Real-World NoSQL Schema Design

Tugdual Grall

June 7th 2016

{“about” : “me”}

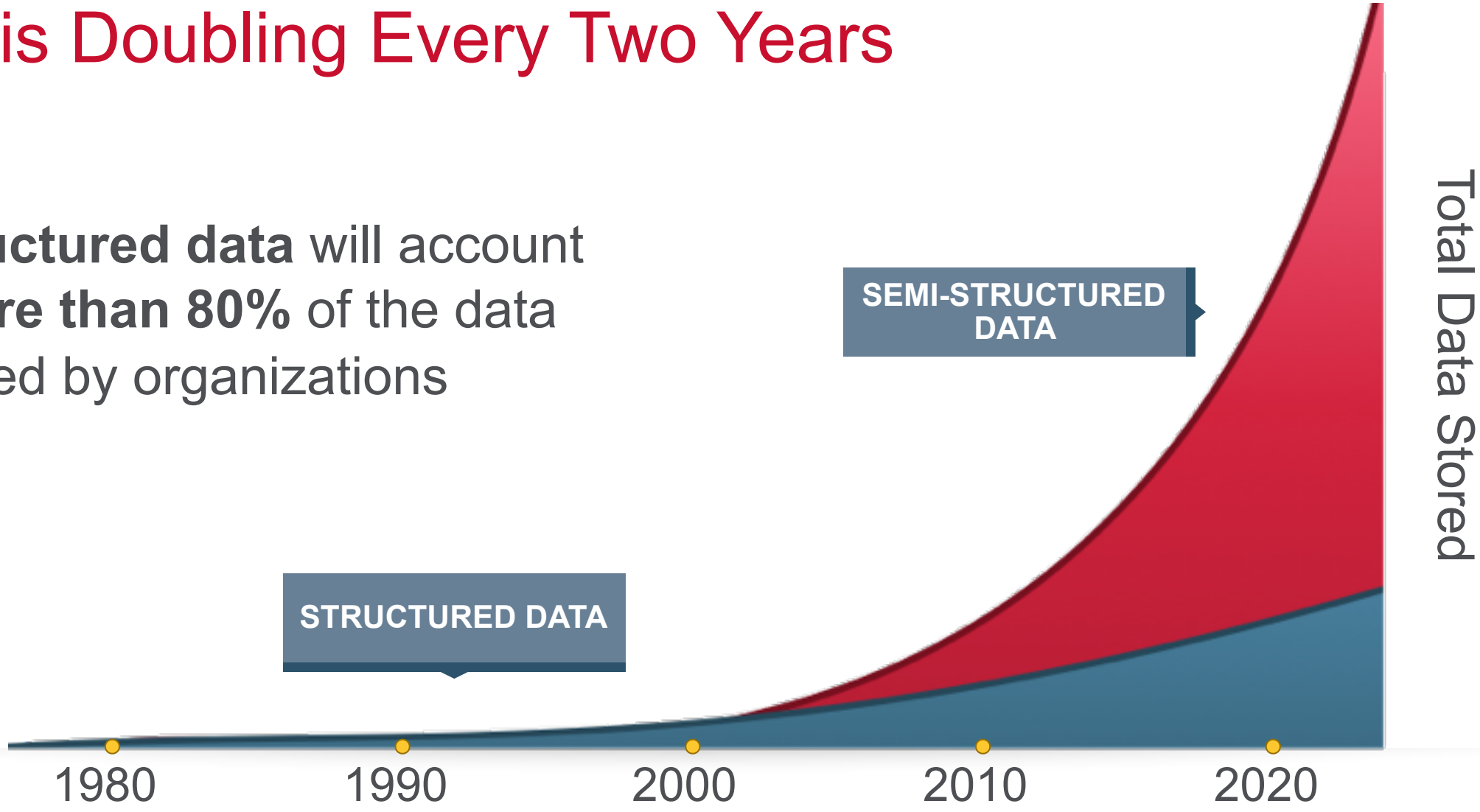
Tugdual “Tug” Grall

- MapR
  - Technical Evangelist
- MongoDB
  - Technical Evangelist
- Couchbase
  - Technical Evangelist
- eXo
  - CTO
- Oracle
  - Developer/Product Manager
  - Mainly Java/SOA
- Developer in consulting firms

- Web
  - @tgrall
  - http://tgrall.github.io
  - tgrall
- NantesJUG co-founder
- Pet Project :
  - http://www.resultri.com
- tug@mapr.com
- tugdual@gmail.com

Data is Doubling Every Two Years

Unstructured data will account for **more than 80%** of the data collected by organizations



Source: Human-Computer Interaction & Knowledge Discovery in Complex Unstructured, Big Data

Big Datastore

Distributed File System
HDFS/MapR-FS

NoSQL Database
HBase/MapR-DB

Store data as File or Row?

HDFS / MapR-FS

- Data stores as “files”
- Fast with Large Scans
- Slow random read/writes

HBase/MapR-DB

- Data stores as row/documents
- Fast with random read/writes

NoSQL Database

Database Schema Design

“ERD” - Logical

For RDBMS:

Entities => Tables

Attributes => Columns

Relationships => Foreign Keys

Many-to-Many => Junction Table

Contrast Relational and HBase Style noSQL

Relational

- Rows containing fields
- Fields contain primitive types
- Structure is fixed and uniform
- Structure is pre-defined
- Referential integrity (optional)

- Expressions over sets of rows

HBase / MapR DB

- Rows contain fields
- Fields bytes
- Structure is flexible
- No pre-defined structure
- Single key
- *Column families*
- *Timestamps*
- *Versions*

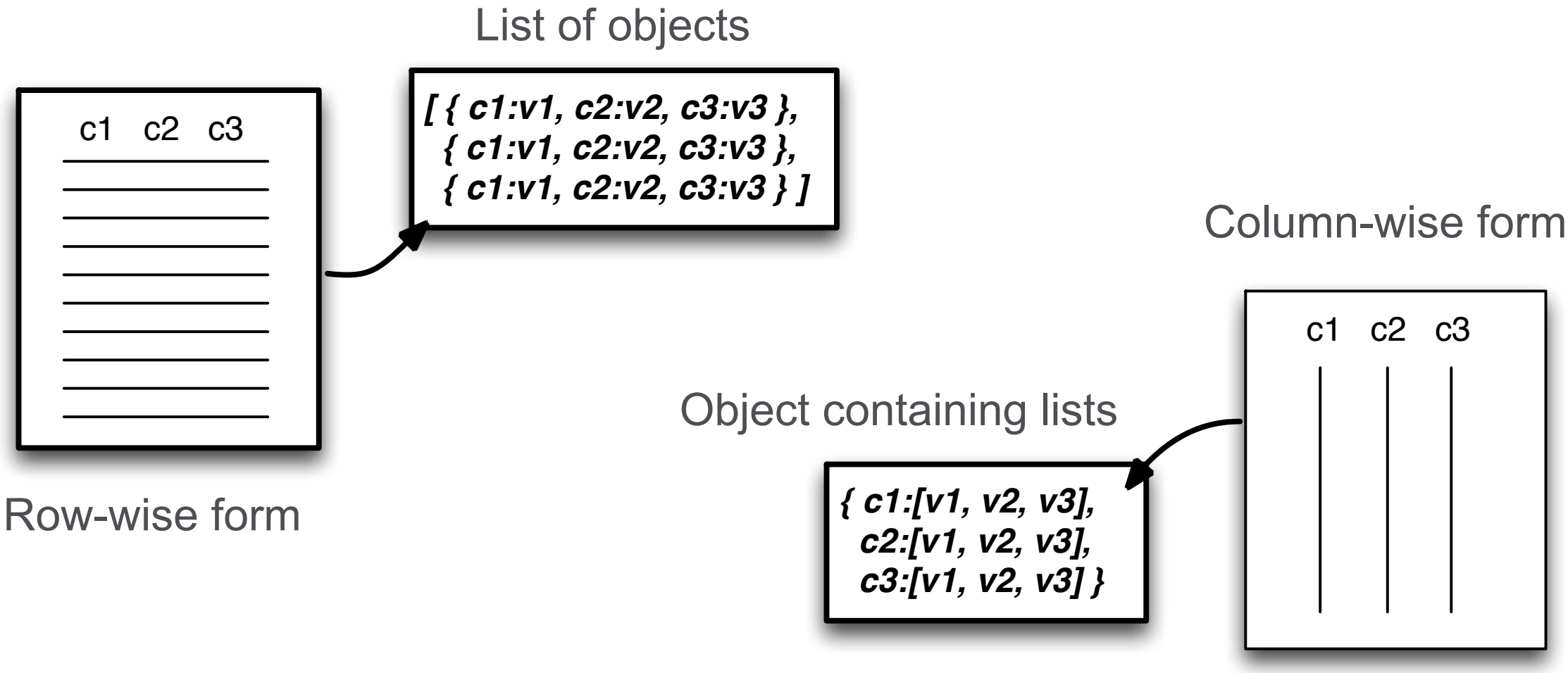
Mix Models for Databases

- Allows complex objects in field values
  - JSON style lists and objects
- Allow references to objects via join
  - Includes references localized within lists
- Lists of objects and objects of lists are isomorphic to tables so …

