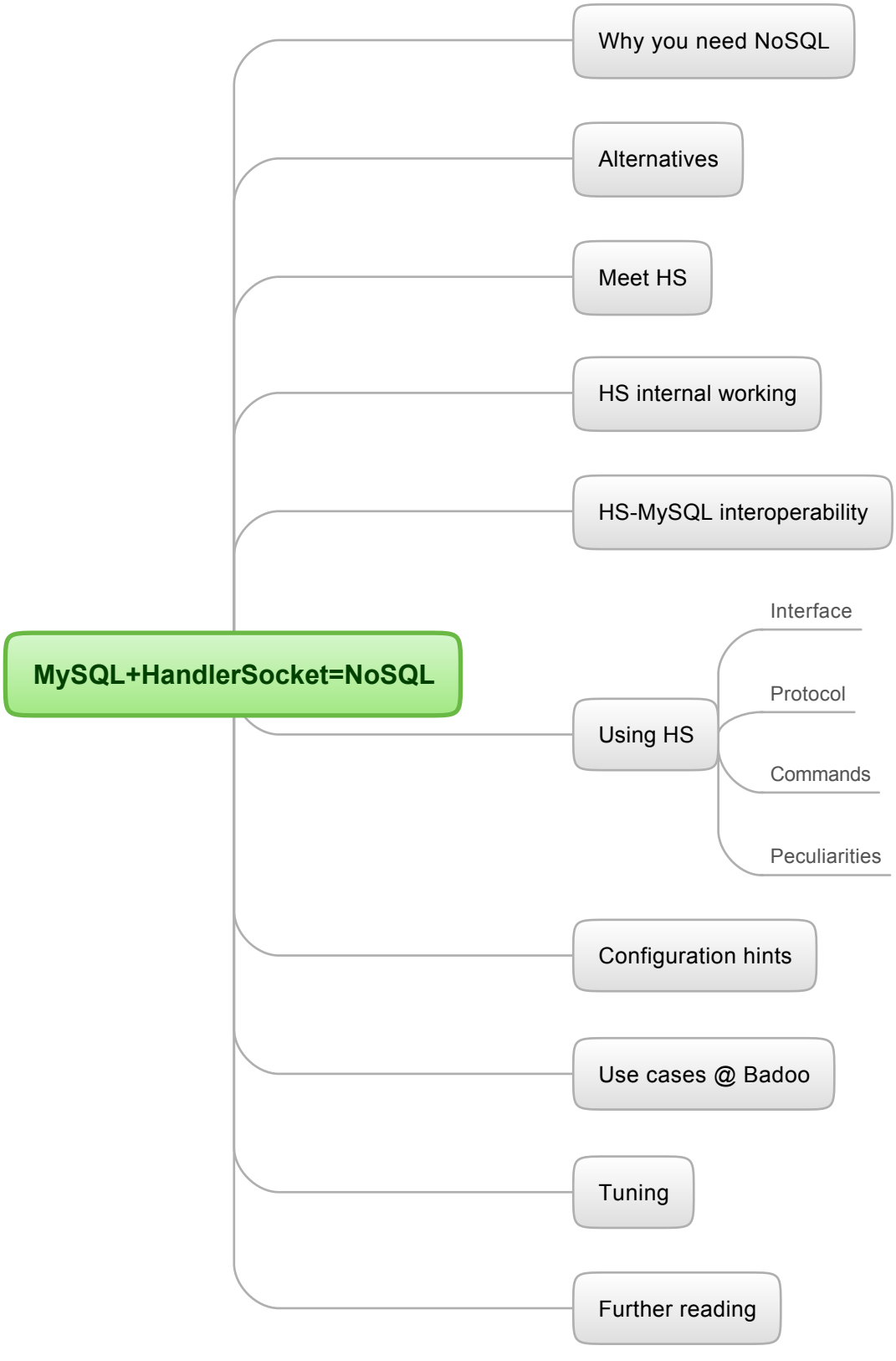
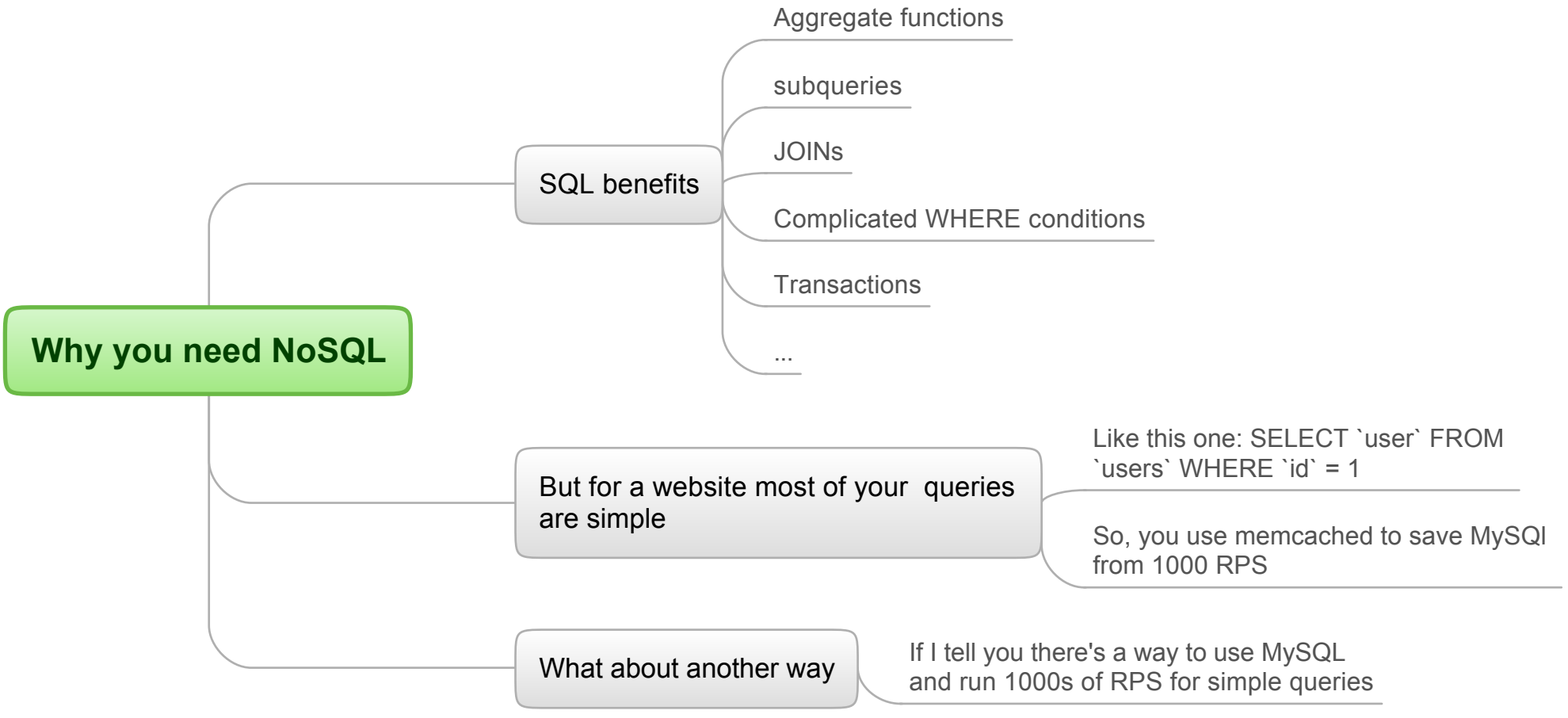

