

# GETTING STARTED WITH NoSQL

Aaron Benton / [@bentonam](#) / [github.com/bentonam](#)



- The problem with SQL
- History of NoSQL
- Database theories
- Modeling
- Patterns

# THE PROBLEM WITH SQL

- Designed to run on large servers
- Built for Vertical Scaling
- Separated Models i.e. Tables

# IN DEVELOPMENT

- We assemble objects as a whole
  - Cart
  - Order
  - Profile
  - Product
- Saving objects requires
  - Deconstructing
  - Multiple rows
  - Multiple tables

# IMPEDANCE MISMATCH

*"The object-relational impedance mismatch is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style, particularly when objects or class definitions are mapped in a straightforward way to database tables or relational schema." - Wikipedia*

**YOU CAN IGNORE  
SQL**



**AND TRUST THE ORM  
MAGIC**

[memegenerator.net](http://memegenerator.net)





**Google BigTable**

**Amazon DynamoDB**

# NoSQL

# No SQL

# Not only SQL





# #nosql

# WHAT IS NOSQL?

- non-relational
- cluster friendly
- generally open-source
- 21st century
- schema-less



# TYPES OF NOSQL DATABASES

- Key-Value: Redis, Riak, Memcached
- Column-family: Cassandra, HBase, BigTable
- Document: CouchDB, Couchbase, MongoDB
- Graph: Neo4J, Giraph, OrientD

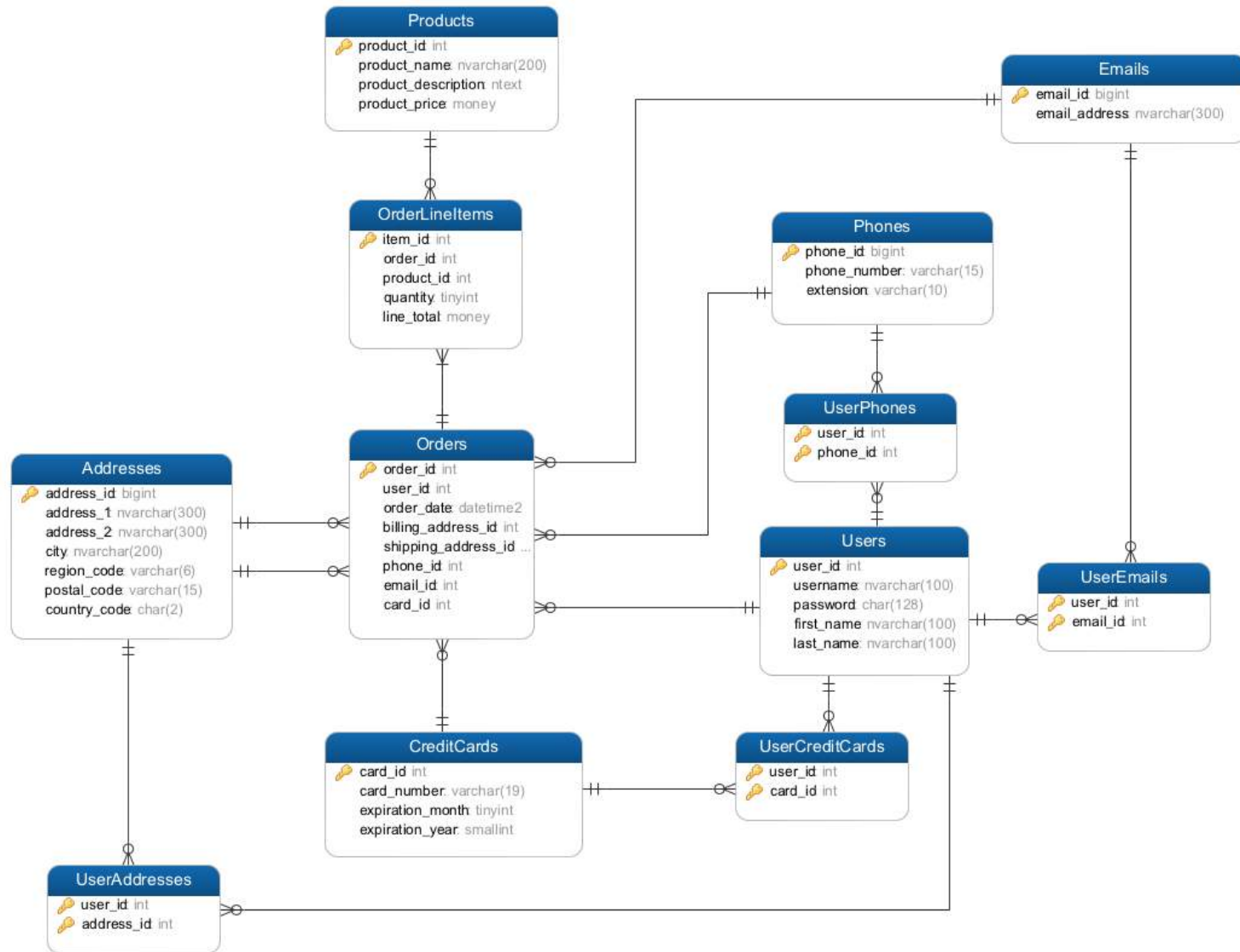
# AGGREGATES

```
// Order.cfc
component accessors="true"{
    property="order_id" type="numeric";
    property="order_date" type="date";
    property="products" type="array";
    property="user_id" type="numeric";
    property="billing_address_1" type="string";
    property="billing_address_2" type="string";
    property="billing_city" type="string";
    property="billing_region_code" type="string";
    property="billing_postal_code" type="string";
    property="billing_country_code" type="string";
    property="shipping_address_1" type="string";
    property="shipping_address_2" type="string";
    property="shipping_city" type="string";
    property="shipping_region_code" type="string";
    property="shipping_postal_code" type="string";
    property="shipping_country_code" type="string";
    property="card_number" type="string";
    property="expiration_month" type="numeric";
    property="expiration_year" type="numeric";
}
```

```
// Order.cfc
component accessors="true"{
    property="order_id" type="numeric";
    property="order_date" type="date";
    property="products" type="array";
    property="user" type="User";
    property="billing" type="Address";
    property="shipping" type="Address";
    property="cc_info" type="CreditCard";
}
```

