Relationships are Hard

What we won't be covering

- Why you should never use a RDBMS again
- Map/Reduce, Views, or other NoSQL query options
- Why NoSQL databases are the cats pajamas
- How to prove the theory of relativity

What we will be covering

- Where NoSQL came from
- Some database theory
- When its good or not good to use NoSQL
- A few Key Patterns
- A few Data Modeling Patterns
- Samples and Ideas of when to use both

The Obligatory Who Am I?

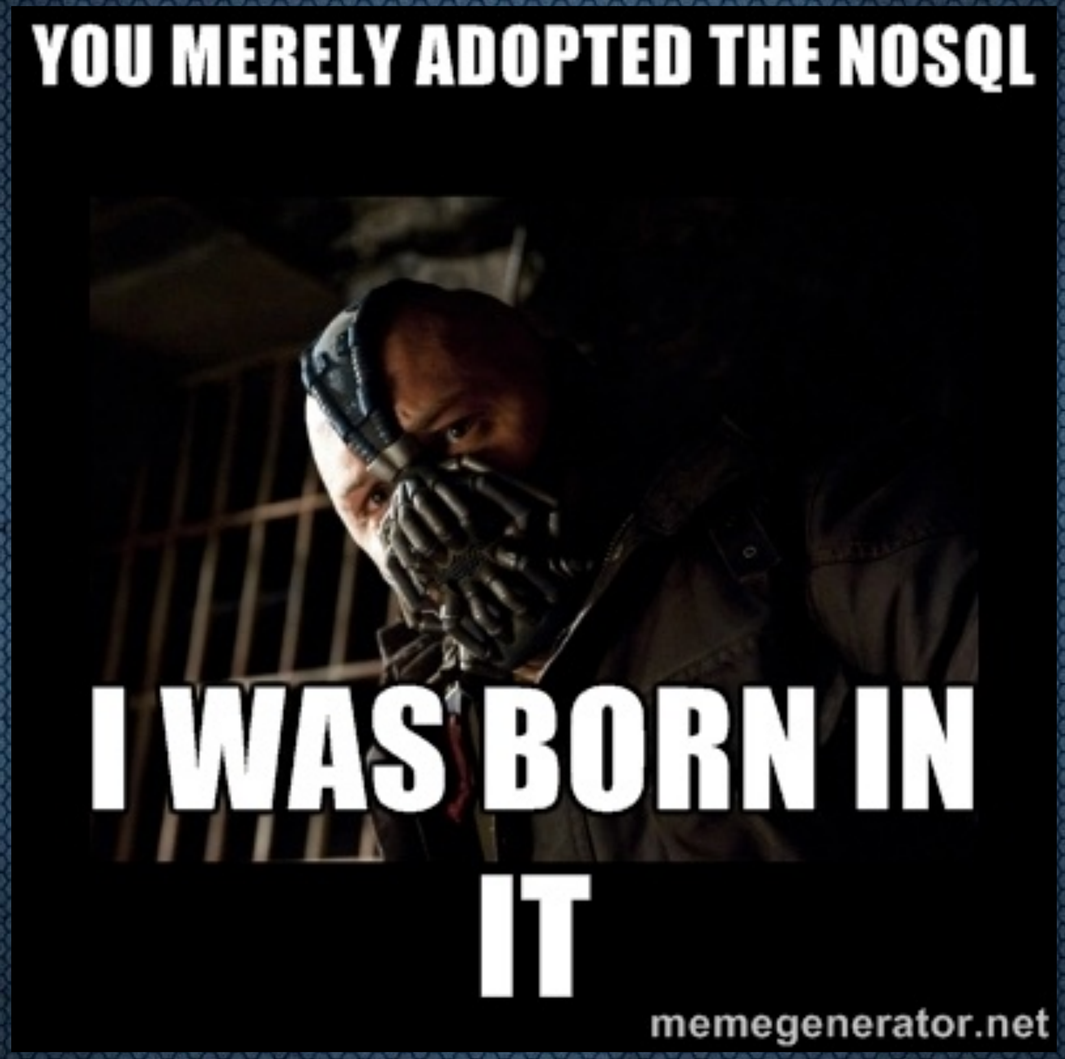
- Husband
- Dad
- Coach (CC and Track)
- Co-Owner of CKH Consulting
- Runner
- Polyglot