MySQL+HandlerSocket=NoSQL
Why you need NoSQL
Alternatives
Meet HS
HS internal working
HS-MySQL interoperability
Using HS
Interface
Protocol
Commands
Peculiarities
Configuration hints
Use cases @ Badoo
Tuning
Further reading


Why you need NoSQL
SQL benefits
Aggregate functions
subqueries
JOINs
Complicated WHERE conditions
Transactions
...
But for a website most of your queries are simple
Like this one: SELECT `user` FROM `users` WHERE `id` = 1
So, you use memcached to save MySQl from 1000 RPS
What about another way
If I tell you there's a way to use MySQL and run 1000s of RPS for simple queries

Alternatives

- MySQL HANDLER statement
  - Okay, but I've heard MySQL already provides NoSQL access
  - http://dev.mysql.com/doc/refman/5.0/en/handler.html
  - Yes, this is NoSQL-via-SQL query

```
handler `try1` open as h;
handler h read `PRIMARY` >= (1) where `value` LIKE 'v
%' AND `key` RLIKE 'k' LIMIT 3;
handler h read `PRIMARY` first where `value` LIKE 'v
%';
handler h CLOSE;
```

  - Cons
    - Read-only!
    - You get all fields
    - Not consistent
    - Not really fast
      - Faster in MariaDB

```
# SELECT * FROM family WHERE id = 1;
```

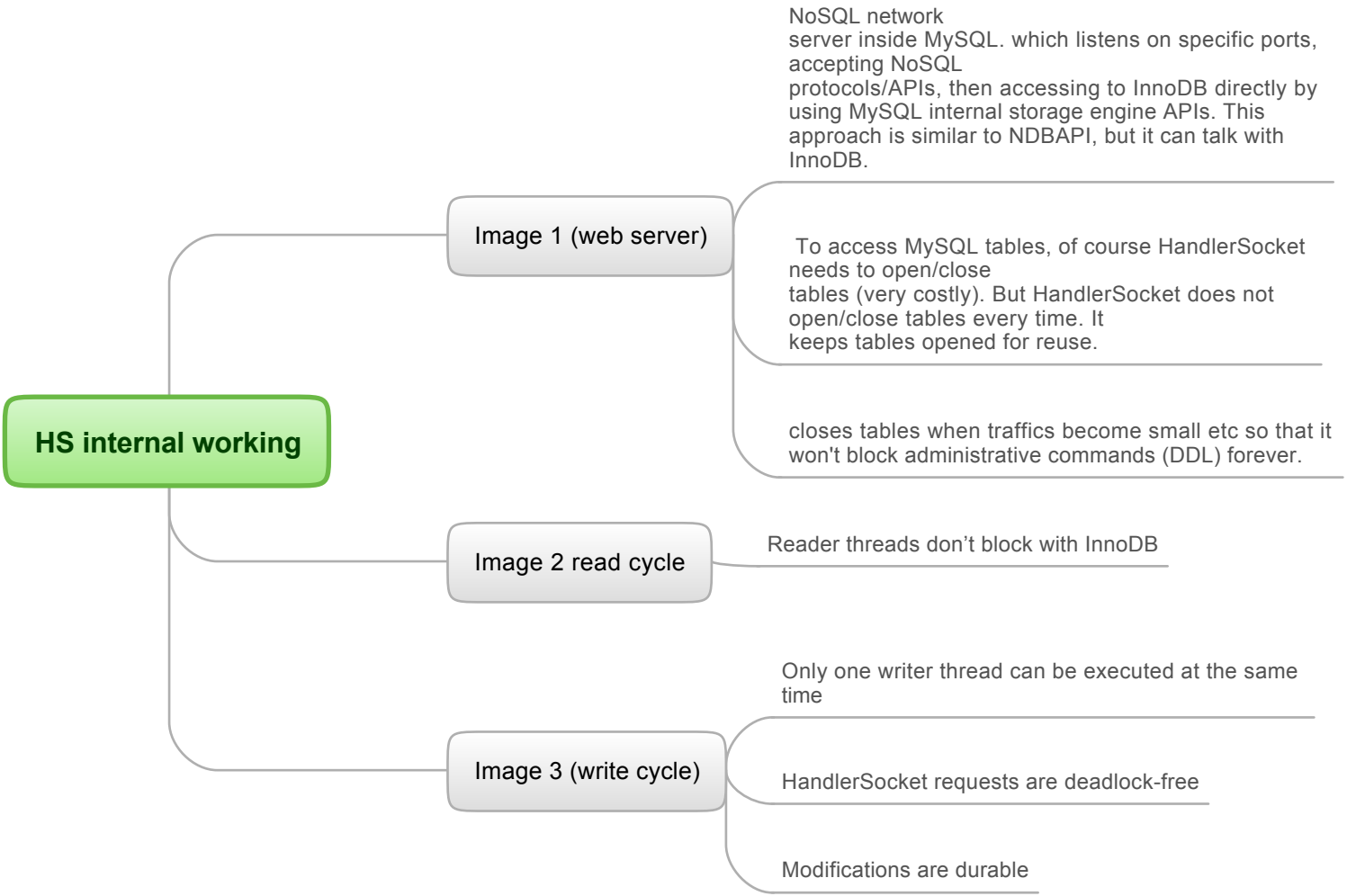
```
# MySQL
HANDLER family OPEN;
HANDLER family READ `PRIMARY` = (id)
WHERE id = 1;
HANDLER family CLOSE;
```

