Data storage, formats

Topics for today

In today's class, we'll look at:

- data storage
- data formats

Once we collect/acquire data, these are what we need to deal with - what storage scheme to pick, and what storage format to use.

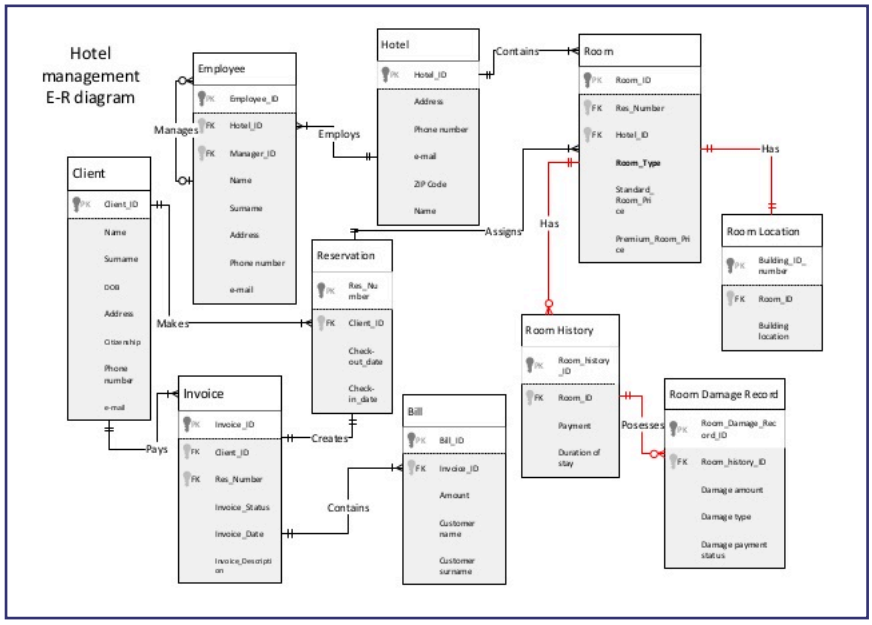
Data models -> databases

Previously, we looked at data representation (data models). The choice of data model directly translates to the type of database we will need to use, to store our data.

data model: relations

As we saw, a robust, popular way to model data is to use 'entities', and connect them using 'relations'. Diagrams that show such modeling are called 'E-R diagrams' (entity-relationship diagrams)

For example, this is an E-R diagram for a hotel management system:



In the above, each rectangle is an entity, and each wire (connection) is a relationship.

Relational DBs

An E-R diagram is directly 'translatable' to a RELATIONAL DATABASE, where each entity in the diagram becomes a table (rectangular arrangement of data, with each row containing a single 'record' (a student, a driver's license, a bank account...), with the columns representing attributes or properties (eg. for a student table, these would be studentID, name, major, GPA, graduation date, etc.).

From a Microsoft doc: 'Relational databases organize data as a series of two-dimensional tables with rows and columns. Each table has its own columns, and every row in a table has the same set of columns.'

SQL

So, how do we:

- CREATE blank tables from entities in an E-R diagram, and relate them?
- POPULATE the tables (add rows of data)?
- EDIT the data?
- QUERY/SEARCH the data?

Answer: SQL (Structured Query Language).

For example, below is an E-R diagram, and SQL syntax for table creation:

