Handler

and NoSQL ;-)

SQL is cool

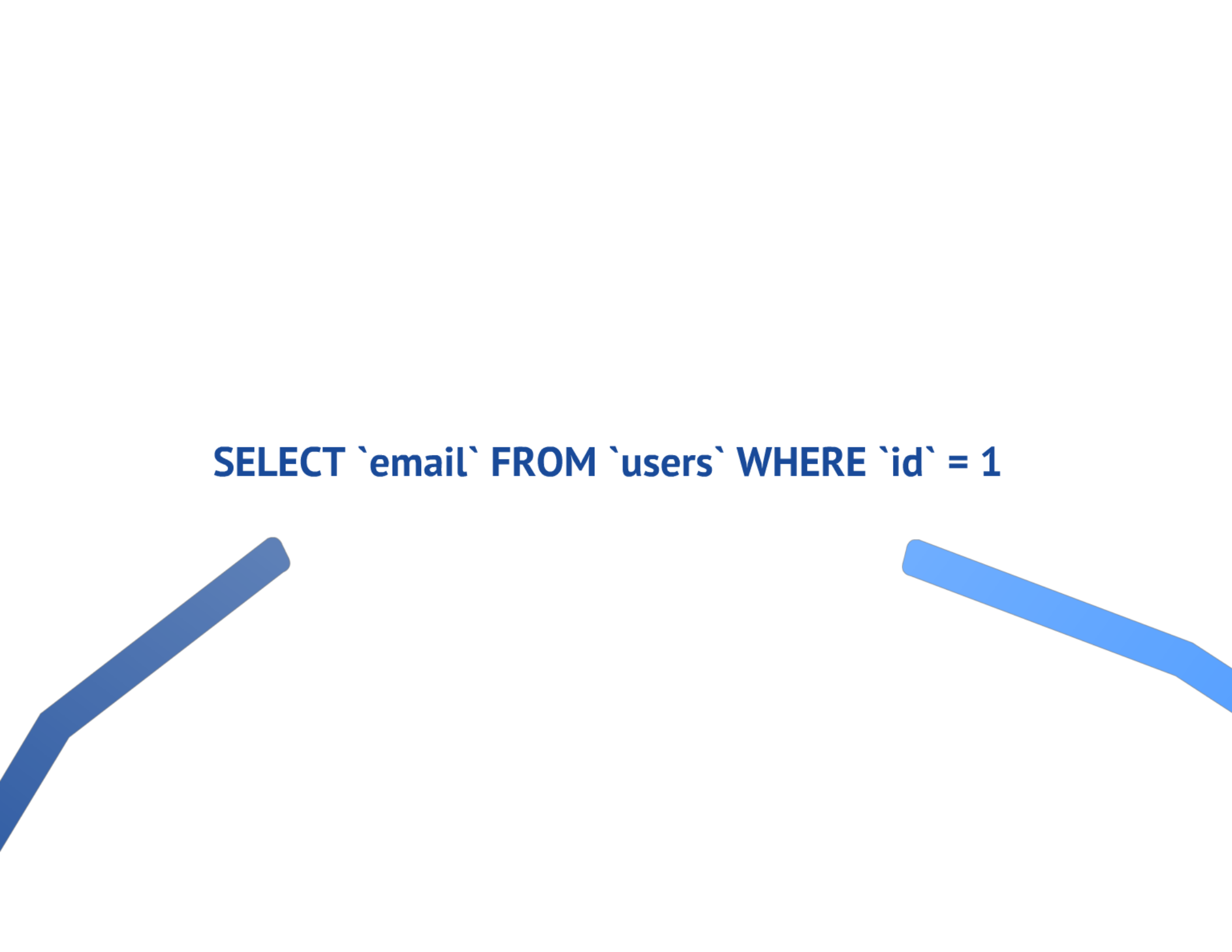
Complicated...

```
SELECT sr.receiving_id, sc.collection_id FROM
stock_collection as sc, stock_requisation as srq,
stock_receiving as sr WHERE (sc.stock_id = '" .
strStockID . "' AND sc.datemm_issued = '" . strM
AND sc.qty_issued >= 0 AND sc.collection_id =
srq.requisition_id AND srq.active_status = 'Activ
(sr.stock_id = '" . strStockID . "' AND
sr.datemm_received = '" . strMM . "' AND
sr.qty_received >= 0)
```

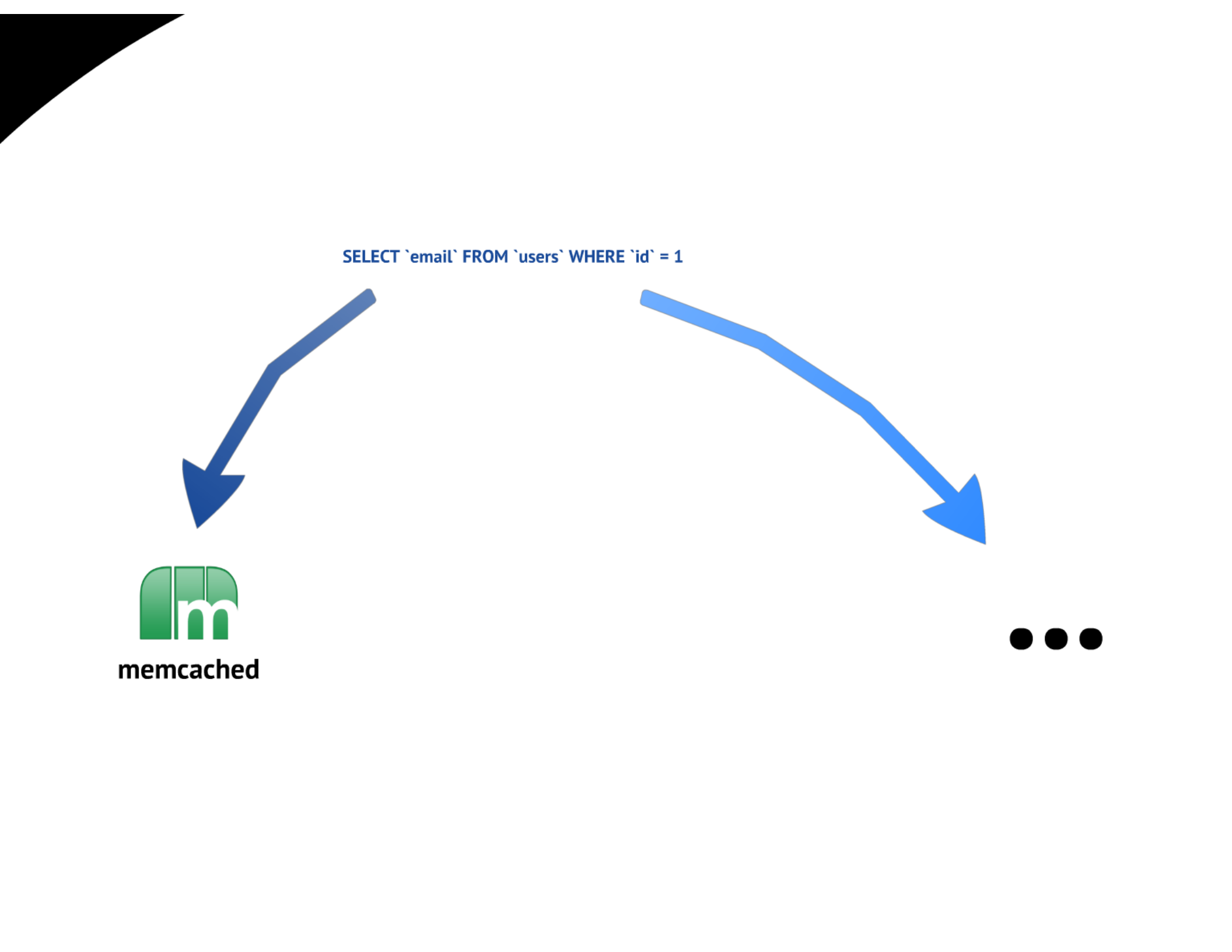
Has lots of cool stuff

- Aggregate functions
- Subqueries
- JOINs
- Complicated WHERE conditions
- Transactions
- ...

