Beyond MySQL

Lorna Jane Mitchell, IBM Analytics

CodeMotion Berlin 2016

http://www.lornajane.net/resources

Beyond MySQL

MySQL is great!

If you're ready for something different, how about:

- PostgreSQL
- Redis
- CouchDB

PostgreSQL

About PostgreSQL

Homepage: https://www.postgresql.org/

- Open source project
- Powerful, relational database

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

Not true. They are both approachable from both CLI and other web/GUI tools, PostgreSQL has the best CLI help I've ever seen.

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

Not true. They are both approachable from both CLI and other web/GUI tools, PostgreSQL has the best CLI help I've ever seen.

Myth 2: PostgreSQL is more strict than MySQL

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

Not true. They are both approachable from both CLI and other web/GUI tools, PostgreSQL has the best CLI help I've ever seen.

Myth 2: PostgreSQL is more strict than MySQL

True! But standards-compliant is a feature IMO

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

Not true. They are both approachable from both CLI and other web/GUI tools, PostgreSQL has the best CLI help I've ever seen.

Myth 2: PostgreSQL is more strict than MySQL

True! But standards-compliant is a feature IMO

Myth 3: PostgreSQL is slower than MySQL for simple things

PostgreSQL Myths and Surprises

Myth 1: PostgreSQL is more complicated than MySQL

Not true. They are both approachable from both CLI and other web/GUI tools, PostgreSQL has the best CLI help I've ever seen.

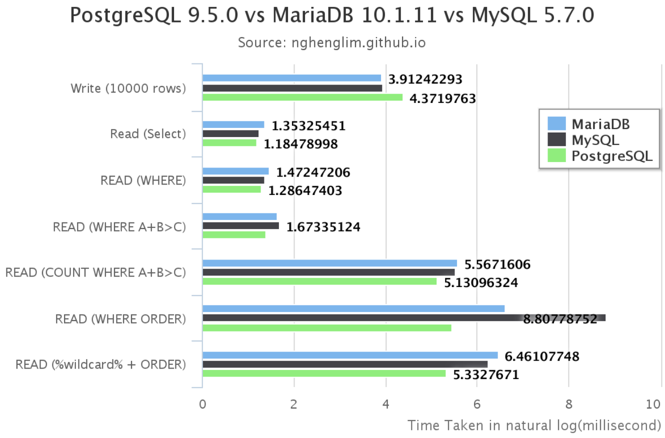
Myth 2: PostgreSQL is more strict than MySQL

True! But standards-compliant is a feature IMO

Myth 3: PostgreSQL is slower than MySQL for simple things

Not true. PostgreSQL has better query planning so is likely to be faster at everything, and also has more features.

PostgreSQL Performance



Additional Data Types: UUID

PostgreSQL has a UUID data type to create unique identifiers

We can use it as a primary key:

```
CREATE TABLE products (
  product_id uuid primary key default uuid_generate_v4(),
  display_name varchar(255)
);

INSERT INTO products (display_name)
  VALUES ('Jumper') RETURNING product_id;
```

(you may need to `create extension "uuid-ossp"` first)

Additional Data Types: UUID

Look in the table:

```
             product_id              | display_name
--------------------------------------+--------------
73089ae3-c0a9-4c0a-8287-e0f6ec41a200 | Jumper
```

RETURNING Keyword

Look at that insert statement again

```
INSERT INTO products (display_name)
  VALUES ('Jumper') RETURNING product_id;
```

The `RETURNING` keyword allows us to retrieve a field in one step - removes the need for a `last_insert_id()` call.

Additional Data Types: array and hstore

Add some more interesting columns to the table:

```
ALTER TABLE products ADD COLUMN depts varchar(255)[];
```

```
ALTER TABLE products ADD COLUMN attrs hstore;
```

(you may need to enable hstore with `create extension hstore`)

Additional Data Types: array and hstore

Insert some data into the table

```
INSERT INTO products (display_name, depts, attrs)
  VALUES ('T-Shirt', '{"kids"}',
  'colour => red, size => L, pockets => 1');
```

```
display_ |     depts       |                    attrs
---------+-----------------+------------------------------------------
 Jumper  |                 |
 T-Shirt | {kids}          | "size"=>"L", "colour"=>"red", "pockets"=>"1"
 Hat     | {kids,holiday} | "colour"=>"white"
```

Additional Data Types: array and hstore

We can fetch data using those fields

```
SELECT display_name FROM products
  WHERE 'kids' = ANY(depts);

SELECT display_name FROM products
  WHERE attrs->'colour' = 'red';
```