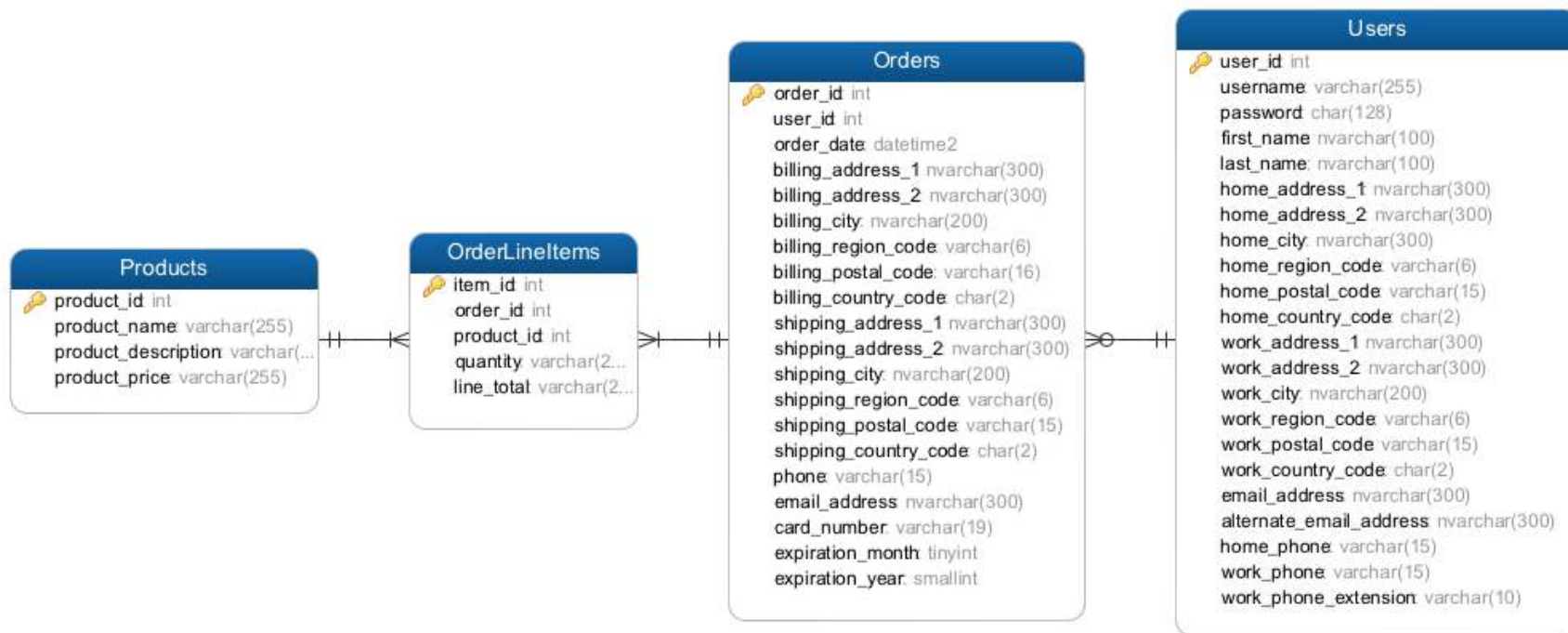
# NORMALIZATION

- Minimize data redundancy
- Structured models
- Logical queries
- Fast inserts / updates
- Less storage requirements