But for a general website most queries are simple

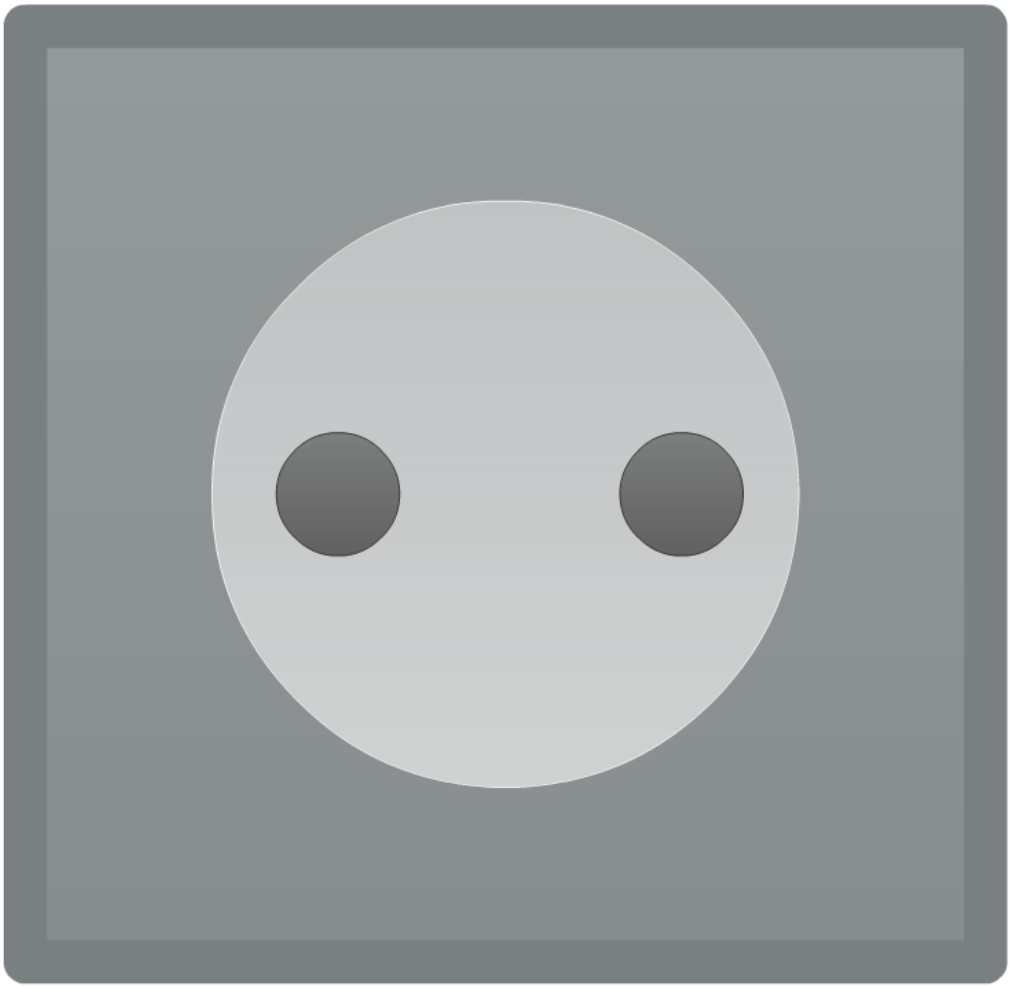

SELECT `email` FROM `users` WHERE `id` = 1

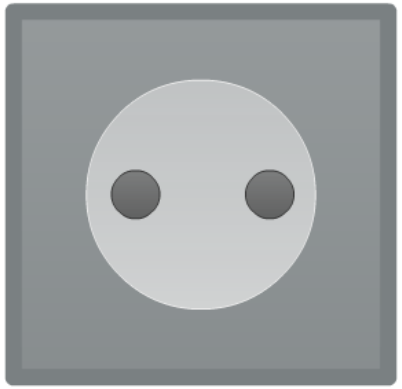
memcached


SELECT `email` FROM `users` WHERE `id` = 1
memcached

e simple

H



What H  is...

MySQL plug-in with direct access to InnoDB/XtraDB

- **Opens its own socket**
  - **Own protocol, own commands**

No SQL support.
None at all.

Pros:

- faster
- batch-processes requests
- compact protocol
- works okay with 10K connections
- doesn't disable SQL access
- can be used with replication
- bundled with Percona Server

So there is

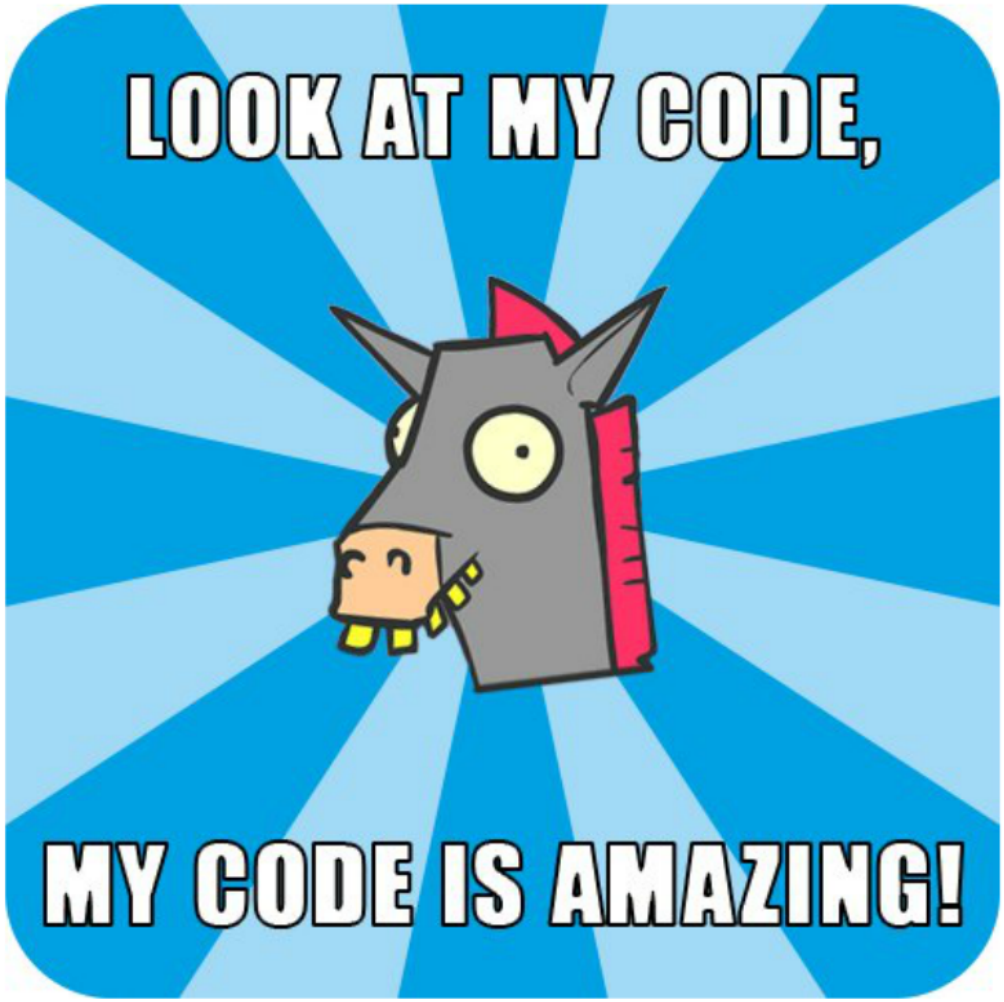
no need for memcached → extra memory

no duplicate data → it's now consistent

Cons:

- InnoDB/XtraDB only, partly supports others
- no transactions/stored procedures
- some data types/core MySQL features are not supported
- no commercial support
- immature
- blocks table modifications, locking

Sometimes that buggy:

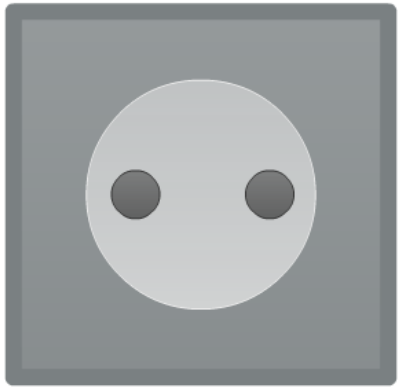


and also...

- unclear docs
- work logic & protocol change from time to time w/o any warning

Cons:

- InnoDB/XtraDB only, partly supports others
- no transactions/stored procedures
- some data types/core MySQL features are not supported
- no commercial support
- immature
- blocks table modifications, locking

What H  is...

NOT

Not a key-value/document store

Not a ‘binary SQL’ server

Not a tool for complex queries

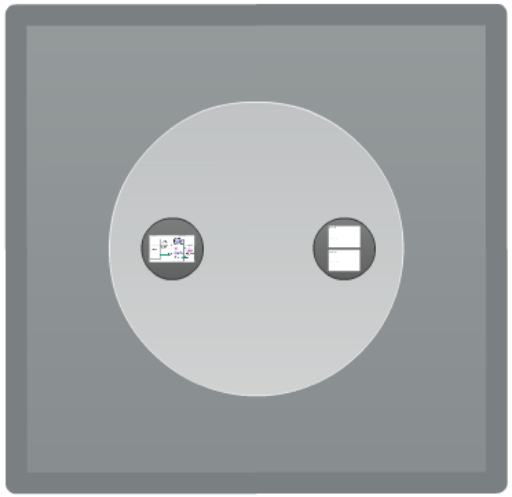
Not a tool for complex queries

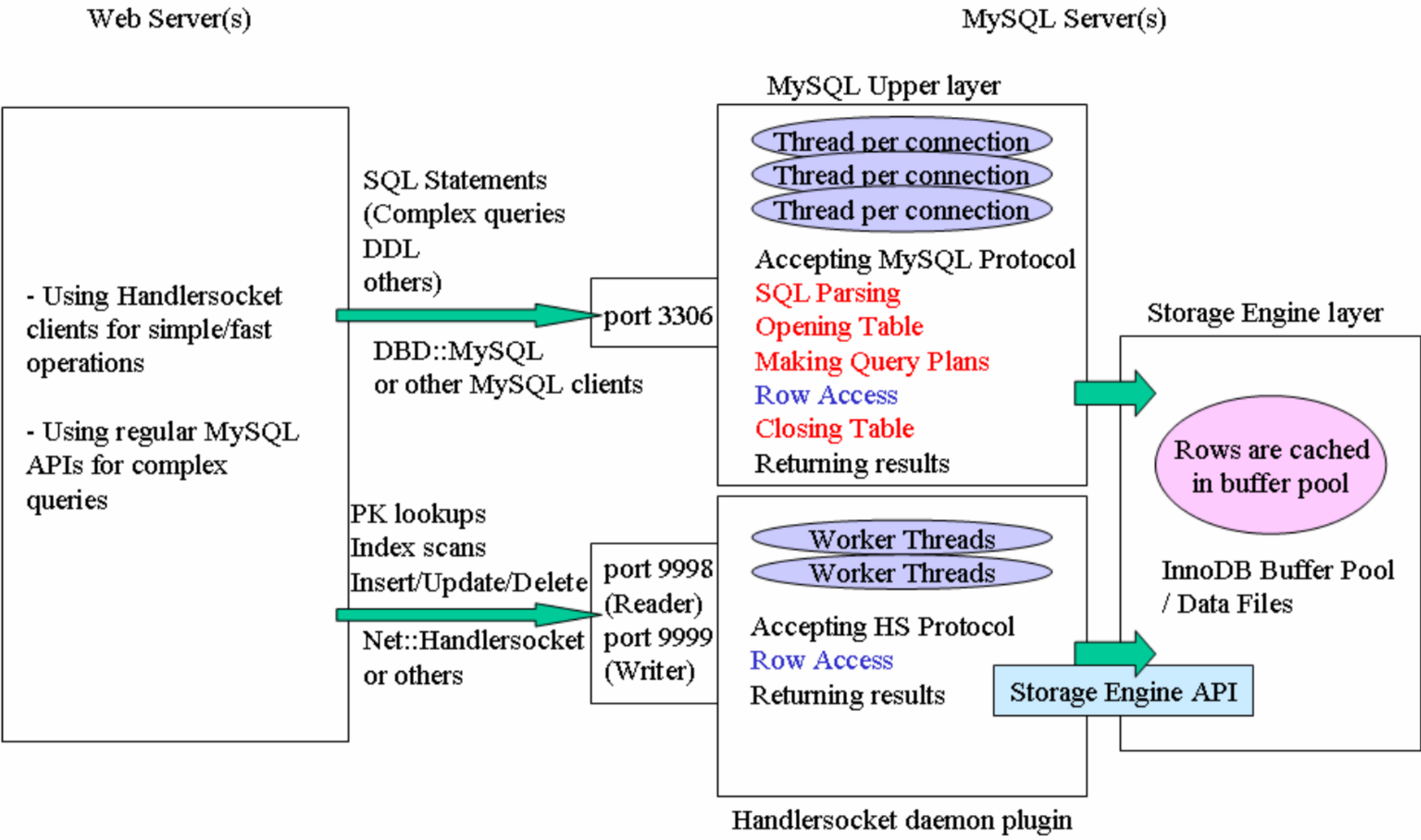
Not a tool for creating/modifying tables

