# SWC DB: Abstract

author: Kashirin Alex
kashirin.alex@gmail.com

# SWC DB (Super Wide Column Database)

The database is structured in columns , not by Tables or Namespaces, as familiar SQL is '`select columns from “table_name”;`', with “Super Wide Column DB” It is '`select [where_clause [Columns-Intervals] ];` '.

The “Super Wide Column” comes to its meaning by the column's key is a list-set of keys eg. A column cell's key is keys=(k1,k2,k3,k4,kN), comparing to a “Wide Column” key that is row, column-family, column-family-qualifier in the “Super Wide Column DB” it is equal to keys=(k1(row),k2(cf),k3(cq)) or(similar) column=cf with keys=(k1(row),k3(cq)) . Majority of the developments are planned to be on bases of Hypertable (https://github.com/kashirin-alex/hypertable).

The storage-form in the “Super Wide Column DB” is based on column-id and range-id, which on path consist CellStores and CommitLogs files at any point one server is responsible for a range-id on column-id.

The CellStores are Files storing Cells in serialized form that are after latest compaction whereas CommitLogs are the open-file-descriptor to which current data is added.
The Serialization of data in a CellStore/CommitLog file: (delimited with “|” for visual-representation of NONE)
*|Blocks(Header | Compressor(Cells)) | Fixed-Index | Variable-Index | CellStore-Trailer|*
The Cell-Serialization: *|Key-length(int32)|Key-serialized| Value-length(int32)|Value-Data|*
Key serialization: *|Key-flag(int8)|Key-control(int8)|Keys-count(int8)|joined(Keys[N]\0)|Timestamp(int64)|Revision(int64)|*

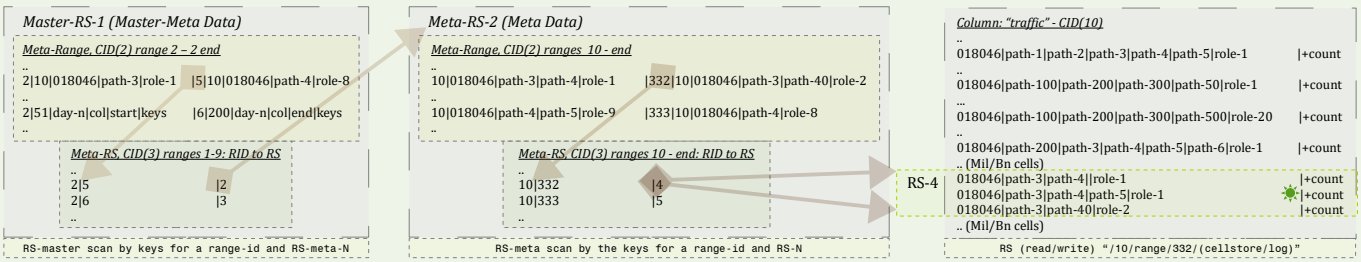
The Cells Ranges, a range is a Keys-start to Keys-end, SWC DB use a self-explanatory master-ranges that define ranges to meta-ranges of data-ranges(cells-range) whereas on range-locator it includes the Keys comparison on the comparators of request, resulting in as most narrowed scan of cells-ranges.

System's reserved columns id[1-9],
1: IDENTIFIER a counter type column
2: RANGES
3: RID(RANGE-ID) to RS(N)
4: Column ID, Column Name and serialized Column-Scheme

The limitations that can be over-seen are:
- ✗ Maximum number of columns, it is store-size of int64($2^{64}$) – 10(reserved cols) which can be improved by CID to be a string-type.
- ✗ Maximum size of Value or Key(after serialization), it is 4GB, while for such data size other limitations probably apply
- ✗ Maximum number of concurrent connections to a given server instance, it is the total available ports on the server, which can be further improved by using IPv6 and so as several IPs.