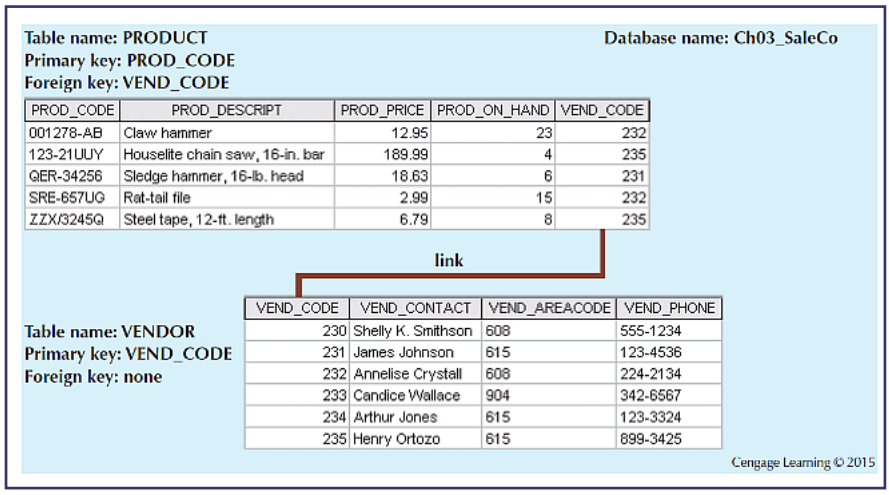


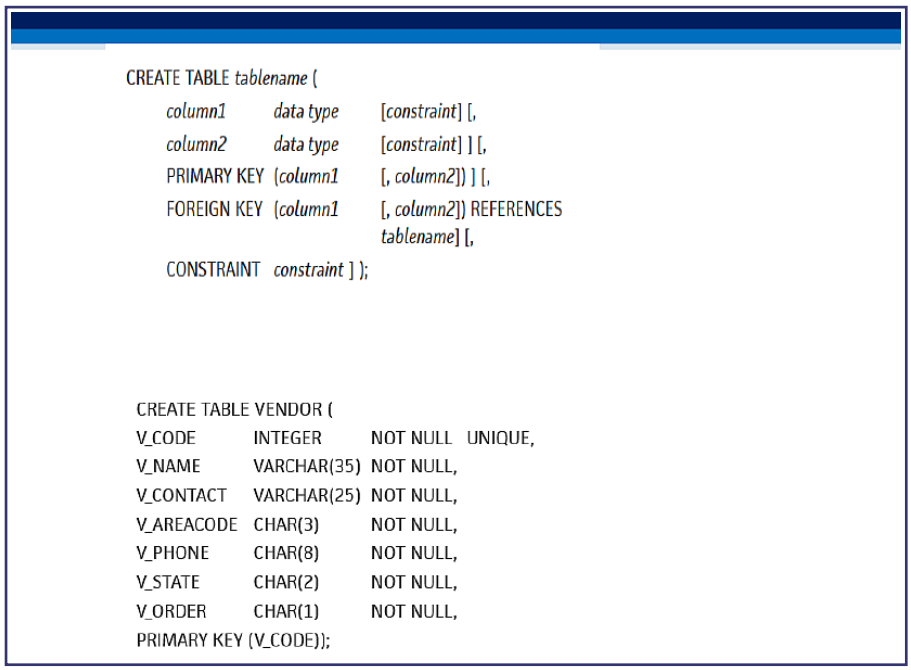
Table name: PRODUCT — Database name: Ch03_SaleCo
Primary key: PROD_CODE
Foreign key: VEND_CODE

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425



```
CREATE TABLE tablename (
    column1     data type     [constraint] [,
    column2     data type     [constraint] ] [,
    PRIMARY KEY (column1      [, column2]) ] [,
    FOREIGN KEY (column1      [, column2]) REFERENCES
                              tablename] [,
    CONSTRAINT  constraint ] );
```