- Complex data in tables,
- But also tables in complex data,
- Even tables containing complex data containing tables

A Catalog of NoSQL Idioms

Tables as Objects, Objects as Tables



A first example: Time-series data

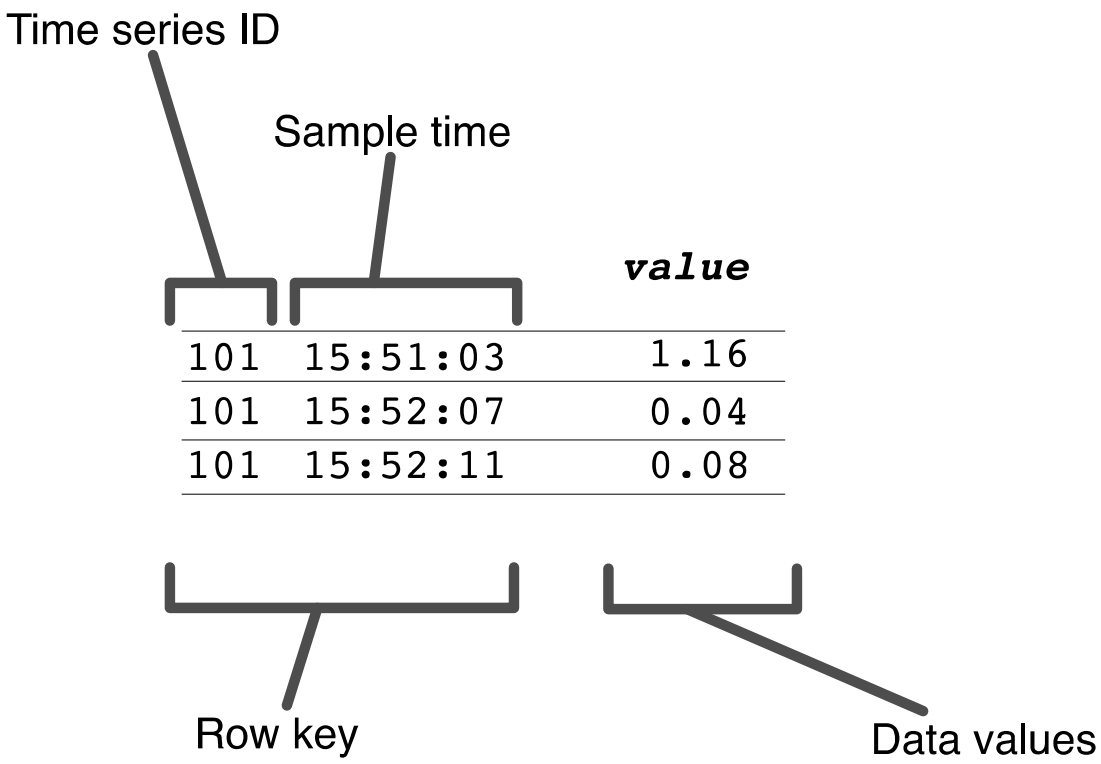
Column names as data

- When **column names** are not pre-defined, they can convey **information**

- Examples
  - Time offsets within a window for time series
  - Top-level domains for web crawlers
  - Vendor id's for customer purchase profiles

- Predefined schema is impossible for this idiom

Relational Model for Time-Series



NoSQL Table Design: Point-by-Point

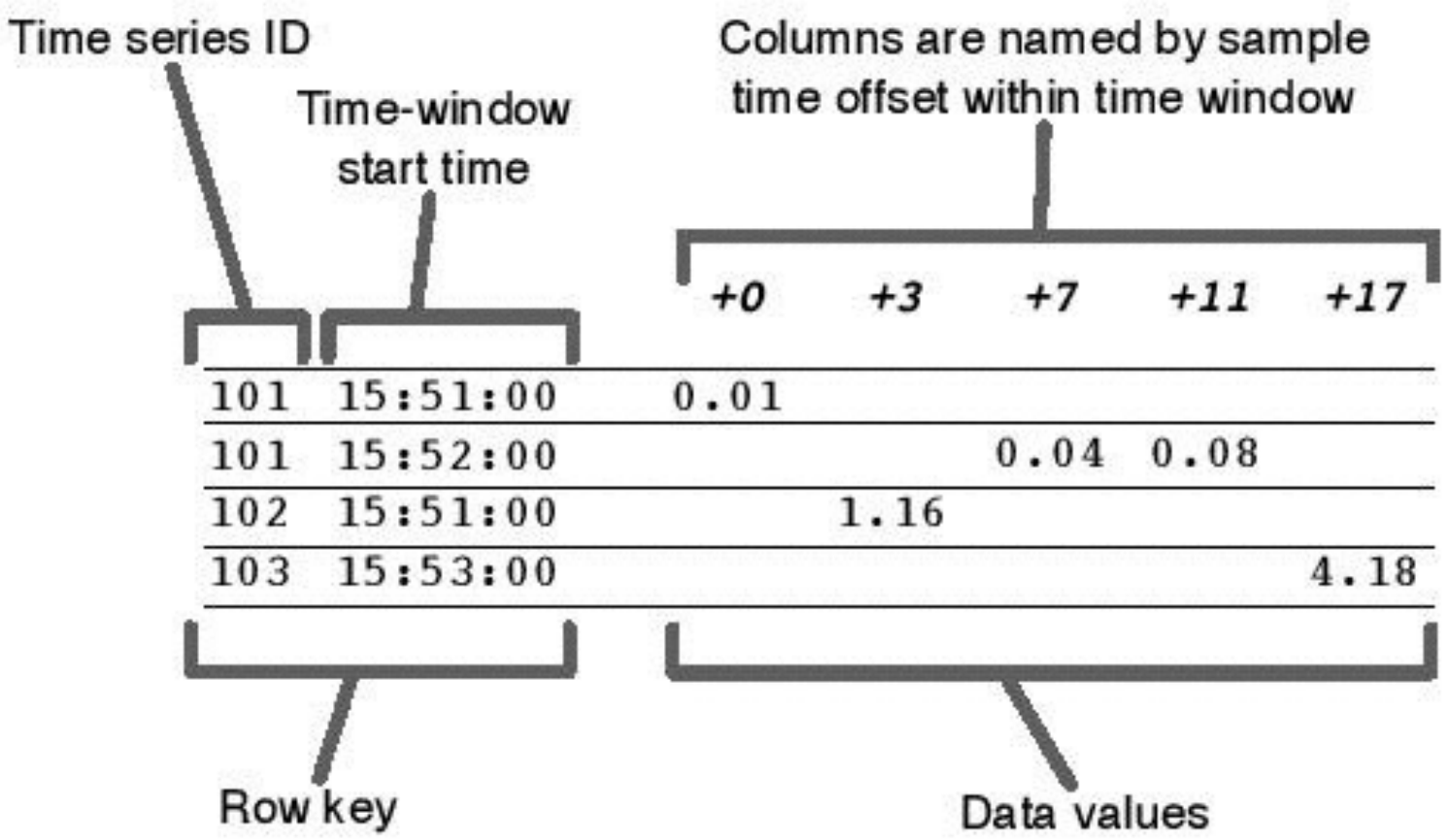
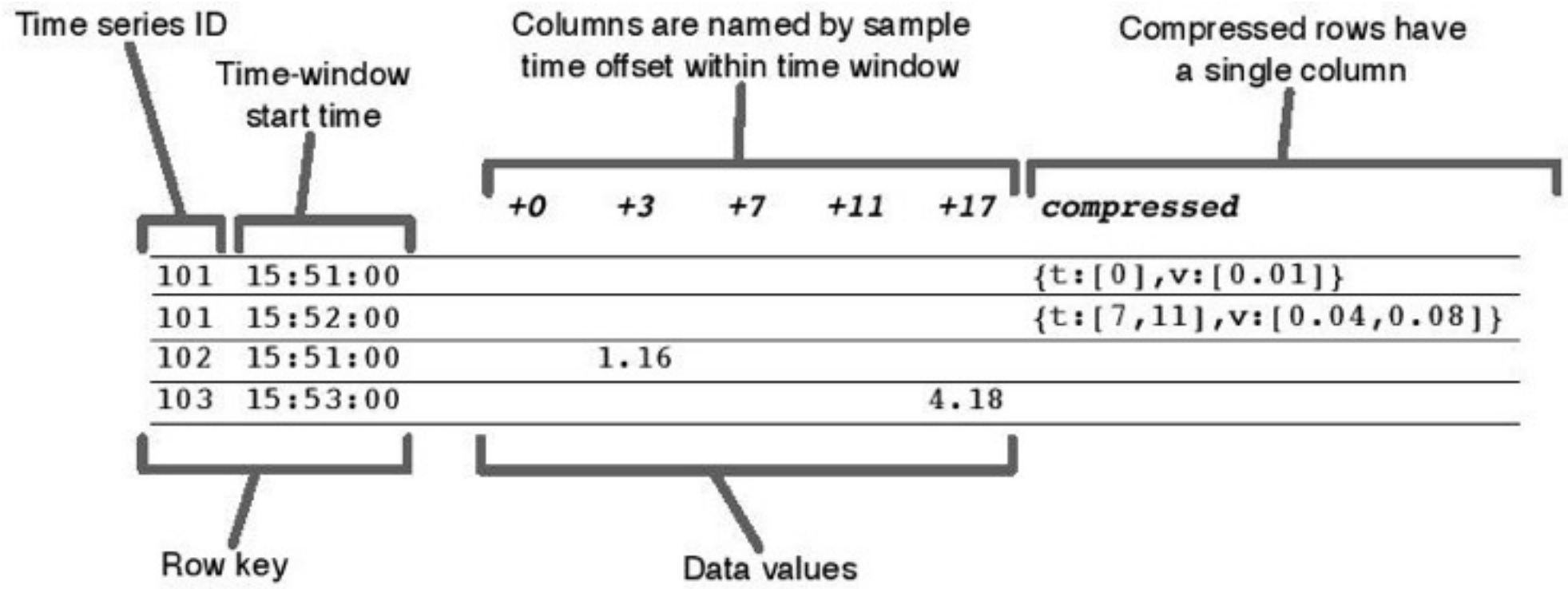