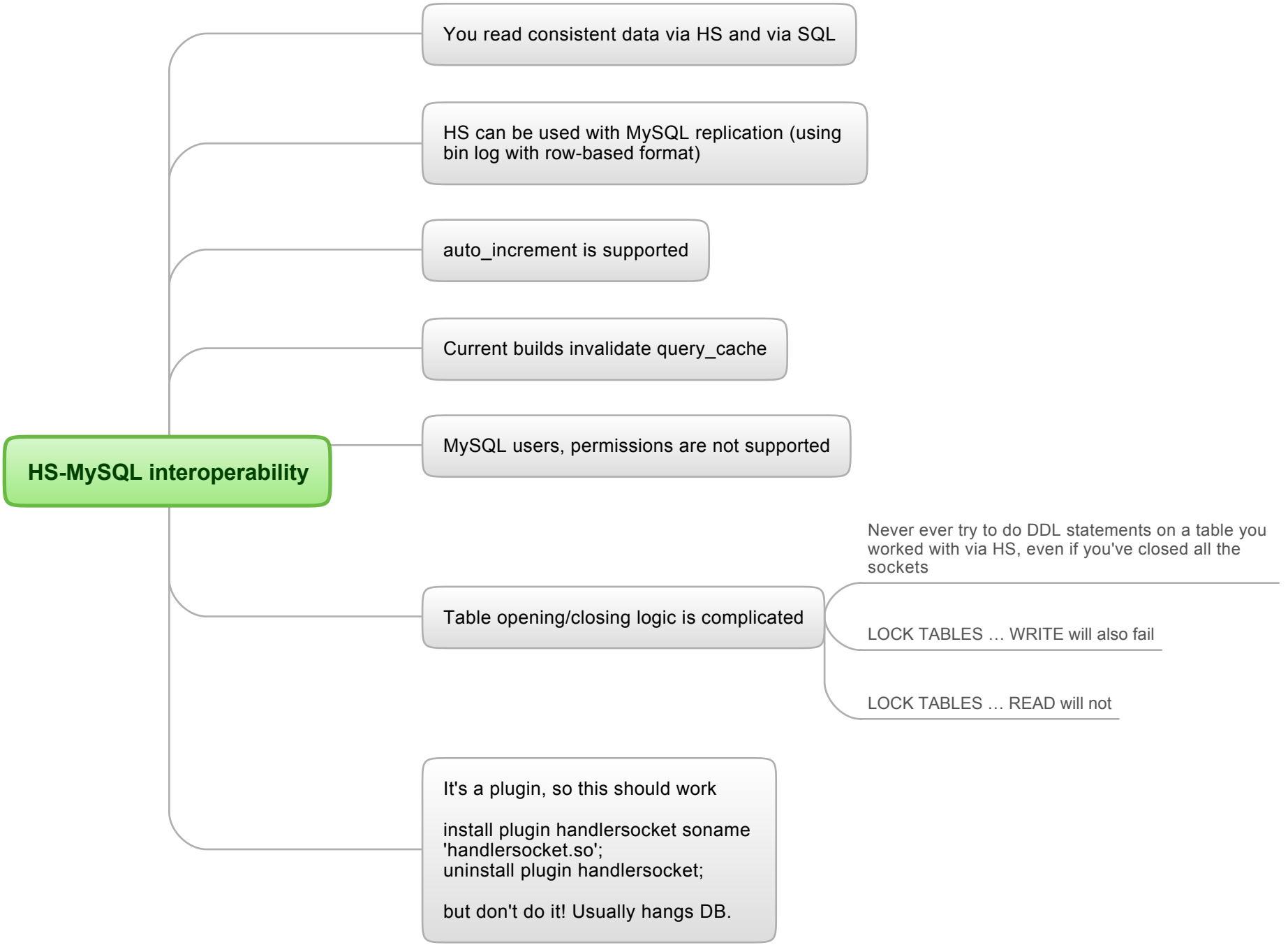
```
# With MariaDB 5.3
HANDLER family OPEN;
PREPARE stmt FROM 'HANDLER family
READ `PRIMARY` = (id) WHERE id = ?'; set
@id=1;
EXECUTE stmt USING @id;
DEALLOCATE PREPARE stmt;
HANDLER family CLOSE;
```