ckh Consulting

#noSQL


#IF #YOU #COULD #STOP #WRITING
#LIKE #THIS #IN #EVERY #SOCIAL
#NETWORK
THAT'D BE GREAT
MEMEFUL.COM

It’s Just a Buzz Word



What does it mean?

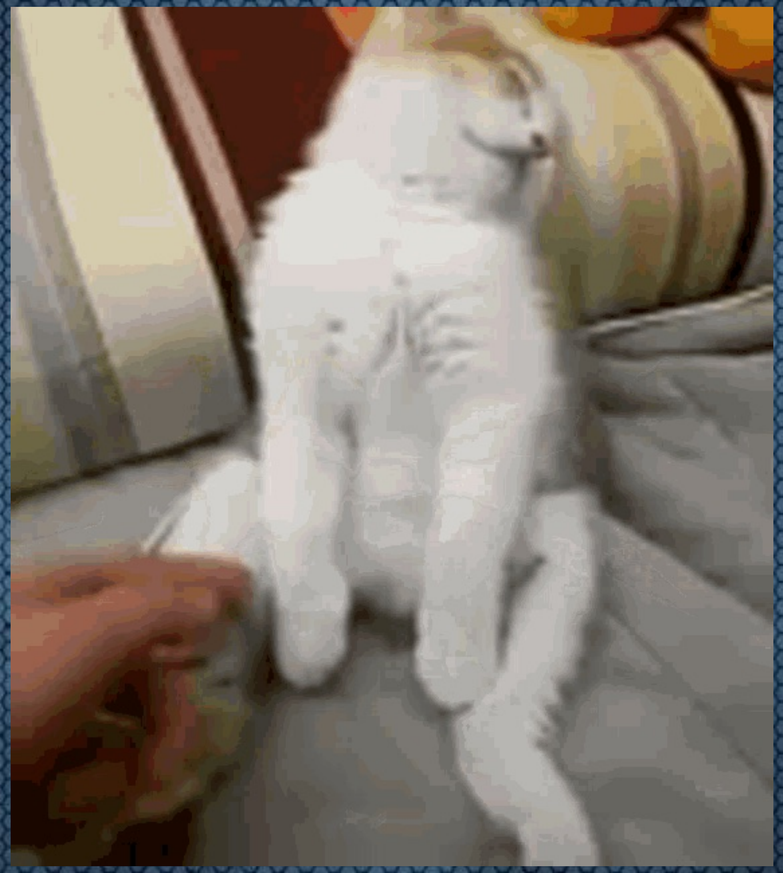
Brief History

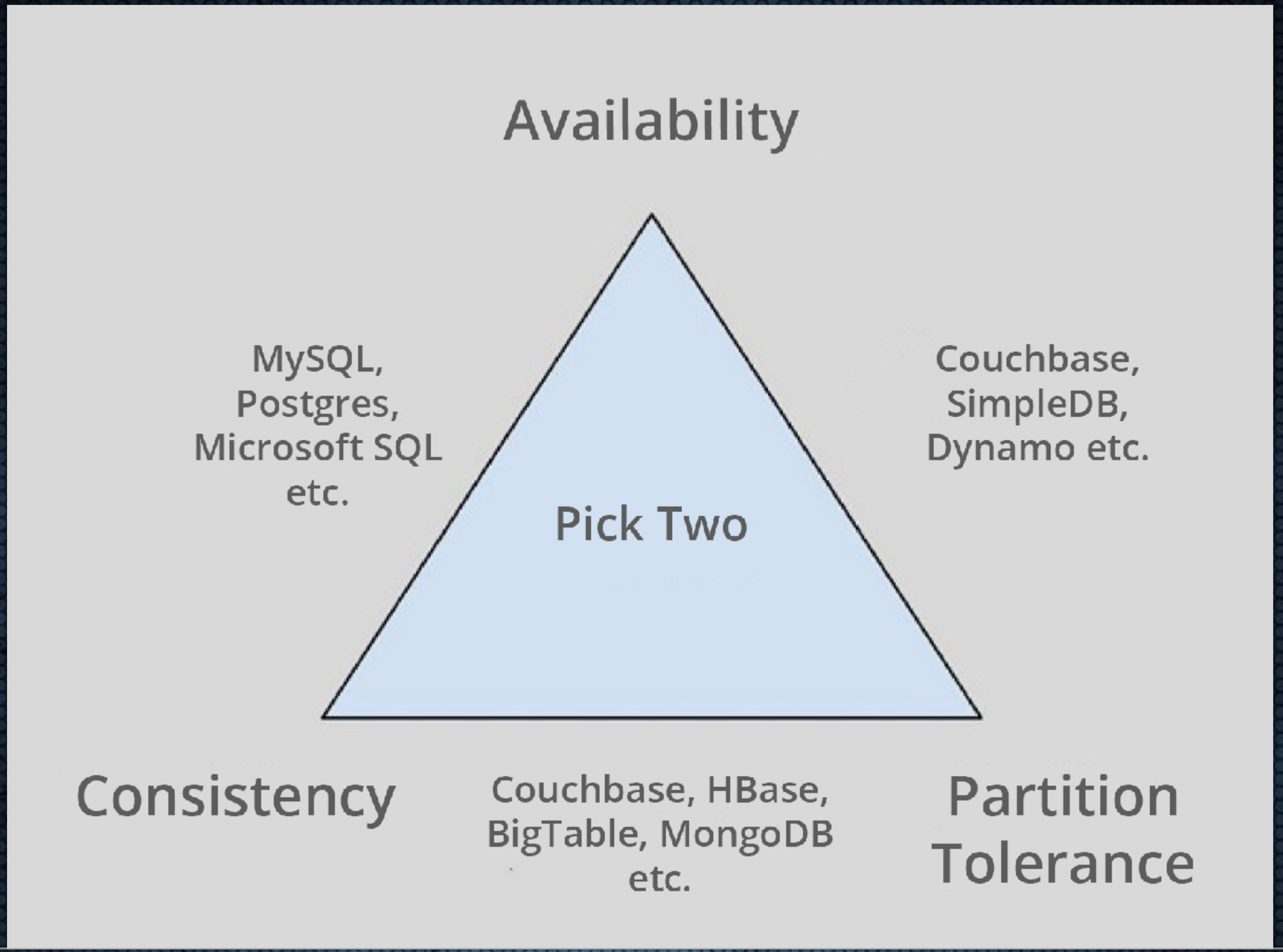
- Characteristics
  - non-relational
  - open-source (some)
  - cluster-friendly (some)
  - 21st Century
  - schema less

Brief History

- Types
  - key/value
    - MemcacheDB
  - document
    - Couchbase, Redis, MongoDB
  - column
    - Cassandra, HBase
  - graph
    - neo4j, InfiniteGraph