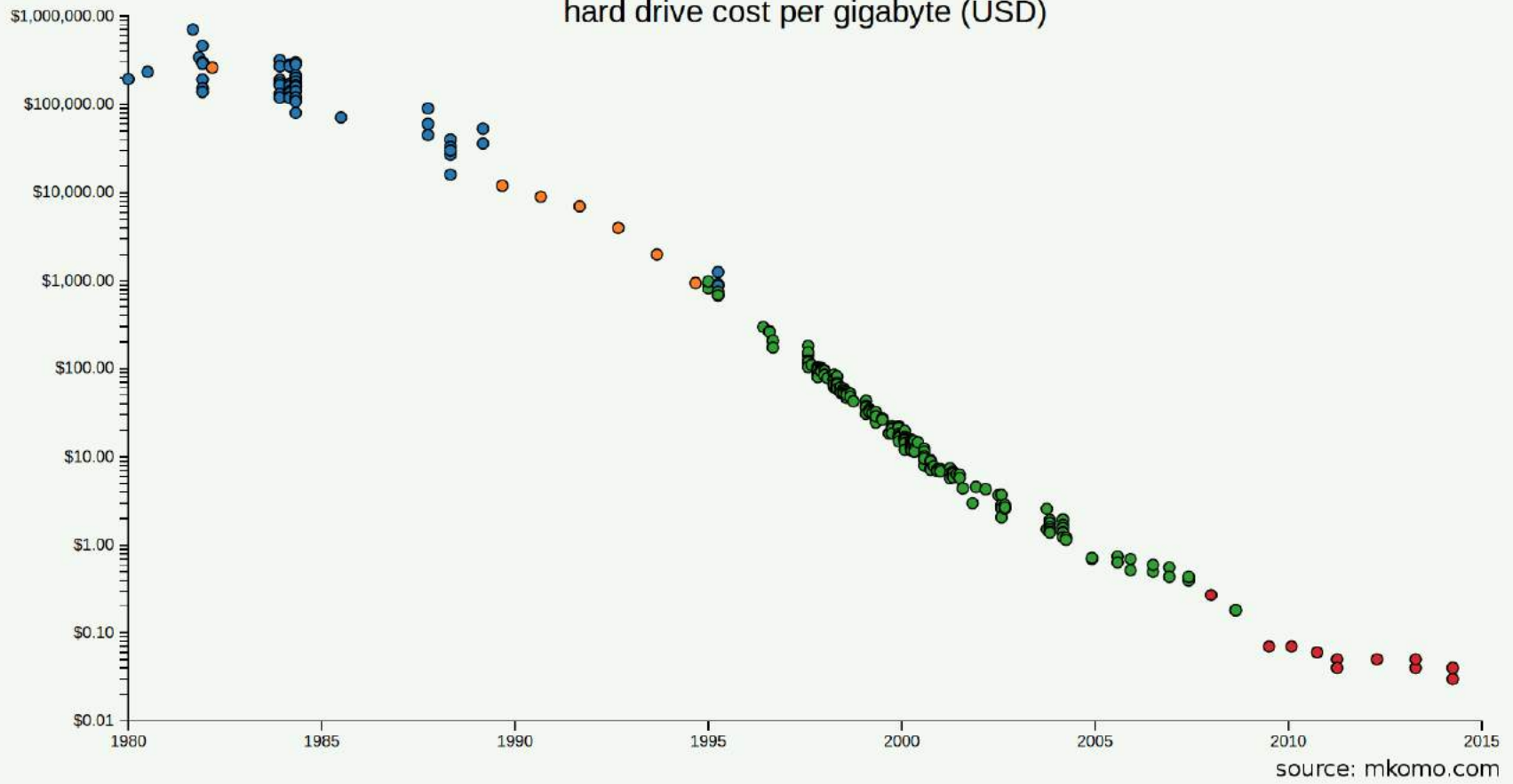
# DENORMALIZATION

- Minimize JOINS
- Fast reads
- Repeated data
- More storage





hard drive cost per gigabyte (USD)