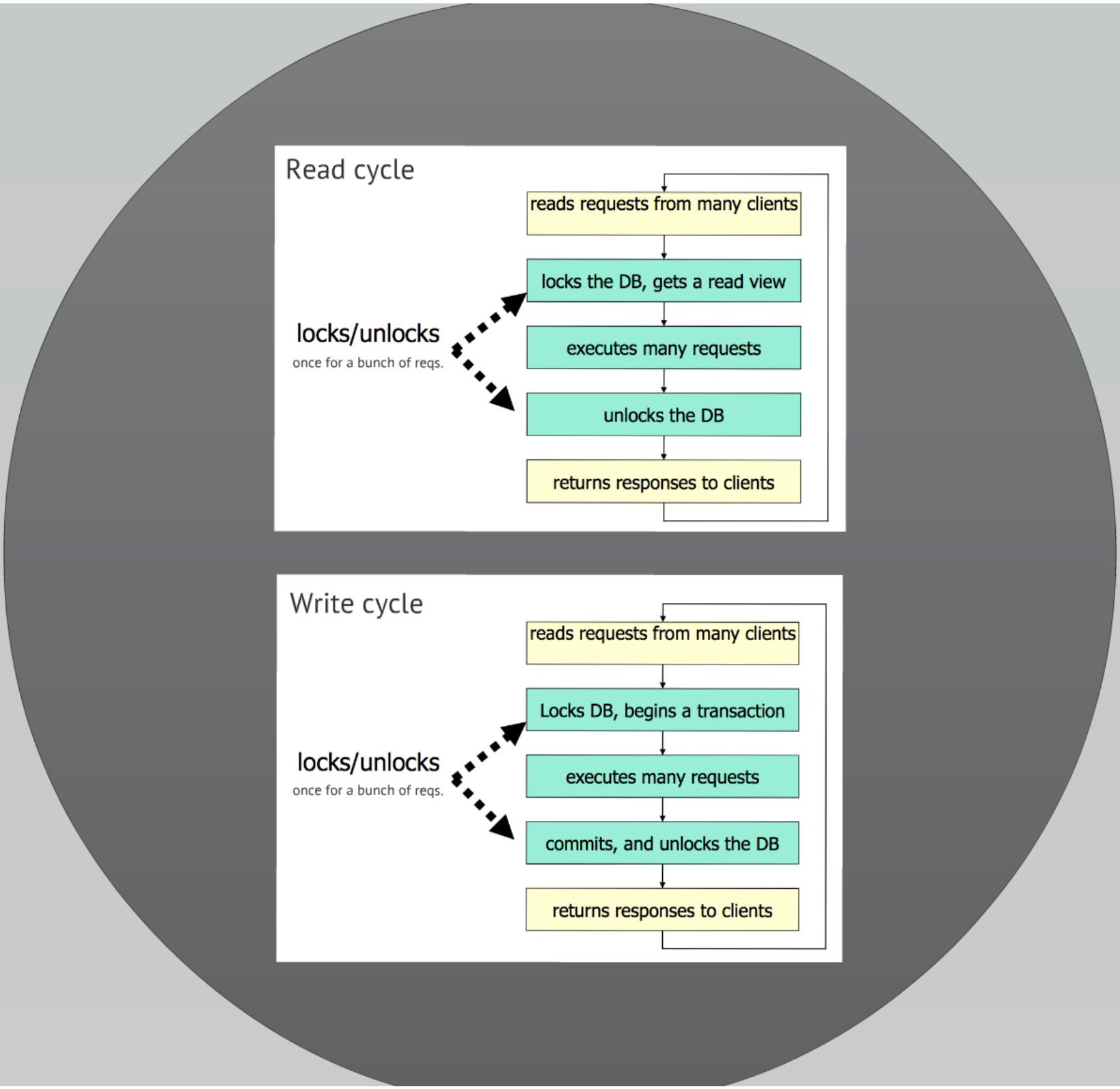
Not for hosting/shared usage

Not for hosting/shared usage

Inside H



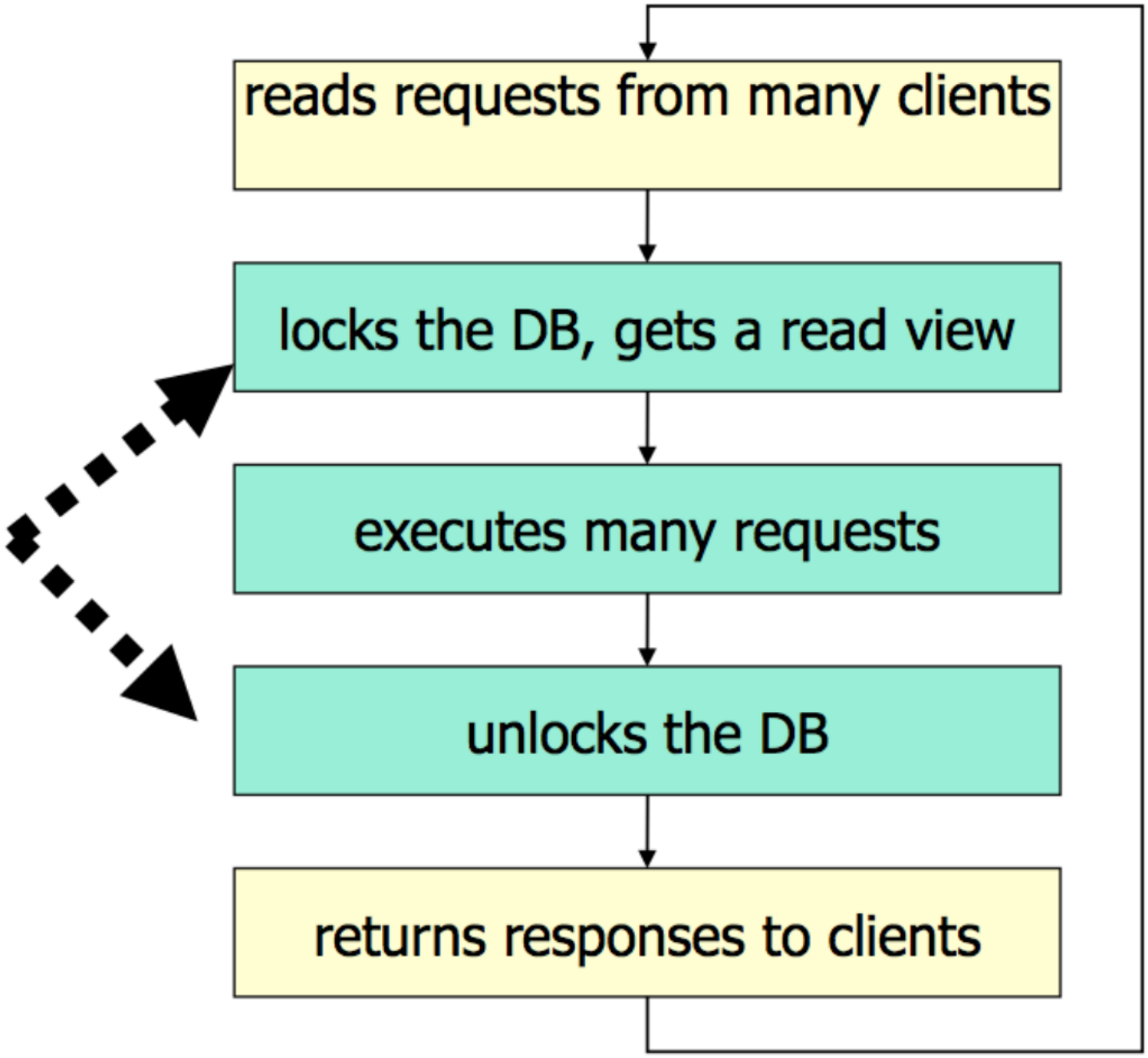

Web Server(s)
MySQL Server(s)
MySQL Upper layer
- Using Handlersocket clients for simple/fast operations
- Using regular MySQL APIs for complex queries
SQL Statements
(Complex queries
DDL
others)
DBD::MySQL
or other MySQL clients
port 3306
Thread per connection
Thread per connection
Thread per connection
Accepting MySQL Protocol
SQL Parsing
Opening Table
Making Query Plans
Row Access
Closing Table
Returning results
Storage Engine layer
Rows are cached in buffer pool
InnoDB Buffer Pool / Data Files
PK lookups
Index scans
Insert/Update/Delete
Net::Handlersocket
or others
port 9998
(Reader)
port 9999
(Writer)
Worker Threads
Worker Threads
Accepting HS Protocol
Row Access
Returning results
Storage Engine API
Handlersocket daemon plugin


Read cycle
reads requests from many clients
locks the DB, gets a read view
locks/unlocks
once for a bunch of reqs.
executes many requests
unlocks the DB
returns responses to clients
Write cycle
reads requests from many clients
Locks DB, begins a transaction
locks/unlocks
once for a bunch of reqs.
executes many requests
commits, and unlocks the DB
returns responses to clients