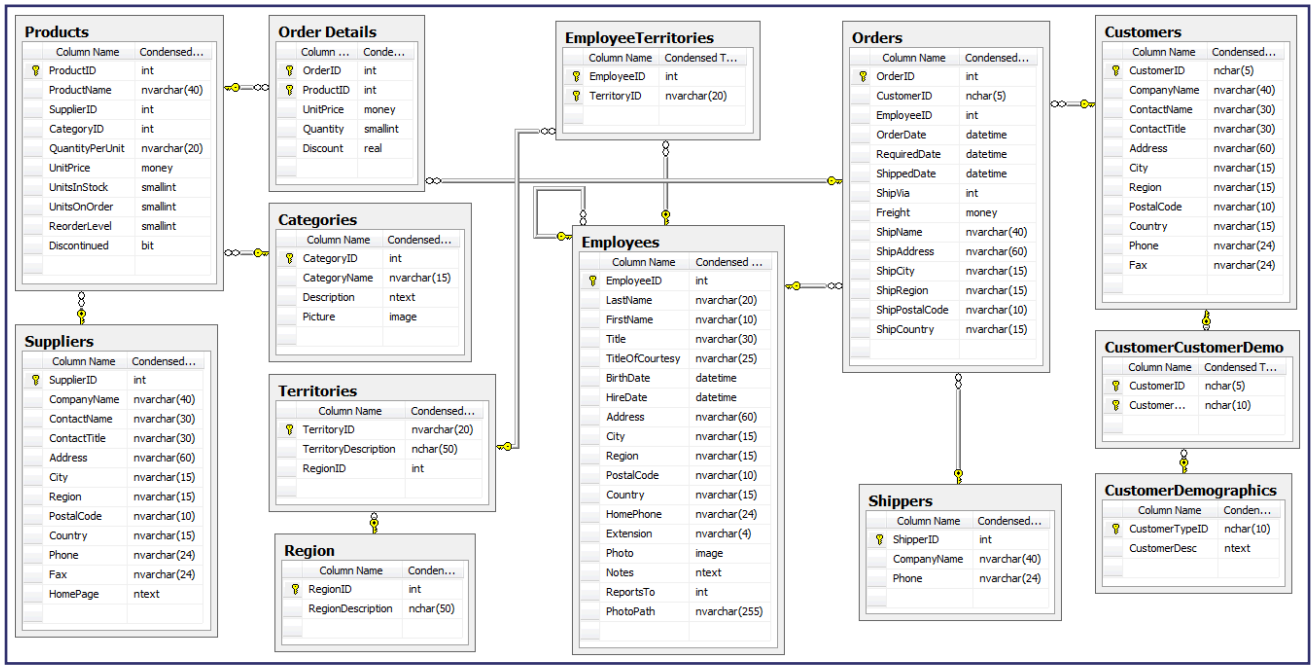
```
CREATE TABLE VENDOR (
V_CODE       INTEGER      NOT NULL  UNIQUE,
V_NAME       VARCHAR(35)  NOT NULL,
V_CONTACT    VARCHAR(25)  NOT NULL,
V_AREACODE   CHAR(3)      NOT NULL,
V_PHONE      CHAR(8)      NOT NULL,
V_STATE      CHAR(2)      NOT NULL,
V_ORDER      CHAR(1)      NOT NULL,
PRIMARY KEY (V_CODE));
```

The 'CREATE TABLE' command would create us a blank table, with the columns specified - now we can fill it with data, edit it, and most importantly, query it (all done using SQL as well).

SQL online: w3schools

A great way to learn a lot of SQL is to use the w3schools SQL tutorial.

Do spend some time, going through the (interactive) tutorial! The tutorial uses a small but adequate sample DB called the Northwind database [comes pre-filled with data], that ships with Microsoft's db products:



SQL online: sqlfiddle

Another online SQL learning platform is sqlfiddle, where you can create tables and connect them, fill them with data, and query them.

For example, here is a sample 'fiddle' I created (some tables, data, and queries). Play around with these, and modify the data (eg. add more rows, delete rows...).

SQL is **very** worth knowing, as a data science practitioner!

Oracle

You can install Oracle on your own machine, for creating and populating tables and querying them - Oracle is the most popular relational DB out there.

SQL Server

Microsoft's 'SQL Server' is a popular alternative to Oracle.

sqlite

sqlite is a tiny, mighty and heavily used relational DB (Android devices use it, for example!), where you can query data not only using SQL, but also using programming languages such as Java and Python; you can even read the raw binary data using C++ or Java.