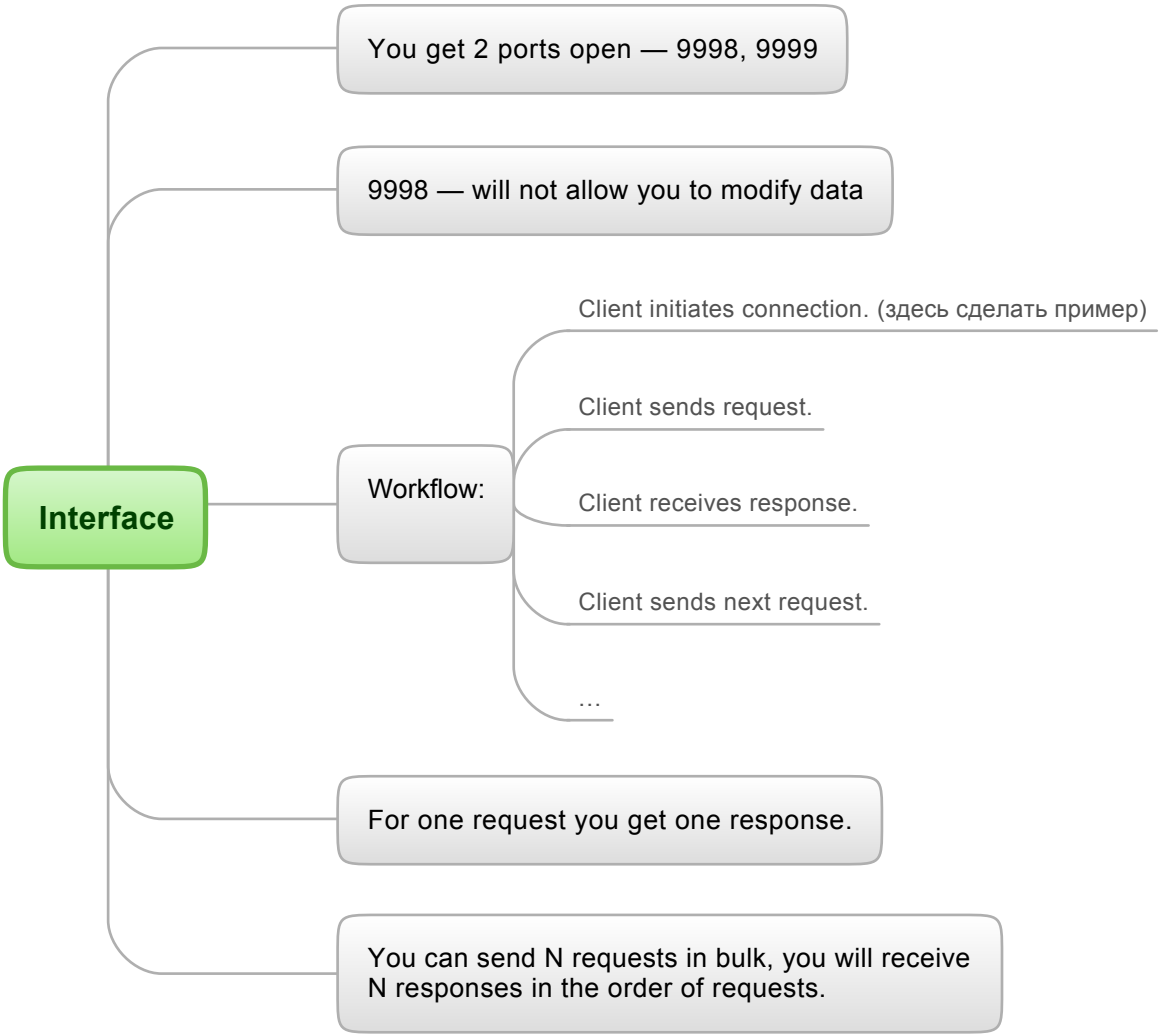
      - Even better with persistent connections!
  - Pros
    - where_condition can be full-featured MySQL expression, except subqueries or JOINS
    - No special client library needed
    - where_condition can be applied to any fields
    - You can fetch multiple rows per query
- Mysql innodb memcahed plugin
  - http://dev.mysql.com/doc/refman/5.6/en/innodb-memcached.html
  - Cons
    - early beta
    - a little bit complicated to deploy
    - works only on single row - not sure
    - preview is currently N/A for download
    - Returns one row
    - Very limited filtering abilities. No result sub-filtering, no >, <, <=, >=, != selectors
  - Pros
    - can retrieve/store multiple columns of your choice
    - You can use memcached itself
- NDBAPI
  - http://dev.mysql.com/doc/ndbapi/en/overview-ndb-api.html
  - Cons
    - Separate MySQL product with lots of peculiarities, complicated
    - NDBstorage engine only
    - Complicated API with a limited subset of languages supported officially

