SQL:
Data Definition Language

part 2

Data Definition Language

- Create
  - Table
  - Index
  - View
- Alter
  - Table
- Drop
  - Table
  - Index
  - View
- Grant
- Revoke

Data Definition Language

- Create
  - ~~Table~~
  - Index
  - View
- Alter
  - ~~Table~~
- Drop
  - ~~Table~~
  - Index
  - View
- Grant
- Revoke

Covered in SQL: Data Definition Language Part 1

Create View

Simplified...

CREATE VIEW *view* [(*column1* [, *column2*]...)]
AS *select* [WITH CHECK OPTION];

- CHECK OPTION enforces *view* WHERE clause when *view* is used in UPDATE or INSERT
- *column1*, etc., allows renaming of columns returned by *select* statement

From Uncle Ted's slides

Create View

From the MySQL docs.

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Extra material that will not be on the final exam.

Tables versus Views

- Data is physically stored in tables.
- Data is *not physically stored* in views.
- Views are a prescription for generating data.
  - View rows are generated “on demand.”
- Tables are a physical construct.
- Views are a logical construct.

Guaranteed to be on the final!

Create View

```
CREATE VIEW bucks
  AS SELECT *
       FROM Faculty
       WHERE fsalary > 60000
  WITH CHECK OPTION;
```

Create View

```
CREATE VIEW bucks2
    (id, last, first, mi, dept, salary, manager)
  AS SELECT *
       FROM Faculty
       WHERE fsalary > 60000
  WITH CHECK OPTION;
```

Create View

```
CREATE VIEW canadian_faculty
  (fid, flast, canadian_salary)
  AS SELECT fid, flast, fsalary*1.34
   FROM Faculty
```

Better conversions use constants:

- Quarts → Liters
- Miles → Kilometers

Create View

From the MySQL docs.

- The `CREATE VIEW` statement creates a new view, or replaces an existing view if the `OR REPLACE` clause is given.
  - If the view does not exist, `CREATE OR REPLACE VIEW` is the same as `CREATE VIEW`.
  - If the view does exist, `CREATE OR REPLACE VIEW` is the same as `ALTER VIEW`.

Extra material that will not be on the final exam.

Create View

From the MySQL docs.

- The ***select_statement*** is a `SELECT` statement that provides the definition of the view.
  - (Selecting from the view selects, in effect, using the `SELECT` statement.)
  - The ***select_statement*** can select from base tables or other views.

Create View

From the MySQL docs.

- The view definition is “frozen” at creation time and is not affected by subsequent changes to the definitions of the underlying tables.
  - For example, if a view is defined as `SELECT *` on a table
    - new columns added to the table later do not become part of the view
    - and columns dropped from the table will result in an error when selecting from the view.

“UPDATE-able” Views

- SQL-92 standard
  - SELECT statement uses only one table
  - View must include all the columns of the underlying table’s primary key
  - All columns excluded from the view must either be NULLable or have default values
- More practical points
  - SELECT statement doesn’t have DISTINCT
  - Can’t use GROUP BY or HAVING

Updatable View

From the MySQL docs.

- Some views are updatable and references to them can be used to specify tables to be updated in data change statements.
- That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table.
- For the specific rules, see:

  http://dev.mysql.com/doc/refman/5.7/en/view-updatability.html

INSERT / UPDATE with Views

```
INSERT INTO bucks
   VALUES (36742, 'Smith', 'Alice', null,'SPN', 56700, 52110);
```

```
UPDATE bucks
  SET fsalary = 52000 WHERE fid = 22321;
```

```
UPDATE bucks
  SET fsalary = 65000 WHERE fid = 31890;
```

Updatable Views

From the MySQL docs.

Extra material that will not be on the final exam.

For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. A view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Certain joins
- Reference to nonupdatable view in the `FROM` clause
- Subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table (fails for `INSERT`, okay for `UPDATE`, `DELETE`)

This is a subset of the rules. For a more complete explanation, see http://dev.mysql.com/doc/refman/5.7/en/view-updatability.html

Create View

From the MySQL docs.

- The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view.

Guaranteed to be on the final!

