2013/10/28

Managed designs – Mauro servienti

Mauro.servienti@Manageddesigns.it

Goals:

What is raven

How to use it

How to avoid screwing up

Comparison with other technologies

Problem: orm misuse

Polyglot persistence:

* several persistence technologies for different kinds of data
* need for it is a function of how many use cases a product implements (e.g. erp)

Example: Analysis services + oltp

* Uses ad-hoc data structure fit for a very specific purpose

Driver behind sql/e-r model adoption: hardware cost

Storage cost estimation should include backup costs

Examples of document-oriented db:

* Exchange
* Active directory
* Lotus notes

RavenDB exploits a **native windows** component: name?

Relational model problem: scaling (two examples)

* Performance (essential for cloud/saas)
* High availability (e.g. different timezones, slecht network prestaties) (essential for cloud/saas)

CAP Theorem

Consitency high performance high availability, you can only achieve TWO

(e.g. e-r model use case: high read/writes on same table)

Possible solution: eventual consistency (but how fast????)

e.g. you have inserted an invoice even though you can’t see it ☺ (i.e. ui-related problem)

PROBLEM: change orm isolation level depending on query, orm does not encourage you to think about it

**Missing in RDBMS:**

Failure mode (distributed data, one machine fails, what then?)

Write on a, read from b , data must be the same because we want consistency but what about performance???

**Sharding**: (feature-based data partitioning technique)

Example: geographically distributed data (2 branches: London, Milan, each one stores their orders in a geographically close location, distributed queries are transparent to the application layer)

Advantages: paged distributed queries

Application configures query behavior in case of failure e.g. a remote system is down:

* I want an exception
* I want partial results

Failed rdbms implementations

* Xml columns: kill rdbms
* Azure Table storage (limited by underlying SQL server)

(odata schema)

Document-based db are NOT hot-swappable

(e.g. nhibernate + user chooses backend)

Persistency-ignorance is not easily achieved, true for reading modules

Attributes: eav/vertical table

SQL Server solution: cte

Problem: how to get Nhibernate to understand it? 🡺 user type + dialect extension

Document-based db: you get it for nothing

**NOSQL**

Denormalised data, downside: storage needs, synchronisation

No schema, advantages: no big up front design + painless to change data models

Data versioning:

* Rdbms: application version tightly coupled to db schema (must deploy at the same time)
  + Case study: multinational clothes retailer
  + How can I update my backend???? I am screwed! (downtime 🡺 lost sales)
* Document-based db: incremental schema update + application is responsible for saving data in the correct format (i.e. read version 1.0 save version 1.1, use metadata to track version information, avoid polluting domain-related information with infrastructure-related information (i.e. versioning) )
  + Problem: your application manages data versioning
  + **Golden rule: no two applications use same db (enforce separation of application lifecycles, a soa tenet)**
    - **How can two applications share data safely? 🡪 apis**

Many if not all nosql dbs use json as data persistence format

Problem: json typesystem is limited, two different applications may interpret same data differently.

Sacrifice consistence to performance and scalability.

Ravendb: Master goes down, his slave becomes readonly

Cassandra/redis/couchdb: key/value pairs, combine with elastic search to perform lookups

Graphdb: look at relationships between different pieces of data (difficult to do with document-based db or rdbms) example: neo4j

Document: represents atomic information

No way of joining/relating two documents using a docu-based db.

Order rows: part of aggregate root Order 🡺 save as a unit 🡺 some doc-dbs do not support transactions (mongo)

BISON: binary json

No limit to serializable type complexity (via JSON.Net)

ACID Transactions !!! (on write, on read by key, not the case for QUERIES)

Aggregation (map/reduce)

Transformations (e.g. query on a table and apply a view to resultset BEFORE it is returned to caller)

**Doc-based dbs encourage you to use cqrs:**

* Write aggregates (golden rule)
* Read viewmodels (depending on use case)

(protocol buffer)

Slide deck 02

AGPL license

Web folder is iis ready

How to run RavenDB

1. On my machine (start.cmd) (q: close and exit)(console app)
   1. esent(esent, c:\windows\system32\eseutils.exe /?)
      1. Backup
      2. Defrag
      3. Recovery
   2. –browser
2. Hosted on iis
   1. Rest api ☺
   2. Advantage:
      1. no administrative permissions needed ☺ (esent takes care of writing)
      2. Iis takes you back in case of failure
   3. Disadvantages:
      1. little time to boot/shutdown my process 🡺truncation problem
      2. Recycled application pools: changed to always running (machine.config in 2008)
      3. Solution: choose a hosting provider that does not replace application pools
      4. Solution: use two web roles(one public one not ☺ )
3. Windows service:
   1. Shutdown time: configurable
   2. http: uses windows svc