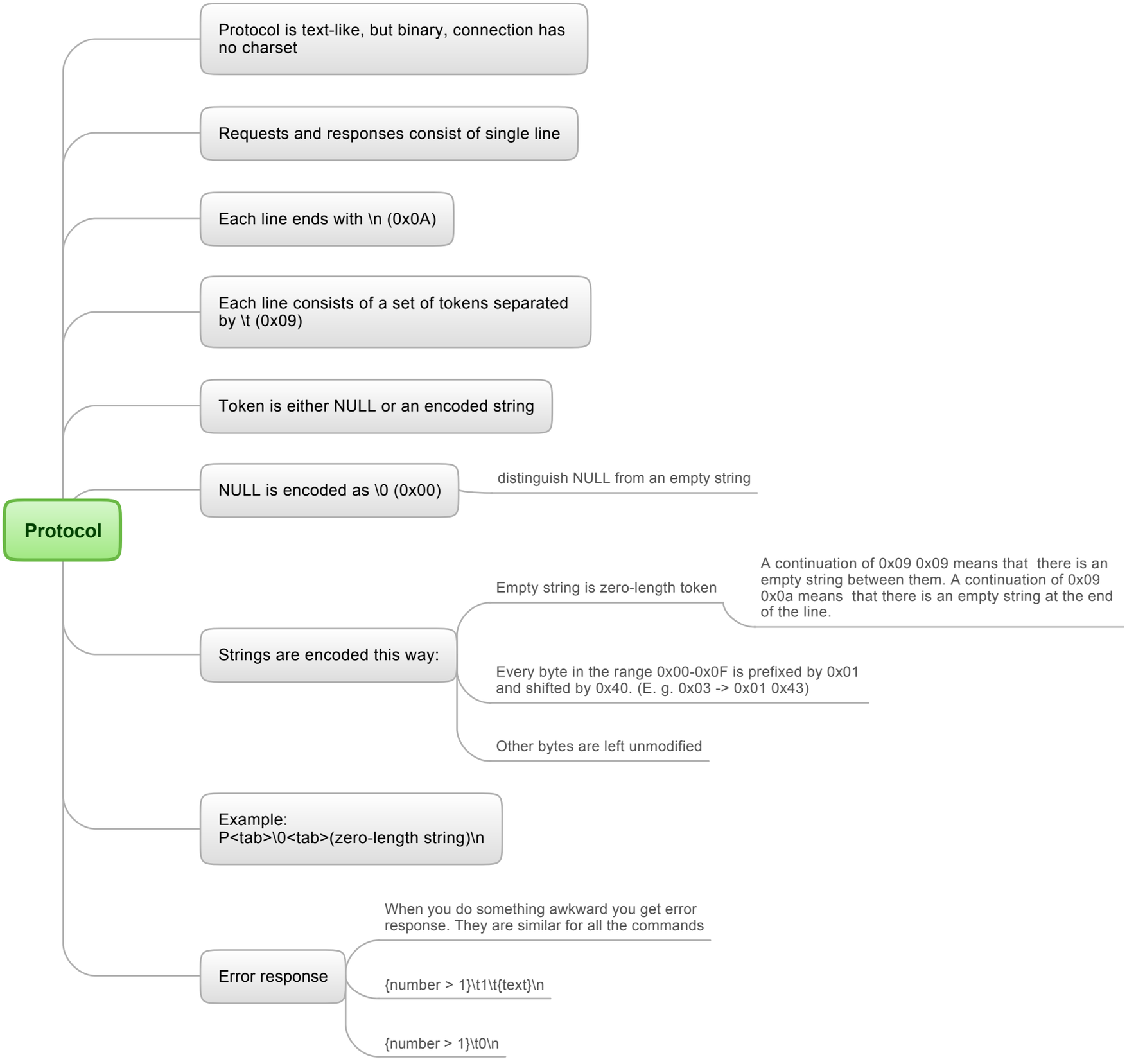
Meet HS

- What HS is
  - MySQL plug-in with direct access to InnoDB/XtraDB
  - It works as a daemon inside the mysqld process, accepting TCP connections, and executing requests from clients.
  - HandlerSocket does not support SQL queries. Instead, it has its own simple protocol for CRUD operations on tables.
- Handlersocket history
  - Was originally intended for fast PK lookups
  - Community contribution by DeNa corp, Japan. Written by Akira Higuchi
    - http://www.dena.jp/en/index.html
  - Later was loved by the community, gained extra functionality
- Pros
  - To lower CPU usage it does not parse SQL.
  - It batch-processes requests where possible, which further reduces CPU usage and lowers disk usage.
  - The client/server protocol is very compact compared to mysql/libmysql, which reduces network usage.
  - Also simple protocol means easy debugging, you can try it yourself with telnet
  - Withstands 10000+ simultaneous connections
  - Allows you to work with the same table via SQL simultaneously
  - Can be used with MySQL replication
  - Lots of client libraries on the net
  - Bundled with Percona Server
  - No second cache -> no data inconsistency
  - No duplicate cache (throw out memcached)
- Cons
  - Originally InnoDB only, but works with some other storage engines
  - No transactions/stored procedures
  - Some data types are not fully supported
  - Charsets/collations/timezones functionality is very limited
  - Poor support
    - Not supported by Oracle.
    - unpredictable time of bugfixes (22 open bug reports for example). My favorite ‘off by one error in IN() requests’ is still not fixed.
  - Unmature
    - Buggy in lesser used functionality
    - loose documentation — learn by trial and error (this talk will cover 99% you will need)
    - Update/insert commands behavior was once changed dramatically, drawing broken inserts/updates
    - you can't even get handlersocket plugin's version through handler socket interface
  - Sometimes holds tables open in some cases, preventing table structure modifications
  - Protocol was not designed for persistent connections, need some tricks
  - You will possibly want to write your own client library with persistent connections after this talk
- What HS is not
  - Not a key-value/document store
    - This is rather an interface for reading/writing relational data
  - Not a ‘binary SQL’
    - operations you can make are limited
  - Not a tool for complex queries
    - No subqueries, no JOINs, no aggregate functions
  - Not an interface for HANDLER MySQL statement
  - Not a tool for creating/modifying tables
  - Not for hosting/shared usage
    - limited authentication, MySQL users are not supported
  - Not intended for working with datasets which don't fit in RAM
    - You'll work with hard disk then, HS will make no sense