Table Design: Hybrid Point-by-Point + Sub-table



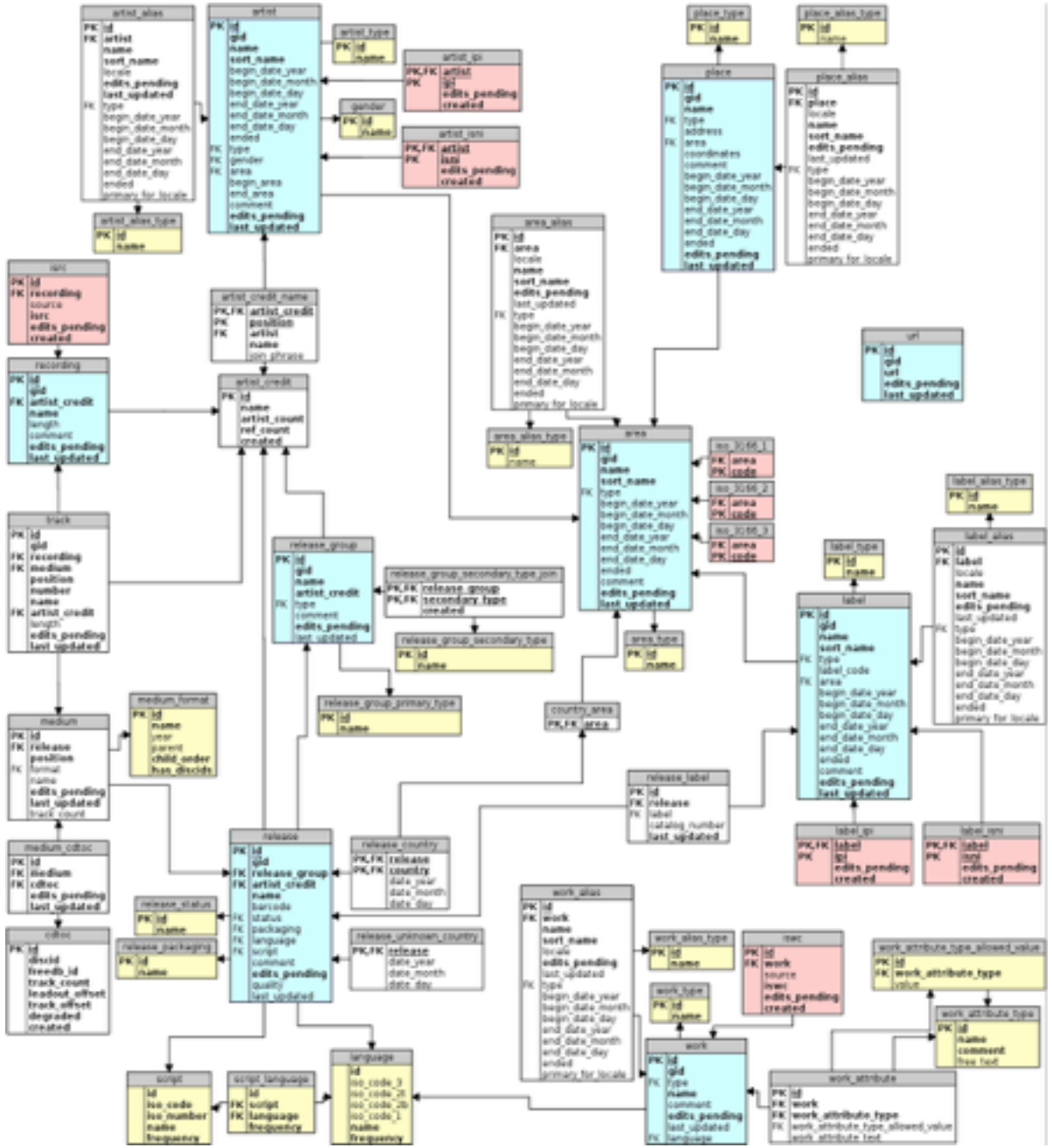
After close of window, data in row is restated as column-oriented tabular value in different column family.

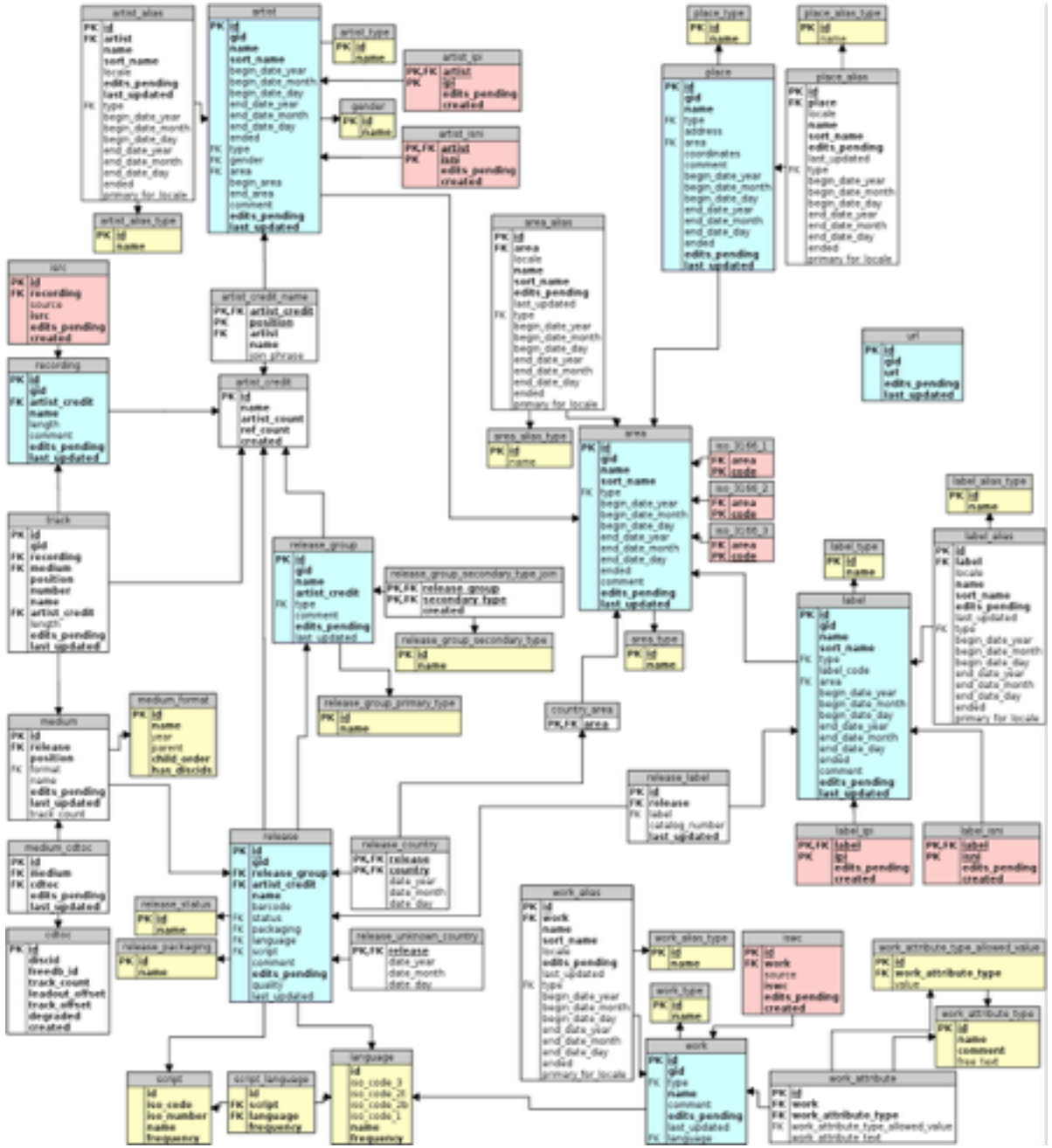
A second example: Music Application

https://musicbrainz.org/

MusicBrainz on NoSQL

- Artists, albums, tracks and labels are key objects
- Reality check:
  - Add works (compositions), recordings, release, release group
- 7 tables for artist alone
- 12 for place, 7 for label, 17 for release/group, 8 for work
  - (but only 4 for recording!)
  - Total of 12 + 7 + 17 + 8 + 4 = 48 tables
- But wait, there's more!
  - 10 annotation tables, 10 edit tables, 19 tag tables, 5 rating tables, 86 link tables, 5 cover art tables and 3 tables for CD timing info (138 total)
  - And 50 more tables that aren't documented yet