Views Demo

```
CREATE TABLE foo (
  a INT,
  b INT
);

CREATE VIEW foo_view (x, y)
  AS SELECT *
       FROM foo
       WHERE (a % 2) != 1;

CREATE VIEW foo_check_view (x, y)
  AS SELECT *
       FROM foo
       WHERE (a % 2) != 1
  WITH CHECK OPTION;
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
ERROR 1054 (42S22): Unknown column 'a' in 'field list'
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
ERROR 1054 (42S22): Unknown column 'a' in 'field list'

mysql> INSERT INTO foo_view(x,y) VALUES (2,2);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
ERROR 1054 (42S22): Unknown column 'a' in 'field list'

mysql> INSERT INTO foo_view(x,y) VALUES (2,2);
Query OK, 1 row affected (0.10 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
ERROR 1054 (42S22): Unknown column 'a' in 'field list'

mysql> INSERT INTO foo_view(x,y) VALUES (2,2);
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO foo_view(x,y) VALUES (3,3);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo(a,b) VALUES (1,1);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO foo_view(a,b) VALUES (2,2);
ERROR 1054 (42S22): Unknown column 'a' in 'field list'

mysql> INSERT INTO foo_view(x,y) VALUES (2,2);
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO foo_view(x,y) VALUES (3,3);
Query OK, 1 row affected (0.12 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo_check_view(x,y) VALUES (4,4);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo_check_view(x,y) VALUES (4,4);
Query OK, 1 row affected (0.14 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo_check_view(x,y) VALUES (4,4);
Query OK, 1 row affected (0.14 sec)

mysql> INSERT INTO foo_check_view(x,y) VALUES (5,5);
```

Guaranteed to be on the final!

Views Demo

```
mysql> INSERT INTO foo_check_view(x,y) VALUES (4,4);
Query OK, 1 row affected (0.14 sec)

mysql> INSERT INTO foo_check_view(x,y) VALUES (5,5);
ERROR 1369 (HY000): CHECK OPTION failed 'sandbox.foo_check_view'
```

Guaranteed to be on the final!

Views Demo

```
mysql> SELECT * FROM foo;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    3 |
|    4 |    4 |
+------+------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM foo_view;
+------+------+
| x    | y    |
+------+------+
|    2 |    2 |
|    4 |    4 |
+------+------+
2 rows in set (0.00 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> ALTER TABLE foo DROP COLUMN a;
Query OK, 0 rows affected (0.86 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM foo;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set (0.00 sec)
```

Guaranteed to be on the final!

Views Demo

```
mysql> SELECT * FROM foo_view;
```

Guaranteed to be on the final!

Views Demo

```
mysql> SELECT * FROM foo_view;
ERROR 1356 (HY000): View 'sandbox.foo_view' references invalid
table(s) or column(s) or function(s) or definer/invoker of view
lack rights to use them
```

Guaranteed to be on the final!

Views Demo

```
mysql> ALTER TABLE foo ADD COLUMN b DATE;
Query OK, 0 rows affected (0.86 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE foo SET b = CURDATE();
Query OK, 0 rows affected (0.00 sec)
Rows matched: 4  Changed: 0  Warnings: 0
```

Guaranteed to be on the final!

Views Demo

```
mysql> SELECT * FROM foo_view;
```

Guaranteed to be on the final!

Views Demo

```
mysql> SELECT * FROM foo_view;
+------+------------+
| x    | y          |
+------+------------+
|    2 | 2016-10-31 |
|    4 | 2016-10-31 |
+------+------------+
2 rows in set (0.00 sec)
```

Guaranteed to be on the final!

Create View

From the MySQL docs.

The optional `ALGORITHM` clause for `CREATE VIEW` or `ALTER VIEW` affects how MySQL processes the view.

`ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`.

- For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.
- For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.
- For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.

Create View

From the MySQL docs.

`MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view.

The following example briefly illustrates how the `MERGE` algorithm works. The example assumes that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
  SELECT c1, c2
    FROM t
    WHERE c3 > 100;
```

Create View

From the MySQL docs.

`MERGE` Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`
- `*` becomes `vc1,vc2`, which corresponds to `c1,c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2
  FROM t
 WHERE c3 > 100;
```

Create View

From the MySQL docs.

Example 2: Suppose that we issue this statement:

```
SELECT *
  FROM v_merge
  WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Create View

From the MySQL docs.

If the `MERGE` algorithm cannot be used, a temporary table must be used instead.

`MERGE` cannot be used if the view contains any constructs in the following list.

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `LIMIT`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Assignment to user variables
- Refers only to literal values (in this case, there is no underlying table)

Extra material that will not be on the final exam.

View Uses

- Commonly used SELECTs
- SELECT building blocks
- Enforce integrity rules
- Hide columns or rows
- Unit conversion
- Mask physical table changes
- Rename columns

Create Index

Basic syntax.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    ON tbl_name (index_col_name,...);
```

Create Index

From the MySQL docs.