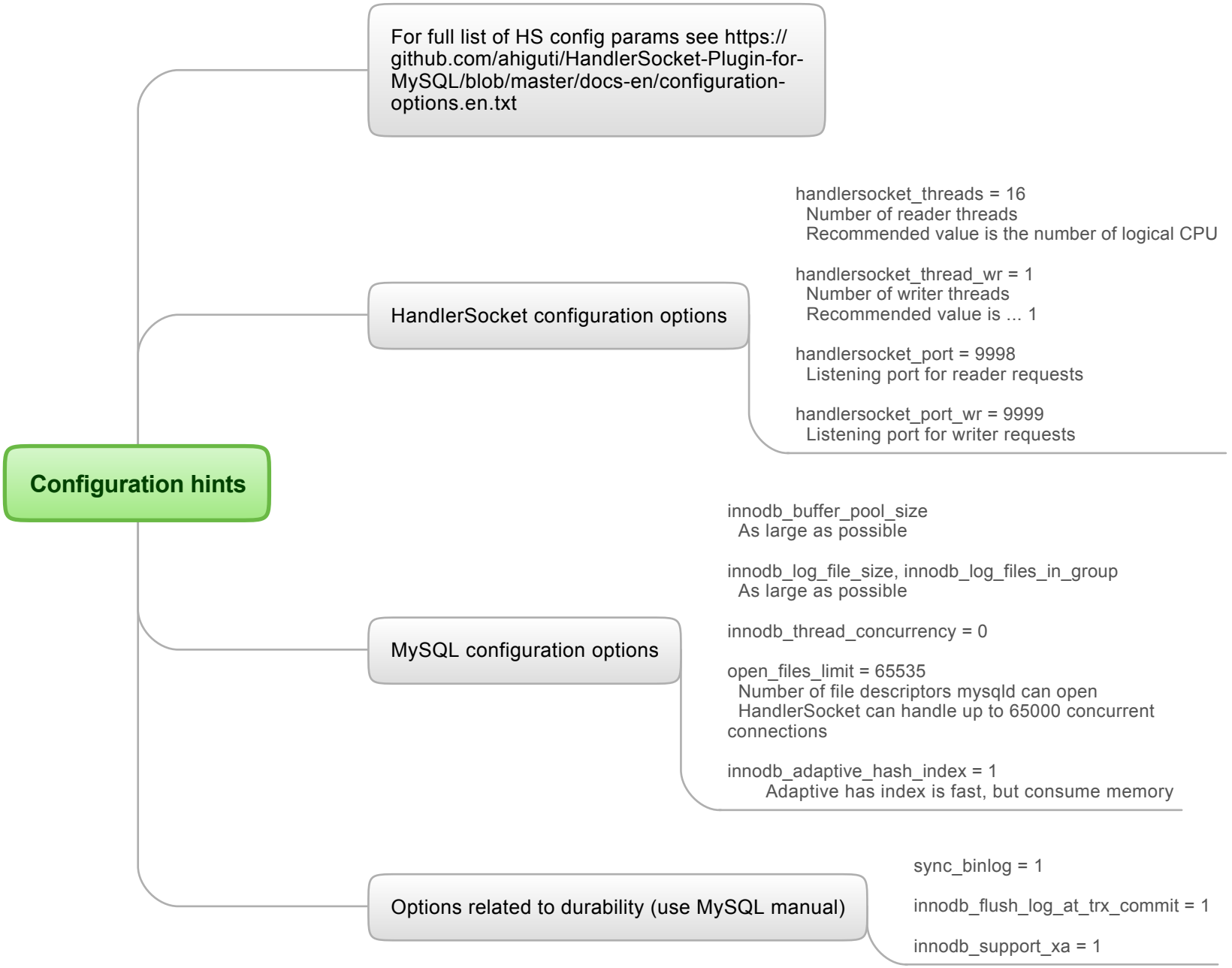

HS internal working
Image 1 (web server)
NoSQL network
server inside MySQL. which listens on specific ports, accepting NoSQL
protocols/APIs, then accessing to InnoDB directly by using MySQL internal storage engine APIs. This approach is similar to NDBAPI, but it can talk with InnoDB.
To access MySQL tables, of course HandlerSocket needs to open/close
tables (very costly). But HandlerSocket does not open/close tables every time. It
keeps tables opened for reuse.
closes tables when traffics become small etc so that it won't block administrative commands (DDL) forever.
Image 2 read cycle
Reader threads don’t block with InnoDB
Image 3 (write cycle)
Only one writer thread can be executed at the same time
HandlerSocket requests are deadlock-free
Modifications are durable