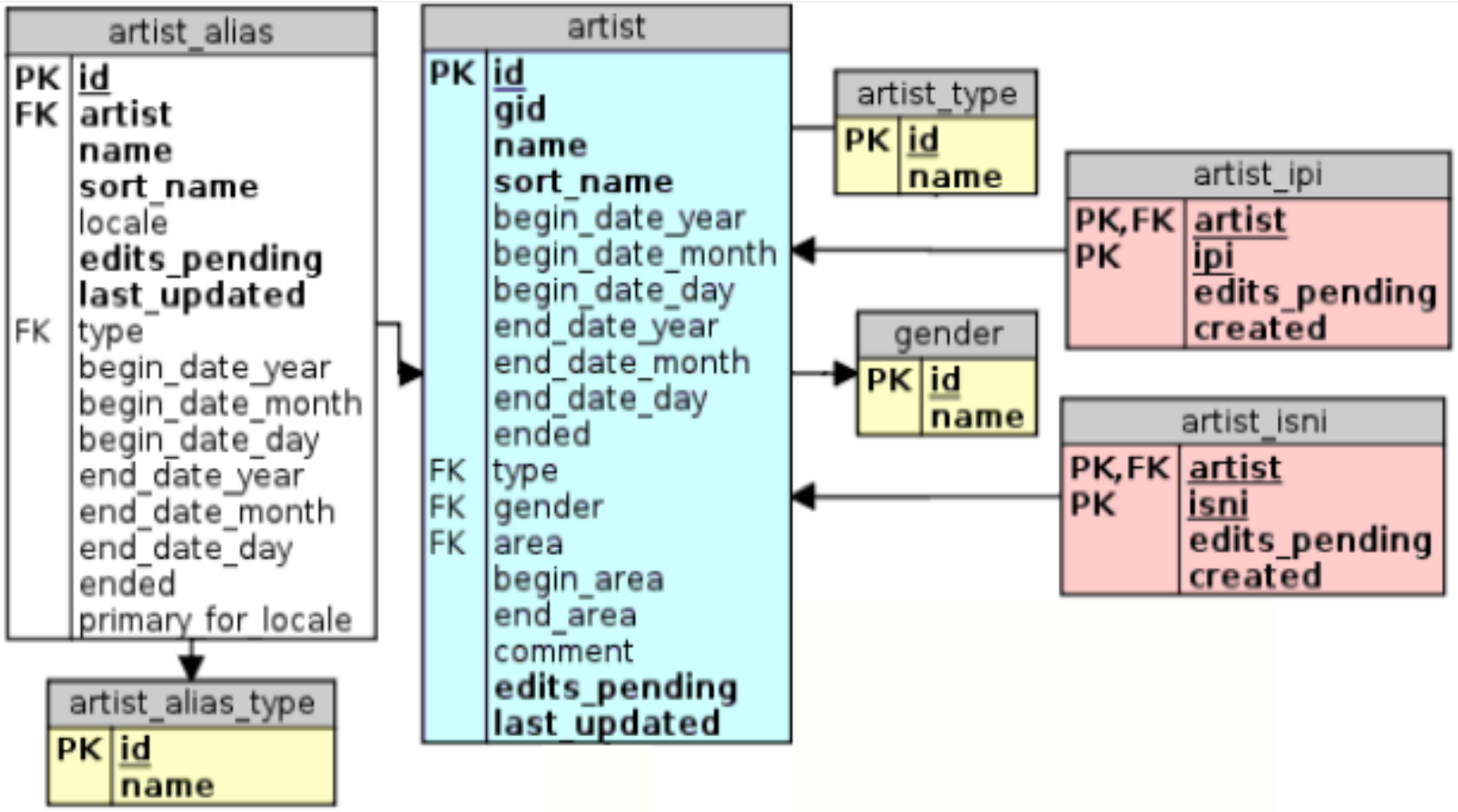


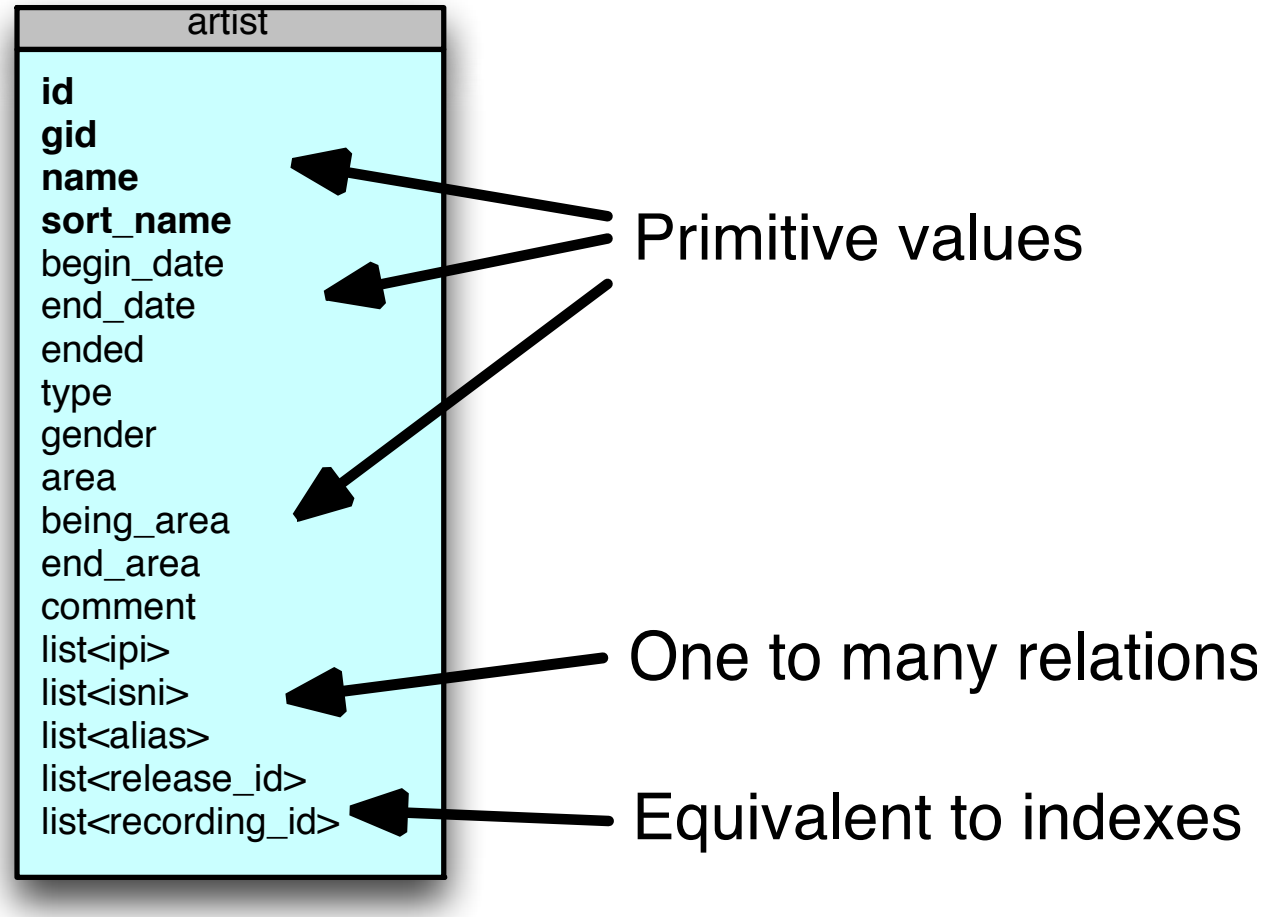


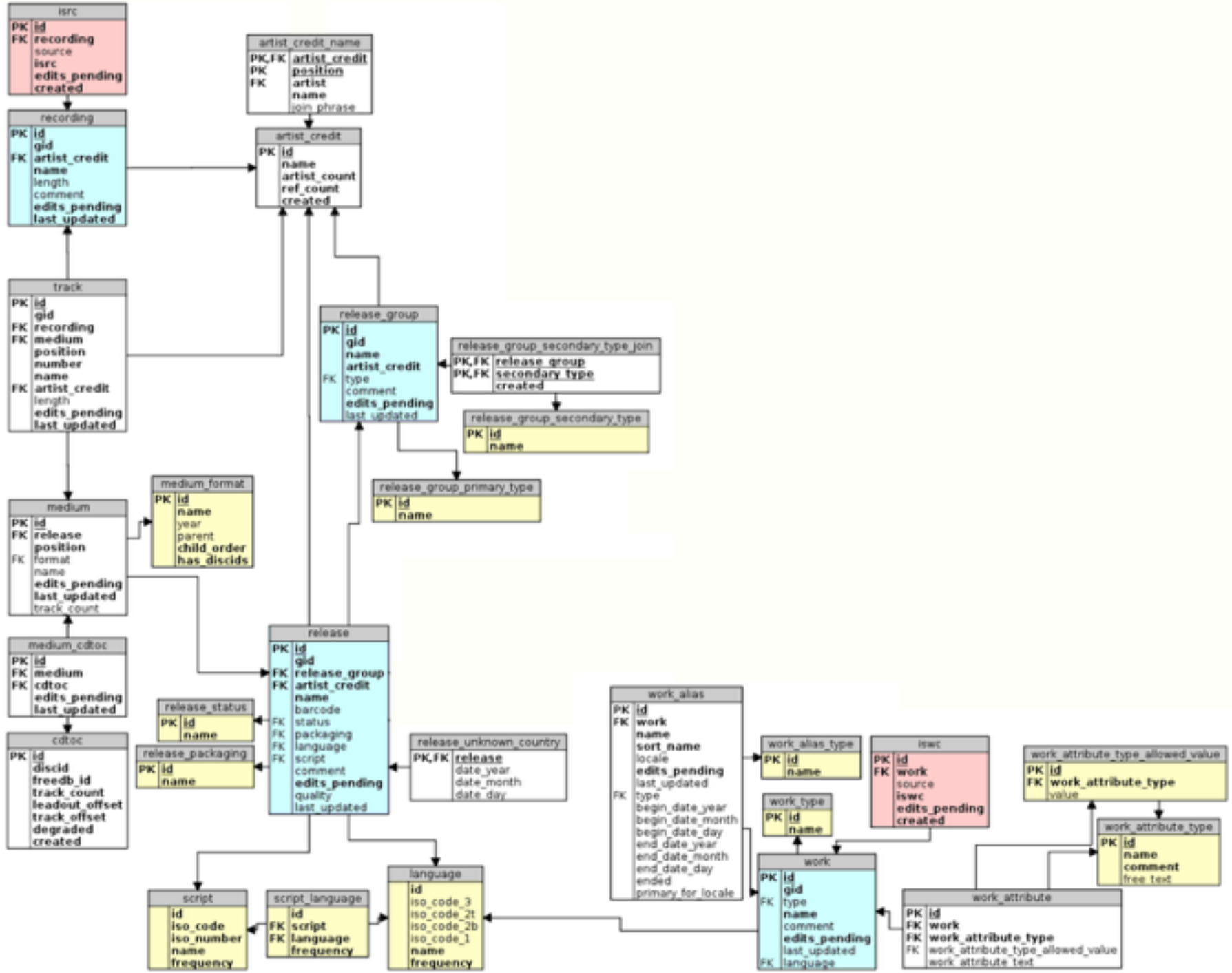
180 tables
not shown

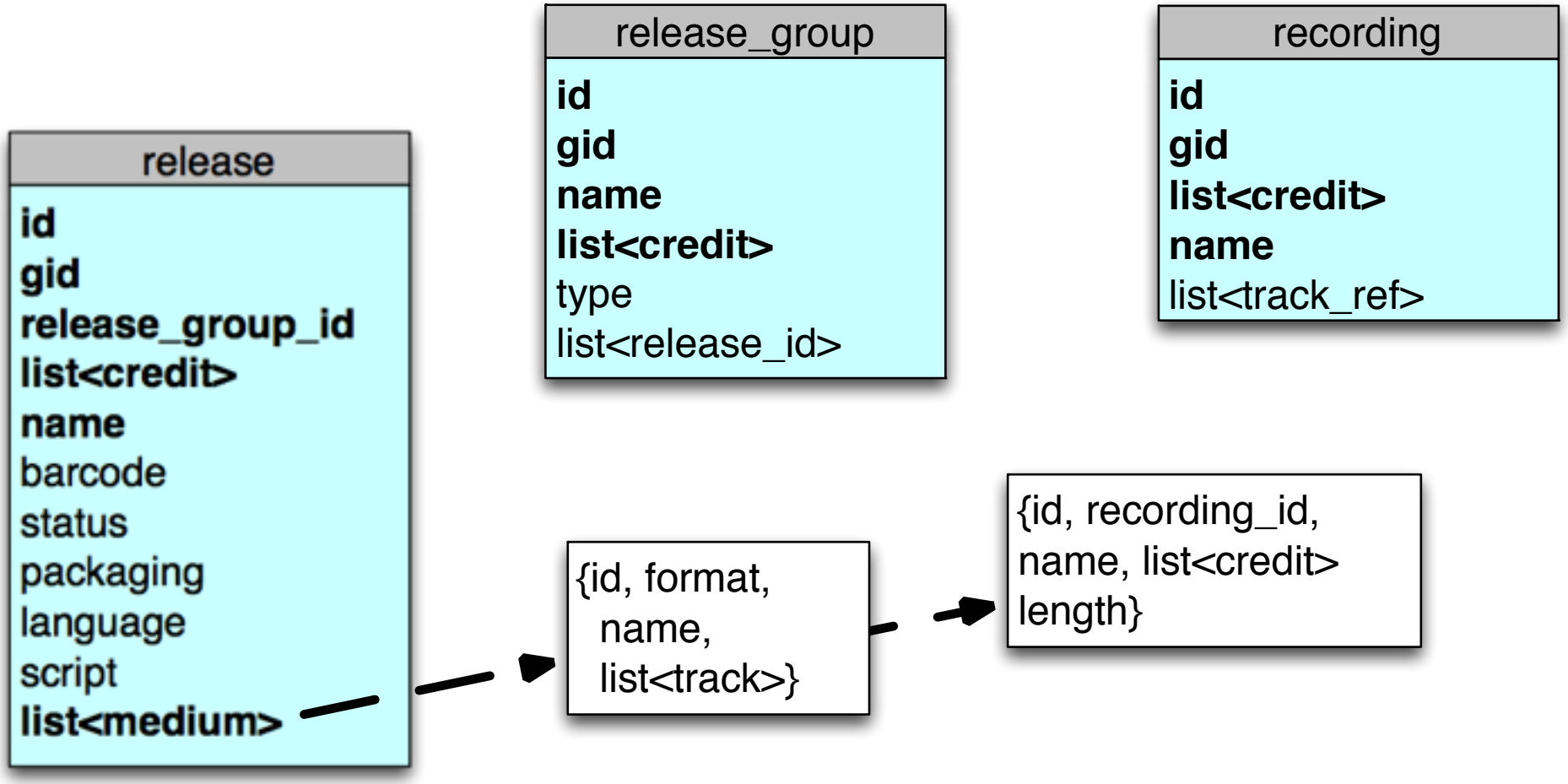
236 tables
to describe 7 kinds of things

Can we do better?






isrc
PK id
FK recording
source
isrc
edits_pending
created
artist_credit_name
PK,FK artist_credit
PK position
FK artist
name
join_phrase
recording
PK id
gid
FK artist_credit
name
length
comment
edits_pending
last_updated
artist_credit
PK id
name
artist_count
ref_count
created
track
PK id
gid
FK recording
FK medium
position
number
name
FK artist_credit
length
edits_pending
last_updated
release_group
PK id
gid
name
artist_credit
FK type
comment
edits_pending
last_updated
release_group_secondary_type_join
PK,FK release_group
PK,FK secondary_type
created
release_group_secondary_type
PK id
name
release_group_primary_type
PK id
name
medium_format
PK id
name
year
parent
child_order
has_discids
medium
PK id
FK release
position
FK format
name
edits_pending
last_updated
track_count
medium_cdtoc
PK id
FK medium
FK cdtoc
edits_pending
last_updated
cdtoc
PK id
discid
freedb_id
track_count
leadout_offset
track_offset
degraded
created
release
PK id
gid
FK release_group
FK artist_credit
name
barcode
FK status
FK packaging
FK language
FK script
comment
edits_pending
quality
last_updated
release_status
PK id
name
release_packaging
PK id
name
release_unknown_country
PK,FK release
date_year
date_month
date_day
work_alias
PK id
FK work
name
sort_name
locale
edits_pending
last_updated
FK type
begin_date_year
begin_date_month
begin_date_day
end_date_year
end_date_month
end_date_day
ended
primary_for_locale
work_alias_type
PK id
name
work_type
PK id
name
iswc
PK id
FK work
source
iswc
edits_pending
created
work_attribute_type_allowed_value
PK id
FK work_attribute_type
value
work_attribute_type
PK id
name
comment
free_text
work
PK id
gid
FK type
name
comment
edits_pending
last_updated
FK language
work_attribute
PK id
FK work
FK work_attribute_type
FK work_attribute_type_allowed_value
work_attribute_text
script
id
iso_code
iso_number
name
frequency
script_language
id
FK script
FK language
frequency
language
id
iso_code_3
iso_code_2t
iso_code_2b
iso_code_1
name
frequency