Complete syntax.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option]
    [algorithm_option | lock_option] ...
```

Extra material that will not be on the final exam.

Create Index

From the MySQL docs.

```
CREATE UNIQUE INDEX index_name
    ON tbl_name (index_col_name,...);
```

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row.

A UNIQUE index permits multiple NULL values for columns that can contain NULL.

If you specify a prefix value for a column in a UNIQUE index, the column values must be unique within the prefix.

Unique Index

```
CREATE UNIQUE INDEX email_uniq_idx
    ON Faculty (femail);
```

FULLTEXT Index

From the MySQL docs.

```
CREATE FULLTEXT INDEX index_name
    ON tbl_name (index_col_name,...);
```

`FULLTEXT` indexes are supported only for `InnoDB` and `MyISAM` tables and can include only `CHAR`, `VARCHAR`, and `TEXT` columns.

Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified.

FULLTEXT Index

```
CREATE TABLE bbc_articles (
  id      INT AUTO_INCREMENT PRIMARY KEY,
  topic   VARCHAR(50),
  title   VARCHAR(256) CHARACTER SET utf8,
  body    TEXT CHARACTER SET utf8
);


CREATE FULLTEXT INDEX bbc_articles_fulltext_idx
  ON bbc_articles (title, body);
```

Based on: http://dev.mysql.com/doc/refman/5.7/en/fulltext-natural-language.html

Using this dataset: http://mlg.ucd.ie/datasets/bbc.html

FULLTEXT Index

From the MySQL docs.

Full-text searching is performed using `MATCH()...AGAINST` syntax.

- `MATCH()` takes a comma-separated list that names the columns to be searched.
- `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform.
- The search string must be a string value that is constant during query evaluation. *This rules out, for example, a table column because that can differ for each row.*

Full-Text Natural Language Search

```
SELECT topic, title
  FROM bbc_articles
  WHERE MATCH (title, body)
        AGAINST ('pile of pants' IN NATURAL LANGUAGE MODE);
```

A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies.

Full-text searches are natural language searches if the `IN NATURAL LANGUAGE MODE` modifier is given or if no modifier is given. For more information, see Section 13.9.1, “Natural Language Full-Text Searches”.

```
+----------+---------------------------------+
| topic    | title                           |
+----------+---------------------------------+
| politics | Jowell rejects 'Las Vegas' jibe  |
| business | US to probe airline travel chaos |
| sport    | Mourinho takes swipe at Arsenal  |
| tech     | GTA sequel is criminally good    |
| tech     | Apple Mac mini gets warm welcome |
| tech     | Losing yourself in online gaming |
+----------+---------------------------------+
6 rows in set (0.00 sec)
```

Full-Text Natural Language Search

From the body of “Jowell rejects ‘Las Vegas’ jibe”:

> Meanwhile Labour backbencher Stephen Pound labelled casino-related regeneration schemes "a pile of pants".

From the body of “US to probe airline travel chaos”:

> Adding to the atmosphere of chaos were mountains of luggage left to pile up when a third of US Airways' baggage handling staff called in sick.

Boolean Full-Text Searches

From the MySQL docs.

- A boolean search interprets the search string using the rules of a special query language.
- The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual.
    - For a complete list of operators, see http://dev.mysql.com/doc/refman/5.7/en/fulltext-boolean.html
- Certain common words (stopwords) are omitted from the search index and do not match if present in the search string.

Boolean Full-Text Searches

```
SELECT title
  FROM bbc_articles
  WHERE MATCH (title,body)
        AGAINST ('+putin' IN BOOLEAN MODE);
```