And now for some database theory



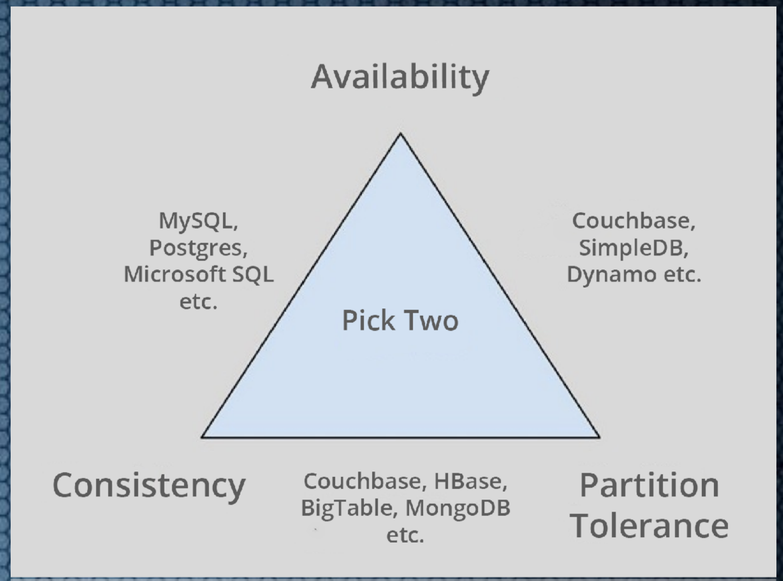


CAP THEOREM

PICK TWO

CAP Theorem (pick 2 please)

- Consistency
  - all nodes see the same data at the same time
- Availability
  - a guarantee that every client making a request for data gets a response, even if one or more nodes are down
- Partition Tolerance
  - the system continues to operate despite arbitrary message loss or failure of part of the system



ACID

- Atomicity
  - All or Nothing (transactions)
- Consistency
  - data is never violated (constraints. validation, etc)
- Isolation
  - operations must not see data from other non-finished operations(locks)
- Durability
  - state of data is persisted when operation finishes (server crash protection)

Acids

and

Bases

BASE

- Basically Available
  - Should be up, but not guaranteed
- Soft state
  - changes happening even without input
- Eventual consistency
  - At some point your changes will be consistent

Normalisation

- Less data duplication
- Less storage space
- Higher data consistency/integrity
- Easy to query
- Updates are EASY

Normalisation

statusID	status
1	active
2	inactive

reference

userID	username	groupID	statusID
1	wither	1	1
2	zombie	1	1
3	slime	2	2
4	enderman	1	2

reference

groupID	group
1	monster
2	block

Denormalisation

- Minimize the need for joins
- Reduce the number of indexes
- Reduce the number of relationships
- Updates are harder (slower)

Denormalisation

userID	username	group	status
1	wither	monster	active
2	zombie	monster	active
3	slime	block	inactive
4	enderman	monster	inactive

WAKE UP

Keying in NoSQL

- Predictable Keys
- Counter-ID
- Lookup Pattern
- Counter/Lookup combined

Predictable Keys

- No built in key mechanisms in most NoSQL
- Use a Unique value for key (email, usernames, sku, isbn, etc)
  - Users
    - u::ender@dragon.com
    - u::enderdragon
  - Products
    - p::231-0321404314123 (isbn)
- Unpredictable Keys (GUID, UUID, etc) require Views (Map-Reduce indexes) to find their documents

Counter-ID Keys

- Similar to IDENTITY column in RDBMS
- Creating New Documents is a pair of operations in your application, increment and add
  - initialize one Key as an Atomic Counter (on app start)
  - increment counter and save new value
    - id = bucket. counter(“blog::slug::comment_count”, {delta: 1})
  - Use the id as component of the key for the new document
    - bucket.upsert(“blog::slug::c” + id, comment_data)

Lookup Pattern

- Create simple documents that has referential data (Key) to primary documents
  - Primary Document u::abc123-12345-4312
  - Lookup Document u::iron@golem.com = u::abc123-12345-4312
- Lookup documents aren't JSON, they should be the key as a string so you skip the JSON parsing
- Requires two get operations, first the lookup doc, then the primary doc
  - key = bucket.get("u::iron@golem.com")
  - doc = bucket.get(key)