Eg. using Python, you can do:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print row

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)
```

sqlite browser

'DB Brower for SQLite', aka 'sqlite browser' is a nice GUI add-on to sqlite, where you can do DB operations using UI and/or raw SQL.

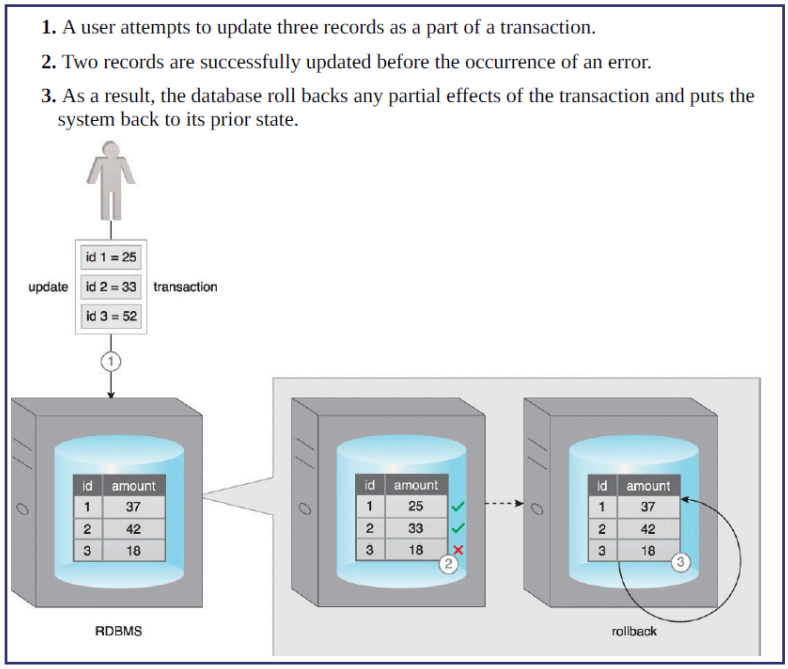
Here is a quickstart guide for SQLite Browser.

'ACID' property

ACID is a database design principle related to transaction management [be sure to read up more on this]. A 'transaction' consists of a series of table reads and writes (that can involve multiple tables). ACID stands for:

- atomicity
- consistency
- isolation
- durability

For example, 'atomicity' means that partial transactions are aborted:



The ACID set of properties are heavily enforced by relational DBs - in the modern world of Internet-oriented DBs, this is a liability, not an asset!

data model: documents

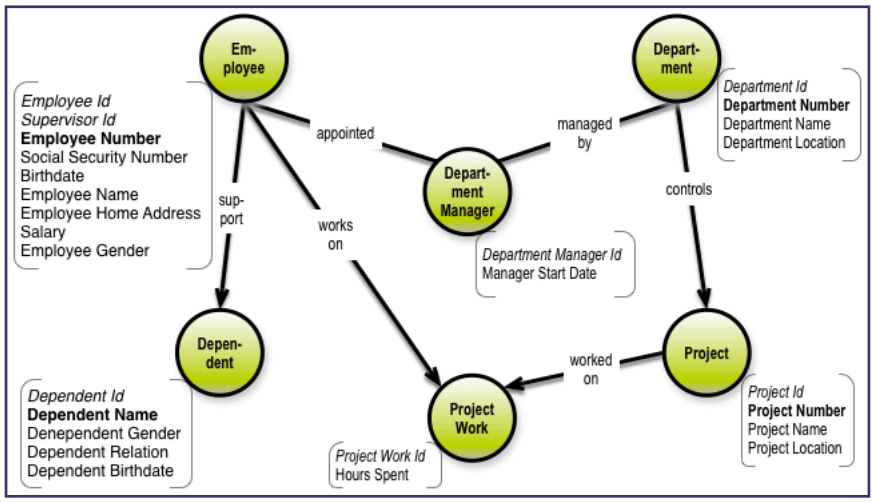
Earlier we saw that data (eg. a customer's contact info) can be modeled as a collection of 'documents':

