schemadiff

In-memory schema analysis, validation, normalization, diffing, and manipulation

Shlomi Noach
PlanetScale

FOSDEM 2025

Incentive: diff, and much beyond

```sql
CREATE TABLE `schema_migrations` (
 `id` int unsigned NOT NULL AUTO_INCREMENT,
 `mysql_table` varchar(128) NOT NULL,
 `migration_statement` text NOT NULL,
 PRIMARY KEY (`id`)
);
```

```sql
CREATE TABLE `schema_migrations` (
 `id` bigint unsigned NOT NULL AUTO_INCREMENT,
 `mysql_table` varchar(128) NOT NULL,
 `migration_statement` text NOT NULL,
 `completed_timestamp` timestamp(6) NULL DEFAULT NULL,
 `migration_status` varchar(128) NOT NULL,
 PRIMARY KEY (`id`),
 KEY `completed_status_idx` (
   `completed_timestamp`,`migration_status`)
);
```

```sql
ALTER TABLE `schema_migrations`
  MODIFY COLUMN `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  ADD COLUMN `completed_timestamp` timestamp(6) NULL,
  ADD COLUMN `migration_status` varchar(128) NOT NULL,
  ADD KEY `completed_status_idx` (`completed_timestamp`, `migration_status`);
```

Agenda

A programmatic approach to schema analysis:

- Parsing
- Normalization, validation
- Diff
- In-memory manipulation
- Change validation and dependencies
- Migration paths
- Performance
- Change analysis

Agenda

Where you will find it useful

About me

Engineer at **PlanetScale**

Maintainer for **Vitess**

Author of **gh-ost**, **orchestrator**, and other open source projects

github.com/shlomi-noach

PlanetScale

The database platform built for scale

Founded Feb. 2018 by co-creators of Vitess

MySQL-compatible serverless database platform, built for developers

Built on top of Vitess

Vitess

MySQL-compatible, horizontally scalable, cloud-native database clustering system.

- CNCF graduated project
- Open source, Apache 2.0 licence
- Contributors from around the community

schemadiff

Objective: do not require MySQL

Source can be any text/file. No INFORMATION_SCHEMA.

Environment

```go
mysqlVersion := "8.0.35"

collEnv := collations.NewEnvironment(mysqlVersion)

vtenv, err := vtenv.New(vtenv.Options{
  MySQLServerVersion: mysqlVersion,
})
if err != nil { ... }

env := schemadiff.NewEnv(
  vtenv, collEnv.DefaultConnectionCharset())
```

sqlparser: Parse()

Low level Vitess parsing library, used by schemadiff

```
stmt, err :=
  env.Parser().ParseStrictDDL(sql)
if err != nil { ... }

// Assume we expect a CREATE TABLE:
createTable, ok :=
stmt.(*sqlparser.CreateTable)
if !ok { ... }
```

Parser: AST

```go
type CreateTable struct {
 Temp         bool
 Table        TableName
 IfNotExists  bool
 TableSpec    *TableSpec
 OptLike      *OptLike
 Comments     *ParsedComments
 FullyParsed  bool
}
```

Parser: AST

```go
type TableSpec struct {
 Columns                []*ColumnDefinition
 Indexes                []*IndexDefinition
 Constraints            []*ConstraintDefinition
 Options                TableOptions
 PartitionOption *PartitionOption
}
```

Parser: AST

```
type ColumnDefinition struct {
 Name IdentifierCI
 Type *ColumnType
}
```

Parser: AST

```
type ColumnType struct {
 Type string
 Options *ColumnTypeOptions
 Length   *int
 Unsigned bool
 Zerofill bool
 Scale    *int
 Charset ColumnCharset
 EnumValues []string
}
```

sqlparser: format

```
stmt, err :=
  env.Parser().ParseStrictDDL(sql)
if err != nil { ... }
fmt.Println(
  sqlparser.CanonicalString(stmt))
```

```
> ALTER TABLE `t` MODIFY COLUMN `i` bigint
```

Normalization

```
create table t (
  id int primary key,
  i int(11) default null,
  v varchar
    charset utf8mb4
);
```

```
create table t (
  id INT,
  i int,
  v VARCHAR,
  primary key (id)
);
```

Normalization

Aiming for a minimalistic presentation.

```
CREATE TABLE `t` (
 `id` int,
 `i` int,
 `v` varchar,
 PRIMARY KEY (`id`)
);
```

Semantics

The parser only validates syntax

Validation

Per table and cross schema

```
create table t (
  id int primary key,
  i int,
  key i2_idx (i2),
  constraint t_fk foreign key (i)
    references parent (id, j),
  primary key (i)
)
```

Validation

Per table and cross schema

```
create table t (id int ...);
create view v as select * from t;
drop table t;
```