# TRANSACTION PROCESSING

# ACID

- Atomicity
- Consistency
- Isolation
- Durability

# BASE

- Basically Available
- Soft State
- Eventual Consistency

# ATOMIC TRANSACTIONS

**Aggregate Orientated == NoSQL - Graph**

# CONFLICT RESOLUTION

# **CONSISTENCY**

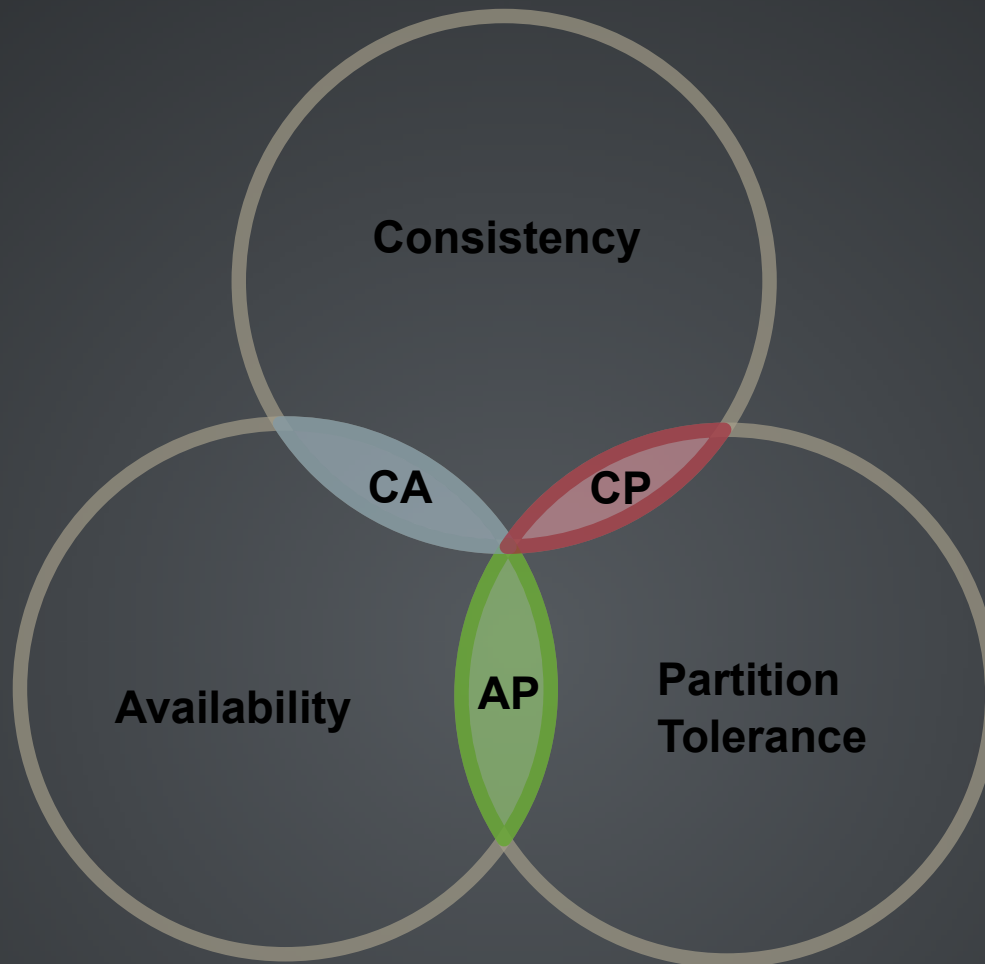
## **SHARDING VS. REPLICATION**



# CAP THEOREM

- Consistency
- Availability
- Partition Tolerance

You can only provide 2 of the 3



# RDBMS SCHEMAS

- Known Models
- Fixed Fields
- Data types
- Database managed
- Change can be difficult