The capabilities to expect:
- ✔ Supposing one server can handle 2bil cells with one cell is being keys(1024B) and value(1024B) a 4TB in volume over 1300 ranges, to apply such base to RS-master means there are 500k RS-meta with 667mil RS-data with cell value being just a 4KB, that makes a SWC DB cluster in total handling 5.7ZB(zettabyte) of data volume and more on a compression ratio.
- ✔ A client can read at 100%(while Client's and RS's are equivalent) bandwidth, considering a perfect scan case of each client is requesting on different ranges, number of clients at a given time can be by the number of RS-data using 100% bandwidth each.

Some examples:
- Search indexing at https://thither.direct/opensearch/ with Wide Column it is being row=”sequences-of-words:domain:path” cf=”lang” whereas with Super Wide Column it can be changed to keys=[“sequences-of-words”, “domain”, “path”, “lang”], makes the scan-select much optimized, especially if to query words-data of a domain & path, it would go on to ranges that start with domain & path skipping the seek through ranges of several other many domains that as well include the same word-sequences. While to have the same query on a Wide Column would require tripling the volume of data by using more indexes of word-sequences on a domain (and path) such as. row=”domain:sequences:path” & row=”domain:path:sequences”. At current period the “open-search” on Thither.Direct does not offer querying data(words) on a site:domain or info:url-path as it is unreasonable over the data-volume overheads.
- A theoretical requirement for a building security tracking. Track of how many(an atomic-counter) personnel passed in an area of a building by role on a day:



# SWC DB: Applications

author: Kashirin Alex
kashirin.alex@gmail.com

## Range-Server-Manager{N} (*rs_mngr*)

One or Many with standbys: manage ranges on columns ID range or all
configuration:

```
rs.mngr.host=iP{1}:[1-9]
rs.mngr.host=iP{2}:[1-9]
rs.mngr.host=iP{1}:[10-100]
rs.mngr.host=iP{2}:[10-100]
```

OR

```
rs.mngr.host=iP{1}
rs.mngr.host=iP{2}
```

* assign rs{N} to new RangeServer connection
* 1st RangeServer systematically assigned with META_COLUMNS(1-3)(Range-IDs) (Master-RS)
* on-interval > check-ranges assignments and load balance (unload/assign Ranges )
* on disconnected RangeServer assign ranges to other Range-Servers (incl. Master-RS)
* notified by Range-Server on handled columns scheme updates and add/remove column
* notify Range-Servers handle ranges of updated column schemes or removed column
* assign rs{N} to new added columns
* manage barriers for range-ids
* handle requests from Range-Servers for range-barrier start/finish
* handle requests for address of RangeServer (rs-N) on range-barrier condition
* handle client requests of get_meta from master-rs (a broker to Master-RS requests)

* GC on FS, in-case, ranges removed while cell-stores and logs remained

## File System (*libColsDbFS – lib:Local/Hdfs/Ceph/FsBroker*)

RANGE PATH: `/colsdb/ColmID_path/range/RangeID_path`
COMMIT-LOGS: `(RANGE PATH)/log/LOG-NAME/{N}fragment.log`
CELLSTORES: `(RANGE PATH)/cs/{N}.cs`

## lib-DB-Client (*libSwcDbClient*)

Handle Scan request (select/insert/delete):
Scan requests (+ Range-Locator)
Manage Communications to RangeServers and Managers
Handle column add, remove and scheme updates

## Range-Server {N} (*rs*)

Handle column rename and schemes updates
Notifying to corresponding Range Server Manager on columns changes
Handle range-scans
Handle compaction
Notify Range-Server-Manager on ranges change (start-finish barrier)

### Master {1} - RS

Handle master-ranges + data:
Identifiers, col-1: range: (1 → 1)
Ranges, col-2: range: (2 → 2 END_KEY) Self-Explanatory
RS on RID, col-3: range: (1 → 9 END_KEY)

### Meta {N} - RS

Handle meta-ranges, as defined by Master-RS:
Self-Explanatory Ranges, cols [2-3]: range: (10 → END_KEY)
Requests to Master-RS for RS on RID