Loading schemas

Parse, normalize, validate

```
schema, err := NewSchemaFromSQL(env, sql)
if err != nil {...}
for _, e := range schema.Entities() {
  fmt.Println(
   e.Create().CanonicalStatementString())
}
```

```
> CREATE TABLE t1 (...)
> CREATE TABLE t2 (...)
```

schemadiff CLI

Thin wrapper around schemadiff library

```
$ schemadiff load --source /tmp/my_schema.sql
```

```
CREATE TABLE `schema_migrations` (
 `id` bigint unsigned NOT NULL AUTO_INCREMENT,
 `migration_uuid` varchar(64) NOT NULL,
 PRIMARY KEY (`id`)
);
CREATE TABLE `vreplication` (
 `id` int NOT NULL AUTO_INCREMENT,
 `workflow` varbinary(1000),
 PRIMARY KEY (`id`),
 KEY `workflow_idx` (`workflow`(64))
);
```

github.com/planetscale/schemadiff

Diff: AST

```
type TableSpec struct {
 Columns         []*ColumnDefinition
 Indexes         []*IndexDefinition
 Constraints     []*ConstraintDefinition
 Options         TableOptions
 PartitionOption *PartitionOption
}
```

Diff

Any two tables, two views, or two schemas

```
diff, err := schemadiff.DiffCreateTablesQueries(
  env, from, to, hints)
if err != nil {...}
if diff != nil {
 fmt.Println(diff.CanonicalStatementString())
}
```

```
> ALTER TABLE `t` ALTER CHECK `Check1` ENFORCED
```

Diff

Any two tables, two views, or two schemas

```
diff, err := schemadiff.DiffSchemasSQL(
  env, from, to, hints)
if err != nil {...}
for _, d := range diff.UnorderedDiffs(ctx) {
 fmt.Println(d.CanonicalStatementString())
}
```

```
> DROP TABLE `t1`
> ALTER TABLE `t2` MODIFY COLUMN `id` bigint
```

schemadiff CLI

```
$ schemadiff diff --source /tmp/source.sql \
  --target /tmp/target.sql
```

```
DROP TABLE `a`;
ALTER TABLE `b` MODIFY COLUMN `id` bigint unsigned NOT
NULL AUTO_INCREMENT, ADD KEY `ab_idx` (`a`, `b`);
```

Diff

```
type EntityDiff interface {
 EntityName() string
 Entities() (from Entity, to Entity)
 Statement() sqlparser.Statement
 CanonicalStatementString() string
 ...
}
```

Walk()

```
 _ = sqlparser.Walk(func(node sqlparser.SQLNode)
(kontinue bool, err error) {
    switch node := node.(type) {
    case <one of supported types>:
    case <one of supported types>:
    }
    return true, nil
 }, expression)
}
```

Walk() use cases

Detect *dangerous* operations

Lint/reject certain features

Lint/reject certain attributes

Analyse complex expression

Modify elements

Walk()

```
schemaDiff, err := schemadiff.DiffSchemasSQL(
  env, fromSql, toSql, &schemadiff.DiffHints{})
if err != nil { ... }
danger := false
for _, diff := range schemaDiff.UnorderedDiffs(ctx) {
 _ = sqlparser.Walk(func(node sqlparser.SQLNode) (kontinue bool, err error) {
   switch node := node.(type) {
   case *sqlparser.DropTable:
     danger = true // or e.g. use node.FromTables[0].Name.String()
   case *sqlparser.DropColumn:
     danger = true // use node.Name.Name.String()
   }
   return true, nil
 }, diff.Statement())
}
```

Diff use cases

Schema (change) deployment

Comparing testing/prod env with presumed schema

App managing its backend DB

Vitess backend table deployment

```go
// findTableSchemaDiff gets the diff which needs to be applied
// to the current table schema in order to reach the desired one.
// The result will be an empty string if they match.
// This will be a CREATE statement if the table does not exist
// or an ALTER if the table exists but has a different schema.
func (si *schemaInit) findTableSchemaDiff(tableName, current, desired string) (string, error) {
 hints := &schemadiff.DiffHints{
   TableCharsetCollateStrategy: schemadiff.TableCharsetCollateIgnoreEmpty,
   AlterTableAlgorithmStrategy: schemadiff.AlterTableAlgorithmStrategyCopy,
 }
 env := schemadiff.NewEnv(si.env, si.coll)
 diff, err := schemadiff.DiffCreateTablesQueries(env, current, desired, hints)
 if err != nil {
   return "", err
 }
```

Applying changes

The Diff() can be applied onto a schema/entity

Programmatic schema manipulation

```
schema1, err := NewSchemaFromSQL(env, sql1)
schema2, err := NewSchemaFromSQL(env, sql2)

diff, err := schema1.SchemaDiff(schema2, hints)
diffs, err := diff.UnorderedDiffs(ctx)

result, err := schema1.Apply(diffs)
```