RDBMS vs NoSQL

```sql
INSERT INTO Users
(name, facebookID, email, createdAt)
VALUES (
'Cave Spider', 100167, 'cave@spider.com',
'5/1/2000 2:30am')

--Get user by facebookID
SELECT * FROM Users
WHERE facebookID = 100167

--Get user by email
SELECT * FROM Users
WHERE email = 'cave@spider.com'
```

```javascript
uID = bucket.counter("user_counter", {delta:1});
bucket.insert(`u::${uID}`,userData);
bucket.insert(`em::${userData.email}`,uID);
bucket.insert(`fb::${userData.facebookID}`,uID);

//Get user by facebookID
uID = bucket.get('fb::100167');
userData = bucket.get(`u::${uID}`);

//Get user by email
uID = bucket.get('em::cave@spider.com');
userData = bucket.get(`u::${uID}`);
```

Gets slower with growth

Embedding vs Linking

Embedding

- Embed copies of frequently accessed related objects
  - Users friends into the users profile “friendlist”
  - Allows for quick fetch of users friendlist and all friend profiles in one read
- Denormalized

Embedding

- Can cause data duplication
- Updates are painful
- Easy to become inconsistent

Embedding

- user.friendsList = [{username: “han”, firstName: “Han”, lastName:”Solo”}, {username: “leia”,firstName:”Leia”, lastName:”Solo”}]

Linking

- user.friendsList = [“han”, “leia”]
- If you use linking you have to get the linked data yourself
- The DB Driver doesn’t do it for you, you can delete a link without the reference deleting

Embedding vs Linking

- Embedding when
  - there are no or few updates of the data
  - caching
  - when time-to-live is very restricted
- Otherwise link the related data
- Partial embedding is also possible