27 tables reduce to 4

27 tables reduce to 4
so far

Further Reductions

- All 86 link tables become properties on artists, releases and other entities
- All 44 tag, rating and annotation tables become list properties
- All 5 cover art tables become lists of file references

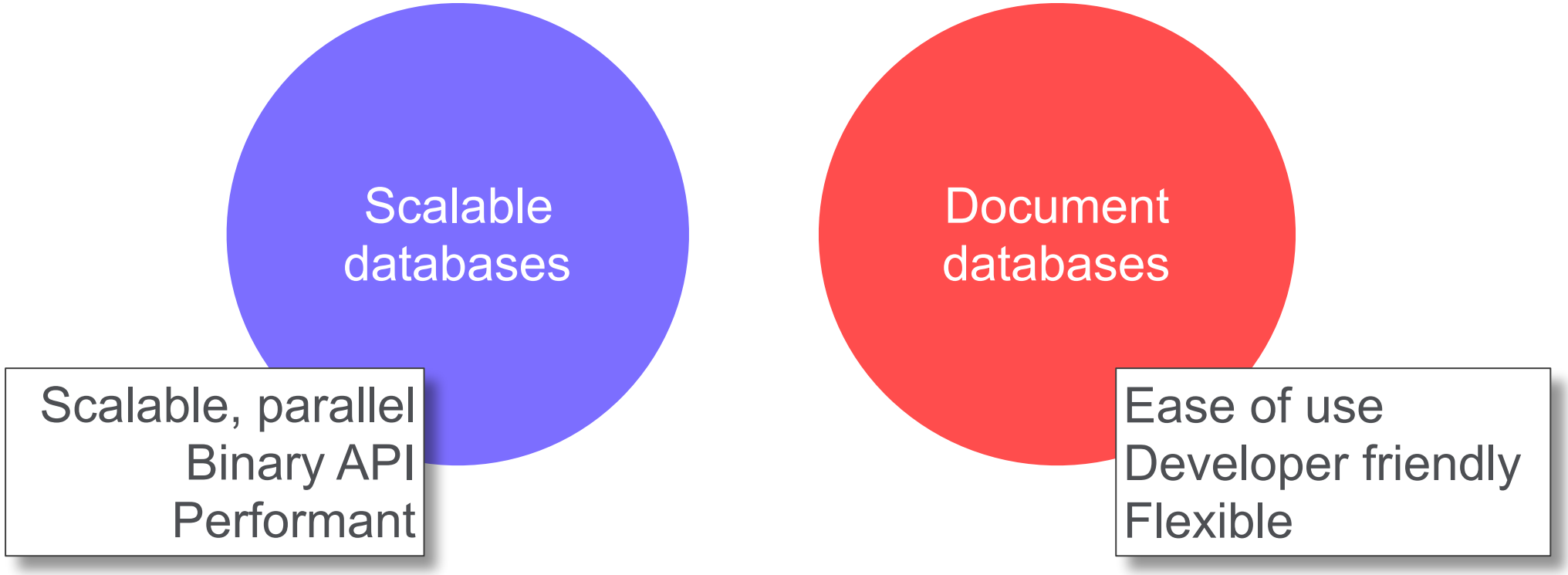
- Current score: 162 tables become 4

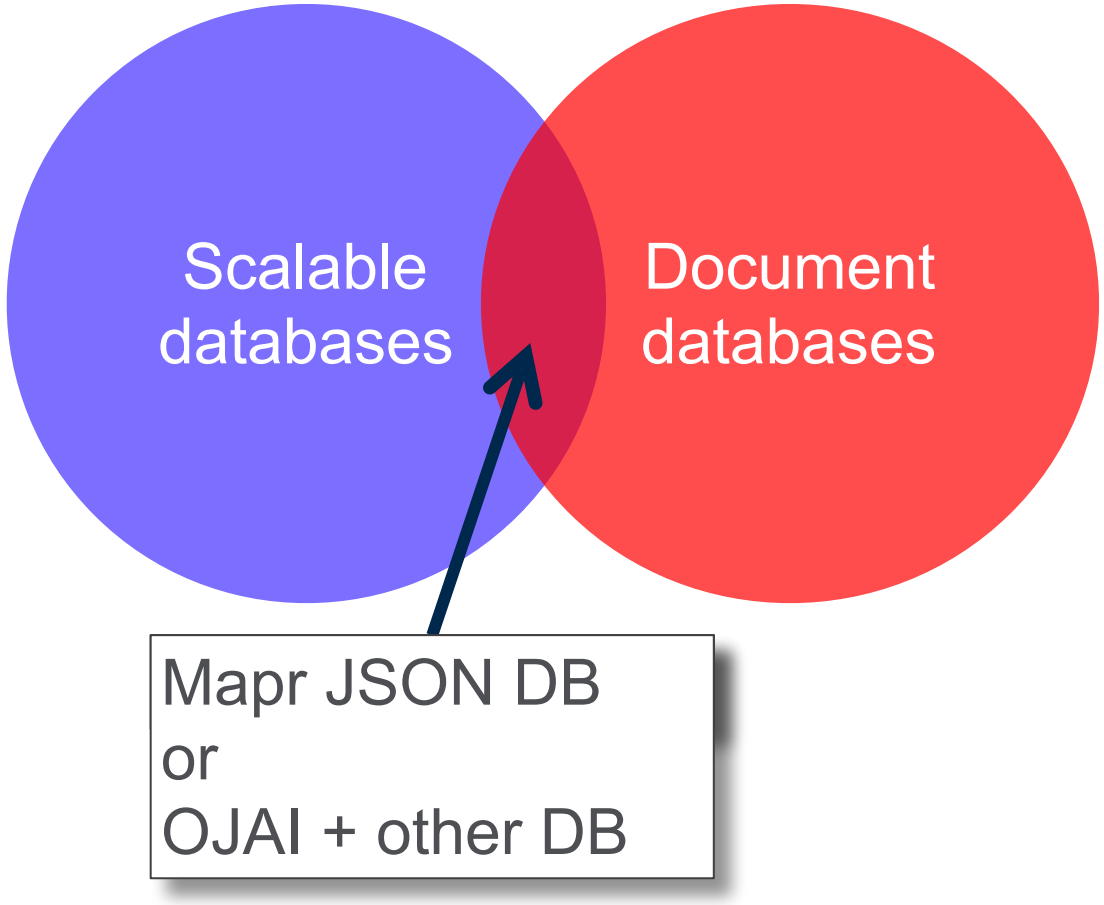
- You get the idea

Artist in HBase/MapR-DB

```
get '/apps/db/music/artist_hbase', 'nirvana'
COLUMN                                CELL
 default:begin_date                   timestamp=1460500945476, value=1988-01-01
 default:end_data                     timestamp=1460500945509, value=1994-04-05
 default:ended                        timestamp=1460500945538, value=true
 default:name                         timestamp=1460500945438, value=Nirvana
 list:albums                          timestamp=1460500945578, value=[
              {“title”:"In Utero", "released" : “1993-09-21"},
              {"title":"Nevermind", "released" : "1991-09-24"},
              {"title":"Bleach", "released" : "1989-06-15"}]
```

