Flagship Features In MySQL 5.0

Presented by:
Trudy Pelzer, Senior Software Architect
MySQL AB
trudy@mysql.com

MySQL Users Conference, 2005
April 18-21, 2005
 Interruption Policy

Any questions?

Interrupt or wait till the end.
The Flagship Features

• Views
• Stored Procedures
• Triggers
• INFORMATION_SCHEMA
The Flagship Features

- Views
- Stored Procedures
- Triggers
- INFORMATION_SCHEMA

Also:
- Precision math
- Strict data type checking
Views

A view is a virtual table.

- It provides another way of looking at existing data.

Advantage:

- A view uses up no disk space.
Views: New DDL Statements

- CREATE VIEW
- ALTER VIEW
- DROP VIEW
CREATE VIEW Syntax

CREATE  [OR REPLACE]
[ALGORITHM =
   {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name
[(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]

CREATE VIEW view1 ...
CREATE VIEW Syntax

CREATE  [OR REPLACE]
[ALGORITHM =
   {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name
[(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]

CREATE OR REPLACE view1 ...
CREATE VIEW Syntax

CREATE [OR REPLACE]
[ALGORITHM =
  {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name
[(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
Two Algorithms for Views (I)

Merge algorithm:

```
SELECT * FROM v;
```

becomes

```
SELECT * FROM t;
```
Two Algorithms for Views (II)

Temporary table (temptable) algorithm:

```
SELECT * FROM v;
```

becomes

```
CREATE TEMPORARY TABLE temp_table
    AS SELECT * FROM t;
SELECT * FROM temp_table;
```
Letting MySQL Choose the Right Algorithm (I)

CREATE TABLE t (coll1 INT);

CREATE VIEW v AS SELECT coll1 FROM t;
-- MERGE algorithm will be used
Letting MySQL Choose the Right Algorithm (II)

CREATE TABLE t (col1 INT);

CREATE VIEW v AS SELECT AVG(col1) FROM t;
-- TEMPTABLE algorithm will be used
Letting MySQL Choose the Right Algorithm (III)

TEMPTABLE algorithm for view definitions with:

- DISTINCT
- Aggregate function
- UNION
- GROUP BY
- HAVING
- Only literal values
Overriding MySQL by Choosing the Algorithm

CREATE ALGORITHM = TEMPTABLE VIEW v AS SELECT * FROM t;
CREATE VIEW Syntax

CREATE  [OR REPLACE]
 [ALGORITHM =
     {MERGE | TEMPTABLE | UNDEFINED}]

VIEW view_name
[(column_list)]
AS select_statement

[WITH [CASCADED | LOCAL] CHECK OPTION]
Omitted [ (column list) ]

By default:

```
CREATE VIEW v AS SELECT col1, col2 FROM t;
```

```
mysql> SELECT * FROM v;
+-------+-------+
| col1  | col2  |
|-------+-------|
| 10    | 20    |
+-------+-------+
```
Included [ (column list) ]

Different names specified:

CREATE VIEW v (view_col1, view_col2) AS
   SELECT col1, col2 FROM t;

mysql> SELECT * FROM v;
+------------------+
| view_col1 | view_col2 |
+------------------+
|    10      |    20     |
+------------------+
CREATE VIEW Syntax

CREATE [OR REPLACE]
[ALGORITHM =
    {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name
[(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
View Queries can Include any SELECT Clause

CREATE VIEW v AS
    SELECT col1 FROM t WHERE col1 > 4;

SELECT * FROM v WHERE col1 < 6;

same effect as:

SELECT * FROM t
    WHERE col1 > 4 AND col1 < 6;
CREATE VIEW Syntax

CREATE [OR REPLACE]
[ALGORITHM =
    {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name
[(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
WITH CHECK OPTION

CREATE VIEW v1 AS
  SELECT col1 FROM t WHERE col1 <> 'A'
  WITH CHECK OPTION;

mysql> INSERT INTO v1 VALUES ('A');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v1'

mysql> UPDATE v1 SET col1 = 'A';
ERROR 1369 (HY000): CHECK OPTION failed 'db.v1'
WITH LOCAL CHECK OPTION