# NOSQL SCHEMAS

- Any type of data
- Flexible
- Application managed
- Change is easy



**IMPLICIT SCHEMA**

**~~SCHEMA LESS~~**

# DATA / ENTITY RELATIONSHIP MODELING

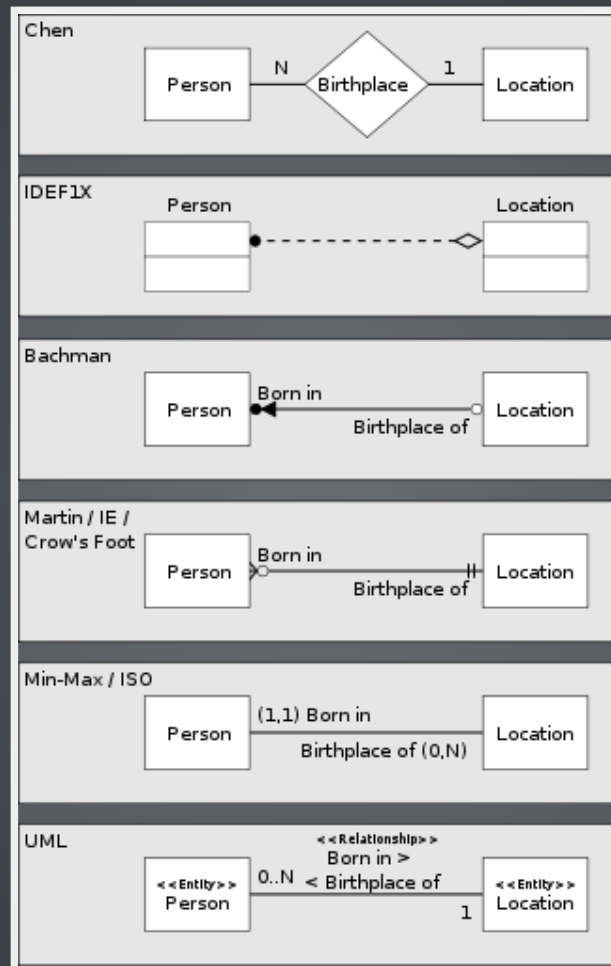
- Conceptual Data Model
- Logical Data Model
- Physical Data Model



