common_schema

DBA's framework for MySQL

Shlomi Noach

http://openark.org

common_schema

DBA's framework for MySQL

- About
- Schema analysis
- eval() and SQL generation
- Process & security
- Status & transaction monitoring
- QueryScript

About common_schema

- common_schema is an open source project, licensed under the New BSD License.
- Authored by Shlomi Noach
  http://openark.org
- Major contributions by Roland Bouman
  http://rpbouman.blogspot.com
- Several contributions & suggestions by the community
  (Thanks!)

About common_schema

- common_schema is a framework: a set of views, routines and a script interpreter
- It is a *schema* that lies next to **INFORMATION_SCHEMA**.
- It lies completely within the MySQL server, and does not require external packages or dependencies. No Perl scripts nor UDFs or plugins.

Getting & Installing

- common_schema is merely a SQL file.
- Currently hosted on Google Code: http://code.google.com/p/common-schema/
- Install by importing SQL file into MySQL:

```
bash$ mysql < /tmp/common_schema-1.2.2.sql
complete
- Base components: installed
- InnoDB Plugin components: installed
- Percona Server components: not installed

Installation complete. Thank you for using
common_schema!
```

Schema analysis views

- Views providing non-trivial information about your table design, keys, foreign keys, and more.
- While **INFORMATION_SCHEMA** provides with complete info, it is ofter difficult to aggregate. It is sometimes too normalized, and at other times too de-normalized.
- We consider a couple examples. Full list: http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/schema_analysis_views.html

redundant_keys

- Find duplicate/redundant keys in your schema.
  http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/redundant_keys.html
  - Similar to pt-duplicate-key-checker

```
mysql> SELECT * FROM redundant_keys WHERE
table_schema='sakila' \G
              table_schema: sakila
                table_name: rental
      redundant_index_name: rental_date_2
   redundant_index_columns: rental_date
redundant_index_non_unique: 1
       dominant_index_name: rental_date
    dominant_index_columns: rental_date, inventory_id,
                              customer_id
 dominant_index_non_unique: 0
            sql_drop_index: ALTER TABLE `sakila`.`rental`
                            DROP INDEX `rental_date_2`
```

sql_range_partitions

- Analyze your range partitioned tables
  - Looks for a pattern in partition values
  - Offers the next-CREATE-statement, next-DROP-statement
- Supports 5.1 and 5.5 notations.
- Elegantly solves what is usually done by home-brewed scripts.

sql_range_partitions

- **sql_add_next_partition** column offers the **ADD/REORGANIZE PARTITION** statement for each table

```
mysql> CREATE TABLE test.report …  PARTITION BY RANGE (…) ;

mysql> select * from sql_range_partitions where
table_name='report' \G
            table_schema: test
              table_name: report
        count_partitions: 7
sql_drop_first_partition: alter table `test`.`report`
                          drop partition `p0`
  sql_add_next_partition: alter table `test`.`report`
   reorganize partition `p6` into (
     partition `p_20090701000000` values less than
     (1246395600) /* 2009-07-01 00:00:00 */ ,
     partition p_maxvalue values less than MAXVALUE
   )
```

common_schema
DBA's framework for MySQL

SQL generation & eval()

- Both views above present with SQL columns, offering a statement to execute.
- This is at the heart of common_schema's views, and is part of the server-side mechanism the framework strongly supports.
- The eval() routine accepts such SQL columns and executes (evaluates) them.

eval()

- Accepts a query returning a text column. Column data is expected to be SQL statements to be executed.

```
mysql> call eval("
   SELECT sql_add_next_partition
   FROM sql_range_partitions
   WHERE table_schema='webprod'");
Query OK, 0 rows affected

- A new partition has just been added on
- all range-partitioned tables in `webprod`
```

Process-list views

- Provide extra info, not presented by **PROCESSLIST** or not at all made easy/available by MySQL.
- Information not offered in **PROCESSLIST**:
  - The GRANTEE owning the process: **john@10.0.0.17** != **john@10.0.%**
  - Special notes about the processes (replication client? Replication slaves? SUPER? Myself?)
- Different views tackle different aspects.

http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/process_views.html

Process-list views

- processlist_grantees extends **PROCESSLIST** and adds missing info.
- Various views aggregate processes by different params (connection origin, state, totals), and provide with runtime metrics
- slave_status provides a query-able minified version of **SHOW SLAVE STATUS**
- slave_hosts lists all known connected slaves