Read cycle

locks/unlocks

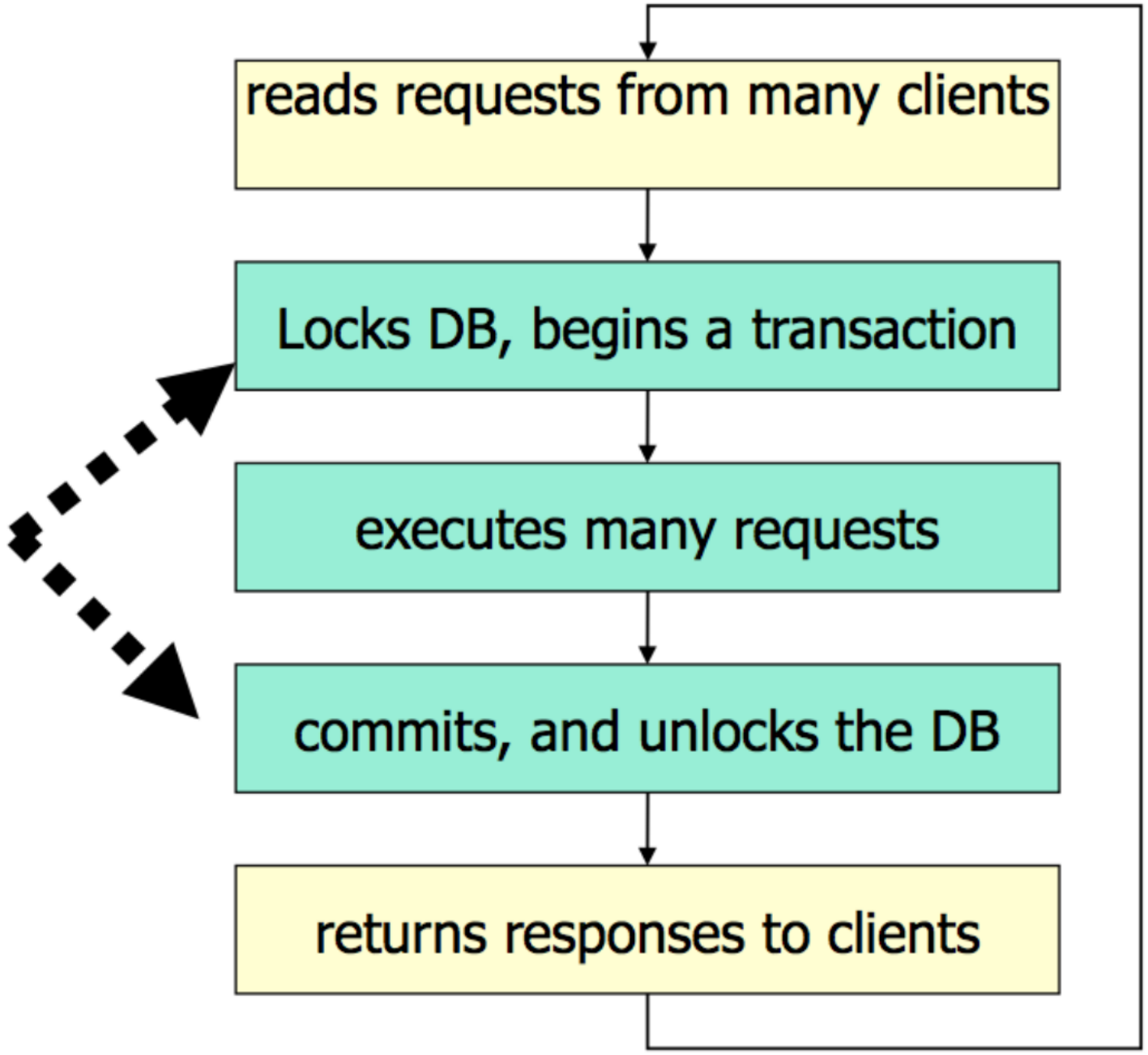
once for a bunch of reqs.



Write cycle

locks/unlocks

once for a bunch of reqs.



Handlersocket-MySQL interoperability

- you read consistent data via HS and via SQL
- HS works okay with MySQL replication
- auto_increment is supported
- current builds invalidate query cache
- MySQL users & permissions are not used in HS
- table locking via HS & via SQL will conflict

So, be careful with:

- ‘LOCK TABLES ... WRITE’
- ‘ALTER TABLE ...’

XtraBackup also won't work.

BTW, it's a plugin, so this should work
(as seen on the internets):

```
install plugin handlersocket soname
'handlersocket.so';

uninstall plugin handlersocket;
```

In practice usually hangs entire DB.

Using Handlersocket

longest chapter...

You get  ports open: 9998, 9999

read only

Workflow:

→ Client initiates connection
→ Client sends request
← Server sends response
→ Client sends next request
...

(1 request → 1 response)

You can send N requests in bulk, you will receive N responses in the order of requests.

Protocol:

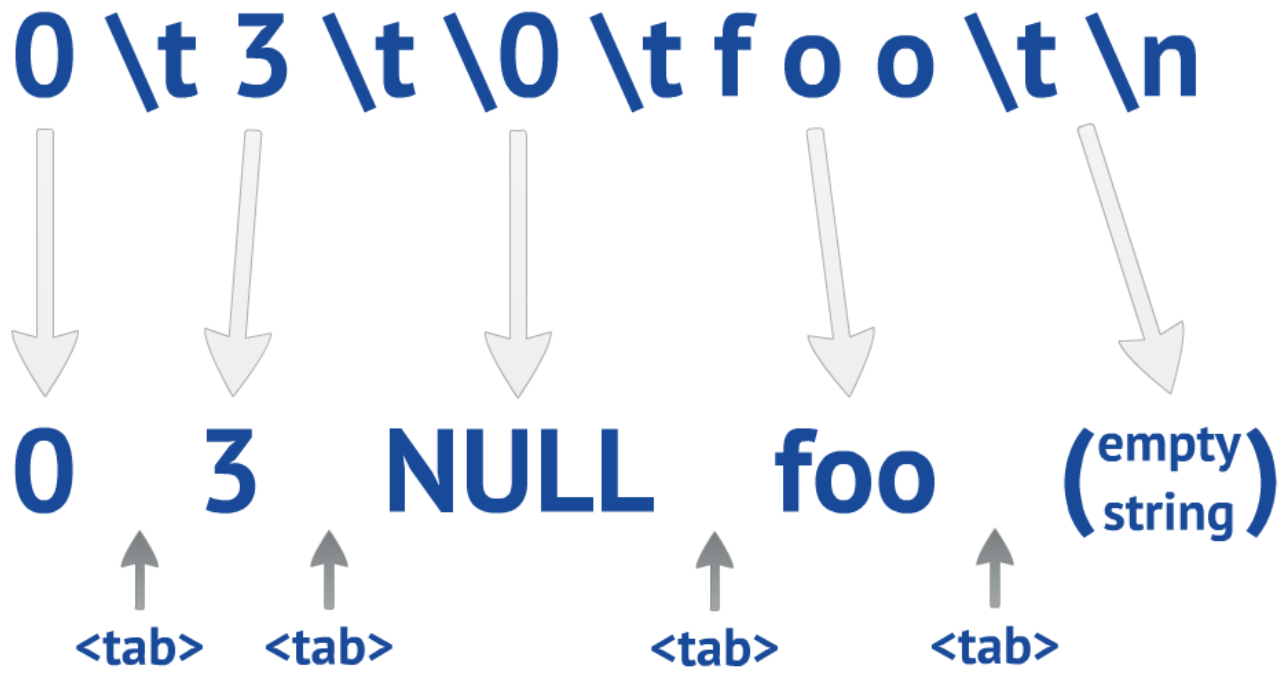
- text-like binary
- requests and responses are one line
- each line ends with \n (0x0A)
- each line consists of a set of tokens separated by \t (0x09)
- Token is either NULL or an encoded string
- NULL is encoded as \0 (0x00)

Strings:

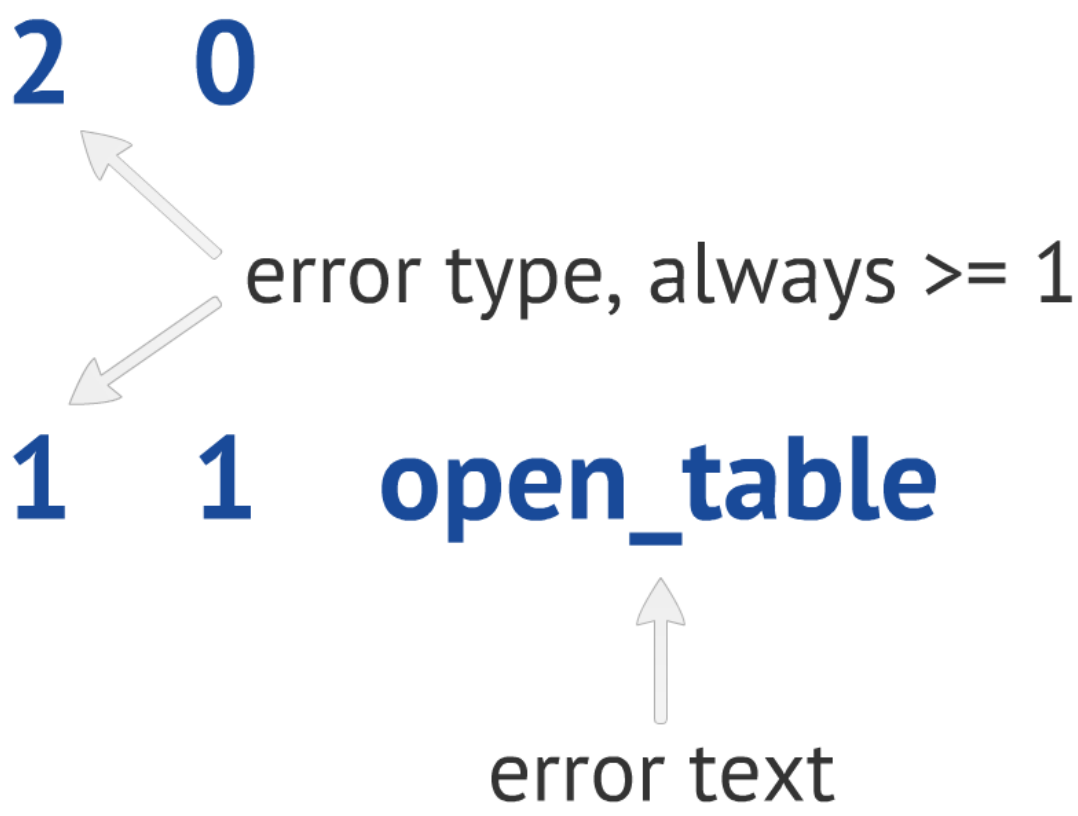
- Empty string is zero-length token
- Every byte in the range 0x00-0x0F is prefixed by 0x01 and shifted by 0x40. (eg. 0x03 -> 0x01 0x43)
- Other bytes are left unmodified

