# Musings on data access

Colin Bradford
cjbradford@gmail.com

# Problem

- Data is relational
- SQL lets us describe the relations
- But the results are a rectangular table

# Example

- Journey has multiple stops
- Journey has multiple passengers
- Joining both tables gives stops x passengers rows
- Need to deduplicate the data

# Possibilities

- Multiple queries
- UNION queries with nulls
- Others?

# Alternative – describe the result

```
{
   "name" : "trip",
   "index" : {
      "name" : "trip_id_index",
      "in" : [ 5744 ]
   },
   "join" : [
      {
         "name"  : "stops",
         "index" : {
            "name"          : "PRIMARY",
            "parent_column" : "trip_id"
         }
      },
      {
         "name"  : "passengers",
         "index" : {
            "name"          : "PRIMARY",
            "parent_column" : "trip_id"
         }
      }
   ]
}
```

# Which gives

```
[
    {
        trip_id => 5744,
        customer_id => 6455,
        stops => [
            { address => 'BBC, W1A 1AA', },
            { address => 'Euston Station', },

        ],
        passengers => [
            { name    => 'Mr J. Smith', },
            { name    => 'Mr A. N. Other', }
        ]
    },
]
```

# Or (faster over the wire)

```
{
    data => [
        { trip_id => 5744, customer_id => 6455, },
    ],
    stops => {
        5744 => [
            { address => 'BBC, W1A 1AA', },
            { address => 'Euston Station', }

        ]
    },
    passengers => {
        5744 => [
            { name    => 'Mr J. Smith', },
            { name    => 'Mr A. N. Other', }
        ]
    },
}
```

# Implementation

- Do joins in Perl
  - which means they run on the client, not the server
- Can target multiple back end data stores
  - SQL
  - Redis
  - MongoDB
  - CouchDB
  - HandlerSocket

# HandlerSocket?

- Access into the InnoDB storage engine of MySQL
- Lightweight protocol
- Can only look up on indexed columns
- Fast
- But Perl interface slows it down

# So, does it work?

- Yes (ish), for reads only
- Initial implementation on GitHub: https://github.com/cjbradford/DataKeyValue
- Not performance optimised

# Advantages

- Can join across data stores
- Can join with data stores that don't support joins
- Can layer: Add a shard layer over another backend
- Load appears to move from the data store to the client

# Performance

- Measured using Benchmark.pm
- Results open to interpretation
- Data set is as similar as I can make it – not optimised for Key/Value data store
- Data stores running on the same machine as the client
- Single SATA disk, 8Gb ram, i5-2500K CPU @3.3GHz

# Query with a join

```
SELECT p.chanid, p.starttime, p.endtime, p.title, p.subtitle,
p.description, c.chanid,
c.channum, c.callsign, c.sourceid, c.serviceid, c.mplexid
FROM program p
LEFT JOIN channel c ON p.chanid = c.chanid
WHERE starttime >= ? AND starttime <= ?
(123 rows back)

   CouchDB: 127 wall ( 4.78 usr 0.24 sys) @    4.98/s (n=25)
   MongoDB:   8 wall ( 5.13 usr 0.68 sys) @   65.92/s (n=383)
     Redis:   6 wall ( 4.35 usr 0.88 sys) @  138.62/s (n=725)
   HSocket:   8 wall ( 5.32 usr 0.08 sys) @  375.19/s (n=2026)
PerlMyISAM:   5 wall ( 5.27 usr 0.03 sys) @  543.77/s (n=2882)
PerlInnoDB:   5 wall ( 5.05 usr 0.00 sys) @  546.53/s (n=2760)
 SQLMyISAM:   6 wall ( 5.12 usr 0.09 sys) @ 1025.91/s (n=5345)
 SQLInnoDB:   5 wall ( 5.14 usr 0.02 sys) @ 1035.85/s (n=5345)
```

# Single table, lots of rows

```
SELECT p.chanid, p.starttime, p.endtime, p.title, p.subtitle,
p.description
FROM program p
WHERE starttime BETWEEN ? AND ?
(123 rows)

   CouchDB:  50 wall ( 5.01 usr 0.36 sys) @   83.24/s (n=447)
   MongoDB:   5 wall ( 4.97 usr 0.03 sys) @  383.40/s (n=1917)
     Redis:   6 wall ( 5.10 usr 0.18 sys) @  410.61/s (n=2168)
PerlMyISAM:   5 wall ( 5.15 usr 0.00 sys) @ 1243.69/s (n=6405)
PerlInnoDB:   6 wall ( 5.21 usr 0.02 sys) @ 1247.23/s (n=6523)
 SQLInnoDB:   5 wall ( 5.28 usr 0.03 sys) @ 1251.60/s (n=6646)
 SQLMyISAM:   5 wall ( 5.23 usr 0.07 sys) @ 1255.85/s (n=6656)
   HSocket:  12 wall ( 6.27 usr 0.26 sys) @ 1365.08/s (n=8914)
```

# Single table, one row

```
SELECT c.chanid, c.channum, c.callsign, c.sourceid, c.serviceid,
c.mplexid
FROM channel c
WHERE chanid BETWEEN ? AND ?
(1 row)

   CouchDB: 133 wall ( 4.80 usr 0.25 sys) @   327.72/s (n=1655)
   MongoDB:   7 wall ( 5.24 usr 0.56 sys) @  8506.72/s (n=49339)
     Redis:   6 wall ( 4.06 usr 1.18 sys) @ 12241.41/s (n=64145)
 SQLInnoDB:   9 wall ( 6.77 usr 0.72 sys) @ 12926.17/s (n=96817)
PerlMyISAM:   6 wall ( 4.60 usr 0.53 sys) @ 15192.98/s (n=77940)
PerlInnoDB:   6 wall ( 4.86 usr 0.37 sys) @ 15373.23/s (n=80402)
 SQLMyISAM:   7 wall ( 5.61 usr 0.56 sys) @ 15691.57/s (n=96817)
   HSocket:   7 wall ( 3.77 usr 1.48 sys) @ 30631.43/s (n=160815)
```

# Single table, PK lookup

```
SELECT c.chanid, c.channum, c.callsign, c.sourceid, c.serviceid,
c.mplexid
FROM channel c
WHERE chanid = ?
(1 row)

   CouchDB: 136 wall ( 4.89 usr 0.36 sys) @   644.19/s (n=3382)
   MongoDB:   7 wall ( 4.33 usr 0.92 sys) @ 12161.52/s (n=63848)
 SQLMyISAM:   6 wall ( 5.24 usr 0.49 sys) @ 16896.51/s (n=96817)
PerlMyISAM:   7 wall ( 4.78 usr 0.47 sys) @ 17568.57/s (n=92235)
PerlInnoDB:   5 wall ( 4.64 usr 0.36 sys) @ 17757.20/s (n=88786)
 SQLInnoDB:   6 wall ( 4.88 usr 0.36 sys) @ 18476.53/s (n=96817)
     Redis:   6 wall ( 3.98 usr 1.19 sys) @ 23353.97/s (n=120740)
   HSocket:   7 wall ( 3.55 usr 1.57 sys) @ 29809.77/s (n=152626)
```

# Closing thoughts

- So far, idea isn't obviously broken
- In most of the applications I work with:
  - Most writes are to a single “table”
  - Transactions can be worked around
  - Moving load to clients is worthwhile

Thank you