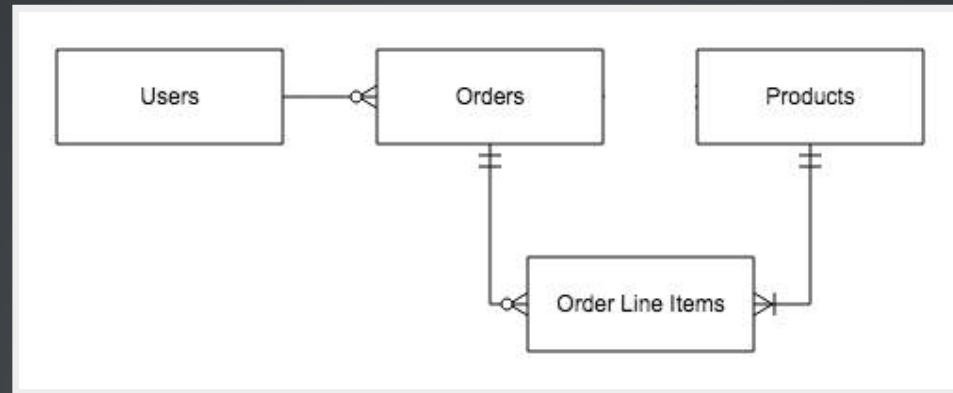
# CONCEPTUAL DATA MODEL

- Entity Names
- Entity Relationships

# MODELING NOTATIONS



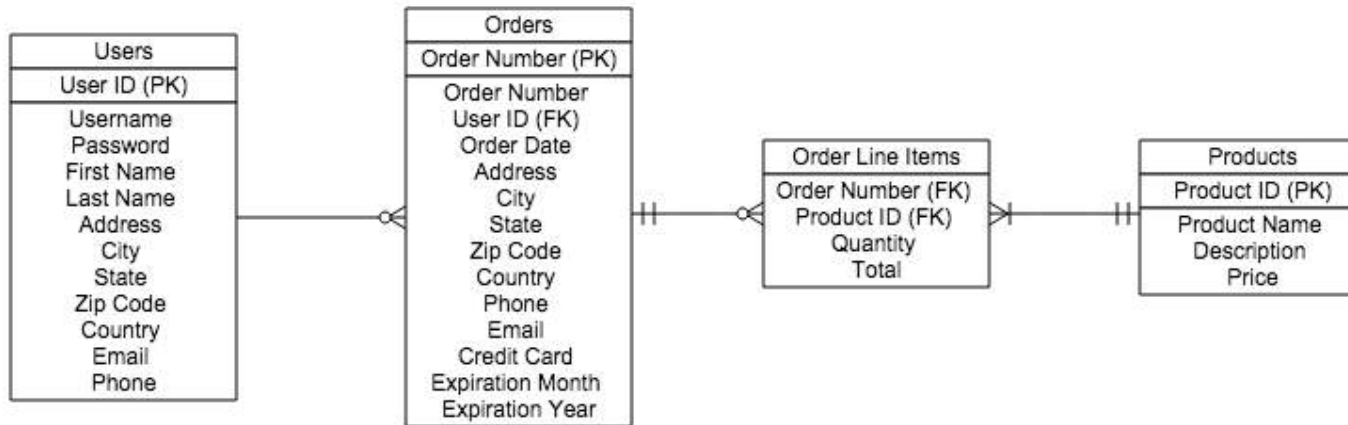
# CONCEPTUAL DATA MODEL



# LOGICAL DATA MODEL

- Entity Names
- Entity Relationships
- Attributes
- Primary / Foreign Keys

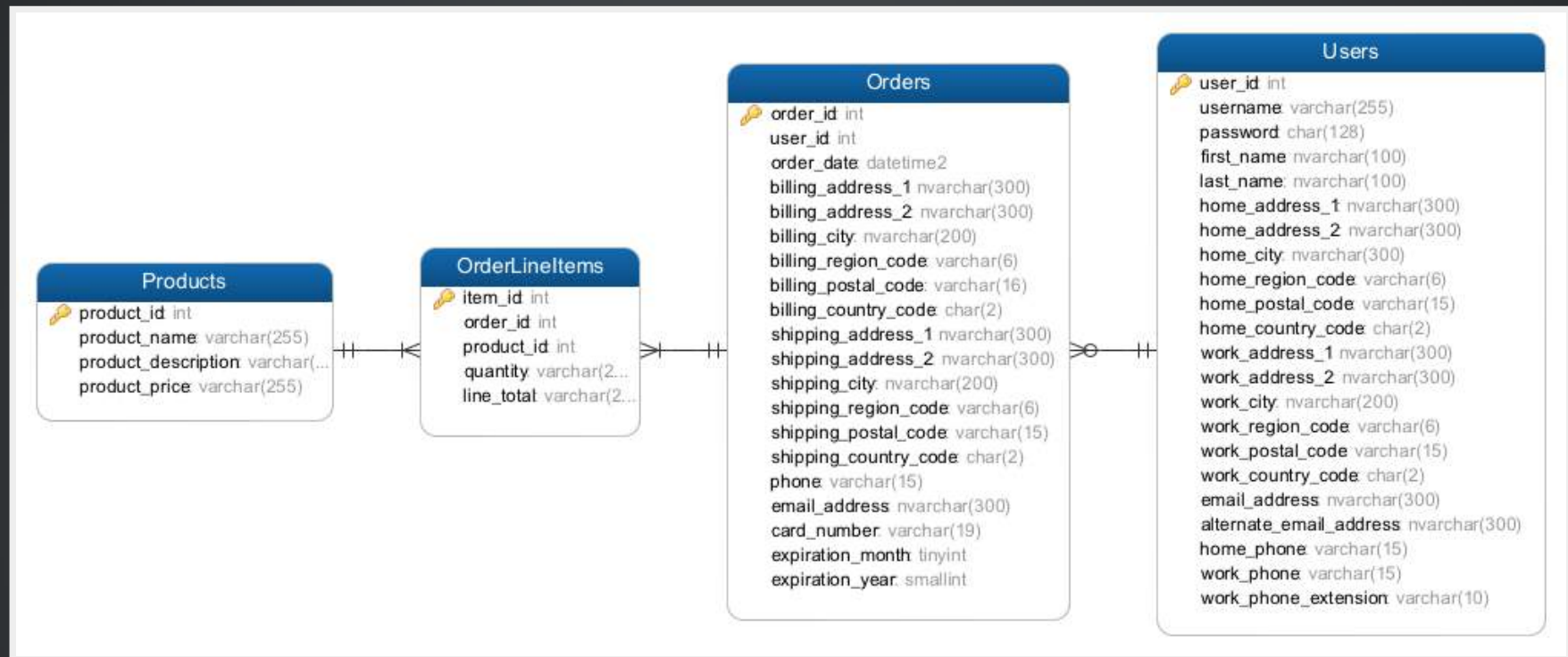
# LOGICAL DATA MODEL



# PHYSICAL DATA MODEL

- Entity -> Table Names
- Attributes -> Field Names
- Keys -> Primary / Foreign Keys
- Data Types

# PHYSICAL DATA MODEL



# PHYSICAL DATA MODEL IN NOSQL



```
{  
  "user_id": 123,  
  "username": "jdoe",  
  "first_name": "John",  
  "last_name": "Doe",  
  "email": "john.doe@mail.com",  
  "password": "88142f883cba2b527fdbbc60a943b899"  
}
```



```
type: object
properties:
  _id:
    type: string
    description: The ID of the document
  user_id:
    type: integer
    description: > An auto-incremented number from `users_counter`
    that is the ID of the user
  first_name:
    type: string
    description: The users first name
  last_name:
    type: string
    description: The users last name
  username:
    type: string
    description: A unique username chosen by the user
  friends:
    type: array
    description: An array of user_id who the user is friends with
  created_on:
    type: integer
    description: An epoch time in seconds when the user was created
```

# FAKEIT

Inspired by [Swagger](#), fakeit is a CLI data generator based on YAML models that outputs JSON, YAML, CSON, or CSV formats

```
npm install fakeit -g
```

- <https://www.npmjs.com/package/fakeit>
- <https://github.com/bentonam/fakeit>
- <https://github.com/bentonam/fakeit-examples>