\t\t or \t\n means there's an empty string in between

Example command:



Error responses:



Commands

Opening index:

P <index_id> <db> <table> <index> <columns> [<fcolumns>]

- <index_id>: decimal number of your choice
- <db>,<table>,<index>: db, table, index names. To open the primary key use PRIMARY as <index>.
- <columns>: comma-separated list of column names you'll retrieve/insert/set
- <fcolumns>*: comma-separated list of column names you'll use for filtering

* — optional

Opening index:

P <index_id> <db> <table> <index> <columns> [<fcolumns>]

P 1 test store PRIMARY id,box fruit

something like prepared statement

SELECT id,box FROM test.store WHERE id=? AND fruit=?

Opening index:

P <index_id> <db> <table> <index> <columns> [<fcolumns>]

- You can re-open an index with the same <index_id> and possibly other <db>/<table>/<index>.
- You can open the same combination of <db>, <table>, <index> multiple times, possibly with different <columns>.
- You can't manually close indexes. Index is open until the client connection is closed.
- Open indices consume memory and make things work slower. Try to use less than 1000 indices.
- For efficiency, keep <index_id> small as far as possible.

Insert:

<index_id> + <vlen> <v1> ... <vn>

- <index_id>: opened index id
- <vlen>: count of <v1> ... <vn>. Must be <= to the count of <columns> in opened index.
- <v1> ... <vn>: field values in order of <columns>. For other columns in table the default values for each column are set.

 Always provide data for all columns you specified with <columns> param. (Will be discussed later)

Insert:

Example:

```
P  89  test  hs4  PRIMARY  warehouse,box,fruit,count
0  1

89  +  4  New York  A1  melon  4
0  1  1  ← last_insert_id

89  +  4  New York  A2  melon  4
0  1  2
      ↑
last_insert_id
```

Get:

broken!

<index_id> <op> <vlen> <v1> ... <vn> [LIM] [IN] [FILTER]

- <index_id>: opened index id
- <op>: operation — one of =, <, <=, >, >=
- <vlen>: count of <v1> ... <vn>. Must be <= to the count of columns forming <index> in opened index.
- <v1> ... <vn>: column values to look for in <index>.
- LIM*: OFFSET-LIMIT clause
- IN*: IN clause 
- FILTER*,**: FILTER clause

* — optional

** — can be used multiple times

Get:

Example:

Retrieve 1 row by exact id 3 (1-col index)

```
P  89  test  hs2  PRIMARY  warehouse,box,fruit,count
0  1

89   =   1   3
0   4   Virginia   A1   grapes   5
```

<columns> count

Get:

<index_id> <op> <vlen> <v1> ... <vn> [LIM] [IN] [FILTER]

LIMIT clause:

<limit> <offset>

Same logic as in SQL.
When omitted, it works as if 1 and 0 were specified.
Applied after FILTER.

Get:

Example:

Retrieve 3 rows starting from id 2 (1-col index)

```
P  89  test  hs2  PRIMARY  warehouse,box,fruit,count
0  1
                          LIM
89   >=   1   2   3   0
0   4   Seattle   B1   banana   4   Virginia   A1   grapes
5   Virginia   B2   watermelon   1
```

❗ note: column count is 4

Get:

<index_id> <op> <vlen> <v1> ... <vn> [LIM] [IN] [FILTER]

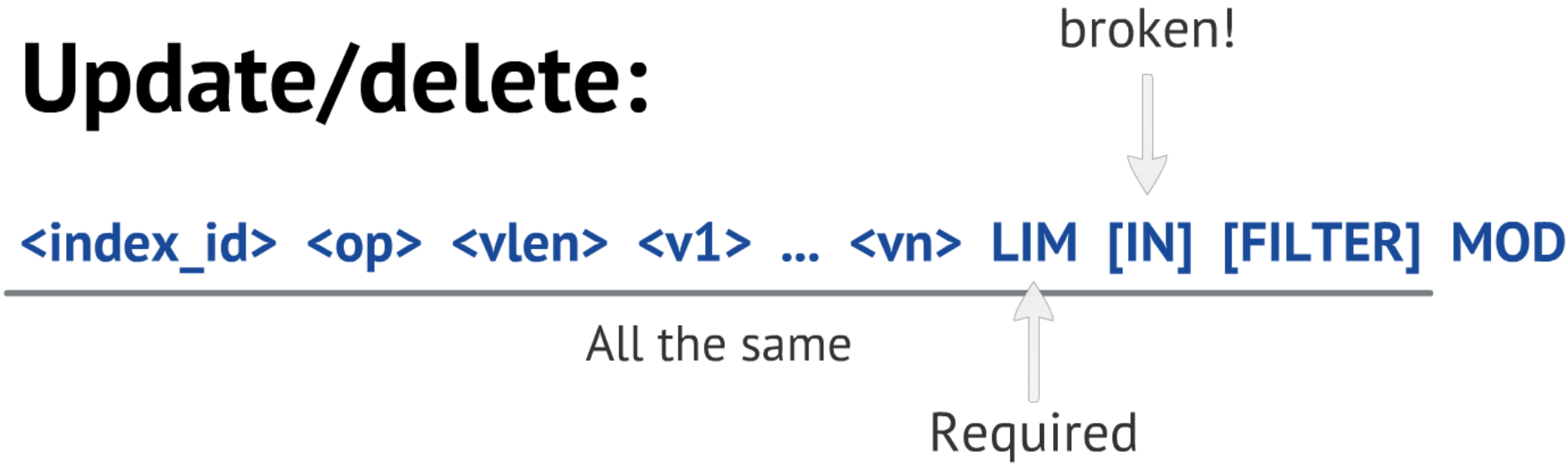
FILTER clause:

<ftyp> <fop> <fcol> <fval>

- <ftyp>: F (skip inappropriate rows) or W (end at first inappropriate row)
- <op>: one of =, !=, <, <=, >, >=
- <fcol>: zero-based number of the column in <fcolumns> in opened index
- <fval>: value

Multiple filters can be specified, and work as the logical AND of them.

Update/delete:



Update/delete:

<index_id> <op> <vlen> <v1> ... <vn> LIM [IN] [FILTER] MOD

MOD clause:

<mop> <m1> ... <mn>

- <mop>: U, U? (update), D, D? (delete), +, +? (increment), -, -? (decrement). Operations with '?' return row values before modifying them.

- <m1> ... <mn>: field values in order of <columns>. Can be <= number of columns in <columns>. Other columns are left intact then. Must be numeric for '+', '-' operations. Ignored for 'D', 'D?'.

Update/delete:

Example:

Get count where id=8 & set count=count+10 where id=8

P 90 test hsmdemo3 PRIMARY count
0 1

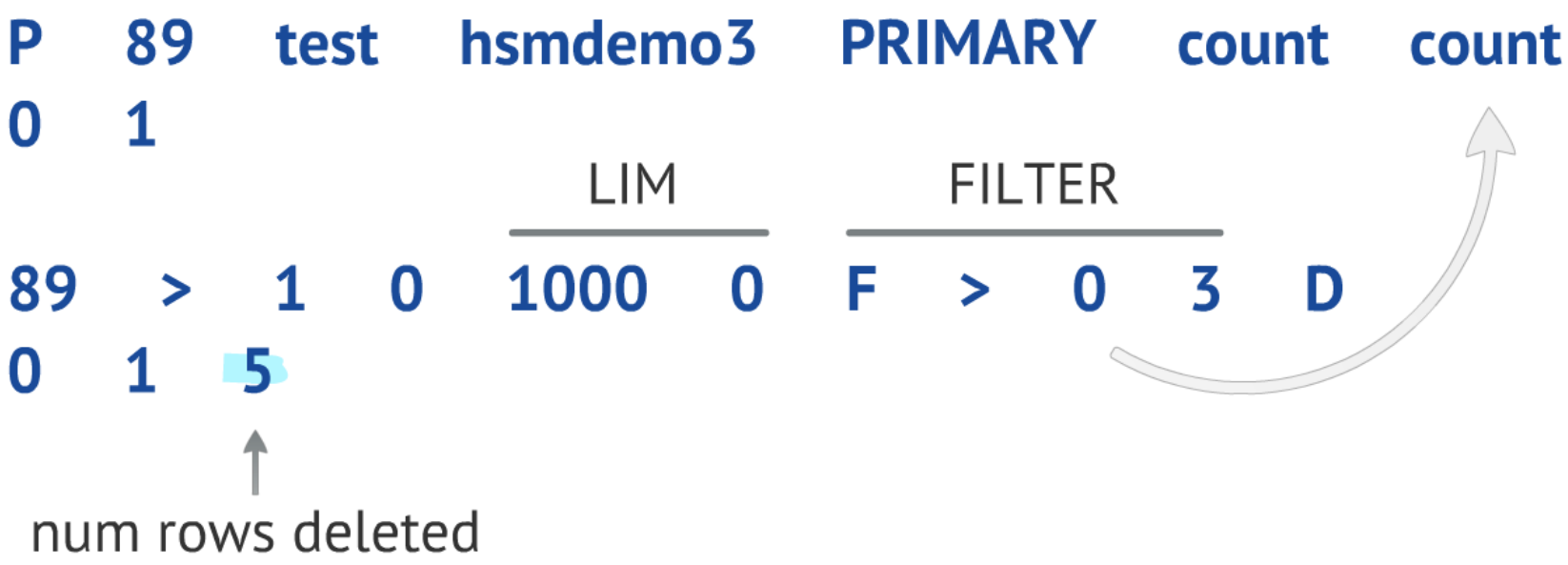
				LIM		MOD	
90	**=**	**1**	**8**	**1**	**0**	**+?**	**10**
0	**1**	**6**					

count

Update/delete:

Example:

Delete rows with id > 0 & count > 3



SQL — HS analogues

SELECT a,b,c FROM ...	Open index with <columns>=a,b,c
... LIMIT 1 OFFSET 0	LIMIT clause
id BETWEEN 1 AND 2	GET by index >= 1 with W-type FILTER while id <= 2
WHERE a < 1 AND b > 2	F-type FILTER a < 1, F-type FILTER b > 2
a IN (...)	IN clause
SELECT ... FOR UPDATE, UPDATE ...	update with '?' operations

That was hard, right?