```
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "addresses": [
        {
            "line1": "100 Some Street",
            "line2": "Unit 1",
            "city": "Seattle",
            "state": "WA",
            "zip": 98012
        }
    ],
    "contactDetails": [
        {"email": "thomas@andersen.com"},
        {"phone": "+1 555 555-5555", "extension": 5555}
    ]
}
```

The above shows a single 'document' (that contains nested data).

data model: graphs

As you know, data can also be modeled as a graph:



Non-relational ('NoSQL') DBs

The newer types of databases that store data based on the document and graph models, are called 'NoSQL' dbs. These are VERY different from relational DBs, in terms of how they internally structure data, which in turn leads to their offering very different ways to search/access the data.

MongoDB

MongoDB [huMONGOus!] is a very popular NoSQL document db.

CouchDB

CouchDB is another popular document db [couch: 'cluster of unreliable commodity hardware' :)].

Querying Couch

Rather than using SQL, NoSQL DBs such as Mongo and Couch allow us to use programming languages such as JavaScript, for querying the document data. For example, here is syntax used in Couch:

Imagine you have a database with user records and you want a view of those records using the last name of each user as keys.

```
function(doc) {
    if (doc.last_name) {
        emit(doc.last_name, doc);
    }
}
```

The above map function will produce and index something like the one below. Because `doc` is used as a value for each entry the entire content of each JSON document will be accessible as the indexed values.