Process-list views

- Like many other views, processlist_grantees offers SQL columns.
- Example: kill all queries executed by “normal” users and which are running for over 10 minutes:

```
mysql> call eval("
   SELECT sql_kill_query
   FROM processlist_grantees
   WHERE TIME > 600
     AND NOT is_super AND NOT is_repl
   ");
Query OK, 0 rows affected
```

Security views

- MySQL's **SHOW GRANTS** command has many limitations:
  - Only for a single grantee
  - Cannot be used by queries
  - Does not provide with metadata about privileges domains
- **INFORMATION_SCHEMA**'s privileges tables are missing routine privileges and are inconveniently denormalized

http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/security_views.html

Security: sql grants

- sql_grants & sql_show_grants overcome said limitations by providing SQL access to account privileges.
- sql_grants example:

```
           GRANTEE: 'world_user'@'localhost'
              user: world_user
              host: localhost
        priv_level: `world`.*
   priv_level_name: schema
current_privileges: INSERT, SELECT, UPDATE
         sql_grant: GRANT INSERT, SELECT, UPDATE ON
                    `world`.* TO 'world_user'@'%'
        sql_revoke: REVOKE INSERT, SELECT, UPDATE ON
                    `world`.* FROM 'world_user'@'%'
```

Security: sql grants

- Consider tables involved in these views:

user

procs_priv

numbers

COLUMN_PRIVILEGES

TABLE_PRIVILEGES

USER_PRIVILEGES

SCHEMA_PRIVILEGES

- Duplicating accounts, finding similar or redundant accounts are usage samples for these views.

InnoDB views

- InnoDB's **INFORMATION_SCHEMA** tables are highly informative, yet very normalized.
- These tables provide info about transactions, runtimes, locks, more...
- Almost every reasonable query against these tables must involve a join or two.
- common_schema offers common useful shortcuts.

http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/innodb_plugin_views.html

innodb_transactions

- This view tells us:
  - Which transactions are running?
    - For how long?
    - How long are they being idle? Locked?
  - What queries are they issuing?
- Example: kill idle transactions:

```
mysql> call eval("
  SELECT sql_kill_query
  FROM innodb_transactions
  WHERE trx_idle_seconds >= 30
  ");
```

innodb_locked_transactions

- Which transaction is being blocked?
  - By which transaction? For how long?
  - What queries are both transaction executing
- Example: which transactions/queries are blocking other transactions for 30 seconds or more?

```
mysql> SELECT DISTINCT locking_trx_query
  FROM innodb_locked_transactions
  WHERE trx_wait_seconds >= 30;
```

Many more views...

- sql_accounts allows for account blocking & releasing without touching privileges.
- global_status_diff_nonzero is a mini-monitoring view, presenting status change over 10 seconds.
- last_query_profiling shows aggregated profile for last executed query.

Routines

- common_schema offers more than 50 useful stored routines, part of the framework's function library.
- From text parsing & manipulation, through process diagnostics, query analysis & dynamic execution, to security routines, the function library extends and complements MySQL's own functions.

http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/execution_routines.html
http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/text_routines.html
http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/security_routines.html
http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/process_routines.html

Security routines

- killall() kills connections by matching text with user, host or grantee.
- security_audit() audits server's privileges tables and configuration to detect such threats as empty or duplicate passwords, excessive privileges, excessive host access, “old passwords” etc.

```
mysql> call killall('analytics');

mysql> call killall('localhost');
```

Execution routines

- eval() evaluates the queries generated by a given query.
- exec(), exec_file() dynamically executes a given query or semicolon delimited list of queries.
- run(), run_file() execute QueryScript code.

```
mysql> call exec('
   CREATE TABLE test.t(id INT);
   INSERT INTO test.t VALUES (2),(3),(5);
');
```

QueryScript

- A SQL oriented scripting language, offering tight integration with SQL commands, easy and familiar control flow syntax and high level abstraction of complex tasks.
- common_schema implements QueryScript via interpreter, based on stored routines.
- This makes QueryScript suitable for administration and bulk tasks, not for OLTP tasks.

http://common-schema.googlecode.com/svn/trunk/common_schema/doc/html/query_script.html

QueryScript

- Code samples:

```
foreach($table, $schema, $engine:
          table in sakila) {
  if ($engine = 'InnoDB')
    ALTER TABLE :${schema}.:${table}
    ENGINE=InnoDB ROW_FORMAT=COMPRESSED
    KEY_BLOCK_SIZE=8;
}
```

```
split (DELETE FROM sakila.rental WHERE
   rental_date < NOW() - INTERVAL 5 YEAR) {
  throttle 2;
}
```

Why QueryScript?

