do with it.

Graph-Like Data Models: Triple-Stores and SPARQL

The RDF data Model

The Resource Description Framework (RDF) was intended as a mechanism for different websites to publish data in a consistent format, allowing data from different websites to be automatically combined into a web of data—a kind of internet-wide “database of everything.”

Graph-Like Data Models: Triple-Stores and SPARQL

The SPARQL query language

SPARQL is a query language for triple-stores using the RDF data model. SPARQL is a nice query language—even if the semantic web never happens, it can be a powerful tool for applications to use internally.

Summary

Historically, data started out being represented as one big tree (the hierarchical model), but that wasn’t good for representing many-to-many relationships, so the relational model was invented to solve that problem.

New nonrelational “NoSQL” datastores have diverged in two main directions:

1. **Document databases** target use cases where data comes in self-contained documents and relationships between one document and another are rare.

2. **Graph databases** go in the opposite direction, targeting use cases where anything is potentially related to everything.

One thing that document and graph databases have in common is that they typically don’t enforce a schema for the data they store, which can make it easier to adapt applications to changing requirements. The schema is explicit (enforced on write) or implicit (handled on read).