Common Table Expressions (CTE)

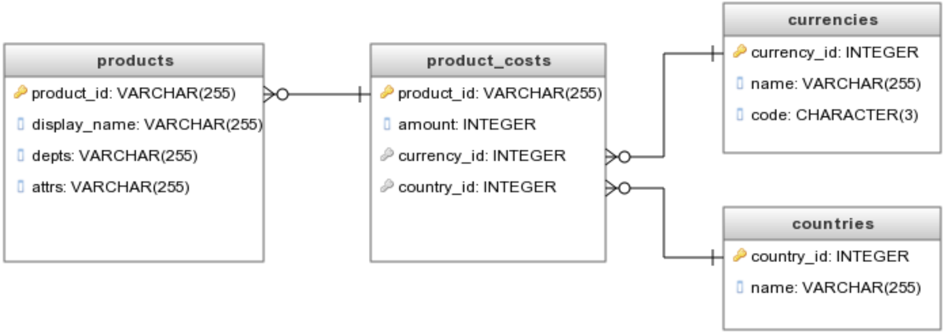
Feature enables declaring extra statements to use later

Moves complexity out of subqueries, making more readable and reusable elements to the query

Syntax:

```
WITH meaningfulname AS
  (subquery goes here joining whatever)
SELECT .... FROM meaningfulname ...
```

Common Table Expressions (CTE)



Common Table Expressions (CTE)

```
WITH costs AS
  (SELECT pc.product_id, pc.amount, cu.code, co.name
  FROM product_costs pc JOIN currencies cu USING (currency_id)
  JOIN countries co USING (country_id))
SELECT display_name, amount, code currency, name country
  FROM products JOIN costs USING (product_id);
```

```
display_name | amount | currency | count
-------------+--------+----------+---------
T-Shirt      |     25 | GBP      | UK
T-Shirt      |     30 | EUR      | Italy
T-Shirt      |     29 | EUR      | France
```

Window Functions

Window functions allow us to calculate aggregate values while still returning the individual rows.

e.g. a list of orders, including how many of this product were ordered in total

Window Functions

```
SELECT o.order_id, p.display_name,
  count(*) OVER (PARTITION BY product_id) AS prod_orders
FROM orders o JOIN products p USING (product_id);
```

```
               order_id                | display_name | prod_orders
---------------------------------------+--------------+-------------
 74806f66-a753-4e99-aeae-6d491f947f08 | T-Shirt      |           6
 9ae83b3f-931e-4e6a-a8e3-93dcf10dd9ab | Hat          |           3
 0030c58a-122c-4fa5-90f4-21ad531d3848 | Hat          |           3
 3d5a0d76-4c7e-433d-b3cf-288ef473912d | Hat          |           3
```

PostgreSQL Tips and Resources

- PhpMyAdmin equivalent: https://www.pgadmin.org/
- Best in-shell help I've ever seen (type `\h [something]`)
- JSON features
- Indexes on expression
- Choose where nulls go by adding `NULLS FIRST|LAST` to your `ORDER BY`
- Fabulous support for geographic data http://postgis.net/
- Get a hosted version from http://compose.com

Redis

About Redis

Homepage: http://redis.io/

Stands for: REmote DIctionary Service

An open source, in-memory datastore for key/value storage, and much more

Uses of Redis

Usually used in addition to a primary data store for:

- caching
- session data
- simple queues

Anywhere you would use Memcache, use Redis

Redis Feature Overview

- stores strings, numbers, arrays, sets, geographical data …
- supports key expiry/lifetime
- great monitoring tools
- very simple protocols

Tools

Install the `redis-server` package and run it.

Be a spectator: `telnet localhost 6379` then type `monitor`

Command line: `redis-cli`

Storing Key/Value Pairs

Store, expire and fetch values.

```
> set risky_feature on
OK
> expire risky_feature 3
(integer) 1
> get risky_feature
"on"
> get risky_feature
(nil)
```

Shorthand for set and expire: `setex risky_feature 3 on`

Storing Hashes

Use a hash for related data (`h` is for hash, `m` is for multi)

```
> hmset featured:hat name Sunhat colour white
OK
> hkeys featured:hat
1) "name"
2) "colour"
> hvals featured:hat
1) "Sunhat"
2) "white"
```

Finding Keys in Redis

The `SCAN` keyword can help us find things

```
127.0.0.1:6379> hset person:lorna twitter lornajane
(integer) 1
127.0.0.1:6379> scan 0 match person:*
1) "0"
2) 1) "person:Lorna"
2) "person:lorna"
127.0.0.1:6379> hscan person:lorna 0
1) "0"
2) 1) "twitter"
2) "lornajane"
```

Configurable Durability

This is a tradeoff between risk of data loss, and speed.