```
[
  ...
  { key: "Clarke", value: { last_name: "Clarke", ... } },
  { key: "Kelly",  value: { last_name: "Kelly",  ... } },
  { key: "Smith",  value: { last_name: "Smith",  ... } },
  ...
]
```

neo4j

neo4j is the most popular graph DB. You can download an eval copy for local use, or try their online sandbox.

Apache Giraph

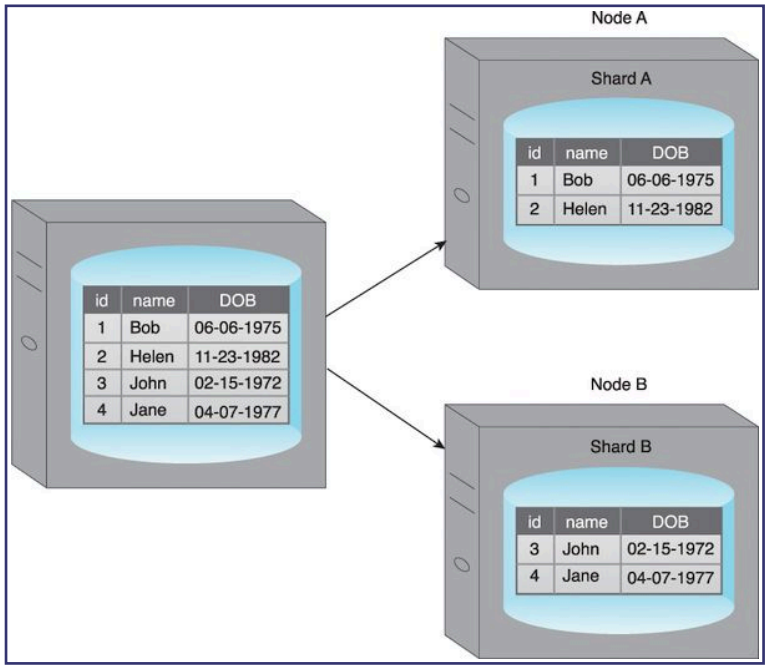
Giraph is an open source graph DB.

Gremlin/TinkerPop

Rather than learn different query languages for different graph DBs [eg. neo4j uses 'cypher'], it would be ideal if a single language could be used on all graph DBs - TinkerPop is that language.

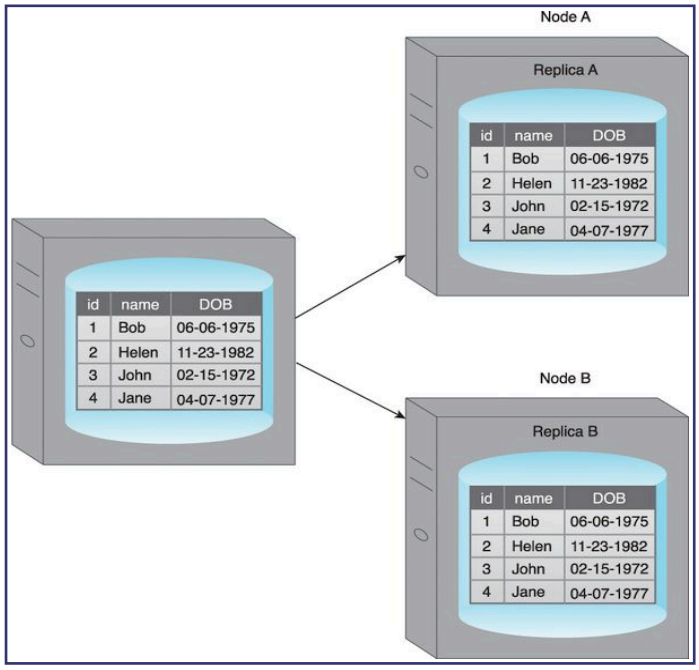
Sharding

Sharding is a way to DISTRIBUTE a table (or a collection of documents) across multiple 'nodes' (machines on a network), using 'horizontal fragmentation' (splitting a single collection of rows/docs into multiple ones).



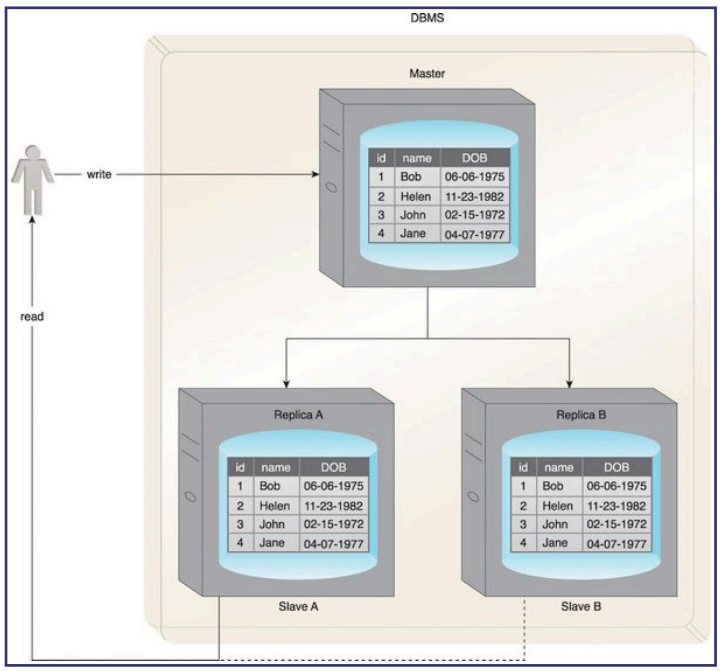
Replication

Replication is where we create multiple copies of tables (or shards), distributed across several nodes - this is done for speeding up access, and as a fault-tolerance mechanism.



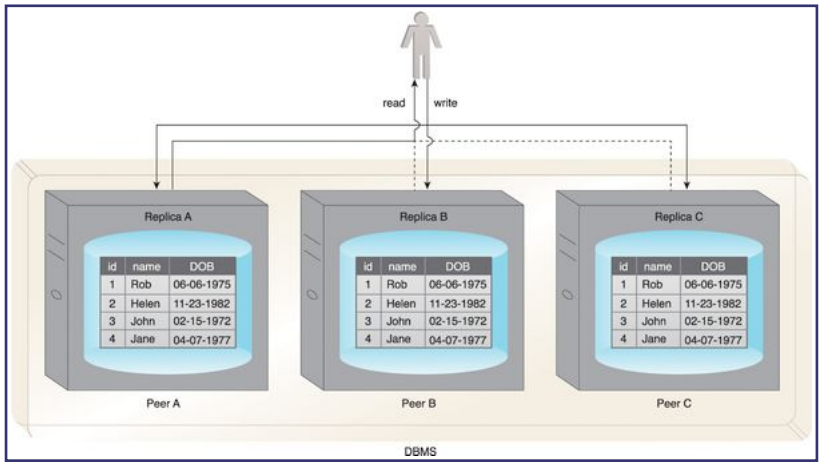
Master/slave config

Master-slave is a replication strategy - updates are done to a master copy, which broadcasts the changes to connected slaves.



Peer to peer config

Peer to peer replication is where there isn't a master/slave config - any node (copy) can be updated, and the changes sent to others.



'BASE' property

NoSQL DBs feature 'BASE' properties, as opposed to relational DBs' 'ACID' - in 'BASE', availability is prioritized over consistency.

Storage formats

Data, regardless of the type of db (relational or NoSQL) needs to stored on disk or memory, for access and analysis. Data 'formats' are specific conventions for storage.

A good data format needs to satisfy three requirements. From Skiena's book:

The best computational data formats have several useful properties:

- *They are easy for computers to parse*: Data written in a useful format is destined to be used again, elsewhere. Sophisticated data formats are often supported by APIs that govern technical details ensuring proper format.
- *They are easy for people to read*: Eyeballing data is an essential operation in many contexts. Which of the data files in this directory is the right one for me to use? What do we know about the data fields in this file? What is the gross range of values for each particular field?