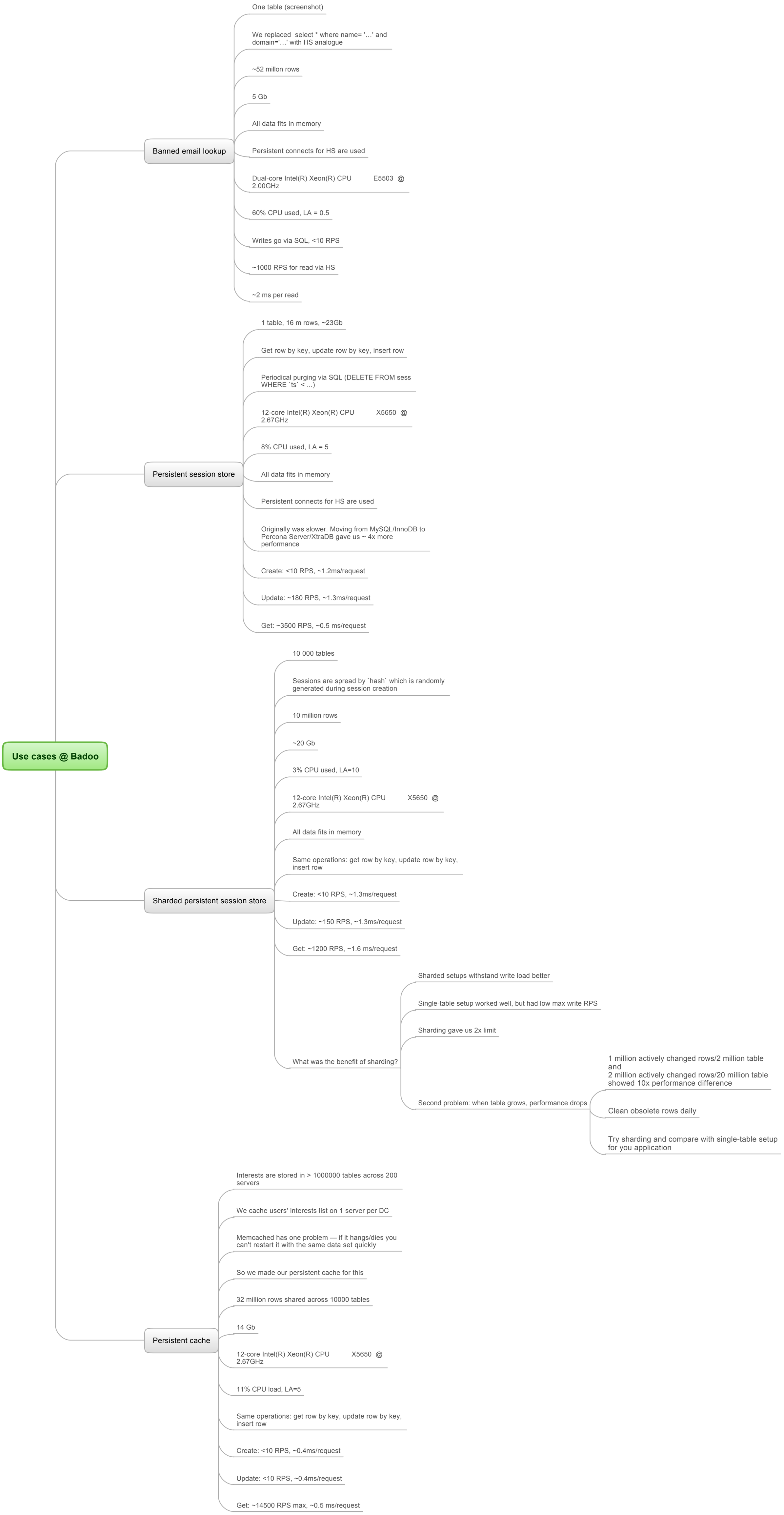

HS-MySQL interoperability
You read consistent data via HS and via SQL
HS can be used with MySQL replication (using bin log with row-based format)
auto_increment is supported
Current builds invalidate query_cache
MySQL users, permissions are not supported
Table opening/closing logic is complicated
Never ever try to do DDL statements on a table you worked with via HS, even if you've closed all the sockets
LOCK TABLES … WRITE will also fail
LOCK TABLES … READ will not
It's a plugin, so this should work
install plugin handlersocket soname 'handlersocket.so';
uninstall plugin handlersocket;
but don't do it! Usually hangs DB.


Interface
You get 2 ports open — 9998, 9999
9998 — will not allow you to modify data
Workflow:
Client initiates connection. (здесь сделать пример)
Client sends request.
Client receives response.
Client sends next request.
…
For one request you get one response.
You can send N requests in bulk, you will receive N responses in the order of requests.


Protocol
Protocol is text-like, but binary, connection has no charset
Requests and responses consist of single 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)
distinguish NULL from an empty string
Strings are encoded this way:
Empty string is zero-length token
A continuation of 0x09 0x09 means that there is an empty string between them. A continuation of 0x09 0x0a means that there is an empty string at the end of the line.
Every byte in the range 0x00-0x0F is prefixed by 0x01 and shifted by 0x40. (E. g. 0x03 -> 0x01 0x43)
Other bytes are left unmodified
Example:
P<tab>\0<tab>(zero-length string)\n
Error response
When you do something awkward you get error response. They are similar for all the commands
{number > 1}\t1\t{text}\n
{number > 1}\t0\n

- **Commands**
  - Authentication
    - Command description — what can be done with it. You don't need this if you have no password
    - Syntax
    - Params
    - Ok response
    - Fail response
    - Simple example
  - Open index
    - Command description — what can be done with it
      - To work with a table you need to get a special descriptor
    - Syntax
    - Params
    - Notes
      - You can re-open an index with the same {index id} and possibly other {db name}/{table name}/{index name}.
      - You can open the same combination of {db name}, {table name}, {index name} multiple times, possibly with different {columns list}.
      - 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.
    - Ok response
    - Simple examples
      - one
      - two
    - Errors: locked table
      - P 89 test hscwdemo2 PRIMARY
      - pk2 1 open_table
  - Create
    - Command description — what can be done with it.
    - Syntax
    - Params
      - You should provide values for the columns you opened with open_index.
      - Bug: Always provide data for all columns you specified with {columns list} param. (Will be discussed later)
    - Ok response
    - Ok response for auto_increment:
    - Error response
    - Example
  - Retrieve
    - Command description — what can be done with it.
    - Syntax
    - Params
    - LIMIT clause:
    - IN clause:
    - FILTER clause:
    - Empty result
    - Non-empty result is
    - Error response
    - Examples
      - 1-column index
        - Retrieve 1 row by exact id
        - Retrieve 3 rows starting from id 2
      - 2-column index
        - Retrieve all rows with warehouse = Virginia (1st column in index used)
        - Retrieve row with warehouse = California & box = A1 (1st & 2nd column in index used)
        - Retrieve all rows after warehouse = Seattle & box = A1 if sorted by warehouse, box:
      - IN
        - Get all rows with id IN (2, 4)
        - Retrieve all rows with warehouse = Virginia & box IN (A1, B2):
      - Filters
        - Get all rows with id > 2 & box != A1 & count < 6
        - Get all rows with id >=0 going down the index until count > 1
        - Filter order has no effect.
  - Update/Delete
    - Command description — what can be done with it.
    - Syntax
    - Small bug: Won't work until you specify limit.
    - Params
    - MOD clause
    - Ok response
    - Error response
    - Examples
      - Update
        - set count = 5 where warehouse = Seattle
        - Get count where id=8 and set count=count+10 where id=8
      - Delete
        - Delete rows with id > 1 & count > 3