NoSQL Data Model



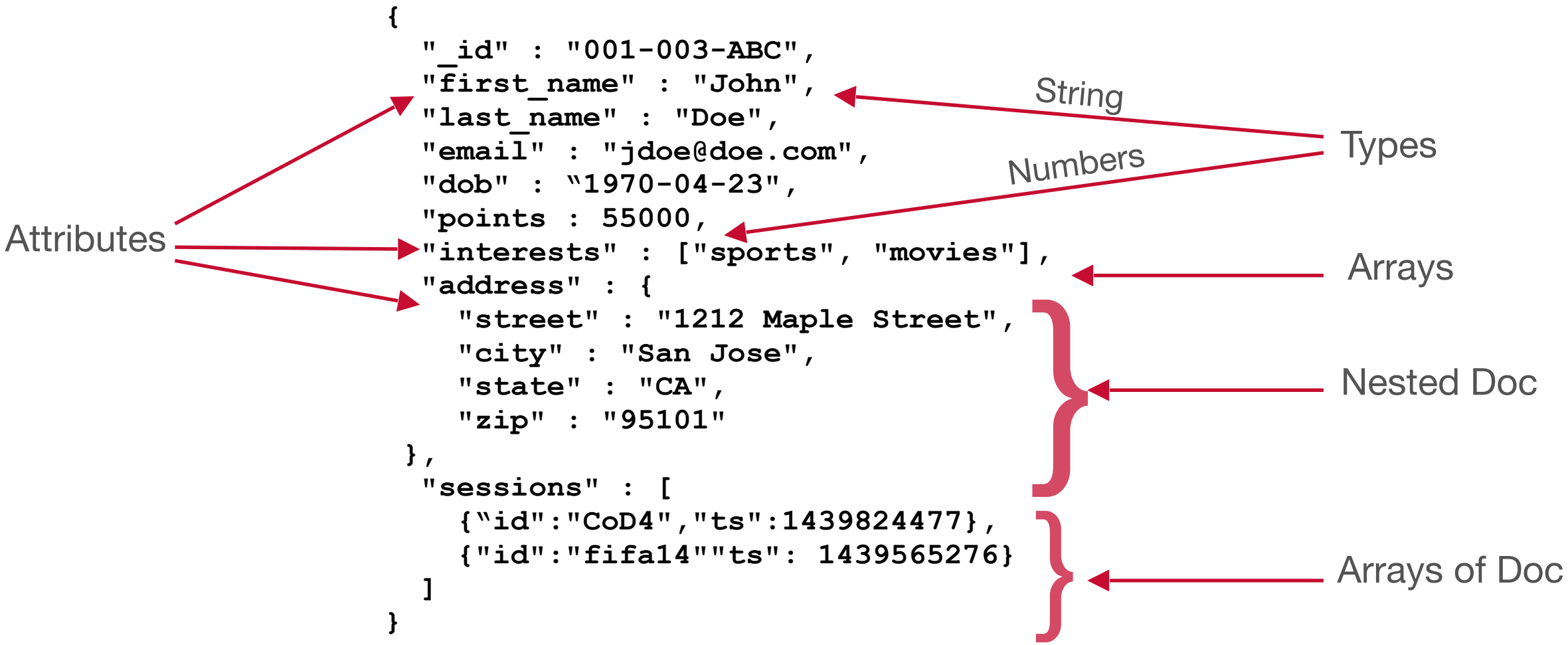


JSON Document : Flexible Schema

- Support Data-Types
- Complex Data Structure
- Developer Friendly
  - http://json.org
- Flexible
  - Easy to evolve
  - Start right, stay right

```
{
  "_id" : "001-003-ABC",
  "first_name" : "John",
  "last_name" : "Doe",
  "email" : "jdoe@doe.com",
  "dob" : "1970-04-23",
  "points : 55000,
  "interests" : ["sports", "movies"],
  "address" : {
    "street" : "1212 Maple Street",
    "city" : "San Jose",
    "state" : "CA",
    "zip" : "95101"
 },
  "sessions" : [
    {"id":"CoD4","ts":1439824477},
    {"id":"fifa14""ts": 1439565276}
  ]
}
```

JSON Document : Flexible Schema



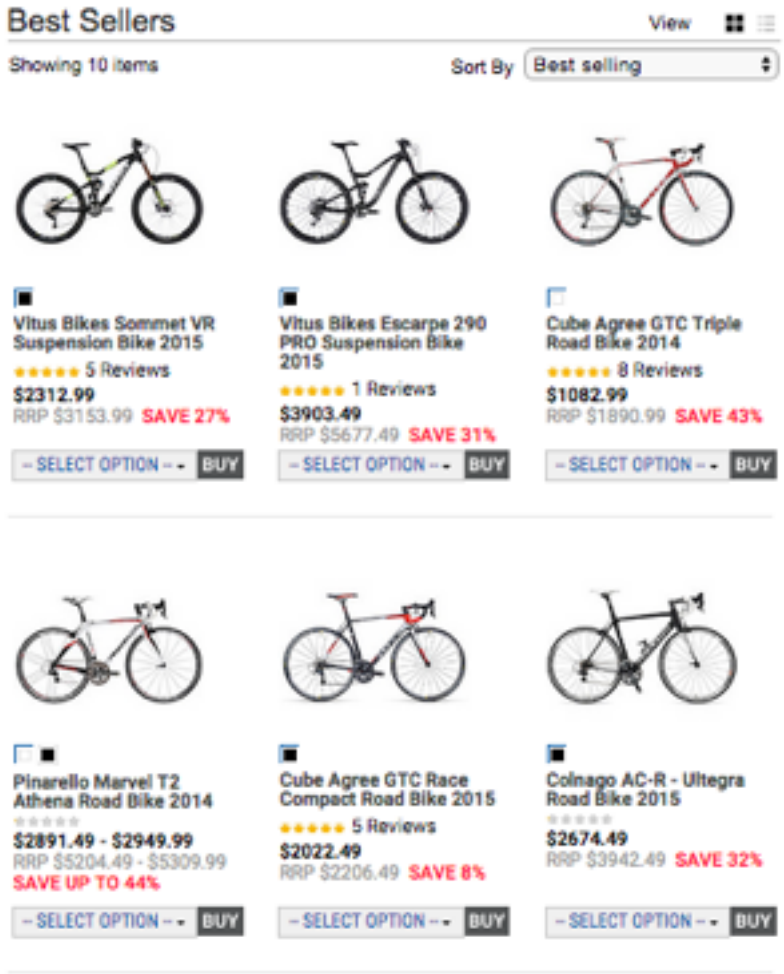
No SQL : use cases

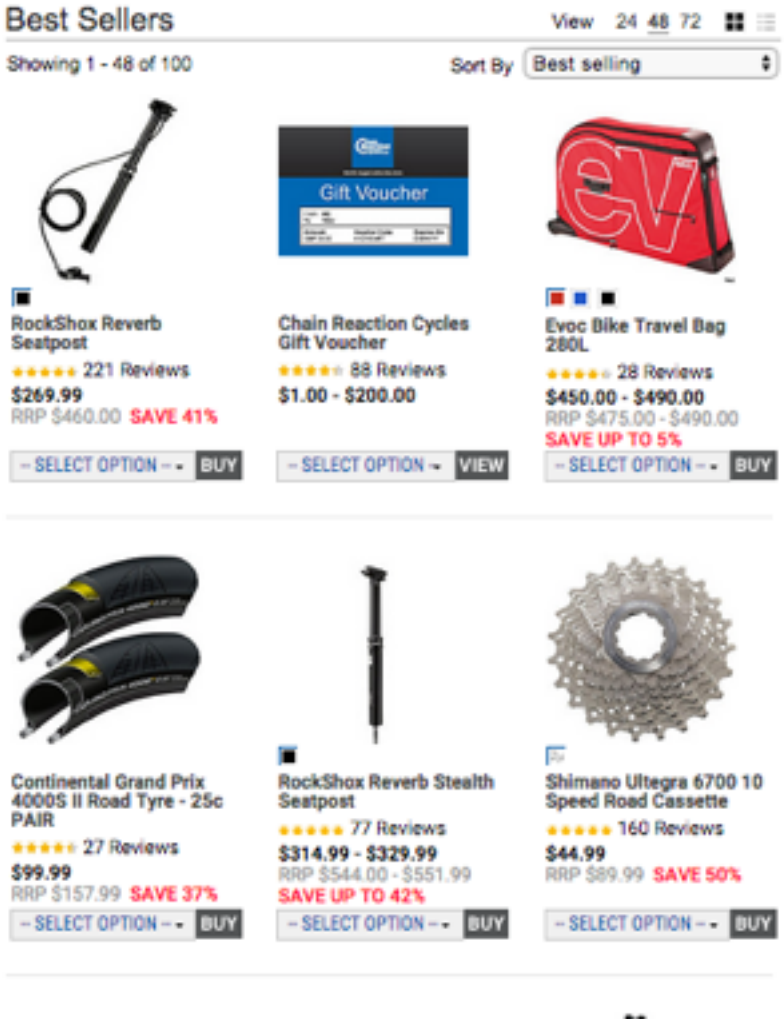
Key Value	Document	Wide Column	Graph
Session Management User Profile/Preferences Shopping Cart	Event Logging Content Management Web Analytics Product Catalog Single View e-Commerce	Event Logging Content Management Counters	Social Network Routing/Dispatch Recommendation on Social Graph
	MapR-DB-JSON	**MapR-DB-HBase**	

No SQL : use cases

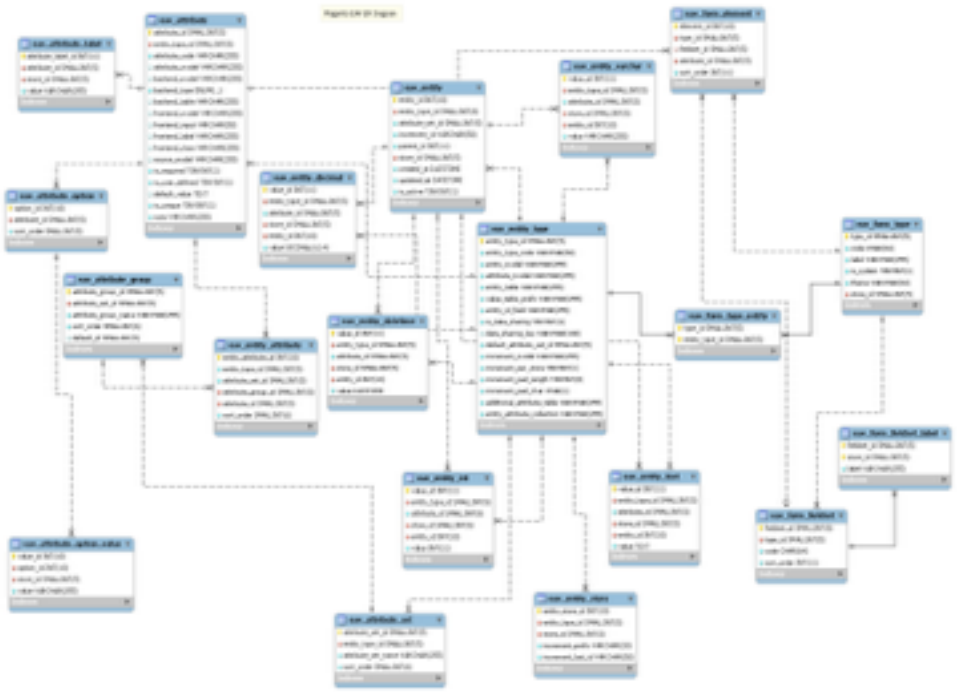
Key Value	Document	Wide Column	Graph
Session Management User Profile/Preferences Shopping Cart	Event Logging Content Management Web Analytics **Product Catalog** Single View e-Commerce	Event Logging Content Management Counters	Social Network Routing/Dispatch Recommendation on Social Graph
	MapR-DB-JSON	**MapR-DB-HBase**	