(System Reserved columns ids [1-9] META_COLUMN)

## Clients:

Range-Servers (master, meta, manager)
ThriftBroker
Utilities
User Application

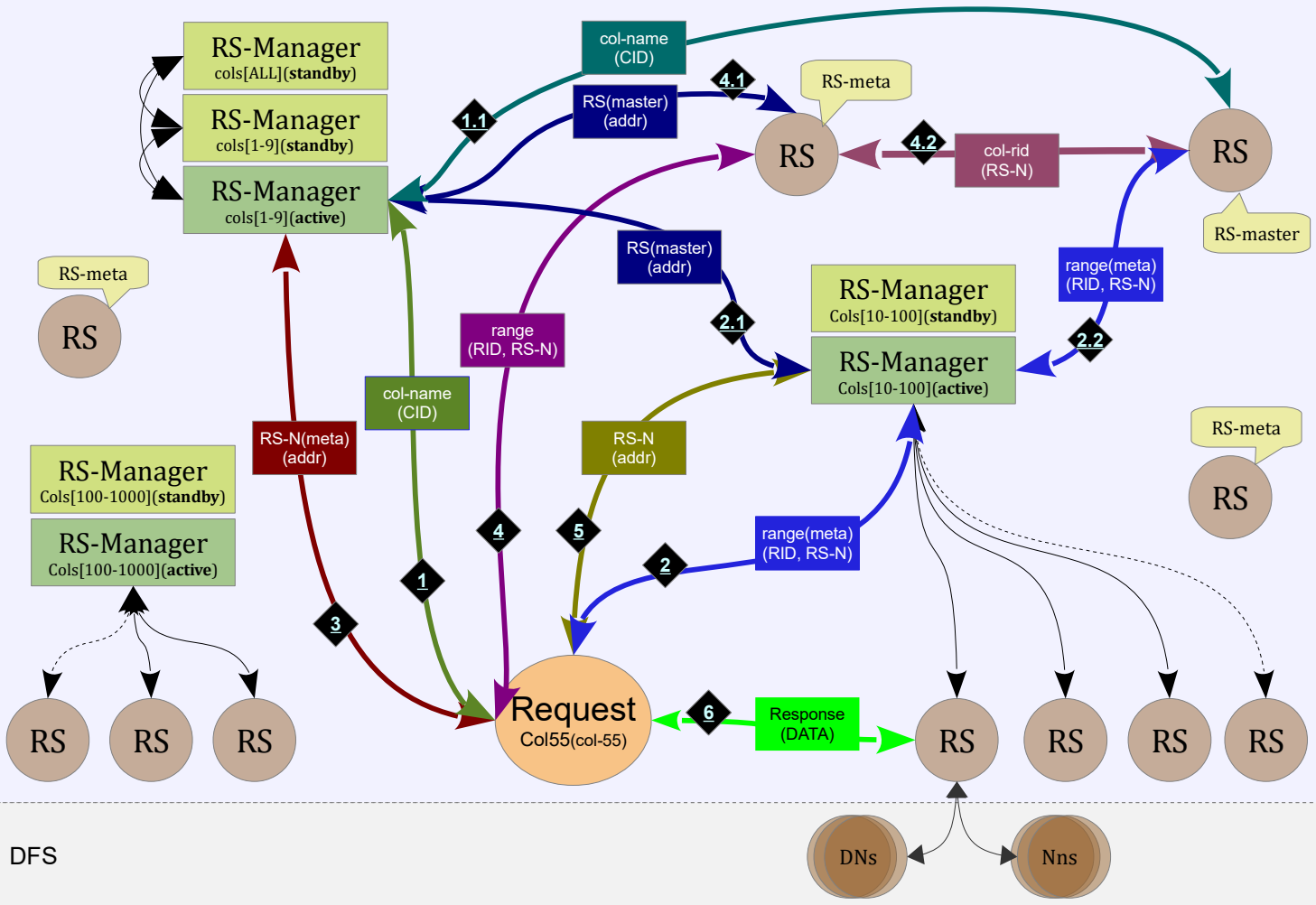
# SWC DB: Failure Tolerance

author: Kashirin Alex
kashirin.alex@gmail.com

- ✔ A failed request to a RS-Manager is a connection fail-over to next in list from 'rs.mngr.host' configuration.
- ✔ A failed request to a RS(Master, Meta, Data)-N is fail-over to the new RS(addr) assign to RS-N by RS-Manager.
- ✔ RS-Manager, on interval or disconnection of a managed RS(either role), request to load ranges to another RS.
- ✔ RS-Master, as been a single instance, have connections only (in the case it is handling only master-ranges) with It's RS-Manager and requests for RS-N by column+rid from other RS-Managers{n} and RS{n}-Meta.
- ✔ Distribute File System, depends on the system and it's feature of routing to a datanode .
- ✔ RS(any) in case a connection or file-descriptor failure  try to reconnect to the DFS.
- ✔ Communication over-heads of a resolved-data to column-name, RID-location or RS-address are kept while connection and data discard by TTL or a notification from RS-Managers on changes.

  In worst case of outdated data being used with a request the RS return an error of a NOT_LOADED_RANGE.

**DB-Client:** *libColsDbClient.so*



# SWC DB: LIB-DB-Client, Scan request (+ Range-Locator)

author: Kashirin Alex
kashirin.alex@gmail.com

## Basic Process Flow of Scan request (+ Range-Locator)

Ranges Scan is done on per column base in-parallel(a client's max-range-locators config) with column's Scan Specifications

Scan-Specifications = cid, ScanSpecCellKeys(*keys_start, keys_finish*)

result = 0, last_cell_keys = NULL, new_start_keys = False, start_keys = ScanSpec.start_keys

```
DO scan_ranges_master:
    1  get column-ID by name => (cid)
            RS-MANAGER[cid-2] – req, (cid-4, [="ReqColName"])
                    get column-ID by name => (cid)
                    RS-MASTER
                            Scan-do, req (1-cell) =>  (cid)