  These use cases speak to the enormous value of being able to open a data file in a text editor to look at it. Typically, this means presenting the data in a human-readable text-encoded format, with records demarcated by separate lines, and fields separated by delimiting symbols.

- *They are widely used by other tools and systems*: The urge to invent proprietary data standard beats firmly in the corporate heart, and most software developers would rather share a toothbrush than a file format. But these are impulses to be avoided. The power of data comes from mixing and matching it with other data resources, which is best facilitated by using popular standard formats.

CSV

CSV [comma separated value] is one of the simplest forms of data storage - they are pretty much tabular.

Sample CSV

Here are sample CSV files. Meta note - this is a *****VERY*** powerful search technique :) :)**

arff

ARFF is a popular format used in machine learning.

XML

XML, by definition, is 'extensible' it can be used to describe data in a hierarchical manner, using HTML-like 'tags'. Here is an intro'.

Sample XML

Here are many XML files...

JSON

JSON is a lightweight XML 'replacement' - using nothing more than , : {} [] it can represent deep hierarchies as well. We looked at JSON earlier:

```
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "addresses": [
        {
            "line1": "100 Some Street",
            "line2": "Unit 1",
            "city": "Seattle",
            "state": "WA",
            "zip": 98012
        }
    ],
    "contactDetails": [
        {"email": "thomas@andersen.com"},
        {"phone": "+1 555 555-5555", "extension": 5555}
    ]
}
```

For larger documents, 'BSON' (Binary JSON) is preferable since it saves storage space.

RDF

RDF is a format used to describe graph data (specifically, 'RDF triples' are used). In other words, 'triples' (subject-predicate-object) are storable in the RDF format. Also, triples can also be stored using two other formats: N-Triples, and Turtle (a subset of Notation3).

Here is a graph viewer, where the data is specified as RDF triples in one of several formats (eg. 'Turtle'); https://docs.stardog.com/tutorials/rdf-graph-data-model contains a graph of Turtle format RDF triples (for use with the viewer).

Summary

We looked at a variety of storage-related concepts, and at the most popular storage formats.