The diff list is not enough

What is a valid sequence to applying them?

```
CREATE TABLE t (...);
CREATE VIEW v AS SELECT * FROM t;


DROP TABLE t;
DROP VIEW v;
```

The diff list is not enough

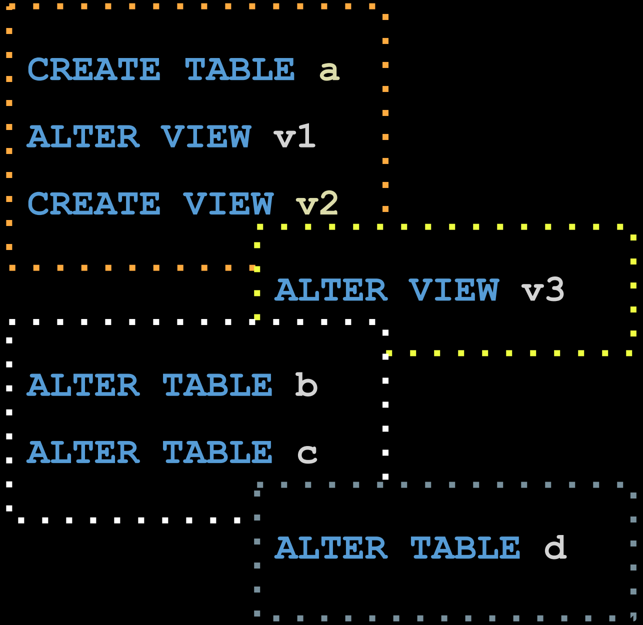
Views and foreign key constraints can create a graph of dependencies within the schema.

A list of diffs can likewise introduce a graph of dependencies within the changes, suggesting an *order* of diffs.

Applying changes

As means to finding a valid order

```
CREATE TABLE a
ALTER VIEW v1
CREATE VIEW v2
ALTER VIEW v3
ALTER TABLE b
ALTER TABLE c
ALTER TABLE d
```



```
CREATE TABLE a
ALTER VIEW v1
CREATE VIEW v2
```

```
ALTER VIEW v3
```

```
ALTER TABLE b
ALTER TABLE c
```

```
ALTER TABLE d
```

Ordered diffs

Or error if no ordering is possible

```
diff, err := DiffSchemasSQL(env, from, to, hints)
if err != nil {...}

diffs, err := diff.OrderedDiffs(ctx)
if err != nil {...}
for _, d := range diffs {
 fmt.Println(diff.CanonicalStatementString())
}
```

schemadiff CLI

```
$ schemadiff ordered-diff --source /tmp/source.sql
\
  --target /tmp/target.sql
```

```
DROP TABLE `a`;
ALTER TABLE `b` MODIFY COLUMN `id` bigint unsigned NOT
NULL AUTO_INCREMENT, ADD KEY `ab_idx` (`a`, `b`);
```

Applying changes

3-way merge

Performance and feasibility

- Textual input vs DB tables input
- Comparing large schemas
- Resolving diff order
- CHECK/KEY/GENERATED Expressions
- Views

Annotated diff

Export a textual visualization of the semantic diff

```go
for _, d := range diffs {
 _, _, unified := d.Annotated()
 fmt.Println(unified.Export())
}
```

```diff
-CREATE TABLE `t1` (
-  `id` int,
-  PRIMARY KEY (`id`)
-)
CREATE TABLE `t2` (
-  `id` int,
+  `id` bigint,
   PRIMARY KEY (`id`)
)
+CREATE TABLE `t4` (
+  `id` int,
+  PRIMARY KEY (`id`)
+)
```

Hints

Control diff behavior:

- AUTO_INCREMENT changes
- Partition rotation
- Constraint names
- Column rename heuristic
- Table rename heuristic
- Enum reordering
- More ...

Beyond the diff

schemadiff further provides schema and schema changes analysis

Expansion/reduction of data scope

Do the changes limit data scope? Is there a risk where migration (or revert) may fail?

```
int -> bigint unsigned

text -> varchar(255)

timestamp null -> timestamp not null
```

(Unique) constraint analysis

Do the changes introduce, or remove, constraints? Is there a risk where migration (or revert) may fail?

INSTANT DDL

Are all changes eligible to use ALGORITHM=INSTANT?

schemadiff precomputes with no need of server (*)

Online DDL analysis

Online DDL eligibility

Find best iteration keys

Map renamed columns

AUTO_INCREMENT changes

Resources

https://vitess.io/blog/2023-04-24-schemadiff/

https://planetscale.com/blog/schemadiff-command-line-tool

https://github.com/planetscale/schemadiff

https://planetscale.com/blog/database-branching-three-way-merge-schema-changes

https://github.com/vitessio/vitess/issues/10203

Thank you!

Reach out on the Vitess Slack workspace

https://vitess.io/slack