    2  get range-master-meta by (cid,  start_keys, end_keys)  => (rid, rs-N, next_meta_rid?)
            RS-MANAGER[cid]
                RS-MANAGER[cid-2]
          2.1           get RS-master-addr => (addr)
                RS-MASTER
          2.2           get range-master-meta => (rid, rs-N, next_meta_rid?)
                            Scan-do (2-cell)(cid-2, [="cid", start_keys], [>="cid", end_keys])  = rid(meta)
                            Scan-do (1-cell)(cid-3, ['cid-2', ="rid"]) = RS-N(meta)
        If no range-master-meta(rid, rs-N)
                goto finish
EXCEPT COMM:
        goto scan_ranges_master
```

```
DO scan_ranges_meta:
    3  get RS-meta-addr by meta(rs-N, rid(barrier) => (addr)
            RS-MANAGER[cid-2] = > (addr)
    4  get range-meta by (cid,  start_keys, end_keys)  => (rid, rs-N, next_rid?)
            RS-META
          4.1   get range-meta by (cid,  start_keys, end_keys)  => (rid, rs-N)
                    Scan-do (2-cell)(["cid", start_keys], ["cid", end_keys])  = rid
          4.2   Scan-request(cid-3, ['cid', ="rid"]) = RS-N
        If no range-meta(rid, rs-N)
                if new_start_keys or next_meta_rid
                        goto scan_ranges_master
                goto finish
EXCEPT COMM, NOT_LOADED_RANGE:
        goto scan_ranges_meta
```

```
DO scan_ranges_data:
    5  get RS-addr by meta(rs-N, rid(barrier) => (addr)
            RS-MANAGER[cid-2] = > (addr)
    6  results = (RS-addr) scan-do(ScanSpecs)
       if no results
                if next_rid
                        goto scan_ranges_meta
                goto finish
        (call_back) (available results), result+=results
        last_cell_keys=Result[-1]
EXCEPT COMM, NOT_LOADED_RANGE:
        goto scan_ranges_data
```

```
if result < limit(cell_limit) && last_cell_keys < start_keys:
        new_start_keys=True
        start_keys = last_cell->keys (all -ge on keys changed to -gt)
        goto scan_ranges_meta
DO finish:
        return result (call_back)
```