- by default, redis snapshots (writes to disk) periodically
- the snapshot frequency is configurable by time and by number of writes
- use the `appendonly` log to make redis *eventually durable*

Redis: Tips and Resources

- Replication is simple!
- Clustering needs external tools but is also fairly easy
- Sorted sets
- Supports pub/sub:
    - `SUBSCRIBE comments` then `PUBLISH comments message`
- Excellent documentation http://redis.io/documentation
- Get a hosted version from http://compose.com

CouchDB

About CouchDB

Homepage: http://couchdb.apache.org/

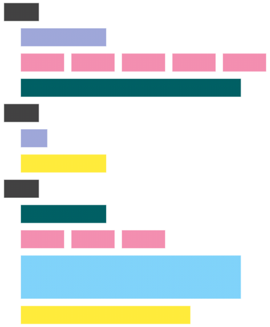
A database built from familiar components

- HTTP interface
- Web interface *Fauxton*
- JS map/reduce views

CouchDB is a Document Database

Schemaless Database Design

We can store data of any shape and size



Documents and Versions

When I create a record, I supply an `id` and it gets a `rev`:

```
$ curl -X PUT http://localhost:5984/products/1234
  -d '{"type": "t-shirt", "dept": "womens", "size": "L"}'

{"ok":true,"id":"1234","rev":"1-bce9d948a37e72729e689145286fd3ee"}
```

(alternatively, `POST` and CouchDB will generate the id)

Update Document

CouchDB has awesome consistency management

To update a document, supply the `rev`:

```
$ curl -X PUT http://localhost:5984/products/1234
  -d '{"_rev": "1-bce9d948a37e72729e689145286fd3ee",
  "type": "t-shirt", "dept": "womens", "size": "XL"}'

{"ok":true,"id":"1234","rev":"2-4b8a7e1bde15d4003aca1517e96d6cfa"}
```

Replication

CouchDB has the best database replication options imaginable:

- ad-hoc or continuous
- one directional or bi directional
- conflicts handled safely (best fault tolerance ever)

CouchDB Views

Querying CouchDB needs forward planning

- no ad-hoc queries
- create views and use them
- mapreduce in javascript

MapReduce

1. Work through the dataset (filtered if appropriate)
2. From those, output some initial keys and values (this is the **map**)
3. Records from step 2 with the same keys get grouped into buckets
4. The buckets are each processed by a **reduce** function to produce the output

CouchDB Views: Example

A view is made of Map and Reduce functions, written in JavaScript

Map:

```
function (doc) {
    emit([doc.dept, doc.type], 1);
}
```

Reduce: try COUNT, SUM or STATS

CouchDB Views: Example

http://localhost:5984/products/_design/products/_view/count?group=true

```
{"rows":[
  {"key":["mens","t-shirt"],"value":1},
  {"key":["womens","bag"],"value":3},
  {"key":["womens","shoes"],"value":1},
  {"key":["womens","t-shirt"],"value":2}
]}
```

CouchDB Views: Example

http://localhost:5984/products/_design/products/_view/count?group_level=1

```
{"rows":[
  {"key":["mens"],"value":1},
  {"key":["womens"],"value":6}
]}
```

Changes API

Get a full list of newest changes since you last asked

http://localhost:5984/products/_changes?since=7

```
~ $ curl http://localhost:5984/products/_changes?since=7
{"results":[
{"seq":9,"id":"123",
    "changes":[{"rev":"2-7d1f78e72d38d6698a917f8834bfb5f8"}]}
],
```

Polling/Long polling or continuous change updates are available, and they can be filtered.

CouchDB Tips and Resources

- CouchDB Definitive Guide http://guide.couchdb.org
- New CouchDB 2.0 release
    - open source, includes Cloudant features
    - has sharding, scalability features
- Javascript implementation https://pouchdb.com/
- My CouchDB + PHP Tutorial on developer.ibm.com
- Get a hosted version from http://cloudant.com

Beyond MySQL

Thanks

Slides: http://lornajane.net/resources

Further reading: Seven Databases in Seven Weeks

Contact:

- `lorna.mitchell@uk.ibm.com`
- @lornajane