Configuration hints
For full list of HS config params see https://github.com/ahiguti/HandlerSocket-Plugin-for-MySQL/blob/master/docs-en/configuration-options.en.txt
HandlerSocket configuration options
handlersocket_threads = 16
Number of reader threads
Recommended value is the number of logical CPU
handlersocket_thread_wr = 1
Number of writer threads
Recommended value is ... 1
handlersocket_port = 9998
Listening port for reader requests
handlersocket_port_wr = 9999
Listening port for writer requests
MySQL configuration options
innodb_buffer_pool_size
As large as possible
innodb_log_file_size, innodb_log_files_in_group
As large as possible
innodb_thread_concurrency = 0
open_files_limit = 65535
Number of file descriptors mysqld can open
HandlerSocket can handle up to 65000 concurrent connections
innodb_adaptive_hash_index = 1
Adaptive has index is fast, but consume memory
Options related to durability (use MySQL manual)
sync_binlog = 1
innodb_flush_log_at_trx_commit = 1
innodb_support_xa = 1


Use cases @ Badoo
Banned email lookup
One table (screenshot)
We replaced select * where name= '…' and domain='…' with HS analogue
~52 millon rows
5 Gb
All data fits in memory
Persistent connects for HS are used
Dual-core Intel(R) Xeon(R) CPU E5503 @ 2.00GHz
60% CPU used, LA = 0.5
Writes go via SQL, <10 RPS
~1000 RPS for read via HS
~2 ms per read
Persistent session store
1 table, 16 m rows, ~23Gb
Get row by key, update row by key, insert row
Periodical purging via SQL (DELETE FROM sess WHERE `ts` < ...)
12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
8% CPU used, LA = 5
All data fits in memory
Persistent connects for HS are used
Originally was slower. Moving from MySQL/InnoDB to Percona Server/XtraDB gave us ~ 4x more performance
Create: <10 RPS, ~1.2ms/request
Update: ~180 RPS, ~1.3ms/request
Get: ~3500 RPS, ~0.5 ms/request
Sharded persistent session store
10 000 tables
Sessions are spread by `hash` which is randomly generated during session creation
10 million rows
~20 Gb
3% CPU used, LA=10
12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
All data fits in memory
Same operations: get row by key, update row by key, insert row
Create: <10 RPS, ~1.3ms/request
Update: ~150 RPS, ~1.3ms/request
Get: ~1200 RPS, ~1.6 ms/request
What was the benefit of sharding?
Sharded setups withstand write load better
Single-table setup worked well, but had low max write RPS
Sharding gave us 2x limit
Second problem: when table grows, performance drops
1 million actively changed rows/2 million table and
2 million actively changed rows/20 million table showed 10x performance difference
Clean obsolete rows daily
Try sharding and compare with single-table setup for you application
Persistent cache
Interests are stored in > 1000000 tables across 200 servers
We cache users' interests list on 1 server per DC
Memcached has one problem — if it hangs/dies you can't restart it with the same data set quickly
So we made our persistent cache for this
32 million rows shared across 10000 tables
14 Gb
12-core Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
11% CPU load, LA=5
Same operations: get row by key, update row by key, insert row
Create: <10 RPS, ~0.4ms/request
Update: <10 RPS, ~0.4ms/request
Get: ~14500 RPS max, ~0.5 ms/request

Tuning

- Try to shard by key with datasets > 10m rows, usually it helps
- User Percona Server/XtraDB
- Use persistent connections
  - There is one problem
    - Requests and responses have no unique id, so you can't securely match requests and responses
    - When working with web requests you can unintentionally leave some unread data in the socket
    - and your process will start serving next request
    - and this next request will inherit this open socket with some data
    - it will send request and get this data and treat is as a response
    - so it will get an answer from previous query and this will be 100% syntactically correct
    - There is no guaranteed way to avoid this when using persistent connections
    - but we're using a simple approach that allows us to sleep with no worry
    - instead of doing
      retrieve value where key is 'abc'

      we do
      retrieve key, value where key is 'abc'

      this can be easily done with HS
    - then we check we got that same key we were asking for
    - This prevents us from reading other rows instead of the row we need
  - Recommendations
    - To lower the probability of problems
    - Alwayse reopen connection on syntax error
    - on socket read/write timeout
    - Always try to read all the data from socket
  - Lazy open_index technique
    - To work with any table you need to open index
    - with normal connections you need to do it every time you open a new connection
    - in the world of websites this is usually connect, open_index, do 2—3 other requests, close connection
    - with persistent connections you can open index very rarely
    - If you don't need to open different indexes under the same index number
    - So, when creating a socket in your program (e. g. pfsockopen() in PHP)
    - check if index is already open and open it if it's not
    - I do it this way:
    - I execute a query
    - if I get 2 1 stmtnum error from HS
    - I open the corresponding index then
    - Remember: open index consumes memory. You won't be able to hold 1000 connections with 1000 open indexes at the same time
- What else you can play with
  - InnoDB ROW_FORMAT
  - InnoDB KEY_BLOCK_SIZE
  - HASH indexes
  - Merging indexes
    - If you need to work with several indexes in HS table, try to merge them in one big multi-column index


Further reading
HS client libraries
https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/blob/master/README
HS sources
https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/
HS docs
https://github.com/DeNADev/HandlerSocket-Plugin-for-MySQL/tree/master/docs-en
HS introduction 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