# SWC DB: Query (SQL) scan

author: Kashirin Alex
kashirin.alex@gmail.com

```
select [where_clause [Columns-Intervals or Cells-Intervals]] [Flags(global-scope)];
```

**Columns-Intervals**: *if not set, it is all columns from keys start to finish.*

```
col(column-name-a1) = ( [Cells-Intervals] [and] [Cells-Intervals] [and] .. [Cells-Intervals] )
[and] ..
col(column-name-b1, .., column-name-b2) = ( [Cells-Intervals] [and] [Cells-Intervals] [and] .. [Cells-Intervals] )
```

**Cells-Intervals**: *if not set, it is keys start to finish.*

```
cells = ( [Cells-Interval] Flags(interval-scope) )
[and]
cells = ( [Cells-Interval] Flags(interval-scope) )
[and]..
cells = ( [Cells-Interval] Flags(interval-scope) )
```

**Cells-Interval**:

```
[ Condition-Keys ] [and] [ Condition-Value ] [and] [ Condition-Timestamp ]
```

**Condition-Keys**: *keys comparator apply to every key that do not have a dedicated comparator, exact-match is* keys=('k1', 'k2',,,'kN')

```
keys [comparator] ( [comparator] "str-1", [comparator] "str-2", [comparator] "str-3", [comparator] "str-N")

      or (in-range)

([comparator] "str-1",  [comparator] "str-N")      [ <=  or  < ]  keys  [ <=  or  < ]      ([comparator] "str-1",  [comparator] "str-N")
```

**Condition-Value**:

```
value [comparator] "string"

      or (for columns of counter type), not applicable comparators (prefix and regexp)

value [comparator] "int64_t(string)"
```

**Condition-Timestamp**: *not applicable comparators (prefix and regexp)*

```
timestamp [comparator] "YYYY/MM/DD HH:MM:ss.mmmuuunnn"

      or (in-range)

"YYYY/MM/DD HH:MM:ss.mmmuuunnn"  [ <=  or  < ]  timestamp  [ <=  or  < ]  "YYYY/MM/DD HH:MM:ss.mmmuuunnn"
```

**Comparator**:

```
[ =^ ]  :   prefix   (starts-with)
[ >  ]  :   -gt      (greater-than)
[ >= ]  :   -ge      (greater-equal)
[ =  ]  :   -eq      (equal)
[ <= ]  :   -le      (lower-equal)
[ <  ]  :   -lt      (lower-than)
[ != ]  :   -ne      (not-equal)
[ re ]  :   regexp   (regular-expression)

* -gt,-ge,-le,-lt are a bit-wise comparison
```

**Flags**: *at global-scope apply to Cells-Interval flags to which does not have flags definitions*

```
[ keys_only ]      =  TRUE on set            # default FALSE
[ return_deletes ] =  TRUE on set            # default FALSE
[ limit            =  NUMBER(uint32_t) ]     # default ALL
[ limit_by         =  "KEYS" or ".." ]       # default KEYS
[ offset           =  NUMBER(uint32_t) ]     # default 0
[ offset_by        =  "KEYS" or ".." ]       # default KEYS
[ max_versions     =  NUMBER(uint32_t) ]     # default ALL
```