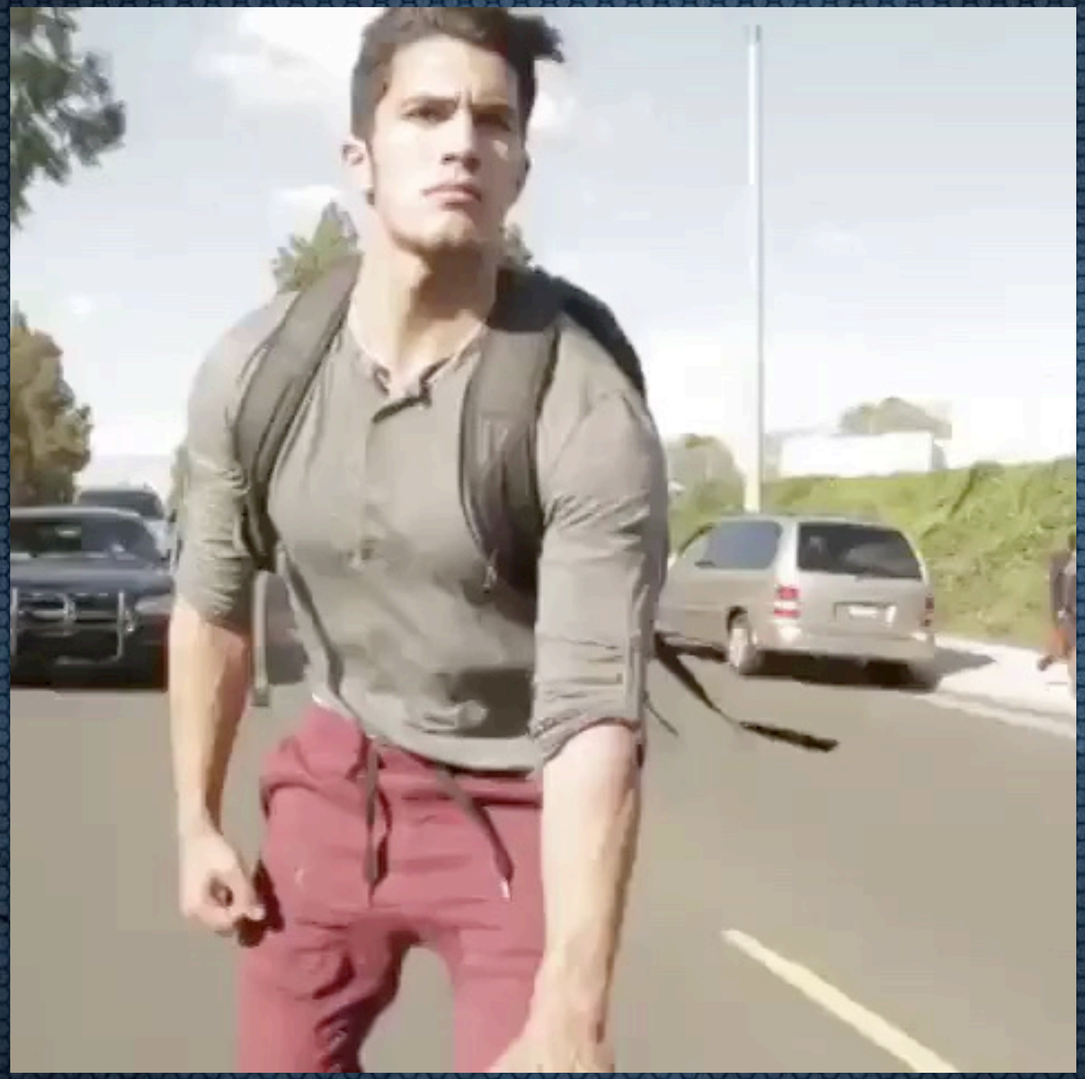
Parent-Referencing

- Logging data
- Lots of records
- Little need to access all the children

Rule of Thumb

- One - To - Few = Embedding
- One - To - Many = Linking
- One - To - Squillions = Parent Referencing

Making a mental shift



Making a mental shift

- In SQL we tend to want to avoid hitting the database as much as possible
  - we know that its costly when tying up connection pools and overloading db servers
  - even with indexing SQL still gets bogged down by complex joins and huge indexes
- In noSQL gets and sets ARE FAST, and not a bottleneck, this is hard for many people to accept and absorb at first

Gets and Sets are fast...

Gets and Sets are fast...

Gets and Sets are fast...

Gets and Sets are fast...

Making a mental shift

- Denormalization takes getting used to
- Data integrity
- Schema changes are easy