CREATE VIEW v2 AS

SELECT * FROM v1 WHERE col1 <> 'B'

WITH LOCAL CHECK OPTION;

mysql> INSERT INTO v2 VALUES ('B');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v2'

mysql> INSERT INTO v2 VALUES ('A');
Query OK, 1 row affected (0.01 sec)
WITH LOCAL CHECK OPTION

CREATE VIEW v2 AS
  SELECT * FROM v1 WHERE col1 <> 'B'
  WITH LOCAL CHECK OPTION;

mysql> INSERT INTO v2 VALUES ('B');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v2'

mysql> INSERT INTO v2 VALUES ('A');
Query OK, 1 row affected (0.01 sec)
WITH CASCADED CHECK OPTION

CREATE VIEW v3 AS
    SELECT * FROM v2 WHERE col1 <> 'C'
    WITH CASCADED CHECK OPTION;

mysql> INSERT INTO v3 VALUES ('C');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v3'
-- because v3’s condition doesn’t allow 'C'
WITH CASCADED CHECK OPTION

CREATE VIEW v3 AS

    SELECT * FROM v2 WHERE col1 <> 'C'

    WITH CASCADED CHECK OPTION;

mysql> INSERT INTO v3 VALUES ('B');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v3' -- because v2's condition doesn't allow 'B' and v3 depends on v2
WITH CASCADED CHECK OPTION

CREATE VIEW v3 AS

    SELECT * FROM v2 WHERE col1 <> 'C'
    WITH CASCADED CHECK OPTION;

mysql> INSERT INTO v3 VALUES ('A');
ERROR 1369 (HY000): CHECK OPTION failed 'db.v3'
-- because v1's condition doesn't allow 'A' and v3 depends on v1 through v2
GRANT/REVOKE:
New Privileges

GRANT CREATE VIEW ON *.* TO peter;

GRANT SHOW VIEW ON *.* TO peter;

REVOKE CREATE VIEW ON *.* FROM peter;

REVOKE SHOW VIEW ON *.* FROM peter;
GRANT/REVOKE: New Privileges

GRANT CREATE VIEW ON *.* TO peter;

GRANT SHOW VIEW ON *.* TO peter;

REVOKE CREATE VIEW ON *.* FROM peter;

REVOKE SHOW VIEW ON *.* FROM peter;
Privileges Needed to Use a View

General rule:
• Same as for tables.

Special rule:
• If you have privilege on a view, you don't need a privilege on its underlying tables.
Updatable Views

What is an updatable view?

- Can accept UPDATE, DELETE
- Can sometimes accept INSERT

Effect:

- As if you changed the data in the underlying base table
Using Views

What are views for?
Use of Views (I)

1. Views can be used in place of constraint checks.

CREATE TABLE t (col1 INT, col2 INT);

CREATE VIEW v AS SELECT col1, col2
FROM t WHERE
  (col1 is NOT NULL OR col2 IS NOT NULL)
  AND col1 BETWEEN 1 AND 5
WITH CHECK OPTION;
Use of Views (I)

1. Views can be used in place of constraint checks.

```sql
CREATE TABLE t (col1 INT, col2 INT);

CREATE VIEW v AS SELECT col1, col2
    FROM t WHERE
    (col1 is NOT NULL OR col2 IS NOT NULL)
    AND col1 BETWEEN 1 AND 5
    WITH CHECK OPTION;
```
Use of Views (I)

1. Views can be used in place of constraint checks.

```
CREATE TABLE t (col1 INT, col2 INT);

CREATE VIEW v AS SELECT col1, col2
  FROM t WHERE
    (col1 is NOT NULL OR col2 IS NOT NULL)
    AND col1 BETWEEN 1 AND 5
  WITH CHECK OPTION;
```
Use of Views (I)