Flexible Schema in Action





Product Catalog - RDBMS



"Entity Value Attribute" Pattern

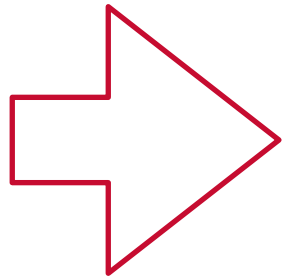
```
SELECT * FROM (
    SELECT
        ce.sku,
        ea.attribute_id,
        ea.attribute_code,
        CASE ea.backend_type
           WHEN 'varchar' THEN ce_varchar.value
           WHEN 'int' THEN ce_int.value
           WHEN 'text' THEN ce_text.value
           WHEN 'decimal' THEN ce_decimal.value
           WHEN 'datetime' THEN ce_datetime.value
           ELSE ea.backend_type
        END AS value,
        ea.is_required AS required
    FROM catalog_product_entity AS ce
    LEFT JOIN eav_attribute AS ea
        ON ce.entity_type_id = ea.entity_type_id
    LEFT JOIN catalog_product_entity_varchar AS ce_varchar
        ON ce.entity_id = ce_varchar.entity_id
        AND ea.attribute_id = ce_varchar.attribute_id
        AND ea.backend_type = 'varchar'
   LEFT JOIN catalog_product_entity_text AS ce_text
        ON ce.entity_id = ce_text.entity_id
        AND ea.attribute_id = ce_text.attribute_id
        AND ea.backend_type = 'text'
    LEFT JOIN catalog_product_entity_decimal AS ce_decimal
        ON ce.entity_id = ce_decimal.entity_id
        AND ea.attribute_id = ce_decimal.attribute_id
        AND ea.backend_type = 'decimal'
    LEFT JOIN catalog_product_entity_datetime AS ce_datetime
        ON ce.entity_id = ce_datetime.entity_id
        AND ea.attribute_id = ce_datetime.attribute_id
        AND ea.backend_type = 'datetime'
    WHERE ce.sku = 'rp-prod132546'
  ) AS tab
  WHERE tab.value != '';
```

To get a single product

Product Catalog - NoSQL/Document

```
{
  "_id" : "rp-prod132546",
  "name" : "Marvel T2 Athena”,
  "brand" : "Pinarello",
  "category" : "bike",
  "type" : "Road Bike”,
  "price" : 2949.99,
```

```
  "size" : "55cm",
  "wheel_size" : "700c",
  "frameset" : {
       "frame" : "Carbon Toryaca",
       "fork" : "Onda 2V C"
  },
  "groupset" : {
       "chainset" : "Camp. Athena 50/34",
       "brake" : "Camp."
  },
  "wheelset" : {
       "wheels" : "Camp. Zonda",
       "tyres" : "Vittoria Pro"
  }
}
```



```
products
  .findById(“rp-prod132546”)
```

Store the product “as a business object”

To get a single product

Easy variation with Documents

```
{
  "_id" : "rp-prod132546",
  "name" : "Marvel T2 Athena”,
  "brand" : "Pinarello",
  "category" : "bike",
  "type" : "Road Bike”,
  "price" : 2949.99,

  "size" : "55cm",
  "wheel_size" : "700c",
  "frameset" : {
       "frame" : "Carbon Toryaca",
       "fork" : "Onda 2V C"
  },
  "groupset" : {
       "chainset" : "Camp. Athena 50/34",
       "brake" : "Camp."
  },
  "wheelset" : {
       "wheels" : "Camp. Zonda",
       "tyres" : "Vittoria Pro"
  }
}
```

```
{
  "_id" : "rp-prod106702",
  "name" : " Ultegra SPD-SL 6800”,
  "brand" : "Shimano",
  "category" : "pedals",
  "type" : "Components,
  "price" : 112.99,

  "features" : [
       "Low profile design increases ...",
       "Supplied with floating SH11 cleats",
       "Weight: 260g (pair)"
  ]
}
```

```
{
  "_id" : "rp-prod113104",
  "name" : "Bianchi Pride Jersey SS15”,
  "brand" : "Nalini",
  "category" : "Jersey",
  "type" : "Clothing,
  "price" : 76.99,

  "features" : [
       "100% Polyester",
       "3/4 hidden zip",
       "3 rear pocket"
  ],
  "color" : "black"
}
```

Back-end data
matches
front-end expectations

Add features easily

- Requirement: *“users can vote and comment product”*

```
{
  "_id" : "rp-prod113104",
  "name" : "Bianchi Pride Jersey SS15”,
  "brand" : "Nalini",
  "category" : "Jersey",
  "type" : "Clothing,
  "price" : 76.99,
  "features" : [
      "100% Polyester",
      "3/4 hidden zip",
      "3 rear pocket"
  ],
  "color" : “black”,
  "comments" : [{…}, {…}],
  "ratings" : [{…, “value" : 5}, {…, value : 3}],
  “rating” : 4
}
```

Done: just store the data!

Means less time to market

JSON documents make data modeling easy

An artist look

```
find '/apps/db/music/artist_json', 'nirvana'
{
 "_id" : "nirvana",
 "name" : "Nirvana",
 "begin_date" : "1988-01-01",
 "end_data" : "1994-04-05"
 "ended" : true,
 "albums" : [
    {"title":"In Utero", "released" : "1993-09-21"},
    {"title":"Nevermind", "released" : "1991-09-24"},
    {"title":"Bleach", "released" : "1989-06-15"}
   ]
}
```

Artist in HBase/MapR-DB Binary

```
get '/apps/db/music/artist_hbase', 'nirvana'
COLUMN                                CELL
 default:begin_date                   timestamp=1460500945476, value=1988-01-01
 default:end_data                     timestamp=1460500945509, value=1994-04-05
 default:ended                        timestamp=1460500945538, value=true
 default:name                         timestamp=1460500945438, value=Nirvana
 list:albums                          timestamp=1460500945578, value=[
              {“title”:"In Utero", "released" : “1993-09-21"},
              {"title":"Nevermind", "released" : "1991-09-24"},
              {"title":"Bleach", "released" : "1989-06-15"}]
```

How does it work

- MapR JSON DB inherits almost all aspects of MapR DB
  - Columns families (but now defined on the fly)
  - Column level security

- MapR-DB JSON
  - does NOT store JSON document “string”
  - stores “real” data types (not Javascript Type)

- OJAI (open source) can add similar capabilities to other db’s
  - http://ojai.io/ (/ OH-hy /)

Developers love JSON because….

- API is really easy to use :
  - you deal with your business objects
- All operations to manipulate the fields, structure
  - Add/Remove Fields
  - Update Values, including sub documents, arrays
  - Increments

```
Document a = new Document();
a.add(new Array(“a”,”b”,”c”));
table.insert(key, a);

Mutation t = new Mutation();
t.set(“address.zip”,”95101”);
table.update(key, t);

Document a = new Document(JSON_String); // with _id field
table.insert(a);
```

Why is the developer so happy?

- Can build a quick and dirty MVP that evolves
- Can update application just a little bit
- Can tune performance later using column families

Why is the developer so happy?

- Expressive: lets you say what you need to say
  - Developer hate circumlocution … say what you mean, don't repeat yourself
- Efficient (remember how much easier to get one product?)
  - Simple designs run better because you can get them right
- Human readable: you can introspect

SQL on NoSQL

HBase Columnar

```
select
  convert_from(t.row_key, 'UTF8') id,
  convert_from(t.`default`.`name`, 'UTF8') name,
 convert_from(convert_from(t.list.albums, 'UTF8'),
'JSON') albums
from hbase.`/apps/db/music/artist_hbase` t
```

Document Database (MapR-DB JSON)

```
select
 t._id,
 t.name,
 t.albums
from maprdb.`/apps/db/artist_json` t
```

Summary

Some Tips

- Carefully select rowkey/id & keys
- Design for “your application”
  - Design for “questions” not “answers” *(Highly Scalable Blog)*
- Rows are updated atomically
- Use Pre Aggregation

Q&A

Engage with us!

@tgrall   maprtech

mapr-technologies   MapR

tug@mapr.com   maprtech