‘Peculiarities’

Supported data types:

- You can read all data types via HS
- You can't write TIMESTAMP fields
- While writing, overflowing data is cut the same way as in SQL
- ‘ON UPDATE CURRENT_TIMESTAMP’ is not supported

Charsets:

- If you work with UTF8 only — don't worry at all. Just comply with HS protocol encoding standard.
- BLOBS are binary — you read what you write w/o any charset applied.
- HS reads & writes fields in the charset of the field. Have UTF8 & CP1251 in the same table? HS works with bytes as-is.
- Out-of-charset bytes are replaced with '?' ascii chars on insert/update.

Sorting (collations):

- Collations affect >, >=, <, <= operations (but not filters) and define the order you get results in
- HS reads records in the order of index
- In MySQL indices are ordered according to the collations of the fields they consist of
- Use http://www.collation-charts.org/mysql60/

Default values:

On insertion omitted fields get their default values.

P <index_id> <db> <table> <index> <columns> [<fcolumns>]

Only for fields not specified in <columns> in opened index!

- always provide all values for fields in <columns> for insert request

 NULL as a default value won't work. You'll get zeroed/empty values instead

 broken for BINARY, ENUM, TIMESTAMP data types

Use cases @ badoo

Use case 1:

Banned email lookup table

id	name	domain	mail_code	created	partner_id	ip
2	avoth	plol.com	8	2006-08-16 15:08:07	1	3232236478
3	anna	store.org	5	2006-08-16 16:30:46	19	3387996560
4	bubble	wiki.org	8	2006-08-16 16:39:30	1	3357320367
5	zlol	c12.net	8	2006-08-16 20:52:09	3	3386664249

We replaced SELECT * FROM ... WHERE name='...' AND domain='...' with HS get

One table, one server per DC. Master-master replicated to the same table in other DC.

~52 million rows
~5 Gb

All data fits into memory.
Persistent connections to HS.

Use case 1:

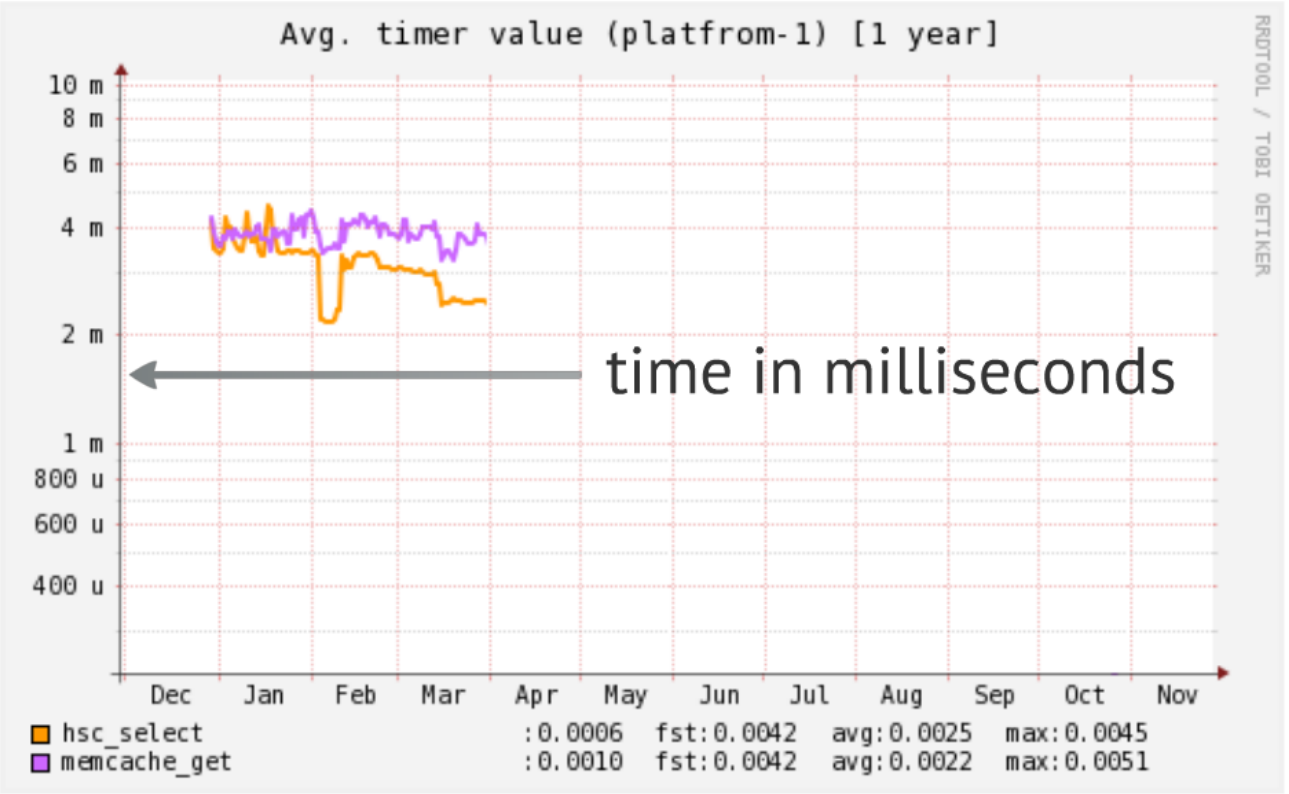
Banned email lookup table

Dual-core Intel(R) Xeon(R) CPU E5503 @ 2.00GHz

60% CPU used, LA = 0.5

Inserts/updates go via SQL, <10 RPS

Selects go via HS ~1000 RPS, 3 ms per read



Use case 2:

Persistent session store

id	ts	data
1:200:012b031ce0aa9f2357860bc884afabe1	1309728779	a:18:{s:7:"user_id";i:162130103;s:13:"located_state";i:0;s:19:"notify_whe
1:200:01c1cc3d66bebca1ada7cbb1bb88c2e2	1313544532	a:18:{s:7:"user_id";i:0;s:13:"located_state";i:0;s:19:"notify_when_located"
1:200:0243ba9fd935643022ad555ccb2e1fcf	1308866762	a:23:{s:7:"user_id";i:144063306;s:13:"located_state";i:0;s:19:"notify_whe
1:200:02b1163810a4b92aec0be5ab2714c41b	1310056296	a:23:{s:7:"user_id";i:245870921;s:13:"located_state";i:0;s:19:"notify_whe
1:200:0827e390a81452270fc1e63624d3fbe6	1315828681	a:18:{s:7:"user_id";i:0;s:13:"located_state";i:0;s:19:"notify_when_located"
1:200:08374703b88a61a14c5ef246f58ea8dc	1310239929	a:11:{s:7:"user_id";i:0;s:13:"located_state";i:0;s:19:"notify_when_located"
1:200:099c8449c6b3825b339260ffe57cca1e	1307918778	a:27:{s:7:"user_id";i:174196751;s:13:"located_state";i:1;s:19:"notify_whe
1:200:0d168e6aa60e4dbf04e4338c12bc8d81	1326767013	a:30:{s:7:"user_id";i:226661853;s:13:"located_state";i:1;s:19:"notify_whe
1:200:0d41dbc234b9d84210d6e3278877d285	1309462393	a:22:{s:7:"user_id";i:211055066;s:13:"located_state";i:0;s:19:"notify_whe
1:200:1a89c579ff3dea888309579fe9a82c01	1311704017	a:19:{s:7:"user_id";i:249410809;s:13:"located_state";i:0;s:19:"notify_whe

Key-value store: select/insert/update row by key via HS

Periodical purge (delete) via SQL

One table, 1 server, ~16 m rows, ~23 Gb

All data fits into memory.
Persistent connections to HS.

Use case 2:

Persistent session store

12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

8% CPU used, LA = 5

Create: <10 RPS, ~1.2ms/request

Update: ~180 RPS, ~1.3ms/request

Get: ~3500 RPS, ~0.5 ms/request

Originally was slower. Moving from MySQL/InnoDB to Percona Server/XtraDB gave us ~ 4x more performance.