# FAKEIT

- Generate fixed or random number of documents per model
- Event Transforms: Pre / Post Run, Pre / Post Build
- Data generation via FakerJS, ChanceJS, Custom or Static
- Data Typing
- Model Dependencies
- JSON, YAML, CSON, CSV output formats
- Output to File, Zip, Couchbase or Sync Gateway

```
name: Users
type: object
key: _id
data:
  min: 200
  max: 500
  pre_run: >
    globals.user_counter = 0;
properties:
  id:
    type: string
    data:
      post_build: "return 'user_' + this.user_id;"
  type:
    type: string
    data:
      value: "user"
  user_id:
    type: integer
    data:
      build: "return ++globals.user_counter;"
  name:
    type: string
    data:
      fake: "{{name.firstName}} {{name.lastName}}"
  phone:
    type: string
    data:
      build: "return chance.phone();"
  created_on:
    type: string
    data:
      fake: "{{date.past}}"
      post_build: "return new Date(this.created_on).toISOString();"
```

# FAKEIT DEMO

# KEY DESIGN

- Prefixing
- Predictable
- Counter ID
- Unpredictable
- Combinations

# PREFIXING

- user\_123
- u::john.doe@mail.com
- user-123
- user\_123\_orders
- order\_123
- o::john.doe@mail.com
- product-123
- user\_123\_orders

# PREDICTABLE

Key: user\_john.doe@mail.com

Key: user\_jdoe





# COUNTER ID

Key: user\_123

```
JSON
{
  user_id : 123
  username : "jdoe"
  first_name : "John"
  last_name : "Doe"
  email : "john.doe@mail.com"
  password : "88142f883cba2b527fdbbc60a943b899"
```

Key: user\_counter

```
JSON
{
  counter : 414
```

# UNPREDICTABLE

Key: 23ad6bac-7599-4874-af98-7af734027834



# COMBINATIONS

- user\_123\_preferences
- user\_jdoe\_order\_23ad6bac-7599-4874-af98-7af734027834
- user\_john.doe@mail.com\_comment\_5664