New database configuration

* path (~ means relative to iis root/ server.exe directory) (ravendatadir)
* log (ravenlogsdir)
* indexes
* best put them on 3 different disks (esent is write intensive)
* what you need to backup
  + northwind/data: esent data
  + northwind/logs: esent logs
  + indexDefinition: indexes definitions (need to back this up!)
  + index: save this if you can to save time☺
* raveserver.exe.config:
  + Port
  + Ravenddatadir, ravenlogdir: read only by server process
* IMPORTANT:
  + Esent: forward but NOT BACKWARD COMPATIBLE!!!!!!!!!!!!!!!!!!! (i.e. win7 -> win xp boooom!)
    - (solution: smuggler: bison ->json->reimport)
    - Replicate using master/slave
  + Voron: new storage model

In system db there’s a DOCUMENT ☺ under ‘System documents’ for each db.

(e.g. create new db, examine contents of system db, same structure of server.config)

Read only at server startup, in a production scenario a replication-based solution is often preferred.

NO CROSS-DATABASE query (by default of course ☺ )

DBS share same server resources

Raw url shown at the bottom of RavenDB studio application (same urls you’d see using fiddler)

Of course there’s also an embeddable client (sql compact –like, hosted in application, can run in-memory, 100% compatible with ravendb therefore can perform 100% integrations tests in memory and replicate data on embedded version)

Replication is http-based

**Android build (!!!!!!!!!!)**

(ravendb file structure)

client/

* Lightweight
* Abstractions
* Mvcintegration -🡪 shows queries on your MVC app pages your app is performing
* Async targeting pack 🡪 add async/await to .net4 project

Backup/

Performs server-side calls

Document

Primary key: unique in a database!

People/ (prefix that represents a set modeled by that document)(convention over configuration: replace slash with something else in settings)

(advantage: rest-friendly urls)

Ids are generate using a hilo algorithm

If I get a document wit a definite key🡺update

Else (undefined key or slash-terminated 🡺 generate key)

Content –length shown on studio page is real document length (except for metadata)

WHY HTTP???

Advantages:

Caching (provided by proxies)

Authority of information (ask)

Transport compression (iis + apply gzip)

**Security**

Server authentication

1. Windows authentication
2. Basic authentication (http header
3. OAuth :) (expose oauth server from ravendb)
4. Levels: Admin(agpl) (, none,get not allowed with agpl license)

Settings tab on system DB:

Windows authentication/api key

BUILTIN\user (BUILTIN 🡺 .net convention to refer to local machine)

\*

System

Localsystem/networkservice not appropriate because they do not have Active dir read permission!

OAUTH: great in a hosting scenario (Azure/Amazon )

Same major version means compatibility between client and server!

DTC (distributed transactions are ) Supported!

Lookup documents: exploit id format ☺ (should not change in time) (can search by prefix)(up to this point we’re using it like Cassandra (k/value store)

Orders/Italy/Lombardy/Milan/1

{

“ordernr”:”128”

}

Dealing with concurrency

Solution: Etag (onderdeel van metadata) (transformed datetime, problems?)

Metadata

* Dictionary
* Transmitted via http
* Sommige onderdelen ervan kan je niet wijzigen
* Raven entity name: used by ravendb studio ( ui) to show similar kinds of data

What you transmit via http has to be http-compatible (characters!)

Transactions protect metadata as well

ETag: involved in caching ☺

Example:

1st load: add etag in http request

2nd load: head http call 🡺 proxy returns cache info☺

**Relationships**

supported by using KEYS (see example)

NO ENFORCEMENT OF FOREIGN-KEY CONSTRAINTS !!!!

{

“some data”:”some value”,

users:[“Users/1”,”Users/2”] //models a relationship between docs, on the application level but RavenDB

}

Q: Why denormalising data makes sense??

A: avoid having to perform a query

Ex:

{

“some data”:”some value”,

Users:[

{

“username”:”A”

},{

“username”:”B”

}

] //models a relationship between docs, on the application level but RavenDB

}

SELECT N+1 problem komt ook voor in RavenDB!

(als je gebruik van een foreign key maakt!)

Relations (ask)

RavenDB is safe by default

Problem: production load scenarios different from development load scenarios🡺kaboom!

Solution:

* Paging: Ravendb forces you to paginate data (max 128 records)
* Self-tuning: behavior-driven automatic index refactoring
  + Problem: if behavior oscillates between two extremes then indexes get thrashed quickly and you spend most of your time rebuilding them