Use case 3:

Sharded persistent session store

bucket	hash	ts	data
301	2b2afd841358c37d24f032675ee8e4be	1325717643	a:32:{s:7:"user_id";i:186309856;s:13:"located_state";i:1;s:19:"notify_when_l...
301	2ee318656f357b8a3aade908da0be37b	1329224832	a:33:{s:7:"user_id";i:240403199;s:13:"located_state";i:1;s:19:"notify_when_l...
301	4409e1663582b2e5563f10b394552361	1326269422	a:33:{s:7:"user_id";i:197481393;s:13:"located_state";i:1;s:19:"notify_when_l...
301	84e765c38a46093e2a06308c0234002e	1328089066	a:22:{s:7:"user_id";i:0;s:13:"located_state";i:0;s:19:"notify_when_located";i:1;...
301	8b8e982c4cf222f78cb9434f82c81b24	1347955687	a:32:{s:7:"user_id";i:211035528;s:13:"located_state";i:1;s:19:"notify_when_l...
301	9bf4917d960b4acbd1f30970ff6e2588	1354080964	a:28:{s:7:"user_id";i:164580752;s:13:"located_state";i:1;s:19:"notify_when_l...
302	948193e78f2cf2d10ce1c4c69c1dc4e7	1329422747	a:28:{s:7:"user_id";i:193050043;s:13:"located_state";i:0;s:19:"notify_when_l...
302	b229d7e37112563fbc3e277d9ddb42ac	1354108825	a:24:{s:7:"user_id";i:169647761;s:13:"located_state";i:0;s:19:"notify_when_l...
302	c0128b94c0e0fbaec0f4f223d134631f	1353396690	a:32:{s:7:"user_id";i:188038190;s:13:"located_state";i:1;s:19:"notify_when_l...
302	c40c38305f5a5eb98dac93b191bc558f	1336586100	a:32:{s:7:"user_id";i:214254546;s:13:"located_state";i:1;s:19:"notify_when_l...
302	db5e3916b9a98b4626afa4a2a32dd15f	1330430079	a:32:{s:7:"user_id";i:239925530;s:13:"located_state";i:1;s:19:"notify_when_l...
303	6b3cb06cb10f348017ac7e4a8354f2f6	1345401572	a:31:{s:7:"user_id";i:233159889;s:13:"located_state";i:0;s:19:"notify_when_l...

Now 10 000 tables in 100 databases, 1 MySQL, 1 server

Sharded by randomly generated hash

~10 m rows, ~20 Gb

All data fits into memory.
Persistent connections to HS.

Use case 3:

Sharded persistent session store

12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

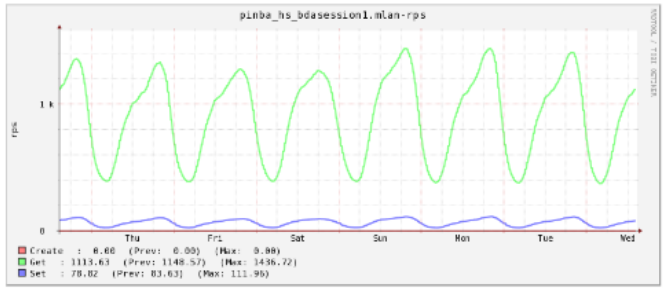
8% CPU used, LA = 5

Create: <10 RPS, ~1.3ms/request

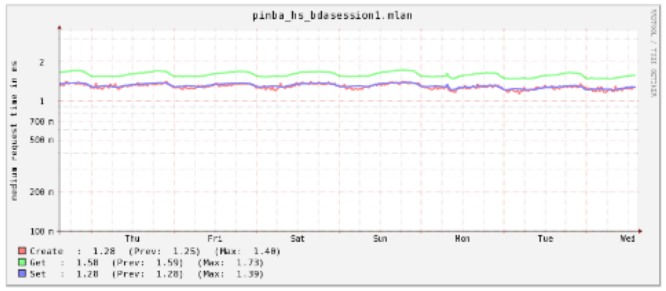
Update: ~150 RPS, ~1.3ms/request

Get: ~1200 RPS, ~1.6 ms/request

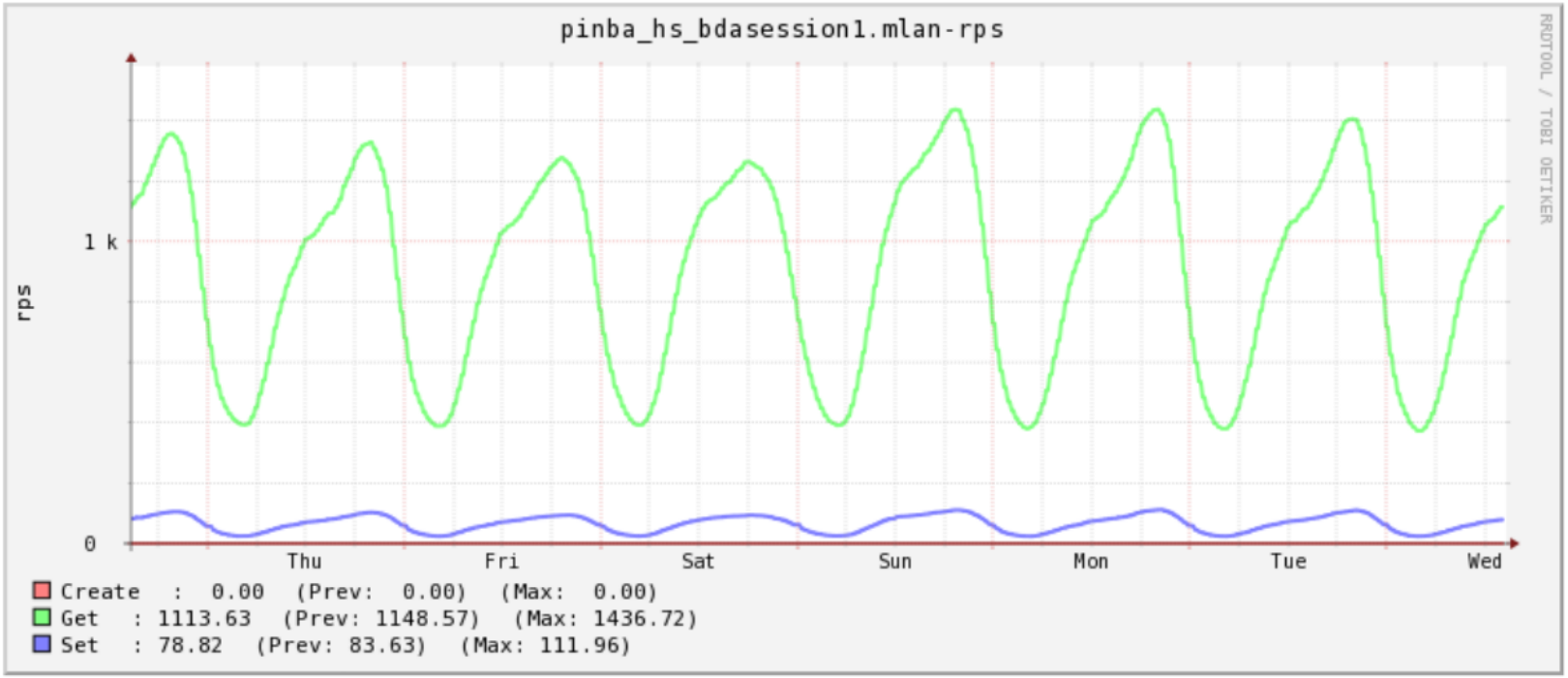
RPS



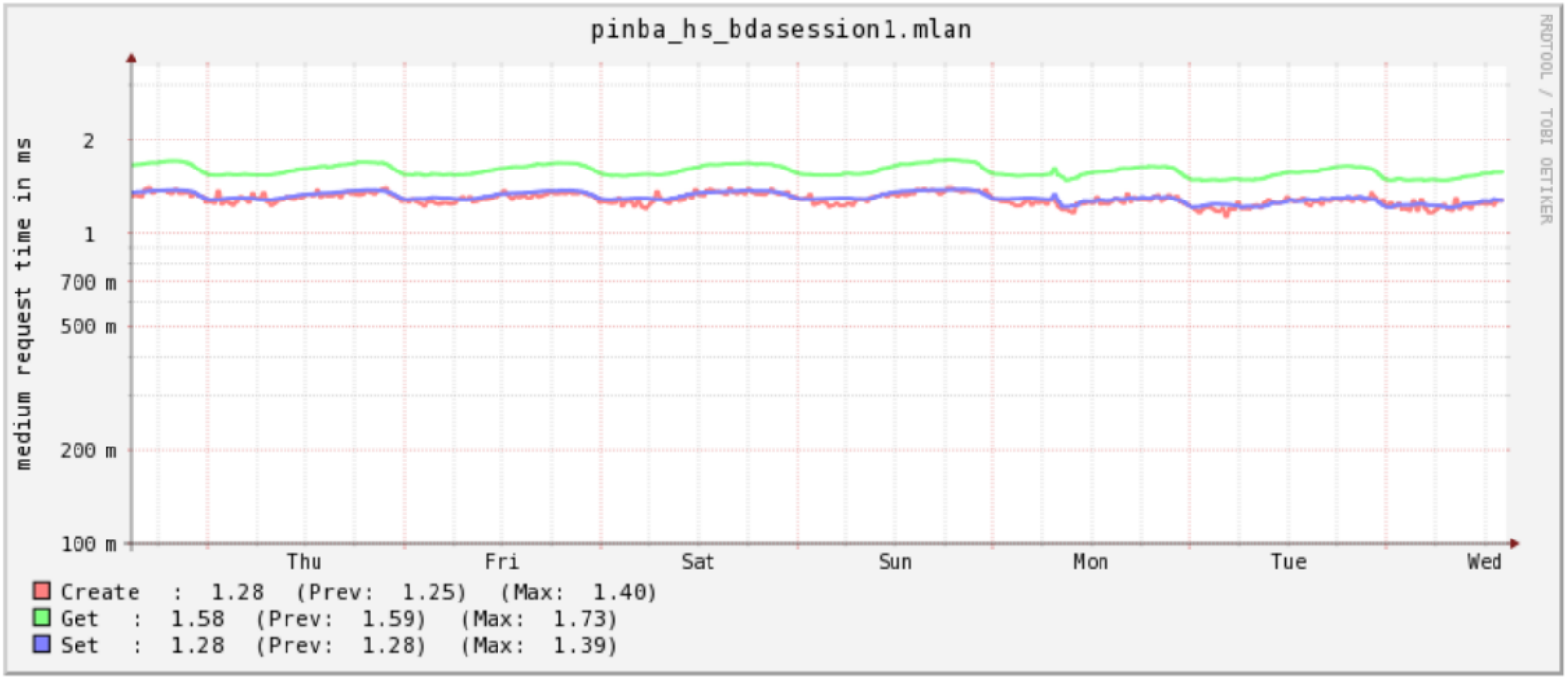
Avg req time in ms



RPS



Avg req time in ms



So what was the benefit of sharding?

Single-table setup worked well, but had low max write RPS. One table was single hot point.

Second problem: when a table grows, performance drops.

Clean obsolete rows daily.

Try sharding and compare with single-table setup for your application.