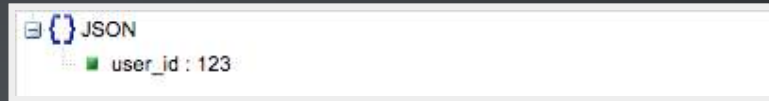
# DOCUMENT PATTERNS

# LOOKUP PATTERN

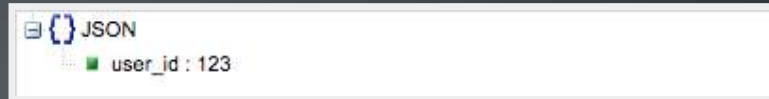
Key: user\_123



Key: user\_john.doe@mail.com



Key: user\_jdoe

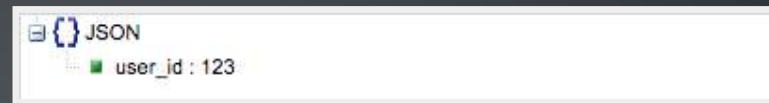


# LOOKUP PATTERN FOR AUTH

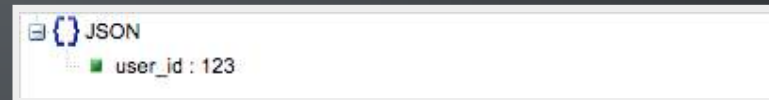
Key: user\_123



Key: user\_john.doe@mail.com\_  
88142f883cba2b527fdbbc60a943b8



Key: user\_jdoe\_  
88142f883cba2b527fdbbc60a943b8



# EMBEDDING

```
... username : "jdoe"
... first_name : "John"
... last_name : "Doe"
... email : "john.doe@mail.com"
... password : "88142f883cba2b527fdbbc60a943b899"
... [] addresses
...   [] 0
...     address_1 : "123 Missing St."
...     address_2 : ""
...     city : "Greensboro"
...     region_code : "NC"
...     postal_code : "27409"
...     type : "home"
...   [] 1
...     address_1 : "321 Smithfield Ave."
...     address_2 : ""
...     city : "Greensboro"
...     region_code : "NC"
...     postal_code : "27409"
...     type : "work"
```

# REFERRING / LINKING

Key: user\_123

```
JSON
{
  user_id : 123
  username : "jdoe"
  first_name : "John"
  last_name : "Doe"
  email : "john.doe@mail.com"
  password : "88142f883cba2b527fdbbc60a943b899"
  friends : [
    0 : 456
    1 : 743
    2 : 834
  ]
}
```

Key: user\_456

```
JSON
{
  user_id : 456
  username : "jsmith"
  first_name : "Jane"
  last_name : "Smith"
  email : "jane.smith@mail.com"
  password : "87a48ef776efbbe92651257bc1a52e84"
  friends : [
    0 : 123
    1 : 654
    2 : 837
  ]
}
```



# PARENT-REFERENCING

Key: product\_000a9863-6015-4dc8-9ee16ec0a00f9ea9

Key: product\_000a9863-6015-4dc8-9ee16ec0a00f9ea9\_review\_343234

```
JSON
{
  availability : "In-Stock"
  long_description : ""
  product_id : "000a9863-6015-4dc8-9ee16ec0a00f9ea9"
  price : 920.19
  sale_price : 0
  brand : "Lorem Ipsum"
  category : "Sports"
  created_on : 779566174000
  image : "http://placeholder.it/400"
  short_description : ""
  title : "Anguine Insanely Kalpis"
}
```

```
JSON
{
  reviewer_email : "james.johnson@mail.com"
  product_id : "000a9863-6015-4dc8-9ee16ec0a00f9ea9"
  review_title : "Idiomaticalness Angeronia Cremasterial"
  doc_type : "review"
  review_id : 343234
  review_date : 1407150480000
  review_body : ""
  rating : 2
  reviewer_name : "James Johnson"
}
```

**Is SQL Going Away?**

**NO**

# CONSIDERATIONS

- How do you work with your data?
- Do you work with the same aggregates all the time?
- What are you trying to achieve?
- Where are you starting at?
- Do you need finite data and highly complex relationships?
- Is the tabular structure working for you?
- Do you want to scale vertically or horizontally?
- Does your data need to be data centralized or decentralized?

# QUESTIONS?

**SLIDES AVAILABLE AT:**

**[bit.ly/gsw-nosql](http://bit.ly/gsw-nosql)**

# RESOURCES

- [Introduction to NoSQL by Martin Fowler](#)
- [Relationships are Hard NoSQL Data Modeling by Curt Gratz](#)
- [Workshop: NoSQL Data Modelling by Jan Steemann](#)
- [CAP Twelve Years Later: How the "Rules" Have Changed by Eric Brewer](#)
- [NoSQL Databases: An Overview by Pramod Sadalage](#)