**An Example:**

```
select
  where
    col(ColNameA1) = (
      cells = ( (>='1-') <= keys = (<='1-1-',="1") and value = "Value-Data-1" and timestamp > "2010/05/29" limit=10 limit_by="KEYS" )
    )
  and
    col(ColNameB1, ColNameB2) = (
      cells = ( (>='2-') <= keys = (<='2-2-',"1") and value = "Value-Data-2" and timestamp > "2010/05/29" )
      and
      cells = ( keys = (<='21-',"1") and timestamp > "2010/05/29" )
    )
  max_versions=1;
```

# SWC DB: Scan Specs & Results

author: Kashirin Alex
kashirin.alex@gmail.com

## Scan Specs, lib-DB-Client:

***ScanSpec*** *(*

```
    ListColumns    columns;
    Flags          flags;
```

*)*

***ColumnIntervals*** *(*

```
    int64_t           cid
    ListCellsInterval cells_intervals
```

*)*

*The object-type is applied to the range-locator (Client)*

***CellsInterval*** *(*

```
    Keys       keys_start, keys_finish;
    Value      value;
    Timestamp  ts_start, ts_finish;
    Flags      flags;
```

*)*

***Keys*** *(*

```
    ListKeys   keys;
```

*)*

***Flags*** *(*

```
    uint32_t    limit, offset, max_versions;
    LimitType   limit_by, offset_by;
    bool        return_deletes, keys_only;
```

*)*

***Key*** *(*

```
    const char*  key;
    size_t       key_len;
    Comparator   comp;
```

*)*

***Value*** *(*

```
    const char*  value;
    size_t       value_len;
    Comparator   comp;
```

*)*

***Timestamp*** *(*

```
    int64_t     ts;
    Comparator  comp;
```

*)*

## Scan Response, lib-DB-Client:

***Result*** *(*

```
    List<Col>   cols
    // ResponseFlag status = OK/PARTIAL/ERROR
    // Strings error_rs = ["N",]
```

*)*

***Col*** *(*

```
    String      name
    String      id
    List<Cell>  cells
```

*)*

***Cell*** *(*

```
    Strings     keys
    int64_t     timestamp
    char-array  value
    uint32_t    value_len
```

*)*

# SWC DB: Column Schema & Actions on Columns

author: Kashirin Alex
kashirin.alex@gmail.com

Although, there are schemas in the SWC-DB these can be considered as schema-less, exception to TTL, Counter and Max-Versions at the Cells level.

### Location of Schema and Column-Name to ID:

Reserved Column-ID 4 with cell:
Keys = ("FLAG" "Col-Name", "CID")
Value = (value-serialized)version(int8)
|counter(int8)|ttl(int32)|compression(int8)|bloomfilter(int8)|max_versions(int32)|time_order(int8)|replication(int8)|blocksize(int32)

### Configuration Options:

The following configurations available in the Column-Schema:

- CELL-LEVEL:
  - TTL: (int32_t) seconds
  - COUNTER: (bool), default False
  - MAX_VERSIONS: (int32_t), default 1 - not applicable with COUNTER
  - TIME_ORDER: ASC/DESC, default ASC – applied to order of MAX_VERSIONS
- BLOCK-LEVEL:
  - COMPERSSION: none/snappy/zlib/zstd/bmz/lzo/quicklz
  - BLOCKSIZE: (int32_t)
- CELLSSTORE-LEVEL:
  - REPLICATION: (int8_t) – replication factor applied to the DFS supporting file-replication , default 3
  - BLOOMFILTER: (int8_t) – none/all-keys

### Adding a Column

**SQL:**

```
add column (
      NAME="string",
      COUNTER=bool,
      MAX_VERSIONS=number,
      TTL=number,
      COMPRESSION="string",
      BLOCKSIZE=number,
      REPLICATION=number,
      BLOOMFILTER="string"
);
```

**lib-DB-Client:**

```
ColumnSpec (
      String           name
      int64_t          cid
      Bool             counter
      int32_t          max_versions
      int32            ttl
      Compressor::ENUM compression
      int8_t           replication
      BloomFilter      bloomfilter(TYPE::ENUM, factor, functions)
)
```