Use case 4:

Persistent cache

user_id	ts	data
212920	1329403894	a:2:{s:9:"interests";a:9:{i:93;i:1;i:1553;i:1;i:3575;i:1;i:126478;i:1;i:287957;i:1;i:314134;i:1;i:...
272920	1350963277	a:2:{s:9:"interests";a:11:{i:357;i:1;i:1454;i:1;i:1532;i:1;i:1612;i:1;i:8469;i:1;i:65846;i:1;i:100...
452920	1324394686	a:2:{s:9:"interests";a:3:{i:1715;i:1;i:2663;i:1;i:15002;i:1;}s:12:"count_active";i:3;}
472920	1341772506	a:2:{s:9:"interests";a:17:{i:122;i:1;i:328;i:1;i:331;i:1;i:357;i:1;i:420;i:1;i:495;i:1;i:3086;i:1;i:4...
2422920	1324992145	a:2:{s:9:"interests";a:5:{i:86;i:1;i:99;i:1;i:1000000499;i:1;i:1000003220;i:1;i:1000009839;i:1...
3572920	1342571425	a:2:{s:9:"interests";a:17:{i:154;i:1;i:436;i:1;i:506;i:1;i:1243;i:1;i:1530;i:1;i:1556;i:1;i:2824;i:1...
6372920	1327340629	a:2:{s:9:"interests";a:7:{i:404;i:1;i:1663;i:1;i:2831;i:1;i:8285;i:1;i:14532;i:1;i:16698;i:1;i:100...
7952920	1326721814	a:2:{s:9:"interests";a:1:{i:420;i:1;}s:12:"count_active";i:1;}
8182920	1326721934	a:2:{s:9:"interests";a:3:{i:14038;i:1;i:15077;i:1;i:16316;i:1;}s:12:"count_active";i:3;}

Originally we used memcached, cache reinitialization was looong...
We made our own persistent cache with MySQL/HS.

32 million rows, 14 Gb,
sharded across 10 000 tables, 1 server per DC

Key-value operations only: get/set/modify row by key.

All data fits into memory. Persistent connections to HS.

Use case 4:

Persistent cache

12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

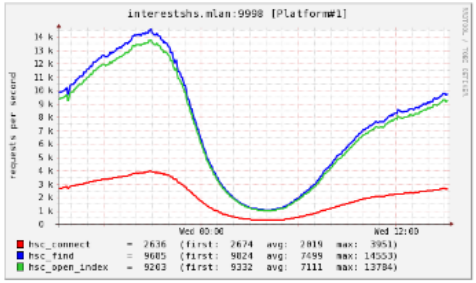
11% CPU used, LA=5

Create: <10 RPS, ~0.4ms/request

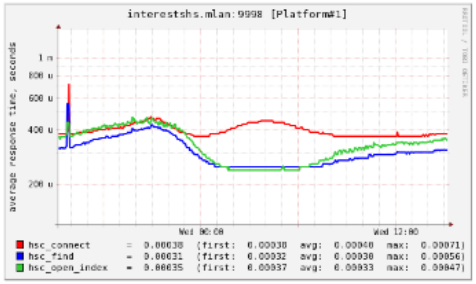
Update: <10 RPS, ~0.4ms/request

Get: ~14500 RPS at peak, ~0.5 ms/request

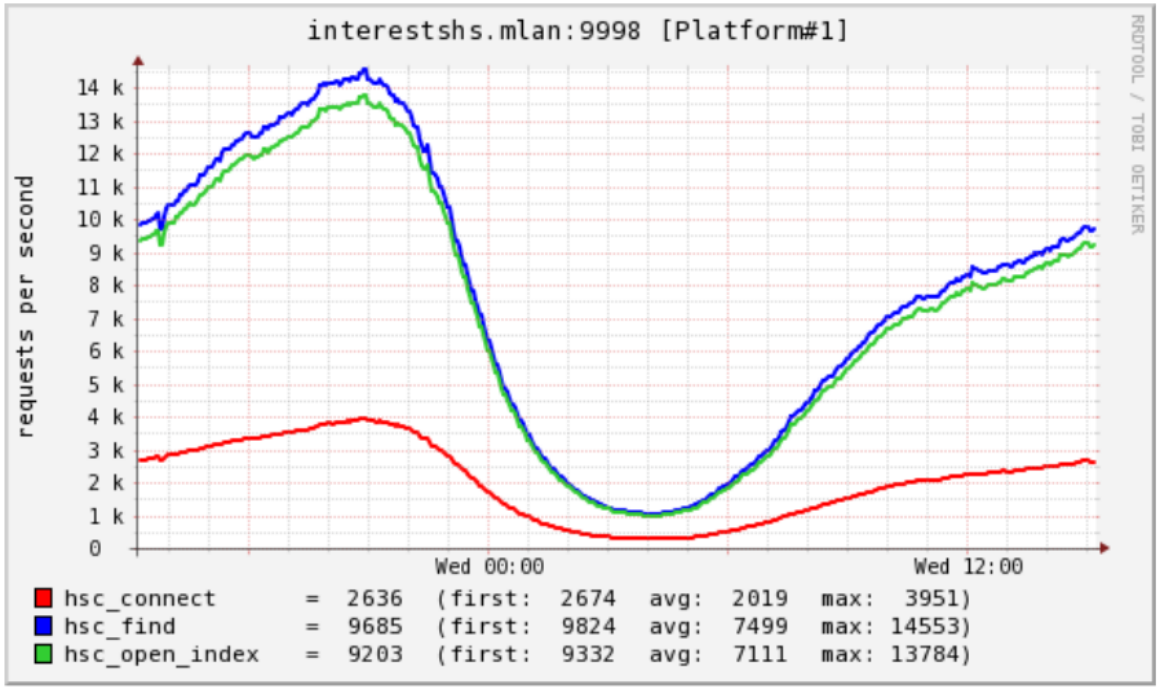
RPS



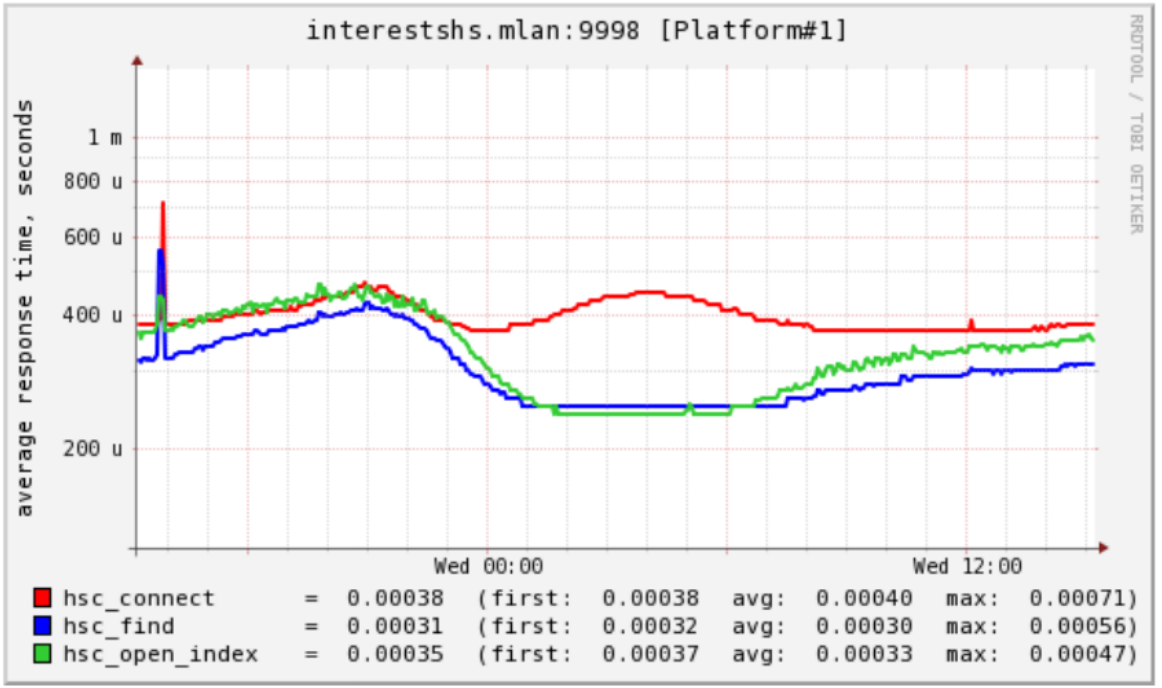
Avg req time in ms



RPS



Avg req time in ms



Tuning

Try to shard by key (for datasets > 10m rows)

Use Percona Server/XtraDB

Use persistent connections

There's one problem with persistent connections:

Next code iteration will inherit your open socket.

- In HS protocol requests and responses have no unique id, so you can't securely match requests and responses
- Get key, value where key = '...', check if result key matches requested key
- Reopen connection on syntax error, I/O error
- Prevent passing socket with your data to the next request

What else you can play with:

- InnoDB ROW_FORMAT
- InnoDB KEY_BLOCK_SIZE
- HASH indices
- Merging indices

Final part

FAQ

Should I use existing client library or should I write my own?

If you plan on using persistent connections, adapt existing library to solve the problems with pconnects I mentioned or consider writing your own library.

Why should I use this instead of ‘normal’ NoSQL DB like MongoDB or Redis?

1. You are still able to work with the same data via SQL — for those who can't abandon MySQL entirely.

2. If you're able to fit your needs in simple HS command set you'll get some core ‘old good SQL DB’ benefits — ACID, good multi-core scaling, efficient data storage.

Further reading

HS client libraries

https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/blob/master/README

HS source code

https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/

HS docs

https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/tree/master/docs-en

An article from authors

http://yoshinorimatsunobu.blogspot.ru/2010/10/using-mysql-as-nosql-story-for.html

Percona's HS page

http://www.percona.com/doc/percona-server/5.5/performance/handlersocket.html

Must-see presentation from HS author

http://www.slideshare.net/akirahiguchi/handlersocket-20100629en-5698215

Thanks!

PERCONA LIVE

badoo

Questions?

@ryba_xek

s@averin.ru

Slides, code, mindmap:
http://bit.ly/pluk2012

Prezi