- Stored routine programming is a pain:
  - Requires one to actually store the routine within the schema: can't just *run something*.
  - Syntax is cumbersome (ANSI:SQL).
  - Does not offer deeper insight into MySQL's limitations and bottlenecks.
  - Does not provide with syntax for oh-so-common tasks
  - Verbose. Can't see the forest for the trees.

Compare: stored routine

```
DELIMITER $$

DROP PROCEDURE IF EXISTS some_proc $$
CREATE PROCEDURE some_proc()
READS SQL DATA
SQL SECURITY INVOKER
begin
  declare some_id bigint unsigned default null;
  declare done tinyint default 0;
  declare my_cursor cursor for SELECT some_id FROM some_table;
  declare continue handler for NOT FOUND set done = 1;

  open my_cursor;
  read_loop: loop
    fetch my_cursor into some_id;
    if done then
      leave read_loop;
    end if;
    -- do something with some_id
  end loop;

  close my_cursor;
end $$

DELIMITER ;
```

Compare: QueryScript

```
set @script := '
  foreach ($some_id: SELECT some_id FROM some_table)
  {
    -- do something with $some_id
  }
  ';
call run(@script);
```

- Significantly less overhead
- Familiar C-family syntax
- No need to store the code in schema
- Can execute script directly from file

```
call run('/path/to/script.qs');
```

QueryScript conditions

- Familiar if, while or loop-while statements.
- But conditions are also tightly integrated with SQL, and so queries make for valid conditions.

```
if (@val > 3) {
  pass;
}
while (DELETE FROM world.Country
       WHERE Continent = 'Asia' LIMIT 10)
{
  throttle 2;
}
```

QueryScript foreach

- A sophisticated looping device, allowing iteration over queries, sets, numbers range, databases, tables...
- But also acknowledges MySQL limitations and provides with a safer access method.
- Table iteration uses **I_S** optimizations to avoid excessive locks:

```
foreach($tbl, $scm: table like wp_posts)
  ALTER TABLE :${scm}.:${tbl} ADD COLUMN
  post_geo_location VARCHAR(128);
```

QueryScript split

- Automagically breaks a bulk operation into smaller chunks.
- Supports **DELETE**, **UPDATE**, **INSERT…SELECT**, **REPLACE…SELECT**
- Supports single and multi table statements

```
split (DELETE FROM sakila.rental WHERE
  rental_date < NOW() - INTERVAL 5 YEAR);

split (insert into world.City_duplicate
       select * from world.City) {
  throttle 1;
}
```

QueryScript: more goodies

- throttle statement controls script execution time, reduces server load
- Local variables auto-cleanup, can be used (expanded) in statements where variables not allowed (table names, LIMIT value etc.)
- try-catch statement is available: easy error handling mechanism
- echo, eval, throw statements make for easy development

call for help()

- common_schema documentation is available in these formats:
  - Online documentation (directly in code repository)
  - Bundled HTML download
  - Inline: Just call for help():

```
call help('split');
+--------------------------------------------------------------------------------+
| help                                                                           |
+--------------------------------------------------------------------------------+
| QueryScript Flow Control: split statement                                      |
|                                                                                |
| SYNOPSIS                                                                       |
|                                                                                |
| Single table operations, autodetect mode:
```

Support common_schema

- Download and try it out
- Report issues
  http://code.google.com/p/common-schema/issues/list
- Happy to receive ideas and contributions
  http://bit.ly/UPC3vh
  http://code.google.com/p/common-schema/issues/entry?template=Request%20for%20new%20component
- Above all else: spread the word!

Thank you!

- Visit http://openark.org for news & updates.
- Other open source projects:
  - openark kit
    http://openarkkit.googlecode.com/svn/trunk/openarkkit/doc/html/introduction.html
  - mycheckpoint
    http://code.openark.org/forge/mycheckpoint