1. Views can be used in place of constraint checks.

```sql
CREATE TABLE t (col1 INT, col2 INT);

CREATE VIEW v AS SELECT col1, col2
    FROM t WHERE
    (col1 is NOT NULL OR col2 IS NOT NULL)
    AND col1 BETWEEN 1 AND 5
    WITH CHECK OPTION;
```
Use of Views (II)

2. Table Restructuring:
Views can be used to hide or move columns.

CREATE TABLE t (  
    col1 INT, col2 CHAR(5), col3 DATE);  

CREATE VIEW v AS SELECT col3, col1 FROM t;  

GRANT SELECT ON v TO some_user;
Use of Views (III)

3. Data Security:
Views can be used to hide rows.

CREATE TABLE t (
  col1 INT, col2 CHAR(5), COL3 DATE);

CREATE VIEW v AS SELECT * FROM t
  WHERE col1 BETWEEN 10 AND 100;

GRANT SELECT ON v TO some_user;
Why Views (I)

A view is a virtual table.
• So it doesn't take up extra disk space.

A view provides another way of looking at existing data.
• So you can effectively "reorganize" your data.
Why Views (II)

MySQL's view syntax is nearly 100% standard SQL.

- So porting your applications is easy.

The CREATE VIEW ALGORITHM clause provides the option to specify how MySQL will process a view.

- So you can tailor your views for the most efficient processing.
Why Views (III)

A view's SELECT definition looks like any SELECT.

- So if you know how to query your database, you also know how to define a view.

A SELECT on a view is the same as a SELECT on a regular table.

- So if you know how to query your database, you also know how to query a view.
Why Views (IV)

The CREATE VIEW WITH CHECK OPTION clause defines the data that belongs in a view.

• So you have an extra layer of data validity checks.

Users cannot create a view without the CREATE VIEW privilege.

• So you can ensure that only users who should be able to change the database definition can do so.
Why Views (V)

Users cannot see information on underlying tables without appropriate privileges.
• So you can ensure sensitive information is concealed.

Use of a view requires the same privileges that are needed to use a table.
• So views let you tailor the actions users may perform.
Why Views (VI)

Many views will allow INSERT, UPDATE, and DELETE.

- So views are easy to use and -- because they act just like regular tables -- their nature can be hidden from users.
Stored Procedures

A stored procedure is a routine that is stored in the database.

Two kinds:
- Procedures -- which are called with CALL
- Functions -- whose return values are used in SQL statements
Stored Procedures:
New DDL Statements

- CREATE PROCEDURE/FUNCTION
- ALTER PROCEDURE/FUNCTION
- DROP PROCEDURE/FUNCTION
CREATE Procedure Syntax

```
CREATE PROCEDURE procedure_name
([parameter [...]])
[characteristic ...]
routine_body
```

CREATE Function Syntax

```
CREATE FUNCTION function_name
([parameter [, ...]])
RETURNS data_type
[characteristic ...]
routine_body
```
Privilege Checking (I)

CREATE PROCEDURE p1 (param1 INT)
  LANGUAGE SQL
  DETERMINISTIC
  SQL SECURITY DEFINER ...

CREATE FUNCTION f1 (param1 INT)
  RETURNS INT
  LANGUAGE SQL
  NOT DETERMINISTIC
  SQL SECURITY INVOKER ...
Privilege Checking (II)

CREATE PROCEDURE p1 (param1 INT)
    LANGUAGE SQL
    DETERMINISTIC
    SQL SECURITY DEFINER ...

CREATE FUNCTION f1 (param1 INT)
    RETURNS INT
    LANGUAGE SQL
    NOT DETERMINISTIC
    SQL SECURITY INVOKER ...
Routine Body

Routine body:

- Specifies what the stored procedure will do
- Can contain one or more SQL statements
Compound Statements

Syntax:
[begin_label:] BEGIN [statement_list] END
[end_label]

CREATE PROCEDURE p1 ()
BEGIN
    SELECT * FROM t;
END; //
BEGIN ... END Statement List

Can include:

- INSERT
- UPDATE
- DELETE
- SELECT
- REPLACE
- SET
- COMMIT / ROLLBACK

...
BEGIN ... END:
Also Allows DECLARE

[begin_label:] BEGIN
  [DECLARE variable_list]
  [DECLARE conditions]
  [DECLARE cursors]
  [DECLARE handlers]
  [statement_list]
END [end_label]
DECLARE Variables

Syntax:
DECLARE variable_name [, ...]
  data_type [DEFAULT value];

CREATE PROCEDURE p1 ()
  BEGIN
  DECLARE v1 INT DEFAULT 10;
  SET v1 = 5;
  INSERT INTO t VALUES (v1);
  END; //
DECLARE Conditions

Syntax:
DECLARE condition_name CONDITION FOR
{SQLSTATE [VALUE] 'sqlstate' | mysql_error_code};

CREATE PROCEDURE p1 () BEGIN
  DECLARE v1 INT DEFAULT 10;
  DECLARE syntax_error CONDITION FOR SQLSTATE '42000';
  SELECT col1 INTO v1 FROM t LIMIT 1;
END; //
DECLARE Handlers

Syntax:

DECLARE {CONTINUE | EXIT} HANDLER FOR condition_value [,....] action;

action:

• Is an SQL statement.
• The statement is executed only if the condition_value is returned by the server.
DECLARE Cursors

Syntax:
DECLARE cursor_name CURSOR FOR select_query;

CREATE PROCEDURE p1 () BEGIN
    DECLARE v1 VARCHAR(7) DEFAULT 'OK';
    DECLARE syntax_error CONDITION FOR SQLSTATE '42000';
    DECLARE c1 CURSOR FOR SELECT * FROM t;
    DECLARE CONTINUE HANDLER FOR SQLWARNING
        SET v1 = 'Warning';
    ...

    ; //
Flow Control

- **IF** -- if `<condition>` do `<action>`
- **CASE** -- switch depending on condition
- **LOOP** -- do `<action>` repeatedly
- **LEAVE** -- break out of a loop or block
- **ITERATE** -- restart loop
- **REPEAT** -- repeat `<action>` until `<condition>` is true
- **WHILE** -- repeat `<action>` as long as `<condition>` is true
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- **Stored procedures are fast**
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- **Stored procedures are components**
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Why Stored Procedures

- Stored procedures are proven technology
- Stored procedures are fast
- Stored procedures are components
- Stored procedures are portable
- Stored procedures are stored
- Stored procedures are migratory
Use of Stored Procedures: Summary

Use stored procedures to:

- improve performance
- save time, money, resources
Use of Stored Procedures: Summary

Use stored procedures to:

• improve performance
• save time, money, resources
Triggers

A trigger is a named chain reaction that starts when a specific event occurs for a specific table.
Triggers: New DDL Statements

- CREATE TRIGGER
- DROP TRIGGER
CREATE TRIGGER Syntax

CREATE TRIGGER trigger_name

{BEFORE | AFTER}

{INSERT | UPDATE | DELETE} ON table_name

FOR EACH ROW trigger_action

CREATE TRIGGER tg1

BEFORE DELETE ON t1

FOR EACH ROW SET @x = 1;
CREATE TRIGGER Syntax

CREATE TRIGGER trigger_name
   {BEFORE | AFTER}
   {INSERT | UPDATE | DELETE} ON table_name
FOR EACH ROW trigger_action

CREATE TRIGGER tg1
   AFTER INSERT ON t1
FOR EACH ROW <compound SQL statement>;

- BEFORE:
  - Performs the trigger action before the data is affected.
- AFTER:
  - Performs the trigger action after the data is affected.
- INSERT:
  - Performs when a row is inserted into a table.
- UPDATE:
  - Performs when a row is updated in a table.
- DELETE:
  - Performs when a row is deleted from a table.