```
+---------------------------------------+
| title                                 |
+---------------------------------------+
| Putin backs state grab for Yukos      |
| Could Yukos be a blessing in disguise? |
| Yukos seeks court action on sale      |
| Yukos heading back to US courts       |
| Indian oil firm eyes Yukos assets     |
| Minister hits out at Yukos sale       |
| Khodorkovsky quits Yukos shares       |
| Russia gets investment blessing       |
| Khodorkovsky ally denies charges      |
| Russia WTO talks 'make progress'      |
| Oil companies get Russian setback     |
| Yukos unit fetches bn at auction      |
| Deutsche attacks Yukos case           |
| Gazprom 'in m back-tax claim'         |
| Russian ex-spy on hunger strike       |
+---------------------------------------+
15 rows in set (0.00 sec)
```

Sample Operators	
+	A leading or trailing plus sign indicates that this word *must* be present in each row that is returned. `InnoDB` only supports leading plus signs.
-	A leading or trailing minus sign indicates that this word must *not* be present in any of the rows that are returned. `InnoDB` only supports leading minus signs.

Boolean Full-Text Searches

```
SELECT title
  FROM bbc_articles
  WHERE MATCH (title,body)
        AGAINST ('+putin -oil' IN BOOLEAN MODE);
```

Sample Operators	
+	A leading or trailing plus sign indicates that this word *must* be present in each row that is returned. `InnoDB` only supports leading plus signs.
-	A leading or trailing minus sign indicates that this word must *not* be present in any of the rows that are returned. `InnoDB` only supports leading minus signs.

```
+---------------------------------+
| title                           |
+---------------------------------+
| Khodorkovsky ally denies charges |
| Russia WTO talks 'make progress' |
| Russian ex-spy on hunger strike  |
+---------------------------------+
3 rows in set (0.00 sec)
```

Create Index

From the MySQL docs.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option]
    [algorithm_option | lock_option] ...
```

Spatial indexes (created using `SPATIAL INDEX`) have these characteristics:

- Available only for `MyISAM` and (as of MySQL 5.7.5) `InnoDB` tables. Specifying `SPATIAL INDEX` for other storage engines results in an error.
- Indexed columns must be `NOT NULL`.
- Column prefix lengths are prohibited. The full width of each column is indexed.

Create Index

From the MySQL docs.

```
index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Extra material that will not be on the final exam.

Create Index

From the MySQL docs.

```
index_col_name:
    col_name [(length)] [ASC | DESC]
```

For string columns, indexes can be created that use only the leading part of column values, using ***col_name***(***length***) syntax to specify an index prefix length.

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column.

Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations.

Create Index

***index_type*:**

```
    USING {BTREE | HASH}
```

index_type

Some storage engines permit you to specify an index type when creating an index.

Table 14.1, “Index Types Per Storage Engine” shows the permissible index type values supported by different storage engines. Where multiple index types are listed, the first one is the default when no index type specifier is given. Storage engines not listed in the table do not support an ***index_type***clause in index definitions.

Table 14.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH, BTREE (see note in text)

From the MySQL docs.

Extra material that will not be on the final exam.

DROP TABLE

From the MySQL docs.

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

`RESTRICT` and `CASCADE` are permitted to make porting easier.

In MySQL 5.7, they do nothing.

DROP VIEW

From the MySQL docs.

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

DROP VIEW

From the MySQL docs.

```
DROP INDEX index_name ON tbl_name
```

Grant

From the MySQL docs.

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [, user_specification] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH {GRANT OPTION | resource_option} ...]
```

Grant

From the MySQL docs.

```
priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}
```

```
object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}
```

Privilege	Meaning and Grantable Levels
`ALL [PRIVILEGES]`	Grant all privileges at specified access level except `GRANT OPTION`
`ALTER`	Enable use of `ALTER TABLE`. Levels: Global, database, table.
`CREATE`	Enable database and table creation. Levels: Global, database, table.
`CREATE TEMPORARY TABLES`	Enable use of `CREATE TEMPORARY TABLE`. Levels: Global, database.
`CREATE USER`	Enable use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. Level: Global.
`CREATE VIEW`	Enable views to be created or altered. Levels: Global, database, table.
`DELETE`	Enable use of `DELETE`. Level: Global, database, table.
`DROP`	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
`INDEX`	Enable indexes to be created or dropped. Levels: Global, database, table.
`INSERT`	Enable use of `INSERT`. Levels: Global, database, table, column.
`REFERENCES`	Enable foreign key creation. Levels: Global, database, table, column.
`SELECT`	Enable use of `SELECT`. Levels: Global, database, table, column.
`UPDATE`	Enable use of `UPDATE`. Levels: Global, database, table, column.
`USAGE`	Synonym for “no privileges”

From the MySQL docs.

REVOKE

From the MySQL docs.

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```