noSQL is Web Scale

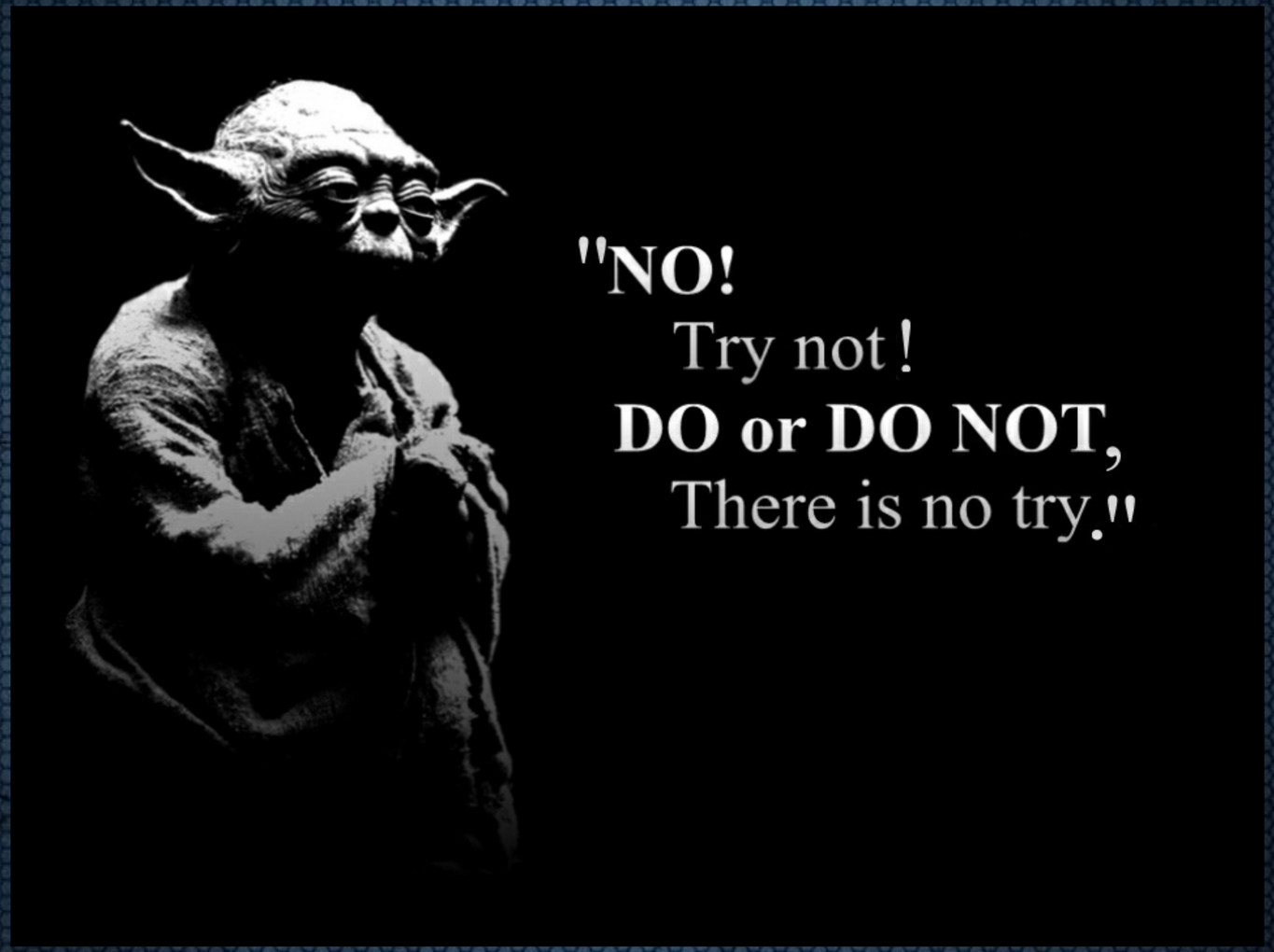
3 Big reasons for a NoSQL DB

- Big Data
- Impedance mismatch
- Frequently Changing schema

Polyglot Persistence

- Document DB where it makes sense
- Relational DB where it makes sense
- Graph DB where it makes sense

Simple e-Commerce Example



Specs

- Store order history (including price at the time of order)
- Keep product inventory
- Know the number of orders per product
- Know the number of orders per customer
- Human readable category URLs

Customer

- Email as key
- Embedded addresses
- Orders linking

Category

- slug as key
- Products count
- Orders count
- Products parent referencing

Product

- SKU as key
- Orders count
- Inventory
- Categories linking

Cart

- User Session ID as key
- Products Count
- Total
- Products embedded (partially)
- Status (active, completed, expired)

Order

- Order Counter for Key
- Embed customer info (partially)
- Embed product info (partially)
- Totals

Order

- Set cart status to completed
- decrement inventory, adjust “inCart”
- create/update customer record
- add order counter ID to customer orders
- Update order counts on category and product

More Information

- Introduction to NoSQL (Martin Folwer)
  - https://www.youtube.com/watch?v=qI_g07C_Q5I
- Data modeling in key/value and document database
  - http://openslava.sk/2014/Presentations/Data-modeling-in-KeyValue-and-Document-Stores_Philipp-Fehre.pdf
- NoSQL data modeling techniques
  - https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

“Thank you.”

–hold up applause sign here

Contact Info

- Email - gratzc@compknowhow.com
- Blog - https://ckhconsulting.com/blog/
- Twitter - gratzc
- LinkedIn - gratzc
- League of Legends - gratzc