For each row "trigger_action" is used to define the action to be performed by the trigger.
Use of Triggers (I)

1. Log/audit/monitor data-change activity.

CREATE TRIGGER tg1 AFTER UPDATE ON t1
  FOR EACH ROW
  <record who did it, when it was done, what row was changed, what value was changed>;
Use of Triggers (II)

2. **Data validation.**
Change or negate INSERT, UPDATE, DELETE on a table.

```sql
CREATE TRIGGER tgl BEFORE UPDATE ON t1
    FOR EACH ROW
    <don't allow the update under specific conditions>;
```
Use of Triggers (III)

3. **Consistency and clean-up.**

Perform related actions on related tables.

```
CREATE TRIGGER tg1 AFTER DELETE ON t1
  FOR EACH ROW
  <delete corresponding rows from related tables>;
```
INFORMATION_SCHEMA

The INFORMATION_SCHEMA is the standard SQL way to access database metadata.

- A series of tables that describe database objects.
- Query them with SELECT.
Why INFORMATION_SCHEMA

Disadvantages of SHOW:

- SHOW commands are non-standard; they are specific to MySQL.
- SHOW commands require that you learn an entire set of commands to be able to access the metadata you need.
Why INFORMATION_SCHEMA

Advantages of INFORMATION_SCHEMA:

• INFORMATION_SCHEMA is standard SQL, so other DBMSs have it too.

• INFORMATION_SCHEMA can be queried via a SELECT statement, so there is no need to learn a new set of commands to be able to access the metadata you need.
SELECT ... FROM ...

mysql> SELECT table_name, table_type, engine
   -> FROM INFORMATION_SCHEMA.tables
   -> WHERE table_schema = 'tp'
   -> ORDER BY table_type, table_name DESC;

+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| t2         | BASE TABLE | MyISAM |
| t1         | BASE TABLE | InnoDB |
| v1         | VIEW       | NULL   |
+------------+------------+--------+
Using INFORMATION_SCHEMA

• INFORMATION_SCHEMA is a new, virtual database.

• There is no need to create an INFORMATION_SCHEMA file.

• MySQL creates and populates the tables automatically.

• Only SELECT is allowed.
Privileges

Easy and automatic:

- SELECT on every INFORMATION_SCHEMA table is automatically granted to every user.

Secure:

- You can see only the metadata that applies to you.
What Character Sets/Collations Can I Use?

- CHARACTER_SETS
- COLLATIONS
- COLLATION_CHARACTER_SET_APPLICABILITY
What Objects Can I Use?

- SCHEMATA
- TABLES
- VIEWS
- ROUTINES
What Objects Can I Use?

- COLUMNS

```sql
SELECT * FROM INFORMATION_SCHEMA.COLUMNS ...
```

*************** 1. row ***************

- TABLE_CATALOG: NULL
- TABLE_SCHEMA: tp
- TABLE_NAME: t1
- COLUMN_NAME: col1

...
What Indexes Can I Use?

- STATISTICS
- TABLE_CONSTRAINTS
- KEY_COLUMN_USAGE
What Privileges Are Granted?

- **USER_PRIVILEGES**
- **SCHEMA_PRIVILEGES**
- **TABLE_PRIVILEGES**
- **COLUMN_PRIVILEGES**
Also New: Precision Math

New math library means:

- DECIMAL/INTEGER arithmetic is done with exact numeric, not floating-point, calculations.

- Now 1/.1 is always 10.
Also New: Strict Mode (I)

mysql> CREATE TABLE t1 (-> col1 TINYINT, col2 CHAR(3));
Query OK, 0 rows affected (0.06 sec)

mysql> SET SQL_MODE='TRADITIONAL';
Query OK, 0 rows affected (0.00 sec)
Also New: Strict Mode (II)

mysql> INSERT INTO t1 (col1) VALUES (300);
ERROR 1264 (22003): Out of range value adjusted for column 'col1' at row 1

- TINYINT range is -128 to +127.
Also New: Strict Mode (III)

mysql> INSERT INTO t1 (col2) VALUES ('abcd');
ERROR 1406 (22001): Data too long for column 'col2' at row 1

• Column's maximum size is 3 characters.

mysql> select * from t1;
Empty set (0.00 sec)
Thank You!

More information: forums.mysql.com