Proposal for Negative Grants Project

Reference:
[1] https://jira.mariadb.org/browse/MDEV-14443
[2] https://mariadb.com/kb/en/library/google-summer-of-code-2018/

SQLServer has a **'DENY'** command that blocks access and trumps all other accesses including GRANT. If a user has both a GRANT and a DENY on a given object, DENY will take effect. For the negative privileges project, we can take inspiration from this 'DENY' command and have a logic wherein the 'DENY' dominates over 'GRANT'.
If we have a scenario wherein there are millions of tables in a database and we want to grant a user or a role access to all the tables except for one particular table, it is cumbersome and inefficient to traverse the GRANT graph of all the tables, granting access to all tables and giving negative access to this particular table.

We have different levels of privileges in MariaDB: global level, database level, table level, column level, procedure level, function and proxy level privileges. [1] A user may not be granted create privilege at the user level, but may still have create permission on certain tables or databases.[2]

Each of these privileges has a table associated with them.
For eg. Global level: The mysql.user table contains information about users that have permission to access the MariaDB server, and their global privileges.The table can be queried and although it is possible to directly update it, it is best to use GRANT and CREATE USER for adding users and privileges.

These tables are for positive privileges (grants) and the primary key is a combination of HOST, USER and PASSWORD. One approach for introducing negative grants was to add an extra **is_deny** column and set it to 'Yes'. However, this would create a conflict as there can't be conflicting entries (with both grant and deny for the same primary key (user and host)).
To solve this issue, an efficient way of going about this would be to have **revoke_deny** tables with exactly the same type of columns as the grant privilege tables, except that when 'revoke' of a privilege is set to 'yes' here, this dominates over the grant privilege value in the grants table. Meaning, if a grant privilege is set to 'yes' and we want to deny it, instead of updating the grant table (for eg. the mysqluser-table for global privileges), we instead add an entry to its corresponding revoke table.

Six new tables to be added for reverse privileges:

reverse_priv_global, reverse_priv_db, reverse_priv_table, reverse_priv_proc, reverse_priv_columns, reverse_priv_function,
where columns are identical to the ones in mysql.db, mysql.table_priv, etc.

The code needs to read these tables and store this information in memory using some data-structure. The system needs to be able to answer quickly: is this user (or role) denied access to object X, where object X is a database, a table, a column or a stored procedure.
Need to make the datastructure answer the question in O(1).
We can have updates to the datastructure be slow, O(N) <- N is the number of entries in the structure
acl_load reads these tables and adds info into the datastructure in memory.

Example:
create user foo; <- This creates a row in mysql.user;
Grant select on test.* to foo; <- This creates a row in mysql.db
Grant ALL on test.* to foo; <- This creates a row in mysql.db
DENY update on test.* to foo; <- This info would get stored in the reverse_grants.db table
HOST, USER, DB <- Serves as the primary key for the reverse table as well as the grants table.

GRANT gives access, DENY adds a rule to reject privileges, if they were granted
REVOKE SELECT -> This modifies the regular grant table
REVOKE DENY SELECT -> This modifies reject privileges

Whenever the user issues an action, the check_access function gets triggered, which calls acl_load, which would load these reverse_grants tables into memory. We add a check to see if revoke is 'yes'. If so, we need not check the grants table at all as revoke dominates. If it is 'no', then only the check happens on the grants table (the existing logic).

Illustration:
GRANT SELECT on test.*
DENY SELECT on test.t1;
select c1 from test.t1; //User gets access denied error here

REVOKE DENY SELECT on test.t1; //Removes the deny by altering the Revoke table
DENY SELECT on test.t1 column c2;
SELECT * from t1; //User will get all columns, except c2;

DENY SELECT on test.t1;

```
SELECT * from t1; //User will not see the table;
REVOKE DENY SELECT on test.t1;
SELECT * from t1; //User will get all columns, except c2;
```

Summary:

The basic idea is to persist the entries in the grant privilege tables (not modify them) and to add more revoke tables (to keep the revoke/deny privileges). Any user action would check the revoke table first, thus dominating over the check for the grants privilege tables. It is possible to take inspiration from the existing grants table to design the data structure for the revoke tables. Finally add the 'DENY' command syntax in the grammar.

The first step before implementing user facing commands is to write the logic in check_access for testing the deny grants.
We don't need deny syntax to test this, as we can insert stuff manually into the tables.

```
INSERT into reverse_db_priv values (...)
flush privileges;
```

'Flush' calls acl_load and loads the negative privilege information into memory.
With this approach, we only need to implement insert into and fetch from data-structure. No need for implementing update/delete.

References:
[1]. https://mariadb.com/kb/en/library/grant/
[2]. https://mariadb.com/kb/en/library/mysqluser-table/
[3].https://www.mssqltips.com/sqlservertip/2894/understanding-grant-deny-and-revoke-in-sql-server/

Entry points into the code-base:
check_access
acl_get
acl_getaccess
mysql-select
mysql_select
acl_load -> will read these tables
and add info into the data-structure
Go into grammar
sql_yacc.yy
and extend grammar with DENY syntax